项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 赵子昱

学 号： 1951459

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](#_Toc59868377)

[1.1 背景分析 1](#_Toc59868378)

[1.2 功能分析 1](#_Toc59868379)

[2 设计 2](#_Toc59868380)

[2.1 数据结构设计 2](#_Toc59868381)

[2.2 类结构设计 2](#_Toc59868382)

[2.3 主函数设计 2](#_Toc59868383)

[3 实现 3](#_Toc59868384)

[3.1 构造函数的实现 3](#_Toc59868385)

[3.1.1 构造函数代码 3](#_Toc59868386)

[3.1.2 构造函数截屏展示 4](#_Toc59868387)

[3.2 Go()函数的实现 4](#_Toc59868388)

[3.2.1 Go()函数流程图 5](#_Toc59868389)

[3.2.2 Go() 、Go(int &x1, int &y1, int &x2, int &y2) 函数的代码 6](#_Toc59868390)

[3.2.3 Go() 函数功能截屏展示 7](#_Toc59868391)

[3.3 showMap() 函数的实现 8](#_Toc59868392)

[3.3.1 showMap() 函数的代码 8](#_Toc59868393)

[3.4 main() 函数的实现 9](#_Toc59868394)

[3.4.1 main() 函数的代码 9](#_Toc59868395)

[4 测试 10](#_Toc59868396)

[4.1 功能测试 10](#_Toc59868397)

[4.1.1 建立迷宫功能测试 10](#_Toc59868398)

[4.1.2 搜索并打印路径功能测试 10](#_Toc59868399)

[4.2 边界测试 11](#_Toc59868400)

[4.2.1 起点或终点坐标有障碍 11](#_Toc59868401)

[4.2.2 起点与终点坐标一样 12](#_Toc59868402)

[4.2.3 没有可行路径 13](#_Toc59868403)

[4.3 出错测试 14](#_Toc59868404)

[4.3.1 建立迷宫行数或列数小于2 14](#_Toc59868405)

[4.3.2 起点或终点坐标超出范围 14](#_Toc59868406)

# 1 分析

## 背景分析

迷宫只有两个门，一个门叫入口，另一个门叫出口。一个骑士骑马从入口进入迷宫，迷宫设置很多障碍，骑士需要在迷宫中寻找通路以到达出口。

## 1.2 功能分析

作为一个迷宫游戏，应该包含以下基本功能：

* 让玩家设置迷宫大小并建立迷宫
* 让玩家设置起点、与终点（设置不合理时给出提示）
* 搜索可行的路径（若没有可行路径给出提示）
* 打印迷宫，并标注可行路径
* 打印路径坐标

# 2 设计

## 2.1 数据结构设计

在搜索可行路径时会遇到无路可走的情况，此时需要原路返回（回溯），故需要记录走过的路径，并且首先返回到最近走过的一个节点，这符合LIFO的特点，故用栈（Stack）来保存走过的路径。

迷宫本身采用二维字符数组储存，为了避免因为回路产生的重复搜索，另外用一个二维布尔数组标记每个节点是否访问过。

## 2.2 类结构设计

项目中除了用到了栈以外，还设计了Maze 类，封装了迷宫的数据与操作， Maze 中的成员如下：

class Maze {

public:

Maze(); //包含迷宫的建立

~Maze();

void Go(); //寻找有效路径并输出（走迷宫）

private:

char \*\*Map; //二维字符数组存储迷宫

bool \*\*sign; //二维布尔数组存储每个节点是否访问过

int row, col; //迷宫的行数，列数

void showMap() const; //打印迷宫

void Go(int &x1, int &y1, int &x2, int &y2); //被Go()调用

};

## 2.3 主函数设计

主函数中首先调用构造函数Maze() 进行让用户进行迷宫的建立，之后调用Go() 函数来进行起点终点的选择，路径的查找、迷宫的输出。

# 3 实现

## 3.1 构造函数的实现

构造函数实现了让用户指定迷宫大小并建立迷宫（为了使搜索操作更加简洁，将迷宫外围加了一圈围墙，也就是将迷宫数组增加两行两列，每个元素都为‘#’），建立与迷宫大小相同的标记节点是否访问过的布尔数组。

### 3.1.1 构造函数代码

Maze::Maze() {

cout << "首先请建立迷宫！" << endl;

cout << "请输入迷宫的行数与列数：";

cin >> row >> col;

if (row < 2 || col < 2) {

cout << "行数与列数至少为2！" << endl;

system("pause");

exit(0);

}//if

cout << "无障碍用 0 表示，有障碍用 # 表示" << endl;

row += 2, col += 2;

