   
   
  **Unidad Profesional Interdisciplinaria de Ingeniería Campus Zacatecas (UPIIZ)**

**REPORTE DEL EJERCICIO DE LA MOCHILA DINAMICA**

**Materia: Analisis de Algoritmos**

**Instituto Politécnico Nacional**

**Fecha:7/11/2019**

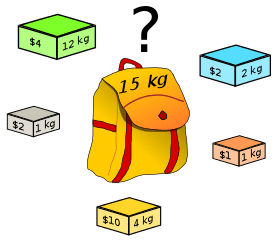
**Alumno: José Moisés Luna Montes  
 No.Boleta: 2017670761**

**Ingeniería en Sistemas Computacionales**

**Grupo:3cm1**

**MARCO TEORICO**

 el problema de la mochila, comúnmente abreviado por KP (del [inglés](https://es.wikipedia.org/wiki/Idioma_ingl%C3%A9s) Knapsack problem) es un problema de [optimización combinatoria](https://es.wikipedia.org/wiki/Optimizaci%C3%B3n_combinatoria), es decir, que busca la mejor solución entre un conjunto finito de posibles soluciones a un problema. Modela una situación análoga al llenar una [mochila](https://es.wikipedia.org/wiki/Mochila), incapaz de soportar más de un peso determinado, con todo o parte de un conjunto de objetos, cada uno con un peso y [valor](https://es.wikipedia.org/wiki/Valor_econ%C3%B3mico) específicos. Los objetos colocados en la mochila deben maximizar el valor total sin exceder el peso máximo.  
Consiste en un excursionista que debe preparar su mochila, la cual tiene una capacidad limitada y por tanto no le permite llevar todos los artículos que quisiera tener en la excursión. Cada artículo que el excursionista puede incluir en la mochila le reporta una determinada utilidad. Luego el problema consiste en seleccionar un subconjunto de objetos de forma tal que se maximice la utilidad que el excursionista obtiene, pero sin sobrepasar la capacidad de acarrear objetos.



En este contexto existen varias aplicaciones que guardan una similitud conceptual con el Problema de la Mochila y en consecuencia nos podemos beneficiar de la formulación y resolución de un modelo de optimización matemática para dicho propósito. Consideremos el siguiente ejemplo:

**DESARROLLO**

Recursos necesarios :  
\*Maquina con neatbeans, eclipse u otro entorno de programación.   
\*Tener instalado Java con sus diversas rutas en path.

**Código en java:**

package mochila;

import java.util.ArrayList;

/\*\*

\*

\* @author Mouxes13

\*/

public class MochilaDinamica {

private ArrayList<Articulo> articulos;

private ArrayList<Articulo> artSolucion;

private int[][] mBeneficios;

private int \_W;

private int maxBenefit;

public MochilaDinamica(ArrayList<Articulo> articulos, int \_W) { //Constructor de la clase con el tamaño \_W y los articulos

this.articulos = articulos;

this.\_W = \_W;

generadorMatriz();

}

public void imprimirArticulos() {

System.out.println("Articulo Valor Peso ");

for (int i = 0; i < articulos.size(); i++) {

System.out.println(" " + i + " " + articulos.get(i).getValor() + " " + articulos.get(i).getPeso());

}

}

private void generadorMatriz() {

this.mBeneficios = new int[this.articulos.size() + 1][this.\_W + 1]; // construimos la matriz de beneficios

for (int i = 0; i <= this.articulos.size(); i++) // agregar en la primer columna puros ceros

{

this.mBeneficios[i][0] = 0;

}

for (int x = 0; x <= this.\_W; x++) // agregar en la primer fila puros ceros

{

this.mBeneficios[0][x] = 0;

}

}

public void buscarSolucion() {

for (int i = 1; i <= this.articulos.size(); i++) {

for (int w = 0; w <= this.\_W; w++) {

if (this.articulos.get(i - 1).getPeso() <= w) { //Validar que el articulos tenga espacio en la mochila

if ((this.articulos.get(i - 1).getValor() + this.mBeneficios[i - 1][w - this.articulos.get(i - 1).getPeso()]) > this.mBeneficios[i - 1][w]) ////Sumamos el valor del articulo y el beneficio maximo anterior y compararlo a ver si es mayor al anterior

{

this.mBeneficios[i][w] = (int) this.articulos.get(i - 1).getValor() + this.mBeneficios[i - 1][w - this.articulos.get(i - 1).getPeso()]; //remplazamos el nuevo mejor beneficio

} else {

this.mBeneficios[i][w] = this.mBeneficios[i - 1][w]; //mantenemos el anterior mejor beneficio

}

} else {

this.mBeneficios[i][w] = this.mBeneficios[i - 1][w]; //mantenemos el anterior mejor beneficio

}

}

}

this.maxBenefit = (int) this.mBeneficios[articulos.size()][\_W]; //regresamos el beneficio maximo

this.artSolucion = new ArrayList<>();

int i = this.articulos.size();

int j = this.\_W;

int pesoT = 0;

//Imprimimos la matriz resuelta

System.out.println("Matriz de Beneficios");

for (int k = 0; k < i + 1; k++) {

for (int l = 0; l < j + 1; l++) {

System.out.print(this.mBeneficios[k][l] + ", ");

}

System.out.println();

}

}

public static void main(String[] args) {

ArrayList<Articulo> articulos = Articulo.generador(10, 20, 20); //Generador de Articulos generador(numero de articulos,rango para Valor, rango para peso

MochilaDinamica m = new MochilaDinamica(articulos, 20); //Mochila(articulos,tamaño \_W).

m.imprimirArticulos();

m.buscarSolucion();

}

}

Conclusión

Concluyo que este ejercicio fue lo suficientemente dinámico en la clase lo que facilito al momento de programación y el programa nos puede ayudar de diversas forma para formar nuestra lógica o completar futuros proyectos.

BIBLIOGRAFIA

<https://www.gestiondeoperaciones.net/programacion-entera/problema-de-la-mochila-en-programacion-entera-resuelto-con-opensolver/> GESTION DE OPERACION 23/01/2015