Universidad Autónoma de Querétaro

Facultad de Informática

PROYECTO FINAL

“LABERINTO”

Estructura de Datos

Ingeniería de software

Grupo 30

Integrantes:

- Raúl Israel Cázares Medrano - 319826
- Macal Palacio Jesús Santiago - 319890
- Javier Osvaldo Tavarez Muñiz -319855
- Ricardo Sandoval Marín - 319861
- Alondra Monserrat Rico Guerrero -319882

Fecha de Entrega: 20 de Noviembre del 2024

INDICE

Introducción

En nuestro proyecto final de Estructura de Datos nos toco abordar el problema del “laberinto” el cual consiste en encontrar la salida de un laberinto, tomando en cuenta que hay puertas de colores que obstruyen el paso por ende primero debemos de buscar las llaves del color correspondiente a la puerta para que así de esta manera podamos abrirla y continuar con nuestra búsqueda de la salida.

Para abordar el caso del laberinto utilizaremos algoritmos de grafos ya que el laberinto lo podemos representar como un grafo en el que cada casilla es un nodo y los posibles movimientos son las aristas que conectan a los nodos. El algoritmo que utilizamos es el algoritmo de Búsqueda de Profundidad (DFS) ya que era el que mejor se adaptaba al proyecto debido a la capacidad que tiene de realizar una exploración exhaustiva de rutas y gestionar las llaves guardadas además de que nos permite recorrer caminos completos hasta llegar a bloqueos (puertas) y al mismo tiempo nos facilita el seguimiento de estados (recolección de llaves) para desbloquear áreas (abrir las puertas y continuar con el camino).

Para este proyecto consideramos 3 llaves y 3 puertas de color azul, rojo y verde, en el cual debe de recolectar la llave azul para poder continuar, posteriormente la roja y por último la verde.

En este informe detallaremos cada una de las etapas del proyecto desde como hicimos el grafo y el modelo del laberinto hasta como lo implementamos el código mostrando las pruebas y su funcionamiento, compartiendo nuestras conclusiones y puntos de vista.

Antecedentes

- **Búsqueda profundidad:**

Un grafo o árbol representado como una lista de adyacencia o matriz de adyacencia. Es una técnica para recorrer o buscar nodos en estructuras de datos como árboles y gráficos. Es uno de los algoritmos fundamentales en teoría de grafos y se utiliza ampliamente en problemas de búsqueda y navegación en grafos.
El algoritmo explora un grafo o árbol comenzando desde un nodo inicial y sigue avanzando en profundidad hasta llegar a un nodo sin vecinos no visitados. Luego, retrocede para explorar otros caminos desde los nodos anteriores, repitiendo este proceso hasta haber visitado todos los nodos conectados.

- **Entrada:** Un nodo de inicio (nodo raíz).
- **Salida:** Un recorrido de los nudos en el orden en que fueron visitados.

Pasos:

1. **Inicialización**:

Marcar el nodo de inicio como visitado.

- Explorar recursivamente cada vecino del nodo actual que no haya sido visitado.

2. **Proceso recursivo**:

Para cada nodo no visitado, realizamos el siguiente proceso:

- Marcar el nudo como visitado.

Para cada vecino de ese nudo:

- Si el vecino no ha sido visitado, haga una llamada recursiva al vecino.

3. **Terminar**:

- Cuando no queden nodos sin visitar, el recorrido o búsqueda habrá terminado.

- **Grafo:**

Un grafo es una colección de nodos o vértices unidos por líneas o aristas. Se usa para representar relaciones binarias entre los elementos. O, en otras palabras, es una representación matemática que se utiliza para modelar relaciones entre objetos.

Los grafos contienen nodos, los cuales se representan como un punto de conexión dentro de la estructura, y las aristas son las conexiones entre estos nodos, las cuales pueden tener o no, un peso, valor o costo entre cada conexión y puede ser ponderado o no ponderado.

- No dirigido: si para cada par de nodos conectados, puedes ir de un nodo al otro en ambas direcciones.
- Dirigido: si para cada par de nodos conectados, solo puedes ir de un nodo a otro en una dirección específica. Usamos flechas en lugar de líneas sencillas para representar arcos dirigidos.

Desarrollo del Proyecto

El desarrollo del proyecto fue un proceso organizado y estructurado que incluyó varias etapas clave, cada una destinada a cumplir con los objetivos específicos del proyecto. A lo largo de este proceso, se llevaron a cabo investigaciones, diseño, implementación y pruebas, asegurando que el sistema cumpliera con los requisitos planteados y que se lograra una solución efectiva y eficiente para la resolución de laberintos con puertas y llaves

Conceptos y teorías:

- **Algoritmo de Profundidad:** Utilizaremos este algoritmo ya que se basa en retroceder cuando no puede avanzar deshaciendo su recorrido hasta un punto anterior para probar otro camino.
- **Teoría de Grafos:** Utilizaremos los principios de la teoría de grafos, representando el laberinto como un grafo, las casillas como nodos y los movimientos como las aristas las cuales conectan a los nodos.
- **Recursos:** Para poder implementar el código primero hicimos el diseño del laberinto y el grafo, donde determinamos la cantidad de llaves, puertas y el posible recorrido que se haría para encontrar la salida.
- **Matriz y Lista de Adyacencia:** Usamos una matriz de adyacencia para ver rápidamente qué celdas están conectadas, y una lista de adyacencia para almacenar los datos.

Planteamiento de la solución:

Para poder encontrar la salida del laberinto recolectando llaves para poder abrir las puertas tomamos como base el código en C++ hecho en clase haciéndole varias modificaciones adaptándolo a nuestro proyecto, además de que usamos un archivo .txt llamado “laberinto1” donde colocamos los datos necesarios para poder resolver la matriz y su lectura mediante el código.

Archivo usado .txt: este archivo tiene la finalidad de ingresar los datos necesarios al programa, leerlos y así poder desarrollar los métodos necesarios para encontrar la solución al problema.

Usamos un laberinto de 10x10 donde las coordenadas son representadas por 0 y 1 utilizando un tipo arreglo empezando desde la posición 0 hasta la posición 9, y los números diferentes a ellos son las llaves y puertas.

	0	1	2	3	4	5	6	7	8	9
0	1	0	1	1	1	1	1	1	1	1
1	1	0	0	4	0	0	0	0	1	1
2	1	0	1	1	1	0	1	0	0	1
3	1	0	1	7	1	1	1	1	0	1
4	1	0	1	0	1	0	0	0	0	1
5	1	0	0	0	1	0	1	1	0	1
6	1	0	1	1	1	5	1	1	8	1
7	1	0	0	1	1	0	0	1	1	1
8	1	1	0	9	1	1	0	0	6	0
9	1	1	1	1	1	1	1	1	1	1

- 4: Puerta Azul 7: Llave Azul
- 5: Puerta Verde 8: Llave Verde
- 6: Puerta Roja 9: Llave Roja

Usamos el guion (-) como formato para separar los datos que serán ingresados.

Siendo 10 el numero de grafos totales indicamos la posición de las llaves y puertas atravez de las coordenadas.

