BACKTRACKING

Juan Antonio Echeverrías Aranda nº exp 289

Samuel Martín Gómez-Calcerrada nº exp 280

Apartado 1: Explicación del algoritmo diseñado

El método Calcular va introduciendo todas las combinaciones posibles de objetos en la mochila hasta que esta está llena. En ese momento la envía a compararse con el resto de mochilas llenas y se escoge la óptima.

Apartado 2: Explicación de los atributos y métodos de cada clase

Clase Algoritmo

Atributos

La clase *Algoritmo* consta de tres atributos estáticos:

- **inventario**: es un array de tipo *Objeto*, contiene diversos objetos.

- **optima***:* es de tipo *Mochila* y es donde el método 'calcular' dejará depositado la mochila con los objetos que componen el beneficio óptimo.

- **capacidad** *: es* de tipo *int*, determina la capacidad máxima que tendrá la mochila.

Métodos

Consta del siguiente método:

- ***public static void calcular (Mochila mochila)***

Este método desarrolla un algoritmo basado en la técnica de '*vuelta atrás'* generando todas las posibles soluciones y quedándose con la óptima.

El esquema general de un algoritmo '*backtracking*' es el siguiente:

```
void Backtracking ( Mochila mochila){
        for (int k =0; k<n; k++){
            //generar estado ;
            if ( estado valido ){
                // incluirlo en solucion ;
                if (estado es hoja )
                    mostrar/guardar( solucion );
                else
                    Backtracking(n, i+ 1, solucion );
                // borrarlo de solucion ;
            }
        }
    }
```

Siguiendo este esquema hemos desarrollado el código del método '**calcular**', que mostramos a continuación:

```
public static void calcular ( Mochila mochila){

  public static void calcular ( Mochila mochila){

              for (int i=0; i<inventario.length;i++){

                if (inventario[i].getUnidades()!=0 && mochila.cabe(inventario[i])){

                        mochila.añadir(inventario[i]);

                        inventario[i].decrementar();

                           if   ( ((i==inventario.length-
                          1)&&inventario[i].getUnidades()==0) ||
                          (!mochila.cabe(inventario[i])
                          &&inventario[i].getUnidades()!=0) ){

                                  if (mochila.beneficio()>optima.beneficio()){

                                          optima=mochila.copia();

                             }

                         }

                         else{

                             calcular(mochila );

                         }

                         mochila.sacar(inventario[i]);

                        inventario[i].incrementar();

                     }//if (mochila.cabe(inventario[i]))

                }//fin for inventario.length

            }// fin backtracking
```

Para hacer más visual el algoritmo hemos creado un nuevo atributo en el objeto llamado nombre. Este es el resultado de ejecutar el programa.

```
run:
CONTENIDO DEL INVENTARIO:
nombre: banana, peso: 50, beneficio: 1000, unidades: 3
nombre: patata, peso: 35, beneficio: 700, unidades: 7
nombre: piña, peso: 78, beneficio: 2000, unidades: 1
nombre: pepino, peso: 58, beneficio: 1400, unidades: 4
nombre: tomate, peso: 28, beneficio: 1100, unidades: 2
nombre: calabacin, peso: 32, beneficio: 600, unidades: 6
nombre: manzana, peso: 10, beneficio: 350, unidades: 3
nombre: berenjena, peso: 9, beneficio: 280, unidades: 2
nombre: apio, peso: 24, beneficio: 650, unidades: 2
nombre: naranja, peso: 80, beneficio: 800, unidades: 1
  - Carga total: 1129
  - Beneficio total: 8880

CONTENIDO ÓPTIMO DE LA MOCHILA:
El objeto contenido en la posicion 0 es de tipo: nombre: tomate, peso: 28, beneficio: 1100, unidades: 2
El objeto contenido en la posicion 1 es de tipo: nombre: manzana, peso: 10, beneficio: 350, unidades: 3
El objeto contenido en la posicion 2 es de tipo: nombre: apio, peso: 24, beneficio: 650, unidades: 2
El objeto contenido en la posicion 3 es de tipo: nombre: berenjena, peso: 9, beneficio: 280, unidades: 1
En total, la mochila lleva un total de: 8 objetos
  - Carga: 143.0
  - Ganancias: 4830.0

GENERACIÓN CORRECTA (total time: 0 seconds)
```

Clase Objeto:

Atributos

La clase *Objeto* consta de tres atributos privados:

- **nombre**: es de tipo *String,* indica el nombre del tipo de objeto.
- **peso**: es de tipo *int*, indica el peso del objeto.
- **beneficio**: es de tipo *int,* indica el beneficio del objeto.
- **unidades**: es de tipo *int*, indica las unidades que hay del objeto.

