Cuaderno de Practicas de Algorítmica

Algorítmica
Memoria de Practicas

Carlos de la Torre DNI: 75145459C
Grupo A

Índice

1. Objetivos

En este cuaderno se va a intentar explicar poco a poco las diferentes practicas que hay que solucionar a lo largo de la asignatura, concretamente serán 6 practicas que iremos resolviendo poco a poco sin perder de vista el global de la asignatura que no es mas que saber contar instrucciones y poder calcular la eficiencia de los algoritmos que nosotros mismo haremos.

2. Primera Practica.....Corregida

2.1. La Teoría

En esta primera practica tenemos que intentar contar las instrucciones de dos algoritmos en concreto, Algoritmo 1 y Algoritmo 2, los cuales *no son recursivos* por lo tanto en principio deberían se bastante fácil, lo primero que vamos ha hacer es explicar cual sera el proceso por el cual vamos a contar estos dos algoritmos.

```
1 i:= 1
2 mientras i <= n hacer
3     si a[i] >= a[n] entonces
4         a[n]:=a[i]
5     finsi
6     i:= i * 2
7 finmientras
```

Algoritmo 1

```
1 cont:=0
2 para i:= 1,...,n hacer
3     para j:= 1,...,i-1 hacer
4         si a[i] < a[j] entonces
5             cont:= cont + 1
6         finsi
7     finpara
8 finpara
```

Algoritmo 2

2.2. Metodología.

Para poder contar las instrucciones que tenemos en este algoritmo lo único que tenemos que tener claro es cuales son las sentencias atómicas que se ejecuta en cada paso, por ejemplo en la linea 1 del Algoritmo 1 podemos ver que hay una asignación de un valor a una variable, pues bien en esta linea tenemos 1 sentencia atómica puesto que la el valor lo tenemos en el propio código y no hay que leerlo desde ningún sitio solamente se realiza una asignación de valor a la variable, sin embargo en la linea 6 del Algoritmo 1 la cosa cambia, al fijarnos nos damos cuenta de que a la variable y se le realiza una multiplicación por lo tanto la cantidad de sentencias atómicas dependerá del repertorio de instrucciones que tenga la maquina donde vamos a ejecutar este algoritmo puesto que si la maquina contempla en una sola instrucción la multiplicación la contaremos como una pero si por el contrario la multiplicación de esta maquina la realiza sumando 2 veces el numero i tendremos que contar 2 instrucciones atómicas para la multiplicación.

Para efectos practico en este caso usaremos el primer supuesto así que la linea 6 del código tendría 3 sentencias atómicas ¿por que?, pues esta claro por que la variable i tenemos que leerla y escribirla así pues una lectura una multiplicación y una escritura son las instrucciones atómicas que nos encontramos en esta linea.

Bien una ves que tenemos claro como se realiza el conteo básico de estas lineas de código lo único que tenemos que hacer es lo mismo con todas, por supuesto que habrá situaciones mas complicadas pero en esos casos usaremos el paradigma de divide y vencerás.

2.3. Contando el Primer Algoritmo.

Aparte de las directivas antes explicadas, podemos seguir unos sencillos pasos para poder resolver los algoritmos 1 y 2 que detallamos a continuación, para no hacernos un lio con las explicaciones vamos a comenzar explicando los pasos a seguir con el algoritmo 1 y si hiciera falta a posteriori explicaríamos los pasos del segundo algoritmo.

1. Analizamos la estructura del algoritmo y observamos el por que no se puede saber el tiempo exacto del algoritmo 1

2. Nos fijamos en el predicado booleano de la sentencia if (linea 3) y nos damos cuenta que no podemos saber nunca al 100 % cuando es cierta o cuando es falsa, es por esta razón por la que se hace un cálculo aproximado y no un cálculo exacto de lo que tardara el algoritmo.

3. Como ya nos hemos fijado antes en el algoritmo ya sabemos que el contador se incrementa con valores multiplicados por 2 (osea 2 * 2 * 2 * 2) por lo tanto la función que nos dice cuantas veces estará contenido el 2 dentro de N es el logaritmo en base 2 de N $log_2 N$ por lo tanto la sumatoria del primer while (linea 2) quedaría: $\sum_{i=1}^{Log_2 N} 2$

4. Se calcula el mejor de los casos, en este caso se trata de cuando en la linea 2 el predicado booleano es siempre falso. Por lo tanto la sumatoria es:$\sum_{i=1}^{Log_2 N} 2 + 3$ el 3 es de evaluar el predicado booleano de la linea 2 del while, leemos i leemos n y comparamos y el 2 es de contar la asignación del 1 a la variable i y el salto de la condición false del while.

5. Ya tenemos el mejor de los casos ahora calculamos el peor de los casos, para ello repetimos lo anterior pero esta vez contemplamos que el predicado booleano (linea 2) es verdadero de esta manera podemos darnos cuenta que la sumatoria nos saldrá igual pero sumándole las 3+3 instrucciones atómicas que están dentro del if (lineas 3 y 4) estas 6 instrucciones las sumamos por que primero leemos a[n] después leemos a[i] y luego comparamos o igualamos el resultado, en esta ocasión la sumatoria quedaría de esta manera: $\sum_{i=1}^{Log_2 N} 2 + 3 + 3 + 3$

6. Como ya tenemos las sumatorias del mejor y el peor caso ahora solo queda calcular el caso promedio que en este caso hace uso de las dos sumatorias anteriores, para que sea mas fácil de identificar cada una de las partes pondremos la formula general:

$$t(n) = \sum_{i=1}^{log_2 n} Probabilidad \times \left[\frac{a[i] \geq a[n]}{cierto\ en\ J\ ocaciones}\right] + \left[\frac{a[i] \geq a[n]}{falso\ en\ K\ ocaciones}\right]$$

   después de ver la formula nos podemos dar cuenta que lo que significa J y de lo que significa K, lo primero en lo que nos tenemos que fijar es en la probabilidad, que podemos calcularla igual que si fuera un dado, osea si lo pensamos tenemos que la probabilidad de sacar un 6 es un dado es de 1/6 puesto que tenemos 6 caras y solo puede salir un numero osea que decimos que sera de una entre seis, pues en nuestro algoritmo tenemos que la probabilidad sera de 1 entre el limite superior de la sumatoria que hace el recorrido de las N opciones que tenemos, pero claro, en nuestro algoritmo no siempre va a ser así puesto que tenemos un predicado booleano que no sabemos cuando sera cierto o cuando sera falso. De esto podemos deducir que la J sera el numero de veces que nuestro predicado sera falso (el mejor caso) y que K sera cuando nuestro predicado booleano sera verdadero (el peor caso), pues bien despues de esto solo nos queda sustituir.

7. Bien en el paso anterior hemos visto cuales son todo los intervinientes de la formula ahora la formula quedaría así:

$$t(n) = \sum_{i=1}^{log_2 n} \frac{1}{log_2 n + 1} \times \left[1 + \sum_{K=1}^{J} (1 + 3 + 3 + 3 + 2) + \sum_{K=J+1}^{log_2 n} (1 + 3 + 2)\right]$$

   y nos damos cuenta de que la probabilidad es 1 entre la cantidad de veces que el 2 esta repetido en N ($log_2 n$) mas 1 esto es por que la probabilidad contempla que llegamos al final de nuestras iteraciones, bien después de esto vemos un 1 esto lo ponemos por que también tenemos que contar en nuestras probabilidades la asignación que se realiza al principio de nuestro algoritmo, después de esto lo único que hacemos es sustituir en la formula la mejor opción y la peor opción de nuestro algoritmo, y mirar cuantas veces se repite cada una de ellas.

8. Por ultimo lo que nos queda es:

$$t(n) = \sum_{i=1}^{log_2 n} \frac{1}{log_2 n + 1} \times [1 + 12J + 6(log_2 n - J)] \Rightarrow \sum_{i=1}^{log_2 n} \frac{1}{log_2 n + 1} \times [1 + 12J + 6log_2 n - 6J] \Rightarrow$$

$$\sum_{i=1}^{log_2 n} \frac{1}{log_2 n+1} \times [1+6J+6log_2 n] \Rightarrow \frac{1}{log_2 n+1} \times \left(\sum_{i=1}^{log_2 n} 1 + (6\, log_2 n \times \sum_{i=1}^{log_2 n} 1) + 6 \times \sum_{i=1}^{log_2 n} J \right) \Rightarrow$$

$$\frac{1}{log_2 n+1} \times \left((x+1) + 6\, log_2 n\, (x+1) + \frac{6\, log_2 n\, (x+1)}{2} \right) \Rightarrow 1 + 6\, log_2 n + \frac{6}{2}\, log_2 n$$

$$t(n) = 1 + 6\, log_2 n + 3\, log_2 n \Rightarrow O(log_2 n)$$

9. Como ya tenemos todo despejado, nos queda que $t(n)$ es un polinomio y que como en el calculo de la eficiencia descartamos las constantes pues podemos decir que la eficiencia de nuestro algoritmo es de$O(log_2 n)$

2.4. Contando el Segundo Algoritmo.

En este segundo algoritmo seguiremos los mismos pasos que en el anterior, claro esta respetando la estructura de este, como en el anterior realizaremos la misma metodología para poder detectar cuales son las instrucciones que tenemos que contar.

Simplemente echando un vistazo por encima a el algoritmo nos damos cuenta que tenemos dos sumatorias diferentes ya que tenemos dos for anidados, bien después de esto podemos ver que al igual que el anterior algoritmo tenemos un predicado booleano que sera el que controle realmente el numero de instrucciones que se ejecutaran, ya que a priori tampoco podemos saber la cantidad de veces que este predicado sera cierto o sera falso.

Vamos a empezar contando instrucciones pero vamos a hacerlo por pasos para tener claro cuales son los valores de cada una sumatorias:

1. El primer for seria la sumatoria: $\sum_{i=1}^{n}$y como esta sumatoria lo que sumaria seria la sumatoria del segundo for no la ponemos por ahora.

2. El segundo for seria la sumatoria $\sum_{j=1}^{i-1} 1$ y como esta sumatoria lo que sumaria seria las instrucciones del if de momento no lo ponemos, pero lo que si sabemos es que lo que hay dentro de esta sumatoria se ejecutara solo una vez, ya sea tanto verdadero como falso, por eso ponemos el 1.

3. Después tenemos que contar las instrucciones que hay en el if que serian leer el valor del vector en la posición i-esima, leer el valor del vector en la posición j-esima y comparar si el segundo es mas grande que el primero entonces decimos que nuestro predicado booleano tiene 3 instrucciones atómicas.

