ÍNDICE GENERAL

Capítulo 0

Introducción

0.1. DEFINICIÓN DE OBJETIVOS

Cuando nos encontramos ante un gráfico, la información que en él aparece constituye un elemento central de cara a la consecución de su principal objetivo: ser capaz de exponer de la forma más clara posible el objeto de dicho gráfico. Así por ejemplo en la mayor parte de los trabajos relacionados con la Cartografía, como mapas, planos cartográficos, etc. (véase la *Figura 0.1.1*), incluso en la elaboración de organigramas de planificación, diagramas técnicos de bloques, diseño asistido, diseño de circuitos, etc., nos encontramos con la necesidad de colocar diferente información que ayude a identificar distintos elementos del gráfico.

Debido a la irrupción de los medios informáticos cobra un enorme auge la tecnología de la composición y con ello aumenta considerablemente la información que tiene que ser incorporada a cualquier gráfico, hasta el punto que se estima que la tarea de situar la información sobre los distintos elementos de un mapa constituye el 50% del tiempo total que se emplea en la elaboración del mismo [3].

Por este motivo resulta importante encontrar algoritmos que resuelvan de una manera eficaz este problema. Así aparecen numerosos trabajos poniendo de manifiesto el gran interés que el tema suscita en la comunidad científica internacional.

Cuando pretendemos ofrecer información sobre los elementos de un gráfico es frecuente la necesidad de resaltar algunos de dichos elementos, para ello se suelen establecer criterios que los identifiquen, así por ejemplo se utilizan tipos de fuente diferentes para resaltar distintos elementos del gráfico, o bien utilizar etiquetas de distinto tamaño, dependiendo de la relevancia del elemento a señalar.

Estos criterios son fundamentalmente estéticos: el tamaño de las etiquetas ha de ser lo suficientemente grande para que la información que en ellas aparece pueda ser legible, la ubicación de las etiquetas debe ser de forma que evite la ambigüedad, etc.

Figura 0.1.1.: Mapa de carreteras de Sevilla

El problema de decisión general de etiquetado podría enunciarse de la forma siguiente:

PROBLEMA GENERAL DE ETIQUETADO:

ENTRADA: Un conjunto de *n* objetos en un gráfico y para cada uno de ellos una etiqueta con información.

PREGUNTA: ¿Se puede encontrar una posición para cada etiqueta cercana al objeto a que se refiere, sin que se produzcan intersecciones entre ellas?

Una importante clase de estos problemas la constituyen el etiquetado de puntos con etiquetas rectangulares, debido a que en gran cantidad de aplicaciones los elementos a etiquetar son puntos: ciudades en un mapa, estaciones de metro, puntos en un circuito impreso, etc.

Para esta clase de problemas existen dos modelos diferentes de etiquetado. Podemos abordar el etiquetado con *etiquetas fijas*, presentado por *Formann y Wagner* [4] en el que cada etiqueta presenta un número finito *m* de opciones para su ubicación. Los casos más frecuentes son *m = 4*, siendo posible la ubicación de la etiqueta rectangular de forma que el punto a etiquetar quede situado en cualquiera de los vértices del rectángulo; *m = 8*, si añadimos a los cuatro puntos anteriores los correspondientes a las cuatro posiciones norte, sur, este y oeste; etc.

El otro modelo de etiquetado, conocido como etiquetado con *etiquetas deslizantes*, presenta la novedad de que existe un número no finito de posiciones para ubicar la etiqueta, ya que ésta puede ser colocada de forma que el punto pueda quedar situado sobre cualquiera de los puntos del contorno del rectángulo. Este modelo fue introducido por *Van Kreveld y otros* [5].

En los dos trabajos mencionados anteriormente se pone de manifiesto el carácter intratable de los dos problemas generales de etiquetado, tanto con etiquetas fijas como deslizantes. Por este motivo aparecen muchos artículos en los que se incorporan restricciones al problema: etiquetas de dimensiones específicas, etiquetas de proporciones determinadas, etc.

En esta línea de acercamiento al problema general mediante el estudio de casos particulares del mismo y teniendo en cuenta que en multitud de ocasiones los puntos a etiquetar se encuentran alineados, como es el caso de etiquetado de redes de metro (véase la *Figura 0.1.3*), etiquetado de circuitos impresos (véase la *Figura 0.1.4*), etc., resulta interesante abordar el problema de etiquetado de puntos alineados. Estos problemas serán objeto de estudio en esta memoria.

El planteamiento a seguir será el siguiente: manteniendo el tamaño dado para las etiquetas, obtener el mayor número de tales etiquetas que pueden ser colocadas sobre los objetos del gráfico, sin intersecciones entre ellas:

MAXIMIZACIÓN DEL NÚMERO DE ETIQUETAS:

ENTRADA: Un conjunto de n objetos en un gráfico y para cada uno de ellos una etiqueta con información.

PREGUNTA: ¿Cuál será el mayor subconjunto de etiquetas para las que se puede encontrar una posición cercana al objeto a que se refiere, sin que se produzcan intersecciones entre ellas?

En esta memoria afrontaremos el estudio de los problemas de optimización del tamaño de las etiquetas en el etiquetado de puntos alineados con etiquetas rectangulares, tanto en el modelo de etiquetado con etiquetas fijas como con etiquetas deslizantes. Consideraremos que las etiquetas son topológicamente abiertas, es decir está permitido que dos etiquetas estén en contacto.

Cuando tratamos de colocar etiquetas con información sobre puntos el tamaño de las etiquetas viene provocado por la extensión del texto a colocar en ellas. Por este motivo resulta frecuente que la anchura de las etiquetas sea variable, mientras que la altura sea constante. Teniendo en cuenta que si los puntos se encuentran situados sobre una línea horizontal la altura de las etiquetas no tiene relevancia para el estudio del problema, mientras que si por el contrario los puntos están situados sobre una línea vertical es la anchura de las etiquetas la que no tiene trascendencia para la resolución del problema, es de interés estudiar tanto problemas de etiquetado con etiquetas iguales (en este apartado se encontrarían los problemas de etiquetado de puntos sobre una línea vertical) como de etiquetado con etiquetas diferentes (situaciones que se presentarían cuando los puntos se encuentran sobre una línea horizontal).

Por esta razón dentro de cada modelo de etiquetado estudiaremos los problemas concernientes a etiquetas iguales y a etiquetas diferentes. En este sentido realizaremos un proceso de generalización en los tamaños de las etiquetas: etiquetado con etiquetas cuadradas iguales, cuadradas diferentes, rectangulares de igual tamaño, etc.

Este estudio lo hemos dividido en dos apartados según que los puntos se encuentren sobre una línea horizontal (o vertical) o sobre una

línea oblicua y considerando los dos modelos de etiquetado con etiquetas fijas y deslizantes.

Generalmente cuando utilizamos etiquetas para ofrecer información sobre elementos de un gráfico, el tamaño de tales etiquetas viene determinado por la longitud del texto a situar en ellas. Por este motivo juega un papel mucho más importante, con el sistema de escritura occidental, la anchura de las etiquetas que su correspondiente altura. Ahora bien, como se aprecia en la *Figura 0.1.2*, si los puntos están situados sobre una línea horizontal la altura de las etiquetas tiene poca relevancia, ya que no influye en el desarrollo del problema. Por el contrario si los puntos están situados sobre una línea vertical es la anchura de las etiquetas la componente que no tiene interés para el estudio del problema.

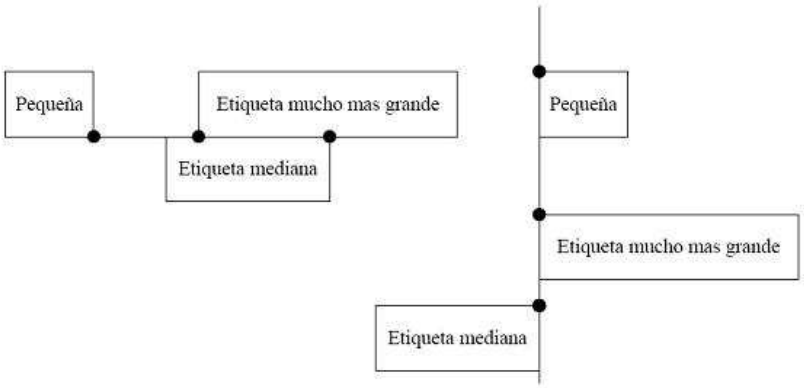


Figura 0.1.2.: Relevancia de las dimensiones de las etiquetas

Es por este motivo por lo que estudiaremos tanto problemas con etiquetas iguales (que pueden ser asociadas a situaciones de etiquetado sobre una línea vertical) como problemas con etiquetas de tamaños diferentes (que corresponderían a situaciones de etiquetado sobre una línea horizontal).

En cada uno de los modelos de etiquetado seguiremos un proceso de generalización en el tipo de etiquetas, comenzando con el caso más simple en que todas las etiquetas son cuadradas e iguales para ir generalizando los tipos de etiquetas.

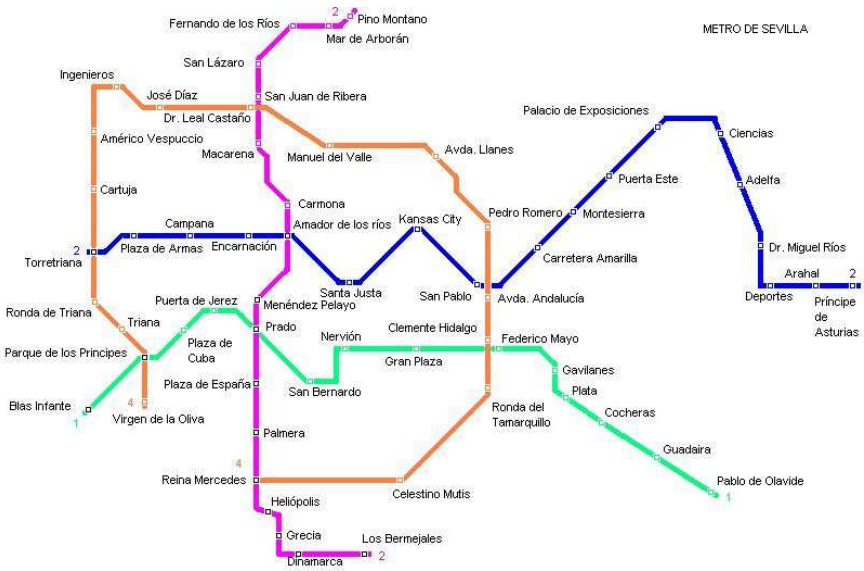


Figura 0.1.3.: Plano del metro de Sevilla

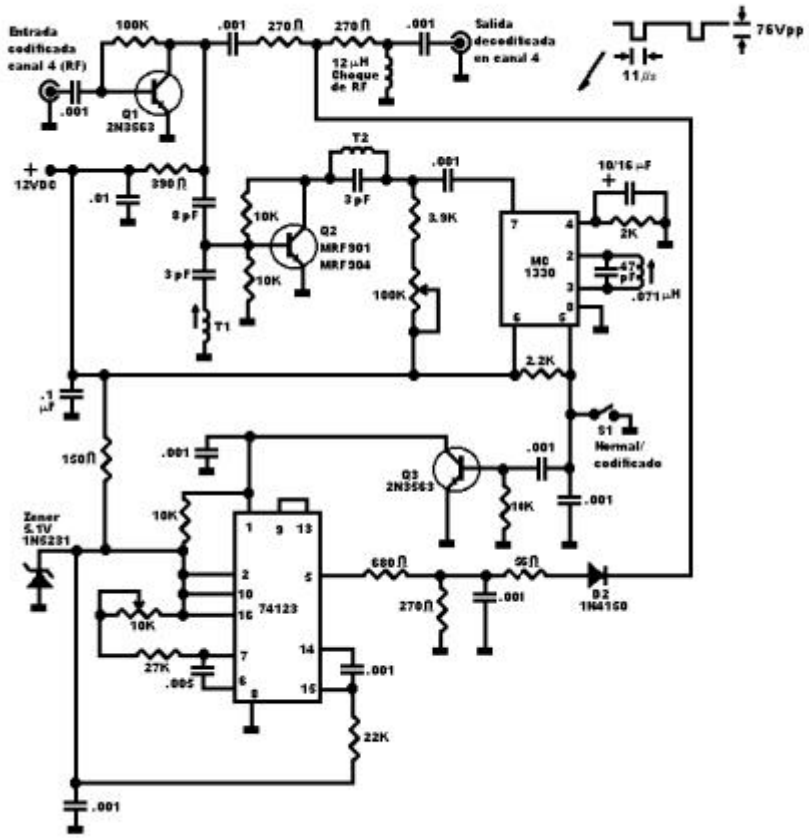


Figura 0.1.4.: Circuito decodificador de video

0.2. HISTORIA

Motivada por sus aplicaciones en diferentes áreas como la Cartografía, diseño asistido por ordenador, trazado de circuitos, etc., la comunidad científica internacional ha mostrado gran interés por los problemas de *etiquetado*, como muestra la cantidad de trabajos que aparecen sobre la materia. El interés por este campo queda refrendado por el hecho de que la ACM (*American Computing Machinery*) lo incorpora continuamente como área preferente de investigación dentro del campo de la Geometría Computacional en su informe *Computational Geometry Impact Task Force* [6].

Dentro de este campo, y debido a la enorme complejidad de sus problemas, surgen multitud de variantes con distintas particularidades. Una de las variantes más naturales es la consistente en el etiquetado de puntos con etiquetas rectangulares, presentando a su vez al menos dos modelos: etiquetado con etiquetas fijas, donde cada etiqueta puede ser colocada en un número finito de posiciones, y etiquetado con etiquetas deslizantes, de forma que cada etiqueta puede ser colocada de forma que el punto quede situado en cualquiera de los puntos del contorno de la etiqueta.

En numerosos problemas de etiquetado, como trabajos relacionados con la cartografía, trazado de mapas, organigramas, dibujos de grafos, diseño de circuitos impresos, etc., nos encontramos con la necesidad de ofrecer información sobre distintos elementos situados en un gráfico en el que los puntos a etiquetar suelen estar alineados. Estos modelos de etiquetado son el objeto de estudio de esta memoria, donde se estudiará la complejidad computacional de los distintos problemas que se pueden plantear, tanto cuando los puntos estén situados sobre una línea horizontal como en una línea oblicua. Abordaremos los problemas derivados de decidir si puede realizarse el correspondiente etiquetado, así como los derivados de obtener los tamaños máximos de las etiquetas que pueden ser colocadas.

Existe abundante bibliografía donde acudir para aclarar los conceptos básicos utilizados en esta memoria. En este sentido podemos recomendar los textos de *Garey y Johnson* [7] y *Ausiello y otros* [8] sobre

complejidad computacional y algoritmos de aproximación; el texto de *Bollobas* [9] sobre teoría de grafos; los textos sobre geometría computacional de *Preparata y Shamos* [10], de de *Berg y otros* [11] y el manual de *Sack y Urrutia* [12], y el texto de *Tamassia y otros* [13] sobre visualización y trazado de grafos. No obstante en el capítulo introductorio se ofrecen a modo de resumen las nociones previas más importantes que son necesarias para el desarrollo de esta memoria.

0.3. SOBRE EL LENGUAJE JAVA

Java surgió en 1991 cuando un grupo de ingenieros de *Sun Microsystems* trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. La reducida potencia de cálculo y memoria de los electrodomésticos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido.

Debido a la existencia de distintos tipos de CPUs y a los continuos cambios, era importante conseguir una herramienta independiente del tipo de CPU utilizada. Desarrollaron un código "neutro" que no dependía del tipo de electrodoméstico, el cual se ejecutaba sobre una "*máquina hipotética o virtual*" denominada *Java Virtual Machine* (***JVM***). Era la *JVM* quien interpretaba el código neutro convirtiéndolo a código particular de la CPU utilizada. Esto permitía lo que luego se ha convertido en el principal lema del lenguaje: "*Write Once, Run Everywhere*". A pesar de los esfuerzos realizados por sus creadores, ninguna empresa de electrodomésticos se interesó por el nuevo lenguaje.

Como lenguaje de programación para computadores, *Java* se introdujo a finales de 1995. La clave fue la incorporación de un intérprete *Java* en la versión 2.0 del programa Netscape Navigator, produciendo una verdadera revolución en Internet. *Java 1.1* apareció a principios de 1997, mejorando sustancialmente la primera versión del lenguaje. *Java 1.2*, más tarde rebautizado como ***Java 2***, nació a finales de 1998.

Al programar en *Java* no se parte de cero. Cualquier aplicación que se desarrolle "cuelga" (o se apoya, según como se quiera ver) en un gran número de *clases* preexistentes. Algunas de ellas las ha podido hacer el propio usuario, otras pueden ser comerciales, pero siempre hay un número muy importante de clases que forman parte del propio lenguaje (el ***API*** o *Application Programming Interface* de *Java*). *Java* incorpora en el propio lenguaje muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (threads, ejecución remota, componentes, seguridad, acceso a bases de datos, etc.). Por eso muchos expertos opinan que *Java* es el lenguaje ideal para aprender la informática

moderna, porque incorpora todos estos conceptos de un modo estándar, mucho más sencillo y claro que con las citadas extensiones de otros lenguajes. Esto es consecuencia de haber sido diseñado más recientemente y por un único equipo.

El principal objetivo del lenguaje *Java* es llegar a ser el "nexo universal" que conecte a los usuarios con la información, esté ésta situada en el ordenador local, en un servidor de *Web*, en una base de datos o en cualquier otro lugar.

Java es un lenguaje muy completo (de hecho se está convirtiendo en un macro-lenguaje: *Java 1.0* tenía 12 packages; *Java 1.1* tenía 23 y *Java 1.2* tiene 59). En cierta forma casi todo depende de casi todo. Por ello, conviene aprenderlo de modo *iterativo*: primero una visión muy general, que se va refinando en sucesivas iteraciones. Una forma de hacerlo es empezar con un ejemplo completo en el que ya aparecen algunas de las características más importantes.

La compañía *Sun* describe el lenguaje *Java* como "*simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico*". Además de una serie de halagos por parte de *Sun* hacia su propia criatura, el hecho es que todo ello describe bastante bien el lenguaje *Java*, aunque en algunas de esas características el lenguaje sea todavía bastante mejorable. Algunas de las anteriores ideas se irán explicando a lo largo de este manual.

Java 2 (antes llamado *Java 1.2* o *JDK 1.2*) es la tercera versión importante del lenguaje de programación *Java*.

No hay cambios conceptuales importantes respecto a *Java 1.1* (en *Java 1.1* sí los hubo respecto a *Java 1.0*), sino extensiones y ampliaciones, lo cual hace que a muchos efectos sea casi lo mismo trabajar con *Java 1.1* o con *Java 1.2*.

Los programas desarrollados en *Java* presentan diversas ventajas frente a los desarrollados en otros lenguajes como C/C++. La ejecución de programas en *Java* tiene muchas posibilidades: ejecución como aplicación independiente (*Stand-alone Application*), ejecución como *applet*, ejecución como *servlet*, etc. Un *applet* es una aplicación especial que se ejecuta dentro de un navegador o browser (por ejemplo

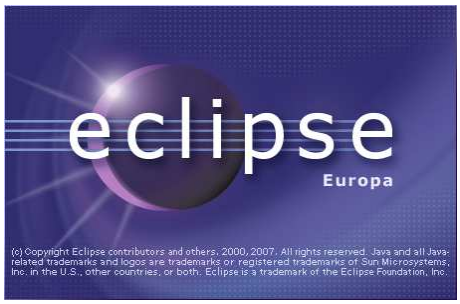
Netscape Navigator o *Internet Explorer*) al cargar una página HTML desde un servidor *Web*. El *applet* se descarga desde el servidor y no requiere instalación en el ordenador donde se encuentra el browser. Un *servlet* es una aplicación sin interface gráfica que se ejecuta en un servidor de Internet. La ejecución como aplicación independiente es análoga a los programas desarrollados con otros lenguajes.

Además de incorporar la ejecución como *Applet*, *Java* permite fácilmente el desarrollo tanto de arquitecturas cliente-servidor como de aplicaciones distribuidas, consistentes en crear aplicaciones capaces de conectarse a otros ordenadores y ejecutar tareas en varios ordenadores simultáneamente, repartiendo por lo tanto el trabajo. Aunque también otros lenguajes de programación permiten crear aplicaciones de este tipo, *Java* incorpora en su propio *API* estas funcionalidades.

Capítulo 1

Herramientas de implementación

Aunque existen distintos programas comerciales que permiten desarrollar código *Java*, para implementar los problemas ya presentados nos apoyaremos en la versión 3.3.0 de ***Eclipse***, ya que se trata de software de libre distribución disponible en la página oficial de *Sun Microsystems*, y que cualquier programador puede descargar libremente. Además, la compañía *Sun*, creadora de *Java*, distribuye gratuitamente el *Java(tm) Development Kit* (*JDK*). Se trata de un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en *Java*.



Incorpora además la posibilidad de ejecutar parcialmente el programa, deteniendo la ejecución en el punto deseado y estudiando en cada momento el valor de cada una de las variables (con el denominado *Debugger*). Cualquier programador con un mínimo de experiencia sabe que una parte muy importante (muchas veces la mayor parte) del tiempo destinado a la elaboración de un programa se destina a la *detección y corrección de errores*. Existe también una versión reducida del *JDK*, denominada *JRE* (*Java Runtime Environment*) destinada únicamente a ejecutar código *Java* (no permite compilar).

Los *IDE*s (*Integrated Development Environment*), tal y como su nombre indica, son entornos de desarrollo integrados. En un mismo programa es posible escribir el código *Java*, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Algunos incluyen una herramienta para realizar *Debug* gráficamente, frente a la versión que incorpora el *JDK* basada en la utilización de una consola (denominada habitualmente ventana de comandos de MS-DOS, en *Windows NT/95/98*) bastante difícil y pesada de utilizar. Estos entornos integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos librerías con *componentes* ya desarrollados, los cuales se incorporan al proyecto o programa. Como inconvenientes se pueden señalar algunos fallos de compatibilidad entre plataformas, y ficheros resultantes de mayor tamaño que los basados en clases estándar.

Capítulo 2

Análisis de requisitos y diseño

2.1. ANÁLISIS DE REQUISITOS

Los requisitos mínimos que debe caracterizar un dispositivo capaz de ejecutar la aplicación que se ha desarrollado varían en función del programa que el usuario desee utilizar.

En nuestro caso, para el desarrollo, compilación y ejecución de la totalidad del código que compone la aplicación que nos ocupa, se ha utilizado la versión 3.3.0 de *Eclipse Europa* con la siguiente configuración en lo que a memoria se refiere:

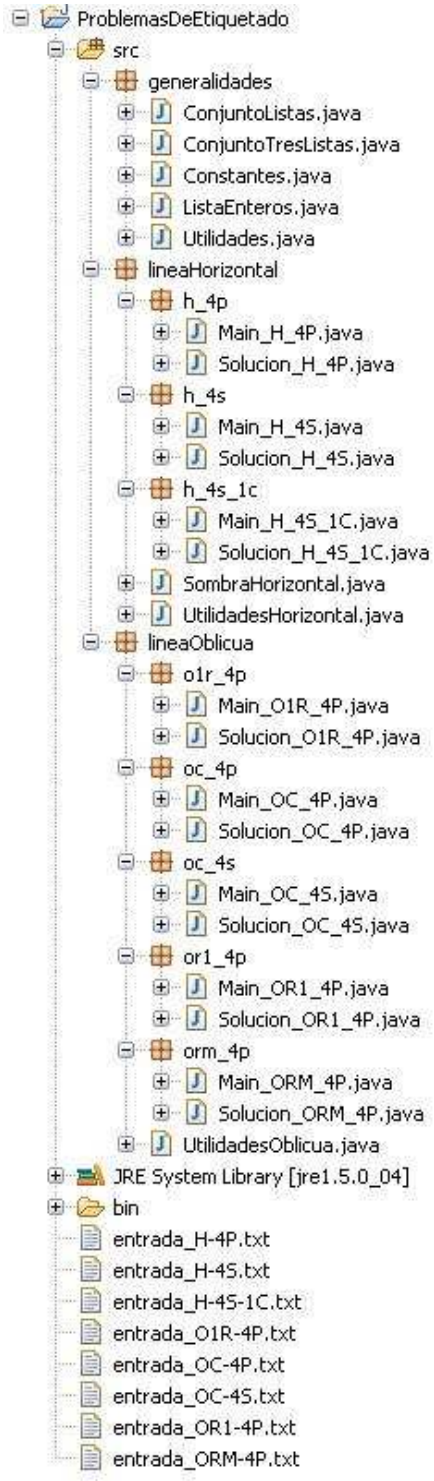
```
-vmargs
-Xms512m
-Xmx512m
```

Nótese que los parámetros anteriormente mencionados se han de ajustar para mejorar el rendimiento de eclipse, disponiendo de un fichero de configuración situado en la misma carpeta que el ejecutable llamado *eclipse.ini*.

Estos parámetros indican a la máquina virtual de Java las condiciones de memoria con las que puede trabajar eclipse. Si queremos que su rendimiento mejore, tendremos que aumentarlas, así además de conseguir una ejecución más fluida evitaremos los molestos errores de "*Out of memory*".

2.2. DISEÑO

El árbol siguiente muestra la organización estructural de la aplicación desarrollada:



2.3. ANÁLISIS TEMPORAL

En primer lugar, el proyecto se divide en las distintas tareas que lo componen. Así, inicialmente se pudo estimar de forma aproximada y general la carga temporal de cada una de dichas tareas, siendo durante el desarrollo de las mismas cuando se conoce su duración exacta.

TAREA	ESTIMACIÓN INICIAL	CARGA EXACTA
ESTUDIO PRELIMINAR	5 horas	5 horas
INVESTIGACIÓN	20 horas	10 horas
ANÁLISIS Y DISEÑO	30 horas	40 horas
IMPLEMENTACIÓN	150 horas	170 horas
PRUEBAS	15 horas	25 horas
MEMORIA	35 horas	25 horas
PRESENTACIÓN	15 horas	10 horas
TOTAL	**270 horas**	**285 horas**

Capítulo 3

Problemas de decisión en etiquetado de puntos alineados

Estudiaremos en este capítulo los problemas de decisión de etiquetado de puntos alineados con etiquetas rectangulares tanto en el caso de etiquetas que han de ser colocadas de forma que el punto quede situado en alguno de los vértices del rectángulo (etiquetas fijas), siendo finito el número de posibles soluciones; como si las etiquetas pueden ser situadas de forma que el punto quede ubicado en alguno de los lados del rectángulo (etiquetas deslizantes), existiendo en este último caso un número infinito de soluciones.

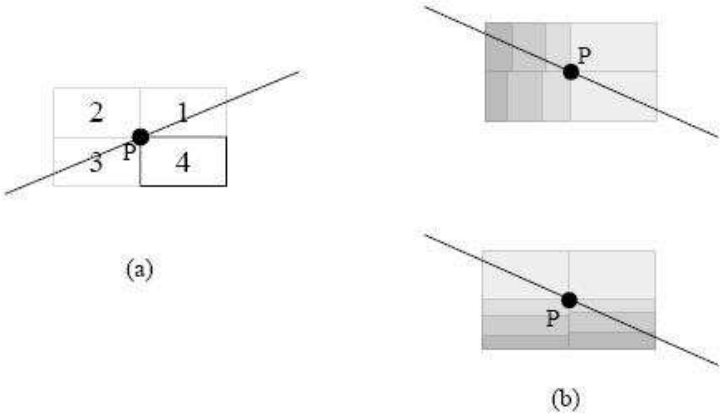


Figura 3.1.: Posiciones de etiquetas fijas (a) y deslizantes (b)

El estudio de todos los problemas lo llevaremos a cabo mediante un mismo algoritmo de tipo incremental. A la presentación de este algoritmo dedicamos el *Capítulo 4*. En los siguientes dos capítulos realizamos el estudio de los diferentes problemas, agrupando dichos problemas según el tipo de línea en la que están situados los puntos. El *Capítulo 5* está centrado en el estudio de los problemas de etiquetado de puntos situados sobre una línea horizontal (o vertical) y, en el *Capítulo 6* los derivados del etiquetado de puntos situados sobre una línea oblicua.

En función de la cantidad o tipo de información que debe contener la etiqueta, la forma y dimensiones de ésta puede variar. Tenemos así tres parámetros que caracterizarán al problema de etiquetado:

Para designar cada problema utilizaremos una notación que resalte los elementos diferenciadores señalados: “LÍNEA [ETIQUETA]–POSICIÓN”. En primer lugar “LÍNEA” se refiere al tipo de línea en la que están situados los puntos, así utilizaremos “H [ETIQUETA]–POSICIÓN” para problemas de etiquetado de puntos que están situados sobre una línea horizontal, y para los casos de puntos sobre una línea oblicua: “O [ETIQUETA]–POSICIÓN”.

El elemento “ETIQUETA” indicará el tipo de etiqueta a utilizar. Los casos que estudiaremos serán “LÍNEA C–POSICIÓN” cuando todas las etiquetas son cuadrados iguales, “LÍNEA R(*m*)–POSICIÓN” cuando éstas son rectángulos de *m* posibles dimensiones diferentes (el caso particular “LÍNEA R(1)–POSICIÓN” corresponderá al caso en que todas las etiquetas son rectangulares e iguales entre sí), “LÍNEA 1R–POSICIÓN” si son rectángulos diferentes de altura 1, no indicándose ningún valor para este elemento si todas las etiquetas fueran rectangulares sin ninguna restricción, “LÍNEA–POSICIÓN”.

Por último, el elemento “POSICIÓN” se refiere a la posición que puede tomar la etiqueta sobre el punto en cuestión, siendo “LÍNEA [ETIQUETA]–4P” si el punto puede quedar ubicado en cualquiera de los vértices de la etiqueta y “LÍNEA [ETIQUETA]–4S”si pudiera quedar ubicado sobre cualquiera de los lados de la misma.

Por ejemplo “H–4P”es utilizado para referirnos al problema “ETIQUETAS FIJAS SOBRE UNA LÍNEA HORIZONTAL”, de etiquetado de puntos sobre una línea horizontal, utilizando etiquetas rectangulares de cualquier tamaño, y de forma que las etiquetas pueden ser colocadas de manera que el punto coincida con alguno de los vértices del rectángulo, “O C–4S”para referirnos al problema de decidir si puede ser ubicado un cuadrado de lado *l* sobre cada uno de los puntos situados sobre una línea oblicua, de manera que dicho punto quede sobre algunos de los lados del cuadrado, etc.

Resulta de interés mencionar que la totalidad de *Lemas*, *Teoremas* y demás afirmaciones que aparecen a lo largo de esta memoria se encuentran debidamente demostrados en la documentación [1] presentada por D. Pedro Reyes Columé para optar al grado de Doctor en Matemáticas por la Universidad de Sevilla, con fecha de Septiembre de 2002.

Capítulo 4

Descripción del Algoritmo Incremental

Para el estudio de todos estos problemas, seguiremos un mismo algoritmo de tipo incremental, el cual se basa en la siguiente base: en cada paso dispondremos de la situación parcial del problema para un número determinado *k* de puntos de la entrada y tratamos de descubrir qué ocurrirá si consideramos un nuevo punto, de manera que las soluciones parciales del problema para los primeros *k+1* puntos las obtendremos a partir de las soluciones parciales del mismo para los primeros *k* puntos. Lógicamente para ello será necesario tener previamente ordenados los puntos que constituyen la entrada del problema.

Sea el problema genérico de etiquetado sobre una línea:

ETIQUETADO SOBRE UNA LÍNEA *E(P,L)*:

ENTRADA: Un conjunto de puntos $P = \{P_1, \ldots\ldots, P_n\}$ alineados y un conjunto de etiquetas $L = \{L_1, \ldots\ldots, L_n\}$ de lados paralelos a los ejes coordenados.

PREGUNTA: ¿Puede colocarse la etiqueta L_i sobre cada punto P_i, con el modelo de etiquetado *E*, sin que se produzcan intersecciones entre las etiquetas?

Para algunos problemas encontraremos algoritmos polinomiales que los resuelven. En tal caso, si tenemos una instancia ordenada del problema y consideramos únicamente el primer punto y su etiqueta, la respuesta podremos ofrecerla de inmediato, de hecho en cualquiera de los problemas que estudiaremos, y que serán presentados en los capítulos que siguen, esta respuesta es afirmativa, ya que siempre se puede etiquetar un sólo punto con cualquiera de estos modelos. Diseñaremos una estrategia de manera que si disponemos de la respuesta a la pregunta en el caso de considerar sólo las primeras *k* etiquetas, nos permita decidir, usando tiempo y espacio polinomial, cuál será la respuesta para el caso de considerar las primeras *k+1* etiquetas.

Como se dijo con anterioridad, los puntos han de estar alineados. En caso contrario, al coste computacional del problema habría que añadir el coste *O(n log n)* correspondiente al proceso de ordenación.

Sea una entrada del problema, formada por un conjunto ordenado $P = \{P_1, \ldots\ldots, P_n\}$ de puntos situados sobre una recta l, llamada *línea de entrada*, y para cada punto P_i una etiqueta rectangular L_i. Diremos que $R_k = \{r_1, \ldots\ldots, r_k\}$ es una *k-realización de P* para el modelo E si cada r_i $(i = 1, \ldots\ldots, k)$ representa una posición de L_i válida según el modelo E, de manera que las etiquetas no intersecan entre sí. Por tanto una k-realización será una configuración que resolvería el problema parcial (considerados los k primeros puntos), ya que su existencia indica la respuesta afirmativa al problema parcial.

Si queremos tener garantizar que nuestro algoritmo pueda decidir si después de realizar el paso k se puede añadir la etiqueta correspondiente al punto P_{k+1}, tendríamos que considerar el conjunto de todas las k-realizaciones y comprobar si puede ser añadida la etiqueta L_{k+1} en cualquiera de las posiciones permitidas por el modelo de etiquetado E. Pero el número de k-realizaciones posibles crece exponencialmente, lo que haría intratable el problema. No obstante este inconveniente lo evitaremos introduciendo el concepto de "*sombra de una k-realización*", que nos permitirá establecer una clasificación del conjunto de k-realizaciones en el "conjunto de sombras" (clases de k-realizaciones con la misma sombra). Introduciremos asimismo una ordenación de este conjunto, de manera que podremos manejar el concepto de minimalidad, en el sentido de que garanticemos que si no puede ser añadida la etiqueta L_{k+1} a una k-realización mínima (cuya sombra es mínima) entonces no puede ser añadida con éxito a ninguna otra k-realización, por lo que nos bastará llevar el control del número de k-realizaciones mínimas en cada paso k y hacerlo obviamente en tiempo y espacio polinomial.

Dada una etiqueta L colocada en una determinada posición r, según el modelo E, llamamos *vértice saliente* de dicha etiqueta al punto de la frontera de L que está más alejado según la dirección de la línea de entrada y llamamos *sombra s(r) de la etiqueta L en la posición r* al cuadrante del plano definido por los lados de la etiqueta L colocada en la posición r que contienen a su vértice saliente si éste está situado como origen de coordenadas. (Véase *Figura 4.1*).

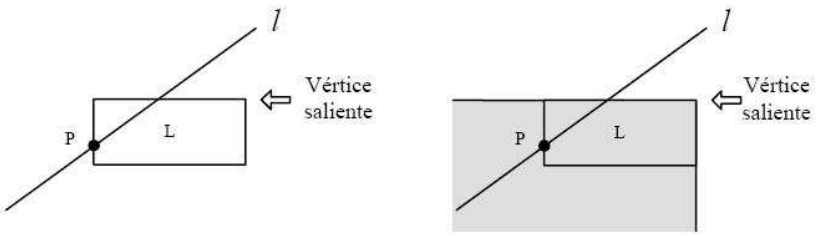


Figura 4.1.: Sombra de una etiqueta

Igualmente dada una *k*-realización $R_k = \{r_1, \ldots\ldots, r_k\}$ llamamos *sombra* $s(R_k)$ *de la realización* R_k a la unión de las sombras de sus etiquetas, $s(R_k) = \bigcup_{i=1}^{k} s(r_i)$. La *Figura 4.2* muestra un ejemplo.

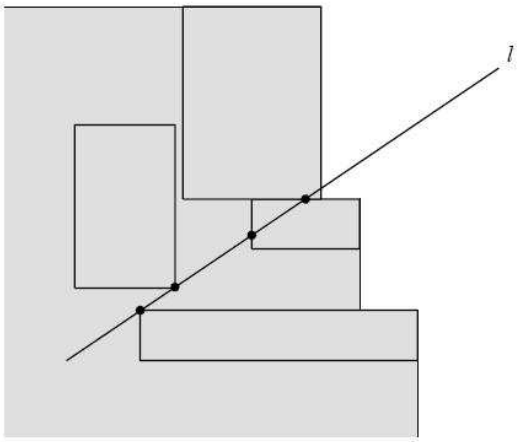


Figura 4.2.: Unión de sombras de etiquetas en una 4-realización

Una vez introducidos estos conceptos previos de carácter general, pasamos a estudiar los distintos problemas concretos, desarrollados en los *Capítulos 5* y *6*.

Capítulo 5

Puntos sobre una línea horizontal

En primer lugar procederemos a dividir la línea horizontal en segmentos del tamaño de un carácter, ya que sería imposible tomar la línea como un conjunto infinito de posiciones donde colocar los puntos a etiquetar. De este modo, para el desarrollo de los distintos algoritmos que nos ayuden a resolver el problema de etiquetado sobre líneas horizontales, se entenderá la línea de entrada de la siguiente manera:

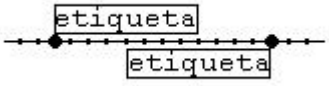


Figura 5.1.: Línea de entrada horizontal

Puede observarse en la figura anterior que la altura de la etiqueta no influye en el etiquetado sobre líneas horizontales. De este modo, para todos los problemas de este capítulo tomaremos las etiquetas de altura *1*.

Una vez aclarado este punto, y como señalábamos anteriormente, los problemas van a ser divididos en función del tipo de ubicación de las etiquetas (fijas o deslizantes). Veremos cómo esta diferencia en el tipo de etiqueta juega un papel muy importante en la complejidad del problema. Como ya hemos señalado, la razón principal de este hecho es que si estamos estudiando problemas de etiquetado con etiquetas fijas, cada etiqueta *L*, correspondiente a un punto *P*, puede ser colocada en cualquiera de los cuatro vértices del rectángulo. La posición que adopte la etiqueta la denotaremos con un número $r \in \{1, 2, 3, 4\}$ que indica el cuadrante en el que se encuentra ubicada la etiqueta, siempre en relación a un sistema de coordenadas con origen en el punto P_i. En cambio, si estudiamos problemas de etiquetado con etiquetas deslizantes, las posiciones posibles para cada etiqueta son infinitas.

