PROBLEMA DE LA MOCHILA

Base teórica

El problema consiste en seleccionar el subconjunto de objetos que haga que se maximice el valor total de la mochila pero sin superar la capacidad de peso máxima que esta admite.

Vamos a resolver cuatro problemas diferentes satisfaciendo una serie de restricciones y pautas. Para ello, recurriremos a la programación dinámica.

La idea básica de la programación dinámica es utilizar una tabla para almacenar las soluciones de los subproblemas resueltos. Si te enfrentas a un subproblema nuevamente, solo necesitas coger la solución de la tabla sin tener que resolverlo nuevamente. Como consecuencia, los algoritmos diseñados por programación dinámica son muy efectivos.

Dimensiones de la tabla = (nº objetos + 1) x (peso mochila + 1)

Las filas de la tabla representan los objetos y las columnas representan los supuestos pesos para mochilas de tamaño x.

En la primera fila no contemplamos objetos por lo que estará rellena de 0 por defecto ya que al crearla usamos double[][]. Como consecuencia, la primera columna también estará rellena de 0.

Los objetos se van a ir valorando de arriba abajo y se van a ir acumulando, es decir, por ejemplo, en la fila 3 que corresponde al objeto 3 se valorarán los objetos 1, 2 y 3.

Por un tema de optimización, los objetos los ordenamos de menor a mayor por el peso (si este coincide los comparamos por el valor) antes de empezar el algoritmo.

Sin embargo, si no se ordenan el algoritmo funcionaría igual.

Teoría rellenar tabla

Para obtener el valor de cada celda el algoritmo sigue:

B[i - 1][j] o Valor(i) + B[i-1][j-Peso(i)]

Pero antes de valorar esto último, hasta antes del Peso(i) se copia B[i-1][j], es decir, lo que haya en la celda de arriba justo de la fila anterior.

Con esto lograremos llenar la tabla y obtener el valor máximo de la mochila (última celda).

Después de esto queda calcular los objetos que componen la mochila.

Se siguen estos pasos básicos:

(celda origen: última celda)

- ¿El valor de la celda de arriba es igual que el de la celda actual?

  Subo y se repite la pregunta.

- ¿El valor de la celda de arriba es distinto que el de la celda actual?

  Escojo el objeto de esa fila

  Lo añado a mi lista de objetos

  Subo

Desplazo el peso del objeto escogido a la izquierda

Se repite el proceso hasta que ya no pueda seguir subiendo.

División del problema

- ### Primera clase: OBJETO

  En esta clase definimos los atributos de los objetos, así como su constructor para instanciarlos y poder usarlos posteriormente en el algoritmo como tal. A esto le añadimos los métodos básicos getter y setter junto al toString para mostrarlos por pantalla con su nombre, valor y peso.

```
private String nombre;
private double peso;
private double valor;
private double cantidad;  //No lo usamos en un principio

public Objeto(String nombre, double peso, double valor) {
    this.nombre = nombre;
    this.peso = peso;
    this.valor = valor;
    this.cantidad = 1;
}
```

- ## Clase principal: MOCHILA

En esta clase encontraremos primeramente los métodos necesarios para poder realizar la carga de archivos y de esta manera cargar y trabajar con los datos proporcionados.

- Método Load normal

```
int contador = 1;
private void loadFile(String basefilename) throws FileNotFoundException { //METODO LOAD
    File f = new File(folder+basefilename+"_c.txt");
    this.objects = new ArrayList<Objeto>();
    Scanner sc = new Scanner(f);
    this.capacidad = Integer.parseInt(sc.nextLine().trim());
    sc.close();

    f = new File(folder+basefilename+"_w.txt");
    sc = new Scanner(f);

    while (sc.hasNextLine()) {
        String linea = sc.nextLine().trim();
        Objeto objetoIterado = new Objeto("Objeto" + contador,Integer.parseInt(linea),0);
        objects.add(objetoIterado);
        contador++;
    }

    sc.close();

    f = new File(folder+basefilename+"_p.txt");
    sc = new Scanner(f);


    for (Objeto ob : objects) {
        String linea = sc.nextLine().trim();
        ob.setValor(Integer.parseInt(linea));

    }

    sc.close();
}
```

- Método Load para decimales

```
private void loadFile2(String basefilename) throws FileNotFoundException { //METODO LOAD
    File f = new File(folder+basefilename+"_c.txt");
    this.objects = new ArrayList<Objeto>();
    Scanner sc = new Scanner(f);
    this.capacidad = Integer.parseInt(sc.nextLine().trim());
    sc.close();

    f = new File(folder+basefilename+"_w.txt");
    sc = new Scanner(f);

    while (sc.hasNextLine()) {
        String linea = sc.nextLine().trim();
        Objeto objetoIterado = new Objeto("Objeto" + contador,Double.parseDouble(linea),0);
        objects.add(objetoIterado);
        contador++;
    }

    sc.close();

    f = new File(folder+basefilename+"_p.txt");
    sc = new Scanner(f);


    for (Objeto ob : objects) {
        String linea = sc.nextLine().trim();
        ob.setValor(Double.parseDouble(linea));

    }

    sc.close();
}
```

```
public Mochila(String basefilename,int opcion) {  //CARGAR DATOS
    try {
        if (opcion == 1){
            loadFile(basefilename);
        }

        if (opcion == 2) {
            loadFile2(basefilename);
        }
    } catch (FileNotFoundException e) {
        System.out.println(e.getMessage());
    }
}
```

