Trabajo práctico 1

Fecha de entrega: viernes 8 de abril, hasta las 18:00 hs.

Este trabajo práctico consta de varios problemas y para aprobar el mismo se requiere aprobar todos los problemas. La nota final será un promedio ponderado de las notas finales de los ejercicios y el trabajo práctico se aprobará con una nota de 5 (*cinco*) o superior. De ser necesario (o si el grupo lo desea), el trabajo podrá reentregarse una vez corregido por los docentes y en ese caso la reentrega deberá estar acompañada por un *informe de modificaciones*. Este informe deberá detallar brevemente las diferencias entre las dos entregas, especificando los cambios, agregados y/o partes eliminadas. Cualquier cambio que no se encuentre en dicho informe podrá no ser tenido en cuenta en la corrección de la reentrega. Para la reentrega del trabajo práctico **podrían pedirse ejercicios adicionales**.

Para cada ejercicio se pide encontrar una solución algorítmica al problema propuesto y desarrollar los siguientes puntos:

1. Describir detalladamente el problema a resolver dando ejemplos del mismo y sus soluciones.
2. Explicar de forma clara, sencilla, estructurada y concisa, las ideas desarrolladas para la resolución del problema. Para esto se pide utilizar pseudocódigo y lenguaje coloquial combinando adecuadamente ambas herramientas (**¡sin usar código fuente!**). Se debe también justificar por qué el procedimiento desarrollado resuelve efectivamente el problema.
3. Deducir una cota de complejidad temporal del algoritmo propuesto (en función de los parámetros que se consideren correctos) y justificar por qué el algoritmo desarrollado para la resolución del problema cumple la cota dada. Utilizar el modelo uniforme salvo que se explicite lo contrario.
4. Dar un código fuente claro que implemente la solución propuesta. El mismo no sólo debe ser correcto sino que además debe seguir las *buenas prácticas de la programación* (comentarios pertinentes, nombres de variables apropiados, estilo de indentación coherente, modularización adecuada, etc.). Se deben incluir las partes relevantes del código como apéndice del informe impreso entregado.
5. Realizar una experimentación computacional para medir la performance del programa implementado. Para ello se debe preparar un conjunto de casos de test que permitan observar los tiempos de ejecución en función de los parámetros de entrada. Deberán desarrollarse tanto experimentos con instancias aleatorias (detallando cómo fueron generadas) como experimentos con instancias particulares (de peor/mejor caso en tiempo de ejecución, por ejemplo). Se debe presentar **adecuadamente** en forma gráfica una comparación entre los tiempos medidos y la complejidad teórica calculada y extraer conclusiones de la experimentación.

Respecto de las implementaciones, se acepta cualquier lenguaje que permita el cálculo de complejidades según la forma vista en la materia. Además, debe poder compilarse y ejecutarse correctamente en las máquinas de los laboratorios del Departamento de Computación. La cátedra recomienda el uso de C++ o Java, y se sugiere consultar con los docentes la elección de otros lenguajes para la implementación.

La entrada y salida de los programas **deberá hacerse por medio de la entrada y salida estándar del sistema**. No se deben considerar los tiempos de lectura/escritura al medir los tiempos de ejecución de los programas implementados.

Deberá entregarse un informe impreso que desarrolle los puntos mencionados. Por otro lado, deberá entregarse el mismo informe en formato digital acompañado de los códigos fuentes desarrollados e instrucciones de compilación, de ser necesarias. Estos archivos deberán enviarse a la dirección `algo3.dc@gmail.com` con el asunto *"TP 1: Apellido_1, ..., Apellido_n"*, donde n es la cantidad de integrantes del grupo y *Apellido_i* es el apellido del i-ésimo integrante.

Problema 1: Kaio Ken

Gokú está entrenando para enfrentar a Majin Boo junto con un ejército de guerreros legendarios, entre los que se encuentran entre otros los Saiyajins Vegetta, Gohan, Trunks y Goten. El ejército de Gokú consta de N guerreros (más Gokú). En este momento los guerreros se encuentran entrenando bajo las órdenes de Gokú, quien desea que se enfrenten en pelea para poder comparar sus poderes. Cada pelea constará de dos bandos no vacíos de guerreros que se enfrentan, perteneciendo cada guerrero a exactamente un bando por pelea.

El objetivo de Gokú, que no participa de ninguna pelea, es que todo par de guerreros se enfrenten en al menos una pelea, pero tampoco quiere que peleen demasiado, para no estar cansados a la hora de la batalla contra Majin Boo. Por eso, nuestro amigo Saiyajin nos pide que le digamos cuántas peleas son necesarias durante el entrenamiento y quiénes se enfrentarán en cada pelea.

Se pide escribir un algoritmo que tome el número de guerreros que entrenan bajo las órdenes de Gokú, e indique cuántas peleas deben realizarse en el entrenamiento como mínimo, seguido del bando de cada guerrero en cada pelea. Si hay más de una solución óptima cualquiera de ellas es considerada válida.

