Trabajo práctico 1

Fecha de entrega: viernes 11 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**. En caso de reentregar, la nota final del trabajo será el 20 % del puntaje otorgado en la primera corrección más el 80 % del puntaje obtenido al recuperar.

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 sugiere utilizar pseudocódigo y lenguaje coloquial combinando adecuadamente 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 fuente!**).
3. Justificar por qué el procedimiento desarrollado en el punto anterior resuelve efectivamente el problema y demostrar formalmente su correctitud.
4. 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.
5. 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* (nombres de variables apropiados, estilo de indentación coherente, modularización adecuada, etc.). Se deben incluir las partes relevantes del código como apéndice del informe entregado.
6. Presentar un conjunto de instancias que permitan verificar la correctitud del programa implementado cubriendo todos los casos posibles y justificar la elección de los mismos. Ejecutar estos casos de prueba y verificar que el programa implementado dé la respuesta correcta en cada caso.
7. 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 que sean relevantes. Deberán desarrollarse tanto experimentos con instancias aleatorias (detallando cómo fueron generadas) como experimentos con instancias particulares (de peor caso en tiempo de ejecución, por ejemplo). 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: Camiones sospechosos

En un puesto de control de camiones se realizan inspecciones a los camiones que pasan, para controlar que los mismos cumplan con las normas de higiene y seguridad establecidas. Uno de los inspectores sospecha que la empresa "il Ravioli" está traficando sustancias ilegales y desea aprovechar los controles a estos camiones para verificar su sospecha. Para ello, va a contratar a un experto en estos temas durante un tiempo con el objetivo de detectar algún camión de esta empresa que lleve una carga ilegal. El problema es que sólo puede contratar a este experto por un período fijo de D días consecutivos. El inspector sabe qué día va a estar llegando al puesto de control cada camión de "il Ravioli" y desea elegir un intervalo de D días de manera tal de maximizar la cantidad de camiones inspeccionados durante ese período.

Escribir un algoritmo que tome la información sobre las llegadas de los camiones y el valor de D e indique un período de D días que maximice la cantidad de camiones inspeccionados. Para simplificar el manejo de los datos, los días de llegada de los camiones se representarán con números enteros positivos. Se pide que el algoritmo desarrollado tenga una complejidad temporal **estrictamente mejor que $\mathbf{O}(n^2)$**, donde n es la cantidad de camiones del problema.

Formato de entrada: La entrada contiene varias instancias del problema. Cada instancia consta de una línea con el siguiente formato:

```
D n d1 d2 ... dn
```

donde `D` son los días del período de contratación del experto, `n` es la cantidad de camiones que llegarán y `d1`, ..., `dn` son los días de llegada (no necesariamente en orden cronológico) de los camiones. Todos los datos son enteros positivos. La entrada concluye con una línea comenzada por `0`, la cual no debe procesarse.

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

```
d c
```

donde `d` es el día inicial del período de contratación para el experto y `c` es la cantidad de camiones que llegarían en dicho período. Si hay más de una solución óptima, el programa puede devolver cualquiera de ellas.

Problema 2: La joya del Río de la Plata

Estamos en 1886 y nuestro amigo y fabricante de joyas Frank O. Yar, se ha comprometido a fabricar y vender una serie de piezas de orfebrería. Frank tiene un problema ya que el valor al cual puede vender sus obras depende mucho del precio del oro y la plata y, a raíz del hallazgo de los yacimientos de *Witwatersrand* en Sudáfrica[1], estos metales se están devaluando día a día. Esto significa que por cada día que Frank no entrega una determinada pieza i, se pierde de cobrar una cantidad d_i de dinero (relacionada a la cantidad de metal en la pieza i). Frank sabe, para cada pieza i, la cantidad t_i de días que tarda en fabricar dicha pieza, y quiere saber en qué orden debería fabricarlas todas, de manera de perder la menor cantidad de ganancia posible. Frank sólo puede trabajar en una pieza al mismo tiempo.

Escribir un algoritmo que tome los datos sobre las piezas a fabricar y ordene la secuencia de trabajos de forma tal de minimizar las pérdidas del joyero (y por lo tanto maximizar sus ganancias). Se pide que el algoritmo desarrollado tenga una complejidad temporal de $\mathbf{O}(n^2)$, donde n es la cantidad de piezas a fabricar.

