Estructuras de Datos.
Facultad de Ciencias, UNAM 2015-2.
Práctica 04. El problema de la mochila.

Armando Ballinas Nangüelú.
armballinas@gmail.com

Fecha de entrega: 24 de marzo de 2015

1. Introducción.

El problema de la mochila es un problema de optimización muy conocido y se encuentra en la clase de los problemas NP-duros. Estos problemas se caracterizan por no tener una solución en tiempo polinomial y más aún, porque todo problema dentro de esta clase es tan "difícil"de ser resuelto como este.

El objetivo de esta práctica es enfrentarse a un problema de esta clase de complejidad y resolverlo de varias maneras.

2. El problema de la mochila.

El problema de la mochila 0-1[1] consiste en lo siguiente:

Tenemos n objetos $X = \{x_1, x_2, \ldots, x_n\}$, cada uno con un peso positivo: $p_i \in \mathbb{N}^+$ y con una ganancia no negativa: $g_i \in \mathbb{R}^+$. También se tiene una mochila que puede soportar cierto peso máximo no negativo: $W \in \mathbb{N}^+$. El problema consiste en encontrar el subconjunto T de objetos cuya ganancia sea máxima sin exceder el peso que puede soportar la mochila, es decir, se busca: $T \subseteq X \mid \sum_{i \in T} p_i \leq W \wedge (\forall\, T' \subseteq X\ (\sum_{j \in T'} p_j \leq W \rightarrow \sum_{j \in T'} g_j \leq \sum_{i \in T} g_i))$

3. Programación dinámica

Este problema puede ser resuelto mediante programación dinámica. Esta es una técnica muy usada en problemas cuya solución depende de soluciones más pequeñas y estas soluciones se utilizan varias veces.

Esta técnica consiste en almacenar estas soluciones parciales en una matriz de modo que la solución total sea una celda de esta matriz.

Para el problema de la mochila se tiene una matriz bidimensional M con $n + 1$ renglones (n es el número de objetos) y W columnas (W es el peso máximo que soporta la mochila). La celda

[1] En inglés: *knapsack* 0-1

$M[i, j]$ representa la ganancia máxima que se puede obtener considerando los primeros i objetos y el peso j de la mochila. Entonces, el valor buscado se encuentra en la celda $M[n, W]$ pues de acuerdo a lo anterior representa la ganancia máxima obtenida con los n objetos y considerando el peso máximo W de la mochila.

La solución óptima se puede obtener de manera recursiva considerando las siguientes decisiones:

1. Si el peso p_i del objeto x_i sobrepasa el valor j de la mochila, entonces no se considera y la solución está dada por la solución óptima considerando los primeros $i - 1$ objetos con la capacidad j, esta solución se encuentra en $M[i - 1, j]$.

2. Si no se da el caso anterior, entonces debemos considerar si conviene meter o no el objeto a nuestra solución óptima. Si no se considera meter el objeto el valor óptimo hasta el momento está en $M[i - 1, j]$ por el argumento anterior. Si sí se considera agregar el objeto, se debe calcular la solución óptima del los primeros $i-1$ objetos, pero ahora con una mochila de peso $j - p_i$ pues se le resta el peso del objeto i. Este valor óptimo se encuentra en $M[i-1, j-p_i]$ debido a la definición de M. A este valor solo debemos sumarle la ganancia del objeto i, es decir la celda $M[i, j]$ tiene valor $g_i + M[i - 1, j - p_i]$.

   De este modo se puede definir el valor de la celda de $M[i, j]$ de la siguiente manera:

$$M(i,j) = \begin{cases} 0 & \text{si } i = 0 \vee j = 0 \\ M(i-1,j) & \text{si } p_i > j \\ \max(P(i-1,j), P(i-1,j-p_i) + g_i) & \text{si } p_i \leq j \end{cases}$$

Con esta forna de calcular el valor de cada celda se puede llenar la matriz renglón por renglón de arriba hacia abajo.

4. Método exhaustivo

El método exhaustivo toma cualquier posible combinación de objetos, calcula su peso y al final recuerda la combinación que maximice la ganancia sin exceder la carga máxima de la mochila. En este método, dado que se verifican todas las combinaciones se deben verificar todos los subconjuntos del conjunto de objetos y sabemos que un conjunto de tamaño n tiene 2^n subconjuntos.

5. Implementación

En la práctica se debe implementar un programa en Java que resuelva el problema de la mochila, es decir, que dados n pesos para los objetos y un peso máximo W de la mochila se obtenga la combinación de objetos que maximice el peso sin exceder la carga total de la mochila, el problema se debe resolver mediante búsqueda exhaustiva y mediante programación dinámica. Y al final se deben comparar los resultados de rendimiento.

5.1. Entrada.

La entrada del programa será mediante entrada estándar. La primera línea será un número entero no negativo n que representa el número de objetos a utilizar. Después se leerán n líneas en donde en cada una habrá dos números, el primero, será un entero no negativo que será el peso del

objeto i y el segundo un número de punto flotante no negativo que será la ganancia por tomar el objeto i. Después se leerá una línea más que contendrá un entero no negativo que representa la carga máxima de la mochila. Un ejemplo de la entrada es el siguiente:

```
10
2 2.3
5 5.24
3 5.6
1 0.23
1 0.92
2 1.63
6 1.0
6 5.6
7 10.3
4 6.75
20
```

5.2. Proceso.

Una vez leída la entrada se debe calcular la solución mediante ambos métodos y se debe medir el tiempo que toma cada uno.

5.3. Salida.

La salida debe ser en la salida estándar y debe mostrar la solución al problema, es decir, el subconjunto de objetos (los objetos son representados por los primeros n números naturales) que maximicen la ganancia sin rebasar la carga de la mochila. Además deben mostrar el tiempo de ambos métodos.

6. Ejercicios.

1. (4 pts.) Realizar la implementación del problema de la mochila usando el método exhaustivo.
2. (5 pts.) Realizar la implementación del problema de la mochila usando el método de programación dinámica.
3. (1 pt.) Realizar un programa de prueba que reciba la entrada y utilice ambos métodos para calcular la solución y la muestre en pantalla.

7. Consejos y observaciones.

1. Observe que para el caso de programación dinámica se debe poder recuperar la solución, es decir, no solo basta saber cuál es la ganancia máxima sino qué objetos logran esa ganancia.
2. La matriz debe de ser eficiente en tiempo y espacio por lo que debe ser implementada en un arreglo lineal mediante un polinomio de direccionamiento. Si se utiliza otra técnica para almacenar u operar la matriz se les bajará puntos en su calificación.

8. Puntos extra.

1. (2 pts.) Responda la siguiente pregunta en el README: Si permitimos que los objetos puedan ser "partidos" de modo que preserven su valor, es decir, si el objeto i pesa p_i y tiene ganancia g_i, podemos partirlo digamos en una tercera parte, de modo que ahora pese $\frac{p_i}{3}$ y tenga ganancia $\frac{g_i}{3}$. Por supuesto se puede partir entre cualquier valor entero. Y la nueva ganancia es proporcional al nuevo peso. Bajo el esquema anterior: ¿El problema sigue siendo NP-Completo? Si crees que el problema deja de ser NP-Completo, da un algoritmo para resolverlo en tiempo polinomial. Si el problema sigue siendo NP-Completo argumenta el porqué.