项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 罗佳瑞

学 号： 1952528

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 分析 1](#_Toc6460)

[1.1 背景分析 1](#_Toc9966)

[1.2 功能分析 1](#_Toc940)

[2 设计 2](#_Toc16612)

[2.1 数据结构设计 2](#_Toc8213)

[2.2 类结构设计 2](#_Toc19049)

[2.3 成员与操作设计 2](#_Toc3689)

[2.4 系统设计 3](#_Toc26982)

[3 实现 4](#_Toc21410)

[3.1 创建功能的实现 4](#_Toc4799)

[3.1.1 创建功能流程图 4](#_Toc19879)

[3.1.2 插入功能核心代码 4](#_Toc7973)

[3.2 寻找路径功能的实现 6](#_Toc9018)

[3.2.1 寻找路径功能流程图 6](#_Toc28610)

[3.2.2 寻找路径功能核心代码 7](#_Toc30854)

[3.3 打印功能的实现 8](#_Toc26899)

[3.3.1 打印功能流程图 8](#_Toc13584)

[3.3.2 打印功能核心代码 8](#_Toc32160)

[3.4 总体系统的实现 10](#_Toc19975)

[3.4.1 总体系统流程图 10](#_Toc3964)

[3.4.2 总体系统截屏示例 11](#_Toc24655)

[4 测试 12](#_Toc26285)

[4.1 功能测试 12](#_Toc32704)

[4.1.1带边界横纵等边的迷宫 12](#_Toc11244)

[4.1.2带边界横纵不等边的迷宫 13](#_Toc3433)

[4.2 边界测试 14](#_Toc4707)

[4.2.1带边界无路径的迷宫 14](#_Toc13525)

[4.2.1无边界的迷宫 15](#_Toc28750)

# 1 分析

## 1.1 背景分析

迷宫指的是充满复杂通道，很难找到从其内部到达入口或从入口到达中心的道路，道路复杂难辨，人进去不容易出来的建筑物。人类建造迷宫已有5000年的历史。在世界的不同文化发展时期，这些奇特的建筑物始终吸引人们沿着弯弯曲曲、困难重重的小路吃力地行走，寻找真相。

由狭义迷宫第一定律可知，只要走遍迷宫所有路径必然可以找到迷宫终点；另外，由于死路或误差的存在，任何一条路线可能被走最多两次，因此所有迷宫的探索都可以靠记忆力完成，比起人脑记忆，而计算机在这一领域可以大显身手——考虑编程用回溯法快速解决迷宫问题。

## 1.2 功能分析

作为一个最简易的迷宫游戏，首先应该有的功能就是用特定的符号建立迷宫并且可以予以显示。其次，还应该能够输入设置起点、终点，并根据选择的起点和终点求解出路径。最后，该游戏还应该确保软件可以正常打开与关闭。

综上所述，一个迷宫游戏至少应该具有输入、求解、输出的功能。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该游戏需要存储迷宫信息，因此考虑使用二维数组的方式进行存储地图。同时，由于用回溯法探索迷宫，选择采用栈的数据结构对探索路径进行存储。而栈中的每个元素结点采用类的数据结构，将横坐标与纵坐标同时存储，以便路径的探索与输出。

## 2.2 类结构设计

栈的结点采取类（Point）的结构，而两个类之间的耦合关系本游戏采用公有函数返回的方式进行处理，使得结点私有成员坐标能够进行安全读取。

栈类（Stack）应含有成员：顶部指针（t）、储存空间（elements），应含有基本操作：判空（empty）、判满（full）、返回大小（size）、入栈（push）、出栈（pop）、取栈顶元素（top）等。

