Algoritmos y Estructuras de Datos III
Segundo Cuatrimestre 2020
Trabajo Práctico 1

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Distanciamiento social

Contexto y motivación

La nueva enfermedad que acecha al mundo trae consigo importantes cambios culturales, y sociales. Por ejemplo, para evitar la rápida propagación del virus, se utiliza el conocido *distanciamiento social* que indica que dos personas no deben estar a menos de dos metros de distancia. En este sentido, gran parte de los locales y negocios de las principales avenidas de la ciudad no pueden reabrir sus puertas porque esto aumentaría las probabilidades de generar un aglomeramiento de gente.

Los distintos sectores del gobierno están evaluando medidas de apertura. Uno de los proyectos que más relevancia ganó en estos últimos tiempos es el denominado *negocio por medio* (NPM). El proyecto de NPM intenta dar una alternativa para abrir los locales que mayor movimiento generan para la economía mientras se mantiene el riesgo de expandir la enfermedad de manera controlada.

El gabinete económico logró clasificar a cada local de las grandes avenidas con un número que indica el beneficio que produce a la economía la apertura de dicho local. Por otra parte, el grupo de expertos de la salud asoció un número a cada local indicando el *valor de contagio* que va a provocar su apertura. La idea es determinar, dada una avenida de la ciudad, cuáles locales abrir de manera tal que **no queden dos locales consecutivos abiertos**, que no se exceda un *límite de contagio* M en los locales abiertos y que se maximice el *beneficio* total.

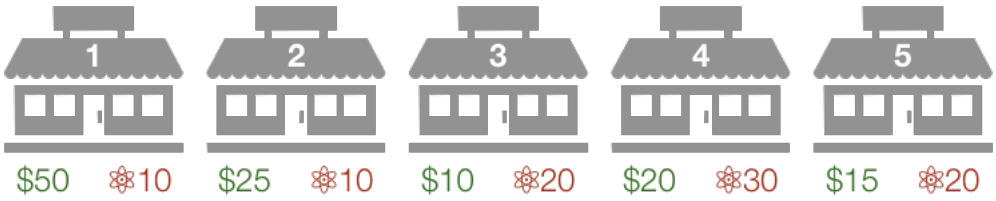

Figura 1: Ejemplo con $n = 5$ locales y $M = 40$ de límite de contagio.

El problema

Dada una secuencia de n locales en orden $L = [1, \ldots, n]$, el beneficio y contagio $b_i, c_i \in \mathbb{N}_{\geq 0}$ de cada local $i \in L$, y el límite de contagio $M \in \mathbb{N}_{\geq 0}$. Una solución *factible* de NPM consiste en un conjunto de locales $L' \subseteq L$ tal que (i) $\sum_{i \in L'} c_i \leq M$ y (ii) no existe $i \in L'$ tal que $i + 1 \in L'$. Se desea definir un algoritmo para encontrar el máximo beneficio que se puede obtener en una solución factible.[1]

La Figura 1 muestra un ejemplo con $n = 5$ locales. Sea $M = 40$, los beneficios $b = [50, 25, 10, 20, 15]$, y los contagios $c = [10, 10, 20, 30, 20]$. La solución óptima tiene beneficio 70, y consiste en abrir

[1] Notar que no es necesario construir la solución, sino, solamente indicar su beneficio.

los locales $L' = \{1, 4\}$. Notar que la solución alternativa $\{1, 2, 4\}$ tiene un beneficio mayor pero no es factible dado que no cumple (ii). La solución alternativa $\{1, 3, 5\}$ también tiene un beneficio mayor pero no es factible porque no cumple la condición (i).

Enunciado

El objetivo del trabajo práctico es resolver el problema propuesto de diferentes maneras, realizando posteriormente una comparación entre los diferentes algoritmos utilizados.

Se debe:

1. Describir el problema a resolver dando ejemplos del mismo y sus soluciones.

Luego, por cada método de resolución:

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.
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.).

Por último:

5. Realizar una experimentación computacional para medir la performance de los programas implementados, comparando el desempeño entre ellos. 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, analizando la idoneidad de cada uno de los métodos programados para diferentes tipos de instancias.

A continuación se listan los algoritmos que se deben considerar, junto con sus complejidades esperadas (siendo n la cantidad de locales a considerar y M el límite de contagio):

- Algoritmo **recursivo** de fuerza bruta. Complejidad temporal perteneciente a $\mathcal{O}(n \times 2^n)$.
- Algoritmo **recursivo** de *Backtracking*. Complejidad temporal perteneciente a $\mathcal{O}(n^2 \times 2^n)$. Se deben implementar dos podas para el árbol de backtracking. Una poda por factibilidad y una poda por optimalidad.

- Algoritmo **top-down** de Programación Dinámica. Complejidad temporal perteneciente a $\mathcal{O}(n \times M)$.

Solo se permite utilizar $c++$ como lenguaje para resolver el problema. Se pueden utilizar otros lenguajes para presentar resultados[2].

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 **con a lo sumo 10 paginas** (sin contar la carátula) que desarrolle los puntos mencionados.

Parámetros y formato de entrada/salida

La entrada consistirá de una primera línea con dos enteros n, M correspondientes a la cantidad de locales en la avenida y a el límite de contagio. Luego le sucederán n lineas, una por cada local $i \in 1, \ldots, n$ que consistirán en dos enteros positivos b_i, c_i indicando el beneficio y valor de contagio.

La salida consistirá de un único número entero que representará el máximo beneficio de una solución factible para el proyecto NPM.

Entrada de ejemplo	**Salida esperada de ejemplo**

`5 40`
`50 10`
`25 10`
`10 20`
`20 30`
`15 20`
	`70`

[2]E.g.: Python, Jupyter, Pandas, Seaborn, etc.