4. Por ultimo miramos cuantas instrucciones tenemos dentro del if, y vemos que hay una lectura de la variable cont, una suma de 1 y una asignación del resultado a la variable count, en total tenemos 3 instrucciones atómicas.

5. Después de haber analizado por separado el algoritmo procedemos a juntar todos los pasos para obtener la ecuación completa, que quedaría de la siguiente manera para el peor de los casos: $t(n) = \sum_{i=1}^{n} \sum_{j=1}^{i-1} 1+3+2$ y de esta otra forma para el mejor de los casos: $t(n) = \sum_{i=1}^{n} \sum_{j=1}^{i-1} 1+3$

Bien después de haber diseccionado nuestro algoritmo solo nos queda sacar la formula general para el mismo que en este caso seria de la siguiente manera:

$$t(n) = \sum_{i=1}^{n} \sum_{j=1}^{i-1} Probabilidad \times \left[\frac{a[i] < a[j]}{cierto\, en\, K\, ocaciones} \right] + \left[\frac{a[i] < a[j]}{falso\, en\, K+1\, ocaciones} \right]$$

De acuerdo, ya tenemos la formula general para nuestro algoritmo, ahora lo primero que tenemos que calcular es la probabilidad que multiplica a las ejecuciones mas lentas y las mas rápidas, así pues para poder saber la probabilidad de nuestro algoritmo necesitamos saber cuantas veces itera nuestro algoritmo, y para poder conocer esto utilizamos un truco, que es, olvidarnos de lo que hay dentro de nuestras funciones iterativas y sustituirlas por un 1, así pues los dos for anidados quedarían de la siguiente forma: $\sum_{i=1}^{n} \sum_{j=1}^{i-1} 1$ y simplificando esto se nos quedaría: $\frac{1}{\frac{n^2-3n}{2}+1}$ por lo tanto la probabilidad quedaría de la siguiente forma: probabilidad $= \frac{1}{\frac{n^2-3n}{2}+1}$, ahora bien una vez realizado esto lo único que nos resta es poner en la formula

general de nuestro algoritmo las ejecuciones mas y menos favorables, para realizar esto lo que haremos es suponer que k sera el numero de iteraciones que el predicado booleano es verdadero por lo tanto el tiempo de k quedaría de esta manera:

$$t(k) = 1 + \sum_{l=1}^{k} 5 + \sum_{l=k+1}^{n^2-3n/2} 3 \Longrightarrow t(n) = \frac{1}{\frac{n^2-3n}{2}+1} \times \left(1 + \sum_{l=1}^{k} 5 + \sum_{l=k+1}^{n^2-3n/2} 3\right)$$

$$t(n) = \frac{1}{\frac{n^2-3n}{2}+1} \times \left(1 + 5k + 3\left(\frac{n^2-3n}{2} - k\right)\right) \Longrightarrow \frac{1}{\frac{n^2-3n}{2}+1} \times \left(1 + 5k + \frac{3}{2}(n^2-3n) - 3k\right)$$

$$t(n) = \frac{1}{\frac{n^2-3n}{2}+1} \times \left(1 + 2k + \frac{3}{2}(n^2-3n)\right) \Longrightarrow \frac{1}{\frac{n^2-3n}{2}+1} \times \left(1 + 2k + \frac{3n^2}{2} - \frac{9n}{2}\right) \Longrightarrow O(n^2)$$

3. Segunda Practica.....Corregida

3.1. Presentación del problema:

En este caso tenemos que implementar un algoritmo de selección, osea, que tendremos que encontrar el valor de un elemento dentro de un vector desordenado dada una posición y que dicho valor de la posición sea el mismo valor que cuando el vector este ordenado.

3.2. Metodología seguida para la solución:

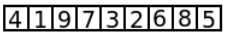
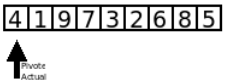
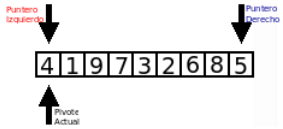
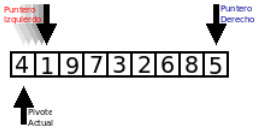
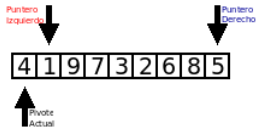
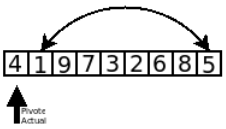
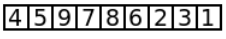
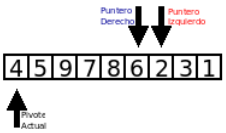
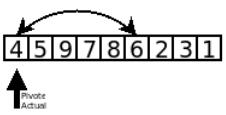
Para realizar esta practica de implementación de algoritmos primero hemos buscado cual es el lenguaje mas cómodo para la implementación del mismo, con esto me refiero al lenguaje que menos librerías tenemos que utilizar y que menos tiempo he tenido que emplear para resolver la practica, ya que en cualquier lenguaje se podría implementar la solución.

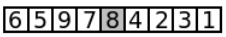
Una vez escogido, he hecho una primera aproximación utilizando el pseudocódigo que viene en las transparencias de clase, para implementar el algoritmo de QuickSort(), en este punto me he dado cuenta de que ha estas lineas le faltaban cosas y le sobraban otras pero después de muchas pruebas y búsquedas, he conseguido que el algoritmo funcionase.

Después de entender la función Pivote() que se usa en el algoritmo de QuickSort(), he comprendido que realmente dicho algoritmo debería llamarse Pivote(), ya que todo el trabajo de ordenación realmente lo realiza esta función, de hecho QuickSort() no es mas que una función wrapper por si antes de intentar ordenar quisiéramos pre procesar el vector.

Después he utilizado la función Pivote() del algoritmo QuickSort() y la he modificado para que pudiera recibir un vector y un enteros que seria la posición del valor que queremos obtener, por supuesto, aunque los ejemplos lo he hecho con números enteros es totalmente factible elaborar un template el cual después de recibir la posición nos devolviera el objeto o el valor de la posición que le hemos pedido sin tener que ordenar el vector de datos, cabe destacar que para que el algoritmo de QuickSort() funcione los objetos o datos que se encuentren en el vector deben de ser escalares, osea que, tiene que poder usarse los operadores, mayor que $>$, menor que $<$ e igual $=$, quizás este demás decirlo, puesto que si queremos ordenar algo esta observación va implícita en la propia ordenación del vector, pero como a mi me costo entender esta idea la recalco para futuras lecturas.

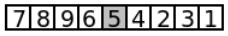
Para terminar de explicar como funciona este algoritmo nos apoyaremos en las siguientes explicaciones gráficas:

Gráfica	Descripción
	Este es nuestro vector el cual queremos encontrar el valor que se encuentra en la mediana del mismo, hay que tener en cuenta que aunque para la explicación hemos usado un vector de enteros este podría contener cualquier tipo de dato.
	En las linea 50 del código se hace la llamada a selección() nada mas llamar a la función esta utiliza tres enteros para posicionar el pivote y los dos punteros (izquierdo y derecho) que va a utilizar para llamar a la función pivote() lo primero que hace es usar el primer dato del vector como si fuera el primer pivote del algoritmo. Esto se hace en las linea de código 35.
	Después de haber posicionado el pivote se hace lo mismo con los dos punteros el izquierdo y el derecho, la primera vez que se usan estos punteros apuntan al primer dato del vector (izquierdo) y al ultimo dato del vector (derecho). Esto se hace en las lineas de código 36 y 37.
	Una vez que los punteros están en sus lugares correspondientes comienza el funcionamiento del algoritmo haciendo que el puntero de la izquierda recorra todo el vector de izquierda a derecha siempre y cuando no se pase del tamaño del vector, este parara cuando el valor que esta en la posición del puntero que estamos recorriendo sea mas grande o igual que el valor que se encuentra en la posición que se encuentra el pivote, esto es lo que hace la linea de código 39
	El puntero derecho hace lo mismo que el izquierdo pero en la otra dirección, osea que recorrerá todo el vector sin pasar desde el principio del mismo y se detendrá cuando en la posición en la que se encuentre haya un valor mayor o igual que el que se encuentra en pivote, esto es lo que hace la linea de código 40. Claro esta que en el caso que nos ocupa este puntero no se moverá del sitio por que el valor que hay en dicho puntero ya cumple con la condición esperada.
	Para el vector que nos ocupa el siguiente paso será el que se refleja en el gráfico y se intercambiaran los valores de los punteros, que como se muestra en la linea 78 si el puntero de la izquierda es menor que el puntero de la derecha se llama a la función intercambia() pasandole como parámetros ambos punteros
	Una vez realizado el intercambio de valores el vector se seguiría recorriendo el vector de tal manera que se harían los mismos pasos hasta que los punteros se cruzaran, despues de un par de iteraciones llegaríamos a una situación similar a la que se muestra en la figura.
	En el siguiente paso podemos comprobar como los dos punteros se han cruzado por lo tanto se va a proceder a intercambiar los valores pero en esta ocasión no serán entre los dos punteros.
	En esta ocasión lo que se va a cambiar es el limite inferior del vector con el puntero derecho, de esta manera, conseguimos que todos los valores que son mas grandes que el valor del dato que estamos buscando están a la izquierda y todos los valores mas chicos están a la derecha de dicho dato.



En el gráfico se puede apreciar como el valor que ha devuelto la función pivote() es 5 esta es la posición actual que tiene el pivote cuyo valor es 4, como la posición que buscamos es menor que la posición devuelta por dicha función lo que se hace a continuación es partir el vector en 2 de tal forma que la posición devuelta menos una unidad será el limite superior quedando el limite en la posición 4 que tiene el valor 8, por lo tanto esta será el nuevo vector que se le pasara a la función pivote().

El proceso que hemos explicado se repite hasta conseguir que el valor que queda en la mediana del vector corresponde al valor que quedaría en esa posición si el vector estuviese ordenado. Como referencia el siguiente gráfico muestra como sería la solución final del vector.