Los casos siguientes son los que estudiaremos dada una línea horizontal:

* ETIQUETAS FIJAS SOBRE UNA LÍNEA HORIZONTAL (*H–4P(P,L)*)
  - ➔ El punto puede situarse sobre cualquiera de los vértices de la etiqueta, tratándose ésta de un rectángulo cualquiera.
  - ➔ Véase la "*SECCIÓN 5.1*"

* ETIQUETAS DESLIZANTES IGUALES SOBRE UNA LÍNEA HORIZONTAL (*H–4S-1C(P,L)*)
    - ➔ El punto se puede situar libremente sobre el contorno de la etiqueta, siendo ésta siempre un cuadrado unitario de dimensión *1x1*.
    - ➔ Véase la "*SECCIÓN 5.2*"

* ETIQUETAS DESLIZANTES SOBRE LÍNEA HORIZONTAL (*H–4S(P,L)*)
    - ➔ Del mismo modo que el anterior caso, el punto puede situarse libremente sobre el contorno de la etiqueta, que esta vez no tiene tamaño preestablecido.
    - ➔ Véase la "*SECCIÓN 5.3*"

SECCIÓN 5.1.
ETIQUETAS FIJAS
SOBRE UNA LÍNEA HORIZONTAL

(*H–4P(P,L)*)

ENTRADA: Un conjunto $P=\{P_1, \ldots\ldots, P_n\}$ de n puntos situados sobre una línea horizontal y un conjunto $L=\{L_1, \ldots\ldots, L_n\}$ de n rectángulos de lados paralelos a los ejes coordenados.

PREGUNTA: ¿Se puede colocar cada rectángulo L_i en su punto correspondiente P_i, de tal forma que dicho punto quede situado sobre alguno de los vértices del rectángulo sin que se produzcan intersecciones entre dicho conjunto de rectángulos?

Como ya indicábamos, para el desarrollo del algoritmo incremental es necesario que los puntos de la entrada del problema estén ordenados. Si esto no fuera así tendremos que añadir el coste computacional $O(n \cdot log(n))$ derivado de su previa ordenación. Supondremos que están ordenados en orden creciente según el valor de su abscisa.

Siguiendo el proceso incremental señalado, comenzando con la etiqueta L_1, trataremos en cada paso de añadir una nueva etiqueta, de manera que si conseguimos colocar la etiqueta L_n el problema daría respuesta afirmativa, y si en algún paso no se pudiera añadir la correspondiente etiqueta, el método utilizado nos debe garantizar que la respuesta del problema sería negativa.

Como la posición de cada etiqueta viene expresada por un número $r \in \{1, 2, 3, 4\}$ (como se apreciaba en la *Figura 3.1*), una k-realización vendrá determinada por una secuencia de números $R_k = \{r_1, \ldots\ldots, r_k, \mid r_i = 1, 2, 3, 4\}$.

Dada una k-realización R_k, si llamamos t_k a la abscisa del vértice saliente de la etiqueta situada más a la derecha sobre la línea de entrada y b_k a la de la situada más a la derecha bajo dicha línea, su sombra viene determinada por el par (t_k, b_k) (véase la *Figura 5.1.1*).

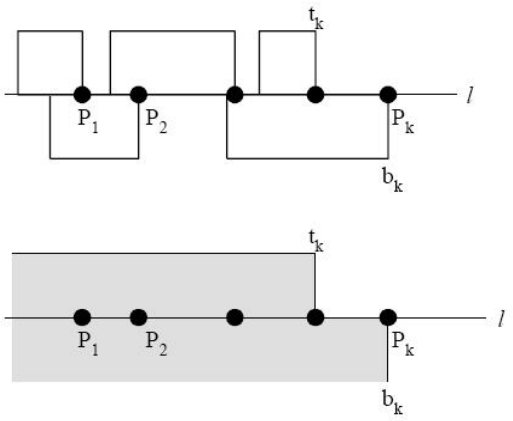


Figura 5.1.1: Sombra de una k-realización

Si en una *k*-realización efectuamos una simetría respecto de la línea de entrada, obtenemos otra k-realización. Esto nos proporciona un fenómeno de *dualidad*. Sea $R_k = \{r_1, \ldots\ldots, r_k\}$ una *k*-realización, se llama *dual* de R_k a la k-realización $R_k^* = \{r_1^*, \ldots\ldots, r_k^*\}$, siendo $r_i^* = 5 - r_i$, para $i = 1, \ldots\ldots, k$. La k-realización R_k^* dual es por tanto la simétrica de la *k*-realización R_k respecto de la línea de entrada (horizontal, en nuestro caso).

Tal como decíamos en el *Capítulo 4,* definimos a continuación la relación que nos permitirá establecer la clasificación del conjunto de *k*-realizaciones en clases de *k*-realizaciones con la misma sombra *s*. Dos *k*-realizaciones R_k y R'_k se dicen que son *equivalentes*, y lo denotaremos por $R_k \sim R'_k$, si $s(R_k) = s(R'_k)$ o $s(R_k^*) = s(R'_k)$.

Dos *k*-realizaciones son por tanto equivalentes si producen sombras iguales. En la *Figura 5.1.2* se muestran dos *k*-realizaciones equivalentes.

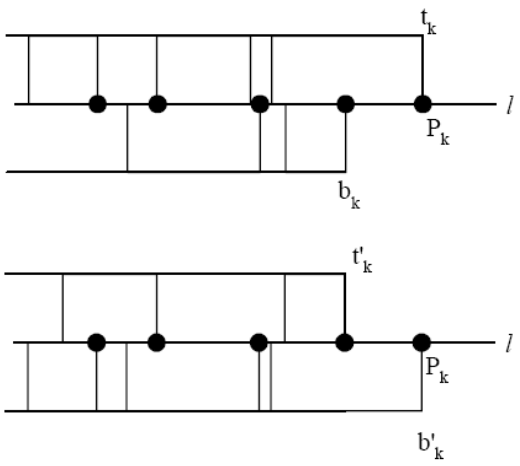


Figura 5.1.2: Dos k-realizaciones equivalentes

Esta definición nos proporciona una relación de equivalencia que clasifica el conjunto de *k*-realizaciones en clases que tienen en común la misma sombra.

Una vez clasificado el conjunto de *k*-realizaciones, introducimos la relación de orden entre ellas. De una forma intuitiva una *k*-realización será menor que otra si deja un mayor espacio libre para poder colocar la etiqueta L_{k+1}. Entonces, formalmente, dadas dos *k*-realizaciones R_k y R'_k, diremos que $R_k \leq R'_k$ si $s(R_k) \subseteq s(R'_k)$ o $s(R^*_k) \subseteq s(R'_k)$.

Obsérvese que, en el caso de etiquetado sobre una línea horizontal que nos ocupa, si identificamos las sombras de las k-realizaciones R_k y R'_k mediante los pares (t_k, b_k) y (t'_k, b'_k), la relación de orden se puede definir de la forma siguiente:

$$(t_k, b_k) \leq (t'_k, b'_k) \Longleftrightarrow \begin{cases} t_k \leq t'_k \quad y \quad b_k \leq b'_k \\ \text{o} \\ t_k \leq b'_k \quad y \quad b_k \leq t'_k \end{cases}$$

Lema 5.1.1. *La relación previamente definida es una relación de orden parcial en el conjunto de clases de k-realizaciones.*

La *Figura 5.1.3* ilustra esta relación de orden entre realizaciones mediante sus sombras.

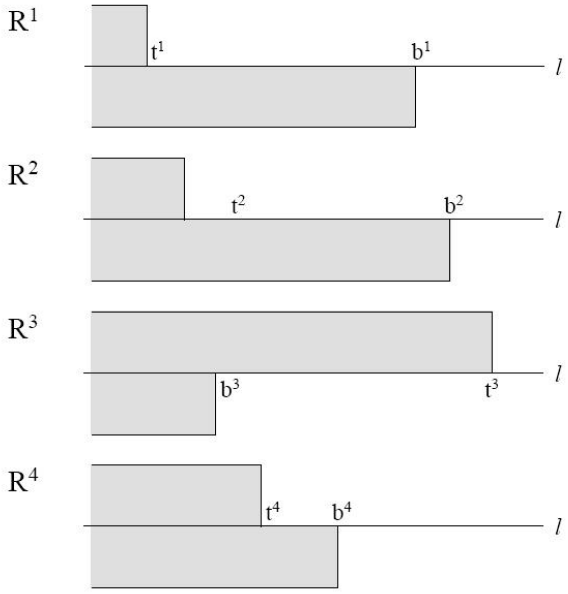


Figura 5.1.3: $R^1 \leq R^2$, $R^2 \leq R^3$, R^4 no es comparable con las demás.

Atendiendo a la figura anterior, hay sombras que no son comparables entre sí (por ejemplo R^3 y R^4), en cuyo caso se tomará como sombra menor cualquiera que sea válida (aleatoriamente).

Esta relación de orden entre sombras de *k*-realizaciones nos permite introducir el concepto de minimalidad en el conjunto de *k*-realizaciones, en el sentido de que una *k*-realización será mínima si no hay otra que ocupe una sombra menor y por tanto ofrezca más posibilidades de colocación de la etiqueta L_{k+1} que ella. Por tanto una *k*-realización R_k se dice que es una k-realización *mínima* si no existe ninguna otra *k*-realización R'_k tal que $R'_k \leq R_k$.

En la *Figura 5.1.4* aparecen dos *k*-realizaciones, R^2 es mínima y R^1 no lo es ($R^2 \leq R^1$).

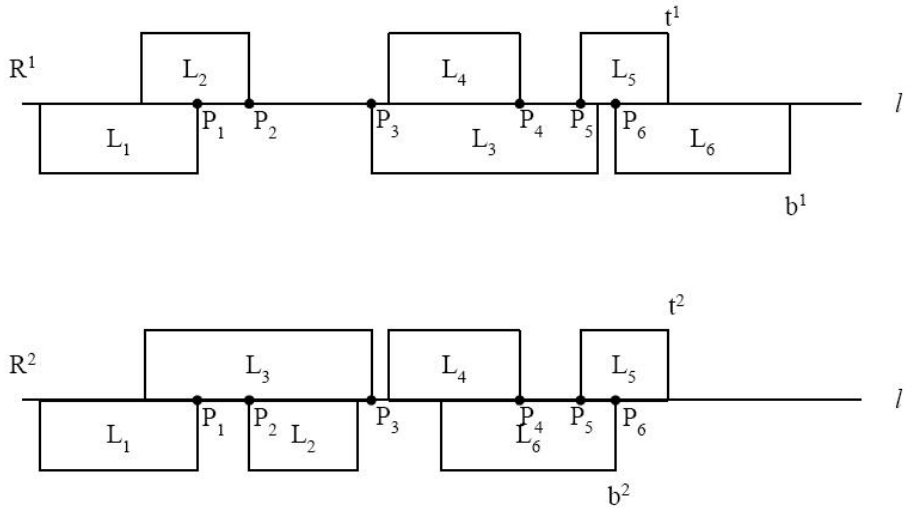


Figura 5.1.4: R^2 es mínima ($R^2 \leq R^1$).

Podemos asegurar por tanto que para poder decidir si existe una (*k+1*)-realización basta con comparar la etiqueta correspondiente al punto P_k+1 con las *k*-realizaciones mínimas.

Lema 5.1.2. *Si existe una (k+1)-realización, entonces debe existir otra (k+1)-realización de tal forma que sus k primeros elementos forman una k-realización mínima, para k = 1, ……, n - 1.*

Si en el paso *k* tenemos un conjunto de *k*-realizaciones mínimas, atendiendo a este lema, en el paso *k+1* de nuestro algoritmo incremental bastará decidir si la etiqueta L_{k+1} puede ser añadida con éxito a alguna de las *k*-realizaciones mínimas. Si no es así, la respuesta al problema sería negativa. En cambio, si la etiqueta L_{k+1} puede ser añadida a alguna de las *k*-realizaciones mínimas, tendremos que considerar todas las posibles (*k+1*)-realizaciones y elegir aquellas que son mínimas, para continuar con el paso siguiente.

No obstante, esta afirmación carecería de interés si no consiguiésemos controlar el número de realizaciones mínimas. Esto lo hacemos en el siguiente lema.

Lema 5.1.3. *En el problema H–4P existen a lo sumo dos k-realizaciones mínimas no equivalentes para k = 1, ……., n.*

Teorema 5.1.4. *Si los puntos están previamente ordenados sobre la línea horizontal, el problema H–4P se puede resolver en tiempo y espacio lineal.*

* Si el usuario lo desea puede introducir los datos a través de un fichero de texto, cuyo formato será el que se indica a continuación:

  LÍNEA *1*: número entero que indica el número de elementos (*N*).
  LÍNEA *2* a LÍNEA (*N+1*): números enteros separados por salto de línea que representan los puntos donde se desea establecer las etiquetas.
  LÍNEA (*N+2*) a LÍNEA (*2N+1*): números enteros separados por salto de línea que representan las etiquetas a establecer sobre los puntos.

  Por ejemplo, para indicar que se pretenden etiquetar 3 puntos (6, 7 y 9) con etiquetas de tamaño (2, 5 y 8), el fichero de texto será:

```
3
6
7
9
2
5
8
```

APARTADO 5.1.1.
IMPLEMENTACIÓN DEL PROBLEMA (*H–4P(P,L)*)

Por medio de la clase principal "`Main_H_4P`", en primer lugar se solicita al usuario que indique si desea introducir los datos manualmente o, por el contrario, si desea que la aplicación obtenga los valores de los puntos y etiquetas a partir de un fichero.

En el primer caso, se solicita al usuario que introduzca el número de puntos ("`numElementos`") que se desean etiquetar. Lógicamente, el número de puntos a etiquetar será igual al número de etiquetas, por lo que se crean dos listas ("`puntos`" y "`etiquetas`") de longitud "`numElementos`" que representarán al conjunto de puntos y al conjunto de etiquetas, respectivamente. Una vez definidas y creadas las listas, se procede a cumplimentarlas con los datos que el propio usuario introduzca a través del teclado cuando les sean solicitados.

Si se especifica que los datos provienen de un fichero, el propio programa procederá a leerlos, obteniendo, por este orden, el número de elementos ("`numElementos`"), la lista de puntos ("`puntos`") y la lista de etiquetas ("`etiquetas`"). Si se produce algún error del tipo:

- No se encuentra el fichero
- Error de entrada/salida
- El número de elementos no coincide con el número de puntos o de etiquetas

la aplicación lanzará la excepción que corresponda, indicando el tipo de error sucedido y finalizando la ejecución del programa.

Conviene aclarar que, durante el proceso, se representará cada etiqueta como un número entero que indique su longitud.

En el momento en que conocemos el número de elementos, los valores de los puntos y las longitudes de las etiquetas, estamos en disposición de proceder a la resolución del problema. Para ello creamos un objeto "`solucion`" utilizando el constructor con parámetros de la clase "`Solucion_H_P4`", al que le pasamos como parámetros las listas de puntos y longitudes de etiquetas, siendo trivial el cálculo del número de elementos del problema (longitud de cualquiera de las dos listas). Habiendo creado este objeto, sólo tendremos que invocar al

método "obtenerSolucion", que nos resolverá el problema en el caso que sea posible.

Lo primero que hay que comprobar es que los valores de puntos y etiquetas introducidos sean válidos, entendiendo como valor válido todo número entero mayor que cero. Esta comprobación la lleva a cabo el método "valoresValidos". Una vez verificado este aspecto, se verifica que la lista de puntos esté ordenada ascendentemente llamando al método "puntosOrdenados". Si el usuario la ha introducido correctamente ordenada, continuamos con la resolución del problema. En caso contrario, el método "ordenarListasInteger" ordenará simultáneamente las listas de puntos y etiquetas, manteniendo la correspondencia de los pares punto-etiqueta.

Hechas ya las comprobaciones previas, y asegurándonos que todos los datos son correctos, se procede a inicializar los valores que utilizaremos a lo largo de la resolución del problema. Para ello sólo tendremos que invocar al método "inicializaValores", el cual creará y/o inicializará los siguientes objetos:

- Un objeto "sombra" de tipo "SombraHorizontal" que contendrá el valor de la sombra en cada instante. Un objeto de este tipo vendrá definido por los pares "Tk, Bk".
- Un número entero "longitudLinea" que, como su propio nombre indica, se trata de la longitud de la línea horizontal a etiquetar.
- Dos listas de Integer "lineaSuperior" y "lineaInferior", que serán necesarias para representar gráficamente las etiquetas sobre la línea horizontal y bajo la línea horizontal, respectivamente.
- Una lista de String "resultado", que indicará la posición de la etiqueta "etiquetas.get(i)" respecto del punto "puntos.get(i)". Los posibles valores que puede tomar cada elemento de esta lista son:
    - "TL" : arriba a la izquierda (top left)
    - "BL" : abajo a la izquierda (bottom left)

- "TR" : arriba a la derecha (top right)
- "BR" : abajo a la derecha (bottom right)
- "- -" : no se ha podido situar la etiqueta sobre el punto

Además de estos objetos, la clase "`Solucion_H_P4`" dispondrá también de los siguientes:

- Dos listas "`puntos`" y "`etiquetas`" que serán copias exactas de las listas construidas en la clase "`Main_H_P4`" a partir de los valores introducidos por el usuario.
- Un número entero "`numElementos`" que contendrá el número total de puntos o etiquetas.

Disponiendo ya de todos los valores y objetos necesarios para el caso que nos ocupa, procedemos a la resolución del problema.

Para cada par punto-etiqueta (y mientras sea posible encontrar una solución correcta), comprobamos si la etiqueta se puede situar sobre el punto, quedando éste en uno de los vértices de dicha etiqueta (TL, BL, TR, BR). La metodología es la siguiente: para el primer punto, obtenemos todas las posibles sombras resultantes de haber colocado la primera etiqueta mediante los métodos "`probarEtiquetaTL`", "`probarEtiquetaBL`", "`probarEtiquetaTR`" y "`probarEtiquetaBR`". De las sombras halladas escogemos la menor y etiquetamos el punto utilizando el método "`aplicarEtiquetaTL`", "`aplicarEtiquetaTR`", "`aplicarEtiquetaBL`" o "`aplicarEtiquetaBR`", según la posición que hayamos elegido, actualizando la sombra "`sombra`". A continuación, con el nuevo valor de sombra procedemos de igual manera a etiquetar el segundo punto, y así sucesivamente hasta llegar al último punto.

Como se ha comentado anteriormente, en el momento en que no sea posible etiquetar un solo punto, se detendrá el bucle y se imprimirá por pantalla el resultado parcial:

Por ejemplo:

```
Se han obtenido las siguientes posiciones:
( TL , TR , BL , BR )
```

La primera etiqueta estaría situada arriba y a la izquierda sobre el primer punto; la segunda etiqueta, arriba y a la derecha; la tercera etiqueta, abajo y a la izquierda; y la cuarta etiqueta, abajo y a la derecha.

Si, por el contrario, todos los puntos pueden ser etiquetados, el bucle finalizará después de tratar el último de ellos e, igualmente, imprimirá por pantalla el resultado (esta vez, final):

Por ejemplo:

```
Se han obtenido las siguientes posiciones:
( TL , TR , BL , BR , -- )
No se ha podido encontrar una solución
válida a este problema con los datos que ha
proporcionado.
```

La primera etiqueta estaría situada arriba y a la izquierda sobre el primer punto; la segunda etiqueta, arriba y a la derecha; la tercera etiqueta, abajo y a la izquierda; y la cuarta etiqueta, abajo y a la derecha. Habría sido imposible situar la última etiqueta.

El método encargado de comparar y seleccionar una de las posibles sombras resultantes de etiquetar un determinado punto es "`hallarSombraMenor4`". En el caso en que sólo existan tres posibles posiciones, "`hallarSombraMenor3`". Y si sólo hay dos posiciones válidas, "`sombraMenorIgualQue`". Habrá que pasarles 4, 3 o 2 sombras como parámetros, según el que se utilice. La siguiente figura indica la sombra escogida, siendo *S1* el primer parámetro, *S2* el segundo, *S3* el tercero y *S4* el cuarto parámetro:

Si comparamos cuatro sombras:

$$\begin{cases} S1 \le S2 \begin{cases} S3 \le S4 \begin{cases} S1 \le S3 \rightarrow S1 \\ S1 > S3 \rightarrow S3 \end{cases} \\ S3 > S4 \begin{cases} S1 \le S4 \rightarrow S1 \\ S1 > S4 \rightarrow S4 \end{cases} \end{cases} \\ S1 > S2 \begin{cases} S3 \le S4 \begin{cases} S2 \le S3 \rightarrow S2 \\ S2 > S3 \rightarrow S3 \end{cases} \\ S3 > S4 \begin{cases} S2 \le S4 \rightarrow S2 \\ S2 > S4 \rightarrow S4 \end{cases} \end{cases} \end{cases}$$

Si comparamos tres sombras:

$$\begin{cases} S1 \le S2 \begin{cases} S1 \le S3 \rightarrow S1 \\ S1 > S3 \rightarrow S3 \end{cases} \\ S1 > S2 \begin{cases} S2 \le S3 \rightarrow S2 \\ S2 > S3 \rightarrow S3 \end{cases} \end{cases}$$

Si comparamos sólo dos sombras:

$$\begin{cases} S1 \le S2 \rightarrow S1 \\ S1 > S2 \rightarrow S2 \end{cases}$$

Figura 5.1.1.1.: Criterio de elección de posición de una etiqueta.

* Clases y métodos utilizados para la resolución de este apartado:

-CLASE-	-MÉTODO-
Main_H_4P	main
Solucion_H_4P	obtenerSolucion
	inicializaValores
	probarEtiquetaTL
	probarEtiquetaBL
	probarEtiquetaTR
	probarEtiquetaBR
	aplicarEtiquetaTL
	aplicarEtiquetaBL
	aplicarEtiquetaTR
	aplicarEtiquetaBR
Utilidades	valoresValidos
	puntosOrdenados
	ordenarListasInteger
Constantes	N/A
SombraHorizontal	getTk
	setTk
	getBk
	setBk
UtilidadesHorizontal	sombraMenorIgualQue
	hallarSombraMenor3
	hallarSombraMenor2
	representar
ConjuntoListas	getListaPuntos
	getListaEtiquetas

SECCIÓN 5.2.
ETIQUETAS DESLIZANTES IGUALES SOBRE UNA LÍNEA HORIZONTAL

(*H–4S-1C(P,L)*)

ENTRADA: Un conjunto $P=\{P_1, \ldots, P_n\}$ de n puntos situados sobre una línea horizontal.

PREGUNTA: ¿Se puede colocar en cada punto P_i un cuadrado C unitario, de tal forma que dicho punto quede situado sobre alguno de los lados horizontales del cuadrado, sin que se produzcan intersecciones entre ellos?

Si tratamos de seguir el algoritmo incremental de los problemas anteriores observamos que el número de *k*-realizaciones mínimas no se mantiene constante. No obstante, debido a que todas las etiquetas son del mismo tamaño observamos que la etiqueta del punto $P_{k+1}(x_{k+1}, 0)$ estará situada más a la derecha del punto $(x_{k+1} - 1, 0)$ por lo que el conjunto de puntos del plano cuya abscisa es menor que $x_{k+1} - 1$ son intrascendentes para la etiqueta del punto P_{k+1} en el estudio del problema (estos puntos aparecen en la *Figura 5.2.1* con relleno discontinuo), de manera que si este nuevo conjunto lo incorporamos a la sombra de la *k*-realización entonces el número de sombras de realizaciones mínimas se reduce. En el ejemplo sólo serían mínimas las realizaciones *C* y *D*. Una de ellas tiene alternativamente las etiquetas por encima y por debajo de la línea de entrada, por este motivo introduciremos el concepto de *k-realización alternada* y veremos que será suficiente controlar estas realizaciones.

Una *k*-realización alternada se obtiene colocando las etiquetas lo más a la izquierda posible y alternativamente por encima y por debajo de la línea de entrada. Obsérvese por tanto que en caso de existir, la *k*-realización alternada es única, salvo dualidad. De ahí que nos refiramos a ella en singular.

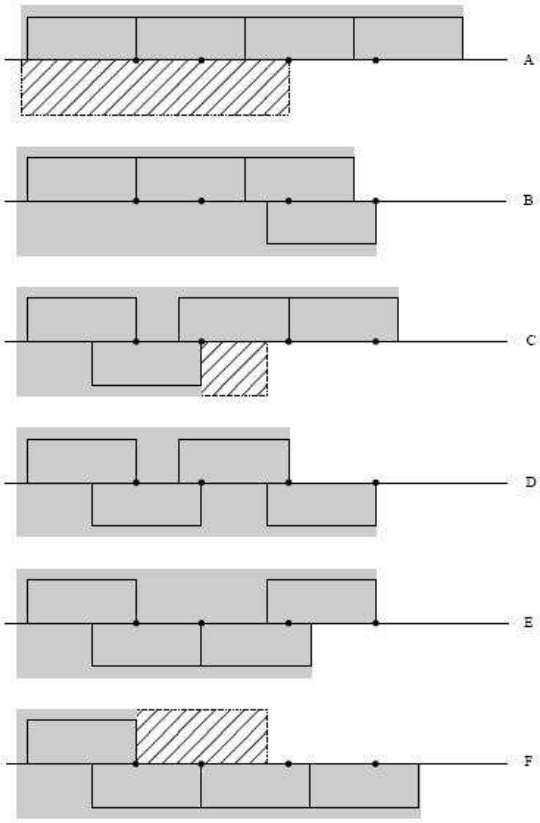


Figura 5.2.1.: k-realizaciones mínimas

Si no existe la *k*-realización alternada es debido a que la sombra de la *(k - 1)*-realización alternada contiene al punto P_k. Es decir, el punto P_k es anterior al vértice saliente de las etiquetas correspondientes a los puntos P_{k-1} y P_{k-2}. La *Figura 5.2.2* muestra esta situación para *k* par (para *k* impar la situación es simétrica). Como se aprecia en dicha figura, el punto P_{k-1}, que corresponde a una etiqueta por encima de la línea de entrada, no coincide con el vértice inferior derecho de su etiqueta L_{k-1} por lo que la etiqueta L_{k-3} debe estar impidiendo su deslizamiento hacia la izquierda, por lo tanto han de estar en contacto las etiquetas L_{k-1} y L_{k-3}, si la etiqueta L_{k-3} no está en posición *2* la etiqueta L_{k-5} debe contactar con ella y así sucesivamente hasta que encontraríamos una etiqueta en posición *2*. Igualmente el punto P_{k-2} no está en la posición *3* por lo que la etiqueta L_{k-4} está haciendo de tope para ella, y así continuamos hasta encontrar una etiqueta L_{k-2j} que esté en posición *3*. Esta etiqueta existe pues al menos L_2 es una de ellas.

Según se aprecia en la *Figura 5.2.2*, tenemos *j* etiquetas en contacto por debajo de la línea y la distancia entre los puntos P_{k-2j} y P_k es menor que *j - 1*.

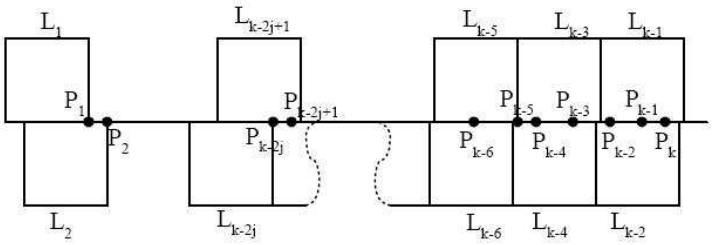


Figura 5.2.2.: No existe la k-realización alternada

Si existiera una *k*-realización cualquiera, el espacio máximo entre estos puntos se produce colocando la etiqueta L_{k-2j+1} en posición *2*, la etiqueta L_{k-1} en posición *1* y la etiqueta L_k en posición *4*, como indica la *Figura 5.2.3*.

Por lo tanto el espacio disponible encima y debajo de la línea es inferior a *j - 1* y hemos de colocar *2j - 3 = 2(j - 1) - 1* etiquetas. Cualquier realización posible tendría que llevar al menos *j - 1* de estas etiquetas en uno de los dos semiplanos, por encima o por debajo de la línea, por lo que resultaría imposible. Entonces no existe ninguna *k*-realización.

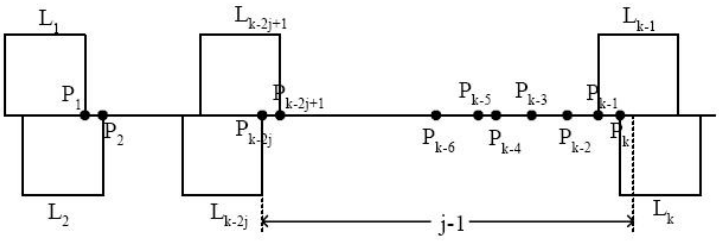


Figura 5.2.3.: Si no existe la k-realización alternada el etiquetado no es posible.

Teniendo en cuenta este resultado, para resolver el problema bastará estudiar las *k*-realizaciones alternadas.

APARTADO 5.2.1.
IMPLEMENTACIÓN DEL PROBLEMA (*H–4S-1C(P,L)*)

La implementación del problema *H-4S-1C* es muy similar a la del problema *H-4P* ya explicado, con la particularidad de que en el caso que nos ocupa, las etiquetas serán cuadradas (en lugar de rectangulares) y el etiquetado es deslizante.

En primer lugar, especificar que las clases que resuelven este problema son "`Main_H_4S_1C`" y "`Solucion_H_4S_1C`". Mediante la clase *main* se toman los valores, que el usuario deberá introducir por teclado o editando un fichero de texto tal y como indica la ayuda del problema. Una vez los datos sean correctos (y coherentes), los métodos incluidos en la clase *solución* se encargarán de encontrar una posición correcta en la que etiquetar cada punto.

La solución será del tipo:

```
Se han obtenido las siguientes posiciones:
( TL , TT , BB , BR )
```

Y si no fuese posible etiquetar sobre los puntos dados, el mensaje que aparecería en la consola será algo como:

```
Se han obtenido las siguientes posiciones:
( TL , TT , BB , BR , -- )
No se ha podido encontrar una solución
válida a este problema con los datos que ha
proporcionado.
```

Cabe destacar que, aunque el etiquetado deslizante permite situar el punto sobre los cuatro lados de la etiqueta (arriba [TT], abajo [BB], a la izquierda [LL] o a la derecha [RR]), sólo tienen sentido el etiquetado TT y el BB para este problema, ya que para el hecho de etiquetar sobre una línea horizontal no influye la altura de las etiquetas.

Nuevamente comentar que, al igual que en el problema anterior (y en todos los que componen esta memoria) se facilita una solución

gráfica, sólo la solución descrita en los párrafos anteriores es válida, siendo las demás simples aproximaciones visuales que no se deben tomar como definitivas.

* Clases y métodos utilizados para la resolución de este apartado:

-CLASE-	-MÉTODO-
Main_H_4S_1C	main
Solucion_H_4S_1C	obtenerSolucion
	inicializaValores
	probarEtiquetaArriba
	probarEtiquetaAbajo
	aplicarEtiquetaArriba
	aplicarEtiquetaAbajo
Utilidades	valoresValidos
	puntosOrdenados
	ordenarListasInteger
	imprimeListaString
	maxValorLista
Constantes	N/A
UtilidadesHorizontal	representar
ConjuntoListas	getListaPuntos
	getListaEtiquetas

SECCIÓN 5.3.
ETIQUETAS DESLIZANTES
SOBRE UNA LÍNEA HORIZONTAL

(*H–4S(P,L)*)

ENTRADA: Un conjunto $P=\{P_1, \ldots, P_n\}$ de n puntos situados sobre una línea horizontal y un conjunto $L=\{L_1, \ldots, L_n\}$ de n rectángulos de lados paralelos a los ejes coordenados.

PREGUNTA: ¿Se puede colocar cada rectángulo L_i en el punto P_i, de tal forma que dicho punto quede situado sobre alguno de los lados horizontales del rectángulo, sin que se produzcan intersecciones entre ellos?

Como se puede apreciar en la *Figura 5.3.1* el deslizamiento vertical no influye en el desarrollo del problema ya que todas las soluciones en las que la etiqueta quedará situada en algún punto de sus lados verticales son mejoradas por cualquiera de las dos soluciones correspondientes en los extremos. Por esta razón consideraremos siempre etiquetas que tienen algún lado horizontal sobre la línea de entrada.

Conservamos para este problema las nociones introducidas para el caso de etiquetas fijas, tales como la relación de orden parcial y el concepto de *k*-realización mínima. Véase de nuevo la *Figura 3.12*.

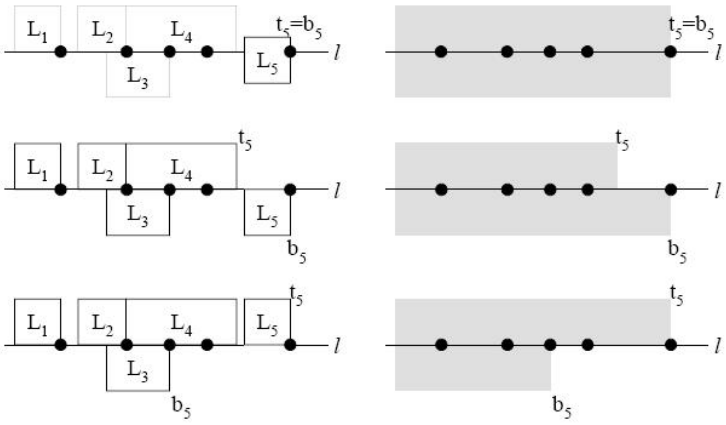


Figura 5.3.1.: Realizaciones y sombras en H-4S

APARTADO 5.3.1.
IMPLEMENTACIÓN DEL PROBLEMA (*H–4S(P,L)*)

Por medio de la clase principal "`Main_H_4S`" y la que contiene los métodos para solucionar el problema "`Solucion_H_4S`", en primer lugar se solicita al usuario que introduzca el número de puntos. Una vez se comprueba que los puntos son correctos y no existe ninguna incoherencia se procede a la resolución del problema.

En esencia, este problema y el anterior (*H-4S-1C*) se resuelven de la misma manera. La única diferencia es que en este caso tenemos etiquetas rectangulares y de tamaños diferentes en lugar de ser cuadradas e iguales.

* Clases y métodos utilizados para la resolución de este apartado:

-CLASE-	-MÉTODO-
Main_H_4S	main
Solucion_H_4S	obtenerSolucion
	inicializaValores
	probarEtiquetaArriba
	probarEtiquetaAbajo
	aplicarEtiquetaArriba
	aplicarEtiquetaAbajo
Utilidades	valoresValidos
	puntosOrdenados
	ordenarListasInteger
	imprimeListaString
	maxValorLista
Constantes	N/A
UtilidadesHorizontal	representar
ConjuntoListas	getListaPuntos
	getListaEtiquetas

Capítulo 6

Puntos sobre una línea oblicua

En muchas de las situaciones descritas previamente nos encontramos con la necesidad de colocar información sobre puntos situados en una línea oblicua, como por ejemplo en el etiquetado de líneas de metro, mapas de carreteras, etc.

Cuando nos ocupábamos de la resolución de problemas de etiquetado sobre líneas horizontales, se llegó a la conclusión de que era necesario establecer las posiciones en las que era factible etiquetar. En otras palabras, desde el punto de vista algorítmico era imposible situar puntos para su posterior etiquetado sobre las infinitas posiciones que permitiría una línea. Este contratiempo se solucionó dividiendo la línea de entrada en segmentos, por ejemplo, del tamaño de un carácter. Notar que se puede dividir la línea de entrada en tantos segmentos como se estimen necesarios, siendo el modo de operación muy similar para todos los casos.

Para el etiquetado de puntos sobre líneas oblicuas se procederá de manera similar al caso de las líneas horizontales, aunque el modo de dividir la línea de entrada será un poco más complejo: tomaremos una línea oblicua con 45° de inclinación positiva, de manera que la división se hará sobre los ejes de coordenadas *x* e *y*:

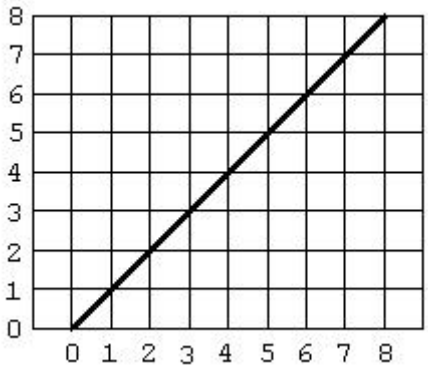


Figura 6.1.: División de los ejes de la línea de entrada oblicua

De esta manera, todo par *(x,y) , x=y*, representará un punto de la recta: *(0,0) ~ punto 0, (1,1) ~ punto 1, (2,2) ~ punto 2, (3,3) ~ punto 3*, etc. Suponiendo que estudiamos cualquier problema en el que los puntos se sitúan sobre los vértices de las etiquetas, un ejemplo de etiquetado podría ser el que se indica en la siguiente figura, en la que se aprecian tres etiquetas de tamaños *(6)*, *(4)* y *(2)*, situadas sobre los puntos (*2*), (*4*) y (*6*), respectivamente:

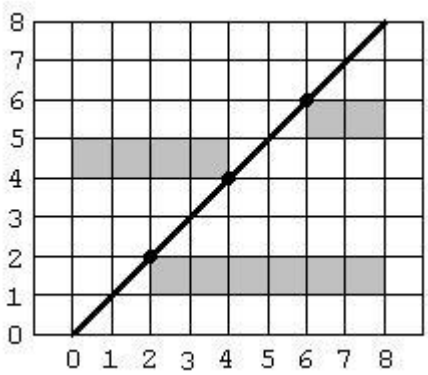


Figura 6.2.: Ejemplo de etiquetado sobre línea oblicua

Vuélvase a notar que, igual que sucediera para las líneas horizontales, se podría haber dividido la línea de entrada en más o menos segmentos (siempre de forma regular), según se estime conveniente, siendo el modo de operar muy similar en todos los casos.

Por simplificación, pasaremos de lo concreto a lo general, estudiando la complejidad de casos particulares del problema, comenzando con el problema con etiquetas cuadradas iguales e iremos generalizando el tipo de etiquetas. Concretamente estudiaremos los problemas de etiquetado de puntos sobre una línea oblicua con etiquetas rectangulares iguales fijas, etiquetas rectangulares de *m* tamaños diferentes fijas, etiquetas rectangulares de altura constante fijas y etiquetas cuadradas iguales deslizantes.

Sin pérdida de generalidad podemos suponer que los puntos se encuentran situados sobre una línea oblicua con pendiente positiva y abordaremos los problemas atendiendo a los diferentes casos dependiendo del tipo de etiquetas (cuadradas o rectangulares) y el lugar de colocación de las mismas (en los vértices o en los lados). En el primero de los casos recordemos que cada etiqueta presentaba cuatro posibilidades de colocación, mientras que el número de posibilidades para etiquetas deslizantes no era finito.

