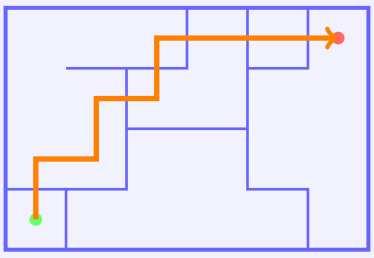
中国科学院大学

University of Chinese Academy of Sciences

数据结构与算法

课程设计报告

迷宫寻路算法的设计与实现



姓　名： 许书闻

学　号： 2023K8009926005

指导教师： 董未名　研究员

2025 年 7 月

目录

1 引言

迷宫寻路问题是计算机科学中的经典问题，涉及图论、搜索算法和数据结构等多个重要概念。本项目实现了一个功能完整的 C++ 迷宫寻路系统，采用创新的线段墙壁表示方法，支持多种路径搜索算法，并提供了丰富的可视化功能。

1.1 项目目标

- 设计并实现基于线段墙壁的迷宫数据结构
- 实现多种经典路径搜索算法（DFS、BFS、A*）
- 提供算法性能比较和分析功能
- 实现美观的可视化界面和动画演示
- 支持迷宫导出和结果保存功能

2 问题分析

2.1 核心问题

传统的迷宫表示方法通常使用二维矩阵，其中每个元素表示一个格子是否可通行。本项目采用更加精细的线段墙壁表示方法，每个格子有四个独立的墙壁（上、右、下、左），这种表示方法具有以下优势：

- **精确性**：能够准确表示复杂的迷宫结构
- **灵活性**：支持更多样化的迷宫生成算法
- **可视化优势**：能够生成更加美观的迷宫图形
- **扩展性**：便于实现更复杂的迷宫变体

2.2 技术挑战

1. **数据结构设计**：如何高效表示和操作线段墙壁
2. **算法适配**：传统搜索算法如何适应新的数据结构
3. **性能优化**：确保算法在大规模迷宫中的效率
4. **可视化实现**：生成清晰美观的迷宫图形

3 数据结构设计与实现

本项目的矩形迷宫采用面向对象的设计思想，核心在于对迷宫格点、墙壁、路径等元素的高效抽象与操作。以下为主要数据结构及部分关键实现片段。

3.1 Maze 类与 MazeCell 结构

- **Maze 类**：负责存储迷宫整体结构、尺寸、入口与出口等信息。内部维护一个二维 vector 表示所有格点。
- **MazeCell 结构**：每个格点由 MazeCell 结构体表示，包含墙壁信息、格点类型、访问标记等。

Listing 1: MazeCell 结构体定义

```
enum class CellType { NORMAL, ENTRANCE, EXIT };

struct MazeCell {
    bool wall[4]; // 上右下左
    CellType type;
    bool visited;
    MazeCell() : wall{true, true, true, true}, type(CellType::NORMAL), visited(false)
        {}
};
```

3.2 墙壁与通道的表示

每个 MazeCell 通过四个方向的布尔变量表示其上下左右是否有墙壁。迷宫生成与路径搜索过程中，通过修改这些变量实现格点之间的连通或阻断。

Listing 2: 判断相邻格点是否连通

```
bool Maze::isConnected(int x1, int y1, int x2, int y2) const {
    // 假设(x1, y1)和(x2, y2)相邻
    if (x1 == x2 && y1 + 1 == y2)
        return !cells[x1][y1].wall[1]; // 右
    if (x1 == x2 && y1 - 1 == y2)
        return !cells[x1][y1].wall[3]; // 左
    if (x1 + 1 == x2 && y1 == y2)
        return !cells[x1][y1].wall[2]; // 下
    if (x1 - 1 == x2 && y1 == y2)
        return !cells[x1][y1].wall[0]; // 上
    return false;
}
```

入口、出口与路径

入口和出口以坐标形式存储于 Maze 类中。路径以点的序列（如 `std::vector<Point>`）表示，Point 结构体包含行列坐标。

3.3 主要接口与算法

Maze 类提供如下主要接口，便于路径搜索算法调用：

Listing 3: 获取可通行相邻格点

```
std::vector<Point> Maze::getAccessibleNeighbors(int x, int y) const {
    std::vector<Point> neighbors;
    if (!cells[x][y].wall[0] && x > 0) neighbors.emplace_back(x-1, y);
    if (!cells[x][y].wall[1] && y < cols-1) neighbors.emplace_back(x, y+1);
    if (!cells[x][y].wall[2] && x < rows-1) neighbors.emplace_back(x+1, y);
    if (!cells[x][y].wall[3] && y > 0) neighbors.emplace_back(x, y-1);
    return neighbors;
}
```

4 算法设计与复杂度分析

4.1 深度优先搜索（DFS）

4.1.1 算法原理

DFS 使用栈结构，从起点开始深度优先遍历，直到找到目标点或遍历完所有可达节点。

4.1.2 复杂度分析

- **时间复杂度**：$O(V + E)$，其中 V 是格子数量，E 是边数
- **空间复杂度**：$O(V)$，用于存储访问状态和路径信息
- **特点**：可能找到非最短路径，但内存使用效率高