Map = new char\*[row];

Map[0] = new char[col];

Map[row -1] = new char[col];

for (int i = 0; i < col; i++) { Map[0][i] = Map[row - 1][i] = '#'; }

for (int i = 0; i < row-2; i++) {

Map[i + 1] = new char[col];

Map[i + 1][0] = Map[i + 1][col - 1] = '#';

cout << "请输入迷宫的第" << i + 1 << "行：";;

for (int j = 0; j < col-2; j++) {

cin >> Map[i + 1][j + 1];

}//for

}//for

sign = new bool\*[7];

for (int i = 0; i < 7; i++) {

sign[i] = new bool[7];

for (int j = 0; j < 7; j++) {

sign[i][j] = false;

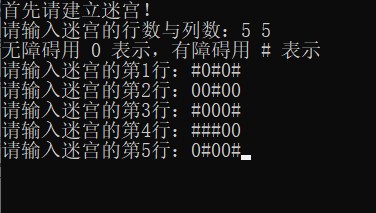
}//for

}//for

system("cls");

}

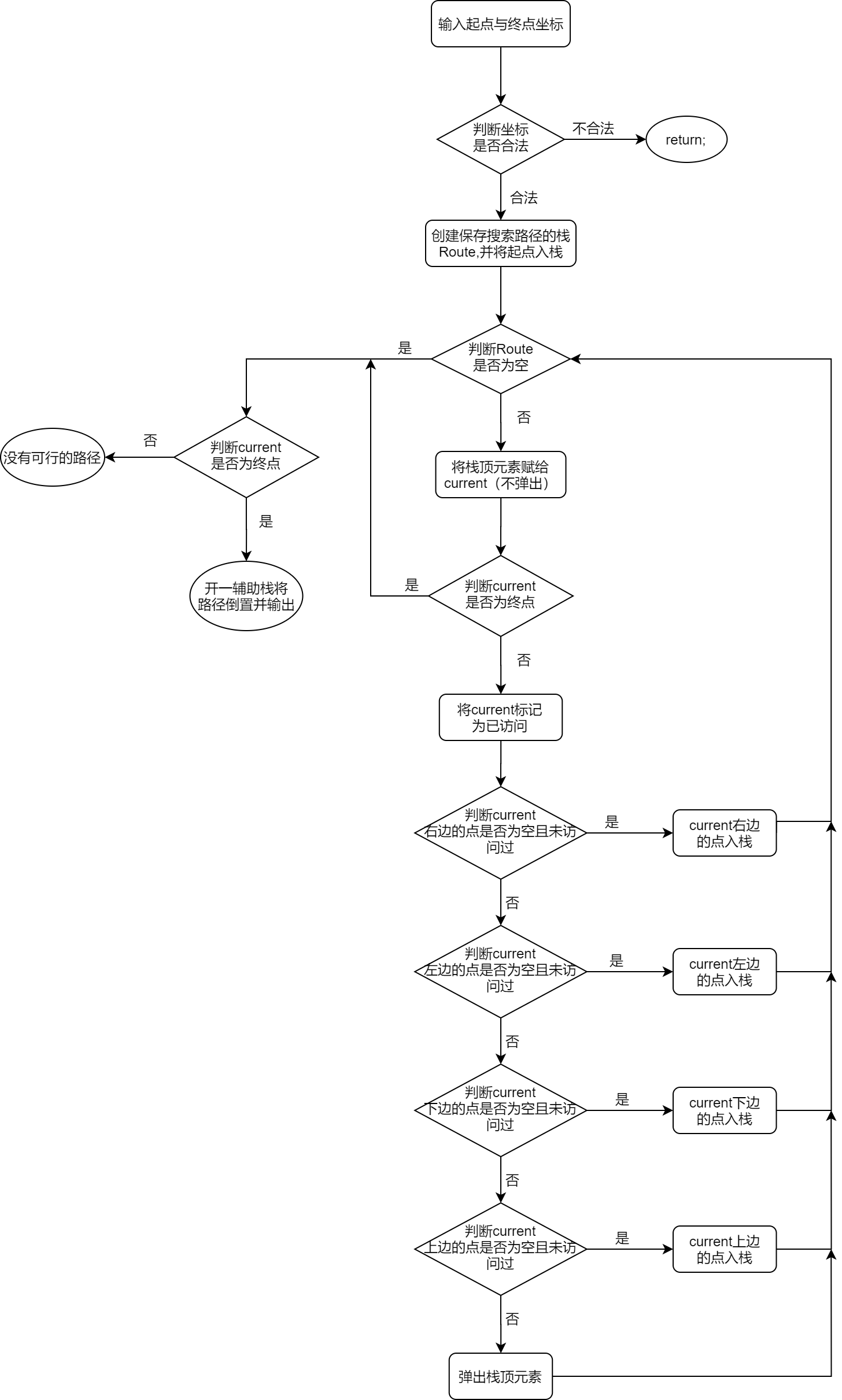
### 3.1.2 构造函数截屏展示



## 3.2 Go()函数的实现

Go() 函数是整个项目的主要函数，实现从起点到终点的路径搜索，主体是一个while循环，当栈不为空时，查看栈顶元素，若栈顶元素是终点则跳出循环，若不为终点则将其标记为已访问并且扫描栈顶上下左右相邻的点，若找到一个为‘0’（可以走）且未被访问的点，入栈，进行下一循环，否则将其弹出，进入下一循环。若找到有效路径将其用一个辅助栈倒置并弹出，否则输出没有路径。

### 3.2.1 Go()函数流程图



### 3.2.2 Go() 、Go(int &x1, int &y1, int &x2, int &y2) 函数的代码

void Maze::Go() {

showMap();

int x1, y1, x2, y2;

cout << "请输入起点与终点坐标：";

cin >> x1 >> y1 >> x2 >> y2;

Go(x1, y1, x2, y2);

}

void Maze::Go(int &x1, int &y1, int &x2, int &y2) {

//(x1,y1)为起点，（x2,y2）为终点

if (x1 >= row || x2 >= row || y1 >= col || y2 >= col

|| x1 < 0 || x2 < 0 || y1 < 0 || y2 < 0 //起点终点左边不合理给出提示

|| Map[x1][y1] != '0' || Map[x2][y2] != '0'||((x1==x2)&&(y1==y2))) {

cout << endl << "\*\*\* 非法的起点或终点!!! \*\*\*" << endl << endl;

return;

}//if

