Trabajo práctico 1

Fecha de entrega: 6 de septiembre, hasta las 18:00 horas.
Fecha de reentrega: Una semana después de devueltas las correcciones.

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. 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: Pascual y el correo

En un centro de distribución del correo se reciben paquetes todos los días y al final del día todos los paquetes recibidos son envíados en camiones hasta la sede central. Cada paquete tiene un peso determinado y los camiones tienen un límite en la cantidad de peso que pueden transportar (el mismo límite para todos los camiones). El encargado de logística, Pascual, tiene un sistema que utiliza desde hace años y que (según él) sirve para agilizar la carga de los camiones asegurando el uso de una baja cantidad de camiones para el envío de paquetes al final del día. Al llegar un paquete, Pascual intenta ubicarlo en algún camión que ya tenga paquetes dentro, eligiendo entre éstos el que menos peso esté cargando hasta ese momento. Si el peso del paquete permite cargarlo en ese camión, se lo carga allí, si no, se utiliza un nuevo camión para este nuevo paquete. Todos en la empresa saben que el sistema de Pascual es bastante malo, pero como el hombre tiene muy mal genio nadie se interesó en contradecirlo, y no será ese nuestro trabajo tampoco.

Muchas veces llegan varios paquetes al mismo tiempo y éstos se encolan hasta que Pascual termine de acomodarlos todos en el orden de llegada. El gerente de operaciones de la empresa necesita saber cuánto peso va a estar transportando cada camión al final del día. El problema es que éste es un dato que sólo puede saber cuando Pascual termina de acomodar todos los paquetes.

Se desea escribir un algoritmo que tome los pesos de los paquetes que hay que acomodar e indique cuántos camiones se van a utilizar al final del día y cuánto peso se cargará en cada camión, si se utiliza el sistema de almacenamiento de Pascual. Todos los camiones tienen el mismo límite para su carga y éste será también un dato de entrada. Se puede suponer que la cantidad de camiones disponibles alcanza para transportar todos los paquetes y que el peso de un paquete no supera la capacidad de carga de los camiones. El espacio que ocupan los paquetes tampoco es importante siendo los pesos el único limitante, tal como se explicó arriba. El gerente desea que la complejidad temporal del algoritmo implementado sea **estrictamente mejor que** $O(n^2)$, donde n es la cantidad de paquetes a acomodar. El algoritmo debe suponer que inicialmente no hay ningún paquete cargado en ningún camión.

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

```
L n p1 p2 ... pn
```

donde `L` es el límite de carga de los camiones (el mismo para todos), `n` es la cantidad de paquetes a acomodar y `p1`, ..., `pn` son los pesos de cada paquete en el orden en que deben ser almacenados. Todos los datos son enteros positivos. 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:

```
k c1 c2 ... ck
```

donde `k` es la cantidad de camiones utilizados y `c1`, ..., `ck` es el peso que se cargó en cada uno de los `k` camiones al final del día. Los camiones se numeran en orden de carga de su primer paquete. Es decir, para `i=2,...,k`, al momento de cargar el primer paquete en el camión `i`, los camiones con índice menor que `i` ya tenían al menos un paquete dentro.

Problema 2: Profesores visitantes

El Departamento de Computación de la FCEyN extendió una invitación a un grupo de profesores extranjeros para realizar un ciclo de cursos dentro del programa de Profesores Visitantes. Según su disponibilidad, cada profesor estipuló una fecha factible para venir a dar su curso y estas fechas son inamovilbes. Los organizadores de este ciclo de cursos quieren que los alumnos puedan cursar sin problemas todos los cursos del ciclo. Lamentablemente algunas de las fechas propuestas por los profesores se solapan y por lo tanto no sería posible dar todos los cursos en este ciclo. El objetivo de los organizadores es elegir la mayor cantidad de cursos, dentro de los ofrecidos, que puedan darse en este ciclo sin que las fechas de los cursos se solapen entre sí.

Escribir un algoritmo que tome las fechas ofrecidas por los profesores para sus cursos e indique qué cursos deberían elegirse para maximizar la cantidad de cursos en el ciclo. Para simplificar el manejo de los datos, los días de inicio y fin de cada curso se representarán con números enteros positivos. Se pide que el algoritmo desarrollado tenga una complejidad **estrictamente mejor que** $O(n^2)$, donde n es la cantidad de cursos totales del problema.

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

```
n i1 f1 i2 f2 ... in fn
```

donde `n` es la cantidad de cursos ofrecidos por los profesores (numerados de `1` a `n`) y los valores `[i1,f1]`, ..., `[in,fn]` representan los días de inicio y fin de cada uno de los `n` cursos. Todos los datos son enteros positivos. 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 en la que se listen los números de los cursos elegidos para el ciclo de cursos.

