项目说明文档

数据结构课程设计

——勇闯迷宫

作 者 姓 名： 张喆

学 号： 1754060

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](#_Toc531644398)

[1.1 背景分析 1](#_Toc531644399)

[1.2 功能分析 1](#_Toc531644400)

[2 设计 2](#_Toc531644401)

[2.1 数据结构设计 2](#_Toc531644402)

[2.2 类结构设计 2](#_Toc531644403)

[2.3 成员与操作设计 2](#_Toc531644404)

[2.4 系统设计 3](#_Toc531644405)

[3 实现 4](#_Toc531644406)

[3.1 读入地图功能的实现 4](#_Toc531644407)

[3.1.1 读入地图功能核心代码 4](#_Toc531644408)

[3.2 寻找缺口功能的实现 4](#_Toc531644409)

[3.2.1 寻找缺口功能核心代码 4](#_Toc531644410)

[3.2.2 寻找缺口功能截屏示例（调试用） 6](#_Toc531644411)

[3.3 寻找路径功能的实现 7](#_Toc531644412)

[3.3.1 寻找路径功能流程图 7](#_Toc531644413)

[3.3.2 寻找路径功能核心代码 8](#_Toc531644414)

[3.3.3 寻找路径功能截屏示例 9](#_Toc531644415)

[3.4 输出寻路结果的实现 10](#_Toc531644416)

[3.4.1 输出寻路结果程序框图 10](#_Toc531644417)

[3.4.1 输出寻路结果核心代码 10](#_Toc531644418)

[3.4.2 输出寻路结果截屏示例 11](#_Toc531644419)

[3.5 总体系统的实现 11](#_Toc531644420)

[3.5.1 总体系统核心代码 11](#_Toc531644421)

[3.5.2 总体系统截屏示例 12](#_Toc531644422)

[4 测试 13](#_Toc531644423)

[4.1 功能测试 13](#_Toc531644424)

[4.1.1 寻找缺口功能测试 13](#_Toc531644425)

[4.1.2 寻路功能测试 14](#_Toc531644426)

[4.2 边界测试 16](#_Toc531644427)

[4.2.1 出发点和终点在同一方向上 16](#_Toc531644428)

[4.2.2 迷宫无解 16](#_Toc531644429)

# 1 分析

## 1.1 背景分析

迷宫问题是综合考察递归，回溯方法等知识点的非常经典的应用。

回溯法将问题的候选解按某种顺序逐一枚举和检验。当发现当前的候选解不可能是解时，就放弃它而选择下一个候选解。如果当前的候选解除了不满足问题的规模要求外，其他所有要求都已经满足，则扩大当前候选解的规模继续试探。如果当前的候选解满足了包括问题规模在内的所有要求，则这个候选解将成为问题的一个解。

在求解迷宫问题的过程中，当沿某一条路径一步步走向出口但发现进入死胡同走不通时，就回溯一步或多步，寻找其他可走的路径。

为了防止重走原路，另外设置一个标志矩阵，一旦行进到迷宫的某个位置，就将这个标志矩阵的这一位标记已经来过，下次这个位置就不能再走了。

## 1.2 功能分析

作为一个勇闯迷宫的问题，最基本的要先能读入用户指定的迷宫。

本题采用用户自定义row长col宽的任意迷宫，需在四周有且仅有两个缺口，以此作为迷宫的入口和出口，算法可自行寻找四周有且仅有的两个缺口。

把用户指定的迷宫地图读入之后应能使用回溯的方法一步步试探的找到走出迷宫的最佳路线。

如果此迷宫有解，则按照前进的路线输出最佳路线上的各个点坐标；如果此迷宫无解，则输出相应信息提示用户。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该迷宫要使用回溯法一步步试探着找到出口。因此，采用这种递归调用的方法最先返回值的应该是最后一步找到出口或者无论怎么样都没不能走出迷宫的情况，然后才能轮到倒数第二步返回，然后是倒数第三步等等。

因此采用一个工作栈的后进先出的存储结构记录行进过程中经过的各个点坐标，这样才能达到正序输出行进的顺序。

## 2.2 类结构设计

设计了坐标类Point，拥有成员x，y用于表示地图上的点坐标

设计了前进方向表类offsets，预先定义某方向所有可能的移动方向。

设计了迷宫类Maze，具有进行读入地图、寻路、输出寻路方案等功能

## 2.3 成员与操作设计

**坐标类（Point）**

**公有成员：**

int x, y;