Normalmente, cuando se trata de etiquetar puntos alineados, estos puntos se encuentran sobre una línea no significativa para la aplicación o al menos de menor trascendencia que los propios puntos a etiquetar. Por este motivo permitiremos que las etiquetas corten a la línea de entrada (véase *Figura 6.3*). En caso contrario sólo serían posibles dos posiciones por etiqueta (véase *Figura 6.4*), por lo que todos los problemas de decisión podrían resolverse en tiempo lineal [14].

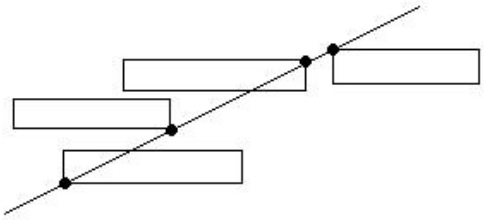


Figura 6.3.: Las etiquetas pueden cortar a la línea

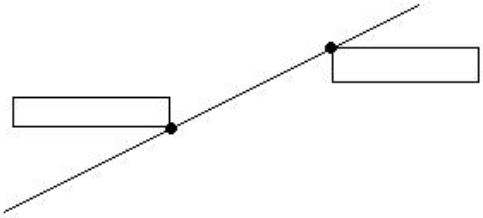


Figura 6.4.: No se permite que las etiquetas corten a la línea

Para abordar estos problemas utilizaremos de nuevo el algoritmo incremental descrito en el *Capítulo 4*. Para ello necesitaremos que los puntos estén previamente ordenados, según los valores crecientes de sus abscisas: sean $P_1(x_1,y_1)$ y $P_2(x_2,y_2)$, entonces $x_1 \leq x_2$.

Continuaremos utilizando los conceptos de *k*-realización y sombra de una *k*-realización. Sin embargo, no haremos uso del concepto de dualidad. Por lo tanto dos realizaciones R_k y R'_k diremos que son *equivalentes*, y escribiremos $R_k \sim R'_k$ si tienen la misma sombra, $s(R_k) = s(R'_k)$.

Una vez clasificadas las *k*-realizaciones según su sombra, introducimos la relación de orden necesaria para poder disponer del concepto de minimalidad. Así, dadas dos *k*-realizaciones R_k y R'_k. Se dice que $R_k \leq R'_k$ si $s(R_k) \subseteq s(R'_k)$. Diremos que una *k*-realización R_k es una *k*-realización mínima si para cualquier otra *k*-realización R'_k se tiene $R_k \leq R'_k$.

Una $(k+1)$-realización R_{k+1} es *hija* de una *k*-realización R_k si los elementos de R_k son los *k* primeros elementos de R_{k+1}. Por lo tanto una $(k+1)$-realización hija se obtiene añadiendo a una *k*-realización la etiqueta correspondiente al punto P_{k+1}.

El *Lema 5.1.2* afirmaba que, para el problema H–4P, si existe una $(k+1)$-realización, entonces existe otra donde sus *k* primeros elementos constituyen una *k*-realización mínima. Podemos asegurar por lo tanto que manteniendo control sobre las *k*-realizaciones mínimas podremos abordar estos problemas con el mismo procedimiento incremental utilizado anteriormente, consistente en cada paso en obtener a partir de

las distintas *k*-realizaciones mínimas sus correspondientes (*k+1*)-realizaciones mínimas hijas.

Pasamos a estudiar los distintos problemas concretos. Comenzamos con el caso de etiquetas cuadradas iguales y fijas, es decir una vez colocada la etiqueta, el punto debe quedar en alguno de los cuatro vértices del cuadrado. Por comodidad, y sin pérdida de generalidad podemos considerarlos cuadrados unitarios.

Resolveremos los siguientes problemas sobre línea oblicua:

* ETIQUETAS CUADRADAS, IGUALES Y FIJAS SOBRE UNA LÍNEA OBLICUA (*OC–4P(P,L)*)
  - ➔ El punto puede situarse sobre cualquiera de los vértices de la etiqueta, tratándose todas ellas de cuadrados de tamaño unidad.
  - ➔ Véase la "*SECCIÓN 6.1*"

* ETIQUETAS RECTANGULARES, IGUALES Y FIJAS SOBRE UNA LÍNEA OBLICUA (*OR(1)-4P(P,L)*)
  - ➔ El punto puede situarse sobre cualquiera de los vértices de la etiqueta, tratándose todas ellas de rectángulos de dimensiones iguales.
  - ➔ Véase la "*SECCIÓN 6.2*"

* ETIQUETAS DE *m* TIPOS RECTANGULARES Y FIJAS SOBRE UNA LÍNEA OBLICUA (*OR(m)-4P(P,L)*)
  - ➔ El punto puede situarse sobre cualquiera de los vértices de la etiqueta, siendo variables sus dimensiones.
  - ➔ Véase la "*SECCIÓN 6.3*"

* ETIQUETAS RECTANGULARES DE ALTURA 1 SOBRE UNA LÍNEA OBLICUA (*O1R-4P(P,L)*)
  - ➔ El punto puede situarse sobre cualquiera de los vértices de la etiqueta, tratándose todas ellas de rectángulos de de altura unidad y anchura la indicada.
  - ➔ Véase la "*SECCIÓN 6.4*"

* ETIQUETAS CUADRADAS, IGUALES Y DESLIZANTES SOBRE UNA LÍNEA OBLICUA (*OC–4S(P,L)*)

- ➔ El punto puede situarse sobre el contorno de la etiqueta, siendo éstas cuadradas y de tamaños iguales.
- ➔ Véase la "*SECCIÓN 6.5*"

SECCIÓN 6.1.

ETIQUETAS CUADRADAS, IGUALES Y FIJAS SOBRE UNA LÍNEA OBLICUA

(*OC–4P(P,L)*)

ENTRADA: Un conjunto $P=\{P_1, \ldots, P_n\}$ de puntos situados sobre una línea oblicua.

PREGUNTA: ¿Puede colocarse en cada uno de los puntos un cuadrado de lado la unidad, de manera que los puntos queden en alguno de los vértices del cuadrado, y no se produzcan intersecciones entre ellos?

Respecto a las posiciones de etiquetado, sus distintas posibilidades, y siguiendo la misma notación que utilizáramos para el caso de las líneas horizontales, representada en la *Figura 6.1.1*, las etiquetas pueden ser colocadas en cuatro posiciones. Si denotamos por L^t a la etiqueta colocada en la posición t, $t = 1, 2, 3, 4$, es claro que las relaciones entre las sombras de dicha etiqueta son $s(L^3) \leq s(L^2)$, $s(L^3) \leq s(L^4)$, $s(L^3) \leq s(L^1)$, $s(L^2) \leq s(L^1)$, $s(L^4) \leq s(L^1)$; siendo no comparables entre ellas las sombras $s(L^2)$ y $s(L^4)$.

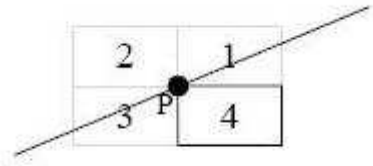


Figura 6.1.1.: Posiciones de etiquetas fijas sobre línea oblicua

Apoyándonos en la imagen anterior, y para el desarrollo del algoritmo, se tomará como posición preferente la posición *3*, seguida de la *2*, la *4* y la *1*. De este modo no será necesario comparar las posibles sombras resultantes de la colocación de una etiqueta sobre un punto estudiando todas sus posibilidades. A efectos prácticos, si una etiqueta

pudiese ser colocada en cualquier posición, se colocaría directamente en la *3*. Si por cualquier motivo no pudiese ser colocada en la *3*, se pasaría a la posición *2*. En el caso de que tampoco pudiese ser colocada en la *2*, se colocaría en la posición *4*, quedando la posición *1* como el último recurso (véase *Figura 6.1.1*).

Una vez aclarado el orden de preferencia para posicionar las etiquetas, resta mencionar que para nuestra aplicación se utilizará la misma notación que se usó en problemas anteriores, a saber:

Posición 1 ~ "TR"
Posición 2 ~ "TL"
Posición 3 ~ "BL"
Posición 4 ~ "BR"

Lema 6.1.1. *Una k-realización, para el problema OC–4P, tiene a lo sumo dos (k+1)-realizaciones hijas que son mínimas.*

Demostración. Sea R_k una k-realización y la etiqueta cuadrada L_{k+1} correspondiente al punto P_{k+1}. Si el punto P_{k+1} ϵ $s(R_k)$ no tendría ninguna hija. Por el contrario, y teniendo en cuenta la relación existente entre las sombras de las diferentes posiciones de una etiqueta, si la etiqueta L_{k+1} puede añadirse en la posición 3 habrá una única (k+1)-realización mínima. En caso contrario podría ocurrir que se pudiera colocar en las posiciones 2 y 4, dando lugar a dos hijas mínimas y si sólo se pudiera colocar en la posición 1, únicamente daría lugar a una (k+1)-realización mínima hija (véase Figura 6.1.2).

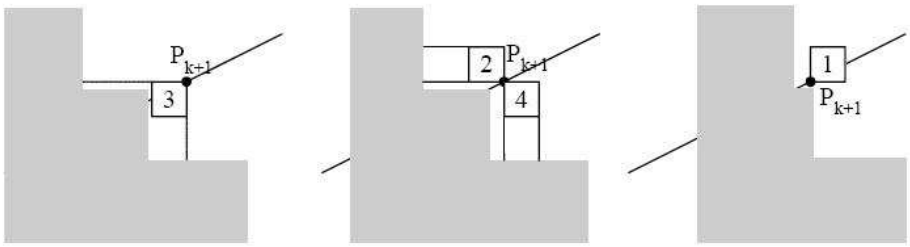


Figura 6.1.2.: Hijos de una k-realización

Veremos a continuación la influencia en la sombra de una *k*-realización mínima la colocación de una nueva etiqueta. De forma intuitiva diremos que el siguiente lema pondrá de manifiesto que en el

proceso incremental, al colocar una nueva etiqueta, bastará considerar su incidencia en las dos últimas etiquetas del paso anterior.

Será suficiente probar que la sombra de cualquier realización estará determinada a lo sumo por las sombras de sus dos últimas etiquetas. Para ello consideraremos las cuatro situaciones más desfavorables según las posiciones de las dos últimas etiquetas:

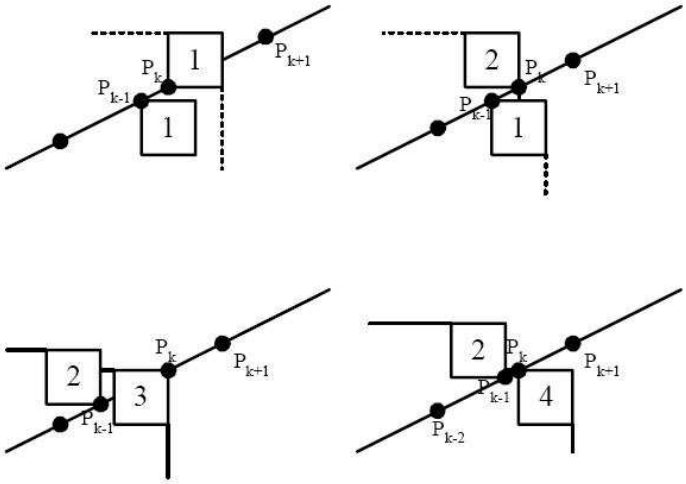


Figura 6.1.3.: Perfil de la sombra de una k-realización

Como se aprecia en la *Figura 6.1.3*, si la etiqueta L_k está en la posición *1*, entonces en el perfil de la sombra sólo aparece la última etiqueta. En cualquiera de las otras tres posiciones aparecen a lo sumo las dos últimas.

Esta afirmación nos garantiza que la sombra de una *k*-realización queda determinada por las de las dos últimas etiquetas, por lo tanto en lo sucesivo cuando se hable de la sombra de una *k*-realización nos referiremos únicamente a la producida por las dos últimas etiquetas, ya que el resto no tiene relevancia para el desarrollo del problema.

Estamos ahora en condiciones de probar que podemos controlar el número de sombras correspondientes a realizaciones mínimas. Esto lo hacemos en el siguiente lema:

Lema 6.1.2. *En el problema OC-4P existen a lo sumo dos sombras distintas correspondientes a k-realizaciones mínimas, para k = 1, …, n.*

También podemos afirmar que dadas las sombras correspondientes a las (*k-1*)-realizaciones mínimas se pueden obtener las sombras correspondientes a *k*-realizaciones mínimas en tiempo constante, ya que es constante el número total de comprobaciones a realizar. Concretamente un número no superior a 4^2, ya que tendremos que comprobar las cuatro posibles posiciones de las dos últimas etiquetas.

Como resultado de este conjunto de conclusiones, es correcto representar gráficamente las siguientes sombras mínimas:

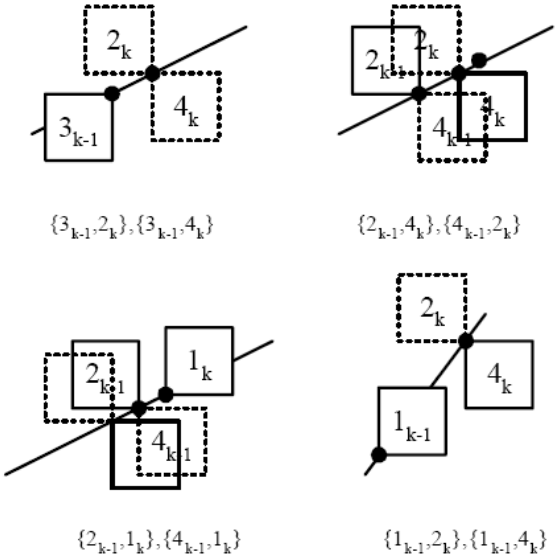


Figura 6.1.4.: Sombras mínimas para el problema OC-4P

En el apartado 6.1.1. se detalla el proceso de etiquetado para el problema OC-4P.

* Si el usuario lo desea puede introducir los datos a través de un fichero de texto, cuyo formato será idéntico al que se estableció para el etiquetado sobre líneas horizontales.

Por ejemplo, para indicar que se pretenden etiquetar 3 puntos (6, 7 y 9) con etiquetas cuadradas de tamaño fijo, el fichero de texto será:

3 6 7 9 2 5 8

Siendo el propio programa el encargado de asignar el tamaño (fijo) de las etiquetas.

APARTADO 6.1.1.
IMPLEMENTACIÓN DEL PROBLEMA (*OC–4P(P,L)*)

El método utilizado para resolver el problema OC-4P es el "`main`" de la clase "`Main_OC_4P`". Éste método se encarga de interactuar con el usuario solicitándole información tal como el número de puntos que se desean etiquetar y la posición de dichos puntos. Se permite la entrada de éstos datos a través del teclado o mediante un fichero de texto que, previamente, el usuario ha debido editar. Además se ofrece información general sobre el problema OC-4P así como el formato del fichero a incluir, en su caso.

Si el usuario prefiere introducir los datos manualmente a través del teclado, primero se le solicitará el número total de puntos que pretende etiquetar sobre la línea oblicua (por comodidad y claridad se tomará una línea recta con una pendiente positiva de 45°). A continuación se irán solicitando uno a uno los números enteros que representarán los puntos sobre la línea. Resulta de interés aclarar que, aunque en la tesis original se propone tomar un tamaño fijo para las etiquetas, aquí se ha optado por permitir al usuario establecer dicho tamaño.

Tanto en el caso en el que el usuario introduce los datos manualmente como en el que se usa un fichero de texto para facilitar los mismos, la lista de etiquetas la generará automáticamente el programa a partir del tamaño que indique el usuario.

En el momento en el que el programa disponga de los datos necesarios para resolver el problema, se llevará a cabo una primera validación de dichos datos: el número de elementos indicados debe ser igual a la totalidad de los puntos que se han añadido.

Posteriormente, y si la validación ha sido correcta, se crea un objeto "`solución`" de tipo "`Solucion_OC_4P`" utilizando las listas de puntos y etiquetas obtenidas y se invoca al método "`obtenerSolucion()`" de la propia clase "`Solucion_OC_4P`", que contiene los algoritmos principales que resolverán este problema.

Una vez dentro de este método, se realiza la segunda validación de datos: mediante una llamada al método "`valoresValidos()`" de la clase "`Utilidades`" se comprueba si los puntos y etiquetas son

valores mayores que cero. Si los valores son correctos, estamos en disposición de resolver el problema; si no habría que ejecutar de nuevo el programa corrigiendo los valores de puntos y/o etiquetas.

Cuando se han validado los valores de puntos y etiquetas se comprueba si la lista de puntos está ordenada ascendentemente, ya que en caso contrario tendríamos la necesidad de ordenarla para que el tiempo de ejecución del problema sea aceptable.

Finalizadas las comprobaciones de datos, llega el momento de la resolución del problema, cuyo algoritmo se basa en lo siguiente:

- Si es posible el etiquetado del primer punto en la posición “BL”, se etiqueta automáticamente, ya que la sombra resultante será la menor.

- Si no se ha podido etiquetar en “BL” se probará para la posición “TL”, considerada la segunda posición más eficiente, junto con “BR”, que será nuestra tercera opción.

- Como última opción de etiquetado estará la posición “TR”, cuya sombra resultante es mayor que las mencionadas anteriormente.

Se repetirán estos pasos para la totalidad de puntos y etiquetas y, en el momento en que sea imposible el etiquetado de un solo punto se detendrá la ejecución y el programa devolverá un texto indicativo de que no se pudo etiquetar la lista completa de puntos introducida.

Los métodos utilizados son:

“`probarEtiquetaBL`”, “`probarEtiquetaTL`”, “`probarEtiquetaBR`”, “`probarEtiquetaTR`”, “`aplicarEtiquetaBL`”, “`aplicarEtiquetaTL`”, “`aplicarEtiquetaBR`” y “`aplicarEtiquetaBL`”.

Una vez ejecutado el algoritmo anterior, se procede a representar la solución encontrada, si la hubiese. Para ello el programa generará una línea del tipo:

```
“Se han obtenido las siguientes posiciones:
     ( BL , TL , BL , TR , BR )”
```

En la que se indica que la primera etiqueta ha sido establecida en la posición “BL”, la segunda en la “TL”, la tercera en la “BL”, la cuarta en la “TR” y la quinta en la posición “BR”.

Como ayuda gráfica a la solución anterior se ha desarrollado además un método (“`representarEspacio()`”) que representa en la consola el espacio de etiquetado. Los datos mostrados por este método son orientativos, siendo la representación anterior la única salida exacta y válida para los puntos y etiquetas introducidos. A continuación se describe de forma general el método que representa gráficamente la salida:

Utilizando el ejemplo de la *Figura 6.1.1.1.*, supongamos que se corresponde con la solución de un problema de etiquetado OC-4P. En ella se puede comprobar que se han etiquetado los puntos *2*, *4* y *6* con etiquetas de tamaño *6*, *4* y *2*, siendo sus posiciones relativas las siguientes: *BR*, *TL* y *BR*, respectivamente:

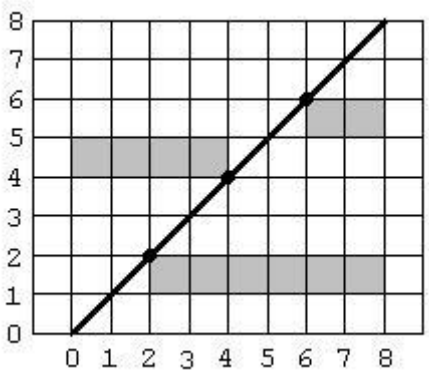


Figura 6.1.1.1.: Ejemplo de etiquetado sobre línea oblicua

Como apoyo gráfico a la solución que obtiene la aplicación, la consola lo representaría de la siguiente manera:

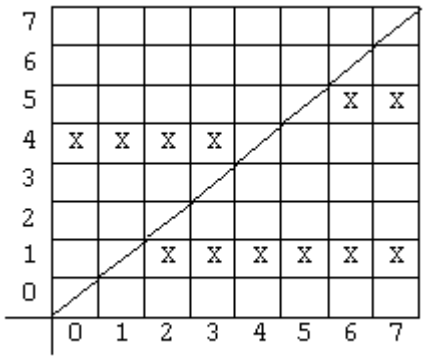


Figura 6.1.1.2: Representación de la solución por consola

Como aclaración, mencionar que el punto (*x,y*) está representado por el vértice inferior izquierdo de la celda (*x,y*).

Recuerde que la representación gráfica que se puede ver en la consola no se puede tomar como una solución exacta del problema, sólo se trata de una ayuda visual para el usuario.

* Clases y métodos utilizados para la resolución de este apartado:

-CLASE-	-MÉTODO-
Main_OC_4P	main
Solucion_OC_4P	obtenerSolucion
	inicializaValores
	probarEtiquetaTL
	probarEtiquetaBL
	probarEtiquetaTR
	probarEtiquetaBR
	aplicarEtiquetaTL
	aplicarEtiquetaBL
	aplicarEtiquetaTR
	aplicarEtiquetaBR
Utilidades	valoresValidos
	puntosOrdenados
	ordenarListasInteger
	imprimeListaString
Constantes	N/A
UtilidadesOblicua	representarEspacio
ConjuntoListas	getListaPuntos
	getListaEtiquetas

SECCIÓN 6.2.
ETIQUETAS RECTANGULARES, IGUALES Y FIJAS SOBRE UNA LÍNEA OBLICUA

(*OR(1)–4P(P,L)*)

ENTRADA: Un conjunto de puntos *P* situados sobre una línea oblicua y un rectángulo *R* de lados paralelos a los ejes coordenados.

PREGUNTA: ¿Puede colocarse un rectángulo *R* en cada uno de los puntos, de manera que estos puntos queden en alguno de los vértices del rectángulo y no se produzcan intersecciones entre ellos?

Podemos convertir, como se aprecia en la *Figura 6.2.1*, cualquier entrada de este problema en una entrada del problema *OC–4P* mediante una transformación afín del plano (estiramiento), resolverlo y, según se muestra en la *Figura 6.2.2*, transformar su solución mediante la transformación afín inversa en una solución de nuestro problema *OR(1)–4P*, por lo que los problemas son equivalentes.

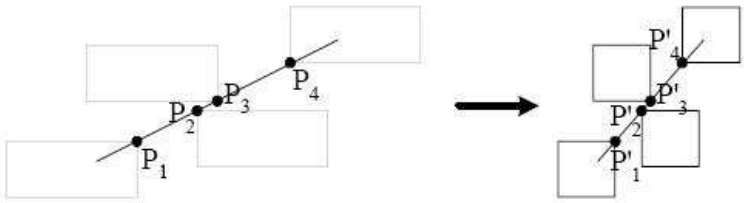


Figura 6.2.1.: Transformación afín del Problema OR(1)-4P al OC-4P.

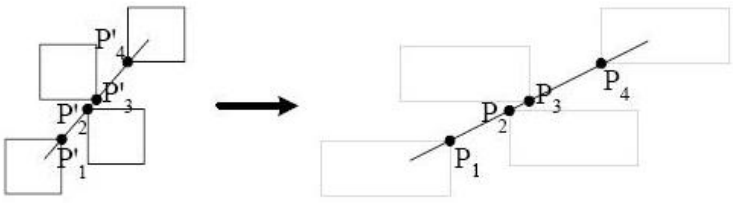


Figura 6.2.2.: Transformación afín del Problema OC-4P al OR(1)-4P.

APARTADO 6.2.1.

IMPLEMENTACIÓN DEL PROBLEMA (*OR(1)–4P(P,L)*)

El problema que nos ocupa se resolverá utilizando, principalmente, las clases "`Main_OR1_4P`" y "`Solucion_OR1_4P`", ambas contenidas en el paquete "`lineaOblicua.or1_4p`".

Como pasa en los problemas anteriores, los datos se solicitarán a través del método "`main`" de la clase "`Main_OR1_4P`", disponiendo el usuario de las dos opciones habituales para introducir dichos datos: a través del teclado o por medio de un fichero de texto.

Si se opta por la primera opción, se le solicitará en primer lugar el número total de puntos que desea etiquetar. Seguidamente deberá introducir cada uno de los puntos, seguidos de un valor entero que represente la altura de las etiquetas y otro que represente su anchura. Nótese que todas las etiquetas serán iguales.

Si, por el contrario, el usuario prefiere introducir los valores a través de un fichero de texto, deberá seguir las indicaciones que se mostrarán en la consola, donde se indicará el formato que debe tener el fichero.

Introducidos los valores por uno u otro método, se comprobará que el número de puntos a etiquetar sea positivo, finalizando automáticamente el programa en caso contrario. A continuación se comprueba la coherencia de los datos introducidos: el número de puntos deberá ser igual a la lista de ellos que se introduzca.

Completado con éxito el paso anterior, el siguiente consiste en crear una instancia de la clase "`Solucion_OR1_4P`", para lo cual se utiliza la lista de puntos ("`puntos`"), la altura de las etiquetas ("`alturaEtiqueta`") y la anchura de las mismas ("`anchuraEtiqueta`"). El objeto creado se utilizará para invocar al método "`obtenerSolucion`" de la misma clase, que nos deberá facilitar la solución al problema, si la hubiera.

Una vez dentro de la clase "`Solucion_OR1_4P`", la resolución del problema es muy similar al problema anterior *OC-4P*.

En primer lugar se comprueba si los puntos y tamaños de etiquetas son valores positivos para, a continuación, verificar que la lista de puntos se encuentra ordenada, y proceder a su ordenación automáticamente si fuese necesario.

En este momento la lista de puntos debe componerse de valores positivos ordenados ascendentemente y, tanto la altura como la anchura de las etiquetas, serán igualmente valores positivos.

Llegamos entonces al momento clave en la resolución del problema: un bucle que recorre la lista de puntos comprobando, para cada uno de ellos, la mejor opción de etiquetado. Se añade la etiqueta siempre en el vértice que menor sombra genere. Concretamente se tratará de etiquetar de la siguiente manera:

I) Se comprueba si es posible etiquetar en la posición "BL" mediante el método "`probarEtiquetaBL`", y si es posible se etiqueta utilizando el método complementario "`aplicarEtiquetaBL`".

II) Si la comprobación anterior resultó fallida, se intenta etiquetar en la posición "TL" con el método "`probarEtiquetaTL`". Si esta vez sí es posible etiquetar, se hace mediante una llamada a "`aplicarEtiquetaTL`".

III) Si tampoco fue posible etiquetar en "TL", se probará con "BR". Los métodos utilizados para la comprobación y el etiquetado, en su caso, son "`probarEtiquetaBR`" y "`aplicarEtiquetaBR`".

IV) La última opción será la posición "TR", que es la que mayor sombra generará. Se comprueba con "`aplicarEtiquetaTR`" y se etiqueta con "`aplicarEtiquetaTR`".

La aplicación devuelve la solución completa o parcial según se haya podido etiquetar o no, respectivamente. Ésta solución será de la forma:

```
"Se han obtenido las siguientes posiciones:
( BL , TL , BL , TR , BR )"
```

Además de la solución textual, y como en los problemas anteriores, el método "`representarEspacio()`" genera una solución gráfica del problema, sirviendo sólo de ayuda al usuario.

* Clases y métodos utilizados para la resolución de este apartado:

-CLASE-	-MÉTODO-
Main_OR1_4P	main
Solucion_OR1_4P	obtenerSolucion
	inicializaValores
	probarEtiquetaTL
	probarEtiquetaBL
	probarEtiquetaTR
	probarEtiquetaBR
	aplicarEtiquetaTL
	aplicarEtiquetaBL
	aplicarEtiquetaTR
	aplicarEtiquetaBR
Utilidades	valoresValidos
	puntosOrdenados
	ordenarListaInteger
	imprimeListaString
	maxValorLista
Constantes	N/A
UtilidadesOblicua	representarEspacio
ListaEnteros	getLista

SECCIÓN 6.3.

ETIQUETAS DE *m* TIPOS RECTANGULARES Y FIJAS SOBRE UNA LÍNEA OBLICUA

(*OR(m)–4P(P,L)*)

ENTRADA: Un conjunto de puntos $P = \{P_1, \ldots, P_n\}$ situados sobre una línea oblicua y un conjunto $L = \{R_1, \ldots, R_n\}$ de *n* rectángulos, de lados paralelos a los ejes coordenados y de *m* tipos diferentes.

PREGUNTA: ¿Puede colocarse el rectángulo R_i en cada punto *Pi*, de manera que el punto quede en alguno de los vértices del rectángulo, y no se produzcan intersecciones entre ellos?

Este problema trata de etiquetar un conjunto de *N* puntos situados sobre una línea oblicua de pendiente positiva de 45°, utilizando un conjunto de *N* etiquetas rectangulares de dimensiones diferentes. El punto debe situarse sobre alguno de los cuatro vértices de la etiqueta que le corresponda.

Podemos apreciar la similitud de este problema con el problema anterior, con la diferencia de que tendremos que controlar en el proceso incremental las dos últimas etiquetas de cada tipo.

Como en las sombras influirán a lo sumo las dos últimas etiquetas de cada tipo de rectángulo, cada sombra vendrá determinada por un número no superior a 2^m etiquetas, lo que explica el siguiente lema:

Lema 3.14. *En el problema OR(m)–4P existen a lo sumo 2^m sombras distintas correspondientes a k-realizaciones mínimas, para k = 1, …, n.*

APARTADO 6.3.1.

IMPLEMENTACIÓN DEL PROBLEMA (*OR(m)–4P(P,L)*)

Por medio de la clase principal "`Main_ORM_4P`", en primer lugar se solicita al usuario que seleccione la manera en la que desea introducir los datos. Al igual que en los problemas anteriores, se pone a disposición suya dos modalidades: puede introducir los valores a través del teclado cuando se solicite en la consola, o bien puede editar un fichero de texto con el formato que se le indique en la ayuda de la aplicación.

El método "`main`" se asegurará, una vez introducidos los datos, que el número de puntos a etiquetar coincida con el número de etiquetas, y que ambos sean mayores que cero. Comprobado este aspecto se creará un objeto del tipo "`Solucion_ORM_4P`" utilizando la lista de puntos ("`puntos`"), de alturas de etiquetas ("`alturas`") y de anchuras de etiquetas ("`anchuras`") para invocar al método "`obtenerSolucion`" de la clase "`Solucion_ORM_4P`" y resolver el problema con los valores dados.

Al igual que para el resto de problemas, se comprueba en primer lugar que los valores de los puntos y los tamaños de las etiquetas sean correctos mediante el método "`valoresValidos`" de la clase "`Utilidades`". En nuestro caso serán correctos si son números enteros positivos. Acto seguido se analiza si la lista de puntos se encuentra ordenada ascendentemente y, si no estuviese ordenada, se ordena automáticamente, ya que en caso contrario se dispararía el tiempo de ejecución del programa. Nótese que, si hubiese que ordenar la lista de puntos, sería necesario también ordenar las listas de alturas y anchuras de las etiquetas, ya que el elemento N de una de ellas se corresponde siempre con el elemento N de las otras dos.

Una vez disponemos de la totalidad de los datos y sabemos que son válidos y se encuentran correctamente ordenados, procederemos a resolver el problema: mediante los método "`probarEtiquetaXX`" y "`aplicarEtiquetaXX`" se comprueba si es posible etiquetar un punto en la posición "XX" y, en su caso, se etiqueta.

El texto “XX” indica la posición en la que se quiere probar o aplicar la etiqueta sobre un determinado punto, pudiendo tratarse de los valores que se indican a continuación:

I) “BL” → Abajo a la Izquierda
II) “TL” → Arriba a la Izquierda
III) “BR” → Abajo a la Derecha
IV) “TR” → Arriba a la Derecha

Si, dadas las posiciones de los puntos y las dimensiones de las etiquetas, fuese posible etiquetar, la aplicación devolverá el resultado de la siguiente manera:

```
“Se han obtenido las siguientes posiciones:
( BL , TR , BR , BL , BR )”
```

Lo cual indica que se la posición en la que se emplaza la primera etiqueta es “Bottom Left” (Abajo a la Izquierda), quedando el primer punto sobre el vértice superior derecho de dicha etiqueta. La segunda etiqueta estaría situada en la posición “Top Right” (Arriba a la derecha) respecto del segundo punto, y así sucesivamente.

Además de la solución textual se ha desarrollado un método “`representarEspacio`” que proporciona una ayuda visual al usuario, dibujando cómo han quedado etiquetados los puntos introducidos.

* Clases y métodos utilizados para la resolución de este apartado:

-CLASE-	-MÉTODO-
Main_ORM_4P	main
Solucion_ORM_4P	obtenerSolucion
	inicializaValores
	probarEtiquetaTL
	probarEtiquetaBL
	probarEtiquetaTR
	probarEtiquetaBR
	aplicarEtiquetaTL
	aplicarEtiquetaBL
	aplicarEtiquetaTR
	aplicarEtiquetaBR
Utilidades	valoresValidos
	puntosOrdenados
	ordenarTresListasInteger
	imprimeListaString
	maxValorLista
Constantes	N/A
UtilidadesOblicua	representarEspacio
ListaEnteros	getListaPuntos
	getAlturas
	getAnchuras

SECCIÓN 6.4.

ETIQUETAS RECTANGULARES DE ALTURA 1 SOBRE UNA LÍNEA OBLICUA

(*O1R–4P(P,L)*)

ENTRADA: Un conjunto de puntos $P = \{P_1, \ldots, P_n\}$ situados sobre una línea oblicua y un conjunto de números reales $W = \{w_1, \ldots, w_n\}$.

PREGUNTA: ¿Puede colocarse un rectángulo de dimensiones $w_i \times 1$ en cada punto P_i, de manera que estos puntos queden en alguno de los vértices del rectángulo, y no se produzcan intersecciones entre ellos?

En este caso, los puntos siguen estando situados sobre una línea oblicua de pendiente positiva de 45°, pero las etiquetas serán de altura fija, pudiendo ser variables sus anchuras. Como en el caso anterior, los puntos deberán situarse sobre uno de los cuatro vértices de la etiqueta.

Cuando se trata de etiquetar mapas, líneas de metro, etc., las etiquetas normalmente tienen una misma altura, dada por el tipo de letra a utilizar, y diferentes anchuras, dadas por la extensión del texto a colocar en ellas. Por esta razón es de interés el problema que nos ocupa.

Siguiendo con el esquema señalado en problemas anteriores, trataremos de llevar el control del número de realizaciones mínimas en cada paso. En lo sucesivo cuando consideremos la sombra de una *k*-realización nos referiremos únicamente a la parte de dicha sombra que es susceptible de ser cortada por la etiqueta del punto P_{k+1}, ya que el resto no influye en el desarrollo del problema.

El siguiente resultado es de demostración evidente:

Lema 6.4.1. *En el problema O1R–4P, la etiqueta L_k correspondiente al punto P_k debe estar en la sombra de cualquier k-realización R_k.*

Llamamos *escalón* de la sombra de una *k*-realización a todo vértice superior derecho de cualquier etiqueta de la frontera de dicha sombra.

Lema 6.4.2. *En el problema O1R–4P, las sombras de k-realizaciones mínimas tienen a lo sumo dos escalones, para k = 1, …, n.*

APARTADO 6.4.1.
IMPLEMENTACIÓN DEL PROBLEMA (*O1R–4P(P,L)*)

El problema denominado O1R-4P trata de etiquetar, sobre una línea oblicua de pendiente positiva de 45°, una serie de puntos con sus respectivas etiquetas. Estos datos los facilitará el usuario en el momento en el que se le solicite, pudiendo hacerlo a través del teclado (manualmente) o editando un fichero de texto según el formato que se especifica en la ayuda del programa.

Para este apartado las etiquetas son rectangulares, de anchura cualquiera y altura fija 1, debiendo quedar el punto en uno de sus cuatro vértices.

Si, como se describe en la introducción de este apartado, se tomara la altura de las etiquetas con valor unidad, la resolución del problema sería bastante simple, ya que, al estar divididos los ejes de coordenadas en segmentos de esa misma longitud, todas las etiquetas se situarían en la posición "BL", y sólo habría que determinar la posición de etiquetado cuando se coloquen varias etiquetas en un mismo punto Si se observa la *Figura 4.4.1.1*, se aprecia que, siendo la lista de puntos (1, 2, 3, 4, 5 y 6), y la anchura de las etiquetas fijas a 3, todas las etiquetas se emplazan en la posición ABAJO-IZQUIERDA (BL). Es por ello que, por defecto, se tomarán las etiquetas de altura 3, evitándose la resolución trivial del problema en la mayoría de los casos.

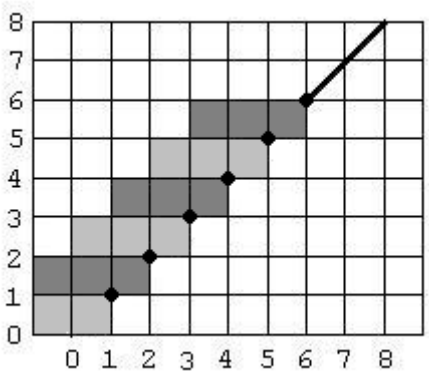


Figura 6.4.1.1.: Etiquetas de altura unidad

Debido a la dimensión horizontal variable de las etiquetas, será necesario indicar su valor para cada una de ellas. En total se solicitará al usuario los siguientes datos:

- Número total de puntos a etiquetar.

- Posición de cada uno de los puntos.
- Anchura de las etiquetas que se utilizarán.

Aclarado este punto, el proceso de resolución del problema OR1-4P es bastante similar al de problemas anteriores.

Mediante el método "`main`" de la clase "`Solucion_O1R_4P`" se solicita al usuario que, bien por teclado, bien mediante un fichero de texto, introduzca los valores requeridos por el programa y que se detallan en el párrafo anterior.

Como siempre, lo primero es verificar que el número de elementos indicados por el usuario coincide con el número total de puntos y etiquetas introducidos.

Una vez realizada esta comprobación previa, se crea un objeto "`solucion`" de tipo "`Solucion_O1R_4P`" mediante la lista de puntos y la de etiquetas. A continuación se solicita la resolución del problema con la llamada al método "`obtenerSolucion()`" de la misma clase "`Solucion_O1R_4P`".

Una vez dentro de este método, se realiza la segunda validación de datos: mediante una llamada al método "`valoresValidos()`" de la clase "`Utilidades`" se comprueba si los puntos y etiquetas son valores mayores que cero. Si los valores son correctos, estamos en disposición de resolver el problema; si no habría que ejecutar de nuevo el programa corrigiendo los valores de puntos y/o etiquetas.

Cuando se han validado los valores de puntos y etiquetas se comprueba si la lista de puntos está ordenada ascendentemente, ya que en caso contrario tendríamos la necesidad de ordenarla para que el tiempo de ejecución del problema sea aceptable.

Finalizadas las comprobaciones de datos, llega el momento de la resolución del problema, cuyo algoritmo se basa en lo siguiente:

- Si es posible el etiquetado del primer punto en la posición "BL", se etiqueta automáticamente, ya que la sombra resultante será la menor.

