Trabajo práctico 2

Fecha de entrega: lunes 16 de mayo, 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 debe 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: Una nueva esperanza

Luke Skywalker está entrenando con el maestro Yoda para aprender a manejar la fuerza y así enfrentar al poderoso Darth Vader. Para eso, Yoda le ha encargado que recorra un planeta del sistema Dagobah que está lleno de cuevas y pasadizos. Cada pasadizo conecta exactamente dos dos cuevas entre sí. No hay forma de pasar de un pasadizo a otro sin pasar por una cueva, ni pasar de una cueva a otra sin pasar por un pasadizo. Es bien conocido que desde cualquier cueva se puede llegar a cualquier otra moviéndose por pasadizos (posiblemente pasando en el camino por otras cuevas). Además, se sabe que en el planeta hay N cuevas, numeradas de 0 a $N-1$.

Existen dos tipos de pasadizos: pueden ser *comunes y corrientes* o *especiales*, que son donde se enfrentará a sus mayores miedos. Sin importar de qué tipo sean los pasadizos, atravesar cada uno de ellos le toma a Luke 1 minuto. En este momento, Luke se encuentra en la cueva 0 y debe llegar a la cueva $N-1$, donde lo espera Yoda. Para poder completar su entrenamiento, Luke debe atravesar al menos dos pasadizos especiales. ¿Cuánto tiempo le tomará como mínimo a Luke llegar a la cueva $N-1$ habiendo pasado dos veces por algún pasadizo especial? Notemos que Luke tiene permitido pasar por la cueva $N-1$ y después recorrer dos pasadizos especiales, para volver a la cueva $N-1$; también puede recorrer más de una vez el mismo pasadizo especial si así lo desea.

El algoritmo debe tener una complejidad temporal $\mathbf{O}(N+M)$, siendo N la cantidad de cuevas y M la cantidad de pasadizos.

Formato de entrada: La primera línea consta de un valor entero positivo `N`, que indica la cantidad de cuevas, y un entero positivo `M`, que indica la cantidad de pasadizos.

A esta línea le siguen `M` líneas, una por cada pasadizo, indicando las cuevas `Ai` y `Bi` que conecta dicho pasadizo (`0` $\leq$ `Ai, Bi` $\leq$ `N-1`, `Ai` $\neq$ `Bi` con `Ai` y `Bi` enteros), y un entero `Ei` que indica si es un pasadizo común y corriente (con el valor 0) o si es uno especial (con el valor 1). Se asegura que no habrá dos o más pasadizos que conecten las mismas dos cuevas. Los pasadizos se pueden recorrer en ambos sentidos.

La entrada contará con el siguiente formato:

```
N M
A0   B0   E0
A1   B1   E1
...
AM-1 BM-1 EM-1
```

Formato de salida: La salida debe constar de una línea que indique la mínima cantidad de minutos que le toma a Luke llegar a la cueva $N-1$, seguido de otra línea con una lista ordenada con las cuevas por las cuales se mueve Luke en orden, con el siguiente formato:

```
T
I1 I2 ... I(T-1)
```

donde `T` es el tiempo en minutos que tarda Luke en llegar a la cueva `N-1` e `Ii` es la `i`-ésima cueva que recorre Luke en su camino. De haber más de un camino posible, cualquiera de ellos será aceptado.

Problema 2: El imperio contraataca

Luke Skywalker se acaba de enterar de que, como venganza por la destrucción de la estrella de la muerte, Darth Vader decidió atacar a la alianza rebelde. Luke debe informarle a todos sus aliados para que estén al tanto del ataque, pero esto no será tarea fácil: la alianza se encuentra dividida en distintos planetas. Luke se encuentra con Leia, Han Solo y el poderoso Chewbacca en el planeta Hoth (también conocido como planeta número 0) y deben asegurarse de que todos los demás planetas (numerados de 1 a $N-1$) sepan que el imperio contraatacará.

En cada planeta hay infinitos rebeldes e infinitos *halcones milenarios*. Los *halcones milenarios* son naves espaciales que viajan todas a la misma velocidad y consumen la misma cantidad de litros de combustible cada un millón de kilómetros. Cuando un halcón milenario llega a un planeta con la noticia del ataque, inmediatamente pueden salir rebeldes a los demás planetas para continuar advirtiendo.

Los halcones milenarios no pueden viajar por cualquier lado, porque se pueden dañar con los numerosos meteoritos de la galaxia. Dichas naves sólo pueden viajar por rutas espaciales: cada una de estas rutas conecta exactamente dos planetas. Sabemos que existe al menos una forma de llegar desde cualquier planeta a cualquier otro a través de rutas espaciales.

