Juan Raul Mellado Garcia y Alejandro Martinez Terrones

Practica 2 EDA II
Programación Dinámica

Introducción a Programación Dinámica

Al igual que en los demás algoritmos y técnicas que hemos visto a lo largo del curso, el objetivo de la programación dinámica se basa en la optimización del tiempo de resolución de un problema pero en este caso lo que priorizamos es el encontrar la solución más óptima .

Para llevar esto a cabo y en lo que se basa este método debemos descomponer nuestro problema en sub problema en subproblemas e ir buscando la solución óptima de estos

mismo, además estas soluciones se van almacenando por si otro subproblema necesita de ellas, optimizando así también el tiempo de ejecución.

Estudio de la implementación

Estudio de la implementación

Antes de empezar con los algoritmos lo primero que vamos a hacer es mirar las clases que usamos para almacenar los datos de cara a la resolución del problema

```java
public class Tesoro implements Comparable<Tesoro>{

    private String id;
    private int peso;
    private int beneficio;
```

```java
public class Tesoro2 implements Comparable<Tesoro2>{

    private String id;
    private double peso;
    private int beneficio;
```

Como podemos ver son dos clases iguales , con la diferencia de que que una está orientada a el algoritmo en el que hacemos uso de pesos INT y la otra en la que los beneficios son DOUBLE, ademas de eso almacenamos la id, para después saber cómo referirnos a él a la hora de imprimir por pantalla la solución y el beneficio de cada uno, que es lo que intentaremos maximizar en cada algoritmo.

Dinámica con INT

En este caso lo que aplicamos es un algoritmo que que se encarga de meter los pesos más óptimos en cada iteración en un array, cada columna del array hace referencia a un peso , y en cada iteración del array se añade un nuevo ítem al los que podemos utilizar para intentar sacar el mayor valor, a los ítem se accede de forma ordenada, es decir , en

la primera iteración solo está disponible el ítem de menos peso y en la segunda estaría disponible este y el segundo de menos peso.

```
public String[] resolucion() {//
    String[] result = new String[2];
    // System.out.println(tesoros);
    String aux = "";
    // Conjunto vacio
    for (int i = 0; i <= cantidadTesoros; i++)
        matriz[i][0] = 0;
    for (int j = 0; j <= maxCapacidad; j++)
        matriz[0][j] = 0;
    // Rellenamos la matriz
    for (int i = 1; i <= cantidadTesoros; i++) {// Comenzamos desde 1 ya que la posicion 0 no se utilizar
        for (int j = 1; j <= maxCapacidad; j++) {
            if (tesoros.get(i).getPeso() > j) // Si el peso > la capacidad de la mochila, no se carga
                matriz[i][j] = matriz[i - 1][j]; // El valor mï¿½ximo no cambia
            else // Cargamos la mejor solucion en caso de que este elemento se pueda cargar
                matriz[i][j] = Math.max(matriz[i - 1][j],
                        matriz[i - 1][j - tesoros.get(i).getPeso()] + tesoros.get(i).getBeneficio());
        }
    }
    // Imprime el valor de la matriz
    System.out.println("Valor de la matriz:");
    for (int i = 0; i <= cantidadTesoros; i++) {
        for (int j = 0; j <= maxCapacidad; j++) {
            // System.out.print(matriz[i][j]+"\t");
            aux += matriz[i][j] + "\t";
        }
        System.out.println();
        aux += "\n";
    }

    result[0] = aux;
    aux = "";
```

```
    // Complete el formulario x [i] para calcular el valor total
    int j = maxCapacidad;
    for (int i = cantidadTesoros; i > 0; i--) {
        if (matriz[i][j] > matriz[i - 1][j]) {// Si se carga el i-ï¿½simo ï¿½tem
            carga[i] = 1; // x [i] estï¿½ configurado
            j -= tesoros.get(i).getPeso(); // Peso de la mochila menos w [i]
        } else // si el ï¿½tem i-ï¿½simo no estï¿½ cargado
            carga[i] = 0; // x [i] se establece en 0, el peso de la mochila permanece sin cambios
    }
    // Imprime los tesoros introducidos
    System.out.println("Tesoros introducidos:");
    for (int i = 1; i <= cantidadTesoros; i++)
        if (carga[i] == 1) {
            // System.out.println(tesoros.get(i));
            aux += tesoros.get(i).toString() + "\n";
        }
    System.out.println("El valor total es: " + matriz[cantidadTesoros][maxCapacidad]);
    result[1] = aux;

    // System.out.println(result[0]);
    // System.out.println(result[1]);

    return result;
}
```