- Si no se ha podido etiquetar en "BL" se probará para la posición "TL", considerada la segunda posición más eficiente, junto con "BR", que será nuestra tercera opción.

- Como última opción de etiquetado estará la posición "TR", cuya sombra resultante es mayor que las mencionadas anteriormente.

Se repetirán estos pasos para la totalidad de puntos y etiquetas y, en el momento en que sea imposible el etiquetado de un solo punto se detendrá la ejecución y el programa devolverá un texto indicativo de que no se pudo etiquetar la lista completa de puntos introducida.

Los métodos utilizados son:

"`probarEtiquetaBL`", "`probarEtiquetaTL`", "`probarEtiquetaBR`", "`probarEtiquetaTR`", "`aplicarEtiquetaBL`", "`aplicarEtiquetaTL`", "`aplicarEtiquetaBR`" y "`aplicarEtiquetaBL`".

Una vez ejecutado el algoritmo anterior, se procede a representar la solución encontrada, si la hubiese. Para ello el programa generará una línea del tipo:

"`Se han obtenido las siguientes posiciones: ( BL , TL , BL , TR , BR )`"

En la que se indica que la primera etiqueta ha sido establecida en la posición "BL", la segunda en la "TL", la tercera en la "BL", la cuarta en la "TR" y la quinta en la posición "BR".

Como ayuda gráfica a la solución anterior se ha desarrollado además un método ("`representarEspacio()`") que representa en la consola el espacio de etiquetado. Los datos mostrados por este método son orientativos, siendo la representación anterior la única salida exacta y válida para los puntos y etiquetas introducidos. A continuación se describe de forma general el método que representa gráficamente la salida:

Utilizando el ejemplo de la *Figura 6.4.1.2.*, supongamos que se corresponde con la solución de un problema de etiquetado OC-4P. En ella se puede comprobar que se han etiquetado los puntos *2*, *4* y *6* con etiquetas de tamaño *6*, *4* y *2*, siendo sus posiciones relativas las siguientes: *BR*, *TL* y *BR*, respectivamente:

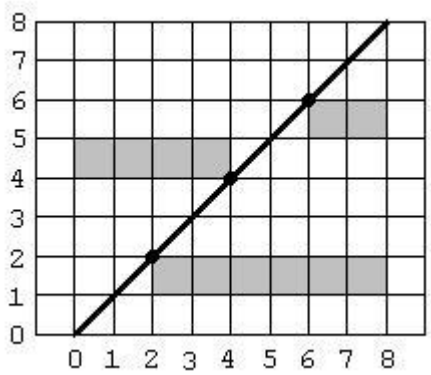


Figura 6.4.1.2.: Ejemplo de etiquetado sobre línea oblicua

Como apoyo gráfico a la solución que obtiene la aplicación, la consola lo representaría de la siguiente manera:

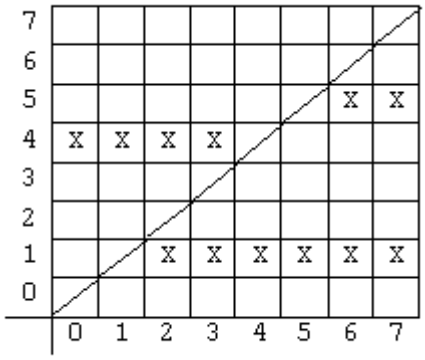


Figura 6.4.1.3: Representación de la solución por consola

Como aclaración, mencionar que el punto (*x,y*) está representado por el vértice inferior izquierdo de la celda (*x,y*).

Recuerde que la representación gráfica que se puede ver en la consola no se puede tomar como una solución exacta del problema, sólo se trata de una ayuda visual para el usuario.

* Clases y métodos utilizados para la resolución de este apartado:

-CLASE-	-MÉTODO-
Main_O1R_4P	main
Solucion_O1R_4P	obtenerSolucion
	inicializaValores
	probarEtiquetaTL
	probarEtiquetaBL
	probarEtiquetaTR
	probarEtiquetaBR
	aplicarEtiquetaTL
	aplicarEtiquetaBL
	aplicarEtiquetaTR
	aplicarEtiquetaBR
Utilidades	valoresValidos
	puntosOrdenados
	ordenarListasInteger
	imprimeListaString
Constantes	N/A
UtilidadesOblicua	representarEspacio
ConjuntoListas	getListaPuntos
	getListaEtiquetas

SECCIÓN 6.5.
ETIQUETAS CUADRADAS, IGUALES Y DESLIZANTES SOBRE UNA LÍNEA OBLICUA

(*OC–4S(P,L)*)

ENTRADA: Un conjunto de puntos *P* situados sobre una línea oblicua.

PREGUNTA: ¿Puede colocarse en cada uno de los puntos un cuadrado de lado la unidad, de manera que los puntos queden en alguno de los lados del cuadrado, y no se produzcan intersecciones entre ellos?

Cuando trabajamos con etiquetas deslizantes el número de posiciones posibles de cada etiqueta no es finito, aunque la posterior implementación nos permita obtener un número finito de posibles soluciones. Para identificar la posición de una etiqueta en una realización mínima la colocamos en la posición *1* y la deslizamos hasta conseguir su posición óptima (ver *Figura 6.1.1*). Para este deslizamiento tenemos dos posibilidades, comenzar con un deslizamiento horizontal hacia la izquierda, hasta llegar a su tope y si fuera posible continuar con un deslizamiento vertical hacia abajo, hasta llegar nuevamente a su tope, esta opción la denotaremos *IA* o realizar en primer lugar el deslizamiento vertical hacia abajo y después deslizar lo más a la izquierda posible, que denotaremos *AI*, como muestra la *Figura 6.5.1*.

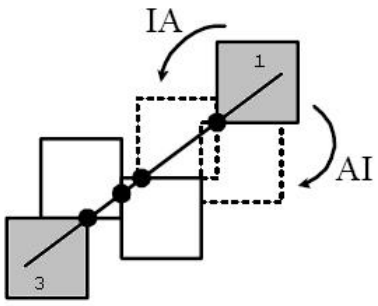


Figura 6.5.1.: Posiciones de una etiqueta en OC-4S.

Obsérvese que todas las sombras que se producen siguiendo un determinado camino (*IA* o *AI*) son comparables entre sí y cada vez menores, por lo que la mínima sombra se alcanza cuando la etiqueta alcanza el tope. Por otro lado las sombras de un camino no son comparables con las del otro camino, excepto en el caso de que por ambos caminos se llegue a la posición *3* en la etiqueta, caso en que coincidirán sus sombras.

Utilizaremos la notación $R_k = \{X_1, \ldots, X_n\}$ para indicar que en la *k*-realización mínima R_k la etiqueta L_i se ha colocado siguiendo el camino *X*, con $X \in \{IA, AI\}$.

Algunos de los resultados de problemas anteriores permanecen. Veamos que sigue siendo válido el siguiente Lema, que permite controlar el número de realizaciones mínimas hijas de una realización.

Lema 3.20. *En el problema OC–4S, cada k-realización tiene a lo sumo dos (k + 1)-realizaciones hijas que son mínimas.*

Se puede asegurar que la sombra de una *k*-realización vendrá determinada exclusivamente por sus dos últimas etiquetas.

Si expresamos el tipo de sombra de sus dos últimas etiquetas por los caminos que describe (*IA* o *AI*) una *k*-realización puede presentar los siguientes tipos de sombras: {*IA* , *IA*}, {*IA* , *AI*}, {*AI* , *IA*}, {*AI* , *AI*}. El siguiente resultado nos permitirá controlar el número de *k*-realizaciones mínimas.

Lema 3.22. *En el problema OC–4S, si dos k-realizaciones son del tipo IA - IA (o AI - AI) entonces son comparables, pudiéndose determinar la menor en tiempo constante.*

APARTADO 6.5.1.

IMPLEMENTACIÓN DEL PROBLEMA (*OC–4S(P,L)*)

Debido a que el problema que nos ocupa difiere del ya estudiado *OC-4P* sólo en la manera de etiquetar, el procedimiento de obtención de datos será idéntico en ambos.

Las diferencias comienzan cuando ya disponemos de los datos y se ha comprobado que son correctos.

Como novedad respecto al problema *OC-4P*, el *OC-4S* trata del etiquetado deslizante, es decir, el punto a etiquetar puede quedar situado sobre cualquier vértice o lado de la etiqueta. Así nos encontramos con que una solución válida al problema puede componerse de las siguientes posiciones relativas:

Posiciones relativas comunes para todos los problemas:

- "*BL*" → La etiqueta se sitúa Abajo y a la Izquierda del punto.
- "*TL*" → La etiqueta se sitúa Arriba y a la Izquierda del punto.
- "*BR*" → La etiqueta se sitúa Abajo y a la Derecha del punto.
- "*TR*" → La etiqueta se sitúa Arriba y a la Derecha del punto.

Posiciones relativas propias del etiquetado deslizante:

- "*RR*" → El punto se sitúa en el lado derecho de la etiqueta.
- "*LL*" → El punto se sitúa en el lado izquierdo de la etiqueta.
- "*TT*" → El punto se sitúa en el lado superior de la etiqueta.
- "*BB*" → El punto se sitúa en el lado inferior de la etiqueta.

Sea "XX" una determinada posición de etiquetado de las citadas anteriormente, el método "`etiquetadoXX()`" de la clase "`Solucion_OC_4S`" engloba la serie de procesos con los que se comprobará si se puede etiquetar en esa posición y, si es posible, añade la etiqueta y pasa al punto siguiente. Dentro de este método se llamará a las funciones "`probarEtiquetaXX()`", que decidirá si se puede etiquetar en la posición "XX", y "`aplicarEtiquetaXX()`", que, en caso de decisión positiva de la función anterior, procede al etiquetado del punto en la posición determinada.

Como ejemplos de etiquetado deslizante, se muestran las siguientes situaciones:

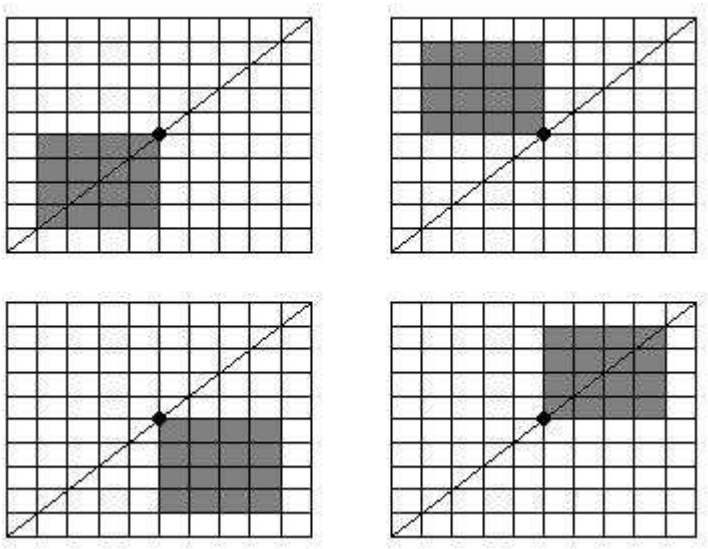


Figura 6.5.1.1.: Ejemplos de etiquetado sobre línea oblícua.
De arriba abajo y de izquierda a derecha: BL, TL, BR y TR.

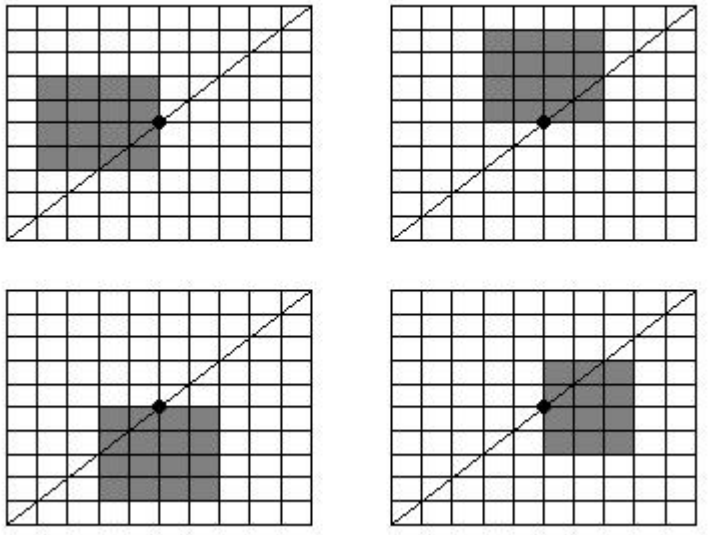


Figura 6.5.1.2.: Ejemplos de etiquetado deslizante sobre línea oblícua.
De arriba abajo y de izquierda a derecha: RR, BB, TT y LL.

Al igual que en problemas anteriores, el resultado se mostrará tanto en forma de texto como de manera gráfica, debiéndose considerar la primera representación como la única solución final.

* Clases y métodos utilizados para la resolución de este apartado:

-CLASE-	-MÉTODO-
Main_OC_4S	main
Solucion_OC_4S	obtenerSolucion
	inicializaValores
	probarEtiquetaTL
	probarEtiquetaBL
	probarEtiquetaTR
	probarEtiquetaBR
	ProbarEtiquetaBB
	ProbarEtiquetaTT
	ProbarEtiquetaRR
	probarEtiquetaLL
	aplicarEtiquetaTL
	aplicarEtiquetaBL
	aplicarEtiquetaTR
	aplicarEtiquetaBR
	AplicarEtiquetaBB
	AplicarEtiquetaTT
	AplicarEtiquetaRR
	aplicarEtiquetaLL
Utilidades	valoresValidos
	puntosOrdenados
	ordenarListaInteger
	imprimeListaString
	imprimeListaInteger
Constantes	N/A
UtilidadesOblicua	representarEspacio
ListaEnteros	getLista

Capítulo 7

Análisis temporal de las pruebas

Pruebas realizadas sobre un ordenador portátil

· *Packard Bell Easy Note MX*.
· Intel Core 2 CPU T5600 1.83GHz, 987 MHz, 2GB de RAM.
· Sistema operativo Windows XP Home Edition SP3.

Utilizando la siguiente versión de Eclipse

· Eclipse Europa. Versión: 3.3.0, Build Id: I20070625-1500.

El tiempo que reflejan las tablas siguientes representa la media aritmética de las cinco mediciones que se han realizado para cada caso, ya que la actividad del PC (y por lo tanto su tiempo de computación) es casi con total seguridad diferente en cada instante.

A continuación se muestran las tablas de resultados.

ETIQUETADO SOBRE LÍNEA HORIZONTAL

PROBLEMA	SOPORTE DATOS	N° PUNTOS	ALTURA ETIQUETA	ANCHURA ETIQUETA	TIEMPO (ms)
H-4P	TECLADO	1	1	1	15
		50	1	1	63
		100	1	1	203
	FICHERO	1	1	1	15
		50	1	1	63
		100	1	1	203
H-4S-1C	TECLADO	1	2	2	0
		50	2	2	15
		100	2	2	31
	FICHERO	1	2	2	0
		50	2	2	15
		100	2	2	31
H-4S	TECLADO	1	1	2	0
		50	1	2	16
		100	1	2	31
	FICHERO	1	1	2	0
		50	1	2	16
		100	1	2	31

[1] Se mide el tiempo transcurrido desde el momento en que se dispone de los datos (número de puntos, lista de puntos y lista de etiquetas) hasta que se obtiene la solución.

[2] Se toman puntos consecutivos: el punto 1, el rango de puntos del 1 al 50, o el rango de puntos del 1 al 100, según sea el caso.

[3] Lógicamente el tiempo empleado en resolver un problema es idéntico para el caso de la entrada de datos por teclado y por fichero ya que una vez se dispone de los valores se almacenan de la misma manera.

ETIQUETADO SOBRE LÍNEA OBLICUA

PROBLEMA	SOPORTE DATOS	N° PUNTOS	ALTURA ETIQUETA	ANCHURA ETIQUETA	TIEMPO (ms)
OC-4P	TECLADO	1	2	2	0
		50	2	2	1547
		100	2	2	9141
	FICHERO	1	2	2	0
		50	2	2	1547
		100	2	2	9141
OR(1)-4P	TECLADO	1	2	10	31
		50	2	10	1956
		100	2	10	10596
	FICHERO	1	2	10	31
		50	2	10	1956
		100	2	10	10596
OR(m)-4P	TECLADO	1	2	6	15
		50	2	6	1781
		100	2	6	12344
	FICHERO	1	2	6	15
		50	2	6	1781
		100	2	6	12344
O1R-4P	TECLADO	1	2	5	15
		50	2	5	1781
		100	2	5	10254
	FICHERO	1	2	5	15
		50	2	5	1781
		100	2	5	10254
OC-4S	TECLADO	1	2	2	15
		50	2	2	1485
		100	2	2	8741
	FICHERO	1	2	2	15
		50	2	2	1485
		100	2	2	8741

[1] Se mide el tiempo transcurrido desde el momento en que se dispone de los datos (número de puntos, lista de puntos y lista de etiquetas) hasta que se obtiene la solución.

[2] Se toman puntos consecutivos: el punto 1, el rango de puntos del 1 al 50, o el rango de puntos del 1 al 100, según sea el caso.

[3] Lógicamente el tiempo empleado en resolver un problema es idéntico para el caso de la entrada de datos por teclado y por fichero ya que una vez se dispone de los valores se almacenan de la misma manera.

Capítulo 8

Comparación con otras alternativas

Muchas son las opciones con las que cuenta un programador a la hora de seleccionar el lenguaje que utilizará para el desarrollo de una nueva aplicación, aunque debido al alto grado de desarrollo, uso intuitivo y flexibilidad que ha alcanzado el lenguaje Java, la mayoría de programadores contemporáneos se decantan por éste.

A continuación se muestra una comparación de las principales características de los lenguajes *C*, *C++* y *Java*, poniendo énfasis en sus mutuas ventajas y desventajas, así como en su aplicabilidad:

Expresividad

El lenguaje C siempre fue distinguido como altamente expresivo y potencialmente muy económico dada su reducida cantidad de palabras clave y el poder de algunos de sus operadores (por ejemplo, de los punteros.) En la actualidad, sin embargo, es frecuente el deseo de soportar estructuras de programación cada vez más complejas (aunque con frecuencia con los mismos algoritmos) con lo cual las implementaciones en lenguaje C tienden a tornarse oscuras (e inseguras) frente a equivalentes en otros lenguajes.

El lenguaje C++ proporciona un gran salto cualitativo frente a C al proporcionar nuevas características útiles en diversos contextos. Por ejemplo, la sobrecarga de operadores dota al lenguaje de una expresividad notable cuando se implementan aplicaciones científico-matemáticas (aunque en otros contextos pueden crear confusión); la sintaxis de clases y objetos permite manipular convenientemente diversas estructuras de datos y operaciones; las excepciones permiten procesar de un modo claro (aunque a veces con más código) los casos de error; los *templates* se pueden considerar (superficialmente) como macros de precompilador pero con muchas más características, etc. sin embargo, todo esto no ha estado exento de errores, en gran parte causados por mantener la compatibilidad con C tanto a nivel de sintaxis de lenguaje (compilación) como durante las etapas de enlace y ejecución.

En suma, el C++ es más expresivo que el C para la mayoría de aplicaciones medianas a grandes, lo cual es de esperarse desde que fue diseñado para abarcar una mayor cantidad de problemas mediante "múltiples paradigmas".

Por su parte, Java adopta una sintaxis muy similar a la del lenguaje C++, aunque eliminando algunas de sus características más oscuras. En particular, la eliminación de los punteros (arrastrados desde el lenguaje C) no lo ha hecho ni más ni menos expresivo, pero sí mucho más seguro.

Bien Definido

El lenguaje C fue considerado por mucho tiempo un buen ejemplo de un lenguaje consistente y sin ambigüedades notorias, especialmente entre sus contemporáneos.

Los creadores le reconocen ciertos inconvenientes en la notación que promueven a confusiones pero esto no es estrictamente un error y suele ser evitable.

Quizá el principal problema radica en la gran cantidad de aspectos que son dejados a criterio del implementador, entre los cuales destaca el tamaño de los tipos de datos.

Evidentemente esto puede crear serios problemas de portabilidad.

Estos inconvenientes lamentablemente fueron íntegramente heredados por C++ y hasta la fecha no tienen una clara solución (aunque el estándar C99 tiene algunas mejoras.)

El lenguaje Java, sin embargo, fue creado desde el inicio con la intención de desterrar las ambigüedades y dependencias del implementador del lenguaje y de sus clases auxiliares, con lo cual actualmente es tal vez el mejor definido de los lenguajes populares.

Tipos y estructuras de datos

El lenguaje C proporciona mecanismos que actualmente se consideran rudimentarios para proporcionar tipos de datos estructurados. Las estructuras (y uniones) se suelen utilizar para definir tipos complejos constituidos a partir de otros más simples (que a la vez pueden ser estructuras y/o uniones) con la posibilidad de crear identificadores auxiliares que simplifican la notación (*typedef*.) Asimismo, los arreglos o *arrays* permiten especificar colecciones homogeneas de longitud fija (en tiempo de compilación), los cuales tienen una relación muy cercana en su manipulación con los punteros. Una carencia notable (o ventaja

según algunos) es la falta de tipos de datos para representar cadenas de texto (*strings*), los cuales son soportados de un modo inusual mediante *arrays de caracteres*.

Si bien este "minimalismo" contribuye al desarrollo de la ejecución (o la optimización en la compilación), son muchos los casos donde se requiere el soporte de tipos más sofisticados (y sus operaciones asociadas) como por ejemplo, vectores, listas enlazadas, colas, etc. para los cuales el lenguaje obliga a construirlos desde sus componentes básicos. En la práctica, existen diversas librerías que complementan estos aspectos (por ejemplo, la popular *Glib*) pero su programación necesariamente es más laboriosa al no estar integrada internamente al lenguaje.

Por su parte, C++ proporciona facilidades que permiten la creación de estructuras de datos muy poderosas y fuertemente integradas en el lenguaje. Asimismo, el desarrollador puede crear sus propios tipos de dato con diversas operaciones asociadas. Gracias a esto, su uso resulta una extensión natural de los tipos de dato primitivos con lo cual se alcanza un alto grado de claridad.

Java proporciona tipos de datos primitivos similares (notablemente, careciendo de punteros) y mediante su librería de clases estándar proporciona todas las estructuras contenedoras "clásicas" antes mencionadas, aunque con una sintaxis que pone claramente de manifiesto que se trata de clases auxiliares.

Modularidad

En la referencia original [16) este criterio estaba referido a la posibilidad de desarrollar componentes de manera independiente los que eventualmente interactuarían.

Es ese sentido, los tres lenguajes analizados permiten desarrollar funciones, clases, y paquetes de modo independiente, cada cual con sus convenciones particulares.

En cuanto a los "niveles de empaquetamiento" de los componentes, el lenguaje C en la práctica proporciona sólo dos niveles: componentes visibles dentro del archivo de código fuente, y componentes visibles globalmente (concretamente, funciones y variables.) En C++ los conceptos de clase y "espacio de nombres" proporcionan dos niveles

adicionales de "empacado", mientras que en Java los equivalentes corresponden a las clases y los "paquetes".

Facilidades de entrada-salida

Siguiendo la referencia [16], este criterio está referido a las facilidades que proporcionan los lenguajes para acceder a archivos de disco, en particular el acceso secuencial, aleatorio e indexado. Asimismo, se hace referencia a la accesibilidad a sistemas de Base de Datos.

Manipulación de archivos

El lenguaje C como tal no proporciona instrucciones de I/O salvo mediante funciones de su "librería estándar", la cual fue diseñada en gran medida para aprovechar las facilidades del sistema de archivos Unix. En ese sentido, las funciones de I/O proporcionan acceso secuencial byte a byte o bloque a bloque, así como desplazamientos arbitrarios en la posición de lectura/escritura; esto en realidad depende de las facilidades inherentes al sistema operativo, pero están presentes en prácticamente todo sistema comercial moderno. En particular, la librería estándar no proporciona funciones de acceso rápido a través de índices, las cuales son implementadas mediante otras librerías adicionales.

C++ y Java proporcionan una interfaz alternativa a la misma funcionalidad a través de jerarquías de clases de I/O con diferente nivel de refinamiento, lo que los hace más extensibles aunque no necesariamente más convenientes. La tendencia en general apunta a no extender el lenguaje en este camino y por el contrario, crear nuevas librerías auxiliares para casos concretos.

Pedagogía

En principio, ni C ni C++ fueron creados para ser sencillos de aprender. C fue creado principalmente para ser eficiente, y C++ para ser a la vez eficiente y rico en características. Java, por el contrario tuvo desde el principio la intención de ser un lenguaje muy fácil de comprender y utilizar, y si bien eso no significa que su aprendizaje sea rápido ni trivial,

ciertamente libera al estudiante de diversos aspectos confusos y sintaxis oscura de los otros lenguajes. Esta es quizá una de las razones más importantes que ha contribuido a su rápida adopción.

Generalidad

Los tres lenguajes estudiados se proponen como "de propósito general", es decir, serían adecuados para atacar prácticamente cualquier clase de problema. En la práctica, el C suele ser utilizado para construir componentes básicos o de bajo nivel (notablemente, el *kernel* de muchos sistemas operativos) mientras que C++ y Java tienen un espectro mucho más amplio (por ejemplo, aplicaciones comerciales de toda clase.) Notablemente Java, en gran medida gracias a la previsión y publicidad de *Sun* y diversos vendedores de "servidores de aplicación", es muy utilizado actualmente en el contexto de servidores Web (Servlets y JSP), acompañado en muchos casos de una arquitectura multicapa.

Cuadro Resumen

Característica	C	C++	Java
Expresividad	Regular	Excesiva	Muy buena
Bien definido	Regular	Muy buena	Muy buena
Tipos de estructuras y datos	Deficiente	Muy buena	Muy buena
Modularidad	Regular	Muy buena	Muy buena
Facilidades de Entrada/Salida	Buena	Buena	Buena
Manipulación de archivos	Regular	Buena	Buena
Pedagogía	Regular	Regular	Buena
Generalidad	Buena	Muy buena	Muy buena

Capítulo 9

Anexos

ProblemasDeEtiquetado/src/generalidades/
ConjuntoListas.java

```
package generalidades;

import java.util.ArrayList;
import java.util.List;

/**
 * Clase que representa un objeto definido por una lista de puntos y una lista
 * de etiquetas.
 *
 * @author José Manuel Ponce
 */
public class ConjuntoListas
{
	/**
	 * Lista de puntos.
	 */
	List<Integer> listaPuntos = new ArrayList<Integer>();

	/**
	 * Lista de etiquetas.
	 */
	List<Integer> listaEtiquetas = new ArrayList<Integer>();

	/**
	 * Constructor por defecto.
	 */
	public ConjuntoListas()
	{
		super();
	}

	/**
	 * Constructor por parámetros.
	 *
	 * @param listPuntos
	 *            Lista de puntos
	 * @param listEtiquetas
	 *            Lista de etiquetas
	 */
	public ConjuntoListas(List<Integer> listPuntos, List<Integer>
listEtiquetas)
	{
		this.listaPuntos = listPuntos;
		this.listaEtiquetas = listEtiquetas;
	}

	/**
	 * Obtiene el Lista de puntos.
	 *
	 * @return listaPuntos Lista de puntos
	 */
	public List<Integer> getListaPuntos()
	{
		return listaPuntos;
	}
```

```
	/**
	 * Establece el Lista de puntos.
	 *
	 * @param listaPuntos
	 *            listaPuntos Lista de puntos
	 */
	public void setListaPuntos(List<Integer> listaPuntos)
	{
		this.listaPuntos = listaPuntos;
	}

	/**
	 * Obtiene el Lista de etiquetas.
	 *
	 * @return listaEtiquetas Lista de etiquetas
	 */
	public List<Integer> getListaEtiquetas()
	{
		return listaEtiquetas;
	}

	/**
	 * Establece el Lista de etiquetas.
	 *
	 * @param listaEtiquetas
	 *            listaEtiquetas Lista de etiquetas
	 */
	public void setListaEtiquetas(List<Integer> listaEtiquetas)
	{
		this.listaEtiquetas = listaEtiquetas;
	}
}
```

ProblemasDeEtiquetado/src/generalidades/
ConjuntoTresListas.java

```
package generalidades;

import java.util.ArrayList;
import java.util.List;

/**
 * Clase que representa un objeto definido por una lista de puntos y una lista
 * de etiquetas.
 *
 * @author José Manuel Ponce
 */
public class ConjuntoTresListas
{
	/**
	 * Lista de puntos.
	 */
	List<Integer> listaPuntos = new ArrayList<Integer>();

	/**
	 * Lista de alturas de etiquetas.
	 */
	List<Integer> alturas = new ArrayList<Integer>();

	/**
	 * Lista de anchuras de etiquetas.
	 */
	List<Integer> anchuras = new ArrayList<Integer>();

	/**
	 * Constructor por defecto.
	 */
	public ConjuntoTresListas()
	{
		super();
	}

	/**
	 * Constructor por parámetros.
	 *
	 * @param listPuntos
	 *            Lista de puntos
	 * @param listAlturas
	 *            Lista de alturas de etiquetas
	 * @param listAnchuras
	 *            Lista de anchuras de etiquetas
	 */
	public ConjuntoTresListas(List<Integer> listPuntos,
			List<Integer> listAlturas, List<Integer> listAnchuras)
	{
		this.listaPuntos = listPuntos;
		this.alturas = listAlturas;
		this.anchuras = listAnchuras;
	}
```

```
	/**
	 * Obtiene el Lista de puntos.
	 *
	 * @return listaPuntos Lista de puntos
	 */
	public List<Integer> getListaPuntos()
	{
		return listaPuntos;
	}

	/**
	 * Establece el Lista de puntos.
	 *
	 * @param listaPuntos
	 *            listaPuntos Lista de puntos
	 */
	public void setListaPuntos(List<Integer> listaPuntos)
	{
		this.listaPuntos = listaPuntos;
	}

	/**
	 * Obtiene el Lista de alturas de etiquetas.
	 * @return alturas Lista de alturas de etiquetas
	 */
	public List<Integer> getAlturas()
	{
		return alturas;
	}

	/**
	 * Establece el Lista de alturas de etiquetas.
	 * @param alturas alturas Lista de alturas de etiquetas
	 */
	public void setAlturas(List<Integer> alturas)
	{
		this.alturas = alturas;
	}

	/**
	 * Obtiene el Lista de anchuras de etiquetas.
	 * @return anchuras Lista de anchuras de etiquetas
	 */
	public List<Integer> getAnchuras()
	{
		return anchuras;
	}

	/**
	 * Establece el Lista de anchuras de etiquetas.
	 * @param anchuras anchuras Lista de anchuras de etiquetas
	 */
	public void setAnchuras(List<Integer> anchuras)
	{
		this.anchuras = anchuras;
	}
}
```

