Trabajo práctico 2

Fecha de entrega: viernes 9 de mayo, hasta las 19:00 hs.

Este trabajo práctico consta de varios problemas y para aprobar el trabajo se requiere aprobar todos los problemas. La nota final del trabajo 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 del trabajo. 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 **podrían pedirse ejercicios adicionales**. En caso de reentregar, la nota final del trabajo será el 20 % del puntaje otorgado en la primera corrección más el 80 % del puntaje obtenido al recuperar.

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 sugiere utilizar pseudocódigo y lenguaje coloquial combinando adecuadamente ambas herramientas. Es importante que lo expuesto en este punto sea suficiente para el desarrollo de los puntos subsiguientes, pero no excesivo (**¡no es un código fuente!**).
3. Justificar por qué el procedimiento desarrollado en el punto anterior resuelve efectivamente el problema y demostrar formalmente su correctitud.
4. 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.
5. 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* (nombres de variables apropiados, estilo de indentación coherente, modularización adecuada, etc.). Se deben incluir las partes relevantes del código como apéndice del informe entregado.
6. Presentar un conjunto de instancias que permitan verificar la correctitud del programa implementado cubriendo todos los casos posibles y justificar la elección de los mismos. Ejecutar estos casos de prueba y verificar que el programa implementado dé la respuesta correcta en cada caso.
7. 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 que sean relevantes. Deberán desarrollarse tanto experimentos con instancias aleatorias (detallando cómo fueron generadas) como experimentos con instancias particulares (de peor caso en tiempo de ejecución, por ejemplo). Presentar **adecuadamente** en forma gráfica una comparación entre los tiempos medidos y la complejidad teórica calculada y extraer conclusiones de la experimentación.

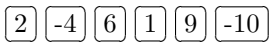
Respecto de las implementaciones, se acepta cualquier lenguaje que permita el cálculo de complejidades según la forma vista en la materia. Además, debe poder compilarse y ejecutarse correctamente en las máquinas de los laboratorios del Departamento de Computación. La cátedra recomienda el uso de C++ 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 2: 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: Robanúmeros

El *Robanúmeros* es un juego de cartas para dos jugadores. Al iniciar se dispone sobre la mesa una secuencia de cartas boca arriba. Cada carta tiene dibujado un número entero (no necesariamente positivo). El *Robanúmeros* se juega por turnos, alternando un turno para cada jugador. En su turno el jugador debe elegir uno de los dos extremos (izquierdo o derecho) de la secuencia que actualmente está en la mesa y robar cartas comenzando por dicho extremo. Puede robar la cantidad de cartas que quiera pero tiene que robar cartas adyacentes a partir del extremo elegido. Por ejemplo, si las cartas en la mesa son



entonces el jugador podría robar las cartas 2, -4 y 6, empezando por la izquierda por ejemplo o bien las cartas -10, 9, 1 y 6 empezando desde la derecha, pero no podría robar las cartas 2, 6 y 1 ya que se estaría salteando la carta -4. Tampoco sería posible robar las cartas 6, 1 y 9 ya que si bien son adyacentes, la secuencia no se inicia en alguno de los extremos.

En su turno, el jugador debe robar al menos una carta y el juego termina cuando ya no quedan cartas en la mesa. En este momento, cada jugador suma los valores de las cartas que robó y el que obtenga una suma mayor gana el juego.

Mingo y Aníbal son dos jugadores expertos de *Robanúmeros* y nos retaron a escribir un algoritmo que juegue tan bien como ellos. Es decir, tenemos que escribir un algoritmo que juegue en forma óptima a este juego suponiendo que su contrincante es tan inteligente como él. El programa a implementar debe tomar una secuencia inicial de cartas y debe indicar qué cartas robarían Mingo y Aníbal en cada turno, asumiendo que ambos juegan de manera óptima a este juego. En caso de haber más de una solución óptima, el algoritmo puede devolver cualquiera de ellas. Se pide que el algoritmo desarrollado tenga una complejidad temporal de peor caso de $\mathbf{O}(n^3)$, donde n es la cantidad de cartas en la secuancia inicial.