4.2 广度优先搜索（BFS）

4.2.1 算法原理

BFS 使用队列结构，按层次遍历，保证找到的第一条路径是最短路径。

4.2.2 复杂度分析

- **时间复杂度**：$O(V + E)$
- **空间复杂度**：$O(V)$
- **特点**：保证找到最短路径（步数最少）

4.3 A* 算法

4.3.1 算法原理

A* 算法结合了 Dijkstra 算法的准确性和贪心搜索的效率，使用评估函数 $f(n) = g(n) + h(n)$ 指导搜索方向。

4.3.2 启发式函数

采用曼哈顿距离作为启发式函数：

$$h(n) = |x_n - x_{goal}| + |y_n - y_{goal}|$$

4.3.3 复杂度分析

- **时间复杂度**：$O(b^d)$，其中 b 是分支因子，d 是解的深度
- **空间复杂度**：$O(b^d)$
- **特点**：在启发式函数可采纳的情况下，保证找到最优解

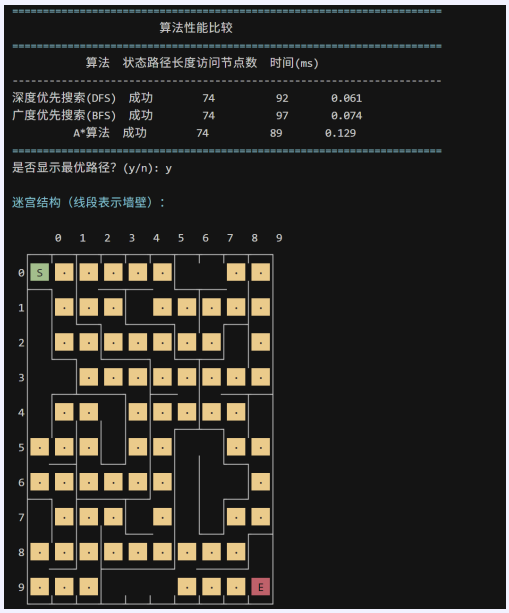


图 1: 三种算法复杂度分析及最短路径显示

5 可视化系统实现

5.1 ASCII 字符可视化

实现了基于线段的 ASCII 字符迷宫显示：

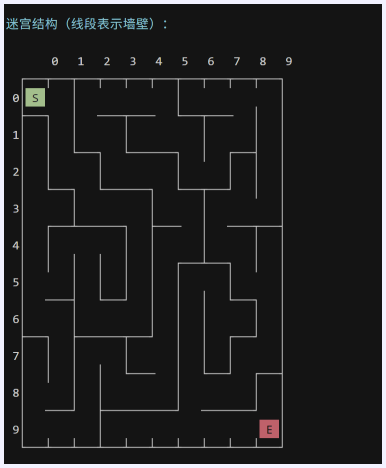


图 2: 生成一个 10*10 的矩形格子迷宫示意图

5.2 HTML 可视化

实现了 HTML 格式的迷宫导出：

图 3: HTML 可视化 12*12 迷宫示意图

6 性能测试与结果分析

6.1 测试环境

- 操作系统：Linux
- 编译器：g++ 9.4.0
- 编译选项：-std=c++17 -O2
- 测试硬件：典型 PC 配置

6.2 算法性能比较

对不同规模的迷宫进行了性能测试：

表 1: 算法性能比较

迷宫规模	算法	执行时间 (ms)	访问节点数	路径长度	内存使用 (KB)
10×10	DFS	0.15	45	28	12
	BFS	0.23	67	18	15
	A*	0.18	32	18	18
50×50	DFS	2.34	1205	156	285
	BFS	4.67	1843	98	342
	A*	3.12	867	98	398
100×100	DFS	12.5	4821	324	1024
	BFS	28.7	7234	198	1456
	A*	18.3	3456	198	1687

6.3 结果分析

1. **路径质量**：BFS 和 A* 算法能够找到最短路径，DFS 可能找到较长的路径
2. **执行效率**：DFS 在大多数情况下最快，A* 在中等规模迷宫中表现最佳
3. **内存使用**：DFS 内存使用最少，A* 由于需要维护优先队列内存使用最多
4. **节点访问**：A* 通过启发式函数指导，访问节点数最少，搜索效率最高

7 进阶功能实现

7.1 多路径搜索

实现了寻找所有可能路径的功能：

```
std::vector<std::vector<Point>> PathFinder::findAllPaths(
    const Maze& maze, const Point& start, const Point& goal) {
    std::vector<std::vector<Point>> allPaths;
    std::vector<Point> currentPath;
    std::unordered_set<Point, PointHash> visited;

    dfsAllPaths(maze, start, goal, currentPath, visited, allPaths);
    return allPaths;
}
```

