|  |  |  |
| --- | --- | --- |
| **CARRERA:**  Ingeniería de Software | **GUÍA**  No. 01 | **TIEMPO ESTIMADO:**  1 mes |
| **ASIGNATURA:**  Estructura de datos  NRC: 2967 | **FECHA DE ELABORACION:** 23-10-2019  **SEMESTRE**: septiembre 2019 – febrero 2020 | |
| **TÍTULO:**  Backtracking | **DOCENTE:** Ing. Fernando Solis | |

**OBJETIVO**

Comprender y aplicar de manera eficiente el uso de Backtracking para la generación de códigos o algoritmos más eficientes, en este caso para la realización del proyecto.

**INSTRUCCIONES**

**Backtracking**

En su forma básica, la idea de backtracking se asemeja a un recorrido en profundidad dentro de un grafo dirigido. El grafo en cuestión suele ser un árbol, o por lo menos no contiene ciclos. Sea cual sea su estructura, existe sólo implícitamente.

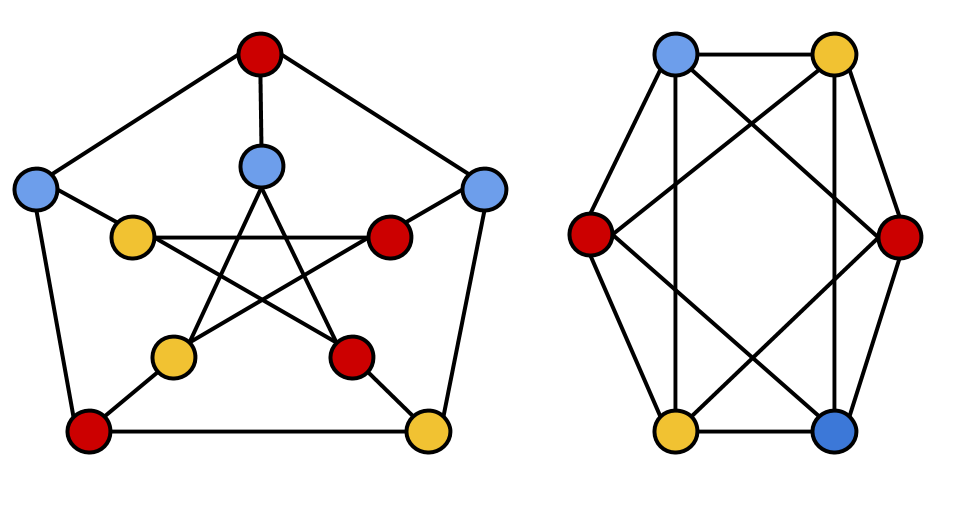
El objetivo del recorrido es encontrar soluciones para algún problema. Esto se consigue construyendo soluciones parciales a medida que progresa el recorrido; estas soluciones parciales limitan las regiones en las que se puede encontrar una solución completa.

El recorrido tiene éxito si, procediendo de esta forma, se puede definir por completo una solución. En este caso el algoritmo puede bien detenerse (si lo único que se necesita es una solución del problema) o bien seguir buscando soluciones alternativas (si deseamos examinarlas todas). Por otra parte, el recorrido no tiene éxito si en alguna etapa la solución parcial construida hasta el momento no se puede completar.

En tal caso, el recorrido vuelve atrás exactamente igual que en un recorrido en profundidad, eliminando sobre la marcha los elementos que se hubieran añadido en cada fase. Cuando vuelve a un nodo que tiene uno o más vecinos sin explorar, prosigue el recorrido de una solución.

Esencialmente, la idea es encontrar la mejor combinación posible en un momento determinado, por eso, se dice que este tipo de algoritmo es una búsqueda en profundidad. Durante la búsqueda, si se encuentra una alternativa incorrecta, la búsqueda retrocede hasta el paso anterior y toma la siguiente alternativa. Cuando se han terminado las posibilidades, se vuelve a la elección anterior y se toma la siguiente opción (hijo [si nos referimos a un árbol]). Si no hay más alternativas la búsqueda falla. De esta manera, se crea un árbol implícito, en el que cada nodo es un estado de la solución (solución parcial en el caso de nodos interiores o solución total en el caso de los nodos hoja).