0-1-0
1-1-0
1-2-0
1-3-4
1-4-0
1-5-0
1-6-0
1-7-0
2-1-0
2-5-0
2-7-0
2-8-0
3-1-0
3-3-7
3-8-0
4-1-0
4-3-0
4-5-0
4-6-0
4-7-0
4-8-0
5-1-0
5-2-0

5-3-0
5-5-0
5-8-0
6-1-0
6-5-5
6-8-8
7-1-0
7-2-0
7-5-0
7-6-0
8-2-0
8-3-9
8-6-0
8-7-0
8-8-6
8-9-0

Librerías usadas:

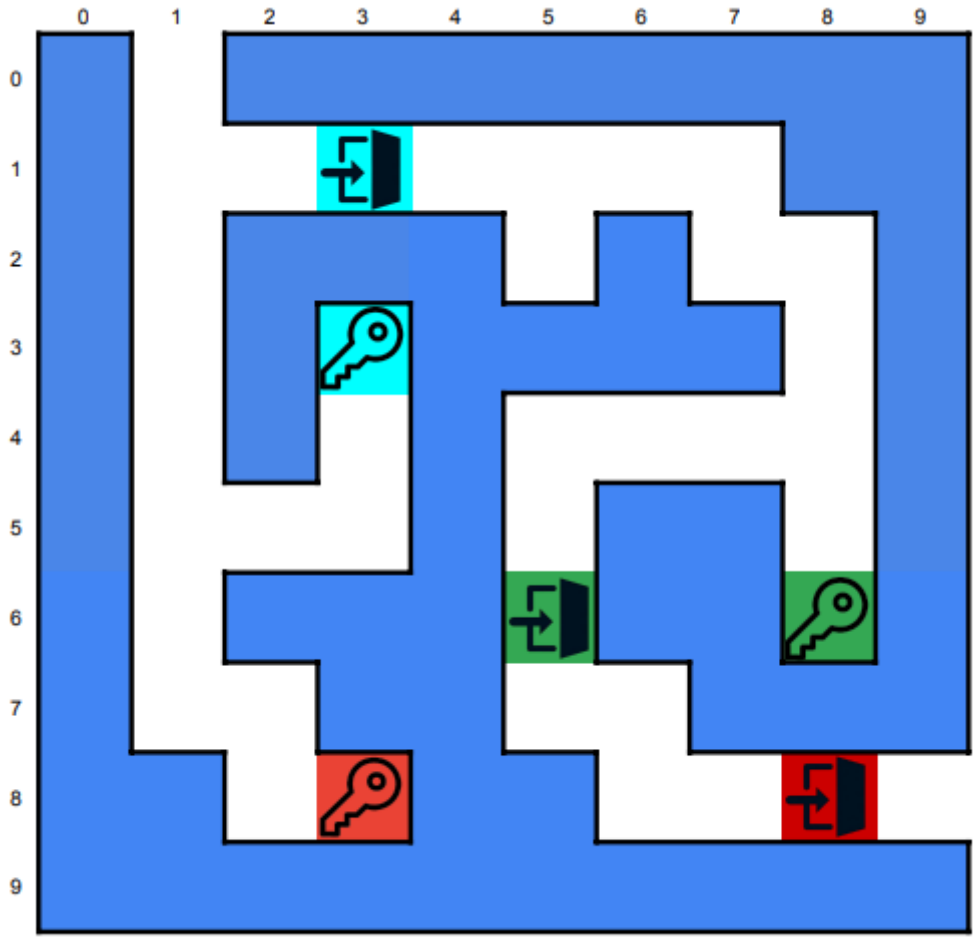
- Fstream: es una librería que sirve para la lectura y escritura de datos
- Cstdlib: nos sirve para definir una colección de funciones y macros para facilitar un código estandarizado, eficiente y de alto rendimiento.

Clases:

- **BusquedaProfundidad:**
  Esta clase nos sirve para para buscar las rutas del laberinto implementando nuestro algoritmo de búsqueda por profundidad. También es donde definimos los métodos para explorar las celdas de manera recursiva y determinamos si existe salida en el camino.

- **Grafo:**
  Esta clase representa un grafo y es la que nos permite leer el archivo de texto y convertirlo a una matriz de adyacencia.

- **Laberinto:**
  Esta clase es la mas importante ya que aquí se hace todo el laberinto basado en el grafo,
  Gestiona las celdas, llaves y puertas de la matriz obtenida del archivo de texto la cual simula el diseño del laberinto. También hace la verificación de las celdas validas y el estado de las puertas (si están cerradas o abiertas).


0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
7
8
9

Evidencias

- **Primera Sesión 16 de octubre**

Se compartieron ideas sobre algoritmos que ayudaran con la elaboración del laberinto y así elegir los más óptimos.

Propuesto por: Macal Palacio Jesús Santiago

1: Algoritmo de Aldous-Broder:
Es un algoritmo de creación de laberintos complejos, pero es algo ineficiente.

Algoritmo: (PSEUDOCÓDIGO)
1: a1,a2
2: repetir
3: matrix(a1,a2)=matrix(a1,a2)+1
4: repetir
5: aux= elige movimiento Norte, Sur, Este u Oeste
6: si Norte y no visitado entonces
7: romper pared Norte
8: moverse a siguiente casilla
9: fin si
10: si Sur y no visitado entonces
11: romper pared Sur
12: moverse a siguiente casilla
13: fin si
14: si Este y no visitado entonces
15: romper pared Este
16: moverse a siguiente casilla
17: fin si
18: si Oeste y no visitado entonces
19: romper pared Oeste
20: moverse a siguiente casilla
21: fin si
22: hasta que hay casillas alrededor sin visitar
23: repetir
24: a1=rnd*nn : a2=rnd*mm
25: hasta que se encuentre una casilla con vecinos sin visitar
26: hasta que haya casillas sin visitar

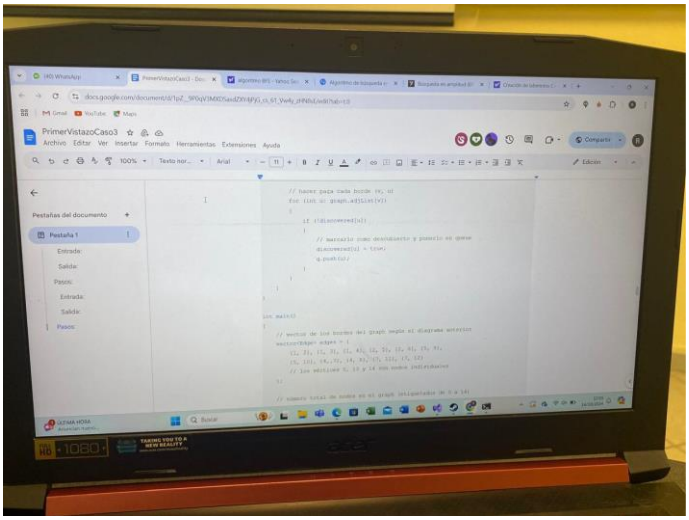
Propuesto por: Javier Osvaldo Tavarez Muñiz

Algoritmo Backtrack:

Backtrack es una técnica de búsqueda sistemática a través de todas las configuraciones posibles dentro del espacio de soluciones. Este método es más complejo que el anterior pero la ventaja de este es que el tiempo de ejecución es menor debido a que resuelve de una manera mucho más eficiente.

Algoritmo 2 Backtrack. (PSEUDOCÓDIGO)

1: a1,a2
2: repetir
3: matrix(a1,a2)=matrix(a1,a2)+1
4: repetir
5: aux= elige movimiento Norte, Sur, Este u Oeste
6: si Norte y no visitado entonces
7: romper pared Norte
8: moverse a siguiente casilla
9: Guardar posición
10: fin si
11: si Sur y no visitado entonces
12: romper pared Sur
13: moverse a siguiente casilla
14: Guardar posición
15: fin si
16: si Este y no visitado entonces
17: romper pared Este
18: moverse a siguiente casilla
19: Guardar posición
20: fin si
21: si Oeste y no visitado entonces
22: romper pared Oeste
23: moverse a siguiente casilla
24: Guardar posición
25: fin si
26: hasta que hay casillas alrededor sin visitar
27: repetir
28: volver una posición
29: si hay casillas sin visitar alrededor entonces
30: comenzar en esta posición
31: fin si
32: hasta que haya posiciones guardadas o posición encontrada
33: hasta que haya posiciones guardadas



Algoritmo de Tremaux

Propuesto por: Ricardo Sandoval Marin

Algoritmo de búsqueda de anchura:
La búsqueda en amplitud (BFS) es un algoritmo para atravesar o buscar estructuras de datos de árboles o graph. Comienza en la raíz del árbol (o algún nodo arbitrario de un graph, a veces denominado "clave de búsqueda") y explora primero los nodos vecinos antes de pasar a los vecinos del siguiente nivel.