## 2.3 成员与操作设计

**结点类（Point）**

**私有成员：**

int x0;//表示结点横坐标

int y0;//表示结点纵坐标

**公有操作：**

Point::Point() ;//默认构造函数

Point::Point(int a, int b);//构造函数

int Point::x()//调取横坐标的值

int Point::y()//调取纵坐标的值

**栈类（Stack）**

**私有成员：**

int t;//顶部指针

T\* elements;//储存空间

**公有操作：**

bool Stack::empty() ;//判断是否为空

bool Stack::full() ;//判断是否已满

int Stack::size() ;//返回队列当前大小

void Stack::push(T element) ;//入栈

void Stack::pop();//出栈

T Stack::top() ;//取栈顶元素

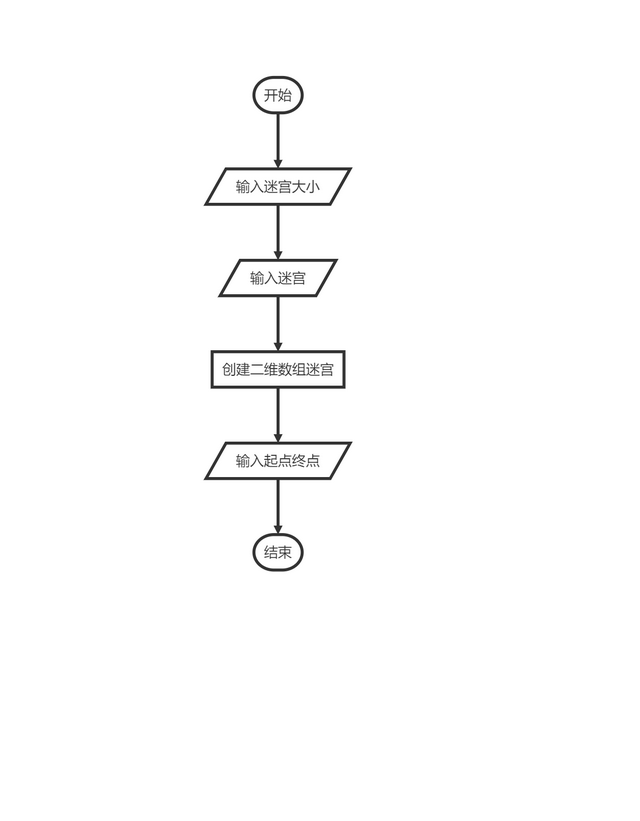
## 2.4 系统设计

系统首先提示用户输入迷宫信息——迷宫大小、地图、起点与终点坐标，完成对二维数组迷宫maze的创建和输入数据工作，然后利用路径栈Point根据回溯法向四个方向不断依次探索，直到完成路径的探索，打印出地图与最终选择的路径。

# 3 实现

## 3.1 创建功能的实现

### 3.1.1 创建功能流程图



### 3.1.2 插入功能核心代码

//创建迷宫

int x\_size, y\_size;

cout << "请输入迷宫大小（横坐标、纵坐标）：";

cin >> x\_size >> y\_size;

int\*\* maze = new int\* [y\_size];//迷宫二维数组

for (int i = 0; i < y\_size; i++)

{

maze[i] = new int[x\_size];

}

for (int i = 0; i < y\_size; i++)

for(int j=0;j<x\_size;j++)

{

maze[i][j]= 1;

}

cout << "请输入迷宫："<<endl;

for (int i = 0; i < x\_size; i++)

for (int j = 0; j < y\_size; j++)

cin >> maze[i][j];

//输入起点终点

int start\_x, start\_y;

int end\_x, end\_y;

cout << "请输入起点坐标（横坐标、纵坐标）：";