Normalmente, se suele implementar este tipo de algoritmos como un procedimiento recursivo. Así, en cada llamada al procedimiento se toma una variable y se le asignan todos los valores posibles, llamando a su vez al procedimiento para cada uno de los nuevos estados. La diferencia con la búsqueda en profundidad es que se suelen diseñar funciones de cota, de forma que no se generen algunos estados si no van a conducir a ninguna solución, o a una solución peor de la que ya se tiene. De esta forma se ahorra espacio en memoria y tiempo de ejecución.



**ACTIVIDADES**

1. **Ubicación de recursos**
2. El grupo está conformado por Angel Cárdenas y Edison Báez.
3. Se utilizo DEV C++
4. **Planteamiento del problema**

Realizar cualquier algoritmo que tenga el uso de backtracking de manera fácil y eficiente. En este caso realizamos el mapa de colores o coloreo de mapas.

1. **Entregable (s)**

**Mapas.cpp**

/\*---------------------------------------------

\* Universidad de las Fuerzas Armadas Espe \*

\* Nombres: Angel Cárdenas, Edison Baéz \*

\* NRC: 2967 \*

\* Fecha: 07/11/2019 \*

\* \*

\*----------------------------------------------\*/

#include<iostream>

#include <sstream>

#include <vector>

#include <stdio.h>

#include <string>

#include <stdlib.h>

#include"Ingreso.h"

#include"Ayuda.h"

#define MAX 1000

using namespace std;

bool mt[MAX][MAX];

int V,E;

bool freeColor(int v, int color[], int c){

for (int i=0;i<V;i++){

if(mt[v][i] && c== color[i]){

return false;

}

}

return true;

}

bool btColoring(int m, int color[],int v){

if(v==V) {

return true;

}

for(int i=1; i<=m; i++){

if(freeColor(v,color, i)){

color[v]=i;

if(btColoring(m, color, v+1)){

return true;

}

color [v] =0;

}

}

return false;

}

void solve(int color[]){

for(int i=0;i<V;i++){

cout<<color[i]<<" ";

}

cout<<"\n";

}

bool isColored(int C){

int color[V];

for(int i=0;i<V;i++){

color[i]=0;

}

if(!btColoring(C,color,0)){

return false;

}

solve(color);

return true;

}

int main(){

int C;

int a,b;

Ingreso ingresar;

Ayuda ayudar;

string numV;

cout<<">Ingrese los vertices y las aristas del grafo a colorear"<<endl;

numV=ingresar.leer("Vertices del grafo: ",1);

istringstream(numV)>>V; //vertices

numV=ingresar.leer("Aristas del grafo: ",1);

istringstream(numV)>>E;//aristas

cout<<">Ingrese las coordenadas de cada nodo: "<<endl;

while(E--){

numV=ingresar.leer("Coordenada1: ",1);

istringstream(numV)>>a;

numV=ingresar.leer("Coordenada2: ",1);

istringstream(numV)>>b;

mt[a][b]=true;

mt[b][a]=true;//grafico no dirigido (adyacencia)

}

numV=ingresar.leer("++INGRESE LOS COLORES QUE DESEA PARA LA SOLUCION++",1);

istringstream(numV)>>C;

if(isColored(C)){

cout<<"Esa es la solucion con "<<C<<" colores"<<endl;

}else

{

cout<<"No hay solucion con "<<C<<" color/es"<<endl;

}

ayudar.mostrarAyuda();

return 0;

}

**Ayuda.h**

/\*---------------------------------------------

\* Universidad de las Fuerzas Armadas Espe \*

\* Nombres: Angel Cárdenas, Edison Baéz \*

\* NRC: 2967 \*

\* Fecha: 07/11/2019 \*

\* \*

\*----------------------------------------------\*/

#include<iostream>

#include <conio.h>

#define AYUDA 59

using namespace std;

class Ayuda{

public:

void mostrarAyuda(){

char tecla;

bool flag;

do{

if(kbhit()){

tecla=getch();

}else{

tecla = 112;

}

switch(tecla){

case AYUDA:

system("Ayuda.chm");

break;

}

}while(true);

}

};