Formato de entrada: La entrada contiene una instancia del problema descripta por una línea con el siguiente formato:

```
n c1 c2 ... cn
```

donde `n` es la cantidad de cartas iniciales del juego y `c1`, ... `cn` son los valores (enteros) de las cartas de izquierda a derecha.

Formato de salida: La salida debe comenzar con una linea con el siguiente formato:

```
t p1 p2
```

donde `t` indica la cantidad de turnos en los que se termina el juego según la solución propuesta y `p1` y `p2` son las cantidades de puntos que suman los jugadores 1 y 2, respectivamente. Estos puntajes deben representar el mejor juego que puede hacer el jugador 1 (i.e., el de mayor diferencia a su favor) frente a un jugador 2 que juegue cada turno de manera óptima. A esta línea le siguen `t` líneas, una para cada turno del juego, con el siguiente formato:

```
e c
```

donde `e` es una cadena de caracteres que indica desde qué extremo robó cartas el jugador correspondiente (deberá ser `izq` para izquierda y `der` para derecha) y `c` es la cantidad de cartas que el jugador roba en este turno.

Problema 2: La centralita (de gas)

En una región del país se está considerando realizar una inversión fuerte para proveer de gas natural a un conjunto de pueblos que no disponen aún de este recurso. Para ello, es posible ubicar centrales distribuidoras de gas en algunos de los pueblos y construir tuberías para distribuir el gas de un pueblo a otro. Un pueblo será provisto de gas siempre que exista algún camino por medio de tuberías hasta alguna de las centrales (incluso si este camino pasa por otros pueblos). Debido al elevado costo de construcción de las centrales distribuidoras, el presupuesto con el que se cuenta alcanza para construir a lo sumo k centrales.

Por otro lado, los ingenieros a cargo de este proyecto saben que mientras más larga sea una tubería construida entre dos pueblos, mayor es el riesgo de roturas y escapes de gas durante el trayecto (la longitud de una tubería que conecta dos pueblos está dada por la distancia entre estos dos pueblos). En este sentido, se definió el *riesgo* asociado cada posible plan de construcción, como la mayor de las longitudes de las tuberías construidas en dicho plan.

Se pide escribir un algoritmo que determine un plan de construcción de tuberías y centrales (a lo sumo k centrales) de forma tal que ningún pueblo quede sin acceso al preciado recurso. El plan debe indicar en qué pueblos se instalarán centrales y entre qué pares de pueblos se construirán tuberías de distribución de gas. El plan propuesto debe tener riesgo mínimo y en caso de haber más de un plan óptimo, el algoritmo puede devolver cualquiera de ellos. Se pide que el algoritmo desarrollado tenga una complejidad temporal de peor caso de $\mathbf{O}(n^2)$, donde n es la cantidad de pueblos del problema.

Formato de entrada: La entrada contiene una instancia del problema y comienza con una línea con el siguiente formato:

```
n k
```

donde `n` es la cantidad de pueblos (numerados de 1 a `n`) y `k` es la cantidad máxima de centrales a construir. A esta línea le siguen `n` líneas con dos enteros cada una indicando las coordenadas en el mapa de cada pueblo[1].

Formato de salida: La salida debe comenzar con una linea con el siguiente formato:

```
q m
```

donde `q` y `m` son la cantidad de centrales y de tuberías construidas, respectivamente. A esta línea le sigue una línea con `q` enteros indicando los números de los pueblos en donde se construirán las `q` centrales y a continuación `m` líneas con dos enteros cada una (separados por un espacio) describiendo las `m` tuberías a construir. Para cada tubería, los enteros especifican los números de pueblo entre los que se construirá dicha tubería.

