Trabajo práctico 2

Primera entrega: domingo 28 de mayo, hasta las 23:59 horas.
Segunda entrega: domingo 18 de junio, hasta las 23:59 horas.

Este trabajo práctico consta de varios problemas y para aprobar el trabajo se requiere aprobar todos los problemas. Cada problema podrá ser presentado en primera o en segunda fecha y aquellos presentados en primera fecha podrán ser recuperados individualmente.

El trabajo práctico se debe realizar en grupos de a 4 personas inscriptas a la materia (sin excepción). Se deberá notificar la conformación de los grupos a la casilla de mail `algo3.dc@gmail.com` con asunto "Conformación de grupo" hasta el día viernes 5 de mayo. La catedra armará grupos al azar con los alumnos que no esten en grupos de a 4 luego de dicha fecha.

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.
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.).
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.

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. **Se debe incluir un script o Makefile que compile un ejecutable que acepte como entrada lo solicitado en cada problema. Cada problema debe tener un ejecutable con nombre `problemaX` donde X es el número de problema.** La cátedra recomienda el uso de C++ o Java, y se sugiere consultar con los docentes la elección de otros lenguajes para la implementación. **Solo se permite utilizar 1 lenguaje para todo el TP.**

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 anteriores. 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: Delivery óptimo

En una provincia de Optilandia las ciudades están conectadas por rutas bidireccionales. Las rutas no se encuentran en las mejores condiciones, por lo que se decidió mejorar algunas rutas que tendrán categoría *premium*. Estas rutas *premium* muchas veces se encuentran en mejor estado y utilizan caminos más directos que suelen ser más cortos que sus alternativas. Todas las rutas tienen asociado la distancia entre las ciudades que une.

Una empresa de logística, Transportex, tiene la tarea de transportar mercadería desde una ciudad de origen hacia una ciudad de destino. Nos pidió que desarrollemos un software que encuentre el mejor camino entre el par de ciudades, es decir, el que recorra la mínima distancia.

El problema que tienen es que, debido a regulaciones provinciales, las empresas de transporte sólo pueden pasar por k rutas *premium*. Esto lo hacen porque que sino habría que mantener el estado de las rutas con demasiada frecuencia.

Escribir un algortimo que tome los datos de las rutas, la máxima cantidad de rutas *premium* que puede tomar un camión, el origen y destino, y calcule cual es el mínimo costo de una ruta que salga del origen y llegue al destino cumpliendo con las normativas vigentes. El algoritmo implementado debería tener una complejidad no peor que $\mathbf{O}(n^2k^2)$, donde n es la candidad de ciudades y k la máxima cantidad de rutas *premium* que puede utilizar la empresa.

Formato de entrada: La entrada contiene varias instancias del problema. Cada instancia comienza con una línea en la que se indica la cantidad n de ciudades de la provincia y m la cantidad de rutas. A continuación sigue una línea con el siguiente formato:

```
origen destino k
```

donde `origen` es la ciudad donde comienza el recorrido, `destino` es la ciudad donde finaliza y `k` es la máxima cantidad cantidad de rutas *premium* que se pueden utilizar.

Finalmente siguen m líneas representando las rutas con el siguiente formato:

```
c1 c2 p d
```

donde `c1` y `c2` son dos ciudades (numeradas de 1 a n), `p` es un valor binario que indica si la ruta entre `c1` y `c2` es *premium* (si `p` vale `1`) o si no (si `p` vale `0`) y finalmente `d` es la distancia entre las ciudades por esa ruta.

La entrada concluye con una línea conteniendo `-1 -1`, la cual no debe procesarse.

Formato de salida: La salida debe contener una línea por cada instancia de entrada conteniendo la mínima longitud de un camino que une la ciudades pasando por a lo sumo k rutas *premium*.

Problema 2: Subsidiando el transporte

En una provincia de Optilandia las ciudades están conectadas por rutas. En esta provincia decidieron que todas las rutas deben estar mantenidas correctamente. Por esta razón el gobierno provincial decidió agregar cabinas de peajes entre cada par de ciudades.