**Ingreso.h**

/\*---------------------------------------------

\* Universidad de las Fuerzas Armadas Espe \*

\* Nombres: Angel Cárdenas, Edison Baéz \*

\* NRC: 2967 \*

\* Fecha: 07/11/2019 \*

\* \*

\*----------------------------------------------\*/

#include <iostream>

#include "Validacion.h"

using namespace std;

class Ingreso {

public:

string leer(string,int);

};

string Ingreso::leer(string mensage,int tipo) {

Validacion validacion;

string entrada;

cout << mensage << endl;

cin >> entrada;

while (validacion.validar(entrada, tipo)) {

cout << "Valor no valido reingrese" << endl;

cin >> entrada;

}

return entrada;

}

**Validacion.h**

/\*---------------------------------------------

\* Universidad de las Fuerzas Armadas Espe \*

\* Nombres: Angel Cárdenas, Edison Baéz \*

\* NRC: 2967 \*

\* Fecha: 07/11/2019 \*

\* \*

\*----------------------------------------------\*/

#include <iostream>

#include <ctype.h>

#include <string.h>

using namespace std;

class Validacion {

public:

bool validar(string, int);

};

/\*\*

@param tipo 1 para enteros 2 para flotantes

\*/

bool Validacion::validar(string entrada, int tipo) {

int contador = 0;

try {

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

if (isalpha(entrada[i])) {

throw 1;

}

if (!isdigit(entrada[i]) && tipo == 1) {

throw 1;

}

if (entrada[i] == '.') {

contador++;

}

if ((isdigit(entrada[i]) == 0 && entrada[i] != '.' ) || (contador>1)) {

throw 1;

}

}

}

catch (int e) {

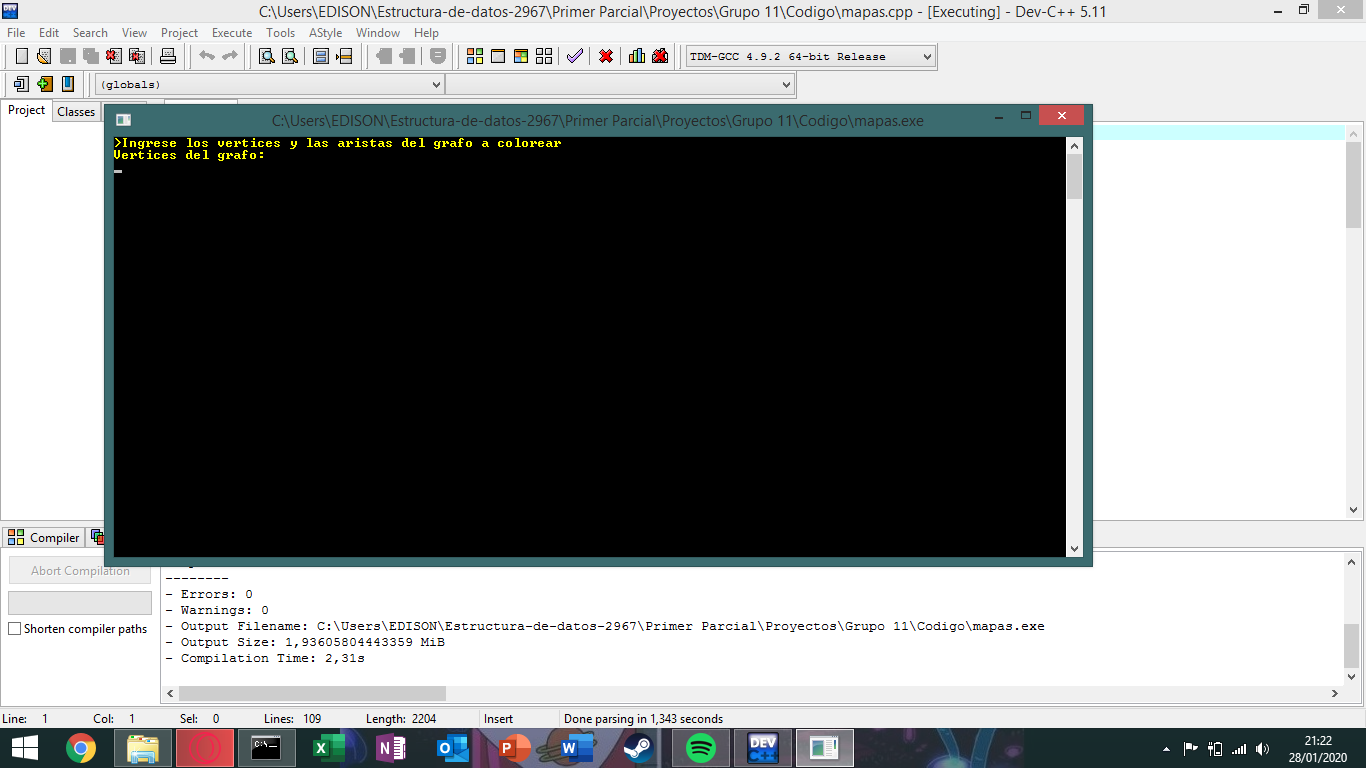
return true;