Problema 3: Saltos en *La Matrix*

La Matrix es un reality show que está apunto de aparecer en los televisores de todo el mundo. En este programa, los participantes tienen que moverse por un gran campo de juego cuadrado. Este campo está dividido en celdas que forman una matriz de $n \times n$. En cada celda hay un resorte propulsor que los participantes pueden usar para saltar hacia otras celdas del campo, siendo ésta la única forma de trasladarse de una celda a otra. Los participantes pueden orientar los resortes para saltar hacia el norte, sur, este u oeste, pero no admiten otras orientaciones (diagonales, por ejemplo). Además, cada resorte tiene un cierto número de *unidades de potencia* máximas p que indica la cantidad de celdas que se pueden atravesar saltando desde este resorte; antes de cada salto, el participante puede fijar la potencia del resorte en cualquier valor entre 1 y p. Cada participante comienza el juego en una celda arbitraria denominada el *origen* y el objetivo es llegar hasta otra celda denominada el *destino*, realizando la menor cantidad de saltos posibles.

Los productores del show (que ya cursaron AED3) se dieron cuenta de que encontrar el camino óptimo en este juego iba a ser demasiado fácil e idearon un agregado para que esto no sea tan fácil. Con este agregado, cada participante contará desde el inicio con k unidades extra de potencia para los saltos. Así, antes de cada salto, un participante puede elegir usar algunas de esas unidades para potenciar más aun el salto que va a efectuar. Por ejemplo, si la celda en la que está tiene una potencia máxima de 2 y él quiere saltar 5 celdas al este, puede gastar 3 de sus unidades extra para potenciar este salto. Obviamente, las unidades extra utilizadas se irán descontando de las k con las que empieza cada participante. No hace falta que el participante gaste todas sus unidades de pontecia extra al terminar el juego pero tampoco se beneficia si termina con unidades de sobra.

Más allá de haber hecho este agregado, los productores siguen creyendo que es fácil encontrar una secuencia óptima de saltos, pero aún no lo pudieron demostrar (no les fue taaaan bien en AED3). Por este motivo, nos pidieron escribir un algoritmo que tome los datos necesarios y devuelva una secuencia de

[1] Para evitar problemas numéricos surgidos del cálculo de las distancias, se puede asumir que cuando dos distancias sean distintas, la diferencia entre ellas será significativa.

saltos válida que resuelva el problema en la menor cantidad de saltos posible. En caso de haber más de una solución óptima, el algoritmo puede devolver cualquiera de ellas. Se pide que el algoritmo desarrollado tenga una complejidad temporal de peor caso de $\mathbf{O}(n^3 \cdot k)$, donde n es la cantidad de filas (y columnas) de la matriz y k es la cantidad de unidades de potencia extra disponibles.

Formato de entrada: La entrada contiene una instancia del problema y comienza con una línea con el siguiente formato:

```
n fo co fd cd k
```

donde `n` es la cantidad de filas y de columnas (ambas numeradas de 1 a `n`) de la matriz del campo de juego (i.e., la matriz tiene $n \times n$ celdas), (`fo`,`co`) y (`fd`,`cd`) son las celdas de origen y destino, respectivamente, y `k` es la cantidad de unidades de potencia extra con la que empiezan los participantes. A esta línea le siguen `n` líneas describiendo las unidades de potencia máximas de cada celda del campo de juego. Cada una de estas líneas contiene `n` enteros positivos separados por un espacio, uno para cada celda de la fila en cuestión. Las celdas (1,1) y (`n`,`n`) representan las esquinas nor-oeste y sud-este del campo, respectivamente.

Formato de salida: La salida debe comenzar con un entero indicando la cantidad `S` de saltos de la solución óptima encontrada. A esta línea deben seguirle `S` líneas describiendo cada salto con el siguiente formato:

```
f c e
```

donde `f` y `c` son la fila y la columna de la celda a la cual se efectúa el salto y `e` es la cantidad de unidades de potencia extra que se usaron en dicho salto (este valor será `0` si no se utilizaron unidades extra en el salto).