Declaración de variables de la mochila.

```
private ArrayList<Objeto> objects = new ArrayList<Objeto>();
private double pesoFinal;
private int capacidad;
private double valorFinal;
private String folder = System.getProperty("user.dir")+File.separator
        +"Practicas"+File.separator
        +"org"+File.separator
        +"eda2"+File.separator
        +"Practica_03"+File.separator;
```

Primer problema

- Objetos irrompibles

- Pesos exactos

- Valores reales positivos

- ¿Valor máximo sin superar el peso límite?

O(NxP)

Este trozo de código lo que hace es rellenar la tabla

```
public ArrayList<Objeto> Prob1Mochila() {  //MOCHILA PARTE 01

    this.objects.sort(new CompararMochila());
    int n = this.objects.size();
    double[][] B = new double[n + 1][capacidad + 1];
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= capacidad; j++) {
            if (this.objects.get(i - 1).getPeso() <= j) {
                B[i][j] = Math.max(B[i - 1][j],
                        this.objects.get(i - 1).getValor() + B[i - 1][j - (int)this.objects.get(i - 1).getPeso()]);
            } else {
                B[i][j] = B[i - 1][j];
            }
        }
    }
    return items(B);
```

Creamos la tabla rellena con 0 por defecto

O(N)

Por tanto, el orden del problema de la mochila por la regla de la suma será de **O(NxP)**.

Después de aplicar este algoritmo se rellena la tabla y el valor total que se alcanza en la mochila es el valor de la última celda.

Ahora hay que calcular los objetos que formaran parte de la mochila. Para ello, en el algoritmo de arriba, concretamente en el return, llamamos al método items pasándole como parámetro la tabla ya rellena y esto nos devolverá una lista con esos objetos.

```
public ArrayList<Objeto> items(double[][] B) {    //Recoger Objetos Problema1
    ArrayList<Objeto> results = new ArrayList<Objeto>();
    valorFinal = B[B.length-1][B[0].length-1];
    pesoFinal = 0;
    int j = B[0].length - 1;
    for (int i = B.length - 1; i > 0; i--) {
        if (B[i][j] != B[i - 1][j]) {
            results.add(this.objects.get(i - 1));
            pesoFinal += this.objects.get(i-1).getPeso();
            j -= this.objects.get(i - 1).getPeso();
        }
    }
    return results;
```

Segundo problema

- Objetos irrompibles

- Peso exacto

- Valor real positivo

- Se puede coger la misma clase de objeto más de una vez

- ¿Valor máximo sin superar el peso límite?

```
public ArrayList<Objeto> mochilaIlimitada(){  //Problema 2 Mochila
    this.objects.sort(new CompararMochila());
    int n = this.objects.size();
    double[] array = new double[capacidad + 1];
    for (int i = 0; i <= capacidad; i++) {
        for (int j = 0; j < n; j++) {
            if(this.objects.get(j).getPeso()  <= i) {
                array[i] = Math.max(array[i], array[i - (int)this.objects.get(j).getPeso()]
                        +this.objects.get(j).getValor());
            }
        }
    }
    return items(array);
}
```

Después de aplicar este algoritmo se rellena la tabla y el valor total que se alcanza en la mochila es el valor de la última celda.

Ahora hay que calcular los objetos que formaran parte de la mochila. Para ello, en el algoritmo de arriba, concretamente en el return, llamamos al método items pasándole como parámetro la tabla ya rellena y esto nos devolverá una lista con esos objetos.

```
private ArrayList<Objeto> items(double[] array) {  //Recuperar objetos Problema2
    valorFinal = 0;
    pesoFinal = 0;
    ArrayList<Objeto> results = new ArrayList<Objeto>();
    int c = this.capacidad;
    double minWeight = this.objects.get(0).getPeso();
    int n = this.objects.size();
    while(c >= minWeight) {
        double maxValue = 0;
        int item = -1;
        for (int i = n-1; i >= 0; i--) {
            if(c - this.objects.get(i).getPeso() >= 0) {
                double newValue = array[c - (int)this.objects.get(i).getPeso()] + this.objects.get(i).getValor();
                if(newValue > maxValue) {
                    maxValue = newValue;
                    item = i;
                }
            }
        }
        if(item == -1) break;
        valorFinal += this.objects.get(item).getValor();
        pesoFinal += this.objects.get(item).getPeso();
        results.add(this.objects.get(item));
        c -= this.objects.get(item).getPeso();
    }
    return results;
}
```

Tercer problema

- Objetos irrompibles

- Pesos pueden ser no exactos (reales positivos)

- Valor real positivo

- ¿Valor máximo sin superar el peso límite?

```
public ArrayList<Objeto> decimalMochila() {  //Problema 3 Mochila


    this.objects.sort(new CompararMochila());
    int n = this.objects.size();
    double[][] B = new double[n + 1][capacidad + 1];
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= capacidad; j++) {
            numeroMaxDecimales();
            if (this.objects.get(i - 1).getPeso() <= j) {
                B[i][j] = Math.max(B[i - 1][j],
                        this.objects.get(i - 1).getValor() + B[i - 1][j - (int)this.objects.get(i - 1).getPeso()]);
            } else {
                B[i][j] = B[i - 1][j];
            }
        }
    }
    return items(B);
```

Después de aplicar este algoritmo se rellena la tabla y el valor total que se alcanza en la mochila es el valor de la última celda.