A medida que se fueron arreglando las rutas y construyendo los peajes se les iba asignando un costo asociado a la distancia de la ruta y de su trafico de tal manera que se logre amortizar el costo de mantener las rutas.

Debido a que se logró pagar el costo de todas las obras rápidamente, el gobierno decidió reducir el costo de todos los peajes por un costo fijo c. El problema es que podrian exisitir peajes que en vez de cobrarle a los vehículos que transiten deberán pagarle. El gobierno consideró que mientras nadie pueda abusar de la situación no tiene inconvenientes en que esto sea así. Se considera un abuso si una persona comienza en la ciudad c y puede hacer un recorrido tal que vuelva a c y haya ganado plata.

Para subsidiar el transporte el gobierno quiere que reducir lo máximo posible el costo de los peajes.

Escribir un algortimo que tome los datos de las rutas con sus peajes y calcule el máximo c tal que nadie pueda sacarle la plata al estado. El algoritmo implementado debería tener una complejidad no peor que

O($nm\ log(c)$), donde n es la cantidad de ciudades, m es la cantidad de rutas y c es el costo del máximo peaje.

Formato de entrada: La entrada contiene varias instancias del problema. Cada instancia comienza con una línea en la que se indica la cantidad n de ciudades y m la cantidad de rutas. A continuación siguen m líneas representando las rutas con el siguiente formato:

```
c1 c2 p
```

donde `c1` y `c2` son dos ciudades (numeradas de 1 a n), `p` es el valor del peaje que une de la ruta entre `c1` y `c2`. La entrada concluye con una línea conteniendo `-1 -1`, la cual no debe procesarse.

Formato de salida: La salida debe contener una línea por cada instancia de entrada conteniendo el máximo valor por el cual se puden disminuir a todos los peajes.

Problema 3: Reconfiguración de rutas

En una provincia de Optilandia, las ciudades están conectadas por rutas. La configuración actual de estas rutas dentro de la provincia no es la ideal: algunas ciudades no están conectadas, y por otro lado hay pares de ciudades entre las cuales se puede viajar de varias maneras (i.e., pasando por ciudades diferentes en el camino). Todo esto dificulta y encarece el control de tránsito que se pretende mantener.

La decisión ejecutiva del gobernador es clara: se harán obras en las rutas de la provincia para lograr que haya una, y sólo una, forma de llegar desde cualquier ciudad a cualquier otra. Para ello se pretende construir nuevas rutas y/o destruir rutas existentes. Por supuesto que tanto la construcción como la destrucción de rutas tiene un costo asociado, bastante alto porque todas las rutas, nuevas o existentes, son doblemano. Afortunadamente se cuenta con estos potenciales costos como dato del problema.

Escribir un algoritmo que tome los datos de las rutas y los costos de construcción y destrucción, e indique un plan de trabajo que logre el objetivo del gobernador gastando la menor cantidad de dinero posible. Entre cada par de ciudades hay una única ruta (existente o por construir, según el caso) que las conecta directamente. El algoritmo implementado debería tener una complejidad no peor que **O**($n^2 log(n)$), donde n es la cantidad de ciudades.

Formato de entrada: La entrada contiene varias instancias del problema. Cada instancia comienza con una línea en la que se indica la cantidad n de ciudades de la provincia y a continuación siguen $\frac{n(n-1)}{2}$ líneas representando las posibles rutas entre cada par de ciuades con el siguiente formato:

```
c1 c2 e p
```

donde `c1` y `c2` son dos ciudades (numeradas de 1 a n), `e` es un valor binario que indica si la ruta entre `c1` y `c2` existe (si `e` vale `1`) o no existe (si `e` vale `0`) y finalmente `p` es el costo de construcción o destrucción (dependiendo de `e`) de la ruta. La entrada concluye con una línea conteniendo `-1`, la cual no debe procesarse.

Formato de salida: La salida debe contener una línea por cada instancia de entrada, con el siguiente formato:

```
p r c11 c12 c21 c22 ... cr1 cr2
```

donde `p` es el costo total de la solución, `r` es la cantidad de rutas que quedaron en la solución y cada par de valores (`ci1`, `ci2`) indica cada una de estas rutas.