Stack<int> Route; //用于存储查找路径，将点的坐标转化为一维处理

int current = x1 \* col + y1, goal = x2 \* col + y2;//当前访问的节点、目标节点的一维坐标

Route.Push(current);

while (!Route.IsEmpty()) {

current = Route.GetFirst(); //从当前栈顶元素开始查找

if (current == goal) { break; } //到达终点

x1 = current / col, y1 = current % col; //转化为二维坐标

sign[x1][y1] = true; //当前节点标记为已访问

if (Map[x1][y1 + 1] == '0'&&sign[x1][y1 + 1] == false) { y1++; }//寻找下一个可以到达且未访问的点

else if (Map[x1][y1 - 1] == '0'&&sign[x1][y1 - 1] == false) { y1--; }

else if (Map[x1 + 1][y1] == '0'&&sign[x1 + 1][y1] == false) { x1++; }

else if (Map[x1 - 1][y1] == '0'&&sign[x1 - 1][y1] == false) { x1--; }

else { Route.Pop(); continue; } //从当前节点无路可走，将当前节点从栈中弹出

current = x1 \* col + y1; //寻找下一个可以到达且未访问的点得坐标转化为一维

Route.Push(current); //入栈

}//while

if (current != goal) { cout << "没有可行的路径!" << endl; }//回溯到起点，Route为空

else { //该情况下找到了有效路径

Stack<int> trueRoute; //该栈用于将Route倒置

while (!Route.IsEmpty()) {

current = Route.Pop();

Map[current / col][current%col] = '\*'; //标记路径

trueRoute.Push(current);

}//while

system("cls");

showMap();

//一下内容为打印路径

current = trueRoute.Pop();

cout << "迷宫路径：" << endl << endl << "(" << current / col << ',' << current % col << ")";//

while (!trueRoute.IsEmpty()) {

current = trueRoute.Pop();

cout << "->(" << current / col << ',' << current % col << ")";

}

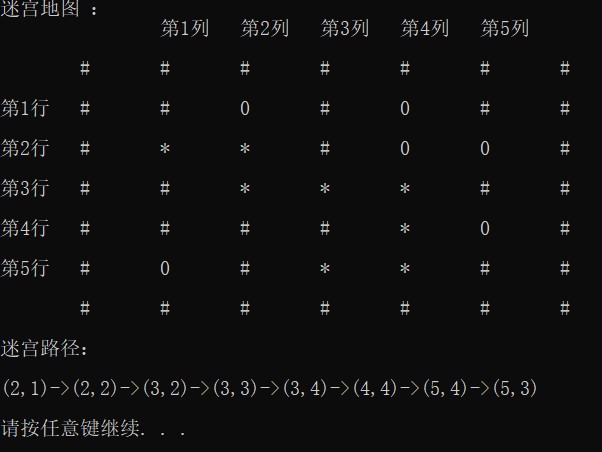
cout << endl << endl;