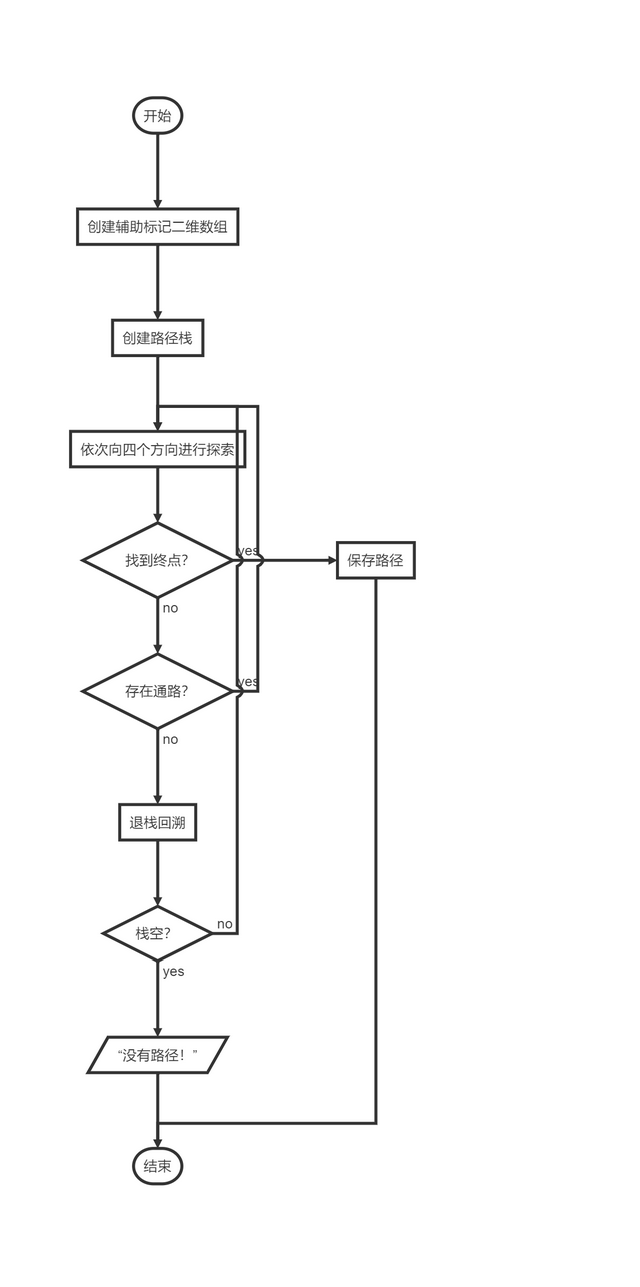
cin >> start\_x >> start\_y;

cout << "请输入终点坐标（横坐标、纵坐标）：";

cin >> end\_x >> end\_y;

## 3.2 寻找路径功能的实现

### 3.2.1 寻找路径功能流程图



### 3.2.2 寻找路径功能核心代码

//寻找路径

int next[4][2] = { {-1,0} ,{1,0},{0,-1}, {0,1} };//移动数组：上下左右

int\*\* mark = new int\* [y\_size];//辅助标记二维数组

for (int i = 0; i < y\_size; i++)

{

mark[i] = new int[x\_size];

}

for (int i = 0; i < y\_size; i++)

for (int j = 0;j < x\_size;j++)

{

mark[i][j] = 0;

}

Stack<Point>route;//路径栈

Point current(start\_x,start\_y);

route.push(current);

int now\_x = 0, now\_y = 0;

while (current.x() != end\_x || current.y() != end\_y) {

int k = 0;

for (;k < 4;k++)//依次向四个方向探索

{

now\_x = current.x() + next[k][0];

now\_y = current.y() + next[k][1];

if (now\_x <0 || now\_x >= x\_size|| now\_y <0 || now\_y >= y\_size)//若出边界则进行下一次探索

continue;

if (maze[now\_x][now\_y] == 1 && mark[now\_x][now\_y] == 0)//若为通路且未被探索过则入栈，并从此处继续向下探索

{

mark[now\_x][now\_y] = 1;

current = Point(now\_x, now\_y);

route.push(current);

break;

}

}

if (k == 4)//若四个方向均无通路，则退栈回溯探索

{

route.pop();

current = route.top();

}

if (route.empty()) {//栈空说明无路径

cout << "没有路径!" << endl;