3.3. Código fuente:

Aunque no es necesario incluir el código fuente para la evaluación de la practica yo lo incluyo para tener el código integrado en la lectura de la memoria.

```
/*
 * Algoritmo Selección
 *        Autor: Carlos de la Torre
 *        Fecha: 07/05/14
 */
#include <vector>
#include <iostream>
#include <iomanip>
#include <string>
#include <cstdlib>
#include <ctime>
#include "../inc/concolor.h"

using namespace std;

#define TAM_MIN_VEC 9
// con esto definimos que el tamaño mínimo
// del vector para que tenga buenos resultados
// la selección de la mediana
#define DEBUGMODE 0
// con esta definición nos aseguramos que solo
// salgan las cifras de tiempo en cada ejecución
// así de esa manera es mas fácil realizar el
// estudio empírico del programa

void intercambia (vector<int> &lista, int posi, int posd){
     int temporal;
    temporal= lista.at(posi);
    lista.at(posi)=lista.at(posd);
    lista.at(posd)=temporal;
}

int Pivote(vector<int> &lista,int limite_izq,int limite_der){
    int tmp, respuesta, pivote;

    pivote = lista.at(limite_izq);
    tmp = limite_izq;
    respuesta = limite_der;

    for(;lista.at(tmp)>=pivote and tmp<limite_der;tmp++);
    if (DEBUGMODE){
     cout <<redb<< "_____" << endl;
     cout << "Posición del puntero de la Izquierda" <<whiteb<< endl;
     cout <<green<< "Valor del pivote: " << pivote << ", Posición del pivote: " << limite_izq <<
    whiteb<< endl;
     for (unsigned int i=0; i<lista.size(); i++) {
               if (i==tmp)
                    cout <<redb<< lista.at(i) <<whiteb;
               else
                    cout << lista.at(i);
```

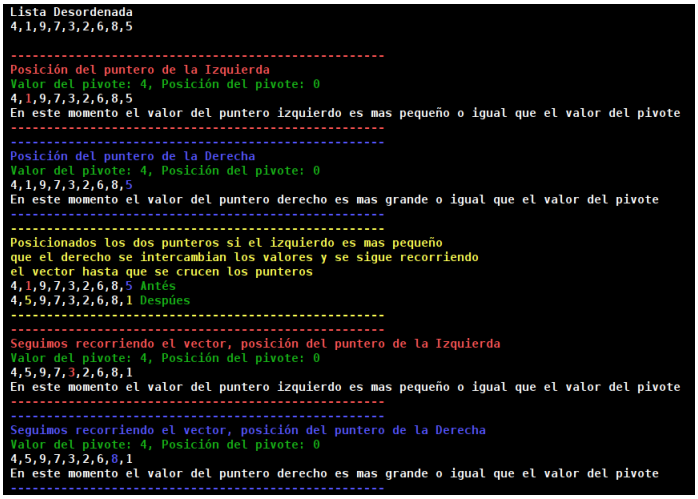
```
50
51                  if(i<lista.size()-1)
52                        cout << ",";
53                  else
54                        cout << endl;
55            }
56            cout << "En este momento el valor del puntero izquierdo es mas pequeño o igual que el
      valor del pivote" << endl;
57       cout <<redb<< "-------------------------------------------------------" <<whiteb<< endl;
58      }
59      for(;lista.at(respuesta)<=pivote and respuesta>limite_izq;respuesta--);
60      if (DEBUGMODE){
61       cout <<blueb<< "-------------------------------------------------------" << endl;
62       cout << "Posición del puntero de la Derecha" <<whiteb<< endl;
63       cout <<green<< "Valor del pivote: " << pivote << ", Posición del pivote: " << limite_izq <<
      whiteb<< endl;
64       for (unsigned int i=0; i<lista.size(); i++) {
65                  if (i==respuesta)
66                        cout <<blueb<< lista.at(i) <<whiteb;
67                  else
68                        cout << lista.at(i);
69
70                  if(i<lista.size()-1)
71                        cout << ",";
72                  else
73                        cout << endl;
74            }
75            cout << "En este momento el valor del puntero derecho es mas grande o igual que el valor
      del pivote" << endl;
76       cout <<blueb<< "-------------------------------------------------------" <<whiteb<< endl;
77      }
78      while (tmp<respuesta){
79       if (DEBUGMODE){
80                  cout <<yellowb<< "-------------------------------------------------------" << endl;
81                  cout << "Posicionados los dos punteros si el izquierdo es mas pequeño " << endl;
82                  cout << "que el derecho se intercambian los valores y se sigue recorriendo" << endl;
83                  cout << "el vector hasta que se crucen los punteros" <<whiteb<< endl;
84                  for (unsigned int i=0; i<lista.size(); i++) {
85                        if (i==tmp)
86                              cout <<redb<< lista.at(i) <<whiteb;
87                        else if (i==respuesta)
88                              cout <<blueb<< lista.at(i) <<whiteb;
89                        else
90                              cout << lista.at(i);
91
92                        if(i<lista.size()-1)
93                              cout << ",";
94                        else
95                              cout <<green<< " Antés" <<whiteb<< endl;
96                  }
97       }
98       intercambia(lista,tmp,respuesta);
99       if (DEBUGMODE){
100                 for (unsigned int i=0; i<lista.size(); i++) {
101                       if (i==tmp)
102                             cout <<yellowb<< lista.at(i) <<whiteb;
103                       else if (i==respuesta)
104                             cout <<yellowb<< lista.at(i) <<whiteb;
105                       else
106                             cout << lista.at(i);
107
108                       if(i<lista.size()-1)
109                             cout << ",";
110                       else
111                             cout <<green<< " Despúes" << endl;
112                 }
113                 cout <<yellowb<< "-------------------------------------------------------" <<whiteb<<
      endl;
114           }
115       for(;lista.at(tmp)>=pivote;tmp++);
116       if (DEBUGMODE){
117                 cout <<redb<< "-------------------------------------------------------" << endl;
118                 cout << "Seguimos recorriendo el vector, posición del puntero de la Izquierda" <<
      whiteb<< endl;
119                 cout <<green<< " Valor del pivote: " << pivote << ", Posición del pivote: " <<
      limite_izq <<whiteb<< endl;
120                 for (unsigned int i=0; i<lista.size(); i++) {
121                       if (i==tmp)
```

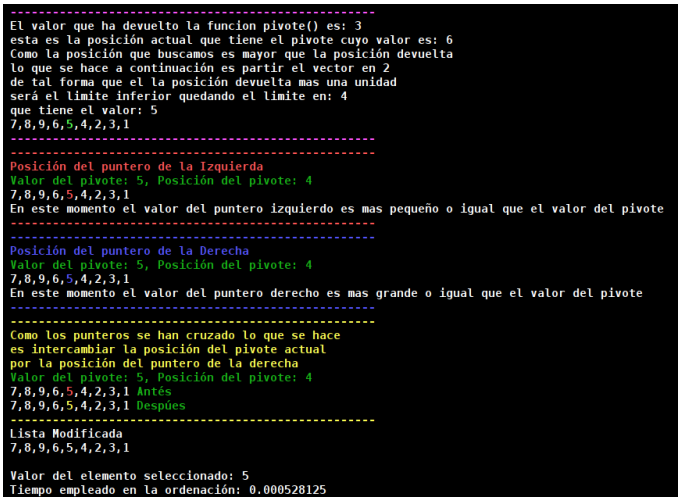
```
122                         cout <<redb<< lista.at(i) <<whiteb;
123                     else
124                         cout << lista.at(i);
125 
126                     if(i<lista.size()-1)
127                         cout << ",";
128                     else
129                         cout << endl;
130                 }
131                 cout << "En este momento el valor del puntero izquierdo es mas pequeño o igual que el valor del pivote" << endl;
132                 cout <<redb<< "------------------------------------------------------------" <<whiteb<< endl;
133             }
134          for(;lista.at(respuesta)<=pivote;respuesta--);
135             if (DEBUGMODE){
136                 cout <<blueb<< "------------------------------------------------------------" << endl;
137                 cout << "Seguimos recorriendo el vector, posición del puntero de la Derecha" <<whiteb<< endl;
138                 cout <<green<< " Valor del pivote: " << pivote << ", Posición del pivote: " << limite_izq <<whiteb<< endl;
139                 for (unsigned int i=0; i<lista.size(); i++) {
140                     if (i==respuesta)
141                         cout <<blueb<< lista.at(i) <<whiteb;
142                     else
143                         cout << lista.at(i);
144 
145                     if(i<lista.size()-1)
146                         cout << ",";
147                     else
148                         cout << endl;
149                 }
150                 cout << "En este momento el valor del puntero derecho es mas grande o igual que el valor del pivote" << endl;
151                 cout <<blueb<< "------------------------------------------------------------" <<whiteb<< endl;
152             }
153         }
154         if (DEBUGMODE){
155                 cout <<yellowb<< "------------------------------------------------------------" << endl;
156                 cout << "Como los punteros se han cruzado lo que se hace" << endl;
157                 cout << "es intercambiar la posición del pivote actual" << endl;
158                 cout << "por la posición del puntero de la derecha" <<whiteb<< endl;
159                 cout <<green<< "Valor del pivote: " << pivote << ", Posición del pivote: " << limite_izq <<whiteb<< endl;
160                 for (unsigned int i=0; i<lista.size(); i++) {
161                     if (i==tmp)
162                         cout <<redb<< lista.at(i) <<whiteb;
163                     else if (i==respuesta)
164                         cout <<blueb<< lista.at(i) <<whiteb;
165                     else
166                         cout << lista.at(i);
167 
168                     if(i<lista.size()-1)
169                         cout << ",";
170                     else
171                         cout <<green<< " Antés" <<whiteb<< endl;
172                 }
173          }
174         intercambia(lista,limite_izq,respuesta);
175         if (DEBUGMODE){
176                 for (unsigned int i=0; i<lista.size(); i++) {
177                     if (i==limite_izq)
178                         cout <<yellowb<< lista.at(i) <<whiteb;
179                     else if (i==respuesta)
180                         cout <<yellowb<< lista.at(i) <<whiteb;
181                     else
182                         cout << lista.at(i);
183 
184                     if(i<lista.size()-1)
185                         cout << ",";
186                     else
187                         cout <<green<< " Despúes" <<whiteb<< endl;
188                 }
189                 cout <<yellowb<< "------------------------------------------------------------" <<whiteb<< endl;
190          }
191          return respuesta;
192 }
193 
194 int seleccion(vector<int> &lista, int pos){
```