ProblemasDeEtiquetado/src/generalidades/
Constantes.java

```
package generalidades;

/**
 * Clase que almacena las constantes que se utilizarán en la aplicación.
 *
 * @author José Manuel Ponce
 */
public class Constantes
{

	/**
	 * Constante que indica (Top Left) que la etiqueta se sitúa arriba a la
	 * izquierda del punto.
	 */
	public static final String POS_ARRIBA_IZQUIERDA = "TL";

	/**
	 * Constante que indica (Top Right) que la etiqueta se sitúa arriba a
	 * la derecha del punto.
	 */
	public static final String POS_ARRIBA_DERECHA = "TR";

	/**
	 * Constante que indica (Bottom Left) que la etiqueta se sitúa abajo a
	 * la izquierda del punto.
	 */
	public static final String POS_ABAJO_IZQUIERDA = "BL";

	/**
	 * Constante que indica (Bottom Right) que la etiqueta se sitúa abajo a
	 * la derecha del punto.
	 */
	public static final String POS_ABAJO_DERECHA = "BR";

	/**
	 * Constante que indica (Top) que la etiqueta se sitúa encima del
	 * punto.
	 */
	public static final String POS_ARRIBA = "TT";

	/**
	 * Constante que indica (Bottom) que la etiqueta se sitúa debajo del
	 * punto.
	 */
	public static final String POS_ABAJO = "BB";

	/**
	 * Constante que indica (Right) que la etiqueta se sitúa a la derecha
	 * del punto.
	 */
	public static final String POS_DERECHA = "RR";

	/**
	 * Constante que indica (Left) que la etiqueta se sitúa a la izquierda
	 * del punto.
	 */
	public static final String POS_IZQUIERDA = "LL";
```

```
        // problema H-4P

        /**
         * Constante con el texto que aparece en la cabecera del problema al
         * que se refiere.
         */
        public static final String CABECERA_H_4P =
"#########################################################################\n"
+"############ EJECUCIÓN DEL PROBLEMA DE ETIQUETADO H-4P ###############\n"
+"#########################################################################\n";

        /**
         * Constante con la información necesaria acerca del problema al que se
         * refiere.
         */
        public static final String INFO_H_4P =
"\nDada una línea horizontal, se va a proceder a etiquetar N puntos con\n"
+ "sus respectivas N etiquetas. Tanto el conjunto de puntos, como el de
etiquetas\n"
+ "serán introducidos ambos por el usuario cuando se solicite por pantalla\n"
+ "o mediante un fichero de texto que también introducirá el usuario.\n"
+ "La salida que proporcionará el programa será una lista de posiciones\n"
+ "tales que, el primer elemento de la lista corresponderá a la posición\n"
+ "de la primera etiqueta respecto del primer punto; el segundo elemento\n"
+ "de la lista de posiciones corresponderá a la posición de la segunda\n"
+ "etiqueta respecto del segundo punto; y así sucesivamente.\n"
+ "Las posiciones relativas serán las siguientes:\n"
+ "     TL -> (Top Left) la etiqueta se sitúa arriba a la izquierda del punto\n"
+ "     TR -> (Top Right) la etiqueta se sitúa arriba a la derecha del punto\n"
+ "     BL -> (Bottom Left) la etiqueta se sitúa abajo a la izquierda del
punto\n"
+ "     BL -> (Bottom Right) la etiqueta se sitúa abajo a la derecha del
punto\n"
+ "Si el problema no tiene solución, se omitirá todo o parte del listado
anterior y se indicará\n"
+ "la imposibilidad de etiquetar sobre los puntos dados.\n";

        /**
         * Constante con la información necesaria acerca del formato de un
         * fichero de entrada de datos para el problema que representa.
         */
        public static final String INFO_FICHERO_H_4P =
"Debe crear un fichero de texto y guardarlo con el nombre \"entrada_H-4P.txt\"
en la ubicación\n"
+ "donde se encuentra el proyecto. El formato del fichero será el siguiente:\n"
+ "- La primera línea es un número entero que representa el total de puntos a
etiquetar (N).\n"
+ "- Las siguientes N líneas serán también números enteros que representan los
puntos.\n"
+ "- Las N últimas serán igualmente números enteros que representan las
etiquetas.\n"
+ "* No se permiten líneas en blanco ni números separados por espacios.\n";
```

```
        // problema H-4S-1C

        /**
         * Constante con el texto que aparece en la cabecera del problema al
         * que se refiere.
         */
        public static final String CABECERA_H_4S_1C =
"###############################################################################\n"
+"############ EJECUCIÓN DEL PROBLEMA DE ETIQUETADO H-4S-1C ###############\n"
+"###############################################################################\n";

        /**
         * Constante con la información necesaria acerca del problema al que se
         * refiere.
         */
        public static final String INFO_H_4S_1C =
"\nDada una línea horizontal, se va a proceder a etiquetar N puntos con\n"
+ "sus respectivas N etiquetas cuadradas. Tanto el conjunto de puntos como la
dimensión de\n"
+ "las etiquetas serán introducidos ambos por el usuario cuando se solicite por
pantalla\n"
+ "o mediante un fichero de texto que también introducirá el usuario.\n"
+ "La salida que proporcionará el programa será una lista de posiciones\n"
+ "tales que, el primer elemento de la lista corresponderá a la posición\n"
+ "de la primera etiqueta respecto del primer punto; el segundo elemento\n"
+ "de la lista de posiciones corresponderá a la posición de la segunda\n"
+ "etiqueta respecto del segundo punto; y así sucesivamente.\n"
+ "Las posiciones relativas serán las siguientes:\n"
+ "     TL -> (Top Left) la etiqueta se sitúa arriba a la izquierda del punto\n"
+ "     TR -> (Top Right) la etiqueta se sitúa arriba a la derecha del punto\n"
+ "     BL -> (Bottom Left) la etiqueta se sitúa abajo a la izquierda del
punto\n"
+ "     BL -> (Bottom Right) la etiqueta se sitúa abajo a la derecha del
punto\n"
+ "     TT -> (Top) la etiqueta se sitúa encima del punto\n"
+ "     BB -> (Bottom) la etiqueta se sitúa debajo del punto\n"
+ "Si el problema no tiene solución, se omitirá todo o parte del listado
anterior y se indicará\n"
+ "la imposibilidad de etiquetar sobre los puntos dados.\n";

        /**
         * Constante con la información necesaria acerca del formato de un
         * fichero de entrada de datos para el problema que representa.
         */
        public static final String INFO_FICHERO_H_4S_1C =
"Debe crear un fichero de texto y guardarlo con el nombre \"entrada_H-4S-
1C.txt\" en la ubicación\n"
+ "donde se encuentra el proyecto. El formato del fichero será el siguiente:\n"
+ "- La primera línea es un número entero que representa el total de puntos a
etiquetar (N).\n"
+ "- Las siguientes N líneas serán también números enteros que representan los
puntos.\n"
+ "- La última línea contiene la dimensión (cuadrada) de las etiquetas.\n"
+ "* No se permiten líneas en blanco ni números separados por espacios.\n";









        // problema H-4S
```

```
        /**
         * Constante con el texto que aparece en la cabecera del problema al
         * que se refiere.
         */
        public static final String CABECERA_H_4S =
"#########################################################################\n"
+"############ EJECUCIÓN DEL PROBLEMA DE ETIQUETADO H-4S ###############\n"
+"#########################################################################\n";

        /**
         * Constante con la información necesaria acerca del problema al que se
         * refiere.
         */
        public static final String INFO_H_4S =
"\nDada una línea horizontal, se va a proceder a etiquetar N puntos con\n"
+ "sus respectivas N etiquetas. Tanto el conjunto de puntos como la dimensión de
las\n"
+ "etiquetas serán introducidos ambos por el usuario cuando se solicite por
pantalla\n"
+ "o mediante un fichero de texto que también introducirá el usuario.\n"
+ "La salida que proporcionará el programa será una lista de posiciones\n"
+ "tales que, el primer elemento de la lista corresponderá a la posición\n"
+ "de la primera etiqueta respecto del primer punto; el segundo elemento\n"
+ "de la lista de posiciones corresponderá a la posición de la segunda\n"
+ "etiqueta respecto del segundo punto; y así sucesivamente.\n"
+ "Las posiciones relativas serán las siguientes:\n"
+ "     TL -> (Top Left) la etiqueta se sitúa arriba a la izquierda del punto\n"
+ "     TR -> (Top Right) la etiqueta se sitúa arriba a la derecha del punto\n"
+ "     BL -> (Bottom Left) la etiqueta se sitúa abajo a la izquierda del
punto\n"
+ "     BL -> (Bottom Right) la etiqueta se sitúa abajo a la derecha del
punto\n"
+ "     TT -> (Top) la etiqueta se sitúa encima del punto\n"
+ "     BB -> (Bottom) la etiqueta se sitúa debajo del punto\n"
+ "Si el problema no tiene solución, se omitirá todo o parte del listado
anterior y se indicará\n"
+ "la imposibilidad de etiquetar sobre los puntos dados.\n";

        /**
         * Constante con la información necesaria acerca del formato de un
         * fichero de entrada de datos para el problema que representa.
         */
        public static final String INFO_FICHERO_H_4S =
"Debe crear un fichero de texto y guardarlo con el nombre \"entrada_H-4S-
1C.txt\" en la ubicación\n"
+ "donde se encuentra el proyecto. El formato del fichero será el siguiente:\n"
+ "- La primera línea es un número entero que representa el total de puntos a
etiquetar (N).\n"
+ "- Las siguientes N líneas serán también números enteros que representan los
puntos.\n"
+ "- Las últimas N línea contendrán las anchuras de las etiquetas.\n"
+ "* No se permiten líneas en blanco ni números separados por espacios.\n";









        // problema OC-4P
```

```
        /**
         * Constante con el texto que aparece en la cabecera del problema al
         * que se refiere.
         */
        public static final String CABECERA_OC_4P =
"#########################################################################\n"
+"############ EJECUCIÓN DEL PROBLEMA DE ETIQUETADO OC-4P ###############\n"
+"#########################################################################\n";

        /**
         * Constante con la información necesaria acerca del problema al que se
         * refiere.
         */
        public static final String INFO_OC_4P =
"\nDada una línea oblicua de pendiente positiva, se va a proceder a etiquetar N
puntos con\n"
+ "sus respectivas N etiquetas. Tanto el conjunto de puntos, como el de
etiquetas\n"
+ "serán introducidos ambos por el usuario cuando se solicite por pantalla\n"
+ "o mediante un fichero de texto que también introducirá el usuario.\n"
+ "Para este problema en particular, las etiquetas serán cuadradas,\n"
+ "debiéndose situar los puntos sobre uno de los cuatro vértices de las
etiquetas.\n"
+ "La salida que proporcionará el programa será una lista de posiciones\n"
+ "tales que, el primer elemento de la lista corresponderá a la posición\n"
+ "de la primera etiqueta respecto del primer punto; el segundo elemento\n"
+ "de la lista de posiciones corresponderá a la posición de la segunda\n"
+ "etiqueta respecto del segundo punto; y así sucesivamente.\n"
+ "Las posiciones relativas serán las siguientes:\n"
+ "     TL -> (Top Left) la etiqueta se sitúa arriba a la izquierda del punto\n"
+ "     TR -> (Top Right) la etiqueta se sitúa arriba a la derecha del punto\n"
+ "     BL -> (Bottom Left) la etiqueta se sitúa abajo a la izquierda del
punto\n"
+ "     BL -> (Bottom Right) la etiqueta se sitúa abajo a la derecha del
punto\n"
+ "Si el problema no tiene solución, se omitirá todo o parte del listado
anterior y se indicará\n"
+ "la imposibilidad de etiquetar sobre los puntos dados.\n";

        /**
         * Constante con la información necesaria acerca del formato de un
         * fichero de entrada de datos para el etiquetado sobre línea oblicua.
         */
        public static final String INFO_FICHERO_OC_4P =
"Debe crear un fichero de texto y guardarlo con el nombre \"entrada_OC-4P.txt\"
en la ubicación\n"
+ "donde se encuentra el proyecto. El formato del fichero será el siguiente:\n"
+ "- La primera línea es un número entero que representa el total de puntos a
etiquetar (N).\n"
+ "- Las siguientes N líneas serán también números enteros que representan los
puntos.\n"
+ "- La última línea será el tamaño (cuadrado) de las etiquetas.\n"
+ "* No se permiten líneas en blanco ni números separados por espacios.\n";









        // problema OR1-4P

        /**
```

```
         * Constante con el texto que aparece en la cabecera del problema al
         * que se refiere.
         */
        public static final String CABECERA_OR1_4P =
"################################################################################\n"
+"############ EJECUCIÓN DEL PROBLEMA DE ETIQUETADO OR(1)-4P ###############\n"
+"################################################################################\n";

        /**
         * Constante con la información necesaria acerca del problema al que se
         * refiere.
         */
        public static final String INFO_OR1_4P =
"\nDada una línea oblicua de pendiente positiva, se va a proceder a etiquetar N
puntos con\n"
+ "sus respectivas N etiquetas. Tanto el conjunto de puntos, como el de
etiquetas\n"
+ "serán introducidos ambos por el usuario cuando se solicite por pantalla\n"
+ "o mediante un fichero de texto que también introducirá el usuario.\n"
+ "Para este problema en particular, las etiquetas serán rectangulares,\n"
+ "debiéndose situar los puntos sobre uno de los cuatro vértices de las
etiquetas.\n"
+ "La salida que proporcionará el programa será una lista de posiciones\n"
+ "tales que, el primer elemento de la lista corresponderá a la posición\n"
+ "de la primera etiqueta respecto del primer punto; el segundo elemento\n"
+ "de la lista de posiciones corresponderá a la posición de la segunda\n"
+ "etiqueta respecto del segundo punto; y así sucesivamente.\n"
+ "Las posiciones relativas serán las siguientes:\n"
+ "     TL -> (Top Left) la etiqueta se sitúa arriba a la izquierda del punto\n"
+ "     TR -> (Top Right) la etiqueta se sitúa arriba a la derecha del punto\n"
+ "     BL -> (Bottom Left) la etiqueta se sitúa abajo a la izquierda del
punto\n"
+ "     BL -> (Bottom Right) la etiqueta se sitúa abajo a la derecha del
punto\n"
+ "Si el problema no tiene solución, se omitirá todo o parte del listado
anterior y se indicará\n"
+ "la imposibilidad de etiquetar sobre los puntos dados.\n";

        /**
         * Constante con la información necesaria acerca del formato de un
         * fichero de entrada de datos para el etiquetado sobre línea oblicua.
         */
        public static final String INFO_FICHERO_OR1_4P =
"Debe crear un fichero de texto y guardarlo con el nombre \"entrada_OR1-4P.txt\"
en la ubicación\n"
+ "donde se encuentra el proyecto. El formato del fichero será el siguiente:\n"
+ "- La primera línea es un número entero que representa el total de puntos a
etiquetar (N).\n"
+ "- Las siguientes N líneas serán también números enteros que representan los
puntos.\n"
+ "- La penúltima línea es otro número entero que representa, en este caso, la
altura de las etiquetas.\n"
+ "- La última línea es un nuevo número entero que representa la anchura de las
etiquetas.\n"
+ "* No se permiten líneas en blanco ni números separados por espacios.\n";
```

```
        // problema ORM-4P

        /**
         * Constante con el texto que aparece en la cabecera del problema al
         * que se refiere.
         */
        public static final String CABECERA_ORM_4P =
"###############################################################################\n"
+"############ EJECUCIÓN DEL PROBLEMA DE ETIQUETADO OR(m)-4P ###############\n"
+"###############################################################################\n";

        /**
         * Constante con la información necesaria acerca del problema al que se
         * refiere.
         */
        public static final String INFO_ORM_4P =
"\nDada una línea oblicua de pendiente positiva, se va a proceder a etiquetar N
puntos con\n"
+ "sus respectivas N etiquetas. Tanto el conjunto de puntos, como el de
etiquetas\n"
+ "serán introducidos ambos por el usuario cuando se solicite por pantalla\n"
+ "o mediante un fichero de texto que también introducirá el usuario.\n"
+ "Para este problema en particular, las etiquetas serán rectangulares,\n"
+ "debiéndose situar los puntos sobre uno de los cuatro vértices de las
etiquetas.\n"
+ "La salida que proporcionará el programa será una lista de posiciones\n"
+ "tales que, el primer elemento de la lista corresponderá a la posición\n"
+ "de la primera etiqueta respecto del primer punto; el segundo elemento\n"
+ "de la lista de posiciones corresponderá a la posición de la segunda\n"
+ "etiqueta respecto del segundo punto; y así sucesivamente.\n"
+ "Las posiciones relativas serán las siguientes:\n"
+ "     TL -> (Top Left) la etiqueta se sitúa arriba a la izquierda del punto\n"
+ "     TR -> (Top Right) la etiqueta se sitúa arriba a la derecha del punto\n"
+ "     BL -> (Bottom Left) la etiqueta se sitúa abajo a la izquierda del
punto\n"
+ "     BL -> (Bottom Right) la etiqueta se sitúa abajo a la derecha del
punto\n"
+ "Si el problema no tiene solución, se omitirá todo o parte del listado
anterior y se indicará\n"
+ "la imposibilidad de etiquetar sobre los puntos dados.\n";

        /**
         * Constante con la información necesaria acerca del formato de un
         * fichero de entrada de datos para el etiquetado sobre línea oblicua.
         */
        public static final String INFO_FICHERO_ORM_4P =
"Debe crear un fichero de texto y guardarlo con el nombre \"entrada_ORM-4P.txt\"
en la ubicación\n"
+ "donde se encuentra el proyecto. El formato del fichero será el siguiente:\n"
+ "- La primera línea es un número entero que representa el total de puntos a
etiquetar (N).\n"
+ "- Las siguientes N líneas serán también números enteros que representan los
puntos.\n"
+ "- Las N siguientes, más números enteros que representan, en este caso, la
altura de las etiquetas.\n"
+ "- Las últimas N líneas son también números enteros que representas la anchura
de las etiquetas.\n"
+ "* No se permiten líneas en blanco ni números separados por espacios.\n";
```

```
        // problema O1R-4P

        /**
         * Constante con el texto que aparece en la cabecera del problema al
         * que se refiere.
         */
        public static final String CABECERA_O1R_4P =
"##############################################################################\n"
+"############ EJECUCIÓN DEL PROBLEMA DE ETIQUETADO O1R-4P ###############\n"
+"##############################################################################\n";

        /**
         * Constante con la información necesaria acerca del problema al que se
         * refiere.
         */
        public static final String INFO_O1R_4P =
"\nDada una línea oblicua de pendiente positiva, se va a proceder a etiquetar N
puntos con\n"
+ "sus respectivas N etiquetas. Tanto el conjunto de puntos, como el de
etiquetas\n"
+ "serán introducidos ambos por el usuario cuando se solicite por pantalla\n"
+ "o mediante un fichero de texto que también introducirá el usuario.\n"
+ "Para este problema en particular, las etiquetas serán rectangulares de\n"
+ "altura unidad y anchura la que se indique, debiéndose situar\n"
+ "los puntos sobre uno de los cuatro vértices de las etiquetas.\n"
+ "La salida que proporcionará el programa será una lista de posiciones\n"
+ "tales que, el primer elemento de la lista corresponderá a la posición\n"
+ "de la primera etiqueta respecto del primer punto; el segundo elemento\n"
+ "de la lista de posiciones corresponderá a la posición de la segunda\n"
+ "etiqueta respecto del segundo punto; y así sucesivamente.\n"
+ "Las posiciones relativas serán las siguientes:\n"
+ "     TL -> (Top Left) la etiqueta se sitúa arriba a la izquierda del punto\n"
+ "     TR -> (Top Right) la etiqueta se sitúa arriba a la derecha del punto\n"
+ "     BL -> (Bottom Left) la etiqueta se sitúa abajo a la izquierda del
punto\n"
+ "     BL -> (Bottom Right) la etiqueta se sitúa abajo a la derecha del
punto\n"
+ "Si el problema no tiene solución, se omitirá todo o parte del listado
anterior y se indicará\n"
+ "la imposibilidad de etiquetar sobre los puntos dados.\n";

        /**
         * Constante con la información necesaria acerca del formato de un
         * fichero de entrada de datos para el etiquetado sobre línea oblicua.
         */
        public static final String INFO_FICHERO_O1R_4P =
"Debe crear un fichero de texto y guardarlo con el nombre \"entrada_O1R-4P.txt\"
en la ubicación\n"
+ "donde se encuentra el proyecto. El formato del fichero será el siguiente:\n"
+ "- La primera línea es un número entero que representa el total de puntos a
etiquetar (N).\n"
+ "- Las siguientes N líneas serán también números enteros que representan los
puntos.\n"
+ "- Las N siguientes líneas también son números enteros que representan la
anchura de las etiquetas."
+ "- No es necesario indicar la altura de las etiquetas, ya que serán todas de
altura unidad.\n"
+ "* No se permiten líneas en blanco ni números separados por espacios.\n";
```

```
        // problema OC-4S

        /**
         * Constante con el texto que aparece en la cabecera del problema al
         * que se refiere.
         */
        public static final String CABECERA_OC_4S =
"##############################################################################\n"
+"############ EJECUCIÓN DEL PROBLEMA DE ETIQUETADO OC-4S ###############\n"
+"##############################################################################\n";

        /**
         * Constante con la información necesaria acerca del problema al que se
         * refiere.
         */
        public static final String INFO_OC_4S =
"\nDada una línea oblicua de pendiente positiva, se va a proceder a etiquetar N
puntos con\n"
+ "sus respectivas N etiquetas. Tanto el conjunto de puntos, como el de
etiquetas\n"
+ "serán introducidos ambos por el usuario cuando se solicite por pantalla\n"
+ "o mediante un fichero de texto que también introducirá el usuario.\n"
+ "Para este problema en particular, las etiquetas serán cuadradas,\n"
+ "debiéndose situar los puntos sobre cualquier lado o vértice de las
etiquetas.\n"
+ "La salida que proporcionará el programa será una lista de posiciones\n"
+ "tales que, el primer elemento de la lista corresponderá a la posición\n"
+ "de la primera etiqueta respecto del primer punto; el segundo elemento\n"
+ "de la lista de posiciones corresponderá a la posición de la segunda\n"
+ "etiqueta respecto del segundo punto; y así sucesivamente.\n"
+ "Las posiciones relativas serán las siguientes:\n"
+ " TL -> (Top Left) la etiqueta se sitúa arriba a la izquierda del punto\n"
+ " TR -> (Top Right) la etiqueta se sitúa arriba a la derecha del punto\n"
+ " BL -> (Bottom Left) la etiqueta se sitúa abajo a la izquierda del punto\n"
+ " BL -> (Bottom Right) la etiqueta se sitúa abajo a la derecha del punto\n"
+ " RR -> (Right) la etiqueta se sitúa a la derecha del punto\n"
+ " LL -> (Left) la etiqueta se sitúa a la izquierda del punto\n"
+ " TT -> (Top) la etiqueta se sitúa encima del punto\n"
+ " BB -> (Bottom) la etiqueta se sitúa debajo del punto\n"
+ "Si el problema no tiene solución, se omitirá todo o parte del listado
anterior y se indicará\n"
+ "la imposibilidad de etiquetar sobre los puntos dados.\n";

        /**
         * Constante con la información necesaria acerca del formato de un
         * fichero de entrada de datos para el etiquetado sobre línea oblicua.
         */
        public static final String INFO_FICHERO_OC_4S =
"Debe crear un fichero de texto y guardarlo con el nombre \"entrada_OC-4P.txt\"
en la ubicación\n"
+ "donde se encuentra el proyecto. El formato del fichero será el siguiente:\n"
+ "- La primera línea es un número entero que representa el total de puntos a
etiquetar (N).\n"
+ "- Las siguientes N líneas serán también números enteros que representan los
puntos.\n"
+ "- La última línea será el tamaño (cuadrado) de las etiquetas.\n"
+ "* No se permiten líneas en blanco ni números separados por espacios.\n";

}
```

ProblemasDeEtiquetado/src/generalidades/
ListaEnteros.java

```
/**
 *
 */
package generalidades;

import java.util.List;

/**
 * Clase que representa una lista de enteros.
 * @author José Manuel Ponce
 *
 */
public class ListaEnteros
{
	/**
	 * Lista de enteros.
	 */
	List<Integer> lista;

	/**
	 * Constructor sin parámetros.
	 */
	public ListaEnteros()
	{
		super();
	}

	/**
	 * Obtiene el Lista de enteros.
	 * @return lista Lista de enteros
	 */
	public List<Integer> getLista()
	{
		return lista;
	}

	/**
	 * Establece el Lista de enteros.
	 * @param lista lista Lista de enteros
	 */
	public void setLista(List<Integer> lista)
	{
		this.lista = lista;
	}
}
```

ProblemasDeEtiquetado/src/generalidades/
Utilidades.java

```
package generalidades;

import java.util.Iterator;
import java.util.List;

/**
 * Clase que contiene las distintas utilidades de carácter general que se
usarán para resolver el conjunto de problemas.
 *
 * @author José Manuel Ponce
 */
public class Utilidades
{
	/**
	 * Método que comprueba si los valores de la lista pasada como
	 * parámetro (que se corresponde con la lista de puntos o de etiquetas)
	 * son válidos.
	 * Se consideran válidos todos los valores mayores o iguales que cero.
	 *
	 * @param lista
	 *            List<Integer> Lista de puntos o etiquetas
	 * @return boolean Cierto si todos los valores de la lista son válidos
	 */
	public static boolean valoresValidos(List<Integer> lista)
	{
		boolean validos = true;
		Integer valor;
		Iterator<Integer> it = lista.listIterator();
		while (it.hasNext() && validos)
		{
			valor = it.next();
			if (0 >= valor)
			{
				validos = false;
			}
		}
		return validos;
	}

	/**
	 * Método que comprueba si todos los puntos se encuentran ordenados
	 * ascendentemente. Se consideran que los puntos están ordenados si
	 * cualquier punto (exceptuando el primero) es mayor o igual que el
	 * anterior.
	 *
	 * @param puntos
	 *            List<Integer> Lista de puntos
	 * @return boolean Cierto si todos los puntos se encuentran ordenados
	 *         ascendentemente
	 */
	public static boolean puntosOrdenados(List<Integer> puntos)
	{
		boolean ordenados = true;
		Integer valorAnterior = -1;
		Integer valorActual;
		Iterator<Integer> it = puntos.listIterator();
		while (it.hasNext() && ordenados)
		{
			valorActual = it.next();
```

```
                        if (valorActual < valorAnterior)
                        {
                                ordenados = false;
                        } else
                        {
                                valorAnterior = valorActual;
                        }
                }
                return ordenados;
        }

        /**
         * Método de ordenación "burbuja". Ordena ascendentemente los elementos
         * de la lista pasada como parámetro.
         *
         * @param lista
         *            List<Integer> Lista desordenada
         * @return ListaEnteros Lista ordenada
         */
        public static ListaEnteros ordenarListaInteger(
                        List<Integer> lista)
        {
                int tam = lista.size();
                ListaEnteros dev = new ListaEnteros();

                int i, j;
                Integer aux;
                for (i = 0; i < tam; i++)
                {
                        for (j = 0; j < tam - 1; j++)
                        {
                                if (lista.get(j) > lista.get(j + 1))
                                {
                                        aux = lista.get(j);
                                        lista.set(j, lista.get(j + 1));
                                        lista.set(j + 1, aux);
                                }
                        }
                }

                dev.setLista(lista);
                return dev;
        }

        /**
         * Método de ordenación "burbuja". Ordena ascendentemente los elementos
         * de las listas pasadas como parámetro.
         *
         * @param listaPuntos
         *            List<Integer> Lista desordenada con los puntos
         * @param listaEtiquetas
         *            List<Integer> Lista desordenada con las etiquetas
         * @return ConjuntoListas Listas de puntos y etiquetas ordenadas
         */
        public static ConjuntoListas ordenarListasInteger(
                List<Integer> listaPuntos, List<Integer> listaEtiquetas)
        {
                int tam = listaPuntos.size();
                ConjuntoListas conjuntoDev;
                int i, j;
                Integer aux;
                for (i = 0; i < tam; i++)
                {
```

```
                    for (j = 0; j < tam - 1; j++)
                    {
                            if (listaPuntos.get(j) > listaPuntos.get(j +
                                                                    1))
                            {
                                    aux = listaPuntos.get(j);
                                    listaPuntos.set(j, listaPuntos.get(j +
                                                                    1));
                                    listaPuntos.set(j + 1, aux);
                                    aux = listaEtiquetas.get(j);
                                    listaEtiquetas.set(j,
                                            listaEtiquetas.get(j + 1));
                                    listaEtiquetas.set(j + 1, aux);
                            }
                    }
            }
            conjuntoDev = new ConjuntoListas(listaPuntos, listaEtiquetas);
            return conjuntoDev;
    }

    /**
     * Método de ordenación "burbuja". Ordena ascendentemente los elementos
     * de las listas pasadas como parámetro.
     *
     * @param listaPuntos
     *            List<Integer> Lista desordenada con los puntos
     * @param listaAlturas
     *            List<Integer> Lista desordenada con las alturas de las
     *                                                    etiquetas
     * @param listaAnchuras
     *            List<Integer> Lista desordenada con las anchuras de las
     *                                                    etiquetas
     * @return ConjuntoTresListas Listas de puntos, anchuras y alturas de
     *                                            etiquetas ordenadas
     */
    public static ConjuntoTresListas ordenarTresListasInteger(
                    List<Integer> listaPuntos, List<Integer> listaAlturas,
                    List<Integer> listaAnchuras)
    {
            int tam = listaPuntos.size();
            ConjuntoTresListas conjuntoDev;

            int i, j;
            Integer aux;
            for (i = 0; i < tam; i++)
            {
                    for (j = 0; j < tam - 1; j++)
                    {
                            if (listaPuntos.get(j) > listaPuntos.get(j +
                                                                    1))
                            {
                                    aux = listaPuntos.get(j);
                                    listaPuntos.set(j, listaPuntos.get(j +
                                                                    1));
                                    listaPuntos.set(j + 1, aux);
                                    aux = listaAlturas.get(j);
                                    listaAlturas.set(j, listaAlturas.get(j
                                                                    + 1));
                                    listaAlturas.set(j + 1, aux);
                                    aux = listaAnchuras.get(j);
                                    listaAnchuras.set(j,
                                            listaAnchuras.get(j + 1));
                                    listaAnchuras.set(j + 1, aux);
```

```
                        }
                }
        }
        conjuntoDev = new ConjuntoTresListas(listaPuntos, listaAlturas,
                                                listaAnchuras);
        return conjuntoDev;
}

/**
 * Método que se encarga de imprimir una lista de Integer por pantalla.
 *
 * @param lista
 *            List<Integer> Lista que se imprime
 */
public static void imprimeListaInteger(List<Integer> lista)
{
        String cadenaLista = "( ";
        int numElementos = lista.size();
        for (int i = 0; i < numElementos; i++)
        {
                cadenaLista += lista.get(i);
                if ((numElementos - 1) != i)
                {
                        cadenaLista += " , ";
                }
        }
        cadenaLista += " )";
        System.out.println(cadenaLista);
}

/**
 * Método que se encarga de imprimir una lista de String por pantalla.
 *
 * @param lista
 *            List<String> Lista que se imprime
 */
public static void imprimeListaString(List<String> lista)
{
        String cadenaLista = "( ";
        int numElementos = lista.size();
        for (int i = 0; i < numElementos; i++)
        {
                cadenaLista += lista.get(i);
                if ((numElementos - 1) != i)
                {
                        cadenaLista += " , ";
                }
        }
        cadenaLista += " )";
        System.out.println(cadenaLista);
}

/**
 * Método que devuelve el tamaño del elemento Integer mayor de la lista
 * que se le pasa como parámetro.
 *
 * @param lista
 *            List<Integer> Lista de elementos Integer
 * @return int Tamaño del elemento mayor de la lista
 */
public static int maxValorLista(List<Integer> lista)
{
        int maxValor = -1;
```

```
                Iterator<Integer> it = lista.listIterator();
                Integer valor;
                while (it.hasNext())
                {
                        valor = it.next();
                        if (valor > maxValor)
                        {
                                maxValor = valor;
                        }
                }

                return maxValor;
        }

        /**
         * Método que devuelve el módulo 10 del entero que se le pasa como
         * parámetro.
         *
         * @param num
         *            Entero del que se quiere hallar el módulo 10
         * @return Módulo 10 del entero que se le pasa como parámetro
         */
        public static int mod10(int num)
        {
                int dev = num % 10;
                return dev;
        }

}
```

ProblemasDeEtiquetado/src/lineaHorizontal/h_4p/
Main_H_4P.java

```
package lineaHorizontal.h_4p;

import generalidades.Constantes;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

/**
 * Clase principal que resuelve el problema H-4P.
 *
 * @author José Manuel Ponce
 */
public class Main_H_4P
{
/**
 * Instancia de un objeto Solucion_H_4P.
 */
private static Solucion_H_4P solucion;

/**
 * Código a ejecutar.
 *
 * @param args
 *           Argumentos
 */
public static void main(String[] args)
{
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
System.out.println(Constantes.CABECERA_H_4P);
System.out.println();

boolean salir = false, error = false, finProblema = false;
int i = 0, numElementos = 0;
String cadenaEntrada = "";
List<Integer> puntos = new ArrayList<Integer>();
List<Integer> etiquetas = new ArrayList<Integer>();

try
{
while (!salir && !finProblema)
{
System.out
      .println("Pulse la tecla \"T\" para introducir los datos desde el
teclado.");
System.out
      .println("Pulse la tecla \"F\" si desea que la aplicación obtenga los
datos a partir de un fichero de texto.");
System.out
      .println("Pulse la tecla \"I\" si desea información adicional acerca de
este problema.");
System.out.println("Pulse la tecla \"S\" para salir de la aplicación.");

cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
```

```
{
System.out.println(Constantes.INFO_H_4P);
}

else
{
if ("T".equals(cadenaEntrada) || "t".equals(cadenaEntrada))
{
// El usuario prefiere introducir los datos desde el
// teclado.
System.out
      .println("Introduzca el número de elementos. Pulse \"S\" para salir.");
cadenaEntrada = in.readLine();

if (cadenaEntrada.equals("S") || cadenaEntrada.equals("s"))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else
{
numElementos = Integer.valueOf(cadenaEntrada).intValue();
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
etiquetas = new ArrayList<Integer>(numElementos);
System.out.println("Ha indicado " + numElementos + " elementos. ");
i = 1;
while (i <= numElementos)
{
System.out.println("Introduzca el punto " + i + ":");
puntos.add(Integer.valueOf(in.readLine()).intValue());
i++;
}
i = 1;
while (i <= numElementos)
{
System.out.println("Introduzca el tamaño de la etiqueta " + i + ":");
etiquetas.add(Integer.valueOf(in.readLine()).intValue());
i++;
}
finProblema = true;
}
}
}

else if (cadenaEntrada.equals("F") || cadenaEntrada.equals("f"))
{
// El usuario prefiere utilizar un fichero de texto para
// indicar los datos.
System.out
      .println("Guarde el fichero de texto con el nombre \"entrada_H-4P.txt\"
en la misma ubicación donde se encuentra");
System.out
      .println("el proyecto y pulse \"C\" para continuar. Pulse \"S\" para
abandonar la aplicación.");
System.out
      .println("Si desea más información acerca del formato del fichero pulse
```

```
la tecla \"I\".");

cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
{
System.out.println(Constantes.INFO_FICHERO_H_4P);
}

else
{
if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else if ("C".equals(cadenaEntrada) || "c".equals(cadenaEntrada))
{
System.out.println("Leyendo el fichero:");
BufferedReader br = new BufferedReader(new FileReader("entrada_H-4P.txt"));
String s;
if ((s = br.readLine()) != null)
{
numElementos = Integer.valueOf(s.trim());
}
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
etiquetas = new ArrayList<Integer>(numElementos);
i = 2;
while (((s = br.readLine()) != null) && !error)
{
if (i >= 2 && i <= (numElementos + 1))
{
puntos.add(Integer.valueOf(s.trim()));
} else if (i >= (numElementos + 2) && i <= ((2 * numElementos) + 1))
{
etiquetas.add(Integer.valueOf(s.trim()));
} else
{
error = true;
}
i++;
}
}
br.close();
finProblema = true;
}
}
}

else if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
}
}
```

```
if (finProblema)
{
if ((puntos.size() != etiquetas.size()) || (puntos.size() != numElementos)
      || (numElementos != etiquetas.size()) || error)
{
System.out
      .println("Se ha producido un error en la obtención de los datos: El
número total de puntos y etiquetas no coincide con el indicado.");
salir = true;
}
}
}

if (finProblema && !salir)
{
System.out.println("Los pares (punto , etiqueta) indicados han sido:");
for (i = 0; i < numElementos; i++)
{
System.out.print("par " + (i + 1) + ": (" + puntos.get(i) + " , "
      + etiquetas.get(i) + ")\n");
}

solucion = new Solucion_H_4P(puntos, etiquetas);
solucion.obtenerSolucion();
}

} catch (NumberFormatException e)
{
System.out
      .println("\nSe ha producido una excepción al introducir un valor
incorrecto.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (IOException e)
{
System.out.println("\nSe ha producido una excepción de entrada/salida.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (ArrayIndexOutOfBoundsException e)
{
System.out.println("\nSe ha producido un error.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
}
}
}
```

ProblemasDeEtiquetado/src/lineaHorizontal/h_4p/
Solucion_H_4P.java

```
package lineaHorizontal.h_4p;

import generalidades.Constantes;
import generalidades.Utilidades;

import java.util.ArrayList;
import java.util.List;

import generalidades.ConjuntoListas;
import lineaHorizontal.SombraHorizontal;
import lineaHorizontal.UtilidadesHorizontal;

/**
 * Clase que contiene los métodos necesarios para solucionar el problema H-4P.
 *
 * @author José Manuel Ponce
 */
public class Solucion_H_4P
{
/**
 * Lista de puntos.
 */
private List<Integer> puntos;

/**
 * Lista de etiquetas.
 */
private List<Integer> etiquetas;

/**
 * Lista de posiciones finales de las etiquetas respecto a los puntos.
 */
private List<String> resultado;

/**
 * Lista auxiliar cuyos elementos denotan si están ocupadas por una etiqueta
las
 * posiciones que representan por encima de la línea de entrada.
 */
private List<Integer> lineaSuperior;

/**
 * Lista auxiliar cuyos elementos denotan si están ocupadas por una etiqueta
las
 * posiciones que representan por debajo de la línea de entrada.
 */
private List<Integer> lineaInferior;

/**
 * Número de pares punto-etiqueta.
 */
private int numElementos;

/**
 * Número de divisiones de la línea de entrada.
 */
private int longitudLinea;

/**
```

```
 * Objeto SombraHorizontal.
 */
private SombraHorizontal sombra;

/**
 * Constructor sin parámetros.
 */
public Solucion_H_4P()
{
super();
}

/**
 * Constructor con parámetros.
 *
 * @param puntos
 *           List<Integer> Lista de puntos
 * @param etiquetas
 *           List<Integer> Lista de etiquetas
 */
public Solucion_H_4P(List<Integer> puntos, List<Integer> etiquetas)
{
this.puntos = puntos;
this.etiquetas = etiquetas;
this.numElementos = puntos.size();
this.resultado = new ArrayList<String>();
this.lineaSuperior = new ArrayList<Integer>();
this.lineaInferior = new ArrayList<Integer>();
}

/**
 * Método que imprime la solución al problema, si la hubiese.
 */
public void obtenerSolucion()
{
ConjuntoListas conjunto;
SombraHorizontal sombraTL;
SombraHorizontal sombraBL;
SombraHorizontal sombraTR;
SombraHorizontal sombraBR;
boolean haySolucion = true;

if ((Utilidades.valoresValidos(this.puntos))
      && (Utilidades.valoresValidos(this.etiquetas)))
{
if (!Utilidades.puntosOrdenados(puntos))
{
conjunto = Utilidades.ordenarListasInteger(this.puntos, this.etiquetas);
this.puntos = conjunto.getListaPuntos();
this.etiquetas = conjunto.getListaEtiquetas();
System.out
      .println("La lista de puntos no está ordenada. Procedemos a ordenarla.");
System.out.println("Los pares (punto , etiqueta) indicados han sido:");
for (int i = 0; i < numElementos; i++)
{
System.out.print("par " + (i + 1) + ": (" + this.puntos.get(i) + " , "
      + this.etiquetas.get(i) + ")\n");
}
} else
{
System.out
      .println("La lista de puntos está ordenada. Procedemos a la resolución
del problema:");
```

```
}
inicializaValores();

for (int i = 0; (i < this.numElementos) && haySolucion; i++)
{
sombraTL = this.probarEtiquetaTL(this.puntos.get(i), this.etiquetas.get(i));
sombraBL = this.probarEtiquetaBL(this.puntos.get(i), this.etiquetas.get(i));
sombraTR = this.probarEtiquetaTR(this.puntos.get(i), this.etiquetas.get(i));
sombraBR = this.probarEtiquetaBR(this.puntos.get(i), this.etiquetas.get(i));
System.out.println(">> Sombras resultantes para el punto " + (i + 1)
      + " y anteriores:");
int idSombraMenor = UtilidadesHorizontal.hallarSombraMenor4(sombraTL, sombraBL,
      sombraTR, sombraBR);
if (null == sombraTL)
{
System.out.println("Sombra TL: (-- , --)");
} else
{
System.out.println("Sombra TL: (" + (sombraTL.getTk()) + " , "
      + (sombraTL.getBk()) + ")");
}
if (null == sombraBL)
{
System.out.println("Sombra BL: (-- , --)");
} else
{
System.out.println("Sombra BL: (" + (sombraBL.getTk()) + " , "
      + (sombraBL.getBk()) + ")");
}
if (null == sombraTR)
{
System.out.println("Sombra TR: (-- , --)");
} else
{
System.out.println("Sombra TR: (" + (sombraTR.getTk()) + " , "
      + (sombraTR.getBk()) + ")");
}
if (null == sombraBR)
{
System.out.println("Sombra BR: (-- , --)");
} else
{
System.out.println("Sombra BR: (" + (sombraBR.getTk()) + " , "
      + (sombraBR.getBk()) + ")");
}
if (-1 != idSombraMenor)
{
if (1 == idSombraMenor)
{
this.aplicarEtiquetaTL(this.puntos.get(i), this.etiquetas.get(i), i + 1);
this.resultado.set(i, Constantes.POS_ARRIBA_IZQUIERDA);
} else if (2 == idSombraMenor)
{
this.aplicarEtiquetaBL(this.puntos.get(i), this.etiquetas.get(i), i + 1);
this.resultado.set(i, Constantes.POS_ABAJO_IZQUIERDA);
} else if (3 == idSombraMenor)
{
this.aplicarEtiquetaTR(this.puntos.get(i), this.etiquetas.get(i), i + 1);
this.resultado.set(i, Constantes.POS_ARRIBA_DERECHA);
} else if (4 == idSombraMenor)
{
this.aplicarEtiquetaBR(this.puntos.get(i), this.etiquetas.get(i), i + 1);
this.resultado.set(i, Constantes.POS_ABAJO_DERECHA);
```

```
}
} else
{
haySolucion = false;
}
}

System.out.println("Se han obtenido las siguientes posiciones:");
Utilidades.imprimeListaString(resultado);
if (!haySolucion)
{
System.out.println("No se ha podido encontrar una solución válida a ");
System.out.println("este problema con los datos que ha proporcionado.");
System.out
      .println("Gráficamente, la solución parcial obtenida ha sido la
siguiente:");
}
if (haySolucion)
{
System.out
      .println("Gráficamente, la solución del problema se puede representar
como sigue:");
}
UtilidadesHorizontal.representar(this.longitudLinea, this.puntos,
      this.lineaSuperior, this.lineaInferior);
System.out.println(">> Fin del problema.");
} else
{
System.out
      .println("Los valores de los puntos y etiquetas han de ser mayores que
cero.");
System.out.println("Vuelva a ejecutar el programa si lo desea.");
}
}

/**
 * Método que inicializa las listas "resultado", "lineaSuperior" y
 * "lineaInferior".
 */
public void inicializaValores()
{
this.sombra = new SombraHorizontal(0, 0);
this.longitudLinea = this.puntos.get(numElementos - 1) + 1
      + Utilidades.maxValorLista(this.etiquetas);
for (int i = 0; i <= this.longitudLinea; i++)
{
this.lineaSuperior.add(i, null);
this.lineaInferior.add(i, null);
}
for (int i = 0; i < this.numElementos; i++)
{
this.resultado.add(i, "--");
}
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición TL para comprobar el tamaño de la sombra resultante.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
```

```
 * @return SombraHorizontal Sombra virtual, null si no se puede
 */
public SombraHorizontal probarEtiquetaTL(Integer punto, Integer etiqueta)
{
SombraHorizontal dev = new SombraHorizontal();
boolean sigue = true;
int i;
for (i = punto - 1; (i >= 0) && (i >= punto - etiqueta) && sigue; i--)
{
if (((i <= sombra.getTk()) && (0 < sombra.getTk()))
      || (null != this.lineaSuperior.get(i)))
{
sigue = false;
}
}
if (sigue)
{
dev.setTk(punto - 1);
dev.setBk(this.sombra.getBk());
} else
{
dev = null;
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición BL para comprobar el tamaño de la sombra resultante.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return SombraHorizontal Sombra virtual, null si no se puede
 */
public SombraHorizontal probarEtiquetaBL(Integer punto, Integer etiqueta)
{
SombraHorizontal dev = new SombraHorizontal();
boolean sigue = true;
int i;
for (i = punto - 1; (i >= 0) && (i >= punto - etiqueta) && sigue; i--)
{
if (((i <= sombra.getBk()) && (0 < sombra.getBk()))
      || (null != this.lineaInferior.get(i)))
{
sigue = false;
}
}
if (sigue)
{
dev.setTk(this.sombra.getTk());
dev.setBk(punto - 1);
} else
{
dev = null;
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición TR para comprobar el tamaño de la sombra resultante.
```

```
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return SombraHorizontal Sombra virtual, null si no se puede
 */
public SombraHorizontal probarEtiquetaTR(Integer punto, Integer etiqueta)
{
SombraHorizontal dev = new SombraHorizontal(-1, -1);
boolean sigue = true;
int i;
for (i = punto + 1; (i <= this.longitudLinea) && (i <= punto + etiqueta)
      && sigue; i++)
{
if (((i <= sombra.getTk()) && (0 < sombra.getTk()))
      || (null != this.lineaSuperior.get(i)))
{
sigue = false;
}
}
if (sigue)
{
dev.setTk(punto + etiqueta);
dev.setBk(this.sombra.getBk());
} else
{
dev = null;
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición BR para comprobar el tamaño de la sombra resultante.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return SombraHorizontal Sombra virtual, null si no se puede
 */
public SombraHorizontal probarEtiquetaBR(Integer punto, Integer etiqueta)
{
SombraHorizontal dev = new SombraHorizontal(-1, -1);
boolean sigue = true;
int i;
for (i = punto + 1; (i <= this.longitudLinea) && (i <= punto + etiqueta)
      && sigue; i++)
{
if (((i <= sombra.getBk()) && (0 < sombra.getBk()))
      || (null != this.lineaInferior.get(i)))
{
sigue = false;
}
}
if (sigue)
{
dev.setTk(this.sombra.getTk());
dev.setBk(punto + etiqueta);
} else
{
dev = null;
```

```
}
return dev;
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param indice
 *           Índice aplicado a la etiqueta que se añade
 * @return boolean Cierto si se añade correctamente la etiqueta
 */
public boolean aplicarEtiquetaTL(Integer punto, Integer etiqueta, int indice)
{
indice = indice % 10;
boolean dev = false;
int i;
for (i = punto - 1; (i >= 0) && (i >= punto - etiqueta); i--)
{
this.lineaSuperior.set(i, indice);
}
this.sombra.setTk(punto - 1);
this.sombra.setBk(this.sombra.getBk());
dev = true;
return dev;
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición BL.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param indice
 *           Índice aplicado a la etiqueta que se añade
 * @return boolean Cierto si se añade correctamente la etiqueta
 */
public boolean aplicarEtiquetaBL(Integer punto, Integer etiqueta, int indice)
{
indice = indice % 10;
boolean dev = false;
int i;
for (i = punto - 1; (i >= 0) && (i >= punto - etiqueta); i--)
{
this.lineaInferior.set(i, indice);
}
this.sombra.setTk(this.sombra.getTk());
this.sombra.setBk(punto - 1);
dev = true;
return dev;
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición TR.
 *
 * @param punto
```

```
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param indice
 *           Índice aplicado a la etiqueta que se añade
 * @return boolean Cierto si se añade correctamente la etiqueta
 */
public boolean aplicarEtiquetaTR(Integer punto, Integer etiqueta, int indice)
{
indice = indice % 10;
boolean dev = false;
int i;
for (i = punto + 1; (i <= this.longitudLinea) && (i <= punto + etiqueta); i++)
{
this.lineaSuperior.set(i, indice);
}
this.sombra.setTk(punto + etiqueta);
this.sombra.setBk(this.sombra.getBk());
dev = true;
return dev;
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición BR.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param indice
 *           Índice aplicado a la etiqueta que se añade
 * @return boolean Cierto si se añade correctamente la etiqueta
 */
public boolean aplicarEtiquetaBR(Integer punto, Integer etiqueta, int indice)
{
indice = indice % 10;
boolean dev = false;
int i;
for (i = punto + 1; (i <= this.longitudLinea) && (i <= punto + etiqueta); i++)
{
this.lineaInferior.set(i, indice);
}
this.sombra.setTk(this.sombra.getTk());
this.sombra.setBk(punto + etiqueta);
dev = true;
return dev;
}
}
```

