|  |  |  |
| --- | --- | --- |
| **CARRERA:**  Ingeniería de Software | **GUÍA**  No. 01 | **TIEMPO ESTIMADO:**  1 mes |
| **ASIGNATURA:**  Estructura de Datos  NRC: 2967 | **FECHA DE ELABORACION:** 23-10-2020  **SEMESTRE**: Septiembre 2019 – Febrero 2020 | |
| **TÍTULO:**  Proyecto Parcial – Backtracking | **DOCENTE:**  Fernando Solís | |

**OBJETIVO**

Utilizar la técnica conocida como backtracking como la base para realizar un proyecto, con el objeto de demostrar su eficiencia y utilidad.

**INSTRUCCIONES**

1. Utilice como material principal, las principales herramientas para desarrollo en C++
2. Utilice información consultada en Internet y conocimiento adquirido en clase.

**ACTIVIDADES**

1. **Ubicación de recursos**
2. Formar grupos de máximo 2 personas
3. IDE para C++ llamado Visual Studio
4. **Planteamiento del problema**

**Ejercicio**

Desarrollar un programa donde se utilice todo los aprendido en el primer parcial, principalmente el backtracking.

Este proyecto esta basado en un juego llamado “La rata en el laberinto”, donde se generara un laberinto aleatoria, y un algoritmo se encargara de encontrar la salida a dicho laberinto utilizando backtracking.

**Marco Teórico**

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.

1. **Entregable (s)**

**Código**

**Clase main.cpp**

#include <stdio.h>

#include <time.h>

#include <iostream>

#include <ctime>

#include "ResolverTablero.h"

#include "dibujarMatriz.h"

using namespace std;

using namespace std;

enum Movimientos {

MovArriba = 0,

MovAbajo = 1,

MovIzquierda = 2,

};

void Mover(int\*\*, const Movimientos);

void Randomizar(int\*\*);

int dimension = 10;

int main()

{

ResolverTablero comandos;

dibujarMatriz dibujo;

int\*\* TableroRellenado = comandos.GenerarMatriz(dimension);

int\*\* TableroInicial = comandos.GenerarMatriz(dimension);

int\*\* Solucion = comandos.GenerarMatriz(dimension);

int nuevoMovimiento;

comandos.Encerar(TableroInicial, dimension);

cout << "\n" << endl;

srand(time(NULL));

while (\*(\*(TableroInicial + dimension - 1) + dimension - 1) != ' ') {

//"1 = Arriba, 0 = Abajo, 2 = Derecha, 3 = Izquierda"

nuevoMovimiento = rand() % (3);

Movimientos movimiento = (Movimientos)nuevoMovimiento;

Mover(TableroInicial, movimiento);

};

int fila;

int columna;

comandos.LocalizarEspacio(fila, columna, TableroInicial, dimension);

\*(\*(TableroInicial + fila) + columna) = 1;

\*(\*(TableroInicial + 7) + 7) = 1;

TableroRellenado = TableroInicial;

cout << "Tablero Inicial:" << endl;

cout << "\n" << endl;

comandos.RellenarEspacios(TableroRellenado, dimension);

comandos.ImprimirTablero(TableroRellenado, dimension);

cout << "\n" << endl;

cout << "Tablero Solucionado:" << endl;

cout << "\n" << endl;

Solucion = comandos.solucion(TableroRellenado);

comandos.ImprimirTablero(Solucion, dimension);

dibujo.DibujarNuevoTablero(TableroRellenado,Solucion);

free(TableroInicial);

return 0;

}

void Mover(int\*\* Tablero, const Movimientos nuevoMovimiento) {

ResolverTablero comandos2;

int PosicionFila;

int PosicionColumna;

comandos2.LocalizarEspacio(PosicionFila, PosicionColumna, Tablero, dimension);

int MovimientoFila(PosicionFila);

int MovimientoColumna(PosicionColumna);

switch (nuevoMovimiento) {

case MovArriba:

{

MovimientoFila = PosicionFila + 1;

break;

}

case MovAbajo:

{

MovimientoFila = PosicionFila - 1;

break;

}

case MovIzquierda:

{

MovimientoColumna = PosicionColumna + 1;

break;

}

}

// Controlar que no se salga de los limites

if (MovimientoFila >= 0 && MovimientoFila < dimension && MovimientoColumna >= 0 && MovimientoColumna < dimension) {