**坐标类的构造函数：**

Point() = default;

Point(int a, int b) :x(a), y(b) {}

**坐标类的重载运算符：**

bool operator==(const Point &rhv);

bool Point::operator==(const Point &rhv)

{

if (this->x == rhv.x && this->y == rhv.y)

{

return true;

}

else { return false; }

}

**前进方向表类（offsets）**

**公有成员：**

Point pos; //方向向量

const char \*dir; //方向名字

**迷宫类（Maze）**

**私有成员：**

int row, col;

vector<vector<int> > Map; //地图

vector<vector<int> > mark; //标记地图, 用于判定某个位置是否已经走过

Point start, dest; //起点和终点

stack<Point> AccessStack; //使用工作栈储存行进中各点的坐标

**迷宫类的构造函数：**

Maze() = default;

Maze(int \_row, int \_col);

**迷宫类的成员函数：**

void InitMap(); //输入迷宫地图

void FindGap(); //寻找起点和终点

void showGap(); //显示起点和终点

void SeekPath(); //无参的寻路函数

bool SeekPath(Point now); //递归调用的寻路函数

void showRoute(); //显示寻路的结果

void showMap(); //输出地图

## 2.4 系统设计

系统预先定义了上下左右四个方向的前进方向表Move用于每一次递归指明前进的方向。

Offsets Move[4] =

{ { Point(-1,0),"N" },{ Point(0,1),"E" },

{ Point(1,0),"S" },{ Point(0,-1),"W" }, };

//四个方位的向量

系统开始时先让用户输入自定义迷宫的行数和列数，接着让用户输入迷宫地图。

调用FindGap(row, col, Maze, start, dest); 寻找此迷宫地图中的出发点和终点坐标；再创建工作栈用于储存行进中各个位置的坐标。

调用SeekPath(start, dest, Maze, mark, AccessStack) 寻找在此迷宫中的前进路径。

如果工作栈不空，则从栈顶至栈底输出栈中的位置坐标；如果栈空则代表此迷宫无解，输出相应的提示信息提示用户此迷宫无解。

# 3 实现

## 3.1 读入地图功能的实现

### 3.1.1 读入地图功能核心代码

Maze::Maze(int \_row, int \_col)

:row(\_row),col(\_col)

{

Map.resize(row, vector<int>(col, 0));

mark.resize(row, vector<int>(col, 0));

}

void Maze::InitMap()

{

//读入row行col列的迷宫

for (int i = 0; i < row; ++i)

{

for (int j = 0; j < col; ++j)

{

mark[i][j] = 0;

cin >> Map[i][j];

}

}

}

## 3.2 寻找缺口功能的实现

### 3.2.1 寻找缺口功能核心代码

void Maze::FindGap()

//在地图上找出发点和终点的位置坐标

//做法是在地图的四周找唯一的两个ACC, 然后随机赋值给出发点和终点

//默认是四周有且仅有两个ACC

{

int i, j;

int cnt = 0;

//上

for (int j = 0; j < col; ++j)

{

if (Map[0][j] == ACC)

{

++cnt;

if (cnt == 1)

{

start = Point(0, j);

}

else if (cnt == 2)

{

dest = Point(0, j);

return;

}

}

}

//下

for (j = 0; j < col; ++j)

{

if (Map[row - 1][j] == ACC)

{

++cnt;

if (cnt == 1)

{

start = Point(row - 1, j);

}

else if (cnt == 2)

{

dest = Point(row - 1, j);

return;

}

}

}

//左

for (i = 1; i < row - 1; ++i)

{

if (Map[i][0] == ACC)

{

++cnt;

if (cnt == 1)

{

start = Point(i, 0);

}

else if (cnt == 2)

{

dest = Point(i, 0);

return;

}

}

}

//右

for (i = 1; i < row - 1; ++i)

{

if (Map[i][col - 1] == ACC)

{

++cnt;

if (cnt == 1)

{

start = Point(i, col - 1);

}

else if (cnt == 2)

{

dest = Point(i, col - 1);

return;