ProblemasDeEtiquetado/src/lineaHorizontal/h_4s/
Main_H_4S.java

```
package lineaHorizontal.h_4s;

import generalidades.Constantes;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

/**
 * Clase principal que resuelve el problema H-4S.
 *
 * @author José Manuel Ponce
 */
public class Main_H_4S
{
/**
 * Instancia de un objeto Solucion_H_4S.
 */
private static Solucion_H_4S solucion;

/**
 * Código a ejecutar.
 *
 * @param args
 *           Argumentos
 */
public static void main(String[] args)
{
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
System.out.println(Constantes.CABECERA_H_4S);
System.out.println();

boolean salir = false, error = false, finProblema = false;
int i = 0, numElementos = 0;
String cadenaEntrada = "";
List<Integer> puntos = new ArrayList<Integer>();
List<Integer> etiquetas = new ArrayList<Integer>();

try
{
while (!salir && !finProblema)
{
System.out
      .println("Pulse la tecla \"T\" para introducir los datos desde el
teclado.");
System.out
      .println("Pulse la tecla \"F\" si desea que la aplicación obtenga los
datos a partir de un fichero de texto.");
System.out
      .println("Pulse la tecla \"I\" si desea información adicional acerca de
este problema.");
System.out.println("Pulse la tecla \"S\" para salir de la aplicación.");

cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
```

```
{
System.out.println(Constantes.INFO_H_4S);
}

else
{
if ("T".equals(cadenaEntrada) || "t".equals(cadenaEntrada))
{
// El usuario prefiere introducir los datos desde el
// teclado.
System.out
      .println("Introduzca el número de elementos. Pulse \"S\" para salir.");
cadenaEntrada = in.readLine();

if (cadenaEntrada.equals("S") || cadenaEntrada.equals("s"))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else
{
numElementos = Integer.valueOf(cadenaEntrada).intValue();
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
System.out.println("Ha indicado " + numElementos + " elementos. ");
i = 1;
while (i <= numElementos)
{
System.out.println("Introduzca el punto " + i + ":");
puntos.add(Integer.valueOf(in.readLine()).intValue());
i++;
}
i = 1;
while (i <= numElementos)
{
System.out.println("Introduzca la anchura de la etiqueta " + i + ":");
etiquetas.add(Integer.valueOf(in.readLine()).intValue());
i++;
}
finProblema = true;
}
}
}

else if (cadenaEntrada.equals("F") || cadenaEntrada.equals("f"))
{
// El usuario prefiere utilizar un fichero de texto para
// indicar los datos.
System.out
      .println("Guarde el fichero de texto con el nombre \"entrada_H-4S.txt\"
en la misma ubicación donde se encuentra");
System.out
      .println("el proyecto y pulse \"C\" para continuar. Pulse \"S\" para
abandonar la aplicación.");
System.out
      .println("Si desea más información acerca del formato del fichero pulse
la tecla \"I\".");
```

```
cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
{
System.out.println(Constantes.INFO_FICHERO_H_4S);
}

else
{
if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else if ("C".equals(cadenaEntrada) || "c".equals(cadenaEntrada))
{
System.out.println("Leyendo el fichero:");
BufferedReader br = new BufferedReader(new FileReader("entrada_H-4S.txt"));
String s;
if ((s = br.readLine()) != null)
{
numElementos = Integer.valueOf(s.trim());
}
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
i = 2;
while (((s = br.readLine()) != null) && !error)
{
if (i >= 2 && i <= (numElementos + 1))
{
puntos.add(Integer.valueOf(s.trim()));
} else if (i >= (numElementos + 2))
{
etiquetas.add(Integer.valueOf(s.trim()));
} else
{
error = true;
}
i++;
}
}
br.close();
finProblema = true;
}
}
}

else if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
}
}

if (finProblema)
```

```
{
if (puntos.size() != numElementos || error)
{
System.out
      .println("Se ha producido un error en la obtención de los datos: El
número total de puntos y etiquetas no coincide con el indicado.");
salir = true;
}
}
}

if (finProblema && !salir)
{
System.out.println("Los pares (punto , etiqueta) indicados han sido:");
for (i = 0; i < numElementos; i++)
{
System.out.print("par " + (i + 1) + ": (" + puntos.get(i) + " , "
      + etiquetas.get(i) + ")\n");
}

solucion = new Solucion_H_4S(puntos, etiquetas);
solucion.obtenerSolucion();
}

} catch (NumberFormatException e)
{
System.out
      .println("\nSe ha producido una excepción al introducir un valor
incorrecto.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (IOException e)
{
System.out.println("\nSe ha producido una excepción de entrada/salida.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (ArrayIndexOutOfBoundsException e)
{
System.out.println("\nSe ha producido un error.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
}
}
}
```

ProblemasDeEtiquetado/src/lineaHorizontal/h_4s/
Solucion_H_4S.java

```
package lineaHorizontal.h_4s;

import generalidades.ConjuntoListas;
import generalidades.Constantes;
import generalidades.Utilidades;

import java.util.ArrayList;
import java.util.List;

import lineaHorizontal.UtilidadesHorizontal;

/**
 * Clase que contiene los métodos necesarios para solucionar el problema H-4S.
 *
 * @author José Manuel Ponce
 */
public class Solucion_H_4S
{
/**
 * Lista de puntos.
 */
private List<Integer> puntos;

/**
 * Lista con los tamaños (cuadrados) de las etiquetas.
 */
private List<Integer> etiquetas;

/**
 * Lista de posiciones finales de las etiquetas respecto a los puntos.
 */
private List<String> resultado;

/**
 * Lista auxiliar cuyos elementos denotan si están ocupadas por una etiqueta
las
 * posiciones que representan por encima de la línea de entrada.
 */
private List<Integer> lineaSuperior;

/**
 * Lista auxiliar cuyos elementos denotan si están ocupadas por una etiqueta
las
 * posiciones que representan por debajo de la línea de entrada.
 */
private List<Integer> lineaInferior;

/**
 * Número de pares punto-etiqueta.
 */
private int numElementos;

/**
 * Número de divisiones de la línea de entrada.
 */
private int longitudLinea;

/**
 * Constructor sin parámetros.
```

```
 */
public Solucion_H_4S()
{
super();
}

/**
 * Constructor con parámetros.
 *
 * @param puntos
 *           List<Integer> Lista de puntos
 * @param etiquetas
 *           Integer Lista de tamaños (cuadrados) de las etiquetas
 */
public Solucion_H_4S(List<Integer> puntos, List<Integer> etiquetas)
{
this.puntos = puntos;
this.etiquetas = etiquetas;
this.numElementos = puntos.size();
this.resultado = new ArrayList<String>();
this.lineaSuperior = new ArrayList<Integer>();
this.lineaInferior = new ArrayList<Integer>();
}

/**
 * Método que imprime la solución al problema, si la hubiese.
 */
public void obtenerSolucion()
{
ConjuntoListas conjunto;
boolean haySolucion = false;

if (Utilidades.valoresValidos(this.puntos)
                && Utilidades.valoresValidos(this.etiquetas))
{
if (!Utilidades.puntosOrdenados(puntos))
{
conjunto = Utilidades.ordenarListasInteger(this.puntos, this.etiquetas);
this.puntos = conjunto.getListaPuntos();
this.etiquetas = conjunto.getListaEtiquetas();
System.out
                .println("La lista de puntos no está ordenada. Procedemos a
ordenarla.");
System.out.println("Los pares (punto , etiqueta) indicados han sido:");
for (int i = 0; i < numElementos; i++)
{
System.out.print("par " + (i + 1) + ": (" + this.puntos.get(i) + " , "
                + this.etiquetas.get(i) + ")\n");
}
} else
{
System.out
                .println("La lista de puntos está ordenada. Procedemos a la
resolución del problema:");
}
inicializaValores();

int desp;
int indice = 1;
for (int i = 0; i < this.numElementos; i++)
{
haySolucion = false;
for (desp = 0; (desp <= (this.etiquetas.get(i) + 1)) && !haySolucion; desp++)
```

```
{
boolean etiquetado = false;
if (!etiquetado
                && probarEtiquetaArriba(this.puntos.get(i),
this.etiquetas.get(i), desp))
{
haySolucion = true;
etiquetado = true;
aplicarEtiquetaArriba(this.puntos.get(i), this.etiquetas.get(i), desp, indice);
indice++;
if (0 == desp)
{
this.resultado.set(i, Constantes.POS_ARRIBA_IZQUIERDA);
} else if ((this.etiquetas.get(i) + 1) == desp)
{
this.resultado.set(i, Constantes.POS_ARRIBA_DERECHA);
} else
{
this.resultado.set(i, Constantes.POS_ARRIBA);
}
} else if (!etiquetado
                && probarEtiquetaAbajo(this.puntos.get(i),
this.etiquetas.get(i), desp))
{
haySolucion = true;
etiquetado = true;
aplicarEtiquetaAbajo(this.puntos.get(i), this.etiquetas.get(i), desp, indice);
indice++;
if (0 == desp)
{
this.resultado.set(i, Constantes.POS_ABAJO_IZQUIERDA);
} else if ((this.etiquetas.get(i) + 1) == desp)
{
this.resultado.set(i, Constantes.POS_ABAJO_DERECHA);
} else
{
this.resultado.set(i, Constantes.POS_ABAJO);
}
}
}
if (!haySolucion)
{
break;
}
}

System.out.println("Se han obtenido las siguientes posiciones:");
Utilidades.imprimeListaString(resultado);
if (!haySolucion)
{
System.out.println("No se ha podido encontrar una solución válida a ");
System.out.println("este problema con los datos que ha proporcionado.");
System.out
                .println("Gráficamente, la solución parcial obtenida ha sido la
siguiente:");
}
if (haySolucion)
{
System.out
                .println("Gráficamente, la solución del problema se puede
representar como sigue:");
}
UtilidadesHorizontal.representar(this.longitudLinea, this.puntos,
```

```
                this.lineaSuperior, this.lineaInferior);
System.out.println(">> Fin del problema.");
} else
{
System.out
                .println("Los valores de los puntos y etiquetas han de ser
mayores que cero.");
System.out.println("Vuelva a ejecutar el programa si lo desea.");
}
}

/**
 * Método que inicializa las listas "resultado", "lineaSuperior" y
 * "lineaInferior".
 */
public void inicializaValores()
{
this.longitudLinea = this.puntos.get(numElementos - 1) + 1
                + Utilidades.maxValorLista(this.etiquetas);
for (int i = 0; i <= this.longitudLinea; i++)
{
this.lineaSuperior.add(i, null);
this.lineaInferior.add(i, null);
}
for (int i = 0; i < this.numElementos; i++)
{
this.resultado.add(i, "--");
}
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado por el
 * desplazamiento en la línea de arriba.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param desp
 *           Desplazamiento
 * @return boolean Cierto si se puede añadir la etiqueta
 */
public boolean probarEtiquetaArriba(Integer punto, Integer etiqueta,
                Integer desp)
{
boolean dev = true;
for (int i = punto - 1 + desp; (i >= 0) && (i >= punto - etiqueta + desp)
                && dev; i--)
{
if (null != this.lineaSuperior.get(i))
{
dev = false;
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado por el
 * desplazamiento en la línea de abajo.
 *
 * @param punto
 *           Punto a etiquetar
```

```
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param desp
 *           Desplazamiento
 * @return boolean Cierto si se puede añadir la etiqueta
 */
public boolean probarEtiquetaAbajo(Integer punto, Integer etiqueta, Integer
desp)
{
boolean dev = true;
int i;
for (i = punto - 1 + desp; (i >= 0) && (i >= punto - etiqueta + desp) && dev;
i--)
{
if (null != this.lineaInferior.get(i))
{
dev = false;
}
}
return dev;
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado por el
 * desplazamiento en la línea de arriba.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param desp
 *           Desplazamiento
 * @param indice
 *           Índice aplicado a la etiqueta que se añade
 * @return boolean Cierto si se añade correctamente la etiqueta
 */
public boolean aplicarEtiquetaArriba(Integer punto, Integer etiqueta,
                Integer desp, int indice)
{
boolean dev = true;
indice = indice % 10;
int i;
for (i = punto - 1 + desp; (i >= 0) && (i >= punto - etiqueta + desp) && dev;
i--)
{
this.lineaSuperior.set(i, indice);
}
return dev;
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado por el
 * desplazamiento en la línea de arriba.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param desp
 *           Desplazamiento
 * @param indice
 *           Índice aplicado a la etiqueta que se añade
 * @return boolean Cierto si se añade correctamente la etiqueta
```

```
 */
public boolean aplicarEtiquetaAbajo(Integer punto, Integer etiqueta,
                Integer desp, int indice)
{
boolean dev = true;
indice = indice % 10;
int i;
for (i = punto - 1 + desp; (i >= 0) && (i >= punto - etiqueta + desp) && dev;
i--)
{
this.lineaInferior.set(i, indice);
}
return dev;
}
}
```

ProblemasDeEtiquetado/src/lineaHorizontal/h_4s_1c/
Main_H_4S_1C.java

```
package lineaHorizontal.h_4s_1c;

import generalidades.Constantes;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

/**
 * Clase principal que resuelve el problema H-4S-1C.
 *
 * @author José Manuel Ponce
 */
public class Main_H_4S_1C
{
/**
 * Instancia de un objeto Solucion_H_4S_1C.
 */
private static Solucion_H_4S_1C solucion;

/**
 * Código a ejecutar.
 *
 * @param args
 *           Argumentos
 */
public static void main(String[] args)
{
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
System.out.println(Constantes.CABECERA_H_4S_1C);
System.out.println();

boolean salir = false, error = false, finProblema = false;
int i = 0, numElementos = 0;
String cadenaEntrada = "";
List<Integer> puntos = new ArrayList<Integer>();
List<Integer> etiquetas = new ArrayList<Integer>();

try
{
while (!salir && !finProblema)
{
System.out
      .println("Pulse la tecla \"T\" para introducir los datos desde el
teclado.");
System.out
      .println("Pulse la tecla \"F\" si desea que la aplicación obtenga los
datos a partir de un fichero de texto.");
System.out
      .println("Pulse la tecla \"I\" si desea información adicional acerca de
este problema.");
System.out.println("Pulse la tecla \"S\" para salir de la aplicación.");

cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
```

```
{
System.out.println(Constantes.INFO_H_4S_1C);
}

else
{
if ("T".equals(cadenaEntrada) || "t".equals(cadenaEntrada))
{
// El usuario prefiere introducir los datos desde el
// teclado.
System.out
      .println("Introduzca el número de elementos. Pulse \"S\" para salir.");
cadenaEntrada = in.readLine();

if (cadenaEntrada.equals("S") || cadenaEntrada.equals("s"))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else
{
numElementos = Integer.valueOf(cadenaEntrada).intValue();
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
System.out.println("Ha indicado " + numElementos + " elementos. ");
i = 1;
while (i <= numElementos)
{
System.out.println("Introduzca el punto " + i + ":");
puntos.add(Integer.valueOf(in.readLine()).intValue());
i++;
}
i = 1;
System.out.println("Introduzca el tamaño (cuadrado) de la etiqueta:");
Integer etiquetaAux = Integer.valueOf(in.readLine());
for (i = 0; i < numElementos; i++)
{
etiquetas.add(etiquetaAux);
}
finProblema = true;
}
}
}

else if (cadenaEntrada.equals("F") || cadenaEntrada.equals("f"))
{
// El usuario prefiere utilizar un fichero de texto para
// indicar los datos.
System.out
      .println("Guarde el fichero de texto con el nombre \"entrada_H-4S-
1C.txt\" en la misma ubicación donde se encuentra");
System.out
      .println("el proyecto y pulse \"C\" para continuar. Pulse \"S\" para
abandonar la aplicación.");
System.out
      .println("Si desea más información acerca del formato del fichero pulse
la tecla \"I\".");
```

```
cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
{
System.out.println(Constantes.INFO_FICHERO_H_4S_1C);
}

else
{
if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else if ("C".equals(cadenaEntrada) || "c".equals(cadenaEntrada))
{
System.out.println("Leyendo el fichero:");
BufferedReader br = new BufferedReader(new FileReader("entrada_H-4S-1C.txt"));
String s;
if ((s = br.readLine()) != null)
{
numElementos = Integer.valueOf(s.trim());
}
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
i = 2;
while (((s = br.readLine()) != null) && !error)
{
if (i >= 2 && i <= (numElementos + 1))
{
puntos.add(Integer.valueOf(s.trim()));
} else if (i == (numElementos + 2))
{
Integer etiquetaAux = Integer.valueOf(s.trim());
for (i = 0; i < numElementos; i++)
{
etiquetas.add(etiquetaAux);
}
} else
{
error = true;
}
i++;
}
}
br.close();
finProblema = true;
}
}
}

else if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
```

```
}
}

if (finProblema)
{
if (puntos.size() != numElementos || error)
{
System.out
      .println("Se ha producido un error en la obtención de los datos: El
número total de puntos y etiquetas no coincide con el indicado.");
salir = true;
}
}
}

if (finProblema && !salir)
{
System.out.println("Los pares (punto , etiqueta) indicados han sido:");
for (i = 0; i < numElementos; i++)
{
System.out.print("par " + (i + 1) + ": (" + puntos.get(i) + " , "
      + etiquetas.get(i) + ")\n");
}

solucion = new Solucion_H_4S_1C(puntos, etiquetas);
solucion.obtenerSolucion();
}

} catch (NumberFormatException e)
{
System.out
      .println("\nSe ha producido una excepción al introducir un valor
incorrecto.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (IOException e)
{
System.out.println("\nSe ha producido una excepción de entrada/salida.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (ArrayIndexOutOfBoundsException e)
{
System.out.println("\nSe ha producido un error.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
}
}
}
```

ProblemasDeEtiquetado/src/lineaHorizontal/h_4s_1c/
Solucion_H_4S_1C.java

```
package lineaHorizontal.h_4s_1c;

import generalidades.ConjuntoListas;
import generalidades.Constantes;
import generalidades.Utilidades;

import java.util.ArrayList;
import java.util.List;

import lineaHorizontal.UtilidadesHorizontal;

/**
 * Clase que contiene los métodos necesarios para solucionar el problema
 * H-4S-1C.
 *
 * @author José Manuel Ponce
 */
public class Solucion_H_4S_1C
{
/**
 * Lista de puntos.
 */
private List<Integer> puntos;

/**
 * Lista con los tamaños (cuadrados) de las etiquetas.
 */
private List<Integer> etiquetas;

/**
 * Lista de posiciones finales de las etiquetas respecto a los puntos.
 */
private List<String> resultado;

/**
 * Lista auxiliar cuyos elementos denotan si están ocupadas por una etiqueta
las
 * posiciones que representan por encima de la línea de entrada.
 */
private List<Integer> lineaSuperior;

/**
 * Lista auxiliar cuyos elementos denotan si están ocupadas por una etiqueta
las
 * posiciones que representan por debajo de la línea de entrada.
 */
private List<Integer> lineaInferior;

/**
 * Número de pares punto-etiqueta.
 */
private int numElementos;

/**
 * Número de divisiones de la línea de entrada.
 */
private int longitudLinea;

/**
```

```
 * Constructor sin parámetros.
 */
public Solucion_H_4S_1C()
{
super();
}

/**
 * Constructor con parámetros.
 *
 * @param puntos
 *           List<Integer> Lista de puntos
 * @param etiquetas
 *           Integer Lista de tamaños (cuadrados) de las etiquetas
 */
public Solucion_H_4S_1C(List<Integer> puntos, List<Integer> etiquetas)
{
this.puntos = puntos;
this.etiquetas = etiquetas;
this.numElementos = puntos.size();
this.resultado = new ArrayList<String>();
this.lineaSuperior = new ArrayList<Integer>();
this.lineaInferior = new ArrayList<Integer>();
}

/**
 * Método que imprime la solución al problema, si la hubiese.
 */
public void obtenerSolucion()
{
ConjuntoListas conjunto;
boolean haySolucion = false;

if (Utilidades.valoresValidos(this.puntos)
                && Utilidades.valoresValidos(this.etiquetas))
{
if (!Utilidades.puntosOrdenados(puntos))
{
conjunto = Utilidades.ordenarListasInteger(this.puntos, this.etiquetas);
this.puntos = conjunto.getListaPuntos();
this.etiquetas = conjunto.getListaEtiquetas();
System.out
                .println("La lista de puntos no está ordenada. Procedemos a
ordenarla.");
System.out.println("Los pares (punto , etiqueta) indicados han sido:");
for (int i = 0; i < numElementos; i++)
{
System.out.print("par " + (i + 1) + ": (" + this.puntos.get(i) + " , "
                + this.etiquetas.get(i) + ")\n");
}
} else
{
System.out
                .println("La lista de puntos está ordenada. Procedemos a la
resolución del problema:");
}
inicializaValores();

int desp;
int indice = 1;
for (int i = 0; i < this.numElementos; i++)
{
haySolucion = false;
```

```
for (desp = 0; (desp <= (this.etiquetas.get(i) + 1)) && !haySolucion; desp++)
{
boolean etiquetado = false;
if (!etiquetado
                && probarEtiquetaArriba(this.puntos.get(i),
this.etiquetas.get(i), desp))
{
haySolucion = true;
etiquetado = true;
aplicarEtiquetaArriba(this.puntos.get(i), this.etiquetas.get(i), desp, indice);
indice++;
if (0 == desp)
{
this.resultado.set(i, Constantes.POS_ARRIBA_IZQUIERDA);
} else if ((this.etiquetas.get(i) + 1) == desp)
{
this.resultado.set(i, Constantes.POS_ARRIBA_DERECHA);
} else
{
this.resultado.set(i, Constantes.POS_ARRIBA);
}
} else if (!etiquetado
                && probarEtiquetaAbajo(this.puntos.get(i),
this.etiquetas.get(i), desp))
{
haySolucion = true;
etiquetado = true;
aplicarEtiquetaAbajo(this.puntos.get(i), this.etiquetas.get(i), desp, indice);
indice++;
if (0 == desp)
{
this.resultado.set(i, Constantes.POS_ABAJO_IZQUIERDA);
} else if ((this.etiquetas.get(i) + 1) == desp)
{
this.resultado.set(i, Constantes.POS_ABAJO_DERECHA);
} else
{
this.resultado.set(i, Constantes.POS_ABAJO);
}
}
}
if (!haySolucion)
{
break;
}
}

System.out.println("Se han obtenido las siguientes posiciones:");
Utilidades.imprimeListaString(resultado);
if (!haySolucion)
{
System.out.println("No se ha podido encontrar una solución válida a ");
System.out.println("este problema con los datos que ha proporcionado.");
System.out
                .println("Gráficamente, la solución parcial obtenida ha sido la
siguiente:");
}
if (haySolucion)
{
System.out
                .println("Gráficamente, la solución del problema se puede
representar como sigue:");
}
```

```
UtilidadesHorizontal.representar(this.longitudLinea, this.puntos,
                this.lineaSuperior, this.lineaInferior);
System.out.println(">> Fin del problema.");
} else
{
System.out
                .println("Los valores de los puntos y etiquetas han de ser
mayores que cero.");
System.out.println("Vuelva a ejecutar el programa si lo desea.");
}
}

/**
 * Método que inicializa las listas "resultado", "lineaSuperior" y
 * "lineaInferior".
 */
public void inicializaValores()
{
this.longitudLinea = this.puntos.get(numElementos - 1) + 1
                + Utilidades.maxValorLista(this.etiquetas);
for (int i = 0; i <= this.longitudLinea; i++)
{
this.lineaSuperior.add(i, null);
this.lineaInferior.add(i, null);
}
for (int i = 0; i < this.numElementos; i++)
{
this.resultado.add(i, "--");
}
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado por el
 * desplazamiento en la línea de arriba.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param desp
 *           Desplazamiento
 * @return boolean Cierto si se puede añadir la etiqueta
 */
public boolean probarEtiquetaArriba(Integer punto, Integer etiqueta,
                Integer desp)
{
boolean dev = true;
for (int i = punto - 1 + desp; (i >= 0) && (i >= punto - etiqueta + desp)
                && dev; i--)
{
if (null != this.lineaSuperior.get(i))
{
dev = false;
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado por el
 * desplazamiento en la línea de abajo.
 *
 * @param punto
```

```
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param desp
 *           Desplazamiento
 * @return boolean Cierto si se puede añadir la etiqueta
 */
public boolean probarEtiquetaAbajo(Integer punto, Integer etiqueta, Integer
desp)
{
boolean dev = true;
int i;
for (i = punto - 1 + desp; (i >= 0) && (i >= punto - etiqueta + desp) && dev;
i--)
{
if (null != this.lineaInferior.get(i))
{
dev = false;
}
}
return dev;
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado por el
 * desplazamiento en la línea de arriba.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param desp
 *           Desplazamiento
 * @param indice
 *           Índice aplicado a la etiqueta que se añade
 * @return boolean Cierto si se añade correctamente la etiqueta
 */
public boolean aplicarEtiquetaArriba(Integer punto, Integer etiqueta,
                Integer desp, int indice)
{
boolean dev = true;
indice = indice % 10;
int i;
for (i = punto - 1 + desp; (i >= 0) && (i >= punto - etiqueta + desp) && dev;
i--)
{
this.lineaSuperior.set(i, indice);
}
return dev;
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado por el
 * desplazamiento en la línea de arriba.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param desp
 *           Desplazamiento
 * @param indice
 *           Índice aplicado a la etiqueta que se añade
```

```
 * @return boolean Cierto si se añade correctamente la etiqueta
 */
public boolean aplicarEtiquetaAbajo(Integer punto, Integer etiqueta,
                Integer desp, int indice)
{
boolean dev = true;
indice = indice % 10;
int i;
for (i = punto - 1 + desp; (i >= 0) && (i >= punto - etiqueta + desp) && dev;
i--)
{
this.lineaInferior.set(i, indice);
}
return dev;
}
}
```

ProblemasDeEtiquetado/src/lineaHorizontal/
SombraHorizontal.java

```
package lineaHorizontal;

/**
 * Clase que implementa el objeto SombraHorizontal, formado por "tk" y "bk".
 *
 * @author José Manuel Ponce
 */
public class SombraHorizontal
{
/**
 * Abscisa del vértice saliente de la etiqueta situada más a la derecha sobre
la
 * línea de entrada.
 */
private int tk;

/**
 * Abscisa del vértice saliente de la etiqueta situada más a la derecha bajo la
 * línea de entrada.
 */
private int bk;

public int getTk()
{
return tk;
}

public void setTk(int tk)
{
this.tk = tk;
}

public int getBk()
{
return bk;
}

public void setBk(int bk)
{
this.bk = bk;
}

/**
 * Constructor sin parámetros.
 */
public SombraHorizontal()
{
super();
}

/**
 * Constructor con parámetros.
 *
 * @param paramTk
 *            Objeto tk
 * @param paramBk
 *            Objeto bk
 */
public SombraHorizontal(int paramTk, int paramBk)
```

```
{
this.tk = paramTk;
this.bk = paramBk;
}
}
```

ProblemasDeEtiquetado/src/lineaHorizontal/
UtilidadesHorizontal.java

```
package lineaHorizontal;

import generalidades.Utilidades;

import java.util.ArrayList;
import java.util.List;

/**
 * Clase que almacena métodos útiles para problemas de etiquetado sobre línea
 * horizontal.
 *
 * @author José Manuel Ponce
 */
public class UtilidadesHorizontal
{
/**
 * Método que refleja en la consola el estado gráfico de las etiquetas.
 *
 * @param longitudLinea
 *           int Longitud de la línea horizontal
 * @param puntos
 *           List<Integer> Lista de puntos
 * @param lineaSuperior
 *           List<Boolean> Línea de etiquetas superior
 * @param lineaInferior
 *           List<Boolean> Línea de etiquetas inferior
 */
public static void representar(int longitudLinea, List<Integer> puntos,
      List<Integer> lineaSuperior, List<Integer> lineaInferior)
{
int i;

// Separador
System.out.println();
for (i = 0; i <= longitudLinea; i++)
{
System.out.print("#");
}
System.out.println();

// Índices
for (i = 0; i <= longitudLinea; i++)
{
System.out.print(Utilidades.mod10(i));
}
System.out.println();

// Línea superior de etiquetas
for (i = 0; i <= longitudLinea; i++)
{
if (null != lineaSuperior.get(i))
{
System.out.print(lineaSuperior.get(i));
} else
{
System.out.print(" ");
}
}
System.out.println();
```

```
// Representación de la línea horizontal con los puntos a etiquetar
int posicion = 0;
int punto = -1;
int anterior = -1;
for (i = 0; i < puntos.size(); i++)
{
punto = puntos.get(i);
while ((posicion <= longitudLinea) && (posicion < punto))
{
System.out.print("-");
posicion++;
}
if (punto != anterior)
{
System.out.print(":");
posicion++;
}
anterior = punto;
}
posicion = punto + 1;
while (posicion <= longitudLinea)
{
System.out.print("-");
posicion++;
}
System.out.println();

// Línea inferior de etiquetas
for (i = 0; i <= longitudLinea; i++)
{
if (null != lineaInferior.get(i))
{
System.out.print(lineaInferior.get(i));
} else
{
System.out.print(" ");
}
}
System.out.println();

// Índices
for (i = 0; i <= longitudLinea; i++)
{
System.out.print(Utilidades.mod10(i));
}
System.out.println();

// Separador
for (i = 0; i <= longitudLinea; i++)
{
System.out.print("#");
}
System.out.println();
System.out.println();
}

/**
 * Método que devuelve cierto si la primera sombra es menor que la segunda.
 *
 * @param s1
 *           Primera sombra
 * @param s2
```

```
 *           Segunda sombra
 * @return boolean Cierto si la primera sombra es menor que la segunda
 */
public static boolean sombraMenorIgualQue(SombraHorizontal s1,
      SombraHorizontal s2)
{
boolean dev = false;
if (((s1.getTk() <= s2.getTk()) && (s1.getBk() <= s2.getBk()))
      || ((s1.getTk() <= s2.getBk()) && (s1.getBk() <= s2.getTk())))
{
dev = true;
}
return dev;
}

/**
 * Método que calcula cuál de las 3 sombras indicadas es la menor.
 *
 * @param s1
 *           Primera sombra
 * @param s2
 *           Segunda sombra
 * @param s3
 *           Tercera sombra
 * @return Posición de la sombra menor. Devuelve -1 si no hay ninguna sombra
 *         válida.
 */
public static int hallarSombraMenor3(SombraHorizontal s1, SombraHorizontal s2,
      SombraHorizontal s3)
{
int dev = -1;
if (sombraMenorIgualQue(s1, s2))
{
if (sombraMenorIgualQue(s1, s3))
{
dev = 1;
} else
{
dev = 3;
}
} else
{
if (sombraMenorIgualQue(s2, s3))
{
dev = 2;
} else
{
dev = 3;
}
}
return dev;
}

/**
 * Método que calcula cuál de las 4 sombras indicadas es la menor.
 *
 * @param s1
 *           Primera sombra
 * @param s2
 *           Segunda sombra
 * @param s3
 *           Tercera sombra
 * @param s4
```

```
 *           Cuarta sombra
 * @return Posición de la sombra menor. Devuelve -1 si no hay ninguna sombra
 *         válida.
 */
public static int hallarSombraMenor4(SombraHorizontal s1, SombraHorizontal s2,
      SombraHorizontal s3, SombraHorizontal s4)
{
int dev = -1;
int menor;
int indice;
List<SombraHorizontal> sombras = new ArrayList<SombraHorizontal>();
List<Boolean> sombrasValidas = new ArrayList<Boolean>();
for (int i = 0; i < 4; i++)
{
sombrasValidas.add(false);
}
// Rellenamos la lista con sombras != null
if (null != s1)
{
sombras.add(s1);
sombrasValidas.set(0, true);
}
if (null != s2)
{
sombras.add(s2);
sombrasValidas.set(1, true);
}
if (null != s3)
{
sombras.add(s3);
sombrasValidas.set(2, true);
}
if (null != s4)
{
sombras.add(s4);
sombrasValidas.set(3, true);
}
// Hay 4 sombras válidas
if (4 == sombras.size())
{
System.out.println("Hay 4 sombras válidas:");
if (UtilidadesHorizontal.sombraMenorIgualQue(s1, s2)
      && UtilidadesHorizontal.sombraMenorIgualQue(s1, s3)
      && UtilidadesHorizontal.sombraMenorIgualQue(s1, s4))
{
dev = 1;
} else if (UtilidadesHorizontal.sombraMenorIgualQue(s2, s1)
      && UtilidadesHorizontal.sombraMenorIgualQue(s2, s3)
      && UtilidadesHorizontal.sombraMenorIgualQue(s2, s4))
{
dev = 2;
} else if (UtilidadesHorizontal.sombraMenorIgualQue(s3, s1)
      && UtilidadesHorizontal.sombraMenorIgualQue(s3, s2)
      && UtilidadesHorizontal.sombraMenorIgualQue(s3, s4))
{
dev = 3;
} else if (UtilidadesHorizontal.sombraMenorIgualQue(s4, s1)
      && UtilidadesHorizontal.sombraMenorIgualQue(s4, s2)
      && UtilidadesHorizontal.sombraMenorIgualQue(s4, s3))
{
dev = 4;
}
} else if (3 == sombras.size()) // Hay 3 sombras válidas
```

```
{
System.out.println("Hay 3 sombras válidas:");
menor = hallarSombraMenor3(sombras.get(0), sombras.get(1), sombras.get(2));
indice = 1;
for (int i = 0; i < 4; i++)
{
if (sombrasValidas.get(i))
{
if (indice == menor)
{
dev = i + 1;
break;
}
indice++;
}
}
} else if (2 == sombras.size()) // Hay 2 sombras válidas
{
System.out.println("Hay 2 sombras válidas:");
if (sombraMenorIgualQue(sombras.get(0), sombras.get(1)))
{
menor = 1;
} else
{
menor = 2;
}
indice = 1;
for (int i = 0; i < 4; i++)
{
if (sombrasValidas.get(i))
{
if (indice == menor)
{
dev = i + 1;
break;
}
i++;
}
}
} else if (1 == sombras.size()) // Hay sólo 1 sombra válida
{
System.out.println("Hay 1 sombras válidas:");
for (int i = 0; i < 4; i++)
{
if (sombrasValidas.get(i))
{
dev = i + 1;
}
}
} else if (0 == sombras.size()) // No hay ninguna sombra válida
{
System.out.println("No hay sombras válidas:");
}
return dev;
}
}
```