Problema 3: Una noche en el museo

El jefe de seguridad de un museo quiere instalar un nuevo sistema de detección de movimientos basado en sensores láser para utilizar durante la noche. Para ello, se ha modelado cada piso del museo con una grilla de $n \times m$ casillas, donde cada casilla representa un metro cuadrado del piso modelado. Algunas casillas estarán libres y otras estarán ocupadas por paredes. El objetivo es que cada casilla libre en la grilla quede cubierta por al menos un sensor de movimiento. Para ello, se cuenta con sensores láser que pueden instalarse en cualquiera de las casillas libres. Un sensor cubre la casilla en la que se lo ha instalado y además emite señales láser que le permiten cubrir las casillas libres de la misma fila y/o columna, dependiendo del tipo de sensor que sea:

- Sensores bidireccionales: un sensor de este tipo emite 2 señales láser y se lo puede ubicar de manera que emita señales hacia la izquierda y la derecha o bien hacia arriba y abajo. Cada sensor de este tipo cuesta $4.000.
- Sensores cuatridireccionales: un sensor de este tipo emite 4 señales láser, una en cada dirección. Cada sensor de este tipo cuesta $6.000.

Las señales láser emitidas desde una casilla cubrirán todas las casillas libres en la dirección de emisión hasta que se encuentren con una pared. Está completamente prohibido ubicar un sensor en forma tal que alguna de sus señales impacte contra otro de los dispositivos, ya que esto puede inhabilitar uno o ambos sensores. No es posible tampoco instalar más de un dispositivo en una misma casilla libre.

Algunos sectores del museo son muy importantes y se desea reforzar su seguridad. Por este motivo, se pedirá que algunas casillas sean cubiertas por dos sensores en lugar de uno sólo. La Figura 1 muestra un ejemplo de este problema y dos soluciones al mismo. Las casillas marcadas con un rombo son las casillas importantes. La primera solución tiene un costo de $44.000 mientras que la segunda tiene un costo de $42.000.

Escribir un algoritmo que lea la descripción de un piso del museo y, en caso de haber solución, indique en dónde deberían ubicarse sensores y de qué tipo, de manera tal de cubrir todas las casillas del tablero adecuadamente (teniendo en cuenta, obviamente, las casillas importantes). La solución provista debe minimizar la cantidad de dinero invertido en sensores. El programa deberá detectar el caso en que no exista una solución al problema e informarlo. Por otro lado, en caso de haber más de una solución mínima, cualquiera de ellas es aceptable. 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.

Formato de entrada: La entrada contiene una instancia del problema. La primera línea indica las dimensiones n y m de la grilla. A esta línea le siguen n líneas, cada una con m valores separados por espacios, indicando el contenido de cada una de las $n \times m$ casillas de la grilla, donde un `0` representa una pared, un `1` representa un casillero libre común y un `2` representa un casillero libre importante.

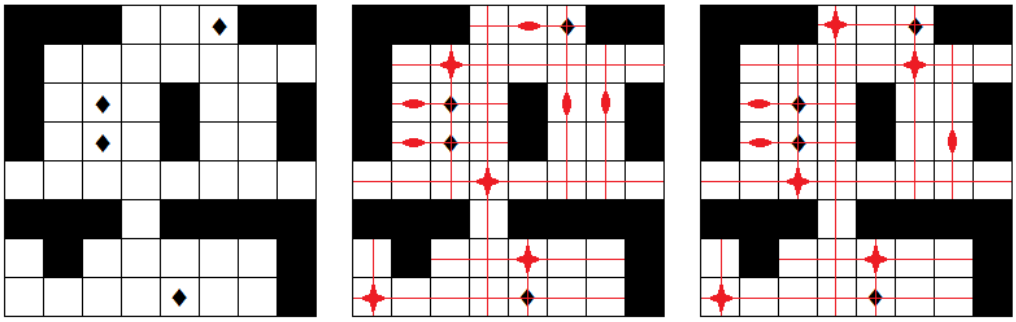

Figura 1: Una instancia del Problema 3 y dos soluciones distintas.

Formato de salida: Si el problema no tiene solución, la salida deberá contener únicamente una línea con el valor `-1`. Si por el contrario el problema admite al menos una solución, la salida debe comenzar con dos números enteros:

```
S C
```

donde `S` es la cantidad de sensores utilizados y `C` es el costo total de la solución. Luego, para cada sensor utilizado, debe haber una línea con el siguiente formato:

```
t f c
```

donde `t` es el tipo de sensor y `(f,c)` son la fila y columna en donde éste se ubica (la esquina superior izquierda de la grilla es la posición $(1, 1)$ y la inferior derecha es la (n, m)). Para los sensores cuatridireccionales, `t` debe valer `1`, para los bidireccionales en forma horizontal `t` debe valer `2` y en forma vertical `3`. Si hay más de una solución óptima, el programa puede devolver cualquiera de ellas.