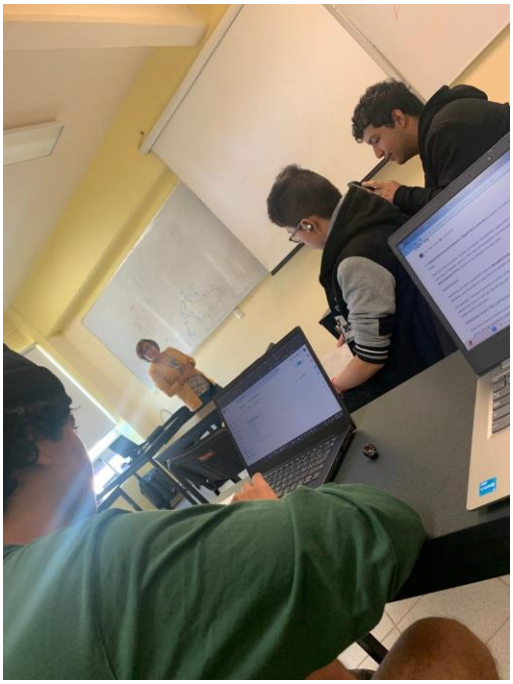
Entrada:
● Un grafo o árbol representado como una lista de adyacencia o matriz de adyacencia.
● Un nodo de inicio (nodo raíz).
Salida:
● Un recorrido de los nudos en el orden en que fueron visitados.
Pasos:
1. **Inicialización** :
○ Crear una cola vacía (cola).
○ Marcar el nodo de inicio como visitado y agregarlo a la cola.
○ Cree una lista o conjunto para rastrear los nodos visitados.
2. **Proceso de búsqueda** :
○ Mientras la cola no esté vacía:
■ Extraer (desescolar) el nudo en la parte frontal de la cola.
■ Para cada vecino de ese nudo que no haya sido visitado:
■ Marcarlo como visitado.
■ Agrega el vecino a la cola.
■ (Opcional) Almacenar el nodo en una lista de recorrido si se desea registrar el orden de visita.
3. **Terminar** :
○ Cuando la cola esté vacía, el recorrido o búsqueda habrá terminado.
Ejemplo de algoritmo:

```
BFS(nodo_inicio, grafo):
Crear una cola vacía (cola)
Crear un conjunto o lista vacío de nodos visitados (visitados)
Agregar nodo_inicio a la cola
Marcar nodo_inicio como visitado
Mientras la cola no esté vacía:
nodo_actual = desencolar la cola
Imprimir nodo_actual (o agregarlo a una lista de recorrido)
Para cada vecino en grafo[nodo_actual]:
Si vecino no está en visitados:
Marcar vecino como visitado
Encolar vecino
Fin
```

Propuesta por: Alondra Monserrat Rico Guerrero.

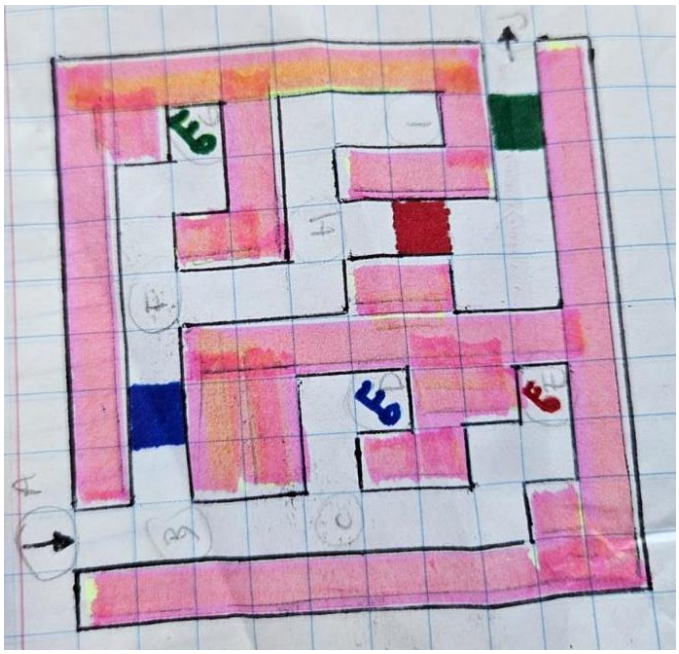
Consiste en:

- Marcamos todo el camino que llevamos y nunca volvemos a pasar por el mismo camino (Sólo para retroceder).
- Al llegar a un cruce tomamos cualquier camino que no hayamos tomado anteriormente.
- Si siguiendo un camino llegamos a un callejón sin salida o a un cruce por el que ya habíamos pasado anteriormente, retrocederemos hasta el cruce anterior y tomaremos un camino diferente.
- No es un algoritmo difícil de memorizar, aunque es necesario poder marcar los caminos que hemos ido tomando. Su principal ventaja respecto al método de la mano es la posible reducción drástica del tiempo de resolución



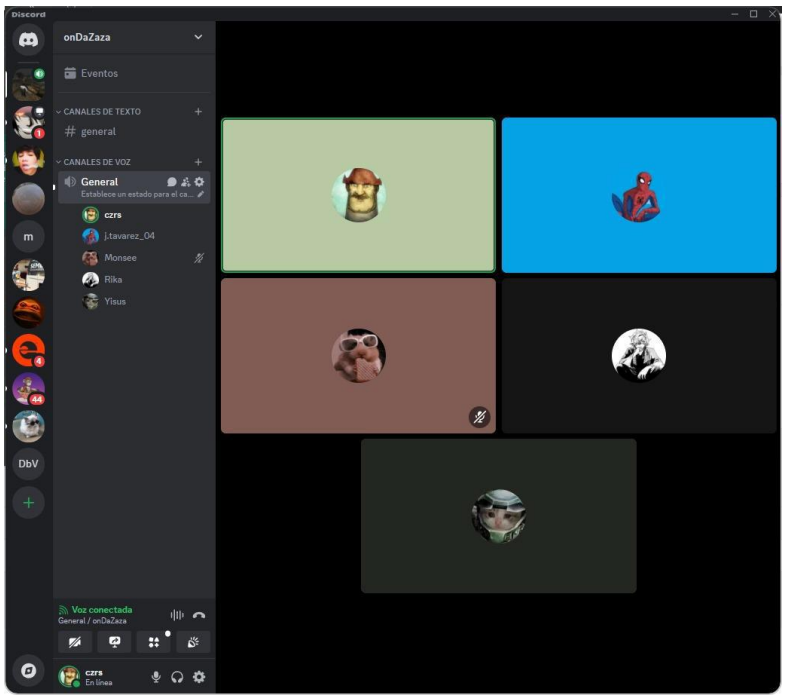
- **Segunda sesión 17 de octubre**

Ser realizó el diseño del laberinto a usar y el grafo para posteriormente llevarlo a código