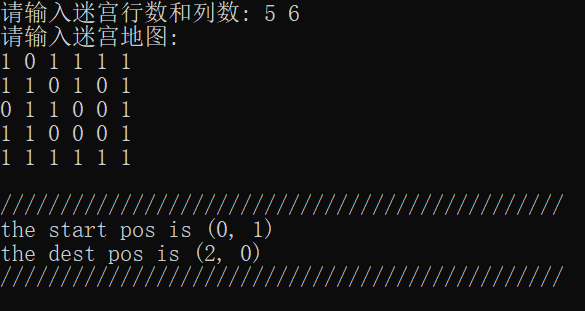
}

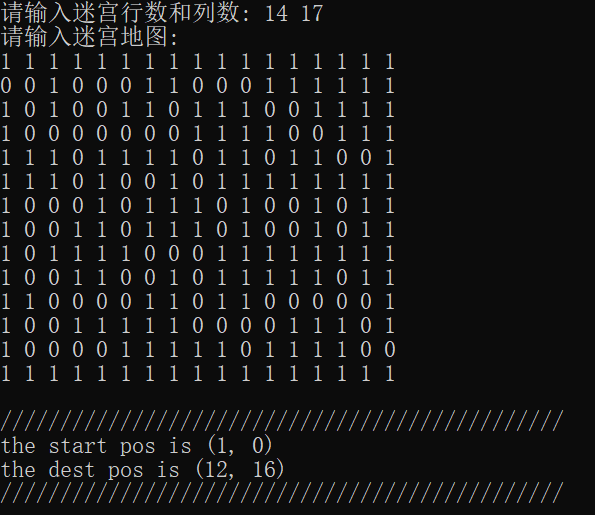
}

}

}

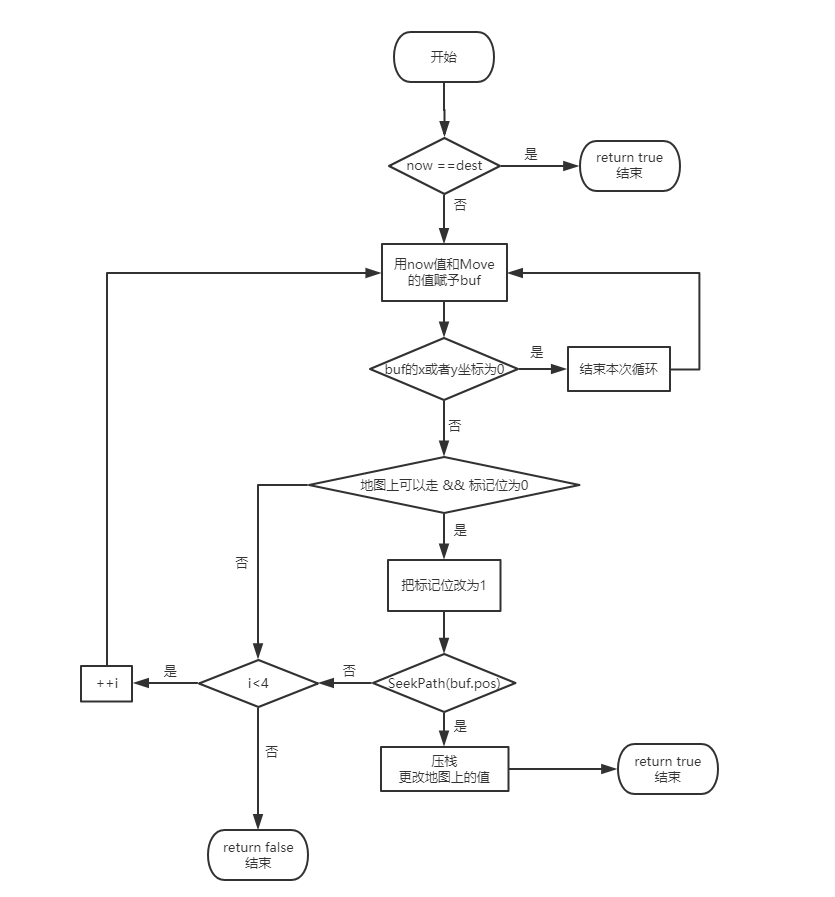
### 3.2.2 寻找缺口功能截屏示例（调试用）





## 3.3 寻找路径功能的实现

### 3.3.1 寻找路径功能流程图



### 3.3.2 寻找路径功能核心代码

void Maze::SeekPath()