Dinámica con Double

```
public String[] resolucion2() {
    // Conjunto vacio
    String[] result = new String[2];
    String aux = "";
    for (int i = 0; i <= cantidadTesoros; i++)
        matriz[i][0] = 0;
    for (int j = 0; j <= maxCapacidad; j++)
        matriz[0][j] = 0;
    // Rellenamos la matriz
    for (int i = 1; i <= cantidadTesoros; i++) {// Comenzamos desde 1 ya que la posicion 0 no se utiliza
        for (int j = 1; j <= maxCapacidad; j++) {
            if (tesoros2.get(i).getPeso() > j) // Si el peso > la capacidad de la mochila, no se carga
                matriz[i][j] = matriz[i - 1][j]; // El valor mï¿¼ximo no cambia
            else // Cargamos la mejor solucion en caso de que este elemento se pueda cargar
                matriz[i][j] = Math.max(matriz[i - 1][j],
                        matriz[i - 1][j - (int) tesoros2.get(i).getPeso()] + tesoros2.get(i).getBeneficio());
            // int value = (int)tesoros2.get(i).getPeso();
            // System.out.println(value);

        }
    }
    // Imprime el valor de la matriz
    System.out.println("Valor de la matriz:");
    for (int i = 0; i <= cantidadTesoros; i++) {
        for (int j = 0; j <= maxCapacidad; j++) {
            System.out.print(matriz[i][j] + "\t");
            aux += matriz[i][j] + "\t";
        }
        System.out.println();
        aux += "\n";
    }

    result[0] = aux;
    aux = "";
```

```
    double j = maxCapacidad;
    for (int i = cantidadTesoros; i > 0; i--) {
        // int index=(int)j;
        // System.out.println(j);
        // System.out.println(matriz[i][(int)j]);
        // System.out.println(matriz[i - 1][(int)j]);
        // System.out.println(matriz[i][(int)j] > matriz[i - 1][(int)j]);
        if (matriz[i][(int) j] > matriz[i - 1][(int) j]) {// Si se carga el i-ï¿¼simo ï¿¼tem
            carga[i] = 1; // Elemento en la mochila
            j -= tesoros2.get(i).getPeso(); // Restamos el peso
            if (j > 0 && i != 1 && j % 1 != 0) // Obtenemos el indice peso con el que podemos trabajar en la
                                               // siguiente iteracion
                j += 1;
            j = (int) j;

            // System.out.println(j);
        } else
            carga[i] = 0; // No cambia el peso de la mochila
    }
    // imprime los objetos cargados
    System.out.println("Tesoros cargados:");
    for (int i = 1; i <= cantidadTesoros; i++)
        if (carga[i] == 1) {
            System.out.println(tesoros2.get(i));
            aux += tesoros2.get(i).toString() + "\n";
        }
    result[1] = aux;
    System.out.println();
    System.out.println("El valor total es: " + matriz[cantidadTesoros][maxCapacidad]);

    return result;
}
```

Aquí como podemos ver es el mismo funcionamiento que el anterior, solo que se usan doubles en vez de int

Greed

```java
public String Greed() {
    Double PesoActualMuchila = (double)maxCapacidad;
    String result = "";
    PriorityQueue<Tesoro2> aux = new PriorityQueue<Tesoro2>(Collections.reverseOrder());
    for (int i = 0; i < cantidadTesoros; i++) {
        aux.add(tesoros2.get(i));
    }

    for (int i = 1; i < cantidadTesoros; i++) {
        if ((PesoActualMuchila - aux.peek().getPeso()) >= 0) {
            result+=aux.peek()+"\n";
            carga[i] = 1;
            PesoActualMuchila -= aux.poll().getPeso();
        } else {
            aux.poll();
            carga[i] = 0;
        }
    }

    System.out.print(result);
    //System.out.println("El valor total es: " + valortotal);

    return result;
}
```

Aquí lo que podemos ver es el algoritmo Greedy, su funcionamiento consiste en añadir al los resultado el ítem con mayor valor a la bolsa, y repetir esto hasta que no haya más objetos o no sea posible añadir ninguno más a la bolsa.

