UNIVERSIDAD
DE GRANADA

Algoritmica
Grado en Ingeniería Informática
Curso 2018-2019

Ejercicio Propuesto.
Cambio de monedas

Félix Ramírez García
felixramirezgarcia@correo.ugr.es

26 de mayo de 2019

Índice

Índice de figuras

Índice de tablas

1. Introducción

El objetivo de esta practica es estudiar cómo habría que modificar el planteamiento del problema de dar cambio resuelto por programación dinámica para el caso en que el número de monedas de cada tipo no sea ilimitado. Para ello vamos a empezar explicando como funciona el algoritmo con numero de monedas ilimitado y luego concluiremos con las modificaciones que realizaremos sobre esta versión para que de solución a un problema con un numero limitado de monedas.

2. Solución con monedas ilimitadas

Este algoritmo de programación dinámica sirve para calcular el número de monedas a retornar de una determinada suma y la cantidad de cada tipo de moneda. Para ello pasaremos como parámetro la cantidad a retornar y un vector con el valor de los diferentes tipos de monedas. Finalmente el algoritmo retorna un vector con la cantidad de cada tipo de monedas a devolver.

En primer lugar debemos pensar como plantear el problema de forma incremental. Consideramos el tipo de moneda de mayor valor, XN. Si XN >C entonces la descartamos y pasamos a considerar monedas de menor valor. Si XN <C tenemos dos opciones:

-tomar una moneda de tipo XN, y completar la cantidad restante C – XN con otras monedas
-no tomar ninguna moneda de tipo XN Y completar la cantidad C con monedas de menor valor.

De las dos opciones nos quedamos con la que requiera un número menor de monedas. El problema lo podemos expresar de la siguiente forma cuando consideramos N tipos de monedas:

```
Cambio(N,C) = cambio(N-1,C) si XN > C
              MIN(cambio(N-1),cambio(N,C-XN)+1) si XN <=C
```

.
Podemos razonar análogamente para monedas de valores k menores que N y para cantidades C' menores que C.

```
Cambio(K,C') = Cambio(K-1,C') si xk > C'
               MIN(cambio(K-1,C'),cambio(N,C'-Xk)+1) si k >= C'
```

.

Llegamos a los casos base de la recurrencia cuando completamos la cantidad C (1), o cuando ya no quedan más tipos de monedas por considerar, pero aún no se ha comple-

tado la cantidad C (2).

(1) cambio(k,0) = 0 si 0<= k <=n
(1) cambio(0,C') = infinito si 0<C' <= C

Podemos construir una tabla para almacenar los resultados parciales que tenga una fila para cada tipo de moneda y una columna para cada cantidad posible entre 1 y C. Cada posición t[i,j] será el número mínimo de monedas necesario para dar una cantidad j con 0 <= j <= c utilizando sólo monedas de los tipos entre 1 e i, con 0 <=i<=n .

La solución al problema será por tanto el contenido de la casilla t[N,C].

Para construir la tabla empezamos rellenando los casos base t[i ,0] = 0, para todo i con 0 <= i <= n . A continuación podemos rellenar la tabla bien por filas de izquierda a derecha, o bien por columnas de arriba a abajo.

Siguiendo el método de la programación dinámica, se rellena una tabla con las filas correspondientes a cada valor para las monedas y las columnas con valores desde el 1 hasta el N (12 en este caso) . Cada posición (i, j) de la tabla nos indica el número mínimo de monedas requeridas para devolver la cantidad j con monedas con valor menor o igual al de i. La siguiente imagen muestra un ejemplo en que se tiene que devolver cambio de 12 y tenemos monedas de 1 , 6 y 10. Pudiendo pagar con 12 monedas de 1 , 2 monedas de 6 o una e 10 y dos de uno, quedándonos con la mejor opción que seria la que menos monedas tuviera, dos de 6 en este caso .

	0	1	2	3	4	5	6	7	8	9	10	11	12
x = 1	0	1	2	3	4	5	6	7	8	9	10	11	12
x = 6	0	1	2	3	4	5	1	2	3	4	5	6	2
x = 10	0	1	2	3	4	5	1	2	3	4	1	2	2

Figura 2.1: Tabla ejemplo a devolver 12 con monedas de 1,6,10.