(\*(\*(Tablero + PosicionFila) + PosicionColumna)) = (\*(\*(Tablero + MovimientoFila) + MovimientoColumna));

(\*(\*(Tablero + PosicionFila) + PosicionColumna)) = 1;

(\*(\*(Tablero + MovimientoFila) + MovimientoColumna)) = ' ';

}

}

///El metodo Randomizar coloca todas los numeros y el espacion en una posicion aleatoria

void Randomizar(int\*\* Tablero) {

srand((unsigned int)time(0));

for (int h = 0; h < 100; ++h) {

int movimiento = (rand() % 4);

switch (movimiento) {

case 0:

{

Mover(Tablero, MovArriba);

break;

}

case 1:

{

Mover(Tablero, MovAbajo);

break;

}

case 2:

{

Mover(Tablero, MovIzquierda);

break;

}

}

}

}

**Clase dibujarMatriz.h**

class dibujarMatriz

{

public:

void DibujarNuevoTablero(int\*\*, int\*\*);

};

**Clase dibujarMatriz.cpp**

#include <SFML/Graphics.hpp>

#include <string>

#include <iostream>

#include "dibujarMatriz.h"

using namespace sf;

using namespace std;

Texture texturaMeta;

Texture texturaMuro;

Texture texturaCamino;

Texture texturaCaminoMalo;

const int dim\_dibujo = 10;

void dibujarMatriz::DibujarNuevoTablero(int\*\* TableroRellenado, int\*\* TableroSolucion) {

RenderWindow window;

window.create(VideoMode(800, 800), "Tablero Resuelto");

window.setKeyRepeatEnabled(false);

if (!texturaMeta.loadFromFile("Imagenes/meta.png")) {

cout << "Error could not load player image" << endl;

}

if (!texturaCamino.loadFromFile("Imagenes/cesped.png")) {

cout << "Error could not load player image" << endl;

}

if (!texturaMuro.loadFromFile("Imagenes/piedra.png")) {

cout << "Error could not load player image" << endl;

}

if (!texturaCaminoMalo.loadFromFile("Imagenes/malo.png")) {

cout << "Error could not load player image" << endl;

}

Sprite Tablero\_Rellenado[dim\_dibujo][dim\_dibujo];

Sprite Tablero\_Solucion[dim\_dibujo][dim\_dibujo];

int h = 0;

int k = 0;

for (int i = 0; i < dim\_dibujo; i++) {

for (int j = 0; j < dim\_dibujo; j++) {

if (\*(\*(TableroRellenado + i) + j) == 0) {

Tablero\_Rellenado[i][j].setTexture(texturaMuro);

}

else

{

Tablero\_Rellenado[i][j].setTexture(texturaCamino);

}

Tablero\_Rellenado[i][j].setPosition(Vector2f(h, k));

h = h + 32;

if (h == 32 \* dim\_dibujo) {

h = 0;

}

}

k = k + 32;

}

int t = 0;

int l = 0;

for (int i = 0; i < dim\_dibujo; i++) {

for (int j = 0; j < dim\_dibujo; j++) {

if ((\*(\*(TableroSolucion + i) + j)) == 0 && (\*(\*(TableroRellenado + i) + j)) == 1) {

Tablero\_Solucion[i][j].setTexture(texturaCaminoMalo);

}

else if ((\*(\*(TableroSolucion + i) + j)) == 0 && (\*(\*(TableroRellenado + i) + j)) == 0) {

Tablero\_Solucion[i][j].setTexture(texturaMuro);

}

else

{

Tablero\_Solucion[i][j].setTexture(texturaCamino);

}

Tablero\_Solucion[i][j].setPosition(Vector2f(t + 350, l));

t = t + 32;

if (t == 32 \* dim\_dibujo) {

t = 0;

}

}

l = l + 32;

}

Tablero\_Rellenado[0][0].setTexture(texturaMeta);

Tablero\_Solucion[0][0].setTexture(texturaMeta);

Tablero\_Rellenado[dim\_dibujo - 1][dim\_dibujo - 1].setTexture(texturaMeta);

Tablero\_Solucion[dim\_dibujo - 1][dim\_dibujo - 1].setTexture(texturaMeta);

while (window.isOpen())

{

for (int i = 0; i < dim\_dibujo; i++) {

for (int j = 0; j < dim\_dibujo; j++) {

window.draw(Tablero\_Rellenado[i][j]);

window.draw(Tablero\_Solucion[i][j]);

}

}

window.display();

system("pause");

window.close();