Estudio Teórico

En cuanto a los algoritmos que tenemos, nos encontramos que dos de ellos son prácticamente lo mismo, siendo estos el de programación dinámica con entero y con reales, en el caso de comparar estos 2 la diferencia no debería ser casi apreciable, simplemente pondremos como un poco mas optimo al de enteros ya que es mas simple para el ordenador procesar los números sin coma flotante.

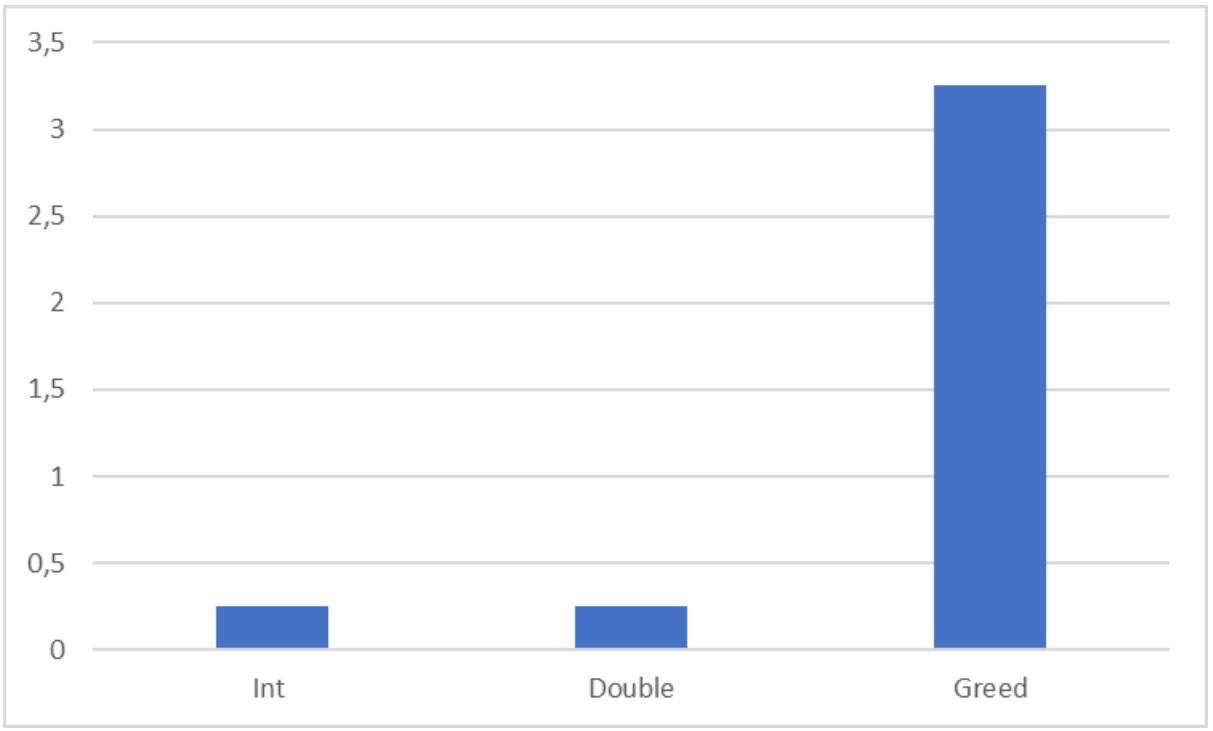
En cuanto a la complejidad algorítmica de estos dos, nos encontramos que recorre un array con todo los tesoros y una vez por peso ,y esto lo recorremos dos veces, primero la recorremos para llenarla de 0 y una segunda vez para dar el valor óptimo en cada caso, aun así, aunque se hagan más iteraciones que en otros algoritmo al guardar los resultados anteriores ahorrando así el tiempo.