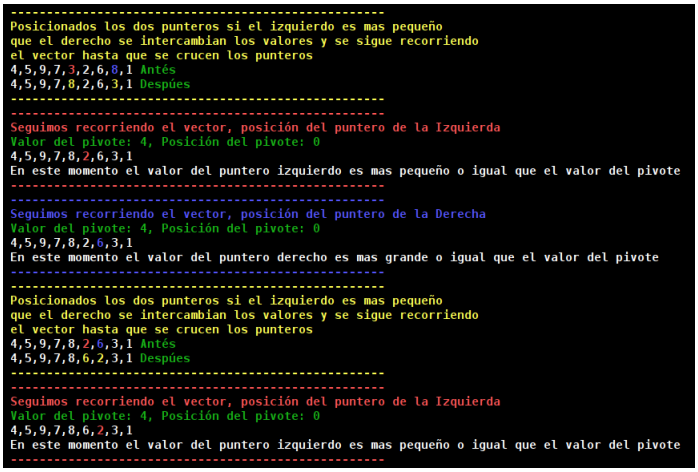
```
195      int pivote = 0;
196      int puntero_izq = 0;
197      int puntero_der = lista.size()-1;
198      while (pivote!=pos){
199           pivote = Pivote(lista,puntero_izq,puntero_der);
200           if (pos < pivote){
201                if (DEBUGMODE){
202                 cout <<magentab<< "_____" <<whiteb<<
     endl;
203                      cout << "El valor que ha devuelto la funcion pivote() es: " << pivote << endl;
204                      cout << "esta es la posición actual que tiene el pivote cuyo valor es: " <<
     lista.at(pivote) << endl;
205                 }
206                 puntero_der = pivote-1;
207                 if (DEBUGMODE){
208                      cout << "Como la posición que buscamos es menor que la posición devuelta" <<
     endl;
209                      cout << "lo que se hace a continuación es partir el vector en 2" << endl;
210                      cout << "de tal forma que la posición devuelta menos una unidad" << endl;
211                      cout << "será el limite superior quedando el limite en: " << puntero_der << endl
     ;
212                      cout << "que tiene el valor: "<< lista.at(puntero_der) << endl;
213                      for (unsigned int i=0; i<lista.size(); i++) {
214                           if (i==puntero_der)
215                                cout <<greenb<< lista.at(i) <<whiteb;
216                           else
217                                cout << lista.at(i);
218
219                           if(i<lista.size()-1)
220                                cout << ",";
221                           else
222                                cout << endl;
223                      }
224                      cout <<magentab<< "_____" <<
     whiteb<< endl;
225                 }
226           }else if (pos > pivote){
227                 if (DEBUGMODE){
228                      cout <<magentab<< "_____" <<
     whiteb<< endl;
229                      cout << "El valor que ha devuelto la funcion pivote() es: " << pivote << endl;
230                      cout << "esta es la posición actual que tiene el pivote cuyo valor es: " <<
     lista.at(pivote) << endl;
231                 }
232                 puntero_izq = pivote+1;
233                 if (DEBUGMODE){
234                      cout << "Como la posición que buscamos es mayor que la posición devuelta" <<
     endl;
235                      cout << "lo que se hace a continuación es partir el vector en 2" << endl;
236                      cout << "de tal forma que el la posición devuelta mas una unidad" << endl;
237                      cout << "será el limite inferior quedando el limite en: " << puntero_izq << endl
     ;
238                      cout << "que tiene el valor: "<< lista.at(puntero_izq) << endl;
239                      for (unsigned int i=0; i<lista.size(); i++) {
240                           if (i==puntero_izq)
241                                cout <<greenb<< lista.at(i) <<whiteb;
242                           else
243                                cout << lista.at(i);
244
245                           if(i<lista.size()-1)
246                                cout << ",";
247                           else
248                                cout << endl;
249                      }
250                      cout <<magentab<< "_____" <<
     whiteb<< endl;
251                 }
252           }
253      }
254      return lista.at(pivote);
255 }
256
257 int main(int argc, const char * argv[]){
258      struct timespec t1, t2;
259      double tresultado;
260     int num=0;
261     if (argv[1]==NULL){
262      cout << " Cuantos numeros quiere generar: " << endl;
```

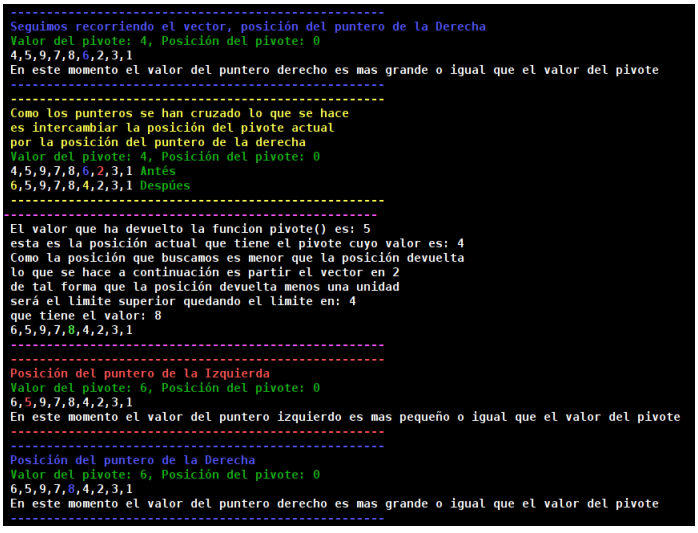
```
263       cin >> num;
264     }else
265       num=atoi(argv[1]);
266 
267     // creamos el vector según el tamaño introducido
268     vector<int> lista(num);
269 
270     // plantamos la semilla para los aleatorios
271     srand(num);
272 
273     // inicializamos el vector con los datos
274 //     for (int i=0; i<num; i++){
275 //         lista.at(i)=rand()%100;
276 //     }
277     lista.at(0)=4;
278     lista.at(1)=1;
279     lista.at(2)=9;
280     lista.at(3)=7;
281     lista.at(4)=3;
282     lista.at(5)=2;
283     lista.at(6)=6;
284     lista.at(7)=8;
285     lista.at(8)=5;
286     // imprimimos la lista sin ordenar
287     if (DEBUGMODE){
288           cout <<whiteb<< "␣Lista␣Desordenada␣" << endl << "␣";
289           for (unsigned int i=0; i<lista.size(); i++) {
290                cout << lista.at(i);
291                if(i<lista.size()-1)
292                     cout << ",";
293                else
294                     cout << endl;
295           }
296           cout <<white<< endl;
297     }
298     // Medición del tiempo de ejecución del algorítmo quickSort();
299     clock_gettime(CLOCK_REALTIME, &t1);
300     int ordenado = seleccion(lista,lista.size()/2);
301     clock_gettime(CLOCK_REALTIME, &t2);
302     tresultado = (double) (t2.tv_sec - t1.tv_sec) + (double) ((t2.tv_nsec - t1.tv_nsec) / (1.e+9));
303 
304     // si hémos ordenado la lista la mostramos
305     if (DEBUGMODE){
306           cout <<whiteb<< "␣Lista␣Modificada␣" << endl << "␣";
307           for (unsigned int i=0; i<lista.size(); i++) {
308                cout << lista.at(i);
309                if(i<lista.size()-1)
310                     cout << ",";
311                else
312                     cout << endl;
313           }
314           cout <<whiteb<< endl;
315           cout << "Valor␣del␣elemento␣seleccionado:␣" << ordenado << endl;
316           cout << "Tiempo␣empleado␣en␣la␣ordenación:␣" << setiosflags(ios::fixed) << setprecision(9)
      << tresultado <<normal<< endl;
317     }else{
318       cout <<whiteb<< "Valor␣del␣elemento␣seleccionado:␣" << ordenado << endl;
319       cout << setiosflags(ios::fixed) << setprecision(9) << tresultado <<normal<< endl;
320     }
321     return 0;
322 }
```

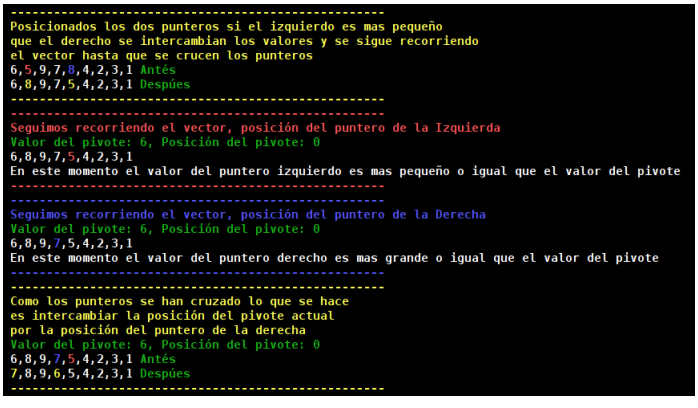
3.4. Traza de una ejecución completa del programa:











3.5. Mediciones del estudio empírico:

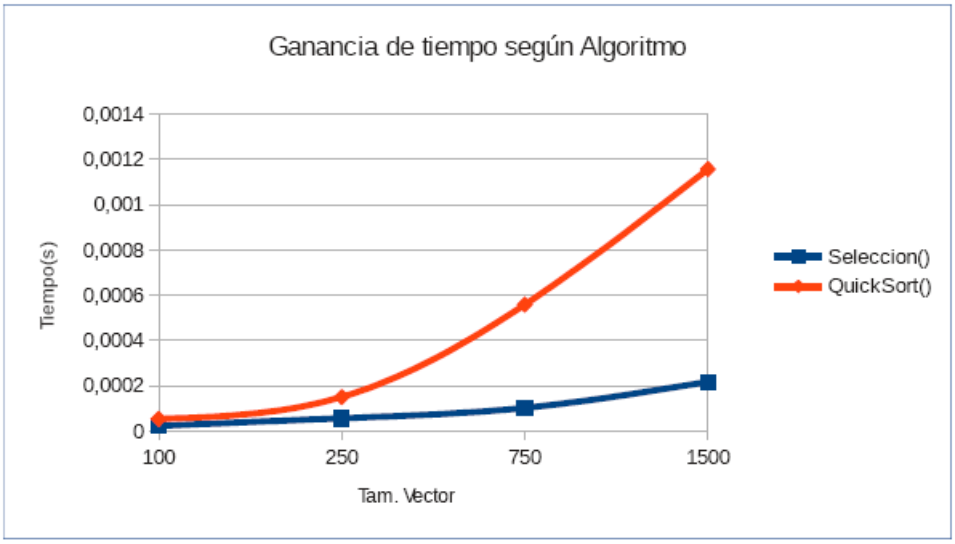
En este apartado lo que vamos ha realizar serán las mediciones necesarias para ver a partir de cuando llega a ser rentable o muy rentable utilizar este método para encontrar el valor de un dato en un vector desordenado con respecto al algoritmo QuickSort(), el cual primero tendría que ordenar el vector y luego extraer el dato de la posición que le hayamos indicado.

Bien aunque las mediciones se van a realizar en el mismo ordenador y en igualdad de condiciones, osea, con el ordenador recién arrancado se indica cual es el modelo de procesador y la cantidad de memoria que posee para posteriores comparaciones con otro tipo de maquina.