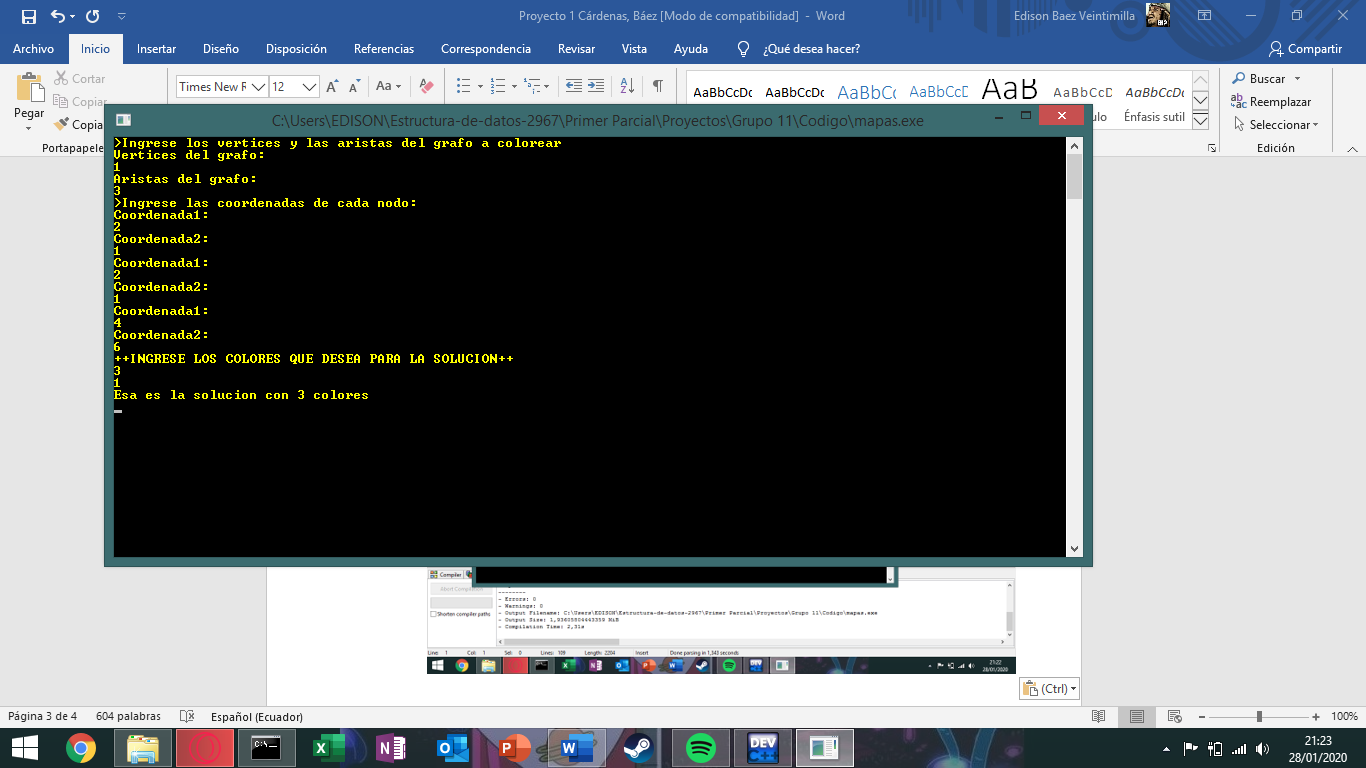
}

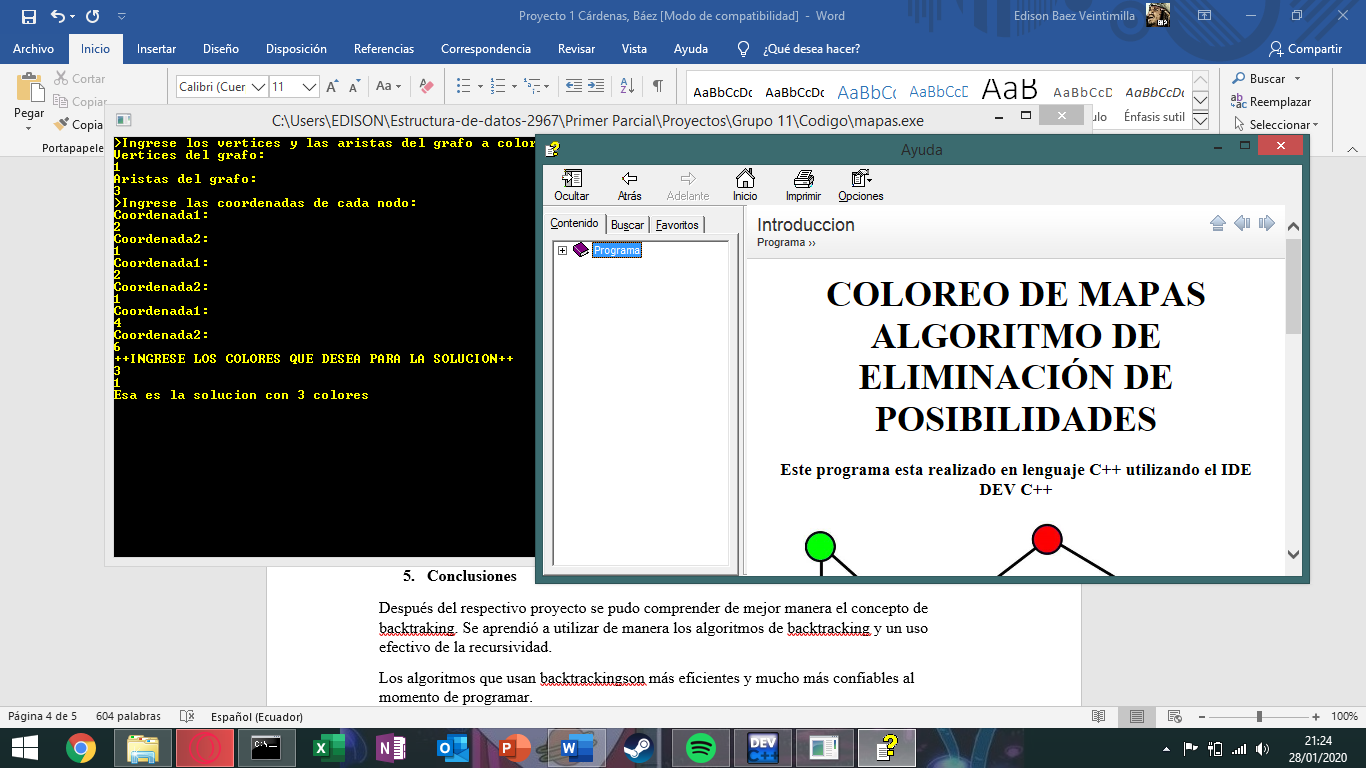
return false;

}

1. **Ejecución**







1. **Conclusiones**

Después del respectivo proyecto se pudo comprender de mejor manera el concepto de backtraking. Se aprendió a utilizar de manera los algoritmos de backtracking y un uso efectivo de la recursividad.

Los algoritmos que usan backtrackingson más eficientes y mucho más confiables al momento de programar.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DOCENTE RESPONSABLE COORDINADOR DE ÁREA

Ing. Fernando Solis. MsC. PhD. Rodrigo Fonseca.