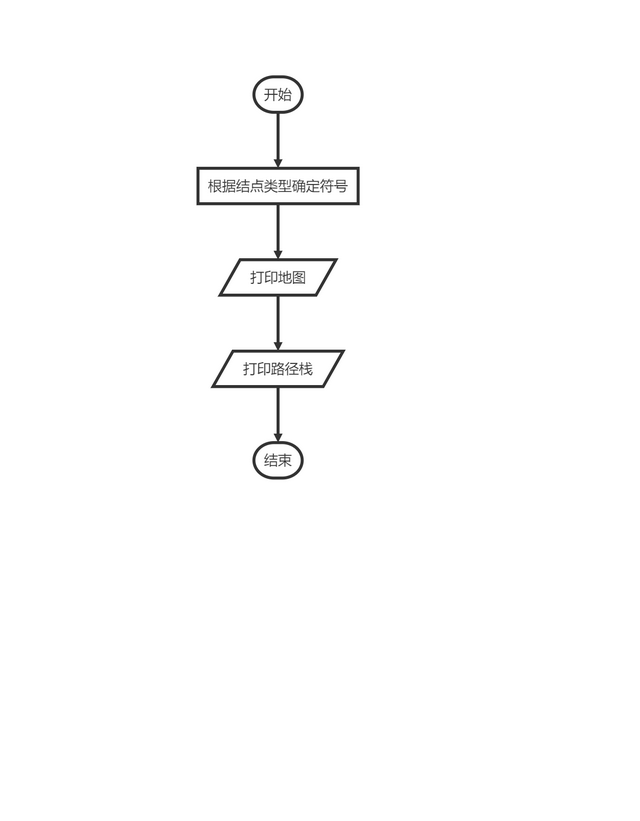
return 0;

}

}

## 3.3 打印功能的实现

### 3.3.1 打印功能流程图



### 3.3.2 打印功能核心代码

//打印地图

Stack<Point> copy\_route;

while (!route.empty()) {

maze[route.top().x()][route.top().y()] = 2;

copy\_route.push(route.top());

route.pop();

}

route = copy\_route;

cout << "迷宫地图 ：" << endl;

for (int j = 0; j < y\_size; j++) {

cout << "\t" << j << "列";

}

cout << endl;

for (int i = 0; i < x\_size; i++) {

cout << i << "行";

for (int j = 0; j < y\_size; j++) {

switch (maze[i][j]) {

default:

case 0:

cout << "\t#";

break;

case 1:

cout << "\t0";

break;

case 2:

cout << "\tx";

break;

}

}

cout << endl;

}

//打印路径

cout << "迷宫路径:" << endl

<< "(" << start\_x << "," << start\_y << ")";

route.pop();

while (!route.empty()) {

cout << " ---> "

<< "(" << route.top().x() << "," << route.top().y() << ")";

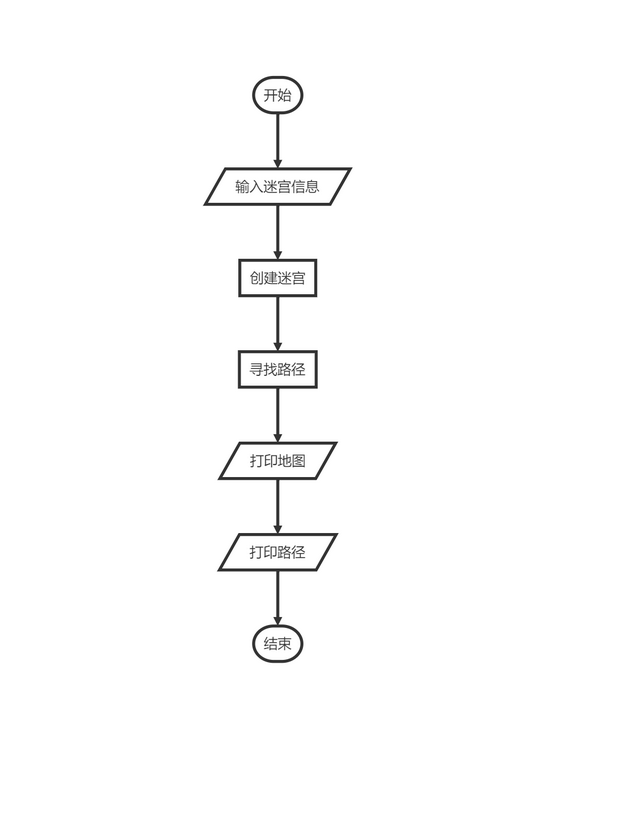
route.pop();