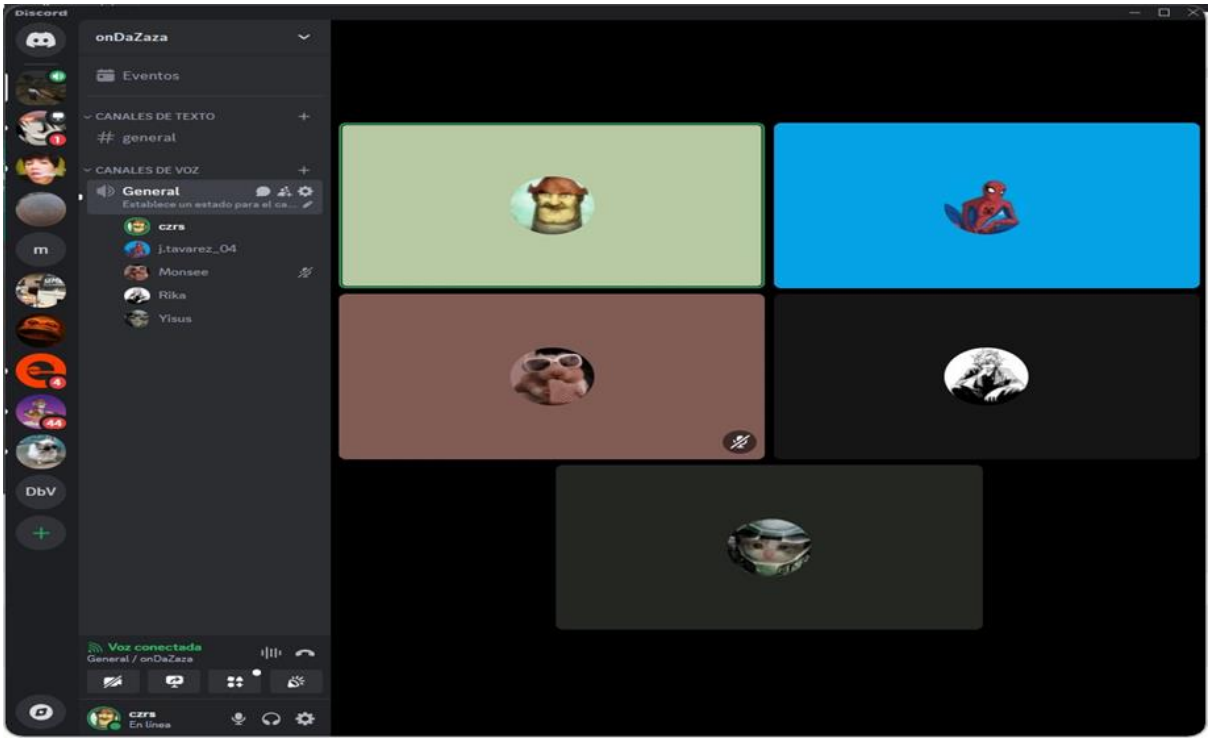
- **Tercera sesión 25 de octubre**

El equipo se reunió para comentar sobre los avances e investigaciones que fueron realizando por separado, al igual que se realizó un avance considerable sobre el algoritmo para poder hacer el laberinto



- **Cuarta Sesión del 27 de Octubre del 2024.**

En la sesión virtual que tuvimos nos pusimos de acuerdo para repartir las tareas que próximamente serán entregadas. Así mismo en el chat grupal aclaramos dudas e hicimos correcciones en algunos trabajos.

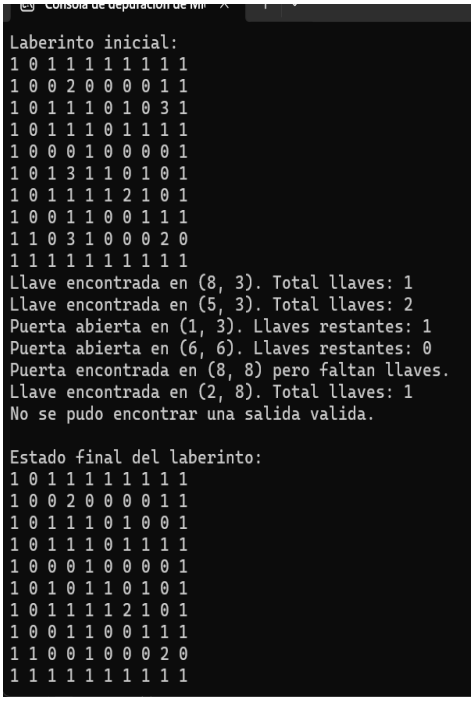


- **Quinta Sesión del 31 de Octubre del 2024**

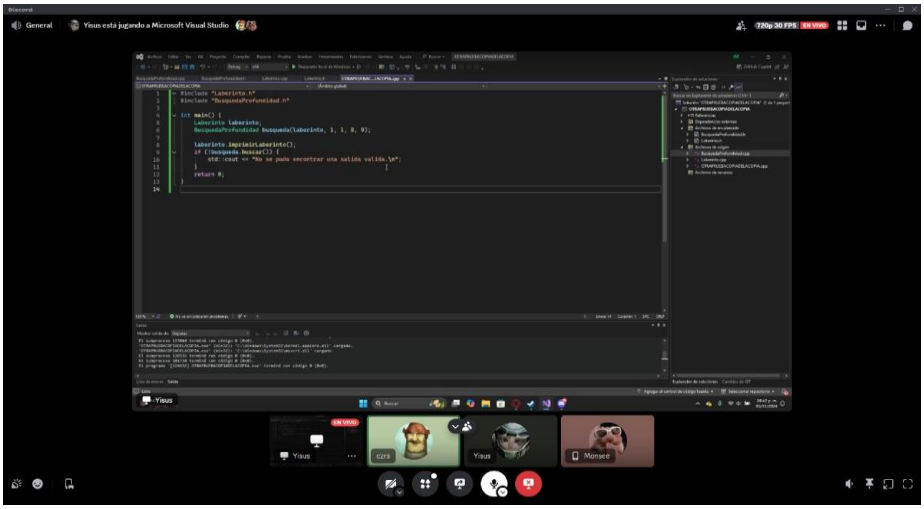
Empezamos a implementar el código haciendo la gestión de las llaves y puertas del laberinto.

- **Sexta Sesión 1 noviembre del 2024**

En esta sesión tratamos de resolver problemas en el código en lo que fueron las clases laberinto y la búsqueda de profundidad pudimos resolver algunos problemas que teníamos anteriormente y así pudimos arreglar la gran mayoría de ellos, pero nos faltan algunos detalles y por último también la interfaz gráfica.



- **Séptima Sesión del 3 de Noviembre del 2024**
  En esta sesión tratamos de resolver dudas y algunos errores sobre el código que aún persisten. Quedando de acuerdo en que le preguntaremos a la maestra en la siguiente clase.



- **Octava Sesión del 4 de Noviembre del 2024**

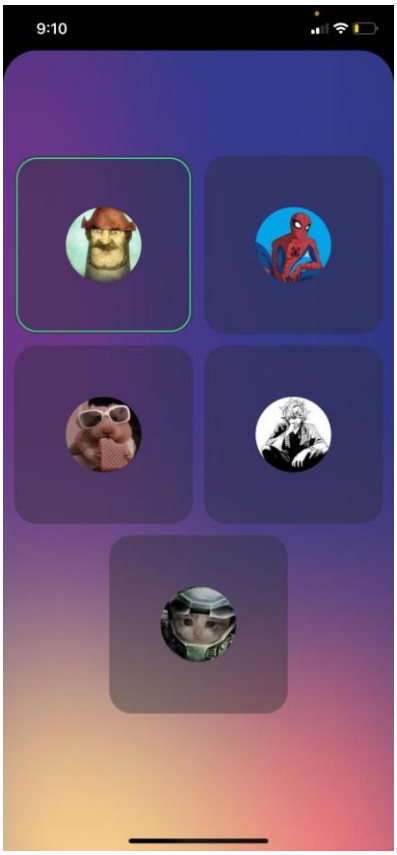
En la clase nos reunimos con la maestra para resolver las dudas que teníamos sobre el proyecto y con base a eso hacer las correcciones correspondientes.

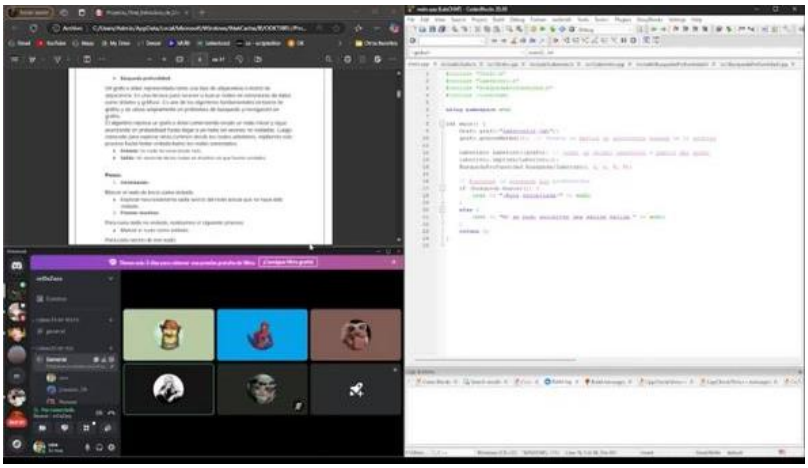
- **Novena sesión 13 de Noviembre del 2024**

En clase nos reunimos con la maestra para que nos ayudara a resolver el problema que teníamos con la lectura del archivo txt.

- **Decima sesión 18 de Noviembre del 2024**

Realizamos una ultima llamada para ajustar los últimos detalles de nuestro proyecto y para grabar el video solicitado.