{

mark[start.x][start.y] = 1; //出发点位置坐标

if (SeekPath(start))

{

AccessStack.push(start); //如果此迷宫有解,则把起始点位置压入栈中

Map[start.x][start.y] = 5;

}

}

bool Maze::SeekPath(Point now)

//当前位置 终点位置 整张地图 记录哪些点走过了的地图 结果栈用于记录每一步的坐标

{

Offsets buf; //假设往这个方向走的替代变量

if (now == dest) { return true; } //如果找到终点了就退出递归并返回true

for (int i = 0; i < 4; ++i)

//4个方向都尝试一遍能不能走

{

buf.pos.x = now.x + Move[i].pos.x;

buf.pos.y = now.y + Move[i].pos.y;

buf.dir = Move[i].dir;

if (buf.pos.x < 0 || buf.pos.y < 0) { continue; } //如果试探的走出了地图则直接换方向

if (Map[buf.pos.x][buf.pos.y] == ACC && mark[buf.pos.x][buf.pos.y] == 0)

//如果地图上这个点可以走 && 以前没往这尝试过

{

mark[buf.pos.x][buf.pos.y] = 1; //往这尝试的时候把mark地图修改一下

if (SeekPath(buf.pos))

{

AccessStack.push(buf.pos);

//因为只有最后一步可以走,才可以把坐标压入栈中

//所以使用后进先出的结构储存行进中各点的坐标

Map[buf.pos.x][buf.pos.y] = 5; //地图上显示走的路径

return true;

}

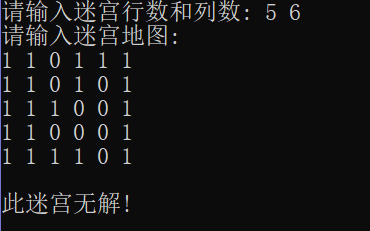
}

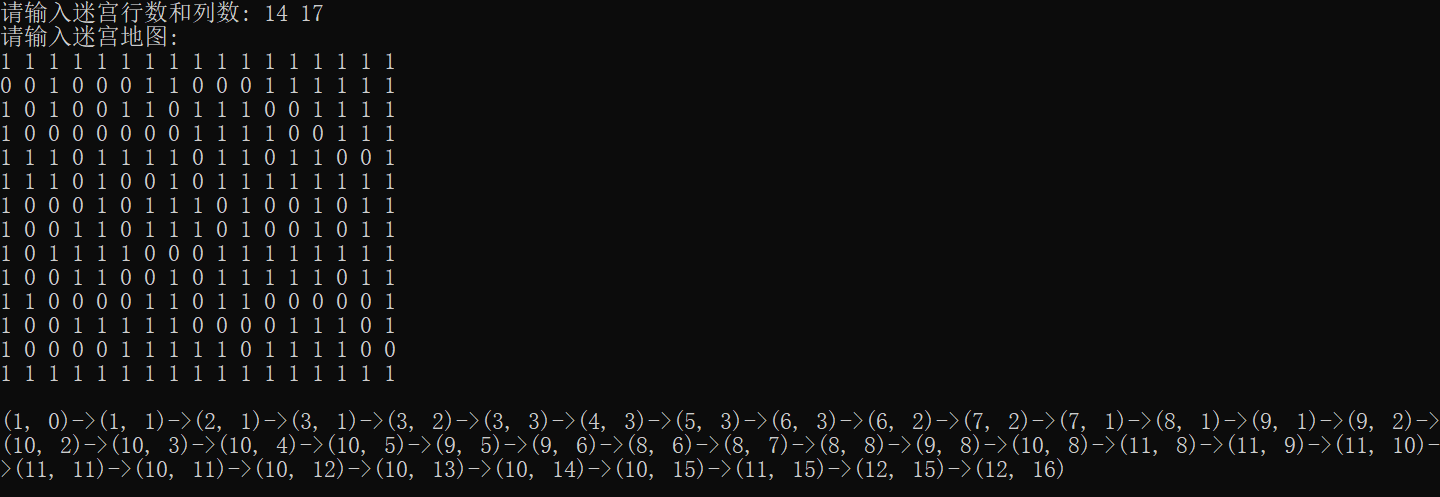
}

return false;

