Trabajo práctico 2

Primera entrega: viernes 10 de mayo, hasta las 18:00 horas.
Segunda entrega: viernes 24 de mayo, hasta las 18:00 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. En caso de recuperar (o entregar en segunda fecha), la nota final de cada ejercicio será el 20 % del puntaje otorgado en primera fecha (el cual será cero si no se entregó nada) más el 80 % del puntaje obtenido al recuperar (o al entregar en segunda fecha). La nota final del trabajo será el promedio de las notas finales de los ejercicios y el trabajo práctico se aprobará con una nota de 5 (*cinco*) o superior.

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 puede utilizar pseudocódigo o lenguaje coloquial, o combinar 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).
3. Justificar por qué el procedimiento desarrollado en el punto anterior resuelve efectivamente el problema.
4. Encontrar una cota de complejidad temporal del algoritmo propuesto, en función de los parámetros que se consideren correctos. Utilizar el modelo uniforme salvo que se explicite lo contrario.
5. Justificar por qué el algoritmo desarrollado para la resolución del problema cumple la cota de complejidad dada en el punto anterior.
6. Dar un código fuente claro que implemente la solución propuesta. El mismo no sólo debe ser correcto sino que además debe estar bien programado. Si bien no se pide que siga ninguna convención de codificación específica, mínimamente el mismo debe tener nombres de variables apropiados y un estilo de indentación coherente.
7. Dar un conjunto de casos de test que cubran todos los casos posibles del problema justificando la elección de los mismos. No se esperan muchísimos casos ni tampoco muy grandes, a menos que el tamaño sea determinante.
8. Dar un conjunto de casos de test que permitan observar la performance del problema, en términos de tiempo. Para esto se deben desarrollar tanto tests patológicos de peor caso como tests aleatorios (con cierta distribución en función del problema). Se debe explicar cómo los tests cubren los casos posibles del problema. No considerar la entrada/salida al medir los tiempos.
9. Presentar en forma gráfica una comparación entre el tiempo de corrida según la complejidad temporal calculada, tiempo medido de corrida para los tests patológicos de peor caso, y tiempo medido de corrida para los tests aleatorios.

Respecto de las implementaciones, se acepta cualquier lenguaje que permita el cálculo de complejidades según la forma vista en la materia. Además, debe correr 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 la entrada y salida estándar del sistema.

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: Mutaciones bacteriales

El estudio de las mutaciones bacteriales es un trabajo arduo y pesado. Una bacteria al nacer pertenece a una cepa determinada. Luego, con el tiempo la bacteria va mutando de una cepa a otra hasta que en algún momento alcanza una *cepa terminal* desde la cual yo no puede seguir mutando. Durante el proceso, estando en una cierta cepa, la bacteria puede mutar hacia algunas otras cepas con una cierta probabilidad, es decir, no desde cualquier cepa puede mutarse directamente a cualquier otra. Afortunadamente, las posibles mutaciones a partir de cada cepa son datos conocidos.

Se desea realizar un estudio estadístico de este comportamiento. Para ello, se estudiarán algunas bacterias y para cada una de ellas se registrará cada mutación ocurrida desde su nacimiento hasta que la misma deje de mutar. Si bien se sabe a qué cepas puede mutar una bacteria a partir de cualquier cepa no se sabe a priori cuánto puede tardar una bacteria en llegar a su estado final. Por este motivo, antes de comenzar este estudio estadístico se desea calcular la mayor cantidad de mutaciones por las que una bacteria puede pasar y conocer tales mutaciones.

Escribir un algoritmo que tome como parámetro las posibles mutaciones de una cepa a otra y que determine la secuencia más larga de mutaciones por las que una bacteria puede pasar antes de alcanzar una cepa terminal. Notar que, en función de las posibles mutaciones entre cepas, podría darse el caso de que una bacteria no termine nunca de mutar. El algoritmo implementado debería detectar estos casos. Se espera que el algoritmo corra en tiempo lineal en función del tamaño de la entrada. Se puede suponer que todas las cepas participan en al menos una mutación.

Formato de entrada: La entrada contiene varias instancias del problema. Cada instancia comienza con una linea en la cual se indica la cantidad k de cepas posibles de las bacterias y la cantidad m de mutaciones posibles entre las cepas. A esta línea, le siguen m líneas, cada una de ellas con dos enteros c_i y c_j indicando la posible mutación de la cepa c_i a la cepa c_j. Cada cepa se identifica con un entero en el rango $1, \ldots, k$. La entrada concluye con una línea comenzada por `#`, la cual no debe procesarse.

Formato de salida: La salida debe contener una línea por cada instancia de entrada. Si el algoritmo detectó que la secuencia de mutaciones podría ser infinita, esta línea deberá contener el número -1. Si no, la línea deberá contener la secuencia de cepas que representa la mayor cantidad de mutaciones por las que una bacteria podría pasar.

Problema 2: Reconfiguración de rutas

En una provincia de un país en el hemisferio sur, 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 solo 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 de la provincia 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 $\mathbf{O}(n^2 log(n))$, donde n es la cantidad de ciudades de la provincia.

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 comenzada por `#`, 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.

Problema 3: Laboratorio farmacológico

En un laboratorio farmacológico se trabaja con un cierto tipo de virus bastante particular. Cuando una persona contrae este virus, el mismo sufre una adaptación al individuo y muta en cierta forma según el ADN del huesped. Una persona infectada puede contagiar a otra pero sólo si comparten cierta información genética particular, la cual depende del individuo infectado. Por este motivo, podría ocurrir que una persona pueda conagiar a otra pero no al revés.

El director de este laboratorio cuenta con un conjunto de investigadores a quienes debe distribuir en grupos para que trabajen con este virus. El director cuenta con la información del ADN de cada investigador y para cada investigador sabe a qué otros investigadores puede contagiar el primero. A un grupo de trabajo se lo considera de *extremo riesgo* si infectando a cualquiera de sus integrantes existe la posibilidad de que todos los integrantes se contagien. El objetivo del director es armar grupos de trabajo de manera que ningún grupo sea un *extremo riesgo*. Cada grupo trabajará independientemente de los demás, con lo cual no hay riesgo de contagio entre grupos.

Escribir un algoritmo que tome la información mencionada arriba y que detecte todos los grupos de *extremo riesgo* maximales en el conjunto total de investigadores. Notar que no se pide armar los grupos para el director, sino simplemente detectar los grupos de extremo riesgo maximales. Se pide escribir un algoritmo eficiente para resovler este problema. El mismo puede resolverse en tiempo lineal en función del tamaño de entrada.

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 investigadores (numerados de 1 a n) y a continuación siguen n líneas, una para cada investigador i, representando los investigadores que pueden ser contagiados por el investigador i. Estas líneas tendrán el siguiente formato:

```
k j1 j2 ... jk
```

donde `k` es la cantidad de investigadores a los que el investigador i puede contagiar y `j1`, ..., `jk` son estos mismos. La entrada concluye con una línea comenzada por `#`, la cual no debe procesarse.

Formato de salida: Para cada instancia procesada, la salida debe comenzar con una línea en la que se especifica la cantidad k de grupos de extremo contagio maximales detectados y luego debe seguir de k líneas, una para cada grupo, con el siguiente formato:

```
t i1 i2 ... it
```

donde `t` es la cantidad de investigadores de este grupo e `i1`, ..., `it` son los investigadores del mismo.