Solucionador de Laberintos con DFS y BFS

DOCENTE

ANDRES ALEXANDER RODRIGUEZ FONSECA

JUAN PABLO DIAZ RICAURTE - 20222020076
ALAN SANTIAGO AGUDELO SARMIENTO - 20222020170

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD DE INGENIERÍA
PROYECTO CURRICULAR INGENIERÍA DE SISTEMAS
CIENCIAS DE LA COMPUTACIÓN I
GRUPO 020-85 BOGOTÁ D.C.
2025

Índice

1. Introducción

Este es un proyecto de la asignatura Ciencias de la computación I, desarrollado en C++, el cual implementa un solucionador de laberintos visual e interactivo. La aplicación se ejecuta en la terminal (TUI - Text-based User Interface) y utiliza códigos de color ANSI para animar y comparar los algoritmos de Búsqueda en Profundidad (DFS) y Búsqueda en Amplitud (BFS).
El objetivo principal es demostrar visualmente las diferencias fundamentales entre estas dos estrategias de recorrido de grafos: DFS encuentra un camino rápidamente (pero a menudo no el óptimo), mientras que BFS garantiza encontrar el camino más corto.

1.1. ¿Qué son DFS y BFS?

Búsqueda en Profundidad (DFS - Depth-First Search): Es un algoritmo que explora "profundamente.^en el grafo. Sigue un solo camino hasta que llega a un callejón sin salida, y solo entonces retrocede"(backtracking) para probar una ruta alternativa. Utiliza una estructura de Pila (LIFO - Last-In, First-Out). Es excelente para determinar si existe un camino, pero no garantiza que el camino encontrado sea el más corto.

Búsqueda en Amplitud (BFS - Breadth-First Search): Es un algoritmo que explora "por niveles.^o .^en amplitud". Visita primero el nodo inicial, luego a todos sus vecinos directos, luego a todos los vecinos de esos vecinos, y así sucesivamente. Utiliza una estructura de Cola (FIFO - First-In, First-Out). Su propiedad más importante es que garantiza encontrar el camino más corto (en términos de número de pasos) desde el inicio hasta el fin.

2. Características Principales

- **Visualización TUI:** Interfaz de usuario interactiva que se ejecuta completamente en la consola.
- **Animación Paso a Paso:** Renderizado en tiempo real de la exploración de los algoritmos.
  - `\033[46m` (Cian): Celda visitada.
  - `\033[45m` (Magenta): Retroceso (Backtracking) de DFS.
  - `\033[43m` (Amarillo): Camino final encontrado por DFS.
  - `\033[44m` (Azul): Camino final (más corto) encontrado por BFS.
- **Implementación de DFS:** Búsqueda en Profundidad para encontrar el primer camino válido.
- **Implementación de BFS:** Búsqueda en Amplitud para encontrar el camino más corto garantizado.
- **Análisis Comparativo:**
  - Muestra un "mapa de calor"visual que superpone las celdas exploradas por ambos algoritmos.

- Proporciona una tabla de estadísticas que compara nodos visitados, longitud del camino y tiempo de ejecución.

- **Carga de Laberintos:** Permite cargar laberintos personalizados desde archivos `.txt`.
- **Código Modular:** El proyecto está refactorizado en módulos lógicos (Modelo-Vista-Controlador) para mayor claridad y mantenimiento.

3. Estructura del Proyecto

El proyecto está organizado en los siguientes módulos para separar responsabilidades:

Figura 1: Organizacion del proyecto.

4. Conceptos Clave Demostrados

- **Grafos Implícitos:** El laberinto se trata como un grafo donde los nodos son celdas (r, c) y las aristas (vecinos) se calculan al vuelo.
- **Algoritmos de Recorrido:** Implementación de DFS y BFS.
- **Estructuras de Datos:** Uso de Pila (Stack) para DFS y Cola (Queue) para BFS.
- **Hashing:** Uso de `std::unordered_map` y `std::unordered_set` para un seguimiento O(1) de celdas visitadas y "padres".
- **Ingeniería de Software:** Separación de la lógica en Interfaz (`.hpp`) e Implementación (`.cpp`).
- **Programación de Sistemas:** Manipulación de la consola con códigos de escape ANSI y configuración de E/S.

5. Requisitos de Compilación

- Un compilador de C++ que soporte C++17 (`g++`, `clang++`, `MSVC`).
- (Opcional) Visual Studio Code con las extensiones C/C++ y CMake Tools.

6. Instrucciones de Compilación

Para compilar el programa, se asume que todos los archivos fuente (`.cpp` y `.hpp`) se encuentran dentro de una carpeta `src/`.

