**项目说明文档**

**数据结构课程设计**

**——勇闯迷宫游戏**

### 作 者 姓 名 尹诚成

学 号 2351279

指 导 教 师 张 颖

学 院 专 业 计算机科学与技术学院 软件工程



二〇二四 年 十二 月 十三 日

**目录**

[1.项目分析 1](#_bookmark0)

* 1. .[项目背景分析 1](#_bookmark1)
  2. .[项目需求分析 1](#_bookmark2)
  3. .[项目功能分析 1](#_bookmark3)

1.3.1.迷宫地图生成功能 1

1.3.2.迷宫寻路算法功能 1

1.3.3.界面设计与可视化功能 2

1.3.4.异常处理功能 2

[2.项目设计 2](#_bookmark12)

* 1. .[数据结构设计 2](#_bookmark13)
  2. .[结构体与类设计 2](#_bookmark14)

2.2.1 LinkNode 结构体的设计 2

2.2.2 List 结构体的设计 2

2.2.3 Stack 类的设计 3

2.2.4 Point 结构体的设计 4

2.2.5 Maze 类的设计 4

* 1. .[项目主体架构设计 5](#_bookmark32)

[3.项目功能实现 6](#_bookmark33)

3.1.项目主体架构的实现 6

3.2.迷宫地图生成功能的实现 6

3.3.迷宫寻路算法功能的实现 6

3.4.界面设计与可视化功能的实现 7

4.项目测试 7

4.1.输入迷宫地图大小功能测试 7

4.1.界面设计与可视化功能测试 7

4.2.迷宫寻路算法功能测试 8

1. **项目分析**
   1. 项目背景分析

迷宫游戏是一个经典的问题，被广泛用于计算机科学和算法学习。它模拟了在复杂环境中找到通路的问题，这可以应用于实际生活中的路径规划、导航系统、 游戏开发等领域。迷宫寻路算法可以用于解决各种路径规划和优化问题。这个项目可以用作寻路算法研究的基础。

* 1. 项目需求分析

基于以上背景分析，本项目需要实现需求如下：

(1)迷宫地图生成：迷宫由一个矩形的网格构成，其中包括起点和终点，迷宫中还包括障碍物（墙壁）；

(2)迷宫寻路算法：迷宫有不同的寻路算法，如递归回溯搜索算法、深度优先搜索(DFS)算法、广度优先搜索(BFS)算法和A\*搜索算法；

(3)界面设计与可视化：迷宫地图与路径的展示需要清晰，可视化程度高；

(4)异常处理机制：实现异常处理机制，确保系统稳定性和安全性，避免因用户输入错误导致系统崩溃或信息丢失。

* 1. 项目功能分析

本项目旨在通过迷宫地图生成、迷宫寻路算法、界面设计与可视化和异常处 理等功能，以实现迷宫游戏的核心逻辑。下面对项目的功能进行详细分析。

* + 1. 迷宫地图生成功能

本程序基于随机Prim生成算法生成迷宫地图。随机Prim生成算法生成的迷宫岔路较多，整体上复杂而又较为自然，算法核心思路为：

(1)让迷宫全是墙；

(2)选一个单元格作为迷宫的通路（一般选择起点），然后把它上下左右方向的邻墙放入列表；

(3)当列表里还有墙时，从列表里随机选一个墙，如果这面墙分隔的两个单元格只有一个单元格被访问过，那就从列表里移除这面墙，即把墙打通，让未访问的单元格成为迷宫的通路，并把这个格子的墙加入列表；如果墙两面的单元格都已经被访问过（都打通了），那就从列表里移除这面墙。