}

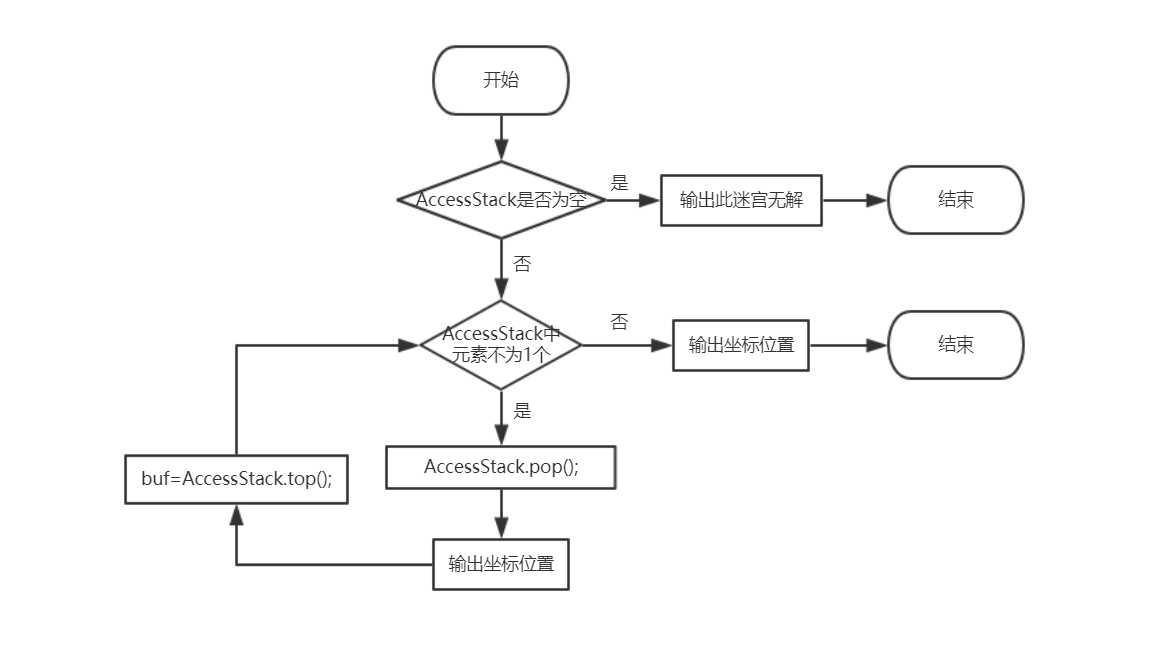
### 3.3.3 寻找路径功能截屏示例





## 3.4 输出寻路结果的实现

### 3.4.1 输出寻路结果程序框图



### 3.4.1 输出寻路结果核心代码

void Maze::showRoute()

{

cout << endl;

if (!AccessStack.empty())

{

Point buf;

for (buf = AccessStack.top(); AccessStack.size() != 1; buf = AccessStack.top())

{

AccessStack.pop();

printf("(%d, %d)->", buf.x, buf.y);