En cuanto al Greed con que solo recorre una vez la lista de tesoros, aunque, este cuenta con un problema y es que no nos asegura que su solución sea la óptima, simplemente nos da una solució, y aun así en muchos caso puede ser menos eficiente que la programación dinámica, ya que este si que almacena sus resultados anteriores para no tener que repetir cálculos .

Estudio Experimental

Cabe señalar que para la realización de este estudio experimental se ha comentado la parte de código de los algoritmos que era la encargada de mostrar los resultados de este para así poder tener un tiempo más realista y poder llegar a una mejor conclusión.

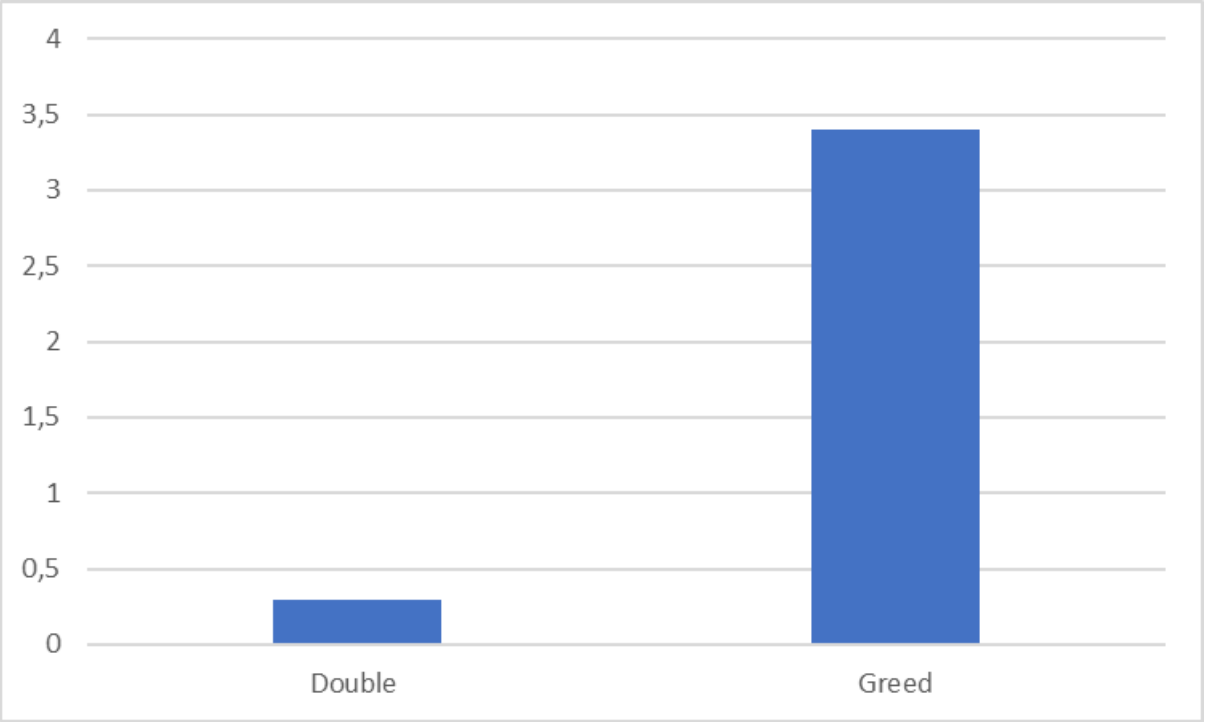
Juego de Pruebas controlado



Como podemos ver, este juego de pruebas es bastante pequeño y sus datos no se diferencian demasiado, pero ya podemos ver que la programación dinámica es superior al greed