Algoritmo y Pseudocódigo

Algoritmo:
Para la realización del algoritmo utilizamos fragmentos de código visto en la clase, como lo fue la clase grafo y la búsqueda de profundidad el cual fue nuestro algoritmo seleccionado para la solución de nuestro problema. Este algoritmo nos resultó eficaz al momento de implementarlo en el código ya que se adaptaba perfectamente a la problemática, además de eso la profesora al solicitarle ayuda nos recomendó trabajar con él.

Pseudocódigo:
Archivo .txt para entrada de datos:
Se utiliza el .txt con la clase grafo, los primeros 2 números identifica la posición de X y Y, son las coordenadas. Algunas coordenadas no aparecen en el .txt ya que de modo predeterminado se considera como paredes (se convierten en 1).
El número 1 significa pared y el numero 0 significa camino libre, al igual los diferentes de 0 y 1 son los numeros respectivos de llaves y puertas.
El número 4 es la puerta azul, el 5 es la puerta verde, el 6 es la puerta roja, el 7 es la llave azul, el 8 es la llave verde y por ultimo el numero 9 es la llave roja.

```
10
0-1-0
1-1-0
1-2-0
1-3-4
1-4-0
1-5-0
```

Clases principales
Grafo: Objeto que representa el gráfico cargado desde un archivo de texto (laberinto1.txt).
Laberinto: Objeto que inicializa el laberinto desde la matriz de adyacencia del grafo.
BusquedaProfundidad: Objeto encargado de resolver el laberinto usando búsqueda en profundidad.

- **ClaseGrafo**

Propósito
Representa el laberinto como un gráfico a través de una **matriz de adyacencia** .

Variables:

- **arch**: Archivo de entrada para leer el gráfico.
- **Propósito** : Leer el archivo con información sobre nudos y caminos del laberinto.
- **numVertices**: Número total de nodos (vértices) en el gráfico.
- **Propósito** : Determina el tamaño de la matriz de adyacencia.
- **matAdj**: Matriz de adyacencia dinámica (nodos × nodos).
- **Propósito** : Representa las conexiones entre nodos y sus pesos.

- ***ClaseLaberinto***

Propósito

Implementa la lógica específica del laberinto, incluyendo llaves, puertas y celdas visitadas.

Variables:

- **laberinto**: Matriz (10×10) que representa el estado actual del laberinto.
- **Propósito** : Contiene información sobre celdas libres, puertas y llaves.
- **visitado**: Matriz booleana (10×10).
- **Propósito** : Rastrea qué celdas han sido visitadas durante la exploración.
- **llaves**: Matriz con 3 enteros ([0, 0, 0]).
- **Propósito** : Cuenta las llaves disponibles de cada color (Azul, Verde, Rojo).
- **Puertas (puertaAzulAbierta, etc.)** : Variables booleanas.
- **Propósito** : Indica si las puertas de colores están abiertas.

- ***ClaseBusquedaProfundidad***

Propósito

Implementa el algoritmo de búsqueda en profundidad para resolver el laberinto.

Variables:

- **inicioX,inicioY** : Coordenadas iniciales.
- **Propósito** : Punto de partida para la búsqueda.
- **finX,finY** : Coordenadas finales.
- **Propósito** : Indica la ubicación de la salida.
- **direcciones**: Array con desplazamientos posibles.
- **Propósito** : Representa las direcciones de movimiento (arriba, abajo, izquierda, derecha).

METODOS

- getMatriz(); // Método para devolver la matriz de adyacencia
- getNumVertices(); // Método para obtener el número de vértices
- int laberinto[10][10]; // Tamaño del laberinto
- visitado[10][10]; //Igual se debe mantener el tamaño del laberinto para los visitados, ya que es el mismo tamaño.
- filas = 10; //Para poder identificar las filas.
- columnas = 10; //Para poder identificar las columnas.
- llaves[3] = { 0, 0, 0 }; // Llaves para cada color (Azul, Verde, Rojo) se inicializa con ceros debido a que estos se iran aumentando en la implementación de laberinto.cpp.
- puertaAzulAbierta; //Una variable boleana para poder identificar el estado de las puertas en el laberinto se explicara más en el laberinto.cpp
- puertaVerdeAbierta; //Lo mismo que arriba, pero con la llave verde.
- puertaRojaAbierta; //Se espera que estas variables se inicialicen como false para implementar despues la apertura de puertas.
- void imprimirLaberinto(); //Este metodo va a imprimir el laberinto, que viene siendo la matriz de adyacencia que anterioemente fue creada por el archivo grafo.cpp
- resetVisitado(); //Este metodo lo que hace es reiniciar las celdas que han sido visitadas, util e importante para la implementación del algoritmo de Busqueda por Profundidad
- setVisitado(int fila, int columna, bool valor); //Este metodo marca como visitadas las celdas que se visitan por el laberinto
- getVisitado(int fila, int columna) const; //Este metodo lo que hace es obtener el estado de la celda si esta visitada o no.
- esValido(int fila, int columna); //Este metodo lo que hace es verificar si existe una celda valida para su exploración incluso si estas son llaves o puertas.
- buscarCamino(int x, int y); //Buscar camino implementa parte de la logica para buscar un camino correcto para la resolución del laberinto.
- buscar(); // Método para iniciar la búsqueda.
- BusquedaProf(int x, int y); // Método recursivo para explorar.

Código C++

//Clase Main

```
#include "Grafo.h"
#include "Laberinto.h"
#include "BusquedaProfundidad.h"
#include <iostream>

using namespace std;

int main() {
    Grafo grafo("laberinto1.txt");
    grafo.generaMatAdj();  // Genera la matriz de adyacencia basada en el archivo

    Laberinto laberinto(grafo); // Crear un objeto Laberinto a partir del grafo
    laberinto.imprimirLaberinto();
    BusquedaProfundidad busqueda(laberinto, 1, 1, 8, 9);

    // Ejecutar la búsqueda por profundidad
    if (busqueda.buscar()) {
        cout << "¡Ruta encontrada!" << endl;
    }
    else {
        cout << "No se pudo encontrar una salida valida." << endl;
    }
    return 0;
}
```

//Clase Grafo .h

```
#ifndef GRAFO_H
#define GRAFO_H
#include <fstream>
#include <iostream>
#include <string>

using namespace std;
```

```
class Grafo {
    fstream arch;
    int numVertices;
    int** matAdj;

public:
    Grafo(string nomArch);
    ~Grafo();
    void generaMatAdj();
    void muestraMatAdyacencia();
    int getpos(const string& nombre);
    int** getMatriz(); // Método para devolver la matriz de adyacencia
    int getNumVertices(); // Método para obtener el número de vértices
};

#endif // GRAFO_H
```

Clase Grafo .cpp

```
#include "Grafo.h"
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib> // Para atoi

using namespace std;

/** Es el constructor en el que a partir del txt, lee el archivo y crea la matriz sin
asignar valores aún, solo con el tamaño especificado de los nodos*/
Grafo::Grafo(string nomArch) {
    arch.open(nomArch, ios::in);
    string cad;
    if (arch.fail()) {
        cout << "Error al abrir el archivo..." << endl;
        exit(1);
    }
    getline(arch, cad);
    numVertices = atoi(cad.c_str());  // Cambié stoi() por atoi()
    matAdj = new int* [numVertices];
```

```
    for (int i = 0; i < numVertices; i++) {
        matAdj[i] = new int[numVertices];
        for (int j = 0; j < numVertices; j++) {
            matAdj[i][j] = 1; // Por defecto, todos son caminos (1)
        }
    }
}

/** Destructor de la matriz para liberar memoria */
Grafo::~Grafo() {
    for (int i = 0; i < numVertices; i++) {
        delete[] matAdj[i];
    }
    delete[] matAdj;
}

/** A partir de la creación de nodos hecha por el .txt se interpretan
los carácteres y se transforman a enteros con 'atoi' según la dispocisión
de filas y columnas en la forma "3-5-0"*/
void Grafo::generaMatAdj() {
    string cad, origen, dest;
    int pos1 = 0;
    int pos2 = 0;
    int peso;
    int i, j;
    while (!arch.eof()) {
        getline(arch, cad);
        pos2 = cad.find("-", pos1);
        origen = cad.substr(0, pos2);
        pos1 = pos2;
        pos2 = cad.find("-", pos1 + 1);
        dest = cad.substr(pos1 + 1, pos2 - pos1 - 1);
        peso = atoi(cad.substr(pos2 + 1, cad.length() - 1).c_str());
        pos1 = 0;
        i = atoi(origen.c_str());
        j = atoi(dest.c_str());
        matAdj[i][j] = peso;
    }
}
```

```
/** Imprime la matriz ya ordenada con los caminos libres y las asignaciones que corresponden*/
void Grafo::muestraMatAdyacencia() {
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            cout << matAdj[i][j] << " ";
        }
        cout << endl;
    }
}

/** Método adicional para devolver la matriz generada */
int** Grafo::getMatriz() {
    return matAdj;
}

/** Nuevo método para obtener el número de vértices */
int Grafo::getNumVertices() {
    return numVertices;
}
```

