Algorítmica

Programación Dinámica

Álvaro Maximino Linares Herrera

Problema del cambio de monedas

La cuarta y última práctica que voy ha realizar es sobre el problema del cambio de monedas en programación dinámica.

El código es el siguiente:

```
#include <iostream>

using namespace std;

//Funcion para crear matriz dinamica

int **CrearMatriz(int fil, int col){
 int **m;
 m=new int *[fil];
 for(int i=0;i<fil;i++)
    m[i]=new int [col];
 return m;
}//Fin CrearMatriz

//Funcion para calcular el minimo de dos numeros

int Minimo(int x, int y){
 if(x<y){
   return x;
 }else{
   return y;
 }
}//Fin Minimi

//Funcion que nos devuelve el cambio optimo.

int Cambio(int tam,int *nm, int cant){

 int **matriz=CrearMatriz(tam+1,cant+1);

  for(int i=0;i<tam+1;i++)
    for(int j=0;j<cant+1;j++)
       matriz[i][j]=0;
```

```
  for(int i=0;i<tam+1;i++){
    matriz[i][0]=0;
  }

  for(int j=1;j<cant+1;j++){
     matriz[0][j]=999999;
  }

  for(int i=1;i<=tam;i++){
    for(int j=1;j<=cant;j++){
        if(nm[i-1]>j){ //Vemos si la moneda es superior a la cantidad a devolver.
              matriz[i][j]=matriz[i-1][j];
            }else{
              int minimo=0;
          minimo=Minimo(matriz[i-1][j],matriz[i][j-nm[i-1]]+1);
          matriz[i][j]=minimo;
        }
      }
  }
int solucion[tam];
for(int i=0; i<tam; i++){
 solucion[i]=0;
}
int k=cant, y=tam;
  while(k!=0 && y!=0){
     if(matriz[y][k]==matriz[y-1][k]){
            y--;
      }else{
            solucion[y-1]=solucion[y-1]+1;
            k=k-nm[y-1];
      }
  }
  for(int p=0; p<tam; p++){
            cout << "Usamos " << solucion[p] << " de valor " << nm[p] << endl;
  }

 return matriz[tam][cant];

}//Fin cambio
```

```
int main(){
//Ejemplo de las transparencias usadas en clase para exponer este problema.
int cant=8;
int nm[3]={1,4,6};

cout << "El cambio es: " << Cambio(3,nm,cant) << endl;

}//Fin main
```

Las funciones de este código son:

- CrearMatriz, que crea una matriz dinámica.

- Mínimo, que nos devuelve el mínimo de dos números enteros.

- Cambio, que nos devuelve un valor entero con el número de monedas mínimas para dar el cambio que le ponemos. También nos muestra por pantalla cuantas monedas de cada tipo nos devuelve. Su uso es muy sencillo, creamos una matriz, la primera fila la iniciamos a 0 y la primera columna a un número muy grande, por ejemplo 999999. Ahora con dos bucles for vamos recorriendo la matriz y vamos viendo que cantidad de monedas vamos metiendo en la matriz, ajustando la cantidad a devolver cada vez que usamos una. Luego recomponemos la solución en un vector que tendrá el número de monedas que usamos de cada tipo, para ello recorremos la matriz y vamos viendo cual fue el mínimo y por tanto fue el que metimos en la matriz.

Orden de eficiencia

El orden de complejidad de este algoritmo sería:

Primero crearíamos la matriz para saber que monedas cogemos y como se trata de n tipos de monedas y una cantidad cant tenemos que el orden de complejidad es O(n*cant).

Como para recomponer la solución nos hace falta recorrer la matriz desde matriz[n][cant] hasta la posición inicial matriz[0][0] tendríamos que el orden de eficiencia es O(n+matriz[n][cant]).

Ejecuciones

Para este problema vamos a plantear un par de ejecuciones:

Vamos a empezar por el ejemplo de las transparencias de las clases teóricas, para ello cogemos como cantidad a devolver 8 y 3 tipos de monedas de valores 1, 4 y 6.

Veamos si el código anterior es capaz de resolverlo.

```
>g++ -o cambio cambio.cpp
>./cambio
Introduce el valor a devolver: 8
Usamos 0 de valor 1
Usamos 2 de valor 4
Usamos 0 de valor 6
El cambio es: 2
```

Como podemos observar nos dice que ante el sistema monetario presentado con anterioridad y con el valor a devolver 8 el número de monedas que necesitamos es 2 y que son de valor 4 con lo cual sabemos que el programa hace el cálculo bien.

Veamos otro ejemplo:

Para el mismo sistema monetario pero esta vez con cantidad para devolver 10 vemos que nos dice que el número óptimo de monedas es 2 y que son una de 4 y otra de 6.

```
>./cambio
Introduce el valor a devolver: 10
Usamos 0 de valor 1
Usamos 1 de valor 4
Usamos 1 de valor 6
El cambio es: 2
```

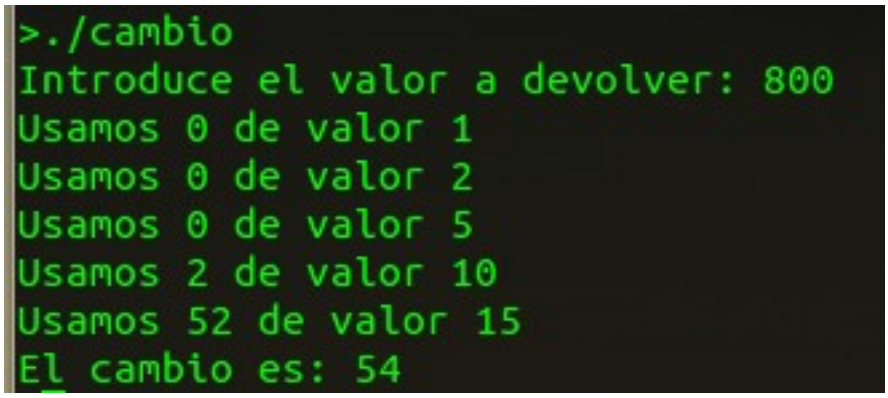
Cambiemos ahora el sistema monetario con 5 monedas de valor 1, 2, 5, 10, 20.

Para el valor a devolver igual a 10...

```
>g++ -o cambio cambio.cpp
>./cambio
Introduce el valor a devolver: 10
Usamos 0 de valor 1
Usamos 0 de valor 2
Usamos 0 de valor 5
Usamos 1 de valor 10
Usamos 0 de valor 15
El cambio es: 1
```

Vemos que bastaría con una sola moneda y que su valor, obviamente, sería de 10.

Para el valor a devolver 800 el cambio óptimo sería:


>./cambio
Introduce el valor a devolver: 800
Usamos 0 de valor 1
Usamos 0 de valor 2
Usamos 0 de valor 5
Usamos 2 de valor 10
Usamos 52 de valor 15
El cambio es: 54