}//else

}

### 3.2.3 Go() 函数功能截屏展示





## 3.3 showMap() 函数的实现

### 3.3.1 showMap() 函数的代码

void Maze::showMap() const{ //打印迷宫地图

cout << "迷宫地图 ：" << endl;

int realCol = col - 2, realRow = row - 2;

cout << " ";

for (int i = 1; i <= realCol; i++) {

cout << "\t第" << i << "列";

}

cout << endl << endl;

for (int j = 0; j <= col; j++) { cout << '\t' << Map[0][j]; }

cout << endl << endl;

for (int i = 1; i <= realRow; i++) {

cout << "第" << i << "行";

for (int j = 0; j <= col; j++) { cout << '\t' << Map[i][j]; }

cout << endl << endl;

}

for (int j = 0; j <= col; j++) { cout << '\t' << Map[row-1][j]; }

cout << endl << endl;

}

## 3.4 main() 函数的实现

main() 函数中创建了一个Maze 对象，自动调用构造函数进行迷宫的建造，然后调用Go() 函数进行起点终点选择，路径的搜索与打印。

## 3.4.1 main() 函数的代码

int main() {

Maze maze1;

maze1.Go();

system("pause");

return 0;

}

# 4 测试

## 4.1 功能测试

### 4.1.1 建立迷宫功能测试

**测试用例：**

5 5

#0#0#

00#00

#000#

###00

0#00#

**预期结果**

建立相应大小与图形的迷宫

**测试结果**

****

### 4.1.2 搜索并打印路径功能测试

**测试用例：**

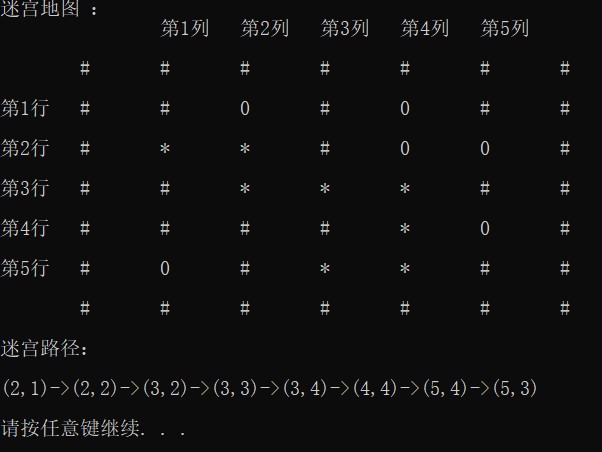
迷宫同上

起终点坐标：2 1 5 3

**预期结果**

存在有效路径，在地图上标注并打印路径

**测试结果**



## 4.2 边界测试

### 4.2.1 起点或终点坐标有障碍

**测试用例：**

1 2 1 1

**预期结果**

给出提示，程序不崩溃

**测试结果**



### 4.2.2 起点与终点坐标一样

**测试用例：**

1 1 1 1

**预期结果**

给出提示，程序不崩溃

**测试结果**



### 4.2.3 没有可行路径

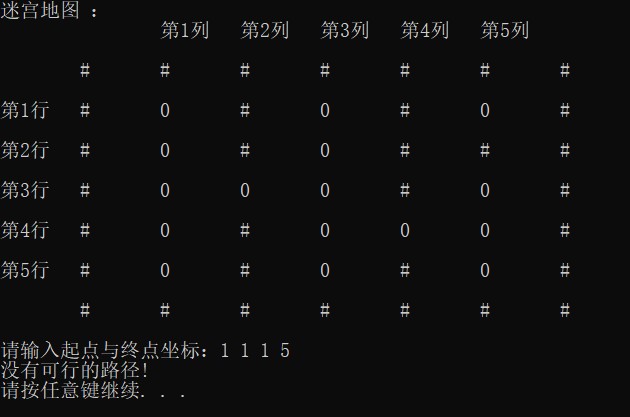
**测试用例：**

1 1 1 1

**预期结果**

给出提示，程序不崩溃

**测试结果**



## 4.3 出错测试

### 4.3.1 建立迷宫行数或列数小于2

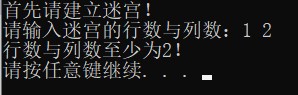
**测试用例：**

1 2

**预期结果**

给出提示，程序不崩溃

**测试结果**

****

### 4.3.2 起点或终点坐标超出范围

**测试用例：**

1 6 1 1

**预期结果**

给出提示，程序不崩溃

**测试结果**

****