ProblemasDeEtiquetado/src/lineaOblicua/o1r_4p/
Main_O1R_4P.java

```
package lineaOblicua.o1r_4p;

import generalidades.Constantes;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

/**
 * Clase principal que resuelve el problema O1R-4P.
 *
 * @author José Manuel Ponce
 */
public class Main_O1R_4P
{
/**
 * Instancia de un objeto Solucion_O1R_4P.
 */
private static Solucion_O1R_4P solucion;

/**
 * Código a ejecutar.
 *
 * @param args
 *           Argumentos
 */
public static void main(String[] args)
{
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
System.out.println(Constantes.CABECERA_O1R_4P);
System.out.println();

boolean salir = false, error = false, finProblema = false;
int i = 0, numElementos = 0;
String cadenaEntrada = "";
List<Integer> puntos = new ArrayList<Integer>();
List<Integer> etiquetas = new ArrayList<Integer>();

try
{
while (!salir && !finProblema)
{
System.out
      .println("Pulse la tecla \"T\" para introducir los datos desde el
teclado.");
System.out
      .println("Pulse la tecla \"F\" si desea que la aplicación obtenga los
datos a partir de un fichero de texto.");
System.out
      .println("Pulse la tecla \"I\" si desea información adicional acerca de
este problema.");
System.out.println("Pulse la tecla \"S\" para salir de la aplicación.");

cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
```

```
{
System.out.println(Constantes.INFO_O1R_4P);
}

else
{
if ("T".equals(cadenaEntrada) || "t".equals(cadenaEntrada))
{
// El usuario prefiere introducir los datos desde el
// teclado.
System.out
      .println("Introduzca el número de elementos. Pulse \"S\" para salir.");
cadenaEntrada = in.readLine();

if (cadenaEntrada.equals("S") || cadenaEntrada.equals("s"))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else
{
numElementos = Integer.valueOf(cadenaEntrada).intValue();
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
etiquetas = new ArrayList<Integer>(numElementos);
System.out.println("Ha indicado " + numElementos + " elementos. ");
i = 1;
while (i <= numElementos)
{
System.out.println("Introduzca el punto " + i + ":");
puntos.add(Integer.valueOf(in.readLine()).intValue());
i++;
}
i = 1;
while (i <= numElementos)
{
System.out.println("Introduzca el tamaño de la etiqueta " + i + ":");
etiquetas.add(Integer.valueOf(in.readLine()).intValue());
i++;
}
finProblema = true;
}
}
}

else if (cadenaEntrada.equals("F") || cadenaEntrada.equals("f"))
{
// El usuario prefiere utilizar un fichero de texto para
// indicar los datos.
System.out
      .println("Guarde el fichero de texto con el nombre \"entrada_O1R-4P.txt\"
en la misma ubicación donde se encuentra");
System.out
      .println("el proyecto y pulse \"C\" para continuar. Pulse \"S\" para
abandonar la aplicación.");
System.out
      .println("Si desea más información acerca del formato del fichero pulse
```

```
la tecla \"I\".");

cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
{
System.out.println(Constantes.INFO_FICHERO_O1R_4P);
}

else
{
if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else if ("C".equals(cadenaEntrada) || "c".equals(cadenaEntrada))
{
System.out.println("Leyendo el fichero:");
BufferedReader br = new BufferedReader(new FileReader("entrada_O1R-4P.txt"));
String s;
if ((s = br.readLine()) != null)
{
numElementos = Integer.valueOf(s.trim());
}
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
etiquetas = new ArrayList<Integer>(numElementos);
i = 2;
while (((s = br.readLine()) != null) && !error)
{
if (i >= 2 && i <= (numElementos + 1))
{
puntos.add(Integer.valueOf(s.trim()));
} else if (i > (numElementos + 1) && i <= ((2 * numElementos) + 1))
{
etiquetas.add(Integer.valueOf(s.trim()));
} else
{
error = true;
}
i++;
}
}
br.close();
finProblema = true;
}
}
}

else if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
}
}
```

```
if (finProblema)
{
if ((puntos.size() != etiquetas.size()) || (puntos.size() != numElementos)
      || (numElementos != etiquetas.size()) || error)
{
System.out
      .println("Se ha producido un error en la obtención de los datos: El
número total de puntos y etiquetas no coincide con el indicado.");
salir = true;
}
}
}

if (finProblema && !salir)
{
System.out.println("Los pares (punto , etiqueta) indicados han sido:");
for (i = 0; i < numElementos; i++)
{
System.out.print("par " + (i + 1) + ": (" + puntos.get(i) + " , "
      + etiquetas.get(i) + ")\n");
}

solucion = new Solucion_O1R_4P(puntos, etiquetas);
solucion.obtenerSolucion();
}

} catch (NumberFormatException e)
{
System.out
      .println("\nSe ha producido una excepción al introducir un valor
incorrecto.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (IOException e)
{
System.out.println("\nSe ha producido una excepción de entrada/salida.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (ArrayIndexOutOfBoundsException e)
{
System.out.println("\nSe ha producido un error.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
}
}
}
```

ProblemasDeEtiquetado/src/lineaOblicua/o1r_4p/
Solucion_O1R_4P.java

```
package lineaOblicua.o1r_4p;

import generalidades.ConjuntoListas;
import generalidades.Constantes;
import generalidades.Utilidades;

import java.util.ArrayList;
import java.util.List;

import lineaOblicua.UtilidadesOblicua;

/**
 * Clase que contiene los métodos necesarios para solucionar el problema O1R-
4P.
 *
 * @author José Manuel Ponce
 */
public class Solucion_O1R_4P
{
/**
 * Altura de las etiquetas.
 */
private static final int ALTURA_ETIQUETA = 3;

/**
 * Lista de puntos.
 */
private List<Integer> puntos;

/**
 * Lista de etiquetas.
 */
private List<Integer> etiquetas;

/**
 * Lista de posiciones finales de las etiquetas respecto a los puntos.
 */
private List<String> resultado;

/**
 * Número de pares punto-etiqueta.
 */
private int numElementos;

/**
 * Número de divisiones de la línea de entrada.
 */
private int longitudLinea;

/**
 * Objeto que almacena la posición de las etiquetas que ya se han añadido.
 */
private boolean[][] espacioEtiquetas;

/**
 * Constructor sin parámetros.
 */
public Solucion_O1R_4P()
{
```

```
super();
}

/**
 * Constructor con parámetros.
 *
 * @param puntos
 *           List<Integer> Lista de puntos
 * @param etiquetas
 *           List<Integer> Lista de etiquetas
 */
public Solucion_O1R_4P(List<Integer> puntos, List<Integer> etiquetas)
{
this.puntos = puntos;
this.etiquetas = etiquetas;
this.numElementos = puntos.size();
this.resultado = new ArrayList<String>();
}

/**
 * Método que imprime la solución al problema, si la hubiese.
 */
public void obtenerSolucion()
{
boolean haySolucion = true;

if ((Utilidades.valoresValidos(this.puntos))
      && (Utilidades.valoresValidos(this.etiquetas)))
{
if (!Utilidades.puntosOrdenados(puntos))
{
ConjuntoListas conjunto = Utilidades.ordenarListasInteger(this.puntos,
      this.etiquetas);
this.puntos = conjunto.getListaPuntos();
this.etiquetas = conjunto.getListaEtiquetas();
System.out
      .println("La lista de puntos no está ordenada. Procedemos a ordenarla.");
System.out.println("Los pares (punto , etiqueta) indicados han sido:");
for (int i = 0; i < numElementos; i++)
{
System.out.print("par " + (i + 1) + ": (" + this.puntos.get(i) + " , "
      + this.etiquetas.get(i) + ")\n");
}
} else
{
System.out
      .println("La lista de puntos está ordenada. Procedemos a la resolución
del problema:");
}
inicializaValores();

for (int i = 0; (i < this.numElementos) && haySolucion; i++)
{
if (this.probarEtiquetaBL(this.puntos.get(i), this.etiquetas.get(i)))
{
this.aplicarEtiquetaBL(this.puntos.get(i), this.etiquetas.get(i));
this.resultado.set(i, Constantes.POS_ABAJO_IZQUIERDA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else if (this.probarEtiquetaTL(this.puntos.get(i), this.etiquetas.get(i)))
{
this.aplicarEtiquetaTL(this.puntos.get(i), this.etiquetas.get(i));
```

```
this.resultado.set(i, Constantes.POS_ARRIBA_IZQUIERDA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else if (this.probarEtiquetaBR(this.puntos.get(i), this.etiquetas.get(i)))
{
this.aplicarEtiquetaBR(this.puntos.get(i), this.etiquetas.get(i));
this.resultado.set(i, Constantes.POS_ABAJO_DERECHA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else if (this.probarEtiquetaTR(this.puntos.get(i), this.etiquetas.get(i)))
{
this.aplicarEtiquetaTR(this.puntos.get(i), this.etiquetas.get(i));
this.resultado.set(i, Constantes.POS_ARRIBA_DERECHA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else
{
haySolucion = false;
}
}

System.out.println("\nSe han obtenido las siguientes posiciones:");
Utilidades.imprimeListaString(resultado);
if (!haySolucion)
{
System.out.println("No se ha podido encontrar una solución válida a ");
System.out.println("este problema con los datos que ha proporcionado.");
System.out
      .println("Gráficamente, la solución parcial obtenida ha sido la
siguiente:");
}
if (haySolucion)
{
System.out
      .println("\nGráficamente, la solución del problema se puede representar
como sigue:");
}
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
System.out.println(">> Fin del problema.");
} else
{
System.out
      .println("Los valores de los puntos y etiquetas han de ser mayores que
cero.");
System.out.println("Vuelva a ejecutar el programa si lo desea.");
}
}

/**
 * Método que inicializa las listas "resultado", "lineaSuperior" y
 * "lineaInferior".
 */
public void inicializaValores()
{
this.longitudLinea = this.puntos.get(numElementos - 1) + 2
      + Utilidades.maxValorLista(this.etiquetas);
this.espacioEtiquetas = new boolean[this.longitudLinea][this.longitudLinea];
for (int i = 0; i < this.longitudLinea; i++)
{
for (int j = 0; j < this.longitudLinea; j++)
```

```
{
this.espacioEtiquetas[i][j] = false;
}
}
for (int i = 0; i < this.numElementos; i++)
{
this.resultado.add(i, "--");
}
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición TL para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaTL(Integer punto, Integer etiqueta)
{
boolean dev = true;
int x;
int y = punto;
for (int desp = 0; (desp < ALTURA_ETIQUETA) && (y + desp < this.longitudLinea)
      && dev; desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta) && dev; x--)
{
if (this.espacioEtiquetas[y + desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición BL para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaBL(Integer punto, Integer etiqueta)
{
boolean dev = true;
int x;
int y = punto - 1;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0) && dev; desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta) && dev; x--)
{
if (this.espacioEtiquetas[y - desp][x])
{
dev = false;
```

```
}
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición TR para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaTR(Integer punto, Integer etiqueta)
{
boolean dev = true;
int x;
int y = punto;
for (int desp = 0; (desp < ALTURA_ETIQUETA) && (y + desp < this.longitudLinea)
      && dev; desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta) && dev; x++)
{
if (this.espacioEtiquetas[y + desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición BR para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaBR(Integer punto, Integer etiqueta)
{
boolean dev = true;
int x;
int y = punto - 1;
for (int desp = 0; (desp < ALTURA_ETIQUETA) && (y - desp >= 0) && dev; desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta) && dev; x++)
{
if (this.espacioEtiquetas[y - desp][x])
{
dev = false;
}
}
}
return dev;
```

```
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición TL.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 */
public void aplicarEtiquetaTL(Integer punto, Integer etiqueta)
{
int x;
int y = punto;
for (int desp = 0; (desp < ALTURA_ETIQUETA) && (y + desp < this.longitudLinea);
desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta); x--)
{
this.espacioEtiquetas[y + desp][x] = true;
}
}
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición BL.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 */
public void aplicarEtiquetaBL(Integer punto, Integer etiqueta)
{
int x;
int y = punto - 1;
for (int desp = 0; (desp < ALTURA_ETIQUETA) && (y - desp >= 0); desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta); x--)
{
this.espacioEtiquetas[y - desp][x] = true;
}
}
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición TR.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 */
public void aplicarEtiquetaTR(Integer punto, Integer etiqueta)
{
int x;
int y = punto;
for (int desp = 0; (desp < ALTURA_ETIQUETA) && (y + desp < this.longitudLinea);
desp++)
{
```

```
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta); x++)
{
this.espacioEtiquetas[y + desp][x] = true;
}
}
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición BR.
 *
 * @param punto
 *            Punto a etiquetar
 * @param etiqueta
 *            Tamaño de la etiqueta que se va a añadir
 */
public void aplicarEtiquetaBR(Integer punto, Integer etiqueta)
{
int x;
int y = punto - 1;
for (int desp = 0; (desp < ALTURA_ETIQUETA) && (y - desp >= 0); desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta); x++)
{
this.espacioEtiquetas[y - desp][x] = true;
}
}
}
}
```

ProblemasDeEtiquetado/src/lineaOblicua/oc_4p/
Main_OC_4P.java

```
package lineaOblicua.oc_4p;

import generalidades.Constantes;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

/**
 * Clase principal que resuelve el problema OC-4P.
 *
 * @author José Manuel Ponce
 */
public class Main_OC_4P
{
/**
 * Instancia de un objeto Solucion_OC_4P.
 */
private static Solucion_OC_4P solucion;

/**
 * Código a ejecutar.
 *
 * @param args
 *           Argumentos
 */
public static void main(String[] args)
{
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
System.out.println(Constantes.CABECERA_OC_4P);
System.out.println();

boolean salir = false, error = false, finProblema = false;
int i = 0, numElementos = 0, tamanyoEtiqueta = 0;
String cadenaEntrada = "";
List<Integer> puntos = new ArrayList<Integer>();
List<Integer> etiquetas = new ArrayList<Integer>();

try
{
while (!salir && !finProblema)
{
System.out
      .println("Pulse la tecla \"T\" para introducir los datos desde el
teclado.");
System.out
      .println("Pulse la tecla \"F\" si desea que la aplicación obtenga los
datos a partir de un fichero de texto.");
System.out
      .println("Pulse la tecla \"I\" si desea información adicional acerca de
este problema.");
System.out.println("Pulse la tecla \"S\" para salir de la aplicación.");

cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
```

```
{
System.out.println(Constantes.INFO_OC_4P);
}

else
{
if ("T".equals(cadenaEntrada) || "t".equals(cadenaEntrada))
{
// El usuario prefiere introducir los datos desde el
// teclado.
System.out
      .println("Introduzca el número de elementos. Pulse \"S\" para salir.");
cadenaEntrada = in.readLine();

if (cadenaEntrada.equals("S") || cadenaEntrada.equals("s"))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else
{
numElementos = Integer.valueOf(cadenaEntrada).intValue();
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
etiquetas = new ArrayList<Integer>(numElementos);
System.out.println("Ha indicado " + numElementos + " elementos. ");
i = 1;
while (i <= numElementos)
{
System.out.println("Introduzca el punto " + i + ":");
puntos.add(Integer.valueOf(in.readLine()).intValue());
i++;
}
System.out.println("Introduzca el tamaño (cuadrado) de las etiquetas:");
tamanyoEtiqueta = Integer.valueOf(in.readLine()).intValue();
i = 1;
while (i <= numElementos)
{
etiquetas.add(tamanyoEtiqueta);
i++;
}
finProblema = true;
}
}
}

else if (cadenaEntrada.equals("F") || cadenaEntrada.equals("f"))
{
// El usuario prefiere utilizar un fichero de texto para
// indicar los datos.
System.out
      .println("Guarde el fichero de texto con el nombre \"entrada_OC-4P.txt\"
en la misma ubicación donde se encuentra");
System.out
      .println("el proyecto y pulse \"C\" para continuar. Pulse \"S\" para
abandonar la aplicación.");
System.out
```

```
      .println("Si desea más información acerca del formato del fichero pulse
la tecla \"I\".");

cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
{
System.out.println(Constantes.INFO_FICHERO_OC_4P);
}

else
{
if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else if ("C".equals(cadenaEntrada) || "c".equals(cadenaEntrada))
{
System.out.println("Leyendo el fichero:");
BufferedReader br = new BufferedReader(new FileReader("entrada_OC-4P.txt"));
String s;
if ((s = br.readLine()) != null)
{
numElementos = Integer.valueOf(s.trim());
}
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
etiquetas = new ArrayList<Integer>(numElementos);
i = 2;
while (((s = br.readLine()) != null) && !error)
{
if (i >= 2 && i <= (numElementos + 1))
{
puntos.add(Integer.valueOf(s.trim()));
} else if (i == numElementos + 2)
{
tamanyoEtiqueta = Integer.valueOf(s.trim());
} else
{
error = true;
}
i++;
}
for (i = 0; i < numElementos; i++)
{
etiquetas.add(tamanyoEtiqueta);
}
}
br.close();
finProblema = true;
}
}
}

else if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
```

```
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
}
}

if (finProblema)
{
if ((puntos.size() != etiquetas.size()) || (puntos.size() != numElementos)
      || (numElementos != etiquetas.size()) || error)
{
System.out
      .println("Se ha producido un error en la obtención de los datos: El
número total de puntos y etiquetas no coincide con el indicado.");
salir = true;
}
}
}

if (finProblema && !salir)
{
System.out.println("Los pares (punto , etiqueta) indicados han sido:");
for (i = 0; i < numElementos; i++)
{
System.out.print("par " + (i + 1) + ": (" + puntos.get(i) + " , "
      + etiquetas.get(i) + ")\n");
}

solucion = new Solucion_OC_4P(puntos, etiquetas);
solucion.obtenerSolucion();
}

} catch (NumberFormatException e)
{
System.out
      .println("\nSe ha producido una excepción al introducir un valor
incorrecto: NumberFormatException.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (IOException e)
{
System.out
      .println("\nSe ha producido una excepción de entrada/salida:
IOException.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (ArrayIndexOutOfBoundsException e)
{
System.out
      .println("\nSe ha producido un error: ArrayIndexOutOfBoundsException.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
}
}
}
```

ProblemasDeEtiquetado/src/lineaOblicua/oc_4p/
Solucion_OC_4P.java

```
package lineaOblicua.oc_4p;

import generalidades.ConjuntoListas;
import generalidades.Constantes;
import generalidades.Utilidades;

import java.util.ArrayList;
import java.util.List;

import lineaOblicua.UtilidadesOblicua;

/**
 * Clase que contiene los métodos necesarios para solucionar el problema OC-4P.
 *
 * @author José Manuel Ponce
 */
public class Solucion_OC_4P
{
/**
 * Lista de puntos.
 */
private List<Integer> puntos;

/**
 * Lista de etiquetas.
 */
private List<Integer> etiquetas;

/**
 * Lista de posiciones finales de las etiquetas respecto a los puntos.
 */
private List<String> resultado;

/**
 * Número de pares punto-etiqueta.
 */
private int numElementos;

/**
 * Número de divisiones de la línea de entrada.
 */
private int longitudLinea;

/**
 * Objeto que almacena la posición de las etiquetas que ya se han añadido.
 */
private boolean[][] espacioEtiquetas;

/**
 * Constructor sin parámetros.
 */
public Solucion_OC_4P()
{
super();
}

/**
 * Constructor con parámetros.
 *
```

```
 * @param puntos
 *           List<Integer> Lista de puntos
 * @param etiquetas
 *           List<Integer> Lista de etiquetas
 */
public Solucion_OC_4P(List<Integer> puntos, List<Integer> etiquetas)
{
this.puntos = puntos;
this.etiquetas = etiquetas;
this.numElementos = puntos.size();
this.resultado = new ArrayList<String>();
}

/**
 * Método que imprime la solución al problema, si la hubiese.
 */
public void obtenerSolucion()
{
boolean haySolucion = true;

if ((Utilidades.valoresValidos(this.puntos))
      && (Utilidades.valoresValidos(this.etiquetas)))
{
if (!Utilidades.puntosOrdenados(puntos))
{
ConjuntoListas conjunto = Utilidades.ordenarListasInteger(this.puntos,
      this.etiquetas);
this.puntos = conjunto.getListaPuntos();
this.etiquetas = conjunto.getListaEtiquetas();
System.out
      .println("La lista de puntos no está ordenada. Procedemos a ordenarla.");
System.out.println("Los pares (punto , etiqueta) indicados han sido:");
for (int i = 0; i < numElementos; i++)
{
System.out.print("par " + (i + 1) + ": (" + this.puntos.get(i) + " , "
      + this.etiquetas.get(i) + ")\n");
}
} else
{
System.out
      .println("La lista de puntos está ordenada. Procedemos a la resolución
del problema:");
}
inicializaValores();

for (int i = 0; (i < this.numElementos) && haySolucion; i++)
{
if (this.probarEtiquetaBL(this.puntos.get(i), this.etiquetas.get(i)))
{
this.aplicarEtiquetaBL(this.puntos.get(i), this.etiquetas.get(i));
this.resultado.set(i, Constantes.POS_ABAJO_IZQUIERDA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else if (this.probarEtiquetaTL(this.puntos.get(i), this.etiquetas.get(i)))
{
this.aplicarEtiquetaTL(this.puntos.get(i), this.etiquetas.get(i));
this.resultado.set(i, Constantes.POS_ARRIBA_IZQUIERDA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else if (this.probarEtiquetaBR(this.puntos.get(i), this.etiquetas.get(i)))
{
```

```
this.aplicarEtiquetaBR(this.puntos.get(i), this.etiquetas.get(i));
this.resultado.set(i, Constantes.POS_ABAJO_DERECHA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else if (this.probarEtiquetaTR(this.puntos.get(i), this.etiquetas.get(i)))
{
this.aplicarEtiquetaTR(this.puntos.get(i), this.etiquetas.get(i));
this.resultado.set(i, Constantes.POS_ARRIBA_DERECHA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else
{
haySolucion = false;
}
}

System.out.println("\nSe han obtenido las siguientes posiciones:");
Utilidades.imprimeListaString(resultado);
if (!haySolucion)
{
System.out.println("No se ha podido encontrar una solución válida a ");
System.out.println("este problema con los datos que ha proporcionado.");
System.out
      .println("Gráficamente, la solución parcial obtenida ha sido la
siguiente:");
}
if (haySolucion)
{
System.out
      .println("\nGráficamente, la solución del problema se puede representar
como sigue:");
}
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
System.out.println(">> Fin del problema.");
} else
{
System.out
      .println("Los valores de los puntos y etiquetas han de ser mayores que
cero.");
System.out.println("Vuelva a ejecutar el programa si lo desea.");
}
}

/**
 * Método que inicializa las listas "resultado", "lineaSuperior" y
 * "lineaInferior".
 */
public void inicializaValores()
{
this.longitudLinea = this.puntos.get(numElementos - 1) + 2
      + Utilidades.maxValorLista(this.etiquetas);
this.espacioEtiquetas = new boolean[this.longitudLinea][this.longitudLinea];
for (int i = 0; i < this.longitudLinea; i++)
{
for (int j = 0; j < this.longitudLinea; j++)
{
this.espacioEtiquetas[i][j] = false;
}
}
for (int i = 0; i < this.numElementos; i++)
{
```

```
this.resultado.add(i, "--");
}
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición TL para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaTL(Integer punto, Integer etiqueta)
{
boolean dev = true;
int x;
int y = punto;
for (int desp = 0; (desp < etiqueta) && (y + desp < this.longitudLinea) && dev;
desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta) && dev; x--)
{
if (this.espacioEtiquetas[y + desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición BL para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaBL(Integer punto, Integer etiqueta)
{
boolean dev = true;
int x;
int y = punto - 1;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0) && dev; desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta) && dev; x--)
{
if (this.espacioEtiquetas[y - desp][x])
{
dev = false;
}
}
}
return dev;
}
```

```
/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición TR para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaTR(Integer punto, Integer etiqueta)
{
boolean dev = true;
int x;
int y = punto;
for (int desp = 0; (desp < etiqueta) && (y + desp < this.longitudLinea) && dev;
desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta) && dev; x++)
{
if (this.espacioEtiquetas[y + desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición BR para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaBR(Integer punto, Integer etiqueta)
{
boolean dev = true;
int x;
int y = punto - 1;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0) && dev; desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta) && dev; x++)
{
if (this.espacioEtiquetas[y - desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición TL.
 *
```

```
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 */
public void aplicarEtiquetaTL(Integer punto, Integer etiqueta)
{
int x;
int y = punto;
for (int desp = 0; (desp < etiqueta) && (y + desp < this.longitudLinea);
desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta); x--)
{
this.espacioEtiquetas[y + desp][x] = true;
}
}
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición BL.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 */
public void aplicarEtiquetaBL(Integer punto, Integer etiqueta)
{
int x;
int y = punto - 1;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0); desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta); x--)
{
this.espacioEtiquetas[y - desp][x] = true;
}
}
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición TR.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 */
public void aplicarEtiquetaTR(Integer punto, Integer etiqueta)
{
int x;
int y = punto;
for (int desp = 0; (desp < etiqueta) && (y + desp < this.longitudLinea);
desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta); x++)
{
this.espacioEtiquetas[y + desp][x] = true;
}
}
}
```

```
/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición BR.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 */
public void aplicarEtiquetaBR(Integer punto, Integer etiqueta)
{
int x;
int y = punto - 1;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0); desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta); x++)
{
this.espacioEtiquetas[y - desp][x] = true;
}
}
}
}
```

ProblemasDeEtiquetado/src/lineaOblicua/oc_4s/
Main_OC_4S.java

```
package lineaOblicua.oc_4s;

import generalidades.Constantes;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

/**
 * Clase principal que resuelve el problema OC-4S.
 *
 * @author José Manuel Ponce
 */
public class Main_OC_4S
{
/**
 * Instancia de un objeto Solucion_OC_4S.
 */
private static Solucion_OC_4S solucion;

/**
 * Código a ejecutar.
 *
 * @param args
 *           Argumentos
 */
public static void main(String[] args)
{
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
System.out.println(Constantes.CABECERA_OC_4S);
System.out.println();

boolean salir = false, error = false, finProblema = false;
int i = 0, numElementos = 0, tamanyoEtiqueta = 0;
String cadenaEntrada = "";
List<Integer> puntos = new ArrayList<Integer>();
List<Integer> etiquetas = new ArrayList<Integer>();

try
{
while (!salir && !finProblema)
{
System.out
      .println("Pulse la tecla \"T\" para introducir los datos desde el
teclado.");
System.out
      .println("Pulse la tecla \"F\" si desea que la aplicación obtenga los
datos a partir de un fichero de texto.");
System.out
      .println("Pulse la tecla \"I\" si desea información adicional acerca de
este problema.");
System.out.println("Pulse la tecla \"S\" para salir de la aplicación.");

cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
```

```
{
System.out.println(Constantes.INFO_OC_4S);
}

else
{
if ("T".equals(cadenaEntrada) || "t".equals(cadenaEntrada))
{
// El usuario prefiere introducir los datos desde el
// teclado.
System.out
      .println("Introduzca el número de elementos. Pulse \"S\" para salir.");
cadenaEntrada = in.readLine();

if (cadenaEntrada.equals("S") || cadenaEntrada.equals("s"))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else
{
numElementos = Integer.valueOf(cadenaEntrada).intValue();
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
etiquetas = new ArrayList<Integer>(numElementos);
System.out.println("Ha indicado " + numElementos + " elementos. ");
i = 1;
while (i <= numElementos)
{
System.out.println("Introduzca el punto " + i + ":");
puntos.add(Integer.valueOf(in.readLine()).intValue());
i++;
}
System.out.println("Introduzca el tamaño (cuadrado) de las etiquetas:");
tamanyoEtiqueta = Integer.valueOf(in.readLine()).intValue();
i = 1;
while (i <= numElementos)
{
etiquetas.add(tamanyoEtiqueta);
i++;
}
finProblema = true;
}
}
}

else if (cadenaEntrada.equals("F") || cadenaEntrada.equals("f"))
{
// El usuario prefiere utilizar un fichero de texto para
// indicar los datos.
System.out
      .println("Guarde el fichero de texto con el nombre \"entrada_OC-4S.txt\"
en la misma ubicación donde se encuentra");
System.out
      .println("el proyecto y pulse \"C\" para continuar. Pulse \"S\" para
abandonar la aplicación.");
System.out
```

```
      .println("Si desea más información acerca del formato del fichero pulse
la tecla \"I\".");

cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
{
System.out.println(Constantes.INFO_FICHERO_OC_4S);
}

else
{
if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else if ("C".equals(cadenaEntrada) || "c".equals(cadenaEntrada))
{
System.out.println("Leyendo el fichero:");
BufferedReader br = new BufferedReader(new FileReader("entrada_OC-4S.txt"));
String s;
if ((s = br.readLine()) != null)
{
numElementos = Integer.valueOf(s.trim());
}
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
etiquetas = new ArrayList<Integer>(numElementos);
i = 2;
while (((s = br.readLine()) != null) && !error)
{
if (i >= 2 && i <= (numElementos + 1))
{
puntos.add(Integer.valueOf(s.trim()));
} else if (i == numElementos + 2)
{
tamanyoEtiqueta = Integer.valueOf(s.trim());
} else
{
error = true;
}
i++;
}
for (i = 0; i < numElementos; i++)
{
etiquetas.add(tamanyoEtiqueta);
}
}
br.close();
finProblema = true;
}
}
}

else if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
```

```
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
}
}

if (finProblema)
{
if ((puntos.size() != etiquetas.size()) || (puntos.size() != numElementos)
      || (numElementos != etiquetas.size()) || error)
{
System.out
      .println("Se ha producido un error en la obtención de los datos: El
número total de puntos y etiquetas no coincide con el indicado.");
salir = true;
}
}
}

if (finProblema && !salir)
{
System.out.println("Los pares (punto , etiqueta) indicados han sido:");
for (i = 0; i < numElementos; i++)
{
System.out.print("par " + (i + 1) + ": (" + puntos.get(i) + " , "
      + etiquetas.get(i) + ")\n");
}

solucion = new Solucion_OC_4S(puntos, etiquetas);
solucion.obtenerSolucion();
}

} catch (NumberFormatException e)
{
System.out
      .println("\nSe ha producido una excepción al introducir un valor
incorrecto: NumberFormatException.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (IOException e)
{
System.out
      .println("\nSe ha producido una excepción de entrada/salida:
IOException.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (ArrayIndexOutOfBoundsException e)
{
System.out
      .println("\nSe ha producido un error: ArrayIndexOutOfBoundsException.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
}
}
}
```

ProblemasDeEtiquetado/src/lineaOblicua/oc_4s/
Solucion_OC_4S.java

```
package lineaOblicua.oc_4s;

import generalidades.ConjuntoListas;
import generalidades.Constantes;
import generalidades.Utilidades;

import java.util.ArrayList;
import java.util.List;

import lineaOblicua.UtilidadesOblicua;

/**
 * Clase que contiene los métodos necesarios para solucionar el problema OC-4S.
 *
 * @author José Manuel Ponce
 */
public class Solucion_OC_4S
{
/**
 * Lista de puntos.
 */
private List<Integer> puntos;

/**
 * Lista de etiquetas.
 */
private List<Integer> etiquetas;

/**
 * Lista de posiciones finales de las etiquetas respecto a los puntos.
 */
private List<String> resultado;

/**
 * Número de pares punto-etiqueta.
 */
private int numElementos;

/**
 * Número de divisiones de la línea de entrada.
 */
private int longitudLinea;

/**
 * Objeto que almacena la posición de las etiquetas que ya se han añadido.
 */
private boolean[][] espacioEtiquetas;

/**
 * Constructor sin parámetros.
 */
public Solucion_OC_4S()
{
super();
}

/**
 * Constructor con parámetros.
 *
```

```
 * @param puntos
 *           List<Integer> Lista de puntos
 * @param etiquetas
 *           List<Integer> Lista de etiquetas
 */
public Solucion_OC_4S(List<Integer> puntos, List<Integer> etiquetas)
{
this.puntos = puntos;
this.etiquetas = etiquetas;
this.numElementos = puntos.size();
this.resultado = new ArrayList<String>();
}

/**
 * Método que imprime la solución al problema, si la hubiese.
 */
public void obtenerSolucion()
{
boolean haySolucion = true;
boolean puntoEtiquetado;

if ((Utilidades.valoresValidos(this.puntos))
      && (Utilidades.valoresValidos(this.etiquetas)))
{
if (!Utilidades.puntosOrdenados(puntos))
{
ConjuntoListas conjunto = Utilidades.ordenarListasInteger(this.puntos,
      this.etiquetas);
this.puntos = conjunto.getListaPuntos();
this.etiquetas = conjunto.getListaEtiquetas();
System.out
      .println("La lista de puntos no está ordenada. Procedemos a ordenarla.");
System.out.println("Los pares (punto , etiqueta) indicados han sido:");
for (int i = 0; i < numElementos; i++)
{
System.out.print("par " + (i + 1) + ": (" + this.puntos.get(i) + " , "
      + this.etiquetas.get(i) + ")\n");
}
} else
{
System.out
      .println("La lista de puntos está ordenada. Procedemos a la resolución
del problema:");
}
inicializaValores();

for (int i = 0; (i < this.numElementos) && haySolucion; i++)
{
puntoEtiquetado = false;

// Etiquetado en la posición ABAJO-IZQUIERDA
if (!puntoEtiquetado
      && this.probarEtiquetaBL(this.puntos.get(i), this.etiquetas.get(i)))
{
puntoEtiquetado = true;
this.aplicarEtiquetaBL(this.puntos.get(i), this.etiquetas.get(i));
this.resultado.set(i, Constantes.POS_ABAJO_IZQUIERDA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
}

// Etiquetado en la posición IZQUIERDA
```

```
if (!puntoEtiquetado)
{
int desliza = this.probarEtiquetaLL(this.puntos.get(i), this.etiquetas.get(i));
if (-1 != desliza)
{
puntoEtiquetado = true;
this.aplicarEtiquetaLL(this.puntos.get(i), this.etiquetas.get(i), desliza);
this.resultado.set(i, Constantes.POS_IZQUIERDA);
System.out.println("\nEstado del espacio después de etiquetar el punto"
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
if (-1 != desliza)
{
System.out.println("Grado de deslizamiento: " + desliza);
}
}
}

// Etiquetado en la posición ABAJO
if (!puntoEtiquetado)
{
int desliza = this.probarEtiquetaBB(this.puntos.get(i), this.etiquetas.get(i));
if (-1 != desliza)
{
puntoEtiquetado = true;
this.aplicarEtiquetaBB(this.puntos.get(i), this.etiquetas.get(i), desliza);
this.resultado.set(i, Constantes.POS_ABAJO);
System.out.println("\nEstado del espacio después de etiquetar el punto"
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
if (-1 != desliza)
{
System.out.println("Grado de deslizamiento: " + desliza);
}
}
}

// Etiquetado en la posición ARRIBA-IZQUIERDA
if (!puntoEtiquetado
      && this.probarEtiquetaTL(this.puntos.get(i), this.etiquetas.get(i)))
{
puntoEtiquetado = true;
this.aplicarEtiquetaTL(this.puntos.get(i), this.etiquetas.get(i));
this.resultado.set(i, Constantes.POS_ARRIBA_IZQUIERDA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
}

// Etiquetado en la posición ABAJO-DERECHA
if (!puntoEtiquetado
      && this.probarEtiquetaBR(this.puntos.get(i), this.etiquetas.get(i)))
{
puntoEtiquetado = true;
this.aplicarEtiquetaBR(this.puntos.get(i), this.etiquetas.get(i));
this.resultado.set(i, Constantes.POS_ABAJO_DERECHA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
}

// Etiquetado en la posición ARRIBA
if (!puntoEtiquetado)
```

```
{
int desliza = this.probarEtiquetaTT(this.puntos.get(i), this.etiquetas.get(i));
if (-1 != desliza)
{
puntoEtiquetado = true;
this.aplicarEtiquetaTT(this.puntos.get(i), this.etiquetas.get(i), desliza);
this.resultado.set(i, Constantes.POS_ARRIBA);
System.out.println("\nEstado del espacio después de etiquetar el punto"
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
if (-1 != desliza)
{
System.out.println("Grado de deslizamiento: " + desliza);
}
}
}

// Etiquetado en la posición DERECHA
if (!puntoEtiquetado)
{
int desliza = this.probarEtiquetaRR(this.puntos.get(i), this.etiquetas.get(i));
if (-1 != desliza)
{
puntoEtiquetado = true;
this.aplicarEtiquetaRR(this.puntos.get(i), this.etiquetas.get(i), desliza);
this.resultado.set(i, Constantes.POS_DERECHA);
System.out.println("\nEstado del espacio después de etiquetar el punto"
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
if (-1 != desliza)
{
System.out.println("Grado de deslizamiento: " + desliza);
}
}
}

// Etiquetado en la posición ARRIBA-DERECHA
if (!puntoEtiquetado
      && this.probarEtiquetaTR(this.puntos.get(i), this.etiquetas.get(i)))
{
puntoEtiquetado = true;
this.aplicarEtiquetaTR(this.puntos.get(i), this.etiquetas.get(i));
this.resultado.set(i, Constantes.POS_ABAJO_DERECHA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
}

if (!puntoEtiquetado)
{
haySolucion = false;
}
}

System.out.println("\nSe han obtenido las siguientes posiciones:");
Utilidades.imprimeListaString(resultado);
if (!haySolucion)
{
System.out.println("No se ha podido encontrar una solución válida a ");
System.out.println("este problema con los datos que ha proporcionado.");
System.out
      .println("Gráficamente, la solución parcial obtenida ha sido la
siguiente:");
```

```
}
if (haySolucion)
{
System.out
      .println("\nGráficamente, la solución del problema se puede representar
como sigue:");
}
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
System.out.println(">> Fin del problema.");
} else
{
System.out
      .println("Los valores de los puntos y etiquetas han de ser mayores que
cero.");
System.out.println("Vuelva a ejecutar el programa si lo desea.");
}
}

/**
 * Método que inicializa las listas "resultado", "lineaSuperior" y
 * "lineaInferior".
 */
public void inicializaValores()
{
this.longitudLinea = this.puntos.get(numElementos - 1) + 2
      + Utilidades.maxValorLista(this.etiquetas);
this.espacioEtiquetas = new boolean[this.longitudLinea][this.longitudLinea];
for (int i = 0; i < this.longitudLinea; i++)
{
for (int j = 0; j < this.longitudLinea; j++)
{
this.espacioEtiquetas[i][j] = false;
}
}
for (int i = 0; i < this.numElementos; i++)
{
this.resultado.add(i, "--");
}
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición TL para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaTL(Integer punto, Integer etiqueta)
{
boolean dev = true;
int x;
int y = punto;
for (int desp = 0; (desp < etiqueta) && (y + desp < this.longitudLinea) && dev;
desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta) && dev; x--)
{
if (this.espacioEtiquetas[y + desp][x])
{
```

```
dev = false;
}
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición BL para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaBL(Integer punto, Integer etiqueta)
{
boolean dev = true;
int x;
int y = punto - 1;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0) && dev; desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta) && dev; x--)
{
if (this.espacioEtiquetas[y - desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición TR para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaTR(Integer punto, Integer etiqueta)
{
boolean dev = true;
int x;
int y = punto;
for (int desp = 0; (desp < etiqueta) && (y + desp < this.longitudLinea) && dev;
desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta) && dev; x++)
{
if (this.espacioEtiquetas[y + desp][x])
{
dev = false;
}
}
}
```

```
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición BR para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaBR(Integer punto, Integer etiqueta)
{
boolean dev = true;
int x;
int y = punto - 1;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0) && dev; desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta) && dev; x++)
{
if (this.espacioEtiquetas[y - desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición LL para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return int Deslizamiento. -1 Si no se puede etiquetar
 */
public int probarEtiquetaLL(Integer punto, Integer etiqueta)
{
boolean fin = false;
boolean sePuedeEtiquetar;
int desliza = -1;
int x;
for (int deslizamiento = 1; (deslizamiento < etiqueta) && !fin;
deslizamiento++)
{
sePuedeEtiquetar = true;
int y = punto - 1 + deslizamiento;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0) && sePuedeEtiquetar
      && !fin; desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta) && sePuedeEtiquetar; x-
-)
{
if (this.espacioEtiquetas[y - desp][x])
{
sePuedeEtiquetar = false;
```

```
}
}
}
if (sePuedeEtiquetar)
{
desliza = deslizamiento;
fin = true;
}
}
return desliza;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición BB para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return int Deslizamiento. -1 Si no se puede etiquetar
 */
public int probarEtiquetaBB(Integer punto, Integer etiqueta)
{
boolean fin = false;
boolean sePuedeEtiquetar;
int desliza = -1;
int x;
for (int deslizamiento = 1; (deslizamiento < etiqueta) && !fin;
deslizamiento++)
{
sePuedeEtiquetar = true;
int y = punto - 1;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0) && sePuedeEtiquetar
      && !fin; desp++)
{
for (x = punto - 1 + deslizamiento; (x >= 0)
      && (x >= punto - etiqueta + deslizamiento) && sePuedeEtiquetar; x--)
{
if (this.espacioEtiquetas[y - desp][x])
{
sePuedeEtiquetar = false;
}
}
}
if (sePuedeEtiquetar)
{
desliza = deslizamiento;
fin = true;
}
}
return desliza;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición TT para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
```

```
 *           Tamaño de la etiqueta que se va a añadir
 * @return int Deslizamiento. -1 Si no se puede etiquetar
 */
public int probarEtiquetaTT(Integer punto, Integer etiqueta)
{
boolean fin = false;
boolean sePuedeEtiquetar;
int desliza = -1;
int x;
for (int deslizamiento = 1; (deslizamiento < etiqueta) && !fin;
deslizamiento++)
{
sePuedeEtiquetar = true;
int y = punto;
for (int desp = 0; (desp < etiqueta) && (y + desp < this.longitudLinea)
      && sePuedeEtiquetar && !fin; desp++)
{
for (x = punto - 1 + deslizamiento; (x >= 0)
      && (x >= punto - etiqueta + deslizamiento) && sePuedeEtiquetar; x--)
{
if (this.espacioEtiquetas[y + desp][x])
{
sePuedeEtiquetar = false;
}
}
}
if (sePuedeEtiquetar)
{
desliza = deslizamiento;
fin = true;
}
}
return desliza;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición RR para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @return int Deslizamiento. -1 Si no se puede etiquetar
 */
public int probarEtiquetaRR(Integer punto, Integer etiqueta)
{
boolean fin = false;
boolean sePuedeEtiquetar;
int desliza = -1;
int x;
for (int deslizamiento = 1; (deslizamiento < etiqueta) && !fin;
deslizamiento++)
{
sePuedeEtiquetar = true;
int y = punto - 1 + deslizamiento;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0) && sePuedeEtiquetar
      && !fin; desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta)
      && sePuedeEtiquetar; x++)
{
```

```
if (this.espacioEtiquetas[y - desp][x])
{
sePuedeEtiquetar = false;
}
}
}
if (sePuedeEtiquetar)
{
desliza = deslizamiento;
fin = true;
}
}
return desliza;
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición TL.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 */
public void aplicarEtiquetaTL(Integer punto, Integer etiqueta)
{
int x;
int y = punto;
for (int desp = 0; (desp < etiqueta) && (y + desp < this.longitudLinea);
desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta); x--)
{
this.espacioEtiquetas[y + desp][x] = true;
}
}
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición BL.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 */
public void aplicarEtiquetaBL(Integer punto, Integer etiqueta)
{
int x;
int y = punto - 1;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0); desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta); x--)
{
this.espacioEtiquetas[y - desp][x] = true;
}
}
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición TR.
```

```
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 */
public void aplicarEtiquetaTR(Integer punto, Integer etiqueta)
{
int x;
int y = punto;
for (int desp = 0; (desp < etiqueta) && (y + desp < this.longitudLinea);
desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta); x++)
{
this.espacioEtiquetas[y + desp][x] = true;
}
}
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición BR.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 */
public void aplicarEtiquetaBR(Integer punto, Integer etiqueta)
{
int x;
int y = punto - 1;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0); desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta); x++)
{
this.espacioEtiquetas[y - desp][x] = true;
}
}
}

/**
 * Método que agrega la etiqueta en la posición LL y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param delizamiento
 *           Deslizamiento
 * @return boolean Cierto si se añade correctamente la etiqueta
 */
public boolean aplicarEtiquetaLL(Integer punto, Integer etiqueta,
      Integer deslizamiento)
{
boolean dev = true;
int x;
int y = punto - 1 + deslizamiento;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0); desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - etiqueta); x--)
{
```

```
this.espacioEtiquetas[y - desp][x] = true;
}
}
return dev;
}

/**
 * Método que agrega la etiqueta en la posición BB y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param delizamiento
 *           Deslizamiento
 * @return boolean Cierto si se añade correctamente la etiqueta
 */
public boolean aplicarEtiquetaBB(Integer punto, Integer etiqueta,
      Integer deslizamiento)
{
boolean dev = true;
int x;
int y = punto - 1;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0); desp++)
{
for (x = punto - 1 + deslizamiento; (x >= 0)
      && (x >= punto - etiqueta + deslizamiento); x--)
{
this.espacioEtiquetas[y - desp][x] = true;
}
}
return dev;
}

/**
 * Método que agrega la etiqueta en la posición TT y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param delizamiento
 *           Deslizamiento
 * @return boolean Cierto si se añade correctamente la etiqueta
 */
public boolean aplicarEtiquetaTT(Integer punto, Integer etiqueta,
      Integer deslizamiento)
{
boolean dev = true;
int x;
int y = punto;
for (int desp = 0; (desp < etiqueta) && (y + desp < this.longitudLinea);
desp++)
{
for (x = punto - 1 + deslizamiento; (x >= 0)
      && (x >= punto - etiqueta + deslizamiento); x--)
{
this.espacioEtiquetas[y + desp][x] = true;
}
}
return dev;
}
```

```
/**
 * Método que agrega la etiqueta en la posición RR y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param etiqueta
 *           Tamaño de la etiqueta que se va a añadir
 * @param delizamiento
 *           Deslizamiento
 * @return boolean Cierto si se añade correctamente la etiqueta
 */
public boolean aplicarEtiquetaRR(Integer punto, Integer etiqueta,
      Integer deslizamiento)
{
boolean dev = true;
int x;
int y = punto - 1 + deslizamiento;
for (int desp = 0; (desp < etiqueta) && (y - desp >= 0); desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + etiqueta); x++)
{
this.espacioEtiquetas[y - desp][x] = true;
}
}
return dev;
}
}
```