1. Navega a la carpeta raíz del proyecto en tu terminal.
2. Entra a la carpeta de código fuente:

```
cd src
```

3. Una vez dentro de la carpeta `src/`, ejecuta el comando de compilación:

```
./solucionador
```

7. Instrucciones de Uso

Una vez que el programa `solucionador` ha sido compilado:

1. Se presentará un menú principal en la consola.
2. Se puede elegir un laberinto de prueba (1-5) que carga los archivos de la carpeta `Laberintos/`.

3. **Importante:** Si cargas un laberinto personalizado, la ruta debe ser relativa a la carpeta `src/`. Por ejemplo, para un laberinto en la carpeta raíz `Laberintos/`, deberás escribir: `../Laberintos/maze1_simple.txt`

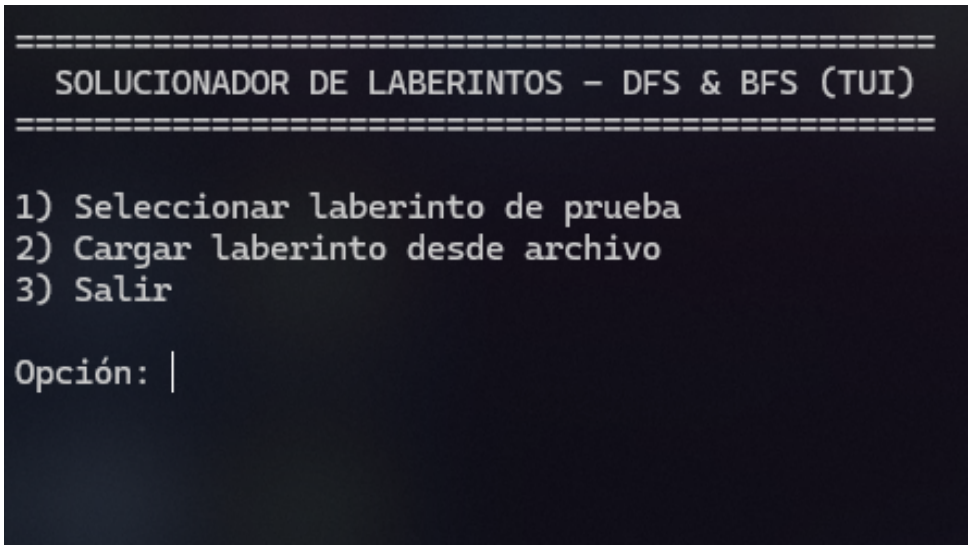
4. El formato de laberinto personalizado debe usar:

   - `#` para paredes.
   - (espacio) para caminos libres.
   - `S` para un (1) único punto de inicio.
   - `E` para un (1) único punto de fin.

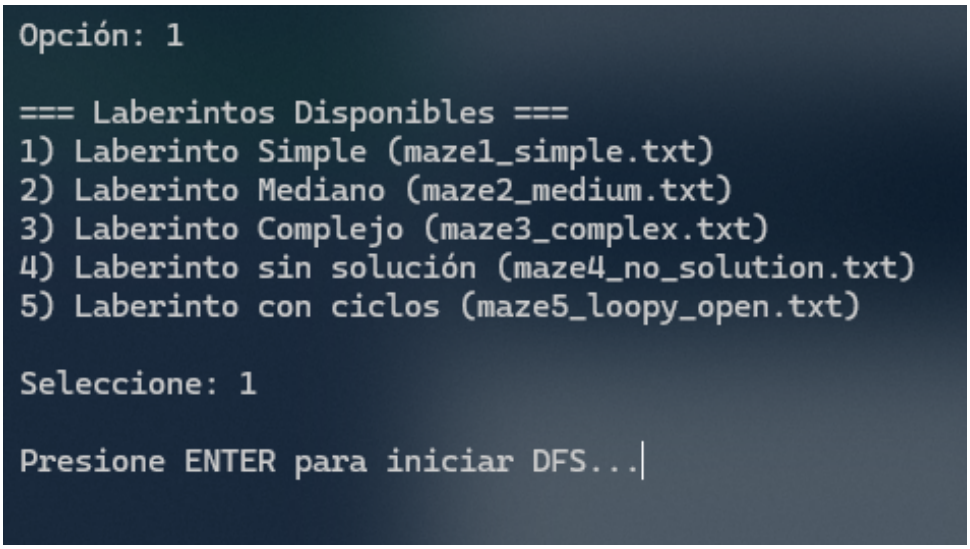
5. Presiona **ENTER** para avanzar por las diferentes etapas: Animación DFS, Resultado DFS, Animación BFS, Resultado BFS y Comparación final.