}

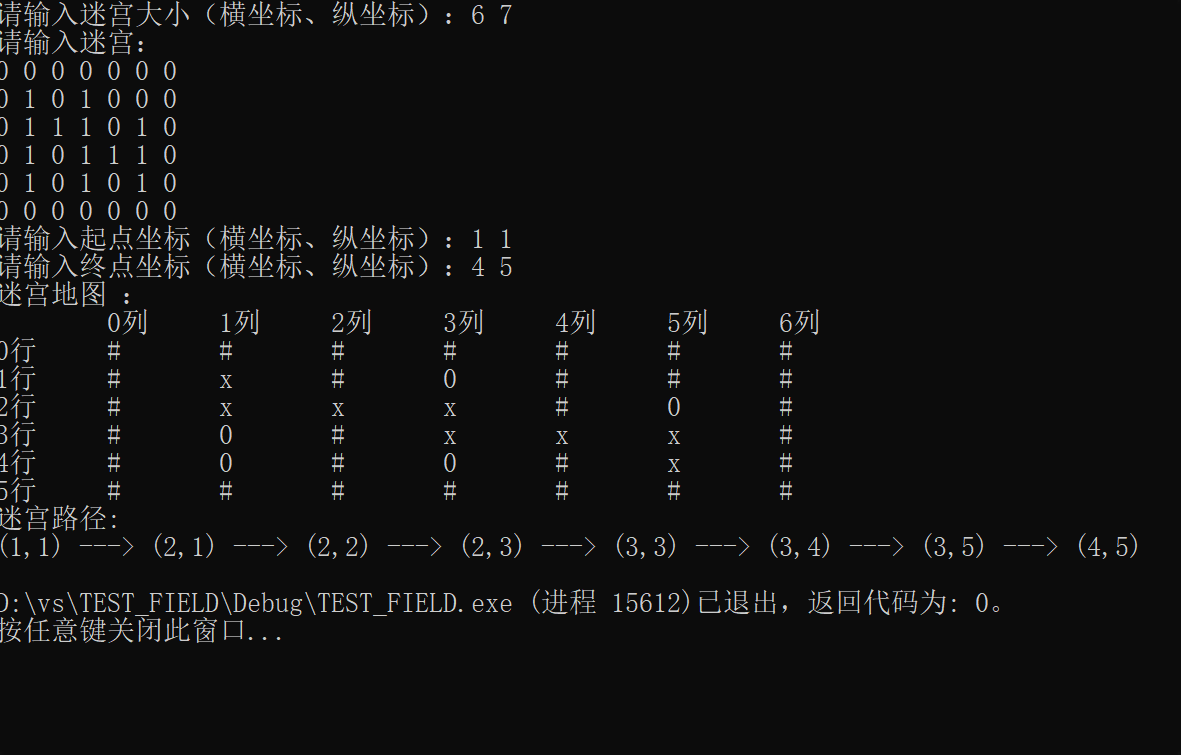
cout << endl;

## 3.4 总体系统的实现

### 3.4.1 总体系统流程图



### 3.4.2 总体系统截屏示例



# 4 测试

## 4.1 功能测试

### 4.1.1带边界横纵等边的迷宫

**测试用例：**

7 7

0 0 0 0 0 0 0

0 1 0 1 1 1 0

0 1 0 1 0 0 0

0 1 1 1 0 1 0

0 1 0 1 1 1 0

0 1 0 1 0 1 0

0 0 0 0 0 0 0

1 1

**5 5**

**预期结果：**

迷宫地图:

0列 1列 2列 3列 4列 5列 6列

0行 # # # # # # #

1行 # x # 0 0 0 #

2行 # x # 0 # # #

3行 # x x x # 0 #

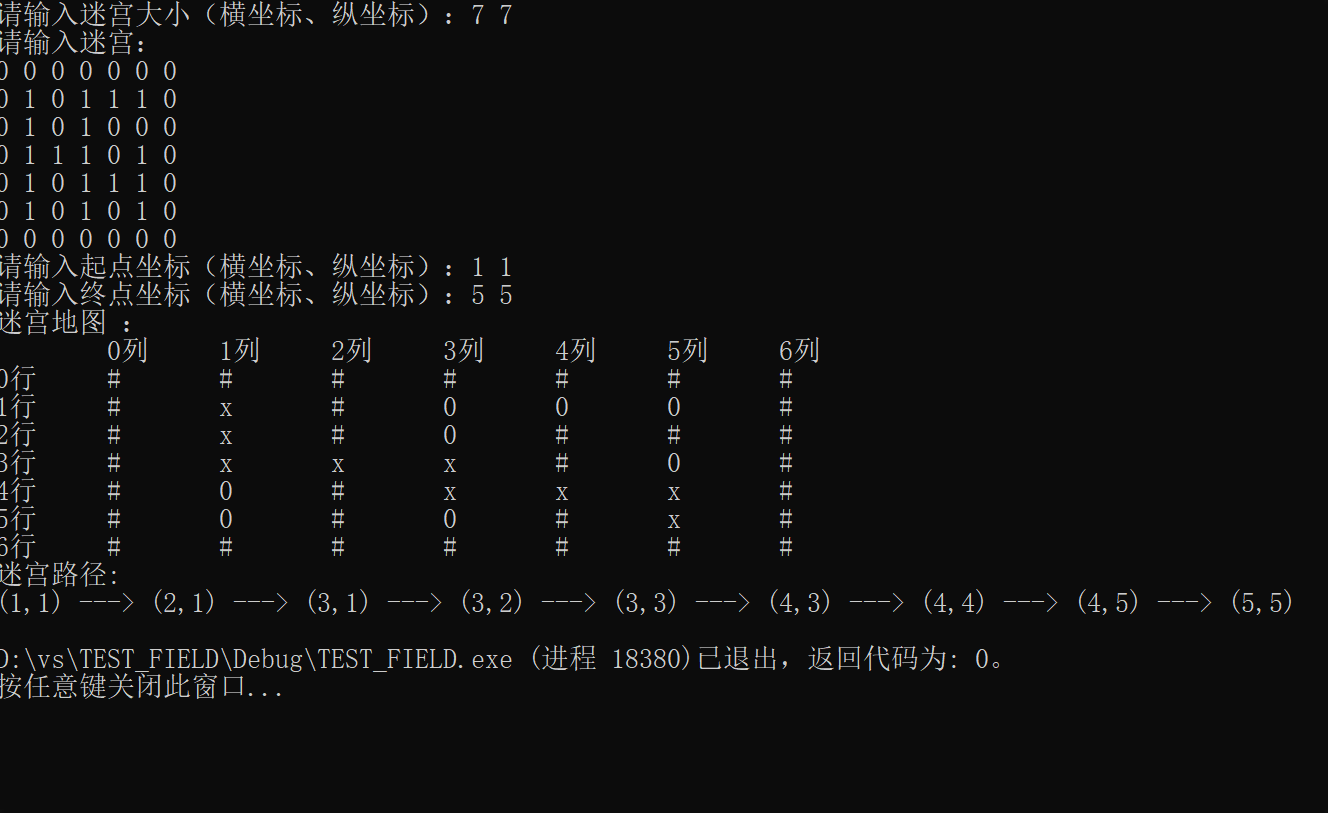
4行 # # # x x x #

5行 # # # # # x #

6行 # # # # # # #

迷宫路径:

(1,1) ---> (2,1) ---> (3,1) ---> (3,2) ---> (3,3) ---> (4,3) ---> (4,4) ---> (4,5) ---> (5,5)

**实验结果：**

### 4.1.2带边界横纵不等边的迷宫

**测试用例：**

6 7

0 0 0 0 0 0 0

0 1 0 1 0 0 0

0 1 1 1 0 1 0

0 1 0 1 1 1 0

0 1 0 1 0 1 0

0 0 0 0 0 0 0

1 1

4 5

**预期结果：**

迷宫地图:

0列 1列 2列 3列 4列 5列 6列

0行 # # # # # # #

1行 # x # 0 # # #

2行 # x x x # 0 #

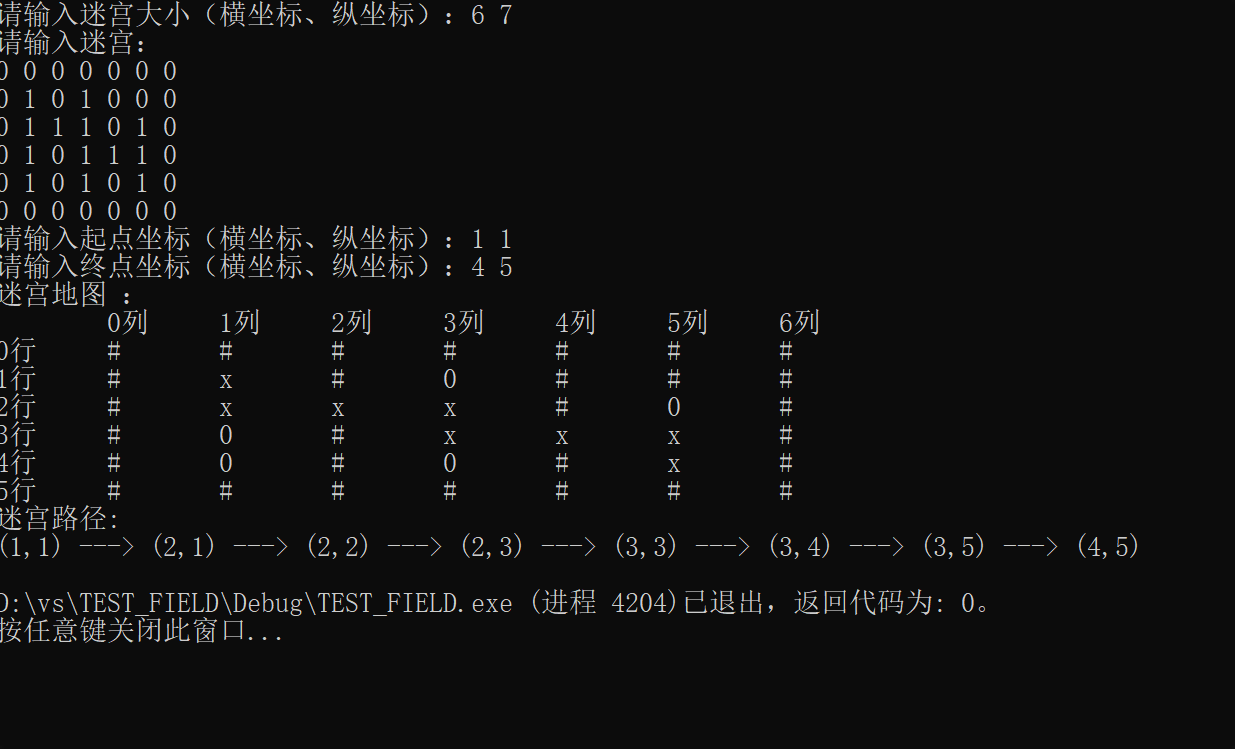
3行 # # # x x x #

4行 # # # # # x #

5行 # # # # # # #

迷宫路径:

(1,1) ---> (2,1) ---> (2,2) ---> (2,3) ---> (3,3) ---> (3,4) ---> (3,5) ---> (4,5)