Métodos

Consta de los siguientes métodos:

- **public String getNombre()**
  Devuelve el atributo *nombre.*

- **public int getPeso()**

Devuelve el valor del atributo *peso.*

- **public int getBeneficio()**
  Devuelve el valor del atributo *beneficio.*

- **public int getUnidades()**

Devuelve el valor del atributo *unidades.*

- **public void setUnidades (int unidades)**

Cambia el valor del atributo *unidades* al valor que se le pasa como argumento. Si el argumento que se le pasa es un entero menor que cero el atributo *unidades* se inicializa a 0.

- **public Objeto copia()**

Devuelve una copia del objeto.

- **public boolean equals(Objeto objeto)**

Devuelve *true* si el objeto es el mismo que el que se le pasa como argumento y *false* en caso contrario. Suponemos que si el atributo *nombre* de los dos objetos coincide, son el mismo tipo de objeto.

- **public void incrementar()**

Incrementa en uno el atributo *unidades.*

- **public void decrementar()**

Decrementa en uno el atributo *unidades.*

- **public String toString()**

Devuelve un String con los valores de los atributos del objeto.

Clase Mochila

Atributos

La clase *Mochila* consta de tres atributos privados:

- *peso*: es de tipo *double*, indica el peso de todos los objetos que contiene.
- *capacidad*: es de tipo *int,* indica el peso máximo que puede soportar.
- *objetos*: es un array de tipo *Objeto*, contiene los objetos que alberga.

Métodos

Consta de los siguientes métodos:

- **public double getPeso()**

Devuelve el valor del atributo *peso.*

- **public int getCapacidad()**

Devuelve el valor del atributo *capacidad.*

- **public Objeto[] getObjetos()**

Devuelve la referencia al atributo *objetos.*

- **public Objeto getObjeto(int i)**

Devuelve la referencia al objeto contenido en el array *objetos* de la posición que se le pasa como argumento.

- **public int numObjetos()**

Devuelve el número de objetos distintos que hay en la mochila, es decir, la longitud del array *objetos*. No confundir con el número de unidades totales de cada objeto contenidos en la mochila.

- **public int numUnidades()**

Devuelve el número total de unidades de todos los objetos contenidos en la mochila.

- **public Mochila copia()**

Devuelve una copia de la mochila.

- **public double beneficio()**

Devuelve el beneficio total de todos los objetos contenidos en la mochila.

- **public boolean cabe (Objeto objeto)**

Devuelve *true* si el objeto que se pasa como argumento cabe en la mochila y *false* en caso contrario.

- **public int esta (Objeto objeto)**

Devuelve la posición en donde se ubica el objeto que se le pasa como argumento, si el objeto no se encontrase en la mochila se devuelve -1.

- **public boolean añadir(Objeto objeto)**

Añade el objeto que se le pasa como argumento a la mochila en caso de que quepa. Lo primero que hará será comprobar si cabe; si no cabe devuelve *false* y si cabe devuelve true, pero antes mira si ese tipo de objeto está ya, en cuyo caso se incrementa en uno sus unidades y en caso contrario se añade el objeto al array *objetos* sin más.

- **public boolean sacar(Objeto objeto)**

Saca el objeto que se le pasa como argumento de la mochila en caso de encontrarse en ella. Lo primero que hará será comprobar si esta; si no está devuelve *false* y si está devuelve true, pero antes mira si las unidades de ese objeto son mayores a uno, en cuyo caso se decrementa en uno sus unidades y en caso contrario se borra el objeto del array *objetos* sin más.

- **public String mostrarMochila()**

Devuelve un String con los valores de los atributos de la mochila.