//Clase Laberinto .h

```
#ifndef LABERINTO_H
#define LABERINTO_H

#include <iostream>
#include <fstream>
#include "Grafo.h"

class Laberinto {
private:
    int laberinto[10][10]; // Tamaño del laberinto
    bool visitado[10][10]; //Igual se debe mantener el tamaño del laberinto para los visitados, ya que es el mismo tamaño.
    int filas = 10; //Para poder identificar las filas.
    int columnas = 10; //Para poder identificar las columnas.
    int llaves[3] = { 0, 0, 0 }; // Llaves para cada color (Azul, Verde, Rojo) se inicializa con ceros debido a que estos se iran aumentando en la implementación de laberinto.cpp.
```

```
    bool puertaAzulAbierta; //Una variable boleana para poder identificar el estado de las puertas en el laberinto se explicara más en el laberinto.cpp
    bool puertaVerdeAbierta; //Lo mismo que arriba, pero con la llave verde.
    bool puertaRojaAbierta; //Se espera que estas variables se inicialicen como false para implementar despues la apertura de puertas.


    // Identificadores de llaves y puertas por color
    enum {
        CAMINO = 0,
        AZULPUERTA = 4, VERDEPUERTA = 5, ROJOPUERTA = 6, //Las puertas tendran un identificador difernte de cero, por lo que se busca que tengan numeros diferentes a sus llaves.
        AZULLLAVE = 7, VERDELLAVE = 8, ROJOLLAVE = 9 //Las llaves igualmente tendran un identificador diferente de cero y tambien diferente a las puertas.
    };

public:
    Laberinto(Grafo& grafo); //Constructor que acepta un objeto Grafo
    void imprimirLaberinto(); //Este metodo va a imprimir el laberinto, que viene siendo la matriz de adyacencia que anterioemente fue creada por el archivo grafo.cpp
    void resetVisitado(); //Este metodo lo que hace es reiniciar las celdas que han sido visitadas, util e importante para la implementación del algoritmo de Busqueda por Profundidad
    void setVisitado(int fila, int columna, bool valor); //Este metodo marca como visitadas las celdas que se visitan por el laberinto
    bool getVisitado(int fila, int columna) const; //Este metodo lo que hace es obtener el estado de la celda si esta visitada o no.
    bool esValido(int fila, int columna); //Este metodo lo que hace es verificar si existe una celda valida para su exploración incluso si estas son llaves o puertas.
    void buscarCamino(int x, int y); //Buscar camino implementa parte de la logica para buscar un camino correcto para la resolución del laberinto.
};

#endif // LABERINTO_H
```

Clase Laberinto .cpp

```
#include "Laberinto.h"
#include <iostream>
#include <fstream>

using namespace std;

// Constructor de la clase Laberinto que acepta un objeto de tipo Grafo.
// Este constructor inicializa el laberinto a partir de un grafo dado, donde
// la matriz de adyacencia del grafo se utiliza para representar el estado
Laberinto::Laberinto(Grafo& grafo) {
    filas = columnas = grafo.getNumVertices();

    // Inicialización de las llaves a cero (ninguna llave recogida al principio).
    for (int i = 0; i < 3; ++i) {
        llaves[i] = 0;  // Las llaves de colores azul, verde y rojo comienzan con 0.
    }

    // Obtener la matriz de adyacencia del grafo, que es una representación del
laberinto.
    int** matAdj = grafo.getMatriz();

    for (int i = 0; i < filas; ++i) {
        for (int j = 0; j < columnas; ++j) {
            laberinto[i][j] = matAdj[i][j];
            visitado[i][j] = false;
        }
    }

    // Inicializar la puerta azul como cerrada al principio.
    puertaAzulAbierta = false;
}


// Método para reiniciar el estado de todas las celdas visitadas a 'false'.
// Esto es útil para reiniciar el estado del laberinto antes de realizar una nueva
búsqueda,
// asegurando que todas las celdas estén marcadas como no visitadas.
void Laberinto::resetVisitado() {
    // Recorremos todas las filas del laberinto.
```

```
    for (int i = 0; i < filas; ++i) {
        // Recorremos todas las columnas de cada fila.
        for (int j = 0; j < columnas; ++j) {
            // Establecemos el valor de la celda (i, j) como 'false', indicando que no ha
sido visitada.
            visitado[i][j] = false;
        }
    }
}


// Metodo para marcar una celda como visitada o no visitada.
// Toma las coordenadas (fila, columna) y el valor (true o false).
// Si las coordenadas están dentro de los límites del laberinto, marca la celda como
visitada o no visitada.
// 'true' indica que la celda ha sido visitada, 'false' indica que la celda no ha sido
visitada.
void Laberinto::setVisitado(int fila, int columna, bool valor) {
    // Comprobamos si las coordenadas están dentro de los límites válidos del
laberinto.
    if (fila >= 0 && fila < filas && columna >= 0 && columna < columnas)
        visitado[fila][columna] = valor; // Asignamos el valor a la celda indicada.
}

// Método para obtener el estado de una celda (si ha sido visitada o no).
// Toma las coordenadas (fila, columna) y devuelve 'true' si la celda ha sido
visitada, o 'false' si no lo ha sido.
// Si las coordenadas están fuera de los límites del laberinto, devuelve 'false' como
valor predeterminado.
bool Laberinto::getVisitado(int fila, int columna) const {
    // Comprobamos si las coordenadas están dentro de los límites válidos del
laberinto.
    if (fila >= 0 && fila < filas && columna >= 0 && columna < columnas)
        return visitado[fila][columna]; // Retorna el estado de la celda (visitada o no).

    return false; // Si las coordenadas están fuera de los límites, retornamos 'false'
por defecto.
}
```

```
// Definición de una estructura simple para almacenar posiciones en la lista de
puertas pendientes
struct Posicion {
    int fila;
    int columna;
};

// Lista de puertas pendientes
Posicion puertasPendientes[3];
int cantidadPuertasPendientes = 0;

bool Laberinto::esValido(int fila, int columna) {
    if (fila < 0 || fila >= filas || columna < 0 || columna >= columnas ||
laberinto[fila][columna] == 1 || visitado[fila][columna])
        return false;

    int celda = laberinto[fila][columna];

    // Mensaje de depuración: Mostrar el valor de la celda actual
    std::cout << "Depuracion - Celda en (" << fila << ", " << columna << "): " <<
celda << std::endl;


    // Verificación de puertas y manejo de llaves
    if (celda >= AZULPUERTA && celda <= ROJOPUERTA) {
        int colorIndex = celda - AZULPUERTA; // 0 para azul, 1 para verde, 2 para
rojo
        if (llaves[colorIndex] > 0) { // Si hay llaves disponibles
            llaves[colorIndex]--; // Gastar una llave
            std::cout << "Puerta de color " << (colorIndex == 0 ? "Azul" : (colorIndex ==
1 ? "Verde" : "Rojo"))
                << " abierta en (" << fila << ", " << columna << ").\n";


            // Desbloquear la llave verde si la puerta azul fue abierta
            if (colorIndex == 0) {
                // Notificar que la llave verde ahora está disponible
                std::cout << "Llave Verde ahora desbloqueada tras abrir la puerta
Azul.\n";
            }
```

```
            return true;
        }
        else {
            puertasPendientes[cantidadPuertasPendientes].fila = fila;
            puertasPendientes[cantidadPuertasPendientes].columna = columna;
            cantidadPuertasPendientes++;
            std::cout << "Puerta de color " << (colorIndex == 0 ? "Azul" : (colorIndex ==
1 ? "Verde" : "Rojo"))
                << " encontrada en (" << fila << ", " << columna << ") pero faltan
llaves.\n";

            if (celda == VERDEPUERTA && !llaves[1]) {
                std::cout << "Retrocediendo por la llave verde.\n";
            }
            return false;
        }
    }

    // Restricción adicional: Si es la llave verde, verificar si la puerta azul ya está
abierta
    if (celda == VERDELLAVE && !llaves[0]) { // Si la llave azul no se ha recogido
aún
        std::cout << "Llave Verde bloqueada hasta abrir la puerta Azul.\n";
        return false;
    }

    // Manejo de llaves
    if (celda == AZULLLAVE) {
        llaves[0]++;
        std::cout << "Llave de color Azul (RLL) recogida en (" << fila << ", " <<
columna << ").\n";
    }
    else if (celda == VERDELLAVE) {
        llaves[1]++;
        // Mensaje de depuración: Mostrar que la llave verde ha sido recogida
        std::cout << "Llave de color Verde (VLL) recogida en (" << fila << ", " <<
columna << ").\n";
    }
    else if (celda == ROJOLLAVE) {
```

```
        llaves[2]++;
        std::cout << "Llave de color Rojo (RLL) recogida en (" << fila << ", " <<
columna << ").\n";
    }

    // Intentar abrir puertas pendientes después de recoger una llave
    for (int i = 0; i < cantidadPuertasPendientes; i++)
    {
        int puertaFila = puertasPendientes[i].fila;
        int puertaColumna = puertasPendientes[i].columna;
        int puertaColor = laberinto[puertaFila][puertaColumna] - AZULPUERTA;

        if (llaves[puertaColor] > 0) { // Si hay llave para esta puerta
            llaves[puertaColor]--; // Usar una llave
            std::cout << "Puerta de color " << (puertaColor == 0 ? "Azul" : (puertaColor
== 1 ? "Verde" : "Rojo"))
                << " abierta en (" << puertaFila << ", " << puertaColumna << ").\n";
            for (int j = i; j < cantidadPuertasPendientes - 1; j++) {
                puertasPendientes[j] = puertasPendientes[j + 1];
            }
            cantidadPuertasPendientes--;
            i--;
        }
    }

    // Verificación de la salida
    if (fila == 8 && columna == 9) {
        std::cout << "¡Salida encontrada satisfactoriamente en (" << fila << ", " <<
columna << ")!\n";
        return true;
    }

    return true;
}


void Laberinto::buscarCamino(int x, int y) { // Verifica si la celda actual contiene
una llave (azul, verde o roja)
//Las llaves están representadas por valores específicos en el laberinto.h
explicamos que indices se utilizan
```

```
    if (laberinto[x][y] == AZULLLAVE || laberinto[x][y] == VERDELLAVE ||
laberinto[x][y] == ROJOLLAVE) {// Calcula el índice de la llave (0 para azul, 1 para
verde, 2 para roja) restando el valor de la llave
        int colorIndex = laberinto[x][y] - AZULLLAVE;
        // Incrementa la cantidad de llaves recogidas de ese color
        llaves[colorIndex]++;
        laberinto[x][y] = 0; //Limpia la celda del laberinto donde estaba la ubicación de
la llave, para indicar que ya ha sido recogida
    }
    if (puertaAzulAbierta && llaves[1] > 0) { //Verifica si la puerta azul está abierta y
si se ha recogido al menos una llave verde
        std::cout << "Intentando pasar por la puerta verde...\n"; //Si ambas
condiciones se cumplieron, muestra un mensaje indicando que se intenta pasar
por la puerta verde
    }
}

// Método que imprime el estado actual del laberinto en la consola.
// Este método recorre la matriz que representa el laberinto y la imprime fila por
fila.
void Laberinto::imprimirLaberinto() {
    // Imprime un encabezado para indicar que se va a mostrar el laberinto.
    std::cout << "Laberinto actual:\n";

    // Recorre las filas del laberinto
    for (int i = 0; i < filas; ++i) {
        // Recorre las columnas de cada fila
        for (int j = 0; j < columnas; ++j) {
            std::cout << laberinto[i][j] << " ";
        }

        std::cout << std::endl;
    }
}
```