8. Prueba de funcionamiento

1. Al ejecutar el solucionador, aparecerá en la terminal un menú con tres opciones

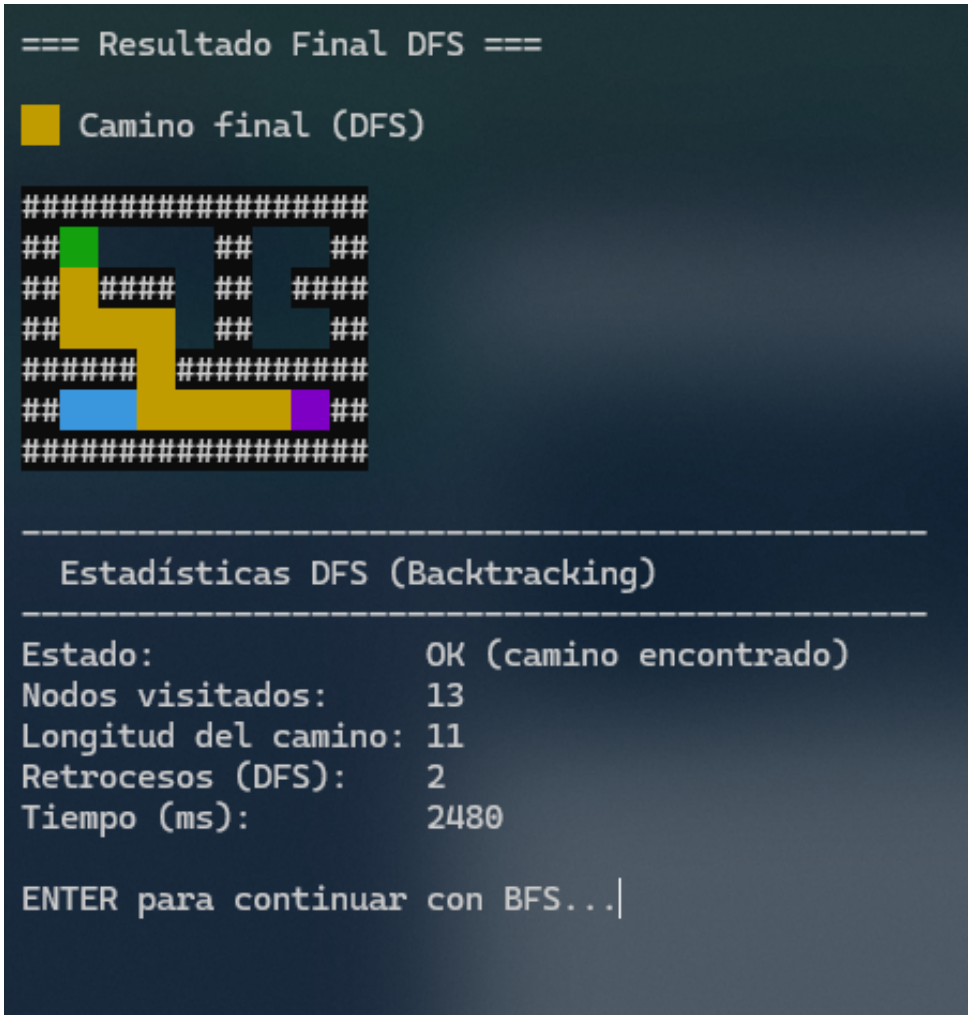


2. Para la prueba se seleccionará la opcion 1, la cual desplegará una lista de laberintos disponibles, estos estarán almacenados en la carpeta "Laberintos". Posteriormente se seleccionará la primera opción "Laberinto simple ($mazel_simple.txt$)z se presionará la tecla ENTER para inciar el recorrido DFS.



3. El programa empezará a recorrer el laberinto mediante el algoritmo DFS, y al finalizar mostrará información importante como el estado (el cual indica si ha encontrado un camino que lleve desde el inicio hasta el fin), los nodos visitados, la longitud del camino encontrado hasta la meta, los retrocesos que se tuvieron que hacer para encontrar el camino y el tiempo total empleado.

Posteriormente se presionará la tecla ENTER para continuar con el algoritmo BFS.



4. Empezará a buscar todas las rutas posibles que lleven al desarrollo del laberinto y señalará la que sea más corta, al finalizar mostrará las estadísticas del algoritmo, el estado, los nodos visitados, la longitud del camino mas corto y el tiempo empleado.

4. Finalmente mostrará la comparación de métricas entre ambos algoritmos de exploración.

8.1. Caso laberinto sin solución

En este caso, dado que el laberinto no tiene como llegar desde el punto de partida hasta la salida, no estará disponible la opción de exploración mediante BFS.