Como la alianza rebelde no llega a fin de mes (y ella paga por todo el combustible), se quiere consumir la menor cantidad de combustible posible para informar a todos sus miembros de la terrible noticia. Se pide escribir un algoritmo que tome la cantidad N de planetas, la cantidad M de rutas espaciales y los extremos de esas rutas, así como la cantidad de litros de combustible que consume un halcón milenario en hacer cada ruta, e indique la mínima cantidad de combustible necesaria para que toda la alianza se entere de la horrorosa novedad. Se puede asumir que se puede llegar de cualquier planeta a cualquier otro a través de una o más rutas espaciales.

El algoritmo debe tener una complejidad temporal $\mathbf{O}(M \log M)$.

Formato de entrada: La primera línea consta de un entero positivo `N`, que indica la cantidad de planetas, y un entero positivo `M`, que indica la cantidad de rutas espaciales. A continuación de esta línea siguen M líneas con enteros `Ai`, `Bi` y `Li`, siendo `Ai` y `Bi` los extremos de la ruta y `Li` la cantidad de litros que se gastan al recorrer esa ruta (`0` $\leq$ `Ai` $\neq$ `Bi` $\leq$ `N-1`). La entrada contará con el siguiente formato:

```
N M
A0   B0   L0
A1   B1   L1
...
AM-1 BM-1 LM-1
```

Formato de salida: La primera línea debe contener la cantidad mínima de litros `L` necesarios para informar a toda la alianza sobre esta noticia, seguida de `N-1` líneas que indican desde qué planeta se viaja para informar de la situación a cada planeta (el vecino inmediato desde el cual se viaja). El formato debe ser el siguiente:

```
L
I1
I2
...
IN-1
```

indicando que al planeta `i` se le informa de la situación con una nave que parte del planeta `Ii`.

Problema 3: El retorno del ~~que te~~ jedi

Ha llegado la hora de que Luke enfrente a su archinémesis. Para llegar a donde está Darth Vader, Luke debe moverse por una grilla rectangular. Cada casilla de la grilla tiene una altura, posiblemente distinta a la de sus vecinos, y de una casilla sólo se puede mover a otra vecina. Para que dos casillas se consideren vecinas tienen que compartir un lado y no sólo un vértice: cada casilla tiene a lo sumo 4 vecinos.

En este momento Luke se encuentra en la posición $(1, 1)$ y debe llegar a la posición (N, M) para enfrentar a Darth Vader. Como las casillas tienen diferentes alturas, moverse entre ellas puede cansar mucho a Luke por ser un salto muy alto o aterrizaje difícil. Por suerte, sabemos exactamente cuánta energía le cuesta a Luke moverse entre casillas, y sólo depende de la diferencia de altura entre la casilla en la que está y a la que va a moverse (llamemos Δ a esta diferencia). Si $|\Delta| \leq H$ entonces moverse le cuesta 0 energía; de lo contrario, moverse entre estas casillas le cuesta $|\Delta| - H$ energía. H es un número fijo que sólo depende de cuán entrenado está Luke.

Además, Luke sólo quiere moverse en sentido creciente de las X o en sentido creciente de las Y, porque prefiere evitar que armen canciones diciendo que *corre para atrás porque no tiene aguante*.

Como parte de la alianza rebelde, queremos ayudar a Luke a que gaste la mínima energía para llegar hasta Darth Vader, y así poder gastarla toda en derrotarlo (o convencerlo de que vuelva al lado bueno de la fuerza). Se pide escribir un algoritmo que tome el tamaño de la grilla y la altura de cada casilla y devuelva el mínimo costo de energía que es necesario para llegar a la posición de Vader desde $(1, 1)$. Además, deberán imprimir uno de los caminos posibles para llegar a Vader utilizando la menor cantidad de energía.

El algoritmo debe tener una complejidad temporal de $\mathbf{O}(N \cdot M)$.

Formato de entrada:

La primera línea constará de tres valores `N`, `M` y `H`, siendo `N` y `M` la cantidad de filas y columnas de la grilla respectivamente y `H` el valor fijo ya descrito. A continuación de esta línea siguen `N` líneas con `M` enteros cada una, indicando la altura de cada casilla. El formato de entrada será el siguiente:

```
N M H
E11 E12 ... E1M
E21 E22 ... E2M
...
EN1 EN2 ... ENM
```

Formato de salida: La primera línea de la salida debe contener un número `C` indicando el mínimo costo de energía necesario para llegar a la posición de Vader. A continuación de esta línea, debe haber `N+M-2` líneas de manera tal que la `i`-ésima linea contenga la dirección del `i`-ésimo movimiento. Si el movimiento es horizontal (o sea, de la forma $(i, j) \to (i + 1, j)$) la línea debe contener el caracter 'X' en mayúscula y si el movimiento es vertical (o sea, de la forma $(i, j) \to (i, j + 1)$) la línea debe contener el caracter 'Y' en mayścula. La salida tendrá el siguiente formato:

```
C
D1
D2
...
DN+M-2
```

Siendo `Di` la dirección del `i`-ésimo movimiento.