Ordenador: Intel(R) Core(TM)2 Quad CPU Q6600 2.40GHz

Memoria: 4 GB de DDR2 1333MHz en 4 bancos de memoria con FSB

Tamaño del Vector	Seleccion()	QuickSort()
100	0,000022061	0,000053141
250	0,000055968	0,00015003
750	0,00010194	0,000557476
1500	0,000216664	0,001155791



4. Tercera Practica.....Corregida

4.1. Presentación del problema:

En esta ocasión tenemos que implementar un algoritmo del tipo voraz, esto quiere decir que tenemos que usar una función de selección muy simple que nos permita reconocer cual es el objeto a analizar sin mucho coste en tiempo así de esa manera el algoritmo conseguir una solución optima, que no tiene que ser necesariamente la mejor, en el menor tiempo posible.

El problema que se plantea para poder solucionar con este tipo de algoritmo es el de coloración de grafos no dirigidos, osea que tenemos un grafo con unos cuantos nodos y unas cuantas aristas conectando dichos nodos y el algoritmo a implementar tiene que ser capaz de poder colorear los distintos nodos de diferentes colores, manteniendo la restricción de que dos nodos conectados entre si no pueden ser del mismo color, y siempre utilizando el mínimo numero de colores para rellenar el grafo.

4.2. Metodología seguida para la solución:

Siguiendo con la metodología seguida en las practicas anteriores vamos a implementar dicho algoritmo en lenguaje c++, para empezar a realizar la practica comenzamos con el esquema general que se encontraba disponible en la pagina 7 del tema 3 de las transparencias de ALG, aunque este esquema no se adapta perfectamente a la practica planteada, si nos sirve para poder comenzar a desarrollar el algoritmo necesario para la coloración de grafos.

Teniendo en cuenta la naturaleza de un grafo, osea que el mismo contiene nodos que contienen los diferentes valores del mismo y que también tiene aristas que unen los diferentes nodos de este, he creado una estructura denominada Nodo() que en su interior contiene:

1. Un identificador de nodo
2. Un identificador del color con el que esta coloreado
3. Un entero con la cantidad de nodos adyacentes
4. Un array de tamaño 5 con el identificador de los nodos adyacentes

Bien, aunque esta sea la manera que he usado para implementar este problema, esta claro que le faltan y le sobran datos, ya que le falta un tipo de dato que contenga el valor propiamente dicho del nodo, osea si el grafo esta destinado para referirse a diferentes ciudades de un mapa faltaría como mínimo un tipo de dato string para poder almacenar en el nodo el nombre de las ciudades a las que nos referimos en el grafo, y si nos fijamos con mas precisión también nos sobra el entero que nos dice cual es la cantidad de nodos adyacentes que tiene el nodo en cuestión, ya que podríamos saber cuantos nodos adyacentes tiene dicho nodos solo con preguntar el tamaño del array que hay en el interior del nodo.

Pues bien, ¿entonces por que no he puesto el valor del nodo y si he puesto el entero con la cantidad de adyacentes?, bien, esto lo he hecho así por que al poner el tipo de dato con lo que seria el valor de del nodo es bastante fácil de implementar, y el dato que me sobra lo he puesto simplemente por comodidad, osea lo uso solo para no tener que comprobar el tamaño del array.

Bien, entrando en materia y fijándonos en el main() del programa podemos ver que solo contiene 4 lineas, la primera genera un array de un tamaño predefinido en el programa, aunque esto es fácilmente modificable para que sea el usuario el que ponga cual seria el tamaño del grafo, el cual contendrá los diferentes nodos del grafo por eso lo he denominado *elGrafo[] y para que sea mas fácil trabajar con el mientras se lo pasamos a las diferentes funciones lo he puesto como un puntero. La segunda linea no es mas que una función que se encarga de generar el grafo y colocarlo en el array que hemos creado con antelación. La tercera linea es la que se encarga de colorear el grafo, y por ultimo tenemos la función que se encarga de imprimir por pantalla el array resultado de colorear dicho grafo.

Por ultimo, explicare superficialmente como funciona el procedimiento de coloración del grafo.

En primer lugar accedemos al primer nodo del array que contiene el grafo y lo abrimos, lo primero que miramos es si esta coloreado o no y cuantos nodos adyacentes tiene, esto nos sirve para saber si abrimos o no el nodo, osea si esta coloreado no lo abrimos y si no esta coloreado y tiene nodos adyacentes lo abrimos, de esta manera solo tenemos que analizar una sola vez cada uno de los nodos del grafo, una vez comprobado que el nodo tiene nodos adyacentes a el, "saltamos" al primer nodo adyacente del nodo que estamos analizando ahora mismo, y comprobamos si esta coloreado, si estuviera coloreado pasaríamos al siguiente nodo para comprobar lo mismo, seguiríamos hasta el final de los nodos adyacentes del nodo que estamos analizando, si el nodo no estuviera coloreado lo que haríamos seria coger el primer color que tenemos disponibles para poder pintar y lo asignaríamos a un color temporal, y ahora recorreríamos los nodos adyacentes del nodo adyacente, osea que en el algún momento estaríamos mirando el nodo padre, osea el nodo que estamos analizando ahora mismo, esto lo hacemos para comparar los colores que pueden tener los nodos adyacentes de este nodo y lo comparamos con el color temporal que hemos escogido si el color temporal que hemos escogido ya esa asignado a algún nodo adyacente de este ultimo nodo lo que hacemos es sustituir el color temporal por el siguiente disponible y volvemos a recorrer los nodos adyacentes de este ultimo nodo, si al recorrer todos los nodos adyacentes del nodo el color es valido para ser utilizado pues coloreamos el nodo y pasamos al siguiente nodo adyacente del nodo padre o nodo analizado, continuamos así hasta colorear todos los nodos adyacentes del nodo analizado, una vez hecho eso volveríamos al nodo padre o nodo analizado y lo colorearíamos con el siguiente color disponible, des esta forma nos aseguramos que no se puede repetir el color de los nodos adyacentes con el color del nodo padre.

Bien, hasta el momento hemos coloreado el nodo root y todos sus nodos adyacentes, ahora lo que hacemos es avanzar al siguiente nodo del grafo por regla general este nodo ya estaría coloreado, pues podría ser un nodo adyacente del nodo root, en ese caso como ya esta coloreado no tendríamos que abrirlo para analizarlo y en consecuencia pasaríamos al siguiente nodo en el grafo, en este punto dependerá muchísimo de la forma del grafo para saber si el siguiente tenemos que colorearlo o no.

En definitiva este procedimiento se tendría que realizar con todos los nodos del grafo. De esta forma es como consigo que se coloree todo el grafo aprovechando al máximo la variedad de colores existentes.

4.3. Código Fuente

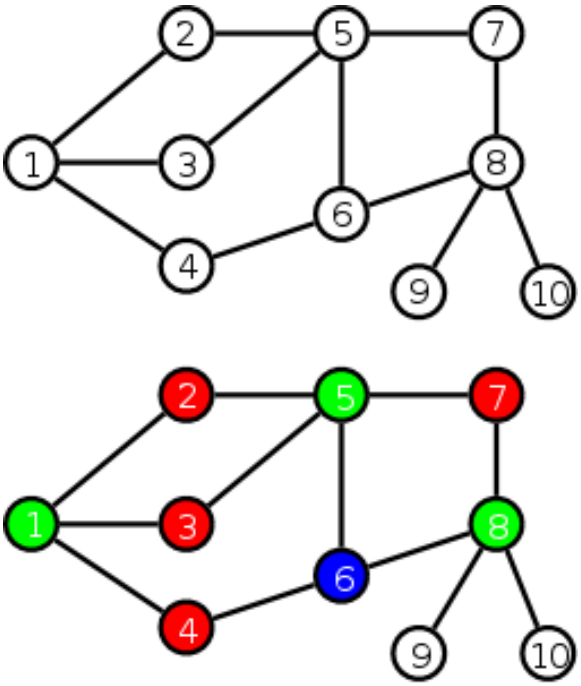
```cpp
/*
 * coloracion.cpp
 *
 *      Fecha: 03/06/14
 *      Desarrollado: Carlos de la Torre
 *
 */
#include <iostream>
#include <vector>
#include "../inc/concolor.h"

#define TAM_GRAFO 10 // Esta es la cantidad de nodos que puede tener el gráfo

using namespace std;

// De que colores se pueden pintar los nodos, el último es para saber la cantidad de colores que hay Cantidad = 6;
enum Color { Blanco, Rojo, Verde, Azul, Amarillo, Magenta, Cian, Cantidad};

// Estructura de un nodo
struct miNodo{
    int numero_nodo;
    Color color;
    int cantidad_nodos_adyacentes;
    miNodo *nodos_adyacentes[5];
};

// Creamos el Grafo que queremos colorear
void CrearGrafo(miNodo *grafo[]){
    miNodo *uno, *dos, *tres, *cuatro, *cinco, *seis, *siete, *ocho, *nueve, *diez;

    // Creando los nodos del gráfo
    uno = new miNodo;
    dos = new miNodo;
    tres = new miNodo;
    cuatro = new miNodo;
    cinco = new miNodo;
    seis = new miNodo;
    siete = new miNodo;
    ocho = new miNodo;
    nueve = new miNodo;
    diez = new miNodo;

    // Hasta aquí hémos creado los nodos a partir de
    // aquí es donde cremos los arcos del gráfo
    uno->numero_nodo = 1;
    uno->color = Color(Blanco);
    uno->cantidad_nodos_adyacentes = 3;
    uno->nodos_adyacentes[0] = dos;
    uno->nodos_adyacentes[1] = tres;
    uno->nodos_adyacentes[2] = cuatro;
    dos->numero_nodo = 2;
    dos->color = Color(Blanco);
    dos->cantidad_nodos_adyacentes = 3;
    dos->nodos_adyacentes[0] = uno;
    dos->nodos_adyacentes[1] = cinco;
    dos->nodos_adyacentes[2] = ocho;
    tres->numero_nodo = 3;
    tres->color = Color(Blanco);
    tres->cantidad_nodos_adyacentes = 2;
    tres->nodos_adyacentes[0] = uno;
    tres->nodos_adyacentes[1] = cinco;
    cuatro->numero_nodo = 4;
    cuatro->color = Color(Blanco);
    cuatro->cantidad_nodos_adyacentes = 2;
    cuatro->nodos_adyacentes[0] = uno;
    cuatro->nodos_adyacentes[1] = seis;
```

```
67      cinco->numero_nodo = 5;
68      cinco->color = Color(Blanco);
69      cinco->cantidad_nodos_adyacentes = 3;
70      cinco->nodos_adyacentes[0] = dos;
71      cinco->nodos_adyacentes[1] = tres;
72      cinco->nodos_adyacentes[2] = seis;
73      seis->numero_nodo = 6;
74      seis->color = Color(Blanco);
75      seis->cantidad_nodos_adyacentes = 3;
76      seis->nodos_adyacentes[0] = cuatro;
77      seis->nodos_adyacentes[1] = cinco;
78      seis->nodos_adyacentes[2] = siete;
79      siete->numero_nodo = 7;
80      siete->color = Color(Blanco);
81      siete->cantidad_nodos_adyacentes = 4;
82      siete->nodos_adyacentes[0] = seis;
83      siete->nodos_adyacentes[1] = ocho;
84      siete->nodos_adyacentes[2] = nueve;
85      siete->nodos_adyacentes[3] = diez;
86      ocho->numero_nodo = 8;
87      ocho->color = Color(Blanco);
88      ocho->cantidad_nodos_adyacentes = 2;
89      ocho->nodos_adyacentes[0] = siete;
90      ocho->nodos_adyacentes[1] = dos;
91      nueve->numero_nodo = 9;
92      nueve->color = Color(Blanco);
93      nueve->cantidad_nodos_adyacentes = 1;
94      nueve->nodos_adyacentes[0] = siete;
95      diez->numero_nodo = 10;
96      diez->color = Color(Blanco);
97      diez->cantidad_nodos_adyacentes = 1;
98      diez->nodos_adyacentes[0] = siete;
99
100     // aqui metemos los nodos del gráfo en un array
101     // para poder recorrerlos lo ideal seria en una
102     // pila para sacarlos mas rapidamente
103     grafo[0] = uno;
104     grafo[1] = dos;
105     grafo[2] = tres;
106     grafo[3] = cuatro;
107     grafo[4] = cinco;
108     grafo[5] = seis;
109     grafo[6] = siete;
110     grafo[7] = ocho;
111     grafo[8] = nueve;
112     grafo[9] = diez;
113 }
114
115 // Colorear el grafo dado
116 void ColorearGrafo(miNodo *grafo[]){
117     int NodosSinPintar = TAM_GRAFO;
118     int color;
119     bool color_valido;
120
121     while(NodosSinPintar > 0){
122         for(unsigned int num_nodo = 0; num_nodo < TAM_GRAFO; num_nodo++){ // Recorro todos
    los Nodos del Grafo
123             if (grafo[num_nodo]->cantidad_nodos_adyacentes > 0){ // compruebo si el nodo
    tiene nodos adyacentes
124                 for (int num_nodo_adya = 0; num_nodo_adya < grafo[num_nodo]->
    cantidad_nodos_adyacentes;++num_nodo_adya){// recorremos todos los nodos adyacentes
125                     if(grafo[num_nodo]->nodos_adyacentes[num_nodo_adya]->color == Color(
    Blanco)){// comprobamos si el nodo adyacente no lo hemos pintado
126                         color = Color(Rojo); // como no hemos pintado el nodo adyacente
    ponemos el color al primero que se puede usar
127                         color_valido = false; // y esta variable auxiliar nos dirá si
    podemos usar el color temporal que hémos escogido
```

```
128                         if (grafo[num_nodo]->nodos_adyacentes[num_nodo_adya]->
    cantidad_nodos_adyacentes > 0){
129                             // con esto lo que hacemos es visitar los nodos adyacentes del
    nodo adyacente que estamos coloreando
130                             // osea que alguno de estos nodos será el nodo désde donde
    venimos
131                             for (int num_nodo_adya2 = 0; num_nodo_adya2 < grafo[num_nodo]->
    nodos_adyacentes[num_nodo_adya]->cantidad_nodos_adyacentes;++num_nodo_adya2){//
    recorremos los nodos adyacentes
132                                 if (grafo[num_nodo]->nodos_adyacentes[num_nodo_adya]->
    nodos_adyacentes[num_nodo_adya2]->color != static_cast<Color>(color))
133                                     // y en este if es donde comprobamos si el color que
    hemos elegido podemos usarlo para pintar el nodo adyacente
134                                     color_valido = true;
135                                 else
136                                     // si no pudieramos usar el color actualmente asignado
    por que ya hay un nodo adyacente con ese color elegimos el siguiente
137                                     color++;
138                             }
139                         }
140                         if (color_valido) // como ya sabemos que tenemos el color correcto
    para colorear el nodo adyacente lo coloreamos
141                             grafo[num_nodo]->nodos_adyacentes[num_nodo_adya]->color =
    static_cast<Color>(color);//pintamos el nodo de otro color que el nodo padre
142                         NodosSinPintar--; // quitamos un nodo del vector sin colorear
143                     }
144                 }
145             }else{ // si no tuviera nodos adyacentes seria un grafo con un solo nodo
146                 if (grafo[num_nodo]->color == Color(Blanco))
147                     grafo[num_nodo]->color = static_cast<Color>(color);
148                 NodosSinPintar--; // quitamos un nodo del vector sin colorear
149             }
150         }
151     }
152 }
153
154 // Funcion para imprimir por pantalla el Grafo dado
155 void ImprimirGrafo(miNodo *grafo[]){
156     for(unsigned int i = 0; i < TAM_GRAFO; i++){
157         if (grafo[i]->color == Blanco)
158             cout<<whiteb;
159         if (grafo[i]->color == Rojo)
160             cout<<redb;
161         if (grafo[i]->color == Verde)
162             cout<<greenb;
163         if (grafo[i]->color == Azul)
164             cout<<blueb;
165         if (grafo[i]->color == Amarillo)
166             cout<<yellowb;
167         if (grafo[i]->color == Magenta)
168             cout<<magentab;
169         if (grafo[i]->color == Cian)
170             cout<<cyanb;
171         cout << "␣" << grafo[i]->numero_nodo;
172         cout<<normal;
173     cout <<endl;
174     }
175 }
176
177 int main(int argc, const char * argv[]){
178     miNodo *elGrafo[TAM_GRAFO];
179     CrearGrafo(elGrafo);
180     ColorearGrafo(elGrafo);
181     ImprimirGrafo(elGrafo);
182     return 0;
183 }
```

4.4. Traza del Programa

```
[usuario@portatil Practica3_ALG]$ ./bin/coloracion
1
2
3
4
5
6
7
8
9
10
[usuario@portatil Practica3_ALG]$
```



5. Cuarta Practica

5.1. Presentación del problema:

En esta ocasión tenemos que implementar un algoritmo del tipo programación dinámica para solucionar el problema del cambio de monedas. En un principio este problema tendría fácil solución si utilizáramos recursividad, pero esta forma seria bastante ineficiente por que en cuanto tuviéramos un sistema monetario de mas de 11 monedas los cálculos que se tendrían que realizar par recorrer el árbol de soluciones serian bastante lentos.

5.2. Metodología seguida para la solución:

Al igual que en las anteriores practicas hemos buscado cual era el lenguaje mas cómodo para realizaría, y llegue a la conclusión que el mas cómodo seria JAVA pero al final la realice en C por que esta practica ya estaba hecha en JAVA en diversos puntos de la red pero no encontré ninguna implementación en C así que me decidí a ponerle un punto mas de dificultad a la misma usando un lenguaje un "tanto" mas complicado que java para desarrollarla.

Lo primero que tenemos que explicar para poder entender esta practica es la función de recurrencia, en este caso no explicaremos como conseguirla (para eso esta la teoría) pero si explicaremos que es cada una de sus partes:

$$Cambio(k,q) = \begin{cases} 0 & Si\, q = 0 \\ +\infty & Si\, q < 0\ ó\ k \leq 0 \\ min\left\{Cambio(k-1,q), 1 + Cambio(k, q-c_k)\right\} & \end{cases}$$

La primera parte es Cambio(k,q), esto es igual que una función en cualquier lenguaje de programación, osea que tenemos dos parámetros de entrada que son k que es el tipo de moneda que vamos a utilizar para devolver el importe, que viene dado por el parámetro q.

Después lo que hay dentro de la llave el primer renglón nos viene a decir el primer caso base de la función de recurrencia, y es que el valor de Cambio(k,q) tiene que ser 0 si el valor a devolver es 0.

El segundo caso base se usa para tener una situación de parada para el procedimiento que vamos a emplear, y es que si el valor a devolver q es menor de 0 es un estado ilógico de la función y si el tipo de moneda que vamos a usar para realizar la devolución es menor o igual a el tipo 0 también es un caso ilógico de la función.

Y por ultimo y no por eso el menos importante la parte del caso general en donde se comprueba el principio de optimalidad de Bellman, el cual reza: *La solución óptima de un problema se obtiene combinando soluciones óptimas de subproblemas.* Que lo que viene a decir es que el valor de la función Cambio(k,q) es el mínimo de entre dos valores, que son el valor de la función Cambio(k,q) pero con un tipo de moneda menor del que tiene actualmente y el valor de 1 mas el valor de la función Cambio(k,q) pero esta vez quitandole a la cantidad a devolver, el valor de la moneda que actualmente estamos evaluando.

Después de haber dado un rápido repaso a lo que es la función de recursividad, podemos confundirnos en que esta metodología de trabajo usa la recursividad para realizar el algoritmo pero nada mas lejos de la realidad, de hecho todo el proceso de construcción de la tabla y posterior reorganización de la misma permite usar la solución sin recurrir a la tan tediosa recursividad.

Bien en esta practica tuve bastantes dificultades para poder ver/entender el concepto de como construir la tabla de datos que se utiliza para poder resolver el problema con programación dinámica, mi gran problema deriva, de que el hecho de tener que construir una tabla pasa por que hay que poner filas y columnas por lo tanto, para construir esta tabla de *subtotales*, también tendría que haber filas y columnas... Pero en realidad para poder realizar la practica hace falta abstraerse de esta idea y pensar que las columnas de esta tabla son valores del dinero que tenemos que devolver al usuario, teniendo en cuenta que el valor de cada columna se incrementa en la cantidad del valor mas bajo del grupo de monedas y las filas es el valor de cada una de las monedas que podemos tener en el monedero.

El párrafo anterior que indica como construir la tabla y que escrito parece bastante complicado (que lo es) de entender se entiendo muchísimo mas fácil con un ejemplo sencillo, para ello vamos a recurrir al siguiente ejemplo en el cual tenemos un monedero con 3 monedas de 1, 4 y 6 Euros y el valor que tenemos que devolver es de 8 Euros. Pues bien a continuación presentamos una serie de tablas que simulan la forma en la que se rellenaría la tabla de *subtotales.*

Cantidad	0	1	2	3	4	5	6	7	8
M_0=1									
M_1=4									
M_2=6									

Esta sería nuestra tabla objetivo, osea que tendríamos que buscar la manera en la que con dos bucles anidados se construyera esta tabla, por supuesto siempre teniendo en cuenta lo que ya se ha comentado en párrafos anteriores, que los valores de las columnas tendrían que ir aumentando de valor, del mismo valor que la moneda mas pequeña, como en este caso es de 1 pues las columnas van de 1 en 1. También hay que tener en cuenta de incluir la columna 0 que será la encargada de certificar uno de los casos base de la función de recurrencia, explicada anteriormente.

Cantidad	0	1	2	3	4	5	6	7	8
M_0=1	**0**								
M_1=4	**0**								
M_2=6	**0**								

En esta segunda tabla lo que hacemos es rellenar la columna cero para asegurarnos que cumplimos con el primer caso base, este paso se separa del anterior caso para recalcar que a la hora de crear la tabla hay que inicializarla y con esto no se pretende insinuar que hay que inicializar la tabla por completo a 0 si no que la columna 0 hay que rellenarla con 0 para que las soluciones a los microproblemas tengan coherencias, en este caso cuando tenemos una cantidad de 0 para devolver claramente tenemos que devolver 0 monedas.

Cantidad	0	1	2	3	4	5	6	7	8
M_0=1	**0**	**1**							
M_1=4	**0**								
M_2=6	**0**								

De acuerdo, en este tercer paso es cuando empieza a funcionar el algoritmo en si, ahora mismo vamos a rellenar la casilla en donde se encuentra el **1**, osea $M_0 = 1$ y valor a devolver 1 entonces lo que hacemos es que usaremos el caso general de la ecuación de recurrencia $min\{Cambio(k-1,q), 1 + Cambio(k, q-c_k)\}$ en donde la parte $Cambio(k-1,q)$ tiene que devolver $+\infty$ por que M_0 es la primera moneda por lo tanto no podemos coger la moneda anterior (restarle 1) y $1 + Cambio(k, q-c_k)$ tiene que devolver 1 por que el tipo de moneda que estamos evaluando es M_0 y el valor que tenemos que

devolver es 1 la función nos quedaría *Cambio(1,1-1)* si nos fijamos en la tabla el valor de esta función es **0** (por que lo que vamos a devolver seria 0 y estamos evaluando la moneda M_0) pero como ha este resultado se le sumaria 1, entonces el mínimo de los dos valores $+\infty$ y 1 es **1**.

Con esto podemos darnos cuenta que el segundo caso base de la ecuación de recurrencia se usa aquí y es el que contempla que no puede haber tipos de monedas negativas o que el cambio a devolver sea también un valor negativo (Ninguna de las dos situaciones seria logica).

Cantidad	0	1	2	3	4	5	6	7	8
M_0=1	**0**	**1**	**2**	**3**	**4**				
M_1=4	**0**	**1**	**2**	**3**	**1**				
M_2=6	**0**	**1**	**2**	**3**					

Vale, seguimos rellenando la tabla de la misma manera que el paso anterior hasta que llegamos a una situación diferente,bien en este caso haremos lo mismo que antes, analizar las partes para la evaluación del mínimo de los dos valores de la función $Cambio(k, q)$,en el primer caso usaremos la moneda anterior a la que estamos analizando, osea que usaremos M_0 ya que estamos analizando M_1 al fijarnos en la tabla podemos darnos cuenta que el valor que da es **4** y si realizamos la operación de la segunda parte del mínimo el valor que nos da será **0** ya que tenemos que devolver 4 y el valor de la moneda actual $M_1 = 4$, osea 4 - 4 = 0 es lo que tenemos devolver y al fijarnos en la tabla esta claro cual es el valor que tenemos que devolver pero aun nos falta sumarle el 1 por lo tanto el mínimo de 4 y de 1 será **1.**

Cantidad	0	1	2	3	4	5	6	7	8
M_0=1	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
M_1=4	**0**	**1**	**2**	**3**	**1**	**2**	**3**	**4**	**2**
M_2=6	**0**	**1**	**2**	**3**	**1**	**2**	**1**	**2**	**2**

Haciendo los siguientes pasos tal y como se ha dicho en los párrafos anteriores, tanto cuando tenemos una opción de poner una moneda del tipo anterior, como cuando no la tenemos y tenemos que devolver $+\infty$ en la función $Cambio(k, q)$, hasta que llegamos a la posición que se muestra en la tabla, que seria el resultado final de construir la tabla de subtotales o lo que es lo mismo las soluciones a los microproblemas de un problema mayor.

Una vez que hemos construido la tabla con sus resultados parciales, lo único que nos queda es poder reconstruir la solución correcta al problema principal, en nuestro caso seria un array o vector con los tipos de monedas que tenemos que utilizar para realizar la devolución y la cantidad de cada tipo de monedas que tenemos que dar a nuestro cliente para devolverle todo el dinero que le corresponde.

Para realizar este procedimiento lo que vamos a hacer es recomponer la solución correcta a partir de los datos que hemos sacado de la tabla de soluciones parciales, ya que en esta tabla podemos ver cuales son las monedas usadas para cada uno de los valores intermedios que se han ido devolviendo antes de llegar a la solución final, utilizaremos estos datos para determinar cual seria la menor cantidad de monedas que tendríamos que usar y el valor que tendrían estas para devolver todo el dinero.

Para poder reconstruir la solución correcta lo que hacemos es utilizar el ultimo valor con el que fue rellenada la tabla osea un **2** y a partir de ahí vamos a ir mirando si con una moneda menos podemos devolver el dinero correspondiente pero con una moneda del tipo inmediatamente anterior, osea que tendría un valor mas bajo que la moneda que estamos analizando en el momento actual.

Si el valor es mas bajo o igual que la moneda que estamos analizando lo que hacemos es volver a realizar el análisis pero esta vez con la moneda del tipo anterior, si al realizar este análisis nos se cumple que la cantidad de monedas que tenemos que utilizar es menor o igual que la que estamos analizando añadimos la moneda actual como una moneda que pertenece a la solución.

A partir de ahora lo que vamos a hacer será ir quitando de la cantidad que tenemos que devolver el valor de la moneda actual, y seguiremos así hasta que se cumpla o bien que hayamos devuelto todo el dinero a nuestro cliente o bien que el valor de la moneda que estamos analizando sea superior al valor que nos queda por devolver en cuyo caso lo que haremos será realizar todo el proceso de nuevo con la moneda inmediatamente anterior, si se diera el caso que ya hemos llegado a la moneda con el valor mas pequeño y aun así no pudiéramos devolver todo el dinero a nuestro cliente se diría que nuestro sistema monetario es inconsistente.

Para poder ver claramente como se realiza el proceso de reconstrucción de la solución utilizaremos nuestro ejemplo anterior para poder ilustrarlo.

Solución	
M_0=1	**0**
M_1=4	**0**
M_2=6	**0**

Solución	
M_0=1	**0**
M_1=4	**1**
M_2=6	**0**

Solución	
M_0=1	**0**
M_1=4	**2**
M_2=6	**0**

Sabemos que la solución final tiene que tener **2** monedas así que lo primero que hacemos es comprobar si con las monedas del tipo inmediatamente superior podemos devolver por completo el dinero con la misma cantidad de monedas, en este caso si **2,** como es igual escogemos esta moneda y volvemos a comprobar si es posible devolver todo el dinero con la moneda inmediatamente anterior **8** en este caso no es posible, por lo tanto lo que hacemos es quitar el valor de la moneda a el dinero total que tenemos que devolver 8 - 4 =4 y sumamos 1 a el vector solución.

Bien ya tenemos la primera moneda en el vector solución ahora repetiremos el ultimo paso hasta que lleguemos a devolver todo el dinero o tengamos que cambiar de moneda 4 - 4 = 0 como hemos llegado a devolver todo el dinero terminamos el programa y sumamos 1 mas al vector solución

5.3. Código Fuente

```c
/*
 * monedas.c
 *
 *         Fecha: 03/06/14
 *         Desarrollado: Carlos de la Torre
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
 * Con esto podemos saber el tamaño de una array en C
 */
#define NELEMENTOS(x) (sizeof(x)/sizeof(x[0]))

/*
 * Variable global para saber el nombre del programa
 */
char* nomPrograma;

/*
 * Este es un truco para poder pasar matrices entre
 * funciones en C sin tener que usar punteros
 */
typedef unsigned int matriz[15][9999]; // <-- Definir los tamaños máximos para el monedero y la vuelta

/*
 * Con esto sacamos por pantalla el valor
 * de todas las monedas que tiene el monedero
 */
void imprimeMonedero(int MONEDERO[], int CANTI_MONEDAS){
      int i = 0;
      printf ("Monedero: [");
      for (;i<CANTI_MONEDAS;++i){
            printf ("%d",MONEDERO[i]);
            if (i < CANTI_MONEDAS-1)
                  printf (", ");
            else
                  printf("]");
      }
      printf("\n");
}
```

```
44
45 /*
46  * Con esto sacamos por pantalla el valor
47  * de la tabla que contiene la cantidad de
48  * monedas a devolver según cuanto tengamos
49  * que devolver
50  * Pasamos el monedero por parámetros para
51  * poder hacer los incrementos de las columnas
52  * por el valor de la moneda mas pequeña
53  */
54 void imprimeTabla(int MONEDERO[], matriz TABLA, int FIL, int COL){
55      int fil = 0, col = 0;
56      printf("\nA Devolver ---> ");
57      for (col = 0;col<=COL;col+=MONEDERO[0]){ // <-- el incremento es igual a la moneda mas pequeña
58           if (MONEDERO[0] > (COL-col)) // <-- El valor de la moneda es mas grande que lo que me
     queda por recorrer
59                printf("%2d ",COL);
60           else
61                printf("%2d ",col);
62      }
63      printf("\n_________________");
64      for (col = 0;col<COL;col+=MONEDERO[0]){
65           printf("___");
66      }
67      printf("_\n");
68      for (fil = 0;fil<FIL;fil++){
69           printf("Mon. tipo %d -> ",fil+1);
70           for (col = 0;col<=COL;col+=MONEDERO[0]){
71                if (MONEDERO[0] > (COL-col)) // <-- El valor de la moneda es mas grande que lo que me
     queda por recorrer
72                     printf("%2d ",TABLA[fil][COL]);
73                else
74                     printf("%2d ",TABLA[fil][col]);
75           }
76           printf("\n");
77      }
78 }
79
80 /*
81  * Esta función controla los posibles errores del programa
82  */
83 void funError(int ERROR){
84      switch (ERROR){
85      case 1:
86           printf(" La cantidad de parametros es incorrecta \n");
87           break;
88      case 2:
89           printf(" La cantidad valores de las monedas es diferente al numero de monedas \n");
90           break;
91      case 3:
92           printf(" No hay una solución optima posible \n");
93           break;
94      }
95
96      printf(" Uso: %s [Cantidad Monedas] [Valor Monedas separadas por ,] [Euros a devolver]\n\n",
     nomPrograma);
97      printf(" Ejemplo: %s 3 1,2,5 13\n",nomPrograma);
98      exit(-ERROR);
99 }
100
101 /*
102  * Función de mínimo en C
103  * la he creado yo por que no he encontrado
104  * ninguna librería de C que la tuviera
105  */
106 int min(int VALOR_A, int VALOR_B){
107      return (VALOR_A<VALOR_B)?VALOR_A:VALOR_B;
108 }
109
110 /*
111  * Con esta función comprobamos cuantas monedas tenemos
112  * que usar para devolver el dinero que nos queda
113  */
114 int Cambio(int MONEDERO[], int CANTI_MONEDAS, int DEVOLVER){
115      // Este es el indice que uso para los for de recorrido
116      int idx = 0;
117      /* Este vector guarda la cantidad de monedas que hemos usado
118         según sea del primer tipo de monedas o del segundo tipo
```

```
119         así sucesivamente la posición es el tipo de monedas
120         y el valor es la cantidad de veces que la hemos usado
121         según la vuelta que necesitamos dar en cada iteración */
122      int monedas_usadas[CANTI_MONEDAS];
123      /* Esta variable la uso para no tener que modificar
124         la cantidad de monedas que me llegan a la función */
125      int auxCantiMonedas = CANTI_MONEDAS;
126      /* Con este for inicializo todos el vector de monedas
127         usadas a 0 de esa forma cada vez que llamo a cambio
128        me aseguro de la veracidad de la cantidad de monedas
129        se supone que el for debería llegar hasta el tamaño
130        total de monedas_usadas menos 1 que es el último
131        elemento de monedas_usadas */
132      for (idx = 0;idx<CANTI_MONEDAS;idx++)
133           monedas_usadas[idx] = 0; // <-- inicializo el vector a 0
134
135      /* Me aseguro que la variable que voy a devolver esté a 0 */
136      int cantidad_monedas = 0;
137
138      /* El condicional que viene a continuación contempla los
139         dos casos base de la función de recurrencía y también
140         el caso general de la misma */
141      if (DEVOLVER==0)
142           cantidad_monedas = 0;
143      else if (DEVOLVER<0 || auxCantiMonedas<=0)
144           cantidad_monedas = 999999; // <-- mas infinito
145      else{
146           while (DEVOLVER>0 && auxCantiMonedas > 0){ // <-- Mientras tenga que devolver y no este en
     la ultima moneda
147                if (DEVOLVER >= MONEDERO[auxCantiMonedas-1]){ // <-- Si el valor de la moneda es mas
     pequeño de lo que tengo que devolver
148                     DEVOLVER -= MONEDERO[auxCantiMonedas-1]; // <-- resto el valor de la moneda a lo
      que me queda por devolver
149                     monedas_usadas[auxCantiMonedas-1]++; // <-- Y añado una moneda de ese tipo a la
     solución
150                }else // <-- Si el valor de la moneda es mas grande que lo que me queda por devolver
151                     auxCantiMonedas--; // <-- cojo la siguiente moneda mas pequeña
152           }
153      /* Este for se encarga de recorrer el vector de monedas
154         usadas para contarlas y acumularlas en cantidad_monedas
155         que es la variable que se devuelve */
156           for (idx=0;idx<=CANTI_MONEDAS-1;idx++){
157                cantidad_monedas += monedas_usadas[idx];
158           }
159      }
160      return cantidad_monedas;
161 }
162
163 /*
164  * Este es el menú principal
165  */
166 int main(int argc, const char *argv[]){
167      if (argc == 1)
168           funError(1);
169      int misMonedas[atoi(argv[1])];// <-- Cantidad de Monedas disponibles
170      nomPrograma = argv[0];
171      char* parametro2 = argv[2];
172      //int misMonedas[] = {1,2,5,9};
173      char *charMoneda = malloc(2);
174      int CD = 0, M = 0, V = 0;
175      int idx = 0; // <-- Indice de cualquier for
176
177      int DEBUGMODE = 0;
178      if ((argc == 5) && (argv[4] != NULL))
179           DEBUGMODE = 1;
180      else
181           DEBUGMODE = 0;
182
183      if (argc == 4 || argc == 5){
184           charMoneda = strtok(parametro2,",");
185           while (charMoneda){
186                misMonedas[idx] = atoi(charMoneda);
187                idx++;
188                charMoneda = strtok(NULL,",");
189           }
190           if (idx!=NELEMENTOS(misMonedas))
191                funError(2);
192           CD = atoi(argv[3]);
```

```
193     }else if (argc < 4)
194         funError(1);
195
196     matriz tablaDinamica;
197     // OJO: aquí hay un "fallo" y es que solo se pueden tener
198     // monederos de 15 monedas y solo se puede devolver hasta 9999 €
199     // Se define en la parte superior del programa.
200
201     /*
202      * Este es el núcleo del algoritmo:
203      * -El 1er for recorre la cantidad que tenemos que devolver desde 0 ya que también se contempla
        la posibilidad de no devolver nada
204      * -El 2do for recorre la cantidad de monedas, nótese que este no recorre hasta <= si no que se
        detiene en el ultimo elemento del array
205      * -El condicional se encarga de llenar las posiciones 0 en los elementos que no tenemos que
       devolver dinero en otro caso ver memoria
206      * para poder entender por que es necesario un par de diagramas para entenderlo
207      * Comentarios para entender el código:
208      * -Variable misMonedas[]: Este sería el monedero del ejercicio
209      * -Variable tablaDinamica: Tabla dinámica que almacenas la cantidad de monedas a devolver en
       cada caso
210      * -Variable CD: esta es la cantidad que le tenemos que devolver al cliente.
211      * -Variable CM: esta es la cantidad de monedas optima que se tienen que devolver al final.
212      * -Variable M: estas son la cantidad de monedas disponibles que tendrá la función Cambio() en
       cada momento.
213      * -Variable V: este sera el valor que tendrá que devolver Cambio() en cada iteración
214      * -Variable C1 y C2: Son variables auxiliare para hacer el código mas legible
215      */
216     unsigned int C1, C2;
217     for (V = 0;V<=CD;V+=misMonedas[0]){ // aquí es donde tengo que poner un if para ver si me paso
       de lo que voy a devolver
218         for (M = 0;M<NELEMENTOS(misMonedas);M++){
219             if(V==0){
220                 tablaDinamica[M][V] = 0; // <-- Cuando el valor a devolver es 0 pongo en la
       tabla 0 monedas a devolver
221             }else{
222                 C1 = Cambio(misMonedas,M+1,V); // <-- Comprobamos el cambio con una moneda mas
       de otro tipo
223                 C2 = 1+Cambio(misMonedas,M,V-misMonedas[M]); // <-- Comprobamos el cambio con
       las mismas monedas pero con menos a devolver
224                 tablaDinamica[M][V] = min(C1,C2); // <-- Comprobamos con cual de las dos
       situaciones damos menos monedas
225             }
226         }
227     }
228
229     /*
230      * Aquí presentamos los datos que tenemos hasta el momento
231      */
232     printf ("Cantidad a devolver: %d €\n",CD);
233     /*
234      * Como en C no hay un length o size, hay que estar
235      * continuamente pasando el tamaño del vector
236      * como los de la matriz
237      */
238     imprimeMonedero(misMonedas,NELEMENTOS(misMonedas));
239     if (DEBUGMODE)
240         imprimeTabla(misMonedas,tablaDinamica,NELEMENTOS(misMonedas),CD);
241
242     /*
243      * Las variables aux1, aux2 y tamSol son para que el código sea mas fácil de leer
244      * esta parte es donde se recompone la solución a partir de los datos que hemos
245      * rellenado en la tabla
246      * Tanto esta parte como la anterior se podrían meter en una función pero al tener
247      * que pasar tantos parámetros he decidido dejarlo en el main
248      */
249     int aux1 = NELEMENTOS(misMonedas)-1; // <-- Le pongo -1 por que es el ultimo elemento del array
250     int aux2 = CD; // <-- Esta es la cantidad de dinero a devolver
251     int solucion[NELEMENTOS(misMonedas)];// <-- Array Solución
252
253     for (idx = 0; idx<NELEMENTOS(solucion);idx++) // <-- Inicializo solución a 0
254         solucion[idx] = 0;
255
256     while (aux1 >= 0 && aux2 != 0){
257         if (aux1 > 0 && tablaDinamica[aux1][aux2]>=tablaDinamica[aux1-1][aux2]) // <-- El orden de
       las comprobaciones IMPORTANTE
258             aux1 -= 1;
259         else{
```

```
260                 solucion[aux1] += 1;
261                 aux2 = aux2 - misMonedas[aux1];
262             }
263         }
264 
265         printf ("\nSolucion: \n");
266         for (aux1 = 0 ;aux1<NELEMENTOS(solucion);aux1++){ // <-- Como tengo que llegar al final de los
        elementos uso <=
267             printf("%2d Monedas de %d €\n",solucion[aux1],misMonedas[aux1]);
268         }
269 
270         return 0;
271 }
```

5.4. Traza del Programa

```
[usuario@portatil Practica4_ALG]$ ./bin/monedas 3 1,3,5 11 1
Cantidad a devolver: 11 €
Monedero: [1, 3, 5]

A Devolver -->  0  1  2  3  4  5  6  7  8  9 10 11
_____________________________________________________
Mon. tipo 1 ->  0  1  2  3  4  5  6  7  8  9 10 11
Mon. tipo 2 ->  0  1  2  1  2  3  2  3  4  3  4  5
Mon. tipo 3 ->  0  1  2  1  2  1  2  3  2  3  2  3

Solucion:
 0 Monedas de 1 €
 2 Monedas de 3 €
 1 Monedas de 5 €
[usuario@portatil Practica4_ALG]$
```