//Clase BusquedaProfundidad

```
#ifndef BUSQUEDAPROFUNDIDAD_H
#define BUSQUEDAPROFUNDIDAD_H

#include "Laberinto.h"

class BusquedaProfundidad {
private:
    Laberinto& laberinto;
    int inicioX, inicioY;
    int finX, finY;

public:
    BusquedaProfundidad(Laberinto& lab, int ix, int iy, int fx, int fy);
    bool buscar();                  // Método para iniciar la búsqueda.
    bool BusquedaProf(int x, int y); // Método recursivo para explorar.
};

#endif // BUSQUEDAPROFUNDIDAD_H
```

Clase BusquedaProfundidad .cpp

```
#include "BusquedaProfundidad.h"
#include <iostream>

using namespace std;

/** Este constructor inicializa los atributos de la clase indicando inicio y final */
BusquedaProfundidad::BusquedaProfundidad(Laberinto& lab, int ix, int iy, int fx, int
fy)
    : laberinto(lab), inicioX(ix), inicioY(iy), finX(fx), finY(fy) {}


/** Inicia la exploración desde las coordenadas iniciales */
bool BusquedaProfundidad::buscar() {
    laberinto.resetVisitado(); // Restablece las celdas visitadas.
    if (BusquedaProf(inicioX, inicioY)) { // Inicia la búsqueda desde el punto inicial.
        return true;
    }
    return false;
}
```

```
/** Verifica límites y validez de la celda actual.
    Si está en la salida, termina la búsqueda con éxito.
    Marca la celda como visitada y realiza acciones específicas (buscarCamino).
    Explora las celdas adyacentes en las cuatro direcciones. */
bool BusquedaProfundidad::BusquedaProf(int x, int y) {
    if (x < 0 || x >= 10 || y < 0 || y >= 10) return false; // Verifica que las coordenadas estén dentro de los límites.
    if (!laberinto.esValido(x, y)) return false; // Verifica que la celda sea transitable.

    if (x == finX && y == finY) { // Verifica si se alcanzó la salida.
        cout << "¡Salida encontrada!\n" << endl;
        return true;
    }

    laberinto.setVisitado(x, y, true); // Marca la celda como visitada.
    laberinto.buscarCamino(x, y); // Ejecuta acciones adicionales, como recoger llaves o abrir puertas.

    // Define las direcciones a explorar.
    const int direcciones[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    for (int i = 0; i < 4; ++i) {
        int nx = x + direcciones[i][0]; // Calcula la siguiente celda en x.
        int ny = y + direcciones[i][1]; // Calcula la siguiente celda en y.

        if (!laberinto.getVisitado(nx, ny) && BusquedaProf(nx, ny)) { // Si la celda no está visitada y es válida, continúa.
            return true;
        }
    }
    return false;
}
```