}

printf("(%d, %d)\n", buf.x, buf.y); //最后一个结点不输出箭头

}

else

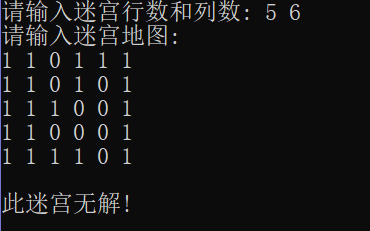
{

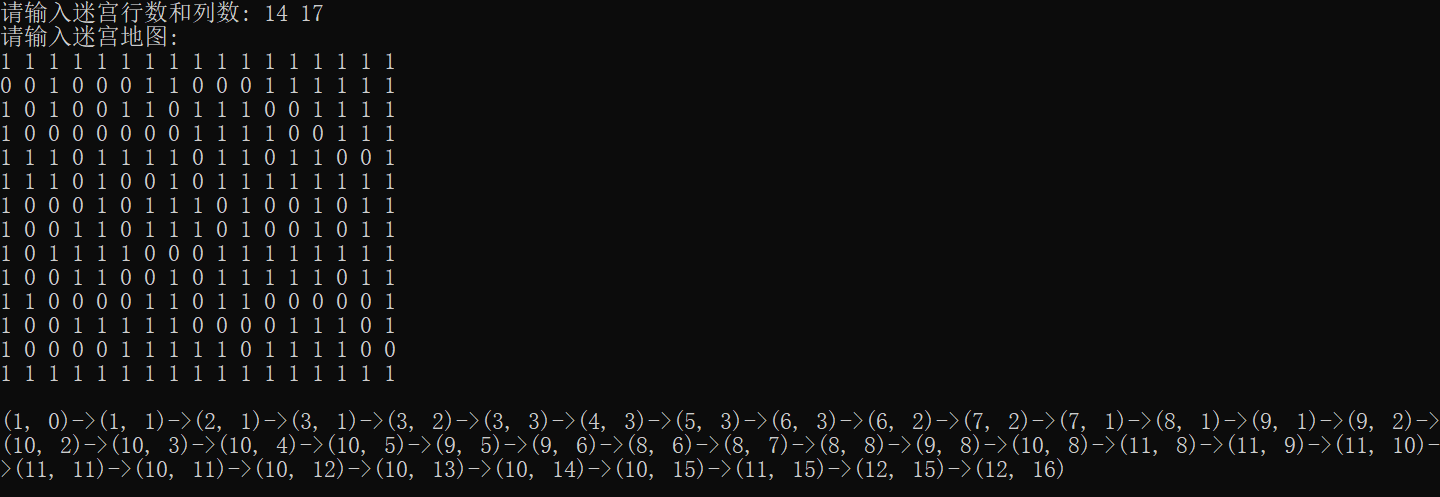
cout << "此迷宫无解!\n";

}

}

### 3.4.2 输出寻路结果截屏示例





## 3.5 总体系统的实现

### 3.5.1 总体系统核心代码

int row, col;

cout << "请输入迷宫行数和列数: ";

cin >> row >> col;

/\*创建 row\*col 尺寸的地图\*/

Maze maze(row, col);

/\*读入地图\*/

cout << "请输入迷宫地图: " << endl;

maze.InitMap();

/\*寻找出发点和终点坐标\*/

maze.FindGap();

/\*调试用: 显示出发点坐标和终点坐标\*/

//maze.showGap();

/\*寻路算法\*/

maze.SeekPath();

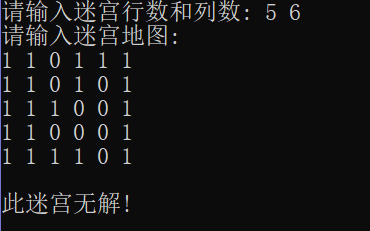
/\*输出找到的路径或者无解信息\*/

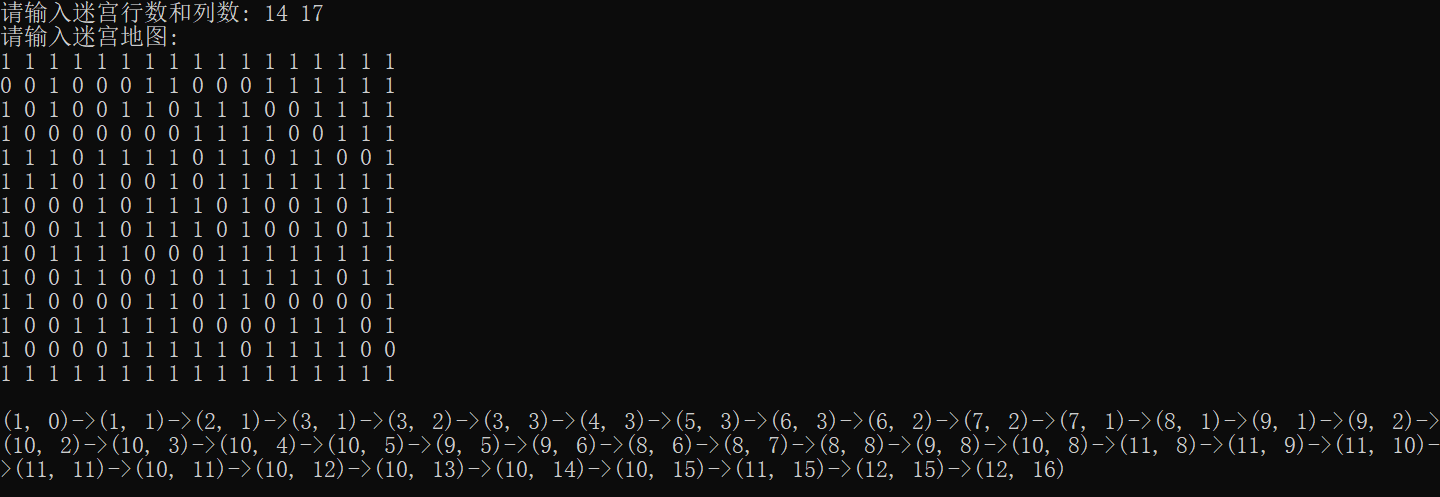
maze.showRoute();

