Trabajo práctico 1

Fecha de entrega: viernes 10 de abril, hasta las 18:00 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**.

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 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 1: 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: Zombieland

¡El país está infestado de zombis! No se sabe exactamente cuándo comenzó la epidemia pero sí se conocen las causas de la misma. Al parecer las primeras infecciones habrían sido causadas por una reconocida bebida cola, al sacar a la venta su nueva línea de bebidas *"Zero"*. Ahora eso ya no importa, lo único que importa es tratar de encontrar la forma de contrarrestar la hecatombe.

En muchas ciudades del país sobreviven cuadrillas de militares dispuestas a acabar con la amenza zombi. Actualmente, estos grupos de soldados se encuentran atrincherados esperando la orden de ataque, ya que se está organizando un ataque coordinado para acabar de una vez con los zombis de todo el país.

Si bien nuestros soldados están altamente capacitados, siempre que el número de zombis en una ciudad supere diez veces el número de soldados en la misma, cualquier intento por limpiar la ciudad será infructuoso. Es decir, para asegurar la limpieza de la ciudad debe haber al menos un soldado por cada 10 zombis que haya que combatir. Afortunadamente, gracias a los enlaces satelitales, sabemos exactamente cuántos zombis y cuántos soldados hay en cada ciudad.

Además de los soldados ya atrincherados en las ciudades, es posible enviar cuadrillas de refuerzo a cualquiera de las ciudades del país, para poder así asegurar la limpieza de la mayor cantidad de ciudades posible. Lamentablemente, el envío de soldados tiene un costo (en recursos) por soldado enviado que depende de la ciudad a la cual se lo envía. Dado que nuestros recursos son acotados, queremos utilizarlos de manera óptima.

Se pide escribir un algoritmo que tome el estado actual del país (cantidad de zombis y soldados por ciudad), el presupuesto disponible y el costo de envío por soldado a cada ciudad del país, y que indique cuántos soldados de refuerzo deberíamos enviar a cada ciudad de manera tal de maximizar la cantidad de ciudades que podrían recuperarse, respetando el presupuesto disponible. Si hay más de una solución óptima, el algoritmo puede devolver cualquiera de ellas. El algoritmo debe tener una complejidad temporal de $\mathbf{O}(n \cdot log(n))$, siendo n la cantidad de ciudades del país.

Además de los ítems pedidos en el enunciado, se pide para este problema, justificar por qué el procedimiento desarrollado resuelve efectivamente el problema **dando una demostración formal de ello**.

Formato de entrada: La primera línea contiene dos enteros positivos, n y P, separados por un espacio, los cuales indican la cantidad de ciudades del país y el presupuesto disponible, respectivamente. A esta línea le siguen n líneas, una para cada ciudad, con el siguiente formato:

```
z s c
```

donde `z` y `s` son la cantidad de zombis y soldados, respectivamente, que hay en la ciudad y `c` es un entero que indica el costo por soldado extra enviado a la misma.

Formato de salida: La salida debe contener una línea con el siguiente formato:

```
C s1 s2 ... sn
```

donde `C` es la cantidad de ciudades que se podrían recuperar siguiendo la solución y `s1`, ..., `sn` son las cantidades de soldados de refuerzo que deberían enviarse a cada una de las n ciudades de la instancia. Es decir, a la ciudad i se enviarán `si` soldados de refuerzo (las ciudades se numeran de 1 a n según el orden dado por la entrada).

Problema 2: Alta frecuencia

Tenemos un enlace a través del cual podemos transmitir información. El enlace utiliza distintas frecuencias para poder ser utilizado en paralelo por distintas aplicaciones. Este enlace tiene un costo por tiempo de uso que depende de la frecuencia utilizada. No todas las frecuencias están disponibles en todo momento, sino que cada frecuencia cuenta con un intervalo de disponibilidad (el cual se conoce de antemano).

Queremos utilizar este enlace para transmitir información secuencial, es decir, no vamos a utilizar distintas frecuencias en paralelo, aunque sí se puede ir cambiando la frecuencia usada a lo largo del tiempo, de ser necesario. El objetivo es transmitir información durante todo el tiempo que sea posible, pero invirtiendo la menor cantidad de dinero. Para ello, se debe seleccionar la frecuencia por la cual transmitir en cada momento con el objetivo de minimizar el costo incurrido por la transmisión.