Resultados:

```
Laberinto actual:
1 0 1 1 1 1 1 1 1 1
1 0 0 4 0 0 0 0 1 1
1 0 1 1 1 0 1 0 0 1
1 0 1 7 1 1 1 1 0 1
1 0 1 0 1 0 0 0 0 1
1 0 0 0 1 0 1 1 0 1
1 0 1 1 1 5 1 1 8 1
1 0 0 1 1 0 0 1 1 1
1 1 0 9 1 1 0 0 6 0
1 1 1 1 1 1 1 1 1 1
Depuracion - Celda en (1, 1): 0
Depuracion - Celda en (0, 1): 0
Depuracion - Celda en (2, 1): 0
Depuracion - Celda en (3, 1): 0
Depuracion - Celda en (4, 1): 0
Depuracion - Celda en (5, 1): 0
Depuracion - Celda en (6, 1): 0
Depuracion - Celda en (7, 1): 0
Depuracion - Celda en (7, 2): 0
Depuracion - Celda en (8, 2): 0
Depuracion - Celda en (8, 3): 9
Llave de color Rojo (RLL) recogida en (8, 3).
Depuracion - Celda en (5, 2): 0
Depuracion - Celda en (5, 3): 0
Depuracion - Celda en (4, 3): 0
Depuracion - Celda en (3, 3): 7
Llave de color Azul (RLL) recogida en (3, 3).
Depuracion - Celda en (1, 2): 0
Depuracion - Celda en (1, 3): 4
Puerta de color Azul abierta en (1, 3).
Llave Verde ahora desbloqueada tras abrir la puerta Azul.
Depuracion - Celda en (1, 4): 0
Depuracion - Celda en (1, 5): 0
Depuracion - Celda en (2, 5): 0
Depuracion - Celda en (1, 6): 0
Depuracion - Celda en (1, 7): 0
Depuracion - Celda en (2, 7): 0
Depuracion - Celda en (2, 8): 0
Depuracion - Celda en (3, 8): 0
Depuracion - Celda en (4, 8): 0
```

Conclusiones:

Raúl Israel Cazares Medrano:

El desarrollo de este proyecto fue un proceso lleno de aprendizajes y retos que, aunque inicialmente parecían complicados, terminaron siendo clave para alcanzar un buen resultado. Durante la elaboración tuvimos que enfrentar varias dificultades que nos obligaron a replantear ideas y adaptarnos a las necesidades del proyecto, lo que fortaleció tanto nuestras habilidades técnicas como nuestra capacidad de trabajar en equipo. Significó un desafío al conocimiento de los temas vistos en clase con los que contábamos. Partiendo de bases dadas por la maestra tocó enfrentar algunas dificultades debido a ciertas inconsistencias entre el código base y las primeras ideas que teníamos sobre cómo abordar el proyecto. Un aspecto clave del proceso fue dedicar tiempo a analizar y evaluar diversas estrategias para resolver el problema del laberinto de manera eficiente.

Comenzamos con un diseño inicial estático que a pesar de no alinearse completamente con las expectativas del proyecto final, nos sirvió como un punto de partida para comprender mejor los requerimientos con los que identificamos cambios clave que nos llevaron a soluciones más efectivas. Este esfuerzo colaborativo fue esencial para incorporar nuestras mejoras al código base, incluyendo la creación de un archivo de texto que representara de forma exacta la matriz del laberinto. Esto nos permitió generar el diseño creado para ser implementado y, con una mejor comprensión del objetivo, implementamos el algoritmo de búsqueda en profundidad para resolver el laberinto, superando el reto adicional de las llaves y puertas.

Con los demás hubo un ambiente de apoyo mutuo que fue necesario para superar los retos del proyecto.

En resumen, este proyecto no solo nos permitió encontrar una solución sólida al problema planteado, sino que también fue una excelente oportunidad para fortalecer nuestro entendimiento de la construcción de estas estructuras vistas en clase como lo son los grafos y el recorrido sobre estos en un caso que lo requería.

Jesús Santiago Macal Palacio:

El desarrollo del proyecto fue un proceso largo, debido que teníamos que utilizar el código base que nos dio la maestra para basar todo nuestro proyecto lo cual era una buena ventaja, pero algo complicada de utilizar debido a algunas inconsistencias a las primeras ideas del proyecto a realizar, lo que hicimos una buena parte del desarrollo del proyecto fue analizar algunas formas de como podríamos resolver el problema del laberinto de forma eficiente.

Primeramente comenzamos con una idea de laberinto estático lo cual estaba muy errado a lo que la maestra quería como proyecto final, pero nos sirvió como una base solida de que es lo que teníamos que hacer, junto con mis compañeros los cuales proporcionaban ideas de como solucionar estos problemas llegamos a conclusiones que nos llevarían a hacer ajustes en la forma de resolver el laberinto. Es así que una vez que teníamos un buen avance nos dedicamos a aplicar todo este avance a la base del proyecto, los archivos Grafo, creamos nuestro txt para que refleje la matriz que íbamos a utilizar para que nuestro laberinto sea perfectamente reflejado al que teníamos en mente. He de decir que sin algunos de mis compañeros que me proporcionaron la información que necesitaba o ideas que fueron de gran importancia me hubiera costado mucho pensar alguna otra solución para el laberinto, es así que una vez realizado todo este ajuste pudimos llegar a una solución solida a la que teníamos en mente. Aplicando todos los métodos que teníamos pensado utilizar.
Durante este trabajo en equipo trabaje con personas que no conocía bien y llego a la conclusión de que son compañeros bastante buenos en lo que hacen y dispuestos a ayudar en caso de que lo necesite. Así como también poder hacer lo mismo en caso de que lo requieran. El proyecto del laberinto fue una buena forma de trabajar en equipo y dividir el trabajo en forma equitativa para que exista un balance entre que debe hacer cada persona. Así como también un apoyo general en caso de que algún miembro se quede atrás.

Alondra Monserrat Rico Guerrero:

Trabajar en este proyecto del laberinto con llaves y puertas fue una experiencia enriquecedora tanto a nivel técnico como personal. Aprendí a colaborar en equipo y adaptarme a un ritmo diferente trabajo combinando ideas para resolver un problema complejo. Uno de los mayores retos fue implementar el algoritmo de búsqueda en profundidad (DFS) y adaptarlo a las necesidades específicas del laberinto, como manejar las puertas bloqueadas y la recolección de llaves.

Gracias a este proyecto pude comprender la importancia de los algoritmos y que a partir de ellos podemos crear programas muy complejos de una forma en la que nos facilitan resolverlo. Con este proyecto pude aplicar los conocimientos adquiridos durante el semestre y me permitió reforzar más el tema de las clases y como es que funcionan utilizando poo.
También me ayudó a desarrollar habilidades como la organización, la lógica algorítmica y la capacidad de modificar y personalizar un algoritmo base para que se ajuste a un contexto particular.

En general, este proyecto me demostró el uso eficiente de estructuras de datos. También fue un gran ejercicio de trabajo en equipo, adaptándome a diferentes ritmos y perspectivas para lograr un resultado conjunto exitoso.

Javier Osvaldo Tavarez Muñiz:

Participar en este proyecto me permitió mejorar mi organización para adaptarme al trabajo en equipo y a un ritmo diferente. Además, aprendí a abstraer conceptos en diversas áreas como programación, algoritmos y métodos, lo cual fue esencial para implementar el proyecto. Partíamos de un archivo base, pero debíamos modificarlo, agregar atributos, implementar el algoritmo de búsqueda en profundidad y diseñar una forma eficiente de generar y almacenar dichos atributos. Posteriormente, realizábamos cálculos como la obtención de como lograr salir del laberinto.
Esta experiencia también fortaleció mi habilidad para diseñar algoritmos y trasladarlos al ámbito de la programación. Por último, el proyecto contribuyó al desarrollo de mi pensamiento crítico, permitiéndome entender mejor el uso y la lógica de las estructuras de datos, así como su funcionamiento y propósito.

Ricardo Sandoval Marín:

La experiencia de trabajar en este proyecto ha sido increíblemente enriquecedora, tanto a nivel técnico como personal. Resolver un laberinto con condiciones dinámicas, como puertas y llaves, me permitió profundizar en el uso del algoritmo de búsqueda en profundidad (DFS) y comprender cómo adaptarlo para resolver problemas más complejos. Cada etapa del desarrollo presentó desafíos únicos, desde el modelado del laberinto hasta la implementación de las restricciones, lo que me enseñó la importancia de la paciencia, la atención al detalle y el pensamiento lógico.
Además, este proyecto me ayudó a valorar la capacidad de planificar y estructurar soluciones. Fue emocionante ver cómo una idea abstracta, basada en principios teóricos, podía transformarse en un programa funcional capaz de resolver un problema real.

Bibliografía:

https://oa.upm.es/51640/1/PFC_JESUS_M_FERNANDEZ_ORCHANDO.pdf
https://academicos.azc.uam.mx/franz/aed/docs/profundidad.pdf