El algoritmo debe tener una complejidad temporal $\mathbf{O}(N \cdot \log(N))$, siendo N la cantidad de guerreros

Formato de entrada: La única línea de la entrada contiene un entero positivo N, que indica la cantidad de guerreros, con el siguiente formato:

```
N
```

Formato de salida: La salida debe contener una línea con la mínima cantidad de peleas que deben realizarse en el entrenamiento, seguido por el bando de cada guerrero en cada pelea, con el siguiente formato:

```
P
B(1,1) B(1,2) ... B(1,N)
...
B(P,1) B(P,2) ... B(P,N)
```

donde `P` es la cantidad de peleas y `B(i,j)` es el bando del guerrero `j` en la pelea `i`, siendo `B(i,j)` 1 o 2.

Problema 2: Genkidama

El planeta Tierra está en peligro. N soldados de Freezer están parados en distintos puntos (X_i, Y_i) sobre nuestro planeta y están dispuestos a acabar con toda la humanidad. Para esto, Gokú desea lanzarles algunas Genkidamas que puedan acabar con ellos. Los puntos cumplen con la propiedad $X_1 > X_2 > \ldots > X_N \geq 0$ y $0 \leq Y_1 < Y_2 < \ldots < Y_N$. Gokú lanzará las Genkidamas a puntos donde hay enemigos, por lo que si en un punto no hay un enemigo no puede lanzar una Genkidama a ese punto (sí puede lanzarla si había un enemigo que ya fue destruido por Gokú con una Genkidama previa, para asegurarse que el enemigo no reviva). Una Genkidama lanzada al punto (X, Y) destruye a todos los enemigos que están en el rectángulo con lados paralelos a los ejes y extremos en $(0, 0)$ y $(X + T, Y + T)$. ¿Cuántas Genkidamas necesita lanzar Gokú, como mínimo, para poder acabar con todos sus enemigos?

Se pide escribir un algoritmo que tome el número de enemigos de Gokú, las posiciones de los mismos, y el valor de T, y que indique la mínima cantidad de Genkidamas debe lanzar Gokú para acabar con todos sus enemigos, junto con los índices de aquellos enemigos a cuyas posiciones lanza las Genkidamas. El algoritmo debe tener una complejidad temporal $\mathbf{O}(N)$, siendo N la cantidad de enemigos. Si hay más de una solución óptima cualquiera de ellas es considerada válida.

Formato de entrada: La primera línea consta de un valor entero positivo N, que indica la cantidad de enemigos, y un entero positivo T, que indica que una Genkidama lanzada al punto (X, Y) destruye a todos los enemigos en el rectángulo con lados paralelos a los ejes y extremos en $(0, 0)$ y $(X + T, Y + T)$. A esta línea le siguen N líneas, una para cada enemigo. La entrada contará con el siguiente formato:

```
N T
X1 Y1
```

```
X2 Y2
...
XN YN
```

donde `Xi` e `Yi` son enteros no negativos, decrecientes los `Xi` y crecientes los `Yi`, que indican las posiciones de los enemigos de Gokú.

Formato de salida: La primera línea de la salida debe contener un entero `G` indicando la cantidad de Genkidamas a lanzar por Gokú. A esta línea debe seguirle una línea con los índices `Ei` de los enemigos a los que debe lanzarle las Genkidamas, contando la salida con el siguiente formato:

```
G
E1 E2 ... EG
```

indicando que Gokú debe lanzarle una Genkidama a los enemigos en posiciones (X_{E1}, Y_{E1}), (X_{E2}, Y_{E2}), ..., (X_{EG}, Y_{EG}).

Problema 3: Kamehameha

Gokú se está enfrentando a N androides y necesita destruirlos con la menor cantidad de Kamehamehas posibles. Los enemigos ode Gokú se encuentran en posiciones (X_i, Y_i) y los Kamehameha recorren una semirrecta desde donde Gokú lo lance, en cualquier dirección que Gokú lo decida. ¿Cuántos Kamehamehas necesita Gokú para desturir a todos los androides del doctor Maki Gero?

Se pide escribir un algoritmo que tome la cantidad de androides N y las posiciones (X_i, Y_i) de los mismos y decida cuántos Kamehameha debe lanzar Gokú y a qué enemigos destruye con cada Kamehameha. Si hay más de una solución óptima, el algoritmo puede devolver cualquiera de ellas. Se pide utilizar la técnica de *Backtracking* y elaborar podas y estrategias para mejorar los tiempos de ejecución; éstas deberán estar apropiadamente documentadas en el informe. El algoritmo debe tener una complejidad temporal $\mathbf{O}(N^{N+2})$ o mejor.

Formato de entrada: La primera línea consta de un entero positivo N, que indica la cantidad de androides. A esta línea le siguen N líneas, una para cada androide, cada línea conteniendo dos enteros X_i e Y_i (ambos enteros no negativos y separados por un espacio) indicando la posición del androide en cuestión. La entrada contará con el siguiente formato:

```
N
X1 Y1
X2 Y2
...
XN YN
```

Formato de salida: La primera línea de la salida debe contener un número `M` indicando la cantidad de Kamehamehas que Gokú debe lanzar. A esta línea deberán seguirle M líneas, una para cada Kamehameha, cada línea conteniendo un entero `Ci`, que indique la cantidad de enemigos que derrota con el `i`-ésimo Kamehameha, seguido de `Ci` enteros indicando los índices `Eij` de los enemigos que derrota con el i-ésimo Kamehameha. La salida tendrá el siguiente formato:

```
M
C1 E11 ... E1C1
...
CM EM1 ... EMCM
```

Los números `Eij` deben ser N números entre 1 y N, sin repetidos.