Se pide escribir un algoritmo que tome el intervalo de disponibilidad y el costo de cada frecuencia y que indique qué frecuencia utilizar en cada momento para transmitir durante todo el tiempo que sea posible minimizando el costo incurrido por la transmisión. El algoritmo debe tener una complejidad temporal de $\mathbf{O}(n \cdot log(n))$, siendo n la cantidad de frecuencias disponibles. Se valorarán las soluciones que utilicen la técnica de *Divide & Conquer*, aunque esto no es una condición excuyente para la aprobación.

Formato de entrada: La primera línea consta de un valor entero positivo n que indica la cantidad de frecuencias disponibles. A esta línea le siguen n líneas, una para cada frecuencia, con el siguiente formato:

```
c i f
```

donde `c` es un entero no negativo que indica el costo de uso por minuto de la frecuencia y los valores `i` y `f` son enteros positivos que indican los tiempos de inicio y fin del intervalo de disponibilidad de la misma. Todos los tiempos se indican en minutos a partir de un momento inicial, el cual se identifica como el minuto 0.

Formato de salida: La primera línea de la salida debe contener un entero `C` indicando el costo incurrido por la solución. A esta línea deben seguirle una línea por cada intervalo de transmisión sobre una misma frecuencia, con el siguiente formato:

```
i f x
```

la cual indica una transmisión desde el minuto `i` hasta el minuto `f` utilizando la frecuencia `x` (numeradas del 1 al `n` según la lectura de la entrada). La salida deberá finalizar con una línea con el valor `-1`.

El siguiente es un ejemplo de una posible salida del algoritmo. La solución indica que se comienza a transmitir en la frecuencia 1 en el minuto 1 con costo de $10 por minuto. En el minuto 6 se decide cambiar a la frecuencia 2 a un costo de $5 por minuto y luego en el minuto 10 se vuelve a transmitir por la frecuencia 1 hasta el minuto 16. En este caso, la salida esperada es:

```
130
1 6 1
6 10 2
10 16 1
-1
```

Problema 3: El señor de los caballos

Tenemos un juego de mesa (para un jugador) que utiliza un tablero de ajedrez de $n \times n$ casillas, para un n variable. Inicialmente algunas de las casillas del tablero están ocupadas con caballos de ajedrez[1] (no más de uno por casilla). Se cuenta además con caballos extra que pueden ubicarse en cualquiera de las casillas libres del tablero con el objetivo de asegurar que todas las casillas del tablero queden *cubiertas*; una casilla se considera *cubierta* si está ocupada o bien si es amenazada por alguno de los caballos ubicados (respetando los movimientos permitidos para un caballo de ajedrez[1]). Las piezas preubicadas no se pueden mover y no hay límite para la cantidad de piezas extra utilizadas.

Se pide escribir un algoritmo que tome las dimensiones del tablero y los caballos preubicados y resuelva el problema utilizando la menor cantidad de caballos extra posible. Si hay más de una solución óptima, el algoritmo puede devolver cualquiera de ellas. Se pide utilizar la técnica de *Backtracking* y elaborar podas y estrategias para mejorar los tiempos de ejecución; éstas deberán estar apropiadamente documentadas en el informe.

Formato de entrada: La primera línea consta de dos valores (separados por un espacio) n y k, donde n es un entero positivo que indica la cantidad de filas (y columnas) del tablero y k es un entero no negativo que indica la cantidad de caballos preubicados en el tablero. A esta línea le siguen k líneas, una para cada caballo, cada línea conteniendo dos enteros f y c (ambos entre 1 y n y separados por un espacio) indicando la posición (fila y columna, respectivamente) del caballo en cuestión.

Formato de salida: La primera línea de la salida debe contener un número `m` indicando la cantidad de caballos extra utilizados en la solución. A esta línea deberán seguirle m líneas, una para cada caballo extra utilizado, cada línea conteniendo dos enteros f y c (ambos entre 1 y n y separados por un espacio) indicando la posición (fila y columna, respectivamente) del caballo en cuestión.

[1] `http://es.wikipedia.org/wiki/Caballo_(ajedrez)`.