**实验结果：**

## 4.2 边界测试

### 4.2.1带边界无路径的迷宫

**测试用例：**

6 7

0 0 0 0 0 0 0

0 1 0 1 0 0 0

0 1 1 1 0 1 0

0 1 0 1 1 1 0

0 1 0 1 0 1 0

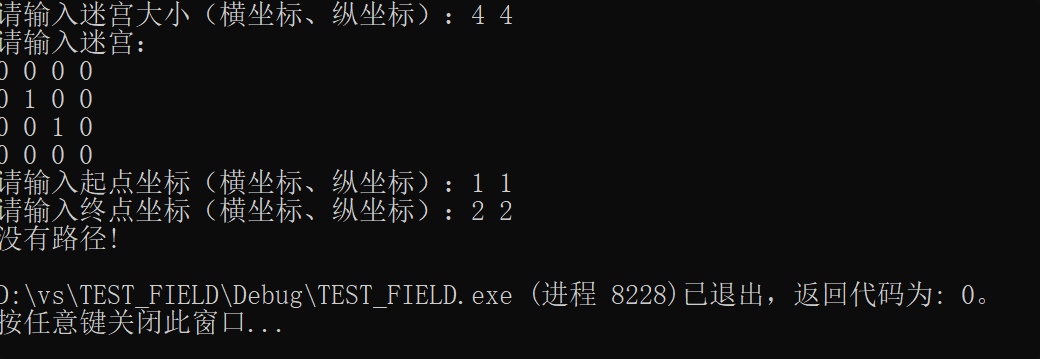
0 0 0 0 0 0 0

1 1

4 5

**预期结果：**

没有路径！

**实验结果：**

### 4.2.1无边界的迷宫

**测试用例：**

5 5

1 0 1 1 1

1 0 1 0 0

1 1 1 0 1

1 0 1 1 1

1 0 1 0 1

0 0

4 4

**预期结果：**

迷宫地图:

0列 1列 2列 3列 4列

0行 x # 0 0 0

1行 x # 0 # #

2行 x x x # 0

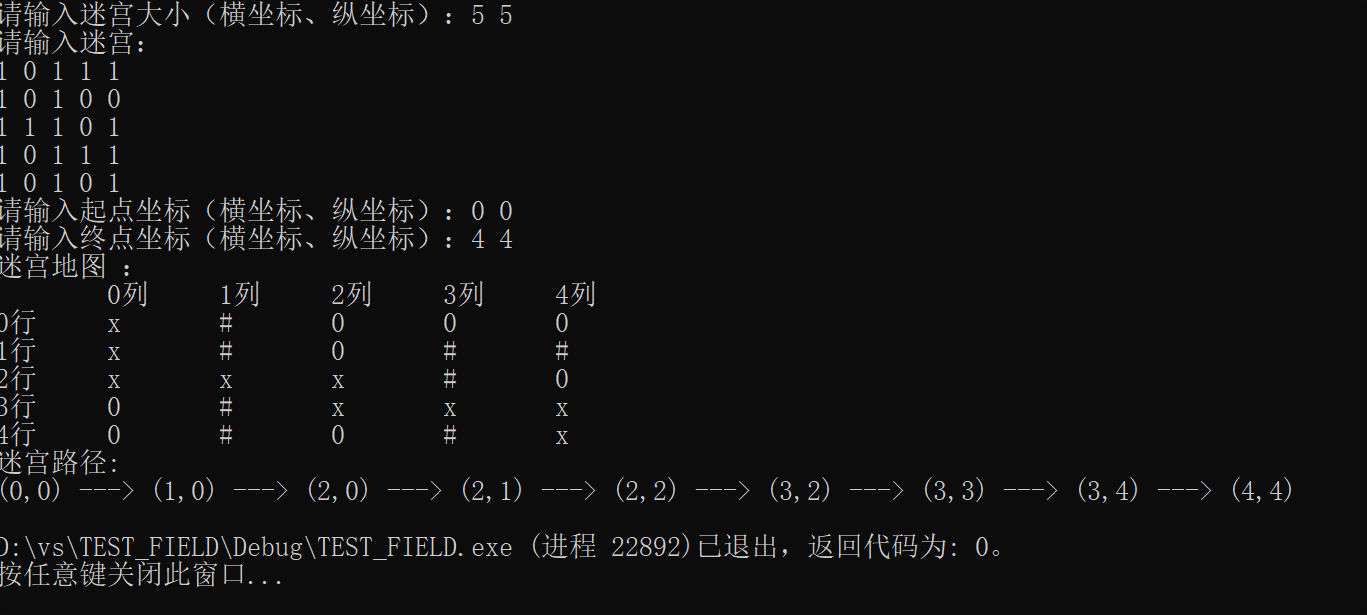
3行 # # x x x

4行 # # # # x

迷宫路径:

(0,0) ---> (1,0) ---> (2,0) ---> (2,1) ---> (2,2) ---> (3,2) ---> (3,3) ---> (3,4) ---> (4,4)

**实验结果：**

****