/\*调试用: 输出寻路后的地图\*/

//maze.showMap();

### 3.5.2 总体系统截屏示例





# 4 测试

# 4.1 功能测试

### 4.1.1 寻找缺口功能测试

**测试用例**：

5 6

1 1 0 1 1 1

1 1 0 1 0 1

1 1 1 0 0 1

1 1 0 0 0 1

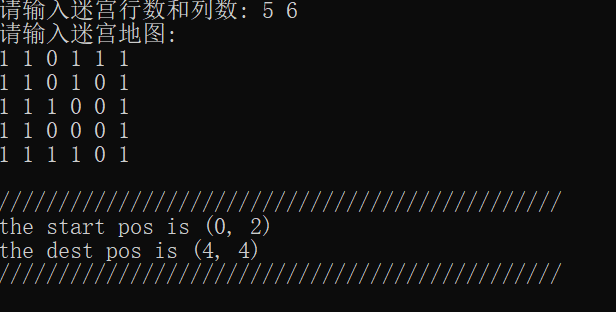
1 1 1 1 0 1

**预期结果**：

the start pos is (0, 2)

the dest pos is (4, 4)

**实验结果**



**测试用例**：

5 6

1 0 1 1 1 1

1 1 0 1 0 1

0 1 1 0 0 1

1 1 0 0 0 1

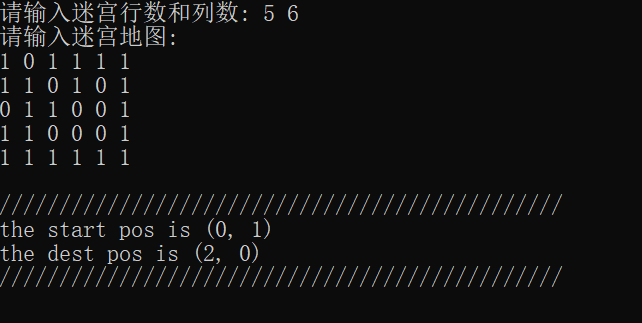
1 1 1 1 1 1

**预期结果**：

the start pos is (0, 1)

the dest pos is (2, 0)