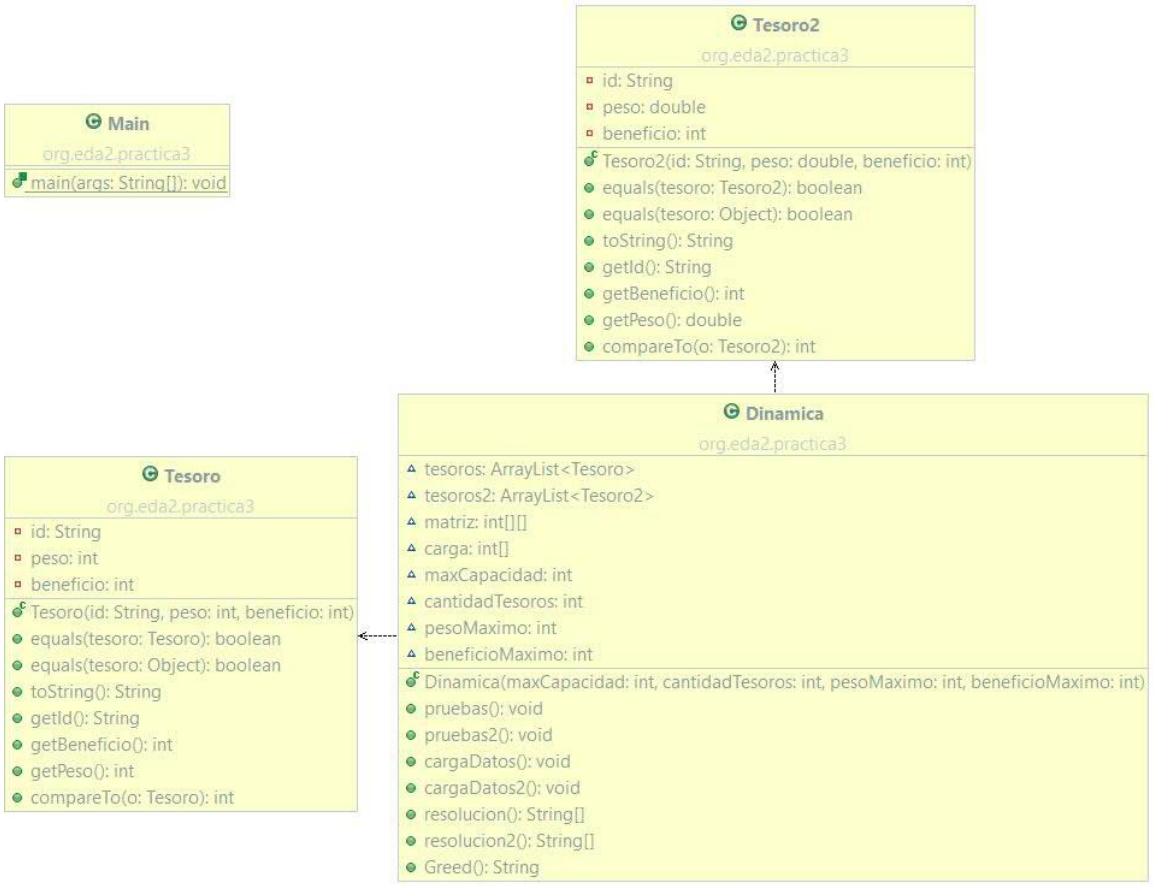
Conjunto de Tesoros Generado Aleatoriamente

En este caso el generador que tenemos genera con valores doubles, y como sabemos que los algoritmos dinámicos que tenemos funcionan prácticamente igual únicamente vamos a comparar únicamente el que usa doubles con el greed.



Y como podemos ver, se mantiene la tendencia de que Greed es más lento que nuestro algoritmo dinámico, ya que este no repite los cálculos que ya ha hecho anteriormente

Anexo



Bibliografía

Para la realización de esta práctica se ha consultado los apuntes proporcionados por el profesor y los siguientes videos como material de apoyo:

- ¿En qué consiste REALMENTE la PROGRAMACIÓN DINÁMICA?
- Programación dinámica. Problema de la Mochila.