Trabajo práctico 2

Primera entrega: 11 de mayo de 2012, hasta las 18:00 horas.
Segunda entrega: 1 de junio de 2012, hasta las 18:00 horas.

Nota: El trabajo práctico se aprobará con la nota de 5 (*cinco*). Cada problema podrá ser recuperado individualmente.

En caso de recuperar o entregar en segunda fecha la nota final será dada por el 20 % del puntaje otorgado a los ejercicios en la primera fecha más el 80 % del puntaje otorgado a esos ejercicios en la segunda entrega.

Este trabajo práctico consta de 3 problemas. Para cada uno se pide encontrar una solución algorítmica al problema propuesto y luego desarrollar los siguientes puntos:

1. Explicar las ideas de forma clara, sencilla, estructurada y concisa. 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).
2. Justificar por qué el procedimiento explicado en el primer punto resuelve efectivamente el problema.
3. Encontrar una cota de complejidad temporal del algoritmo propuesto. Los parámetros en función de los cuales dar la cota están detallados en el problema. Utilizar el modelo uniforme salvo que se explicite lo contrario.
4. Justificar por qué el algoritmo presentado en el primer punto cumple la cota de complejidad dada en el punto anterior.
5. 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.
6. Dar un conjunto exhaustivo de casos de test que permitan corroborar en la práctica la correctitud del problema. Exhaustivo quiere decir que se debe explicar cómo los tests cubren los casos posibles del problema, **no** quiere decir que deben ser necesariamente muchísimos casos ni tampoco muy grandes, salvo que el tamaño sea una necesidad para determinar la correctitud.
7. Dar un conjunto de casos de test que permitan observar la performance en términos de tiempo del problema. Para esto se deben desarrollar tanto tests patológicos de peor caso como tests generados sin una intencionalidad.
8. Presentar en forma de gráfico la comparación entre tiempo estimado 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 sin intencionalidad.

Respecto de las implementaciones, es aceptable 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. Java es el lenguaje sugerido por la Cátedra, en la versión instalada en dichos laboratorios. Se recomienda fuertemente consultar si se duda si un lenguaje es aceptable o no.

La entrada y salida debe hacerse por medio de archivos. No se considerará correcta una implementación que no pase los tests que se mencionaron en los puntos anteriores.

Será valorada la presentación de un análisis de casos críticos para los problemas presentados, podas posibles de ser pertinentes y caminos alternativos que llevaron hasta la solución presentada.

Problema 1

Hay un ascensor en planta baja con una capacidad de movimientos limitada (la cantidad de pisos subidos y bajados que puede hacer) como si fuera la energía restante. Se sabe cuánta gente hay en cada piso esperando para bajar a la planta baja, y la capacidad de la cabina en número de personas.

El ascensor sólo se detiene para permitir el ingreso de personas esperando en los pisos o el egreso en planta baja. Las personas sólo ingresan en el ascensor al detenerse y mientras haya capacidad en la cabina.

Calcular la cantidad máxima de personas que puede bajar el ascensor a la planta baja antes de quedarse sin energía.

Entrada `Tp2Ej1.in`
El archivo contiene varias instancias, cada una consta de dos líneas, una contiene dos enteros positivos que expresan la capacidad de la cabina y la energía disponible; y otra contiene un vector de enteros separados por blancos que indican cuántas personas esperan en cada piso, en orden ascendente.

Salida `Tp2Ej1.out`
Para cada instancia de entrada, se debe indicar una línea con la cantidad máxima posible de personas evacuables con la energía indicada.

Problema 2

Una compañía telefónica del Gran Buenos Aires tiene n localidades interconectadas entre sí. Cada localidad puede conectarse con otra mediante uno o más enlaces. Dicha compañía desea realizar la menor inversión posible para asegurarse que la conectividad entre las localidades nunca se pierda mediante el uso de un nuevo tipo de cable subterráneo indestructible.

El costo de mejora depende de cada enlace en particular. Su deber es informar cuáles enlaces mejorar para, en el caso que todos los demás fallen, asegurar que todas las ciudades puedan aún comunicarse.

Entrada `Tp2Ej2.in`
El archivo contiene varias instancias, cada una consta de una línea con ternas de enteros separados por blancos que representan los enlaces, separados a su vez por ;.

Cada instancia será de la forma: `<enlace>[;<enlace>]`

En donde `enlace: <localidad> <localidad> <costo>` y `localidad` es el identificador de cada localidad en el enlace y `costo` el costo de mejoralo.

Salida `Tp2Ej2.out`
Para cada instancia de entrada se debe indicar una línea con los enlaces a mejorar en la forma: `<enlace>[;<enlace>]`

Problema 3

Un sapo se mueve sobre un tronco, sorprendente y arbitrariamente largo, con n piedras, saltando de piedra en piedra, ubicadas en forma lineal, sin superponerse, en posiciones 'enteras' y puntuales con respecto al inicio del tronco. El sapo puede saltar de su piedra actual a otra si y sólo si la distancia entre dichas piedras está entre p y q, constantes enteras positivas inferiores a 10. Es decir, el sapo no puede saltar ni muy poco ni mucho, y ésta es una capacidad genética invariable del sapo.

Dadas dos piedras, x e y, informar el camino para llegar de una a otra.

Entrada `Tp2Ej3.in`
El archivo contiene varias instancias, y cada una consta de tres líneas. En la primera se indican dos enteros para p y q. En la segunda, un vector de enteros separados por blancos que indican las posiciones de las piedras. En la tercera, una susesión de pares de enteros representando a uno o mas pares de x e y de la siguiente forma:

`x y[;`x_i y_i`]`

Salida `Tp2Ej3.out`
Para cada instancia de entrada, se debe indicar una línea con las posiciones de las piedras para llegar de x a y separadas por blancos o una línea en blanco si no es posible.