Una descripción del algoritmo seria:

```
-Para cada casilla de la tabla hacer:
        -Si el valor de la moneda actual es mayor que la cantidad, se
            paga con el resto de monedas, es decir, se toma el resultado
            de la casilla superior.

        -Si el valor de la moneda actual es menor o igual que la cantidad
            , se toma el minimo entre:
                -Pagar con el resto de monedas, tomando el resultado de
                    la casilla superior.
                -Pagar con una moneda del tipo actual y el resto con el
                    resultado que se hubiera obtenido al pagar la cantidad
                     actual a la que se le ha restado el valor de la
                    moneda actual.
        -Tomar como resultado el valor de la ultima celda.
```

Para saber que tipo de monedas hay que devolver partimos de la casilla final. Para cada casilla t[i,j] el algoritmo va comprobando si su valor ha variado respecto a la casilla de la fila superior.

Si no ha variado podemos deducir que no se ha empleado ninguna moneda del tipo de la fila i, y pasamos a comprobar la casilla superior t[i-1 j].
Si ha variado, anotamos que se ha utilizado una moneda de ese tipo Xi y nos movemos a la casilla t[i, j-moneda[i]], para ver qué monedas se han utilizado para dar la cantidad restante.

Siguiendo esta estrategia terminaremos alcanzando la casilla t[0,0] en la que ya no queda ninguna cantidad pendiente. Esto se hace en el método seleccionMonedas()

Resumiendo ,si tenemos un vector con los tipos de monedas 1,6,10 y queremos saber la cantidad de monedas necesarias para dicho cambio y que tipo de monedas es, el vector que retornara será 0,2,0 es decir 0 monedas de la cantidad 1, 2 monedas de la cantidad 6 y 0 monedas de la cantidad 10.

Para estudiar el coste del algoritmo calculamos el tamaño de la tabla que hay que construir: N x (C + 1). Como las operaciones aritméticas involucradas son de coste constante, el tiempo de ejecución está en O(NC).

A continuación se muestra el código de la implementan de este algoritmo en lenguaje Java:

```
public class Cambio
{
private int[][] matrizCambio;
private int[] vectorMonedas;
private int cantidad;
private int[] vectorSeleccion;
```

```
Cambio(int cantidad, int[]  monedas){
        this.cantidad = cantidad;
        this.vectorMonedas = monedas;
        matrizCambio = calcularMonedas(cantidad, monedas);
        vectorSeleccion = seleccionarMonedas(cantidad, monedas,
            matrizCambio);
}

public int[] getVectorSeleccion(){
        return this.vectorSeleccion;
}

private int[][] calcularMonedas(int cantidad, int[]  monedas){
        int[][] matrizCambio = new int[monedas.length + 1][cantidad + 1];

        for (int i = 0; i < monedas.length; i++) matrizCambio[i][0] = 0;

        for (int j = 1; j <= cantidad; j++) matrizCambio[0][j] = 99;

        for (int i = 1; i <= monedas.length; i++)
        for (int j = 1; j <= cantidad; j++) {
                if (j < monedas[i - 1]) matrizCambio[i][j] = matrizCambio
                    [i - 1][j];
                else {
                        int minimo = 0;
                        minimo = min(matrizCambio[i - 1][j] ,
                            matrizCambio[i][j- monedas[i - 1]] + 1);
                        matrizCambio[i][j] = minimo;
                }
        }

        return matrizCambio;
}

private int[] seleccionarMonedas(int c, int[] monedas, int[][]tabla ){
        int i,j;
        int[] seleccion = new int[monedas.length];
        for(i = 0; i< monedas.length; i++) seleccion[i]=0;
        i= monedas.length;
        j= c;
        while(j>0){
                if(i>1 && tabla[i][j]==tabla[i-1][j]) i--;
                else{
                        seleccion[i-1]++;
                        j = j - monedas[i-1];
                }
        }

        return seleccion;
}

private int min(int a, int b){
        if(a<b) return a;
        else return b;
```

```
}

}
```

Para llamar a la clase con el ejemplo anterior haríamos :

Cambio c = new Cambio(12, new int[]1,6,10)

y con c.getVectorSeleccion(); obtendríamos el vector con las cantidades de cada moneda cuyo indice se corresponde con los tipos de monedas que pasamos al constructor.

3. Solución con monedas limitadas

Para el caso de que tengamos un numero de monedas limitado tendríamos que agregar otro vector con los mismo elementos que tipos de monedas diferentes tengamos. Para el ejemplo explicado serian 3 elementos. Luego los valores del vector serian la cantidad de monedas de cada tipo . Por ejemplo si tenemos los tipos de monedas [1,6,10] un vector cantidad podría ser [2,4,8]. De forma que cada vez que se entregue una moneda de un determinado tipo restas uno a la cantidad de esa moneda en el vector de cantidades. En el caso de que ya no se tengan monedas para solventar el problema se indicaría un mensaje de error o se lanzaría una excepción.