Formato de entrada: La entrada contiene varias instancias del problema. Cada instancia comienza con una línea indicando el número (entero positivo) n de piezas a fabricar. A esta línea le siguen n líneas con el siguiente formato:

```
di ti
```

donde `di` y `ti` son números enteros positivos que representan la pérdida diaria de ganancia y el tiempo

[1] https://en.wikipedia.org/wiki/Witwatersrand

de fabricación (en días) de la pieza i, respectivamente, para $i = 1, \ldots, n$. La entrada concluye con una línea comenzada por `0`, la cual no debe procesarse.

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

```
i1 ... in P
```

donde `i1`, ..., `in` son los números de las piezas de la instancia (numeradas desde 1) en el orden dado por el algoritmo y `P` el monto total perdido de la ganancia, según el orden dado. Si hay más de una solución óptima, el programa puede devolver cualquiera de ellas.

Problema 3: Rompecolores

Se tiene un tablero de $n \times m$ casilleros cuadrados en los que pueden ubicarse ciertas piezas. Estas piezas son cuadradas y cada uno de sus lados está pintado de un color. Cualquier pieza puede ubicarse en cualquier casillero, pero dos piezas pueden ubicarse en casilleros adyacentes sólo si el color en los lados adyacentes de las piezas coincide. Las piezas están dadas en una cierta orientación y no pueden rotarse para ser ubicadas en el tablero. La Figura 1 muestra un ejemplo con un tablero de 3×3 y una posible solución que ubica todas las piezas en el tablero.

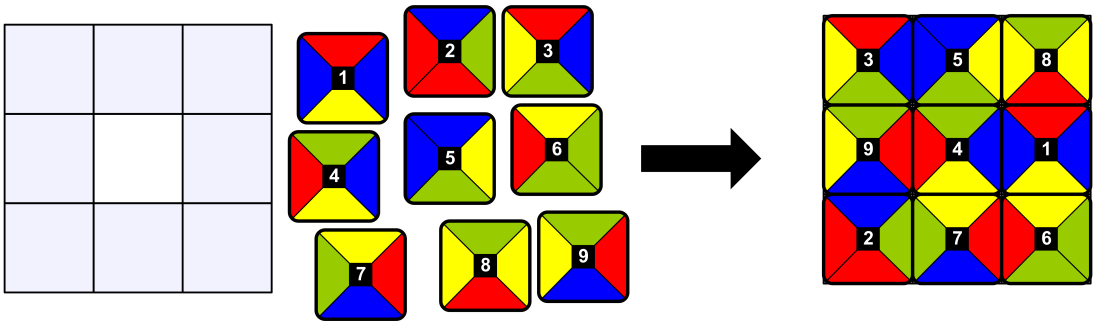

Figura 1: Un tablero de ejemplo de 3×3 y una solución óptima del mismo.

Se debe escribir un algoritmo que tome un tablero y un conjunto de piezas (tantas como casilleros haya en el tablero) e indique dónde debe ubicarse cada pieza de forma tal de cubrir la mayor cantidad de casilleros del tablero. Se pide utilizar la técnica de *Backtracking* y elaborar podas para mejorar los tiempos de ejecución. Estas podas deberán estar apropiadamente documentadas en el informe.

Para simplificar el manejo de los datos de entrada y salida, representaremos a cada color con un número entero positivo.

Formato de entrada: La entrada contiene una instancia del problema. La primera línea contiene 3 números enteros positivos n, m y c, separados por un espacio, indicando la cantidad de filas y columnas del tablero y la cantidad de colores distintos en las piezas, respectivamente. A esta línea le siguen $n \times m$ líneas, cada una describiendo una de las piezas disponibles. Para cada pieza, la línea correspondiente tiene el siguiente formato:

```
sup izq der inf
```

donde `sup`, `izq`, `der` e `inf` son números enteros (entre 1 y c) que indican los colores de los lados superior, izquierdo, derecho e inferior, respectivamente, de la pieza en cuestión.

Formato de salida: La salida debe contener n líneas, cada una con m valores enteros separados por espacios, indicando el contenido de cada una de las $n \times m$ casillas del tablero. Para cada casilla se deberá indicar el número de la pieza que se ubicó en la misma (las piezas se numeran de 1 a $n \times m$ según el orden dado por los datos de entrada), o un `0` si la casilla se dejó vacía. Si hay más de una solución óptima, el programa puede devolver cualquiera de ellas.