* + 1. 迷宫寻路算法功能

本程序使用深度优先搜索（DFS）算法来实现迷宫寻路算法功能。其基本实现思路如下：

(1)深度优先搜索也是一种基于栈的搜索算法，它将一条路径完全探索到底，然后再回溯并尝试其他路径；

(2)从起点开始，选择一个方向前进，一直走到不能再前进为止，然后回退到上一个节点并继续尝试其他方向；

(3)重复这个过程，直到找到终点或者确定没有通路可达终点。

* + 1. 界面设计与可视化功能

迷宫地图与路径的展示需要清晰，可视化程度高。

* + 1. 异常处理功能

实现异常处理机制，处理用户可能输入的非法信息，确保系统的稳定性和安全性。

1. **项目设计**

2.1.数据结构设计

基于项目分析，本项目使用动态分配的int类型二维数组存储迷宫地图，原因如下：

(1)简单高效：使用动态分配的int类型的二维数组来表示迷宫地图是一种简单高效的方式。整数的大小相对较小，占用的内存较少；

(2)可读性和维护性：通过宏定义的方式，表示迷宫地图中的不同状态（空白、墙壁、路径）。这种表示方式容易理解，有利于增加项目的可读性和维护性，并进行必要的修改和调试； 迷宫地图中不同状态的宏定义如下：

#define MAZE\_WALL 0 //墙壁

#define MAZE\_BLANK 1 //空白

#define MAZE\_PATH 2 //路径

(3)利于状态检测：使用整数值来表示不同的状态可以方便地检测迷宫中的当前状态，如检查某个位置是否是墙壁、路径或空白，从而执行相应的操作。

2.2.结构体与类设计

2.2.1.LinkNode结构体的设计

LinkNode 结构体是一个用于构建链表节点的模板结构体。该结构体用于表示链表中的每个节点，其中包括节点存储的数据以及指向下一个节点的指针。本项目希望链表结点类可以直接访问链表结点，所以使用 struct 而不是 class 描述链表结点类。其数据成员和构造函数定义及含义如下：

T data：数据域，存储节点的数据。

LinkNode<T>\* link：指针域，指向下一个节点的指针。

LinkNode(LinkNode\* ptr = NULL)：构造函数，初始化指针域。

LinkNode(const T& item, LinkNode\* ptr = NULL)：构造函数，初始化数据域和指针域。

2.2.2.List类的设计

该通用模板类 MyList 用于表示单链表。此链表以附加的头节点作为起点。链表节点由 MyLinkNode 结构体表示，其中包含数据和指向下一个节点的指针。该链表提供了一系列基本操作函数，包括节点的插入、删除、查找、访问等，以及链表的构造和析构，满足了链表的常见操作需求。其数据成员、构造函数、析构函数、公有成员函数，运算符重载定义及含义如下：

