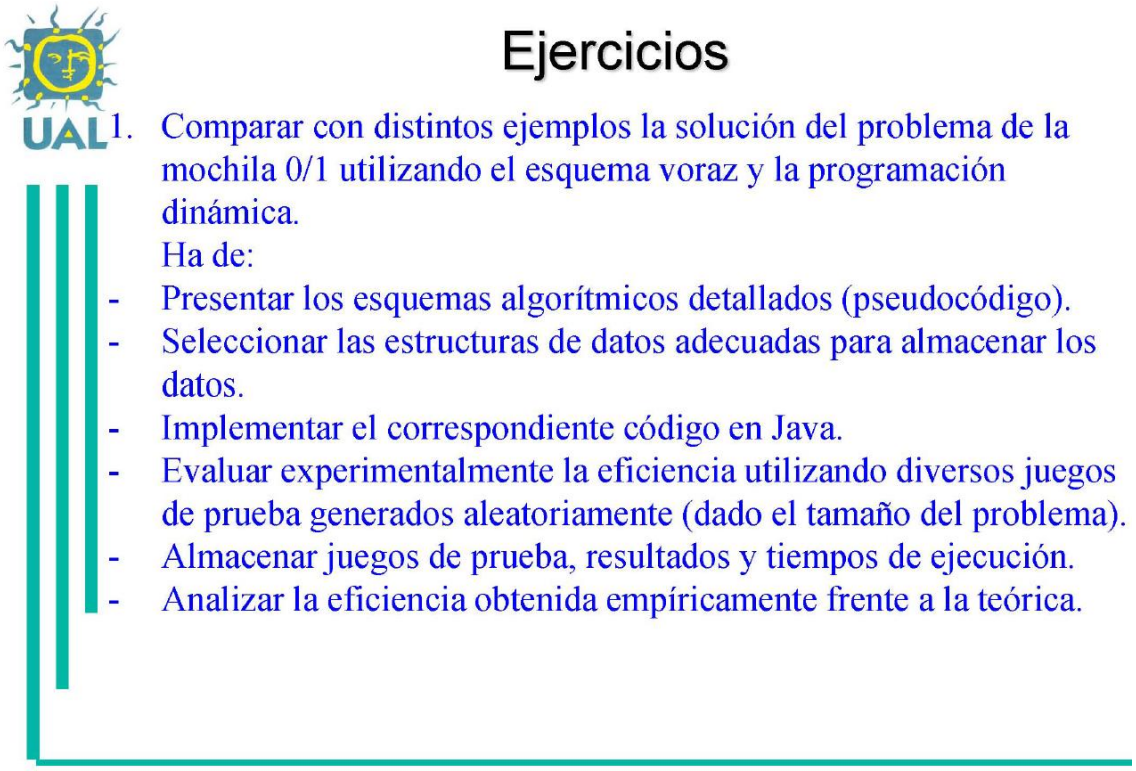
**Autores:** *Germán Ruano García, Francisco Javier Marqués Gaona y David Subires Parra*

# ***Ejercicio 1 Tema 4***

***Ejercicio 1.***



## Esquemas algorítmicos

### Esquema Greedy

```
/*

Suponemos que lista contiene todos los objetos del problema y que capacidad es la capacidad
máxima de la mochila

El método selecciona (lista.siguiente) obtiene el siguiente elemento maximizando la función
valor/peso de cada objeto

*/

Greedy(){
        peso = 0;
        mientras que peso menor que capacidad Y lista no está vacía hacer
                próximo = lista.siguiente()
                si peso + peso(próximo) es menor o igual que capacidad hacer
                        solución.añadir(próximo)
                        peso = peso + peso(próximo)
                fsi
        fmientras
}
```