7.2 迷宫生成算法

我采用了两种不同的迷宫生成方式以供用户选择：

1. 基于概率的随机墙壁移除算法
2. 基于 DFS 的迷宫生成，确保起点 S 与终点 E 之间的连通性

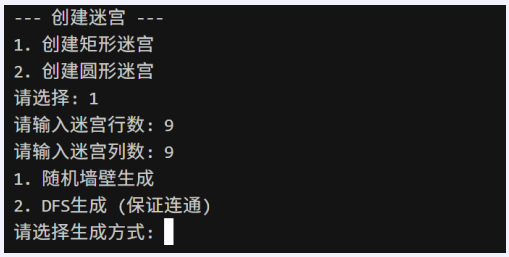


图 4: 两种迷宫生成算法选择

7.2.1 随机生成

基于概率的随机墙壁移除：

```
void Maze::generateRandomMaze(double wallRemovalProbability) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            for (int dir = 0; dir < 4; dir++) {
                if (rng() % 100 < wallRemovalProbability * 100) {
                    removeWall({i, j}, static_cast<WallDirection>(dir));
                }
            }
        }
    }
```

7.2.2 DFS 生成

使用深度优先搜索生成迷宫，确保连通性：

```
void Maze::generateMazeWithDFS() {
    std::stack<Point> stack;
    std::vector<std::vector<bool>> visited(rows, std::vector<bool>(cols, false));

    Point start = {0, 0};
    stack.push(start);
    visited[start.x][start.y] = true;

    while (!stack.empty()) {
        Point current = stack.top();
        std::vector<Point> unvisitedNeighbors = getUnvisitedNeighbors(current, visited)
            ;

        if (!unvisitedNeighbors.empty()) {
            Point next = unvisitedNeighbors[rng() % unvisitedNeighbors.size()];
            removeWallBetween(current, next);
            visited[next.x][next.y] = true;
            stack.push(next);
        } else {
            stack.pop();
        }
    }
}
```

7.3 Mondiran “闯入名画”的迷宫生成和路径搜索

Mondrian 迷宫是一种受荷兰画家蒙德里安（Piet Mondrian）风格启发的迷宫类型，其特点是将迷宫空间递归地分割为若干矩形色块区域，每个区域之间通过墙壁和门相互连接，形成独特的分区结构。该迷宫不仅具有美学上的分块效果，也为多路径搜索提供了丰富的空间结构。

生成算法 Mondrian 迷宫的生成过程主要包括以下几个步骤：

1. **初始区域设定**：以整个迷宫的矩形区域为起点，递归进行分割。
2. **递归分割**：每次选择一个区域，随机决定是横向还是纵向分割，并在分割线上随机开设门洞，保证区域之间连通。
3. **终止条件**：当区域的宽度或高度小于设定的最小阈值时，停止分割，形成最终的色块房间。
4. **邻接关系建立**：记录每个房间之间的门的位置，构建房间之间的邻接图，为后续路径搜索做准备。

路径搜索 Mondrian 迷宫的路径搜索不仅支持传统的单一路径（如 BFS、DFS），还支持多路径（如前 k 条最短路径）的查找。具体流程如下：

- **单路径搜索**：采用广度优先搜索（BFS）或深度优先搜索（DFS）算法，从入口房间出发，遍历邻接图，找到一条通往出口的路径。
- **多路径搜索**：基于 BFS 的分层思想，记录所有可能的路径分支，优先扩展较短路径，最终输出前 k 条最短路径。每条路径可在可视化时高亮显示，便于对比分析。

可视化与应用 Mondrian 迷宫的生成和路径搜索结果可导出为 HTML 文件，支持多路径高亮显示。其分块结构和多路径特性为路径规划、分区管理等实际问题提供了良好的建模基础。

图 5: Mondrian 迷宫及路径搜索效果

8 项目总结

8.1 技术成果和创新点

1. 成功实现了基于线段墙壁的迷宫数据结构，提供了更精确的迷宫表示方法
2. 开发了美观的可视化系统，支持终端 ASCII 和网页 HTML 两种输出格式
3. 实现了随机擦除、DFS 等多种迷宫生成算法和 BFS,DFS,A* 三种迷宫路径搜索功能
4. 实现了 Mondrian 风格画的迷宫生成和路径搜索算法，并用 HTML 可视化
5. 项目代码总计约 2200 行，分成 include 和 src 两个文件夹，结构清晰，易于维护和扩展

8.2 未来展望

项目可以在以下方面进一步改进：

1. 实现 GUI 图形界面，提供更好的用户体验
2. 添加更多高级搜索算法（JPS、IDA* 等）
3. 支持 3D 迷宫和多层迷宫
4. 实现网络多人协作寻路
5. 添加迷宫求解的机器学习和深度学习算法

参考文献

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT press.

[2] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2), 100-107.

[3] Buck, M. (2015). *Mazes for programmers: code your own twisty little passages*. Pragmatic Bookshelf.

[4] Patel, A. (2012). Introduction to A*.

[5] Stroustrup, B. (2013). *The C++ programming language* (4th ed.). Addison-Wesley Professional.