LinkNode\* first：指向链表的第一个节点（头节点）的指针。

LinkNode\* last：指向链表的最后一个节点的指针。

List()：默认构造函数，创建一个空链表。

List(const T& item)：转换构造函数，创建一个只包含一个元素的链表。 List(List& L)：复制构造函数，通过复制另一个链表创建新链表。

~List()：析构函数，释放链表的内存资源，包括所有节点的内存。

void makeEmpty()：清空链表，释放所有结点的内存。

int getLength()const：获取链表中节点的个数。

LinkNode<T>\* getHead()const：获取链表头节点的指针。

LinkNode<T>\* getTail()const：获取链表尾节点的指针。

LinkNode<T>\* search(T item)const：搜索链表中值为 item 的节点，返回指向该节点的指针，若不存在则返回 NULL。

LinkNode<T>\* locate(int i)const：返回链表中第 i 个节点的指针，若 i 超出链表长度或小于 0，则返回 NULL。

int findData(T& item)const：获取链表中值为 item 的节点所在的位置，并通过返回值返回。

bool getData(int i, T& item)：获取链表中第i个节点的数据，并通过引用返回。返回值为操作是否成功。

void setData(int i, T& item)：设置链表中第i个节点的数据。返回值为操作是否成功。

bool insert(int i, T& item)：在链表中第i个节点后插入新节点。返回值为操作是否成功。

bool remove(int i, T& item)：删除链表中第i个节点，并通过引用返回其数据。返回值为操作是否成功。

bool isEmpty()const：检查链表是否为空。

void output()const：输出链表中所有节点的数据。

void addFront(T& item)：在链表的开头插入一个新元素，并通过引用返回其数据。

void removeFront(T& item)：删除链表的第一个元素，并通过引用返回其数据。

List<T>& operator =(List<T>& L)：重载赋值运算符，用于将一个链表赋值给另一个链表。

2.2.3.Stack类的设计

该通用模板类MyStack 用于创建和操作堆栈数据结构。堆栈是一种基本的数据结构，它遵循先进后出的原则，允许用户推入和弹出元素，元素（即链表节点）由MyLinkNode结构体表示，其中包含数据和指针域。此模板类MyStack实现了堆栈的基本功能，并提供了一些辅助方法以方便操作。其数据成员、构造函数、析构函数、公有成员函数，运算符重载定义及含义如下：

LinkNode<T>\* top：指向栈顶元素的指针

Stack() : 默认构造函数，创造一个空的栈。

~Stack() ：析构函数，清空栈并释放内存。

void push(T item)：用于将新元素推入堆栈。它会在堆栈的顶部创建一个新节点，将元素存储在 其中，并将新节点的link指向旧的堆栈顶部。

bool pop(T& item)：从堆栈中弹出顶部元素，将其存储在传入的参数中，并释放相应的节点。如 果堆栈为空，它将返回false，否则返回true。

bool getTop(T& item) const：用于获取堆栈的顶部元素，但不弹出它。如果堆栈为空，它将返回false， 否则返回true。

bool isEmpty() const：判断堆栈是否为空，如果堆栈为空，它将返回true，否则返回false。

int getSize() const：计算堆栈中的元素数量，通过遍历堆栈节点并计数节点的数量来实现。

void makeEmpty()：清空堆栈，释放所有分配的内存。它通过遍历堆栈的节点，释放它们，并将堆栈的top指针置为NULL来实现。

2.2.4.Point结构体的设计

Point结构体是一个用于表示二维坐标的数据结构，用于记录行(x)和列(y)的位置信息、坐标状态(type)信息，方向信息(direction),其数据成员、构造函数定义及含义如下：

int x：行数。

int y：列数。

int type：状态。

int direction：由哪一个方向延申而来。

Point(int a = 0, int b = 0, int t = MAZE\_BLANK, int d = 0) :x(a), y(b), type(t), direction(d) {}：默认构造函数。

2.2.5.Maze类的设计

Maze 类是一个用于生成迷宫并实现多种寻路算法的类。该类的主要目的是提供一个模块化的迷宫生成和寻路解决方案，使用户能够轻松地生成迷宫地图，并进行路径搜索，以及获取找到的路径。其数据成员、构造函数、析构函数、公有成员函数，运算符重载定义及含义如下：

int\*\* mazeMap：用于存储迷宫的二维数组，表示迷宫的结构。

int\*\* mazeMapMark：用于标记迷宫中已经访问的点，避免重复访问。

int row：迷宫的行数。

int col：迷宫的列数。

int startRow：起点的行索引。

int startCol：起点的列索引。

int endRow：终点的行索引。

int endCol：终点的列索引

int currentRow：当前探索的位置的行索引。

int currentCol：当前探索的位置的列索引。

List<Point> blankPointsList：用于存储周围的墙面点，供迷宫生成使用。

Stack<Point> path：用于存储找到的路径。

Point move[4] = { {0,-1,MAZE\_BLANK,LEFT}, {0,1,MAZE\_BLANK,RIGHT}, {-1,0,MAZE\_BLANK,UP}, {1,0,MAZE\_BLANK,DOWN} }：移动方向，包含上、下、左、右的偏移量。

bool isVaildPoint(int x,int y)：检查 (x, y) 点是否在迷宫的有效范围内。

void findNearWalls()：查找当前点周围的墙，更新 blankPointsList。

void createMaze()：生成迷宫的主要函数

Maze(int \_row, int \_col, int \_startRow , int \_startCol, int \_endRow, int \_endCol)：构造函数，用于初始化迷宫的行数、列数、起点和终点。