**Esquema Programación Dinámica**

```
/*

Suponemos que n contiene el número de objetos, capacidad es la capacidad de la mochila,
B[n][capacidad] será la tabla donde iremos almacenando la solución y las funciones peso(0) y
valor(0) nos devuelven el peso y el valor del objeto 0.

*/

PDinamica(){
        B[n][capacidad] enteros
        para w=0 hasta capacidad hacer
                B[0][w] = 0
        fpara
        para k=1 hasta n hacer
                para w=0 hasta peso(k)-1 hacer
                        B[k][w] = B[k-1][w]
                fpara
                para w = peso(k) hasta capacidad hacer
                        B[k][w] = max(B[k-1][w-peso(k)] + valor(k), B[k-1][w])
                fpara
        fpara
}
```

**Estructuras de datos elegidas**

Estructura general del problema

Objeto es una clase que hemos creado para almacenar el valor, el peso, y el nombre de cada objeto candidato a guardar en la mochila.

```
private class Objeto implements Comparable<Objeto> {

    private int valor;
    private int peso;
    private String nombre;

    public Objeto(String n, int v, int p) {
        nombre = n;
        valor = v;
        peso = p;
    }

    @Override
    public String toString() {
        return "Objeto[Nombre:" + nombre + " Valor:" + valor + " Peso:"
                + peso + "]";
    }

    @Override
    public boolean equals(Object o) {
        Objeto ob = (Objeto) o;
        return nombre.equals(ob.nombre);
    }

    public int compareTo(Objeto o) {
        if ( ((double)valor / (double)peso) < ((double)o.valor / (double)o.peso))
            return 1;
        if (((double)valor / (double)peso) > ((double)o.valor / (double)o.peso))
            return -1;
        if(nombre.hashCode() < o.nombre.hashCode())
            return 1;
        if(nombre.hashCode() > o.nombre.hashCode())
            return -1;
        return 0;
    }

}
```

Implementa el método compareTo para poder ordenar conjuntos de esta clase.
La lista de objetos candidatos a guardar en la mochila es un ArrayList<Objeto>.
Sin los dos últimos if del método compareTo no podríamos almacenar objetos con la misma tasa de valor / peso en la estructura TreeSet<Objeto>.

Estructuras escogidas para algoritmo Greedy:

Este algoritmo almacena la solución en un ArrayList<Objeto> el cual contendrá los objetos seleccionados para ser almacenados en la mochila. Además, para este algoritmo hemos usado la estructura TreeSet<Objeto> la cual nos permite ordenar los objetos, al mismo tiempo que los vamos añadiendo, en un tiempo de O(log n) y obtener un elemento en tiempo O(1). Esta estructura se usa para el método selección del algoritmo, el cual nos permite obtener el primer elemento maximizando la función valor/peso.

Estructuras escogidas para algoritmo Programación dinámica:

Este algoritmo usa para almacenar la solución y formar soluciones a partir de soluciones más pequeñas una matriz de enteros B[n][P] donde n es el número de objetos candidatos y P es la capacidad de la mochila.
Además de lo mencionado anteriormente, la tabla o matriz también es usada para reconstruir la solución a través del método obtenerObjetosSeleccionados(int n, int p).

**Evaluar experimentalmente la eficiencia utilizando diversos juegos de prueba generados aleatoriamente**

Para generar aleatoriamente los juegos de prueba hemos empleado la clase GeneradorObjetos.java, la cual nos genera 10 juegos de prueba, inicialmente de tamaño 256 que se irá duplicando en cada juego. La capacidad de la mochila será siempre 11. Los objetos son generados con un valor aleatorio entre 1 y 56 y el peso entre 1 y 11.

**Almacenar juegos de prueba, resultados y tiempos de ejecución**

Los juegos de prueba, resultados y tiempos de ejecución están almacenados en el directorio Datos.