**实验结果**



### 4.1.2 寻路功能测试

**测试用例：**

5 6

1 0 1 1 1 1

1 1 0 1 0 1

0 1 1 0 0 1

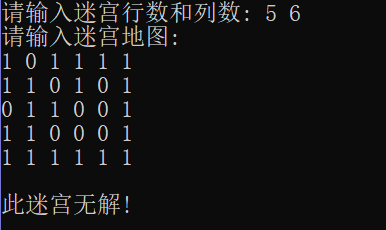
1 1 0 0 0 1

1 1 1 1 1 1

**预期结果：**

此迷宫无解

**实验结果：**



**测试用例：**

14 17

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

0 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1

1 0 1 0 0 1 1 0 1 1 1 0 0 1 1 1 1

1 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1

1 1 1 0 1 1 1 1 0 1 1 0 1 1 0 0 1

1 1 1 0 1 0 0 1 0 1 1 1 1 1 1 1 1

1 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1

1 0 0 1 1 0 1 1 1 0 1 0 0 1 0 1 1

1 0 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1

1 0 0 1 1 0 0 1 0 1 1 1 1 1 0 1 1

1 1 0 0 0 0 1 1 0 1 1 0 0 0 0 0 1

1 0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 1

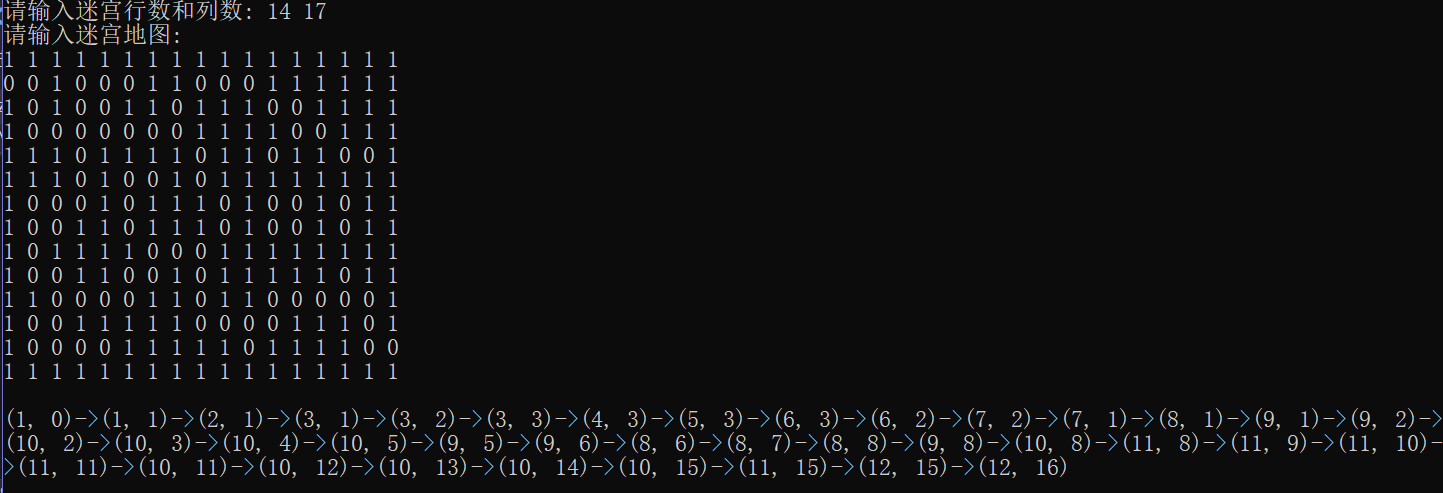
1 0 0 0 0 1 1 1 1 1 0 1 1 1 1 0 0

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

**预期结果：**

(1, 0)->(1, 1)->(2, 1)->(3, 1)->(3, 2)->(3, 3)->(4, 3)->(5, 3)->(6, 3)->(6, 2)->(7, 2)->(7, 1)->(8, 1)->(9, 1)->(9, 2)->(10, 2)->(10, 3)->(10, 4)->(10, 5)->(9, 5)->(9, 6)->(8, 6)->(8, 7)->(8, 8)->(9, 8)->(10, 8)->(11, 8)->(11, 9)->(11, 10)->(11, 11)->(10, 11)->(10, 12)->(10, 13)->(10, 14)->(10, 15)->(11, 15)->(12, 15)->(12, 16)

**实验结果：**



## 4.2 边界测试

### 4.2.1 出发点和终点在同一方向上

**测试用例：**

5 6

1 0 1 1 0 1

1 1 0 1 0 1

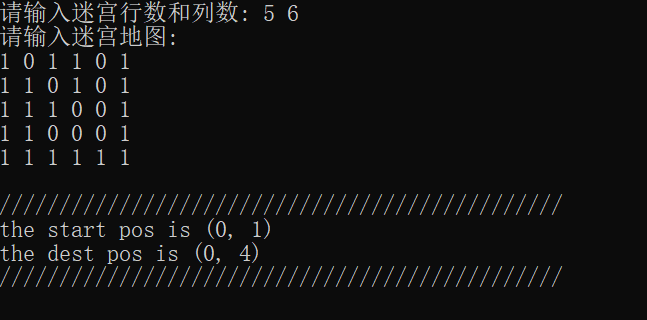
1 1 1 0 0 1

1 1 0 0 0 1

1 1 1 1 1 1

**预期结果：**程序运行正常不崩溃，仍能找出出发点和终点坐标。

**实验结果：**



### 4.2.2 迷宫无解

**测试用例：**

5 6

1 0 1 1 0 1

1 1 0 1 0 1

1 1 1 0 0 1

1 1 0 0 0 1

1 1 1 1 1 1

**预期结果：**程序输出此迷宫无解。

**实验结果：**