~Maze()：析构函数，在对象销毁时释放动态分配的内存，避免内存泄漏。

void outputMaze()：输出迷宫的当前状态。

void outputPath()：输出找到的路径信息。

bool recursive()：递归求解迷宫，寻找从起点到终点的路径。

* 1. 项目主体架构设计

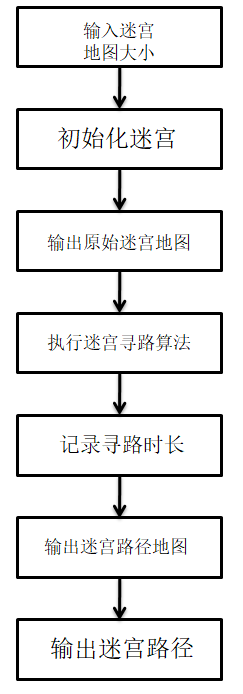


图 2.2 项目主体架构设计流程图

项目主体架构设计：

(1)输入迷宫地图大小：用户被要求在一定范围内输入迷宫的行数和列数；

(2)初始化迷宫对象：使用用户提供的行数、列数，以及迷宫的起始位置和 终点位置，初始化一个迷宫对象；

(3)输出原始迷宫地图：输出初始生成的迷宫地图，显示迷宫中的墙壁和通道； (4)执行迷宫寻路算法

(5)输出迷宫路径地图：输出包含找到的路径的迷宫地图；

(6)输出迷宫路径：如果找到了路径，将路径的坐标输出到屏幕上。如果没有找到路径，则显示未找到路径的提示信息。

1. **项目功能实现**

3.1.项目主体架构的实现

项目主体架构实现思路为：

(1)输入迷宫地图大小：分别调用两次inputMazeSize函数，输入迷宫地图的行数和列数迷宫地图的行数和列数的返回通过宏定义的方式定义，便于修改和维护；

#define MAX\_SIZE 49

#define MIN\_SIZE 7

(2)初始化迷宫对象：创建一个Maze对象，传入迷宫的行数、列数，起始坐标（1, 1），以及终点坐标（row - 2, col - 2）。起始坐标默认为迷宫地图左上角（不算边界），终点坐标为迷宫地图右下角（不算边界）；

(3)调用maze. outputMaze ()函数输出原始迷宫地图；

(4)执行迷宫寻路算法；

(5)如果找到了路径，则调用maze.outputMaze()函数输出迷宫路径地图，将路径的坐标输出到屏幕上，显示从起点到终点的移动路径；如果未找到路径，输出未找到路径的消息。路径坐标信息存储在 Maze 类的私有数据成员自定义栈 path 中，可以通过调用 Maze 类的成员函数 maze.outputPath()获取路径坐标信息。

3.2.迷宫地图生成功能的实现

本程序使用随机Prim生成算法生成迷宫地图的思路为：

(1)在构造函数（Maze::Maze）中，首先初始化迷宫地图（mazeMap）为墙 （MAZE\_WALL），并将起始点（startRow和startCol）设置为迷宫的起点，将目标点（endRow和endCol）设置为迷宫的终点；

(2)使用随机 Prim 算法来生成迷宫路径的关键部分在createMaze函数中。开始时，创建一个空的迷宫点列表（blankPointsList），然后通过findNearWalls函数查找当前位置周围的墙，将这些墙的坐标和方向添加到blankPointsList中；

(3)进入循环，从blankPointsList随机选择一个墙，如果这面墙分隔的两个单元格只有一个单元格被访问过，那就从列表里移除这面墙，即把墙打通（将当 前位置和墙的位置设置为通路 MAZE\_BLANK）。如果墙两面的单元格都已经被访问过（都打通了），那就从列表里移除这面墙；