**Analizar la eficiencia obtenida empíricamente frente a la teórica**

Eficiencia teórica:

Algoritmo Greedy **O(n)**:

```
// declaracion e inicializacion de datos
double peso = 0;
ArrayList<Objeto> contenido = new ArrayList<Objeto>();
TreeSet<Objeto> tree = new TreeSet<Objeto>();
for (Objeto o : lista)
    tree.add(o);


// nucelo algoritmo
while (peso < capacidad && !tree.isEmpty()) {
    Objeto o = tree.pollFirst();

    if (peso + o.peso <= capacidad) {
        contenido.add(o);
        peso += o.peso;
    }
}

return contenido;
```

En el peor de los casos, el bucle while se ejecutaría n veces, por lo tanto sería de tiempo lineal, y el TreeSet<Objeto> ordena la lista de objetos en tiempo logarítmico, por lo tanto tendríamos O(n + log n) → O(n)

Algoritmo programación dinámica **O(n*P)**:

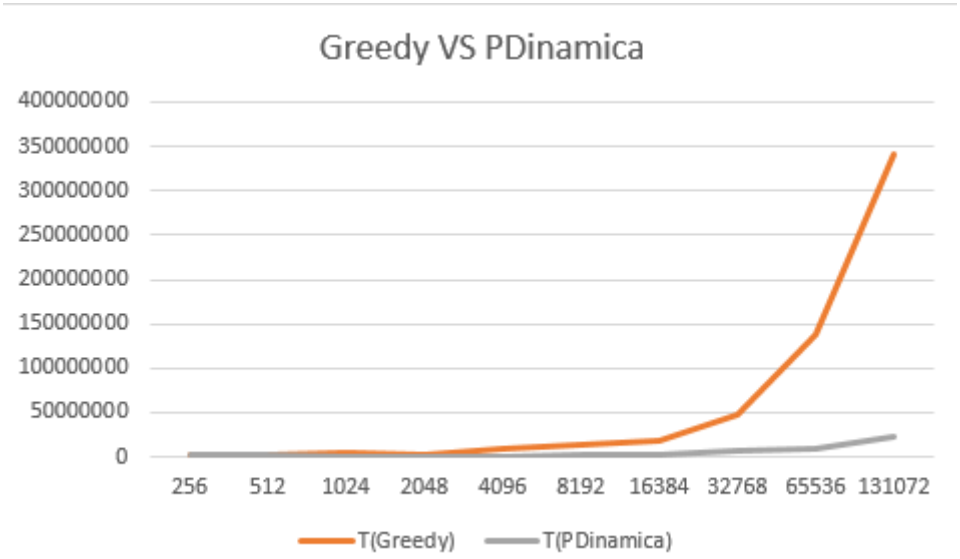
```
for (int w = 0; w <= capacidad; w++)
    B[0][w] = 0;

for (int k = 1; k <= n; k++) {
    for (int w = 0; w <= lista.get(k -1).peso -1; w++)
        B[k][w] = B[k - 1][w];
    for (int w = lista.get(k - 1).peso; w <= capacidad; w++) {
        int temp = B[k - 1][w - lista.get(k - 1).peso] + lista.get(k - 1).valor;
        if (temp > B[k - 1][w])
            B[k][w] = temp;
        else
            B[k][w] = B[k - 1][w];
    }
```

Este algoritmo está basado en cuatro bucles for, dos de ellos anidados, el primero de tamaño n (número de objetos) y el segundo de tamaño P(capacidad de la mochila). Por lo tanto el algoritmo estaría en el orden de O(n*P). Es un algoritmo con tiempo de ejecución "Pseudo-polinómico" (no es polinómico sobre el tamaño de la entrada, es decir, sobre el número de objetos únicamente)

Estos son los datos obtenidos empíricamente, el tiempo de cada algoritmo está expresado en nanosegundos.

| n | T(Greedy) | T(PDinamica) |
|---|---|---|
| 256 | 1362561 | 1879430 |
| 512 | 2067784 | 2882785 |
| 1024 | 4272599 | 1104974 |
| 2048 | 3015197 | 778562 |
| 4096 | 8272161 | 952033 |
| 8192 | 14301015 | 1779864 |
| 16384 | 18135320 | 2520448 |
| 32768 | 46644458 | 7108167 |
| 65536 | 138005186 | 9698413 |
| 131072 | 342213324 | 22577796 |

