Trabajo práctico 1

Primera entrega: viernes 12 de abril, hasta las 18:00 horas.
Segunda entrega: viernes 26 de abril, hasta las 17:00 horas.

Aclaraciones generales

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 junto con un CD, DVD, pendrive o similar conteniendo los códigos fuentes desarrollados e instrucciones de compilación, de ser necesarias.

Problema 1: La fábrica de quesos

En una fábrica de quesos hay n máquinas que trabajan en forma independiente, cada una produciendo un determinado tipo de queso. Una vez encendida, la máquina i tarda p_i minutos en finalizar su producción. Por cuestiones de seguridad, las máquinas comienzan el día con su tanque de combustible totalmente vacío y para que inicien su trabajo es necesario llenar cada tanque. El problema es que se cuenta con un único surtidor para todas las máquinas. Este surtidor demora c_i minutos para cargar el tanque de la máquina i. La máquina i comienza a producir inmediatamente después de llenado su tanque. Se desea encontrar un ordenamiento para la carga de combustible de manera tal que la producción completa de todas las máquinas se termine lo antes posible.

Escribir un algoritmo que resuelva este problema, suponiendo que el tiempo de pasar el surtidor de una máquina a cualquier otra es despreciable. El algoritmo debe devolver el orden en que se cargan las máquinas y el tiempo total en que se termina la producción. Se pide que la complejidad temporal del algoritmo sea a lo sumo de $\mathbf{O}(n^2)$, aunque se valorarán soluciones con mejor cota de complejidad.

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

```
n
c1 c2 ... cn
p1 p2 ... pn
```

donde `n` es la cantidad de máquinas de la fábrica, `c1`, ..., `cn` son los tiempos de carga de combustible para las máquinas y `p1`, ..., `pn` son los tiempos de producción (una vez lleno el tanque) de las máquinas. Todos los datos son enteros no negativos. 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:

```
i1 ... in T
```

donde `i1`, ..., `in` son los números de las máquinas de la instancia (numerados desde 1) en el orden dado por el algoritmo y `T` es el tiempo total de producción según este orden.

Problema 2: Sensores defectuosos

Se tienen n sensores identificados con los índices del 1 al n. El sensor i mide a intervalos regulares de m_i minutos y todas las mediciones son registradas y almacenadas por orden de aparición. Si dos o más sensores miden al mismo tiempo, el orden de almacenamiento de estas mediciones es por orden del índice de los sensores que midieron. En el minuto 0 todos los sensores miden.

Se sabe que hubo un error en la k-ésima medición y se desea saber a qué sensor corresponde la misma. Escribir un algoritmo que resuelva este problema con una complejidad temporal **estrictamente mejor que** $\mathbf{O}(kn)$.

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

```
n k m1 m2 ... mn
```

donde `n` es la cantidad de sensores, `k` > 0 es el índice de la medición fallida y `m1`, ..., `mn` son los intervalos de medición de los sensores `1`, ..., `n` respectivamente. 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 y cada línea debe contener el índice del sensor cuya medición falló.

Problema 3: La caja en el plano

Se tienen k puntos en el plano con coordenadas enteras y se tiene además una caja, representada por un rectángulo de dimensiones dadas. La caja puede ubicarse en cualquier lugar del plano, pero no puede rotarse, es decir, la base de la caja debe quedar paralela al eje x y la altura de la caja debe quedar paralela al eje y. Un punto sobre un borde de la caja se considera dentro de la misma. Se desea ubicar la caja de manera tal que la cantidad de puntos que queden dentro de la caja sea máxima. Escribir un algoritmo que resuelva este problema con una complejidad temporal de a lo sumo $\mathbf{O}(n^3)$.

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

```
k b h
x1 y1 x2 y2 ... xk yk
```

donde `k` es la cantidad de puntos, `b` y `h` son la base y la altura de la caja, respectivamente y cada par de valores `(xi,yi)` indica las coordenadas del `i`-ésimo punto de la instancia. 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:

```
x y p
```

donde `(x,y)` es la posición de la esquina inferior izquierda de la caja y `p` es la cantidad de puntos dentro de la misma.

Problema 4: Puzzle de casilleros

Se tiene un tablero de $n \times m$ casilleros en el cual cada casillero es blanco o negro (no necesariamente alternados). Se tienen además k piezas rectangulares de distintos tamaños, las cuales también tienen casilleros blancos o negros. Una pieza puede ubicarse sobre el tablero siempre y cuando cada casillero de la misma coincida con el casillero del tablero que tiene debajo, aunque dos piezas no pueden estar ocupando el mismo casillero. Para ello, las piezas pueden rotarse a cualquiera de las 4 orientaciones posibles (no pueden voltearse "boca abajo", ya que del otro lado son lisas). Se desea utilizar algunas de las k piezas, de manera tal de cubrir todo el tablero con la menor cantidad de piezas usadas (cada pieza puede ser utilizada una sóla vez).

Escribir un algoritmo que resuelva este problema y devuelva una solución óptima, o que indique que no hay solución si es el caso. En caso de implementar un algoritmo de *backtracking* deben implementarse podas para el mismo.

Formato de entrada: La entrada contiene una instancia del problema. La primera línea indica las dimensiones n y m del tablero y la cantidad k de piezas. A esta línea le siguen n líneas, cada una con m valores separados por espacios, indicando los colores de los $n \times m$ casilleros del tablero, donde un 1 representa un casillero blanco y un 0 representa uno negro. Las líneas que siguen a continuación describen el tamaño y configuración de cada una de las k piezas; para cada pieza i ($1 \leq i \leq k$) se cuenta con una línea indicando el tamaño $n_i \times m_i$ de la misma y a continuación se describen los colores de sus casillas con n_i líneas de m_i valores cada una (siguiendo las mismas reglas que con el tablero).

Formato de salida: Si el problema no tiene solución, la salida deberá contener únicamente una línea con el valor `0`. Si por el contrario el problema admite al menos una solución, la salida debe comenzar con una línea indicando la cantidad p de piezas utilizadas y luego para cada pieza debe haber una línea con el siguiente formato:

```
i r x y
```

donde `i` es el número de pieza, `r` es la cantidad de rotaciones en sentido horario de la pieza original y `(x,y)` es el casillero del tablero en donde se ubica la esquina superior izquierda de la pieza (la esquina superior izquierda del tablero es la posición $(1, 1)$ y la inferior derecha es la (n, m)). Si hay más de una solución óptima, el programa puede devolver cualquiera de ellas.