(4)再次使用findNearWalls函数查找新的周围墙，将它们添加到blankPointsList；

(5)从blankPointsList中移除已经处理的墙，重复上述步骤，直到blankPointsList为空。

3.3.迷宫寻路算法功能的实现

执行递归回溯搜索算法的函数为 Maze 类的成员函数recursive，其实现思路为：

(1)路径搜索：使用recursive函数来尝试从起点出发递归地 搜索路径。在mazeMapMark数组中标记当前位置(currentRow, currentCol)为已访问。 调用recursive函数，该函数会递归探索四个可能的移动方向：上、下、左、右。如果找到了一条从起点到终点的路径，recursive函数会返回true，并在迷宫地图上标记路径上的位置为MAZE\_PATH，同时将这些位置压入path 数据结构中，以便之后获取路径；

(2)返回路径搜索结果：返回 true 表示找到了从起点到终点的路径，否则返回false，表示没有找到路径。

3.4.界面设计与可视化功能的实现

执行界面设计与可视化功能的函数为Maze类的成员函数outputMaze，其实现思路为使用两个嵌套的for循环，遍历整个迷宫的行和列，输出“始”、“终”、“ ”、“■”、“□”分别表示起点、终点、通道、墙壁和路径。

1. **项目测试**

4.1.输入迷宫地图大小功能测试

分别输入超过上下限的整数、偶数、浮点数、字符、字符串，可以验证程序对输入非法的情况进行了处理。 当输入合法时，程序继续运行。

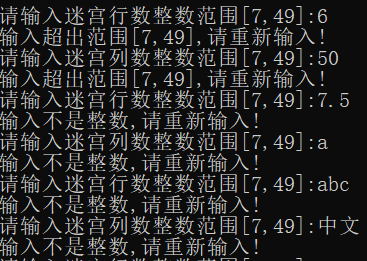


图 4.1 输入迷宫地图大小功能测试

4.2.界面设计与可视化功能测试

Maze 类的成员函数outputMaze使用两个嵌套的for循环，遍历整个迷宫的行和列，输出“始”、“终”、“ ”、“■”、 “□”分别表示起点、终点、通道、墙壁和路径。

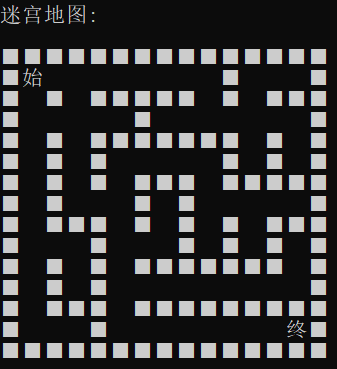


图 4.2 界面设计与可视化功能测试

4.3.迷宫寻路算法功能测试

当深度优先搜索(DFS)算法找到一条路径时，输出迷宫路径地图并打印迷宫路径。

当深度优先搜索(DFS)算法未找到路径时，输出提示信息“未找到迷宫路径”。

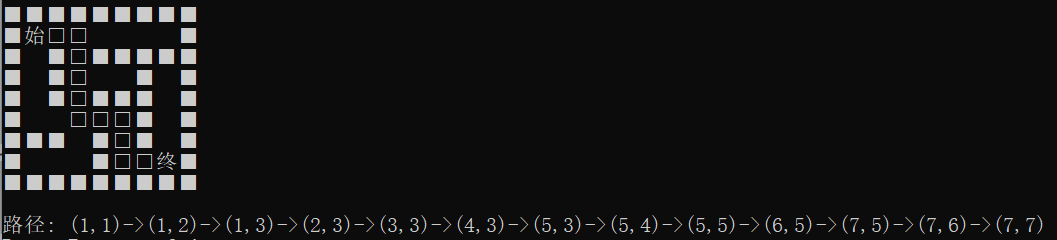


图 4.3 迷宫寻路算法功能测试（找到路径）

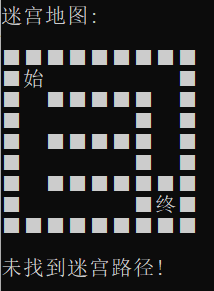


图 4.4 迷宫寻路算法功能测试（未找到路径）