Algoritmos y Estructuras de Datos III
Primer Cuatrimestre 2019
Trabajo Práctico 1

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

Gloppi Ya

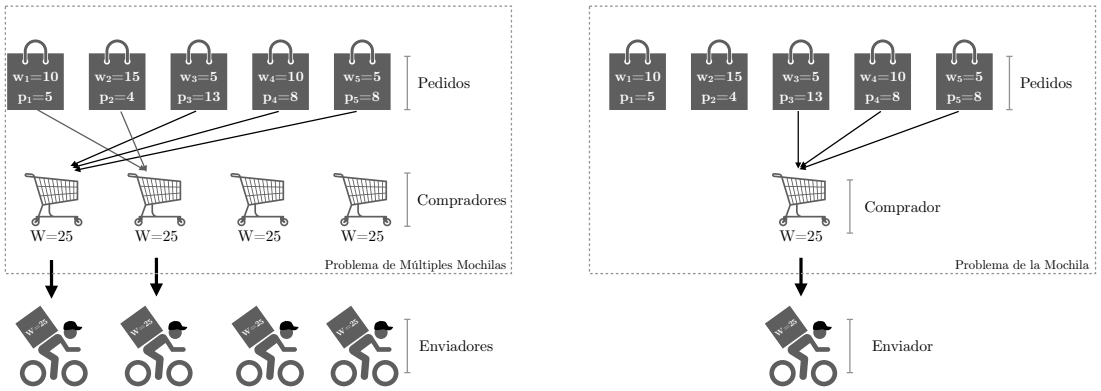
Contexto y motivación

Recientemente hemos sido nombrados jefe de operaciones en supermercados de la flamante unicornio Gloppi Ya. El servicio comienza cuando un cliente realiza una compra de supermercado virtual en nuestra app y solicita un *slot* horario para que esa compra se realice. Los horarios que puede elegir el cliente son períodos de una hora que comienzan en las horas exactas, es decir puede elegir que el pedido se realice entre las 9 y las 10 horas, 10 y 11 horas, y así sucesivamente. En el supermercado tenemos una flota de *compradores* a los cuales les asignaremos los pedidos hechos para que los armen, los paguen y luego los entreguen a nuestros enviadores. Nuestro primer trabajo es buscar una alta eficiencia en la asignación de los pedidos con el fin de poder satisfacer a más clientes con los recursos que tenemos. Para realizar esta asignación podemos abstraer los pedidos como 2 valores de interés p_i y w_i. p_i es el beneficio asociado a realizar el pedido i. w_i es el *tamaño* del pedido i. Denominamos *tamaño* de un pedido a la suma de los artículos que tiene un pedido, ya que este número se asocia al tiempo que le tomará a un comprador para realizar la compra en el supermercado.

Dados todos los pedidos para un horario en particular, la tarea es ver como lo asignamos a nuestra flota de compradores para poder sacar el mayor beneficio posible.

El problema se reduce entonces a asignar un conjunto de ítems con un tamaño y un benificio asociados, a un conjunto de compradores. Este problema se conoce como *Multiple Knapsack Problem* o Problema de Multiples Mochilas. En la Figura 1a se ilustra este escenario para un slot en particular. La resolución exacta de dicho problema no es simple por lo que vamos a realizar la siguiente aproximación. En vez de tratar de asignar los pedidos a todos los compradores, vamos a tomar los compradores de forma sucesiva. Primero le asignaremos los pedidos al primer comprador de la mejor manera posible, con el resto de los pedidos haremos la mejor asignación posible para el segundo comprador, luego para el tercero y así sucesivamente.

Entonces, nuestra misión se reduce a por cada comprador, ver el conjunto de ítems que no fueron asignados a ningún otro y encontrar la mejor asignación posible que se pueda realizar. Un comprador solo puede armar pedidos con una cierta suma total de artículos para que le alcance el tiempo de buscarlos todos y luego ir a la caja a pagarlos. Este problema en particular, es decir, para un solo comprador, se encuentra en la literatura como el *Problema de la Mochila* o *Knapsack problem* en inglés. En la Figura 1b se ilustra un posible escenario para este problema. Dada su importancia tanto de un punto de vista teórico como práctico, existen muchos trabajos abocados a este problema y a sus variaciones. En [3] y [4] se pueden encontrar recopilaciones completas sobre esta categoría de problemas. Es importante notar que los pedidos no pueden partirse. Si se pudiesen partir (realizando por ejemplo solo un quinto de pedido) entonces este problema tendría una resolución exacta golosa como fue demostrado en [1].



(a) Ejemplo del Problema de Mútiples Mochilas. (b) Ejemplo del Problema de la Mochila

Figura 1: Ilustraciones de los escenarios del problema

El problema

Como parte de la solución integral a nuestro problema original, en este trabajo vamos a desarrollar una solución para el Problema de la Mochila.

Dado un conjunto de n pedidos S, cada uno con un *tamaño* asociado w_i y un *beneficio* asociado p_i, y un comprador con una *capacidad* asociada W, encontrar el subcojunto de pedidos de S que maximice el beneficio total sin exceder la capacidad del comprador. Es decir, encontrar $R \subseteq S$ tal que $\sum_{i \in R} p_i$ sea máxima y se cumpla $\sum_{i \in R} w_i \leq W$.

Para este problema, asumiremos que todos los valores mencionados son enteros no negativos.

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 pedidos a considerar y W la capacidad del comprador):

- Algoritmo de fuerza bruta. Complejidad temporal perteneciente a $\mathcal{O}(n \times 2^n)$.
- Algoritmo *meet-in-the-middle* (ver [2]). Complejidad temporal perteneciente a $\mathcal{O}(n \times 2^{\frac{n}{2}})$.
- Algoritmo 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 de Programación Dinámica. Complejidad temporal perteneciente a $\mathcal{O}(n \times W)$.

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

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 12 paginas** que desarrolle los puntos mencionados.

Parámetros y formato de entrada/salida

La entrada consistirá de una primera línea con dos enteros n y W, correspondientes a la cantidad de pedidos disponibles y a la capacidad del comprador, respectivamente. Luego le sucederán n líneas con dos enteros, w_i y p_i, correspondientes a los tamaños y a los beneficios de cada uno de los pedidos.

La salida consistirá de un único número entero que representará el valor máximo de beneficio total alcanzable con una elección óptima de los ítems.

Entrada de ejemplo	Salida esperada de ejemplo
5 25 10 5 15 4 5 13 10 8 5 8	29

Fechas de entrega

- *Formato Electrónico:* Domingo 14 de Abril de 2019, hasta las **23:59 hs, enviando el trabajo (informe + código) a la dirección** `algo3.dc@gmail.com`**. El subject del email debe comenzar con el texto** `[TP1]` **seguido de los apellidos de los alumnos.**
- ***Formato físico:*** **Lunes 15 de Abril de 2019, a las 18 hs. en la clase teórica.**

Importante: El horario es estricto. Los correos recibidos después de la hora indicada no serán considerados.

Referencias

[1] Discrete-variable extremum problems. *Oper. Res.*, 5(2):266–288, April 1957.

[2] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, April 1974.

[3] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems.* Springer, 2004.

[4] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & Sons, Inc., New York, NY, USA, 1990.