ProblemasDeEtiquetado/src/lineaOblicua/or1_4p/
Main_OR1_4P.java

```
package lineaOblicua.or1_4p;

import generalidades.Constantes;
import generalidades.Utilidades;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

/**
 * Clase principal que resuelve el problema OR1-4P.
 *
 * @author José Manuel Ponce
 */
public class Main_OR1_4P
{
/**
 * Instancia de un objeto Solucion_OR1_4P.
 */
private static Solucion_OR1_4P solucion;

/**
 * Código a ejecutar.
 *
 * @param args
 *           Argumentos
 */
public static void main(String[] args)
{
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
System.out.println(Constantes.CABECERA_OR1_4P);
System.out.println();

boolean salir = false, error = false, finProblema = false;
int i = 0, numElementos = 0, anchuraEtiqueta = 0, alturaEtiqueta = 0;
String cadenaEntrada = "";
List<Integer> puntos = new ArrayList<Integer>();

try
{
while (!salir && !finProblema)
{
System.out
      .println("Pulse la tecla \"T\" para introducir los datos desde el
teclado.");
System.out
      .println("Pulse la tecla \"F\" si desea que la aplicación obtenga los
datos a partir de un fichero de texto.");
System.out
      .println("Pulse la tecla \"I\" si desea información adicional acerca de
este problema.");
System.out.println("Pulse la tecla \"S\" para salir de la aplicación.");

cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
```

```
{
System.out.println(Constantes.INFO_OR1_4P);
}

else
{
if ("T".equals(cadenaEntrada) || "t".equals(cadenaEntrada))
{
// El usuario prefiere introducir los datos desde el
// teclado.
System.out
      .println("Introduzca el número de elementos. Pulse \"S\" para salir.");
cadenaEntrada = in.readLine();

if (cadenaEntrada.equals("S") || cadenaEntrada.equals("s"))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else
{
numElementos = Integer.valueOf(cadenaEntrada).intValue();
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
System.out.println("Ha indicado " + numElementos + " elementos. ");
i = 1;
while (i <= numElementos)
{
System.out.println("Introduzca el punto " + i + ":");
puntos.add(Integer.valueOf(in.readLine()).intValue());
i++;
}
System.out.println("Introduzca la altura de las etiquetas:");
alturaEtiqueta = Integer.valueOf(in.readLine()).intValue();
System.out.println("Introduzca la anchura de las etiquetas:");
anchuraEtiqueta = Integer.valueOf(in.readLine()).intValue();
finProblema = true;
}
}
}

else if (cadenaEntrada.equals("F") || cadenaEntrada.equals("f"))
{
// El usuario prefiere utilizar un fichero de texto para
// indicar los datos.
System.out
      .println("Guarde el fichero de texto con el nombre \"entrada_OR1-4P.txt\"
en la misma ubicación donde se encuentra");
System.out
      .println("el proyecto y pulse \"C\" para continuar. Pulse \"S\" para
abandonar la aplicación.");
System.out
      .println("Si desea más información acerca del formato del fichero pulse
la tecla \"I\".");

cadenaEntrada = in.readLine();
```

```
if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
{
System.out.println(Constantes.INFO_FICHERO_OR1_4P);
}

else
{
if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else if ("C".equals(cadenaEntrada) || "c".equals(cadenaEntrada))
{
System.out.println("Leyendo el fichero:");
BufferedReader br = new BufferedReader(new FileReader("entrada_OR1-4P.txt"));
String s;
if ((s = br.readLine()) != null)
{
numElementos = Integer.valueOf(s.trim());
}
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
i = 2;
while (((s = br.readLine()) != null) && !error)
{
if (i >= 2 && i <= (numElementos + 1))
{
puntos.add(Integer.valueOf(s.trim()));
} else if (i == numElementos + 2)
{
alturaEtiqueta = Integer.valueOf(s.trim());
} else if (i == numElementos + 3)
{
anchuraEtiqueta = Integer.valueOf(s.trim());
} else
{
error = true;
}
i++;
}
}
br.close();
finProblema = true;
}
}
}

else if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
}
}

if (finProblema)
```

```
{
if (puntos.size() != numElementos)
{
System.out
      .println("Se ha producido un error en la obtención de los datos: El
número total de puntos y etiquetas no coincide con el indicado.");
salir = true;
}
}
}

if (finProblema && !salir)
{
System.out.println("Los puntos indicados han sido:");
Utilidades.imprimeListaInteger(puntos);
solucion = new Solucion_OR1_4P(puntos, alturaEtiqueta, anchuraEtiqueta);
solucion.obtenerSolucion();
}

} catch (NumberFormatException e)
{
System.out
      .println("\nSe ha producido una excepción al introducir un valor
incorrecto: NumberFormatException.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (IOException e)
{
System.out
      .println("\nSe ha producido una excepción de entrada/salida:
IOException.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (ArrayIndexOutOfBoundsException e)
{
System.out
      .println("\nSe ha producido un error: ArrayIndexOutOfBoundsException.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
}
}
}
```

ProblemasDeEtiquetado/src/lineaOblicua/or1_4p/
Solucion_OR1_4P.java

```
package lineaOblicua.or1_4p;

import generalidades.Constantes;
import generalidades.ListaEnteros;
import generalidades.Utilidades;

import java.util.ArrayList;
import java.util.List;

import lineaOblicua.UtilidadesOblicua;

/**
 * Clase que contiene los métodos necesarios para solucionar el problema OC-4P.
 *
 * @author José Manuel Ponce
 */
public class Solucion_OR1_4P
{
/**
 * Lista de puntos.
 */
private List<Integer> puntos;

/**
 * Lista de posiciones finales de las etiquetas respecto a los puntos.
 */
private List<String> resultado;

/**
 * Número de pares punto-etiqueta.
 */
private int numElementos;

/**
 * Número de divisiones de la línea de entrada.
 */
private int longitudLinea;

/**
 * Anchura de las etiquetas.
 */
private int anchuraEtiqueta;

/**
 * Altura de las etiquetas.
 */
private int alturaEtiqueta;

/**
 * Objeto que almacena la posición de las etiquetas que ya se han añadido.
 */
private boolean[][] espacioEtiquetas;

/**
 * Constructor sin parámetros.
 */
public Solucion_OR1_4P()
{
super();
```

```
}

/**
 * Constructor con parámetros.
 *
 * @param puntos
 *           List<Integer> Lista de puntos
 * @param alturaEtiquetas
 *           int Altura de las etiquetas
 * @param anchuraEtiquetas
 *           int Anchura de las etiquetas
 */
public Solucion_OR1_4P(List<Integer> puntos, int alturaEtiquetas,
      int anchuraEtiquetas)
{
this.puntos = puntos;
this.numElementos = puntos.size();
this.resultado = new ArrayList<String>();
this.anchuraEtiqueta = anchuraEtiquetas;
this.alturaEtiqueta = alturaEtiquetas;
}

/**
 * Método que imprime la solución al problema, si la hubiese.
 */
public void obtenerSolucion()
{
boolean haySolucion = true;

if (Utilidades.valoresValidos(this.puntos) && 0 < this.alturaEtiqueta
      && 0 < this.anchuraEtiqueta)
{
if (!Utilidades.puntosOrdenados(puntos))
{
ListaEnteros lista = Utilidades.ordenarListaInteger(this.puntos);
this.puntos = lista.getLista();
System.out
      .println("La lista de puntos no está ordenada. Procedemos a ordenarla.");
System.out.println("Los puntos indicados han sido:");
Utilidades.imprimeListaInteger(this.puntos);
System.out.println("La altura de las etiquetas es " + this.alturaEtiqueta
      + " y la anchura " + this.anchuraEtiqueta + ".");
} else
{
System.out
      .println("La lista de puntos está ordenada. Procedemos a la resolución
del problema:");
}
inicializaValores();

for (int i = 0; (i < this.numElementos) && haySolucion; i++)
{
if (this.probarEtiquetaBL(this.puntos.get(i)))
{
this.aplicarEtiquetaBL(this.puntos.get(i));
this.resultado.set(i, Constantes.POS_ABAJO_IZQUIERDA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else if (this.probarEtiquetaTL(this.puntos.get(i)))
{
this.aplicarEtiquetaTL(this.puntos.get(i));
this.resultado.set(i, Constantes.POS_ARRIBA_IZQUIERDA);
```

```
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else if (this.probarEtiquetaBR(this.puntos.get(i)))
{
this.aplicarEtiquetaBR(this.puntos.get(i));
this.resultado.set(i, Constantes.POS_ABAJO_DERECHA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else if (this.probarEtiquetaTR(this.puntos.get(i)))
{
this.aplicarEtiquetaTR(this.puntos.get(i));
this.resultado.set(i, Constantes.POS_ARRIBA_DERECHA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else
{
haySolucion = false;
}
}

System.out.println("\nSe han obtenido las siguientes posiciones:");
Utilidades.imprimeListaString(resultado);
if (!haySolucion)
{
System.out.println("No se ha podido encontrar una solución válida a ");
System.out.println("este problema con los datos que ha proporcionado.");
System.out
      .println("Gráficamente, la solución parcial obtenida ha sido la
siguiente:");
}
if (haySolucion)
{
System.out
      .println("\nGráficamente, la solución del problema se puede representar
como sigue:");
}
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
System.out.println(">> Fin del problema.");
} else
{
System.out
      .println("Los valores de los puntos y etiquetas han de ser mayores que
cero.");
System.out.println("Vuelva a ejecutar el programa si lo desea.");
}
}

/**
 * Método que inicializa las listas "resultado", "lineaSuperior" y
 * "lineaInferior".
 */
public void inicializaValores()
{
this.longitudLinea = this.puntos.get(numElementos - 1) + 2
      + this.anchuraEtiqueta;
this.espacioEtiquetas = new boolean[this.longitudLinea][this.longitudLinea];
for (int i = 0; i < this.longitudLinea; i++)
{
for (int j = 0; j < this.longitudLinea; j++)
{
```

```
this.espacioEtiquetas[i][j] = false;
}
}
for (int i = 0; i < this.numElementos; i++)
{
this.resultado.add(i, "--");
}
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición TL para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaTL(Integer punto)
{
boolean dev = true;
int x;
int y = punto;
for (int desp = 0; (desp < this.alturaEtiqueta)
      && (y + desp < this.longitudLinea) && dev; desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - this.anchuraEtiqueta) && dev; x--
)
{
if (this.espacioEtiquetas[y + desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición BL para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaBL(Integer punto)
{
boolean dev = true;
int x;
int y = punto - 1;
for (int desp = 0; (desp < this.alturaEtiqueta) && (y - desp >= 0) && dev;
desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - this.anchuraEtiqueta) && dev; x--
)
{
if (this.espacioEtiquetas[y - desp][x])
{
dev = false;
}
}
```

```
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición TR para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaTR(Integer punto)
{
boolean dev = true;
int x;
int y = punto;
for (int desp = 0; (desp < this.alturaEtiqueta)
      && (y + desp < this.longitudLinea) && dev; desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + this.anchuraEtiqueta)
      && dev; x++)
{
if (this.espacioEtiquetas[y + desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición BR para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaBR(Integer punto)
{
boolean dev = true;
int x;
int y = punto - 1;
for (int desp = 0; (desp < this.alturaEtiqueta) && (y - desp >= 0) && dev;
desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + this.anchuraEtiqueta)
      && dev; x++)
{
if (this.espacioEtiquetas[y - desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
```

```
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición TL.
 *
 * @param punto
 *           Punto a etiquetar
 */
public void aplicarEtiquetaTL(Integer punto)
{
int x;
int y = punto;
for (int desp = 0; (desp < this.alturaEtiqueta)
      && (y + desp < this.longitudLinea); desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - this.anchuraEtiqueta); x--)
{
this.espacioEtiquetas[y + desp][x] = true;
}
}
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición BL.
 *
 * @param punto
 *           Punto a etiquetar
 */
public void aplicarEtiquetaBL(Integer punto)
{
int x;
int y = punto - 1;
for (int desp = 0; (desp < this.alturaEtiqueta) && (y - desp >= 0); desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - this.anchuraEtiqueta); x--)
{
this.espacioEtiquetas[y - desp][x] = true;
}
}
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición TR.
 *
 * @param punto
 *           Punto a etiquetar
 */
public void aplicarEtiquetaTR(Integer punto)
{
int x;
int y = punto;
for (int desp = 0; (desp < this.alturaEtiqueta)
      && (y + desp < this.longitudLinea); desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + this.anchuraEtiqueta);
x++)
{
this.espacioEtiquetas[y + desp][x] = true;
}
}
}

/**
```

```
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición BR.
 *
 * @param punto
 *           Punto a etiquetar
 */
public void aplicarEtiquetaBR(Integer punto)
{
int x;
int y = punto - 1;
for (int desp = 0; (desp < this.alturaEtiqueta) && (y - desp >= 0); desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + this.anchuraEtiqueta);
x++)
{
this.espacioEtiquetas[y - desp][x] = true;
}
}
}
}
```

ProblemasDeEtiquetado/src/lineaOblicua/orm_4p/
Main_ORM_4P.java

```
package lineaOblicua.orm_4p;

import generalidades.Constantes;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

/**
 * Clase principal que resuelve el problema ORM-4P.
 *
 * @author José Manuel Ponce
 */
public class Main_ORM_4P
{
/**
 * Instancia de un objeto Solucion_OR1_4P.
 */
private static Solucion_ORM_4P solucion;

/**
 * Código a ejecutar.
 *
 * @param args
 *           Argumentos
 */
public static void main(String[] args)
{
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
System.out.println(Constantes.CABECERA_ORM_4P);
System.out.println();

boolean salir = false, error = false, finProblema = false;
int i = 0, numElementos = 0;
String cadenaEntrada = "";
List<Integer> puntos = new ArrayList<Integer>();
List<Integer> alturas = new ArrayList<Integer>();
List<Integer> anchuras = new ArrayList<Integer>();

try
{
while (!salir && !finProblema)
{
System.out
      .println("Pulse la tecla \"T\" para introducir los datos desde el
teclado.");
System.out
      .println("Pulse la tecla \"F\" si desea que la aplicación obtenga los
datos a partir de un fichero de texto.");
System.out
      .println("Pulse la tecla \"I\" si desea información adicional acerca de
este problema.");
System.out.println("Pulse la tecla \"S\" para salir de la aplicación.");

cadenaEntrada = in.readLine();
```

```
if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
{
System.out.println(Constantes.INFO_ORM_4P);
}

else
{
if ("T".equals(cadenaEntrada) || "t".equals(cadenaEntrada))
{
// El usuario prefiere introducir los datos desde el
// teclado.
System.out
      .println("Introduzca el número de elementos. Pulse \"S\" para salir.");
cadenaEntrada = in.readLine();

if (cadenaEntrada.equals("S") || cadenaEntrada.equals("s"))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else
{
numElementos = Integer.valueOf(cadenaEntrada).intValue();
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
alturas = new ArrayList<Integer>(numElementos);
anchuras = new ArrayList<Integer>(numElementos);
System.out.println("Ha indicado " + numElementos + " elementos. ");
i = 1;
while (i <= numElementos)
{
System.out.println("Introduzca el punto " + i + ":");
puntos.add(Integer.valueOf(in.readLine()).intValue());
i++;
}
i = 1;
while (i <= numElementos)
{
System.out.println("Introduzca la altura para el punto " + i + ":");
alturas.add(Integer.valueOf(in.readLine()).intValue());
i++;
}
i = 1;
while (i <= numElementos)
{
System.out.println("Introduzca la anchura para el punto " + i + ":");
anchuras.add(Integer.valueOf(in.readLine()).intValue());
i++;
}
finProblema = true;
}
}
}

else if (cadenaEntrada.equals("F") || cadenaEntrada.equals("f"))
{
// El usuario prefiere utilizar un fichero de texto para
```

```
// indicar los datos.
System.out
      .println("Guarde el fichero de texto con el nombre \"entrada_ORM-4P.txt\"
en la misma ubicación donde se encuentra");
System.out
      .println("el proyecto y pulse \"C\" para continuar. Pulse \"S\" para
abandonar la aplicación.");
System.out
      .println("Si desea más información acerca del formato del fichero pulse
la tecla \"I\".");

cadenaEntrada = in.readLine();

if ("I".equals(cadenaEntrada) || "i".equals(cadenaEntrada))
{
System.out.println(Constantes.INFO_FICHERO_ORM_4P);
}

else
{
if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
} else if ("C".equals(cadenaEntrada) || "c".equals(cadenaEntrada))
{
System.out.println("Leyendo el fichero:");
BufferedReader br = new BufferedReader(new FileReader("entrada_ORM-4P.txt"));
String s;
if ((s = br.readLine()) != null)
{
numElementos = Integer.valueOf(s.trim());
}
if (1 > numElementos)
{
System.out.println("El número de elementos debe ser mayor que cero.");
salir = true;
break;
} else
{
puntos = new ArrayList<Integer>(numElementos);
alturas = new ArrayList<Integer>(numElementos);
anchuras = new ArrayList<Integer>(numElementos);
i = 2;
while (((s = br.readLine()) != null) && !error)
{
if (i <= (numElementos + 1))
{
puntos.add(Integer.valueOf(s.trim()));
} else if (i <= 2 * numElementos + 1)
{
alturas.add(Integer.valueOf(s.trim()));
} else if (i <= 3 * numElementos + 1)
{
anchuras.add(Integer.valueOf(s.trim()));
} else
{
error = true;
}
i++;
}
}
```

```
br.close();
finProblema = true;
}
}
}

else if ("S".equals(cadenaEntrada) || "s".equals(cadenaEntrada))
{
System.out.println("La aplicación ha finalizado correctamente.");
salir = true;
break;
}
}

if (finProblema)
{
if (puntos.size() != numElementos || alturas.size() != numElementos
      || anchuras.size() != numElementos)
{
System.out
      .println("Se ha producido un error en la obtención de los datos: El
número total de puntos y etiquetas no coincide con el indicado.");
salir = true;
}
}
}

if (finProblema && !salir)
{
System.out
      .println("Las tripletas (punto, altura, anchura) indicadas han sido:");
for (i = 0; i < numElementos; i++)
{
System.out.println("( " + puntos.get(i) + " , " + alturas.get(i) + " , "
      + anchuras.get(i) + " )");
}
solucion = new Solucion_ORM_4P(puntos, alturas, anchuras);
solucion.obtenerSolucion();
}

} catch (NumberFormatException e)
{
System.out
      .println("\nSe ha producido una excepción al introducir un valor
incorrecto: NumberFormatException.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (IOException e)
{
System.out
      .println("\nSe ha producido una excepción de entrada/salida:
IOException.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
} catch (ArrayIndexOutOfBoundsException e)
{
System.out
      .println("\nSe ha producido un error: ArrayIndexOutOfBoundsException.");
System.out.println("Vuelva a ejecutar el programa si desea continuar.");
e.printStackTrace();
}
}
}
```

ProblemasDeEtiquetado/src/lineaOblicua/orm_4p/
Solucion_ORM_4P.java

```
package lineaOblicua.orm_4p;

import generalidades.ConjuntoTresListas;
import generalidades.Constantes;
import generalidades.Utilidades;

import java.util.ArrayList;
import java.util.List;

import lineaOblicua.UtilidadesOblicua;

/**
 * Clase que contiene los métodos necesarios para solucionar el problema OC-4P.
 *
 * @author José Manuel Ponce
 */
public class Solucion_ORM_4P
{
/**
 * Lista de puntos.
 */
private List<Integer> puntos;

/**
 * Lista de posiciones finales de las etiquetas respecto a los puntos.
 */
private List<String> resultado;

/**
 * Número de pares punto-etiqueta.
 */
private int numElementos;

/**
 * Número de divisiones de la línea de entrada.
 */
private int longitudLinea;

/**
 * Anchuras de las etiquetas.
 */
private List<Integer> anchuras;

/**
 * Alturas de las etiquetas.
 */
private List<Integer> alturas;

/**
 * Objeto que almacena la posición de las etiquetas que ya se han añadido.
 */
private boolean[][] espacioEtiquetas;

/**
 * Constructor sin parámetros.
 */
public Solucion_ORM_4P()
{
super();
```

```
}

/**
 * Constructor con parámetros.
 *
 * @param puntos
 *           List<Integer> Lista de puntos
 * @param alturaEtiquetas
 *           List<Integer> Altura de las etiquetas
 * @param anchuraEtiquetas
 *           List<Integer> Anchura de las etiquetas
 */
public Solucion_ORM_4P(List<Integer> puntos, List<Integer> alturas,
      List<Integer> anchuras)
{
this.puntos = puntos;
this.numElementos = puntos.size();
this.resultado = new ArrayList<String>();
this.anchuras = anchuras;
this.alturas = alturas;
}

/**
 * Método que imprime la solución al problema, si la hubiese.
 */
public void obtenerSolucion()
{
boolean haySolucion = true;

if (Utilidades.valoresValidos(this.puntos)
      && Utilidades.valoresValidos(this.alturas)
      && Utilidades.valoresValidos(this.anchuras))
{
if (!Utilidades.puntosOrdenados(puntos))
{
ConjuntoTresListas conjunto = Utilidades.ordenarTresListasInteger(this.puntos,
      this.alturas, this.anchuras);
this.puntos = conjunto.getListaPuntos();
this.alturas = conjunto.getAlturas();
this.anchuras = conjunto.getAnchuras();
System.out
      .println("La lista de puntos no está ordenada. Procedemos a ordenarla.");
System.out
      .println("Las tripletas (punto, altura, anchura) indicadas han sido:");
for (int i = 0; i < this.numElementos; i++)
{
System.out.println("( " + this.puntos.get(i) + " , " + this.alturas.get(i)
      + " , " + this.anchuras.get(i) + " )");
}
} else
{
System.out
      .println("La lista de puntos está ordenada. Procedemos a la resolución
del problema:");
}
inicializaValores();

for (int i = 0; (i < this.numElementos) && haySolucion; i++)
{
if (this.probarEtiquetaBL(this.puntos.get(i), this.alturas.get(i),
      this.anchuras.get(i)))
{
this.aplicarEtiquetaBL(this.puntos.get(i), this.alturas.get(i), this.anchuras
```

```
      .get(i));
this.resultado.set(i, Constantes.POS_ABAJO_IZQUIERDA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else if (this.probarEtiquetaTL(this.puntos.get(i), this.alturas.get(i),
      this.anchuras.get(i)))
{
this.aplicarEtiquetaTL(this.puntos.get(i), this.alturas.get(i), this.anchuras
      .get(i));
this.resultado.set(i, Constantes.POS_ARRIBA_IZQUIERDA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else if (this.probarEtiquetaBR(this.puntos.get(i), this.alturas.get(i),
      this.anchuras.get(i)))
{
this.aplicarEtiquetaBR(this.puntos.get(i), this.alturas.get(i), this.anchuras
      .get(i));
this.resultado.set(i, Constantes.POS_ABAJO_DERECHA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else if (this.probarEtiquetaTR(this.puntos.get(i), this.alturas.get(i),
      this.anchuras.get(i)))
{
this.aplicarEtiquetaTR(this.puntos.get(i), this.alturas.get(i), this.anchuras
      .get(i));
this.resultado.set(i, Constantes.POS_ARRIBA_DERECHA);
System.out.println("\nEstado del espacio después de etiquetar el punto "
      + (this.puntos.get(i)));
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
} else
{
haySolucion = false;
}
}

System.out.println("\nSe han obtenido las siguientes posiciones:");
Utilidades.imprimeListaString(resultado);
if (!haySolucion)
{
System.out.println("No se ha podido encontrar una solución válida a ");
System.out.println("este problema con los datos que ha proporcionado.");
System.out
      .println("Gráficamente, la solución parcial obtenida ha sido la
siguiente:");
}
if (haySolucion)
{
System.out
      .println("\nGráficamente, la solución del problema se puede representar
como sigue:");
}
UtilidadesOblicua.representarEspacio(this.espacioEtiquetas);
System.out.println(">> Fin del problema.");
} else
{
System.out
      .println("Los valores de los puntos y etiquetas han de ser mayores que
cero.");
System.out.println("Vuelva a ejecutar el programa si lo desea.");
}
```

```
}

/**
 * Método que inicializa las listas "resultado", "lineaSuperior" y
 * "lineaInferior".
 */
public void inicializaValores()
{
this.longitudLinea = this.puntos.get(numElementos - 1) + 2
      + Utilidades.maxValorLista(this.anchuras);
this.espacioEtiquetas = new boolean[this.longitudLinea][this.longitudLinea];
for (int i = 0; i < this.longitudLinea; i++)
{
for (int j = 0; j < this.longitudLinea; j++)
{
this.espacioEtiquetas[i][j] = false;
}
}
for (int i = 0; i < this.numElementos; i++)
{
this.resultado.add(i, "--");
}
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición TL para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param alturaE
 *           Altura de la etiqueta
 * @param anchuraE
 *           Anchura de la etiqueta
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaTL(Integer punto, Integer alturaE, Integer
anchuraE)
{
boolean dev = true;
int x;
int y = punto;
for (int desp = 0; (desp < alturaE) && (y + desp < this.longitudLinea) && dev;
desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - anchuraE) && dev; x--)
{
if (this.espacioEtiquetas[y + desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición BL para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
```

```
 *           Punto a etiquetar
 * @param alturaE
 *           Altura de la etiqueta
 * @param anchuraE
 *           Anchura de la etiqueta
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaBL(Integer punto, Integer alturaE, Integer
anchuraE)
{
boolean dev = true;
int x;
int y = punto - 1;
for (int desp = 0; (desp < alturaE) && (y - desp >= 0) && dev; desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - anchuraE) && dev; x--)
{
if (this.espacioEtiquetas[y - desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición TR para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
 *
 * @param punto
 *           Punto a etiquetar
 * @param alturaE
 *           Altura de la etiqueta
 * @param anchuraE
 *           Anchura de la etiqueta
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaTR(Integer punto, Integer alturaE, Integer
anchuraE)
{
boolean dev = true;
int x;
int y = punto;
for (int desp = 0; (desp < alturaE) && (y + desp < this.longitudLinea) && dev;
desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + anchuraE) && dev; x++)
{
if (this.espacioEtiquetas[y + desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
 * Método que simula el agregar una nueva etiqueta en el punto indicado de la
 * línea en la posición BR para comprobar el tamaño de la sombra resultante y,
 * en su caso, agrega la etiqueta y pasa al siguiente punto.
```

```
 *
 * @param punto
 *           Punto a etiquetar
 * @param alturaE
 *           Altura de la etiqueta
 * @param anchuraE
 *           Anchura de la etiqueta
 * @return boolean Cierto si se puede colocar la etiqueta en esta posición
 */
public boolean probarEtiquetaBR(Integer punto, Integer alturaE, Integer
anchuraE)
{
boolean dev = true;
int x;
int y = punto - 1;
for (int desp = 0; (desp < alturaE) && (y - desp >= 0) && dev; desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + anchuraE) && dev; x++)
{
if (this.espacioEtiquetas[y - desp][x])
{
dev = false;
}
}
}
return dev;
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición TL.
 *
 * @param punto
 *           Punto a etiquetar
 * @param alturaE
 *           Altura de la etiqueta
 * @param anchuraE
 *           Anchura de la etiqueta
 */
public void aplicarEtiquetaTL(Integer punto, Integer alturaE, Integer anchuraE)
{
int x;
int y = punto;
for (int desp = 0; (desp < alturaE) && (y + desp < this.longitudLinea); desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - anchuraE); x--)
{
this.espacioEtiquetas[y + desp][x] = true;
}
}
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición BL.
 *
 * @param punto
 *           Punto a etiquetar
 * @param alturaE
 *           Altura de la etiqueta
 * @param anchuraE
 *           Anchura de la etiqueta
 */
```

```
public void aplicarEtiquetaBL(Integer punto, Integer alturaE, Integer anchuraE)
{
int x;
int y = punto - 1;
for (int desp = 0; (desp < alturaE) && (y - desp >= 0); desp++)
{
for (x = punto - 1; (x >= 0) && (x >= punto - anchuraE); x--)
{
this.espacioEtiquetas[y - desp][x] = true;
}
}
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición TR.
 *
 * @param punto
 *           Punto a etiquetar
 * @param alturaE
 *           Altura de la etiqueta
 * @param anchuraE
 *           Anchura de la etiqueta
 */
public void aplicarEtiquetaTR(Integer punto, Integer alturaE, Integer anchuraE)
{
int x;
int y = punto;
for (int desp = 0; (desp < alturaE) && (y + desp < this.longitudLinea); desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + anchuraE); x++)
{
this.espacioEtiquetas[y + desp][x] = true;
}
}
}

/**
 * Método que agrega una nueva etiqueta en el punto indicado de la línea en la
 * posición BR.
 *
 * @param punto
 *           Punto a etiquetar
 * @param alturaE
 *           Altura de la etiqueta
 * @param anchuraE
 *           Anchura de la etiqueta
 */
public void aplicarEtiquetaBR(Integer punto, Integer alturaE, Integer anchuraE)
{
int x;
int y = punto - 1;
for (int desp = 0; (desp < alturaE) && (y - desp >= 0); desp++)
{
for (x = punto; (x < this.longitudLinea) && (x < punto + anchuraE); x++)
{
this.espacioEtiquetas[y - desp][x] = true;
}
}
}
}
```

ProblemasDeEtiquetado/src/lineaOblicua/
UtilidadesOblicua.java

```
package lineaOblicua;

import generalidades.Utilidades;

/**
 * Clase que almacena métodos útiles para problemas de etiquetado sobre línea
 * oblicua.
 *
 * @author José Manuel Ponce
 */
public class UtilidadesOblicua
{
/**
 * Método que se encarga de representar gráficamente el estado del etiquetado
de
 * puntos sobre una línea oblicua.
 *
 * @param espacioEtiquetas
 *            Sombra actual del problema
 */
public static void representarEspacio(boolean[][] espacioEtiquetas)
{
int tamanyo = espacioEtiquetas.length;
boolean[][] espacioVacio = new boolean[tamanyo][tamanyo];
for (int i = 0; i < tamanyo; i++)
{
for (int j = 0; j < tamanyo; j++)
{
if (i == j)
{
espacioVacio[i][j] = true;
} else
{
espacioVacio[i][j] = false;
}
}
}

for (int y = tamanyo - 1; y >= 0; y--)
{
System.out.print(" " + Utilidades.mod10(y));
for (int x = 0; x < tamanyo; x++)
{
if (espacioEtiquetas[y][x])
{
System.out.print(" X");
} else
{
if (espacioVacio[y][x])
{
System.out.print(" /");
} else
{
System.out.print(" ·");
}
}
}
System.out.println();
}
```

```
System.out.print("  ");
for (int i = 0; i < tamanyo; i++)
{
System.out.print(" " + Utilidades.mod10(i));
}
System.out.println();
}
}
```

Capítulo 10

Referencias y Bibliografía

[1] "*Problemas de Etiquetado: Complejidad Computacional*",
Pedro Reyes Columé, 2002

[2] "*Aprenda Java como si estuviera en primero*",
Universidad de Navarra, 2000

[3] "*The Computer in Contemporary Cartography*",
J. Hopkins, 1980

[4] "*7th. Annual ACM Symposium of Computational Geometry*",
M. Formann y F. Wagner, 1991

[5] "*Computational Geometry: Theory and Applications*",
M. van Kreveld, T. Strijk, y A. Wolff, 1999

[6] "*Advances in Discrete And Computational Geometry, vol. 223*",
B. Chazelle y otros 36 autores, 1999

[7] "*Computers and Intractability: a guide to the theory of NP—completeness*",
M.R. Garey y D.S. Johnson, 1979

[8] "*Complexity and Approximation. Combinatorial Optimization Problems and their Approximability Properties*",
G. Ausiello y otros 5 autores, 1999

[9] "*Modern Graph Theory*",
B. Bollobas, 1998

[10] "*Computational Geometry*",
F.P. Preparata y M.I. Shamos, 1985

[11] "*IEEE-Microwave and Wireless Components Letters*",
H. Choo y H. Ling, 2002

[12] "*Handbook of Computational Geometry*",
J.R. Sack y J. Urrutia, 2000

[13] "*Graph Drawing. Algorithms for the visualization of graphs*",
G. di Battista, P. Eades, R. Tamassia, y I.G. Tollis, 1998

[14] "*A packing problem with applications in lettering of maps*", M. Formann y F. Wagner, 1991.

[15] "*www.americati.com*" AmericaTI EIRL.

[16] "*Lenguajes de Programación*". 2da ed. McGraw-Hill México Allen B. Tucker, Jr., 1992.