Ahora para calcular los objetos que ocupan la mochila el algoritmo recurre al método items del problema 1.

Desde algoritmo principal hace una llamada al método **numeroMaxDecimales()** y desde este llama al método **numeroDecimales().**

El método numeroDecimales() determina el número de decimales que tiene un número. Una vez esto, ahora hay que obtener el máximo número de decimales entre la capacidad o peso de la mochila y todos los objetos y para ello usamos el método numeroMaxDecimales().

Teniendo el número máximo de decimales que hay, si ejecutamos y lo aplicamos al cálculo de la capacidad y peso de los objetos, obtendremos una mochila con pesos enteros para poder trabajar.

```
private int numeroDecimales(double d) {
    if(d == (int)d) return 0;
    String text = Double.toString(Math.abs(d));
    int integerPlaces = text.indexOf('.');
    if(integerPlaces == -1) return 0;
    int decimalPlaces = text.length() - integerPlaces - 1;
    return decimalPlaces;
}

private int numeroMaxDecimales() {
    int max = numeroDecimales(capacidad);
    for (Objeto ob : objects) {
        int n = numeroDecimales(ob.getPeso());
        max = n > max ? n : max;
    }
    return (int) Math.pow(10, max);
}
```

Cuarto Problema (greedy)

- Objetos fraccionables

- ¿Valor máximo sin superar el peso límite?

```
public ArrayList<Objeto> greedyMochila() { //Problema 4 Mochila
    this.objects.sort(new CompararMochila());
    valorFinal = 0;
    pesoFinal = 0;
    ArrayList<Objeto> results = new ArrayList<Objeto>();
    for (Objeto ob : objects) {
        if(pesoFinal + ob.getPeso() <= capacidad) {
            ob.setCantidad(1);
            results.add(ob);
            pesoFinal += ob.getPeso();
            valorFinal += ob.getValor();
            if(pesoFinal == capacidad) break;
        }else {
            ob.setCantidad((capacidad-pesoFinal)/ob.getPeso());
            results.add(ob);
            pesoFinal += ob.getPeso() * ob.getCantidad();
            valorFinal += ob.getValor() * ob.getCantidad();
            break;
        }
    }
    return results;
}
```

- ### CompararMochila

En esta clase creamos un objeto Comparator que pasamos como parámetro al ArrayList de objetos antes de ejecutar cualquiera de los algoritmos.
Lo que conseguimos con esto es ordenar de menor a mayor los objetos según su peso. También hemos añadido que si se da el mismo peso se mire el valor y por último si el valor es también igual se mire el nombre.

```
public class CompararMochila implements Comparator<Objeto>{

    @Override
    public int compare(Objeto o1, Objeto o2) {

    int compare = Double.compare(o1.getPeso(), o2.getPeso());
    if (compare == 0) {
        int compare2 = Double.compare(o1.getValor(),o2.getValor());
        return compare2 == 0 ?  o1.getNombre().compareTo(o2.getNombre()) : compare2;
    }
    return compare;
```

```
this.objects.sort(new CompararMochila());
```

- ## Método Main

  En esta clase comprobamos que los distintos algoritmos funcionan correctamente comparando que la salida por consola coincide con la solución propuesta en los archivos que nos ha proporcionado.

```
public static void main(String[] args) {

    //P01
    Mochila objeto = new Mochila("p01",1);
    ArrayList<Objeto> wm = objeto.Prob1Mochila();
    System.out.println("Mochila clasica ==>"+wm);


    //P02
    Mochila objeto2 = new Mochila("p02", 1);
    ArrayList<Objeto> uk = objeto2.mochilaIlimitada(
    System.out.println("Mochila ilimitada ==>"+uk);


    //P04
    Mochila objeto4 = new Mochila("p02", 1);
    ArrayList<Objeto> gk = objeto4.greedyMochila();
    System.out.println("Greedy Mochila ==>"+gk);


    //P03 (le añadimos decimales en el archivo)
    Mochila objeto8 = new Mochila("p03", 2);
    ArrayList<Objeto> go = objeto8.decimalMochila();
    System.out.println("Decimal Mochila ==>"+go);
```

Bibiografía

https://www.guru99.com/knapsack-problem-dynamic-programming.html#:~:text=The%20basic%20idea%20of%20Knapsack,dynamic%20programming%20are%20very%20effective.