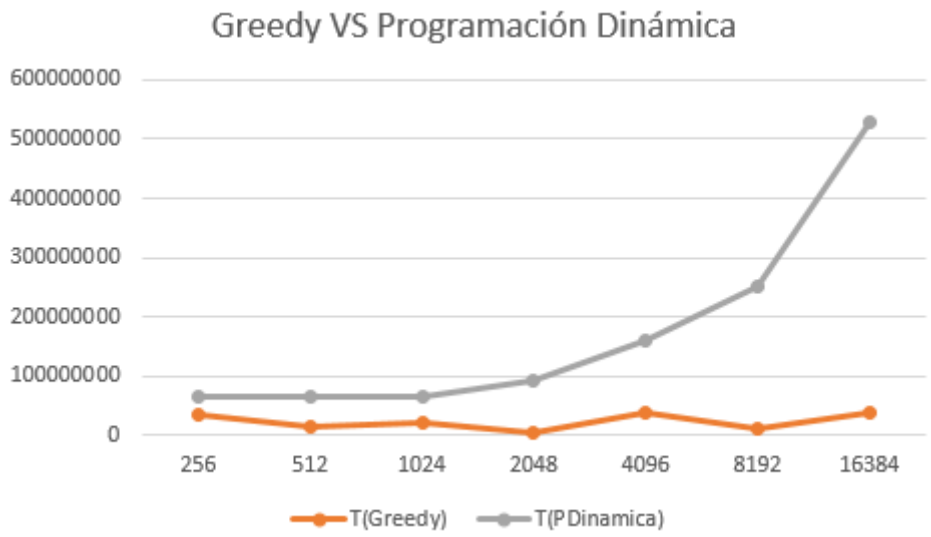
A partir de estos datos obtenemos la siguiente gráfica:




Como podemos comprobar, los datos empíricos y los teóricos coinciden, teniendo el algoritmo Greedy un crecimiento lineal y el algoritmo de programación dinámica un crecimiento lineal también, pero respecto de la capacidad de la mochila, la cual es un valor bajo (11), por eso obtenemos mejores tiempos que con el algoritmo Greedy.

Vamos a comprobar qué ocurre si cambiamos el valor de capacidad de la mochila a uno superior, 3500 por ejemplo:

| tamaño(n) | T(Greedy) | T(PDinamica) |
|---|---|---|
| 256 | 34783827 | 29210203 |
| 512 | 15268445 | 49313228 |
| 1024 | 20843608 | 45401427 |
| 2048 | 4968532 | 87074273 |
| 4096 | 36077667 | 124868678 |
| 8192 | 10941957 | 241446206 |
| 16384 | 36711501 | 490371129 |




Como podemos observar, para tamaños de P (capacidad mochila) grandes, el algoritmo Greedy nos ofrece mejores tiempos que el algoritmo de programación dinámica.

Para comprobar la correcta implementación del algoritmo de programación dinámica, hemos usado como test el ejemplo de ejecución del problema de la mochila del tema 4 en la página 52.

## 4 – Ejemplos: El problema de la Mochila 0/1

UAL

Planteamiento programación dinámica:

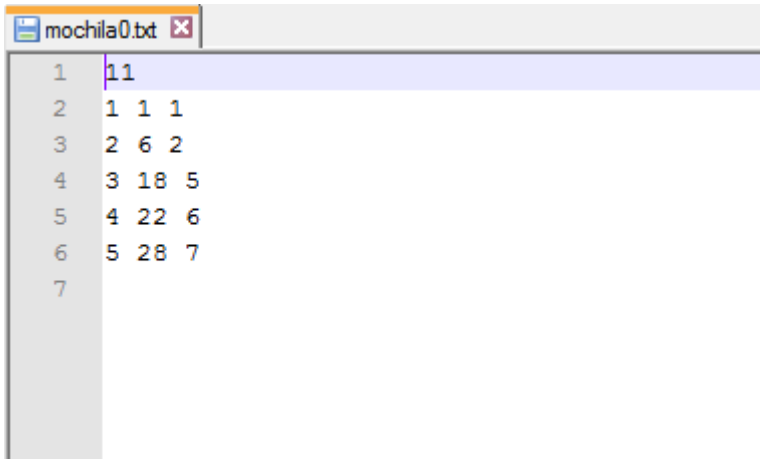
Ejemplo de tabla B:

Mochila de tamaño P = 11
Número de objetos n=5 (ordenados por peso)
Solución óptima {3,4} B = 40

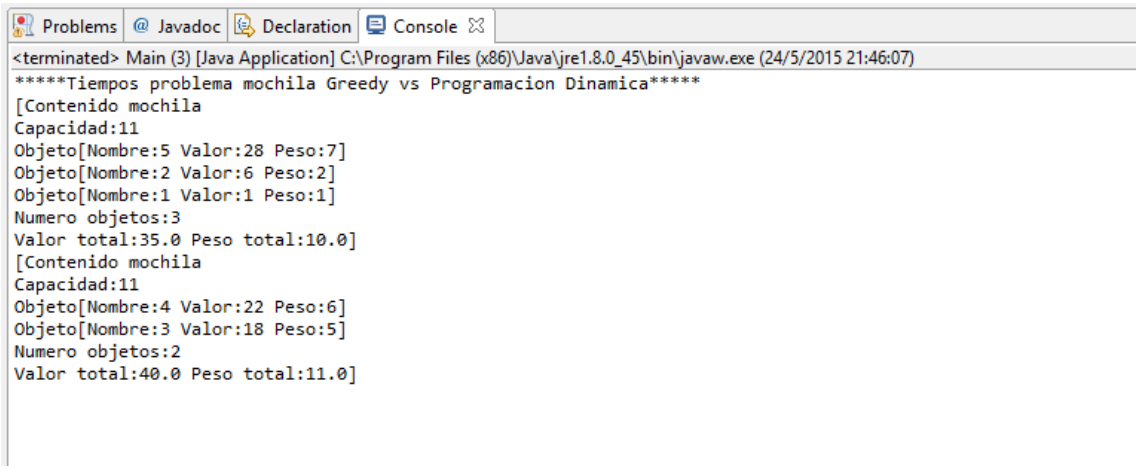
| Objeto | Valor | Peso |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 6 | 2 |
| 3 | 18 | 5 |
| 4 | 22 | 6 |
| 5 | 28 | 7 |

| B | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ϕ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| { 1 } | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| { 1, 2 } | 0 | 1 | 6 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| { 1, 2, 3 } | 0 | 1 | 6 | 7 | 7 | 18 | 19 | 24 | 25 | 25 | 25 | 25 |
| { 1, 2, 3, 4 } | 0 | 1 | 6 | 7 | 7 | 18 | 22 | 24 | 28 | 29 | 29 | 40 |
| { 1, 2, 3, 4, 5 } | 0 | 1 | 6 | 7 | 7 | 18 | 22 | 28 | 29 | 34 | 35 | 40 |

Estos datos están dentro del caso de prueba mochila0.txt:




La ejecución nos devuelve el resultado deseado:




La primera salida es del algoritmo Greedy y la segunda del Programación dinámica.

Como podemos observar, la salida del algoritmo programación dinámica coinciden con el ejemplo, y los valores de las soluciones almacenados en la tabla también:

| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 2 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| 3 | 0 | 1 | 6 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | |
| 4 | 0 | 1 | 6 | 7 | 7 | 18 | 19 | 24 | 25 | 25 | 25 | 25 | |
| 5 | 0 | 1 | 6 | 7 | 7 | 18 | 22 | 24 | 28 | 29 | 29 | 40 | |
| 6 | 0 | 1 | 6 | 7 | 7 | 18 | 22 | 28 | 29 | 34 | 35 | 40 | |
| 7 | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | |