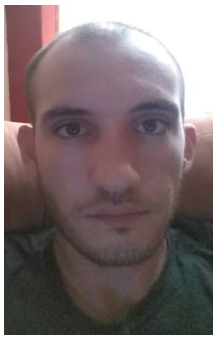


Prácticas de Algorítmica

Jesús Ángel González Novez
76440070F

Nota obtenida:
Los 2 puntos relativos a prácticas,corregidas a través de correos electrónicos (jesusgonzaleznovez@gmail.com)

PRACTICA 1:ANALISIS DE ALGORITMOS

Enunciado

Estudiar t(n) en el caso promedio, para las instrucciones de asignación. Usar probabilidades.

i:= 1
mientras i ≤ n hacer
    si a[i] ≥ a[n] entonces
        a[n]:=a[i]
    finsi
    *i:= i *2*
finmientras

cont:=0
para i:= 1,...,n hacer
    para j:= 1,...,i-1 hacer
        si a[i] < a[j] entonces
            cont:= cont + 1
        finsi
    finpara
finpara

Uso del programa

./practica1
ó
./practica1 N

Si se utiliza la primera opción el tamaño del vector será 10, si se utiliza la segunda opción el tamaño de vector será N.

Código del programa

```
#include <iostream>
#include <cstdlib>
using namespace std;
void mostrar(int v[],int n)
{
 for(int i=0; i<n; i++)
  cout << v[i] << " ";
 cout << endl;
}
int main(int argc,char *argv[])
{
 int N;
 if(argc < 2)
  N=10;
 else
  N=atoi(argv[1]);
 int a[N];
 int inst1=0,inst2=0;
 //Primera función
 for(int i=0; i<N; i++)
  a[i] = rand()%1000;
 for(int i=0; i<N; i++){
  if(a[i] >= a[N-1]){
```

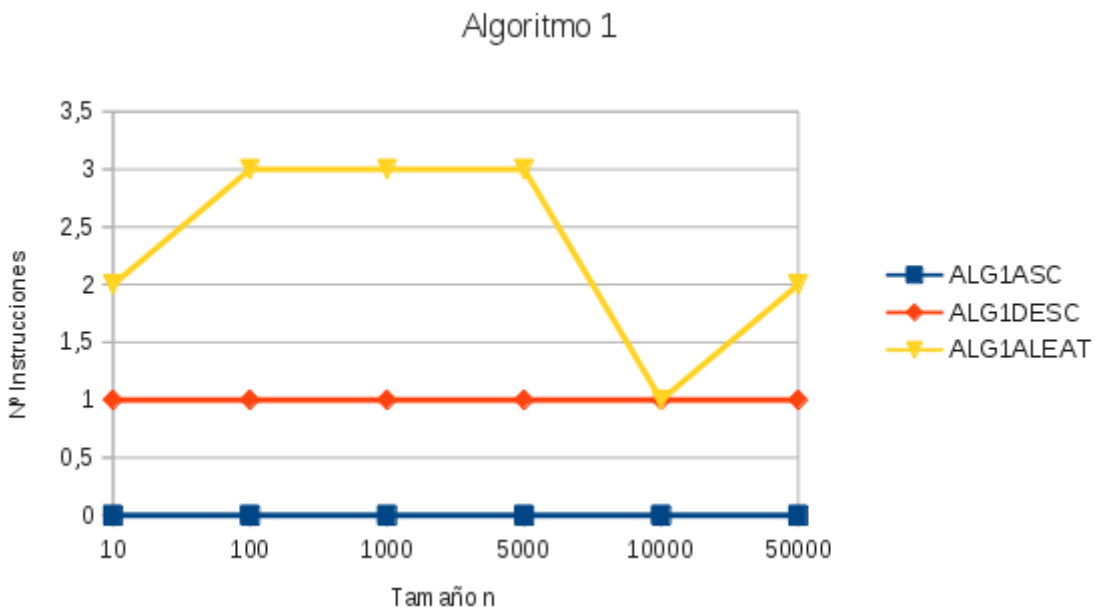
```
   a[N-1] = a[i];
   inst1++;
  }
  i *= 2;
 }

 //Segunda función
 for(int i=0; i<N; i++)
  a[i] = rand()%1000;
 int cont=0;
 for(int i=0; i<N; i++){
  for(int j=0; j<i; j++){
   if(a[i]<a[j]){
    inst2++;
    cont++;
   }
  }
 }
 cout << "N:" << N << endl;
 cout << "Número de asignaciones primer algoritmo:" << inst1 << endl;
 cout << "Número de asignaciones segundo algoritmo:" << inst2 << endl;
}
```

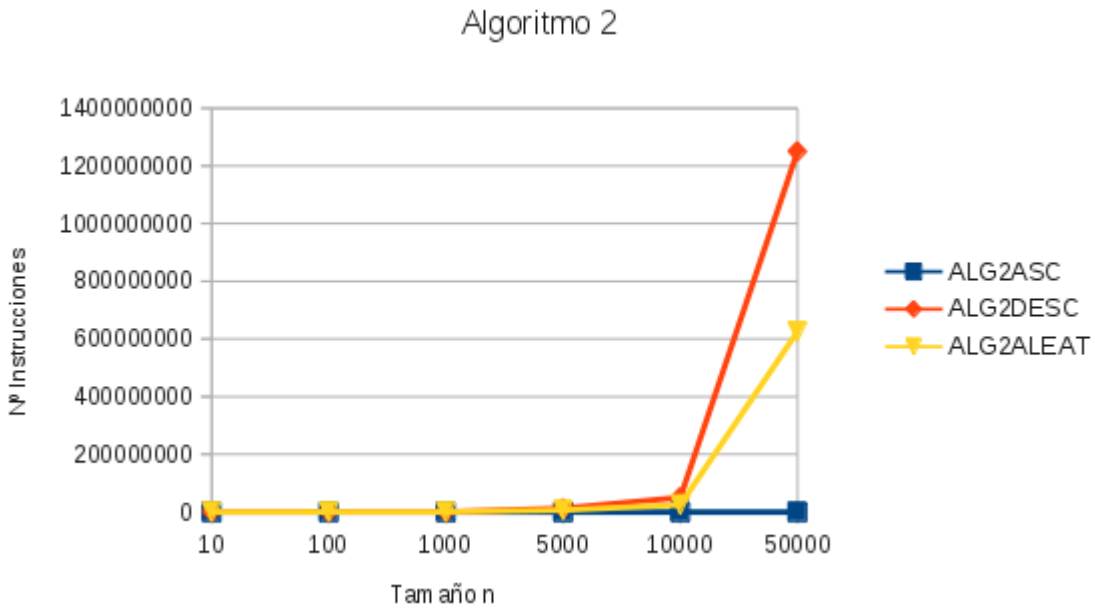
Conclusión

	ALG1ASC	ALG1DESC	ALG1ALEAT
10	0	1	2
100	0	1	3
1000	0	1	3
5000	0	1	3
10000	0	1	1
50000	0	1	2

	ALG2ASC	ALG2DESC	ALG2ALEAT
10	0	45	20
100	0	4950	2669
1000	0	499500	248288
5000	0	12497500	6282977
10000	0	49995000	24761979
50000	0	1249975000	625607873

Como podemos observar el algoritmo 1 no llega hacer asignación si el vector esta ordenado ascendentemente, hace una si esta ordenado descendentemente y en el caso de que no este ordenado que sería el caso promedio vemos que los valores oscilan entre 1 y 3 asignaciones.
En el segundo algoritmo tampoco se llega a hacer asignaciones si el vector esta ordenado ascendentemente, si esta ordenado de forma descendente hace muchisimas asignaciones y finalmente en el caso de no ordenado hace también muchisimas asignaciones pero la mitad que si lo esta de forma descendente.Se añaden unas gráficas para poder mostrarlo con claridad.





Analizando primer algoritmo

Primeramente tenemos un if, que si tomamos el caso en el que el vector no este ordenado no podremos saber siempre cuando será true o false. Dado que el incremento es por 2, tendriamos una sumatoria desde 0 a $\log_2 N$. El mejor caso sería que siempre fuese false el if, por tanto tendriamos la sumatoria anterior de 2+3(por la comprobación del if) y el peor que caso que siempre fuese true en el que tendríamos la sumatoria anterior pero además +3 adicionales por la linea dentro del if.

Si tenemos en cuenta que la probabilidad en este caso sería $1/((\log_2 N)+1)$, 1 es de que llegemos al final de las iteraciones. Por tanto podriamos definir J como el numero de veces que será falso y K como el que será true, si tenemos en cuenta eso:

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

El 1 de dentro de los corchetes es porque hay que contar siempre la asignación inical del algoritmo.Si desarrollamos lo anterior queda lo siguiente:

$$t(n) = \sum_{i=1}^{log_2 n} \frac{1}{log_2 n + 1} \times [1 + 9J + 6(log_2 n - J)] \Rightarrow \sum_{i=1}^{log_2 n} \frac{1}{log_2 n + 1} \times [1 + 9J + 6log_2 n - 6J] \Rightarrow$$

$$\sum_{i=1}^{log_2 n} \frac{1}{log_2 n + 1} \times [1 + 3J + 6log_2 n] \Rightarrow \frac{1}{log_2 n + 1} \times \left(\sum_{i=1}^{log_2 n} 1 + (6\, log_2 n \times \sum_{i=1}^{log_2 n} 1) + 3 \times \sum_{i=1}^{log_2 n} J \right) \Rightarrow$$

$$\frac{1}{log_2 n + 1} \times \left((x+1) + 6\, log_2 n\, (x+1) + \frac{3\, log_2 n\, (x+1)}{2} \right) \Rightarrow 1 + 6\, log_2 n + \frac{3}{2}\, log_2 n$$

$$t(n) = 1 + 6\, log_2 n + \frac{3}{2}\, log_2 n \Rightarrow O(log_2 n)$$

Analizando el segundo algoritmo:
Siguiendo la metodología de análisis anterior analizaremos este algoritmo. Primero tenemos un for de 1 a n y el segundo for sería una sumatoria de 1 a i-1, a continuación tenemos el if que serían 3 instrucciones para la comprobación y otras si se cumpliese la condición, si analizamos todo tenemos que en el peor de los casos t(n) sería la sumatoria de 1 a n de la sumatoria de 1 a i-1 de 1+3+3 pero si es el mejor de los casos sería solo de 1+3. Si seguimos el mismo proceso probabilístico visto anteriormente tendríamos que:

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

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

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

Siendo k las veces en que el if es true.

PRACTICA 2:DIVIDE Y VENCERAS

Enunciado

Hay que seleccionar la mediana de un vector pagina 38 del tema 2, concretamente hay que intentar implementar la función selección en cualquier lenguaje y hacer un estudio empírico con capturas de pantalla de las diferentes ejecuciones .

Uso del programa

g++ -fopenmp practica2.cpp -o practica2(se usa -fopenmp para poder medir los tiempos con omp.h)
./practica2 N opcion
N indica el tamaño del vector.
Opcion indica siendo 1 que estará ordenado el vector, y distinto de 1 que no lo estará.

Ejemplos:
./practica2 1000 1 → vector ordenado de tamaño 1000
./practica2 1000 2 → vector no ordenado de tamaño 1000

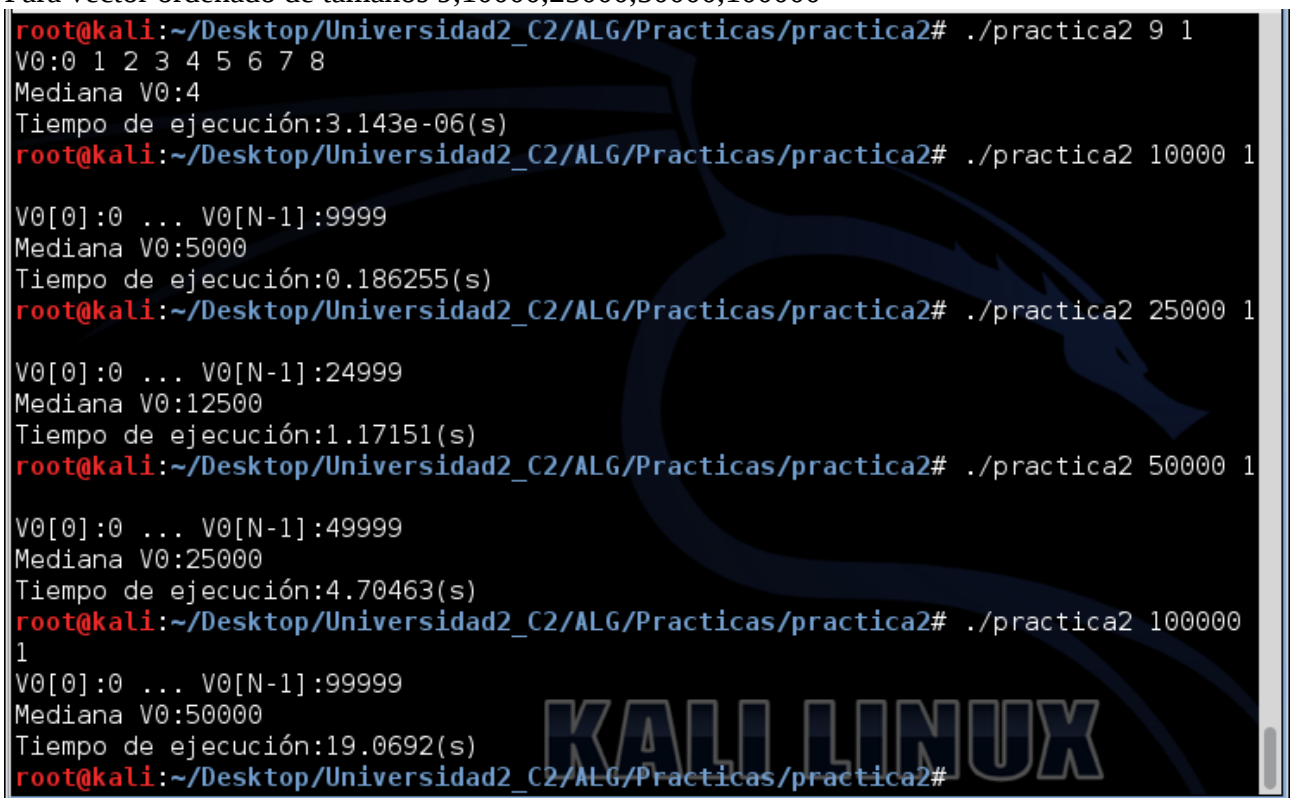
Código del programa

```
#include <iostream>
#include <cstdlib>
#include <omp.h>
using namespace std;
void pivote(int vector[],int N,int i,int j,int &l)
{
 int p,k,aux;
 p = vector[i];
 k = i;
 l = j + 1;
 for(k = k+1; vector[k]<p && k<j; k++);
 for(l = l-1; vector[l]>p; l--);
 while(k<l)
 {
  aux = vector[k];
  vector[k] = vector[l];
  vector[l] = aux;
  for(k = k+1; vector[k]<=p; k++);
  for(l = l-1; vector[l]>p; l--);
 }
 aux = vector[i];
 vector[i] = vector[l];
 vector[l] = aux;
}

int seleccion(int vector[],int N,int s)
{
 int i,j,l;
 i = 0;
 j = N-1;
 do
 {
  pivote(vector,N,i,j,l);
```

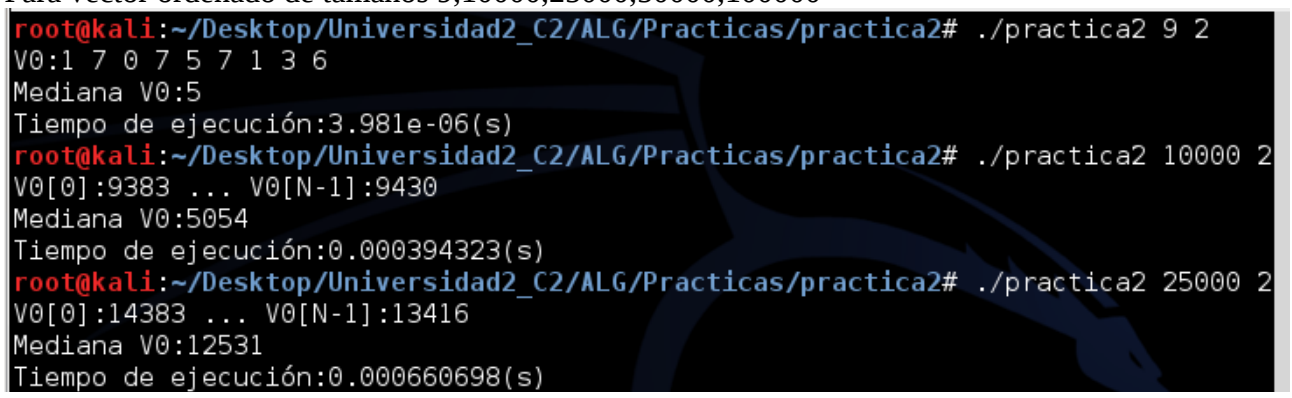
```
  if(s<l) j = l - 1;
  else if(s>l) i = l +1;
 }while(l != s);
 return vector[l];
}

void mostrar(int v[],int N)
{
 for(int i=0; i<N; i++)
  cout << v[i] << " ";
 cout << endl;
}

int main(int argc,char *argv[])
{
 if(argc < 3){
  cout << "Uso ./practica2 N opcion" << endl;
  cout << "N=tamaño" << endl;
  cout << "\033[22;32mopcion= si es 1 ordenado, si distinto de 1
aleatorio" << endl;
  return 0;
 }
 int resultado, mediana;
 int N=atoi(argv[1]);
 int ordenado=atoi(argv[2]);
 int *v0 = (int*) malloc(10*N);
 double t1,t;
 //Rellenamos vector
 if(ordenado == 1)
  for(int i=0; i<N; i++)  v0[i] = i;
 else
  for(int i=0; i<N; i++)  v0[i] = rand()%N;

 //Mostramos vector
 if(N<10){
  cout << "V0:";
  mostrar(v0,N);
 }else{
  cout << "V0[0]:" << v0[0] << " ... V0[N-1]:" << v0[N-1] << endl;
 }
 mediana = N / 2;
 t1 = omp_get_wtime();
 resultado = seleccion(v0,N,mediana);
 t = omp_get_wtime()-t1;
 cout << "Mediana V0:" << resultado << endl;
 cout << "Tiempo de ejecución:" << t << "(s)" << endl;
 free(v0);
}
```

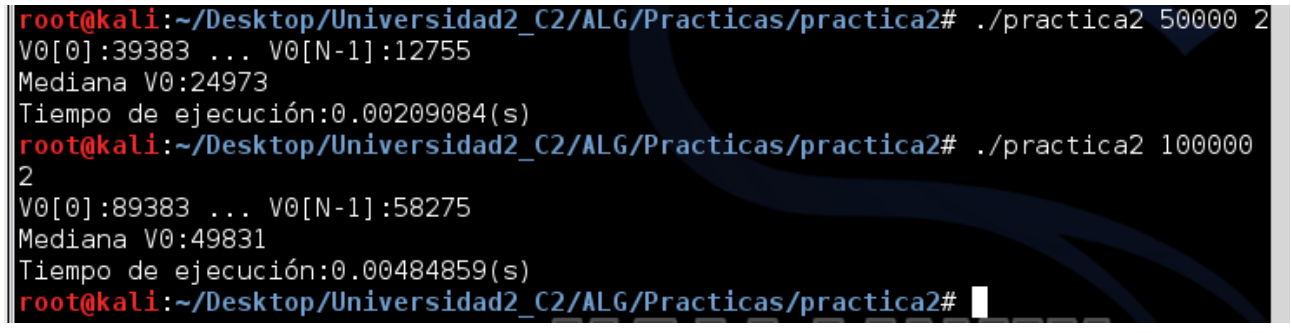
Capturas de pantalla

Para vector ordenado de tamaños 9,10000,25000,50000,100000



Para vector ordenado de tamaños 9,10000,25000,50000,100000





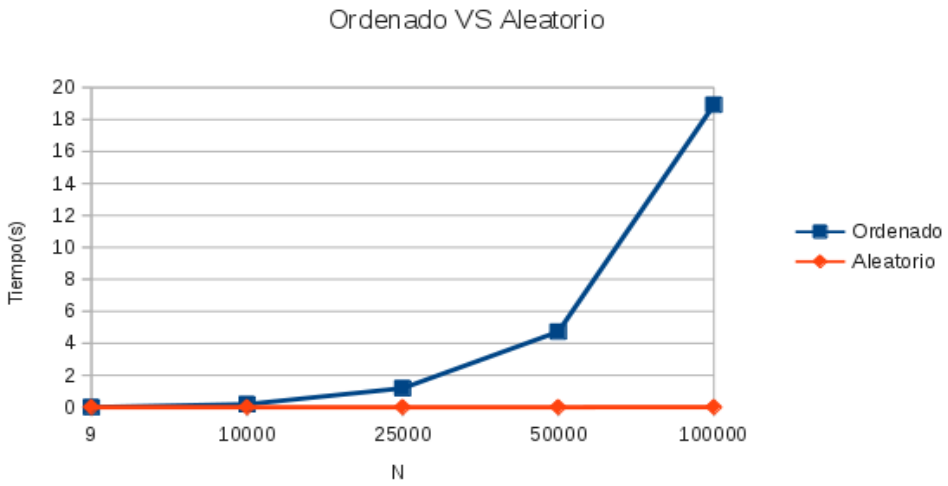
Conclusión

Para medir los tiempos se ha usado la función omp_get_wtime() de omp.h, la hemos estado usando este cuatrimestre en la asignatura Arquitectura de Computadores y la verdad es muy cómoda y precisa.

Sistema Operativo: Kali Linux(Debian,32 bits)

Se puede observar claramente que cuando esta ordenado es mucho peor el resultado.

	Ordenado	Aleatorio
9	0,000003	0,000003
10000	0,185369	0,000396139
25000	1,18371	0,000697435
50000	4,72463	0,0020877
100000	18,9224	0,00496669



PRACTICA 3:ALGORITMOS VORACES

Enunciado

Implementar un algoritmo voraz decente para el problema de coloración de grafos y mostrar varios ejemplos de ejecución de nuestro programa.

Uso del programa

g++ practica3.cpp -o practica3
./practica3

Código del programa

```
#include <iostream>
#include <cstdlib>
#include <omp.h>
using namespace std;
#define N 5

/*-----Códigos de los colores usados --------------
  black \033[22;30m
  red \033[22;31m
  green \033[22;32m
  brown \033[22;33m
  blue \033[22;34m
  yellow \033[01;33m
  cyan \033[22;36m
  white \033[01;37m
----------------------------------------------------*/

//Estructuras auxiliares----------------------------
struct nodo{
 int id;
 int color;
 int n_aristas;
 int * aristas;
};
struct grafo{
 int size;
 nodo *nodos;
};
//----------------------------------------------------

//Pasamos el nodo y el número de aristas que queramos que tenga.
void creaArista(nodo &n, int n_a){
 n.n_aristas = n_a;
 n.aristas = new int[n_a];
}

//Liberamos toda la memoria del grafo que pasamos como argumento.
void liberar(grafo &g){
 for (int i = 0; i < g.size; i++){
  delete[] g.nodos[i].aristas;
```

```
 }

 delete[] g.nodos;
}

//Muestra el grafo indicado con colores (si está coloreado).
void pintaGrafo(grafo &g){
 for (int i = 0; i < g.size; i++){
  cout << "\033[01;37mNodo: ";
  switch(g.nodos[i].color)
  {
   case 0: cout << "\033[22;30m" << g.nodos[i].id;
   break;
   case 1: cout << "\033[22;32m" << g.nodos[i].id;
   break;
   case 2: cout << "\033[22;31m" << g.nodos[i].id;
   break;
   case 3: cout << "\033[22;34m" << g.nodos[i].id;
   break;
   case 4: cout << "\033[22;33m" << g.nodos[i].id;
   break;
   case 5: cout << "\033[01;33m " << g.nodos[i].id;
   break;
   case 6: cout << "\033[22;36m" << g.nodos[i].id;
     break;
     case 7: cout << "\033[01;37m" << g.nodos[i].id;
     break;
  }
  cout << "\033[01;37m --> ";
  for (int j = 0; j < g.nodos[i].n_aristas; j++){
   switch(g.nodos[g.nodos[i].aristas[j]].color)
   {
    case 0: cout << "\033[22;30m"<< g.nodos[i].aristas[j] << " ";
    break;
    case 1: cout << "\033[22;32m" << g.nodos[i].aristas[j] << " ";
    break;
    case 2: cout << "\033[22;31m" << g.nodos[i].aristas[j] << " ";
    break;
    case 3: cout << "\033[22;34m"<< g.nodos[i].aristas[j] << " ";
    break;
    case 4: cout << "\033[22;33m" << g.nodos[i].aristas[j] << " ";
    break;
    case 5: cout << "\033[01;33m " << g.nodos[i].aristas[j] << " ";
    break;
    case 6: cout << "\033[22;36m" << g.nodos[i].aristas[j] << " ";
    break;
    case 7: cout << "\033[01;37m" << g.nodos[i].aristas[j] << " ";
    break;
   }
```

```
   }
   cout << "\033[01;37m" << endl;
  }
}

//Solución
bool solucion(grafo &g){
 bool resultado = true; //Inicialmente es true hasta que encuentra un
nodo blanco.

 for (int i = 0; i < g.size && resultado; i++){
  if (g.nodos[i].color == 7){ //Si el color del nodo es blanco no
tenemos solución.
   resultado = false;
  }
 }

 return resultado;
}

//Colorear el grafo
void coloracion(grafo &g){
 cout << endl << "\033[01;37mColoreando..." << endl;

 int color_actual = 1;

 while(!solucion(g)){ //Solución recorre todos los nodos para
comprobar si hay alguno sin pintar (blanco).
  for (int i = 0; i < g.size; i++){
   if (g.nodos[i].color == 7){ //Si el nodo está sin colorear.
    bool colorear = true;

    for (int j = 0; j < g.nodos[i].n_aristas && colorear; j++){
//Comprobamos que los vecinos no tienen el color actual.
     if (g.nodos[g.nodos[i].aristas[j]].color == color_actual){
      colorear = false;
     }
    }

    if (colorear){ //Se puede colorear con el color actual.
     g.nodos[i].color = color_actual;
    }
   }
  }

  color_actual++;
 }
}

//Creación del grafo
```

```
void creaGrafo(grafo &g){
 g.size = N;
 g.nodos = new nodo[N];

 for (int i = 0; i < N; i++){
  g.nodos[i].id = i;
  g.nodos[i].color = 7; //White.
 }

 creaArista(g.nodos[0], 1);
 creaArista(g.nodos[1], 2);
 creaArista(g.nodos[2], 3);
 creaArista(g.nodos[3], 2);
 creaArista(g.nodos[4], 2);

 g.nodos[0].aristas[0] = 2;
 g.nodos[1].aristas[0] = 3;
 g.nodos[1].aristas[1] = 4;
 g.nodos[2].aristas[0] = 0;
 g.nodos[2].aristas[1] = 3;
 g.nodos[2].aristas[2] = 4;
 g.nodos[3].aristas[0] = 1;
 g.nodos[3].aristas[1] = 2;
 g.nodos[4].aristas[0] = 1;
 g.nodos[4].aristas[1] = 2;
}

//Función main
int main(int argc,char *argv[])
{
 grafo g;
 creaGrafo(g);
 pintaGrafo(g);
 coloracion(g);
 pintaGrafo(g);
 liberar(g);
}
```

Capturas de pantalla

Conclusión

Sistema Operativo: Kali Linux(Debian,32 bits)

Existen mas combinaciones posibles, pero vemos que en esta con 5 nodos hemos usado 3 colores, es una solución media pero existen mejores.

PRACTICA 4:PROGRAMACION DINAMICA

Enunciado

Estudiar e implementar el algoritmo del cambio de monedas.

Uso del programa

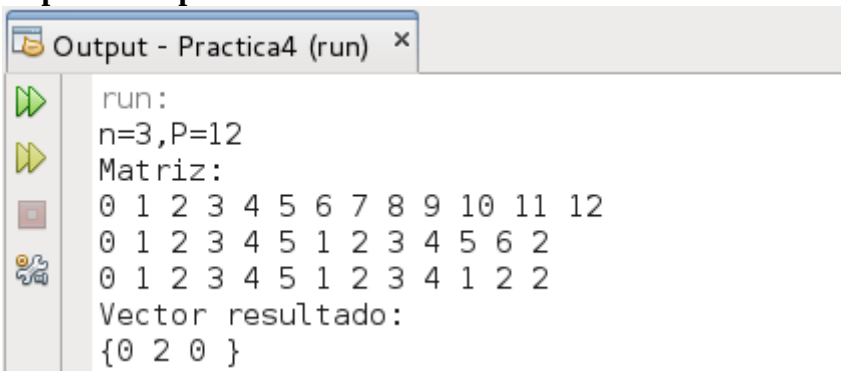
java -jar Practica4.jar

Código del programa

```
package practica4;
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;
  }

  public int[][] getMatrizCambio() {
    return this.matrizCambio;
  }

  private int[][] calcularMonedas(int cantidad, int[] monedas) {

    int[][] matrizCambio2 = new int[monedas.length + 1][cantidad + 1];

    for (int i = 0; i < monedas.length; i++) {
      matrizCambio2[i][0] = 0;
    }

    for (int j = 1; j <= cantidad; j++) {
      matrizCambio2[0][j] = 99;
    }

    for (int i = 1; i <= monedas.length; i++) {
      for (int j = 1; j <= cantidad; j++) {
        if (j < monedas[i - 1]) {

          matrizCambio2[i][j] = matrizCambio2[i - 1][j];
        } else {

          int minimo = 0;
```

```
          minimo = min(matrizCambio2[i - 1][j], matrizCambio2[i][j -
monedas[i - 1]] + 1);
          matrizCambio2[i][j] = minimo;

        }
      }
    }

    return matrizCambio2;
  }

  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;
    }
  }

}
```

Capturas de pantalla



Conclusión

El problema del cambio de moneda como vemos nos permitirá obtener el menor número de monedas posible para pagar una cantidad dada con unos tipos de monedas concretas.
Te pondre un ejemplo para explicartelo mejor:
Suponte que tienes que pagar 14€
Tienes billetes de 1€,2€,5€,10€,20€,50€
P = 14€
monedas[] = {1,2,5,10,20,50}
n = monedas.size() = 6

Ahora se pueden dar dos casos:
Por cada tipo de moneda que tenemos algunas sobrepasan directamente a P y otras no llegan a satisfacerla.
Billete de 50€>14€ ó 20€>14€ por tanto se descartan y pasamos a otro tipo de moneda.
Para el resto de billetes vemos que son < 14€.
Vamos a ver una tabla de ejemplo:
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:

	0,00 €	**1,00 €**	**2,00 €**	**3,00 €**	**4,00 €**	**5,00 €**	**6,00 €**	**7,00 €**	**8,00 €**	**9,00 €**	**10,00 €**	**11,00 €**	**12,00 €**	**13,00 €**	**14,00 €**
1,00 €	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2,00 €	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
5,00 €	0	1	1	2	2	1	2	2	3	3	2	3	3	4	4
10,00 €	0	1	1	2	2	1	2	2	3	3	1	2	2	4	3
~~**20,00 €**~~	~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~1~~	~~2~~	~~2~~	~~3~~	~~3~~	~~1~~	~~2~~	~~2~~	~~4~~	~~3~~
~~**50,00 €**~~	~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~1~~	~~2~~	~~2~~	~~3~~	~~3~~	~~1~~	~~2~~	~~2~~	~~4~~	~~3~~

Para que quede mas claro, supongamos por ejemplo la columna de 5€, pues si nos encontramos en la fila de 1€ tendriamos que dar 5 monedas de valor menor o igual a 1€, si nos encontramos en la fila del de 2€ tendriamos que dar 3 monedas de valor menor o igual a 2€, en ese caso serían 2 monedas de 2€ y una de 1€, si vamos a la fila de 5€ necesitariamos 1 moneda de valor menor o igual a 5€, en este caso 1 de 5€, si vamos a la fila de 10€ tenemos que dar una moneda de valor menor o igual a 10€, es decir una de 5€, y asi sucesivamente.

Si te fijas los billetes de 20€ y 50€ estan descartados, por lo que n ahora es n=4
La solución será matriz[n,P] basandonos en esto tenemos que la solucion es sin duda matriz[4,14]
Esto si nos fijamos da la fila del 10€ ,última columna.
Esto que nos dice que para pagar 14€ nos basta con 3 monedas de valor menor o igual a 10€, y

realmente es asi 10€+2€+2€ = 14€
Pero como saber que tipo de moneda devolver?
Partimos de la casilla final, en este caso si se incluyen los de 20 y 50.
Vamos comprobando si m[i][j] ha variado respecto de m[i-1][j] es decir la casilla de justo encima, si ha variado se habrá utilizado una moneda de ese tipo, si no esque no se ha utilizado, por tanto si nos fijamos vemos que las filas del de 20 y 50 no varian con respecto a la de 10 precisamente por eso!