}

}

**Clase ResolverTablero.h**

class ResolverTablero

{

public:

int\*\* GenerarMatriz(int);

bool backtracking(int\*\*, int, int, int\*\*);

bool isSafe(int\*\*, int, int);

int\*\* solucion(int\*\*);

void Encerar(int\*\*, int);

void ImprimirTablero(int\*\*, int);

void ImprimirText(int\*\*, int);

void RellenarEspacios(int\*\*, int);

void LocalizarEspacio(int&, int&, int\*\*, int);

};

**Clase ResolverTablero.cpp**

#include <iostream>

#include <ctime>

#include <fstream>

#include "ResolverTablero.h"

using namespace std;

int dim = 10;

int n = 1;

fstream enter;

bool ResolverTablero::isSafe(int\*\* Tablero, int x, int y)

{

if (x >= 0 && x < dim && y >= 0 && y < dim && \*(\*(Tablero + x) + y) == 1)

return true;

return false;

}

int\*\* ResolverTablero::solucion(int\*\* Tablero)

{

int\*\* sol = GenerarMatriz(dim);

Encerar(sol, dim);

enter.open("Soluciones/solucion.txt", fstream::out);

enter << "Solucion al laberinto" << endl << endl;

if (backtracking(Tablero, 0, 0, sol) == false) {

printf("Solution doesn't exist");

return false;

}

return sol;

}

bool ResolverTablero::backtracking(int\*\* Tablero, int x, int y, int\*\* sol)

{

if (x == dim - 1 && y == dim - 1) {

\*(\*(sol + x) + y) = 2;

ImprimirText(sol, dim);

enter.close();

return true;

}

if (isSafe(Tablero, x, y) == true) {

\*(\*(sol + x) + y) = 2;

if (backtracking(Tablero, x + 1, y, sol) == true)

return true;

if (backtracking(Tablero, x, y + 1, sol) == true)

return true;

\*(\*(sol + x) + y) = 0;

return false;

}

ImprimirText(sol, dim);

return false;

}

///Encuentra la posicion del espacion en blanco en la matriz

void ResolverTablero::LocalizarEspacio(int& Fila, int& Columna, int\*\* Tablero, int dim) {

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

for (int j = 0; j < dim; j++)

{

if (\*(\*(Tablero + i) + j) == ' ') {

Fila = i;

Columna = j;

break;

}

}

}

}

///Genera la matriz de 4x4

int\*\* ResolverTablero::GenerarMatriz(int dim)

{

int\*\* Tablero, j;

Tablero = (int\*\*)malloc(dim \* sizeof(int\*));

for (j = 0; j < dim; j++)

\* (Tablero + j) = (int\*)malloc(dim \* sizeof(int));

return Tablero;

}

///Todos los valores de la matriz son 0

void ResolverTablero::Encerar(int\*\* Tablero, int dim)

{

for (int i = 0; i < dim; i++)

for (int j = 0; j < dim; j++) {

if (i == 0 && j == 0) {

\*(\*(Tablero + i) + j) = ' ';

}

else {

\*(\*(Tablero + i) + j) = 0;

}

}

return;

}

void ResolverTablero::ImprimirTablero(int\*\* Tablero, int dim)

{

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

for (int j = 0; j < dim; j++)

{

printf("%d", \*(\*(Tablero + i) + j));

printf(" ");

printf(" ");

}

printf("\n");

}

}

void ResolverTablero::RellenarEspacios(int\*\* Tablero, int dim)

{

int i, j;

srand(time(NULL));

for (i = 0; i < dim; i++)

for (j = 0; j < dim; j++) {

if (\*(\*(Tablero + i) + j) == 0) {

\*(\*(Tablero + i) + j) = rand() % 2;

}

}

return;

}

void ResolverTablero::ImprimirText(int\*\* sol, int dim) {

enter << "Intento Numero: " << n << endl;

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

for (int j = 0; j < dim; j++)

{

enter << \*(\*(sol + i) + j) << " " << " ";

}

enter << endl;

}

n++;

}

1. **Conclusiones**

* El backtracking es una herramienta útil para resolver un problema, pero se debe usar en las circunstancias adecuadas.
* Dada que la naturaleza del backtracking es iterativa, se debe considerar la complejidad de un problema. Dado que para problemas de complejidad demasiado elevada el backtracking no es una opción viable por el tiempo y recurso que consumiría.
* La recursividad es esencial para realizar backtracking de manera adecuada.