勇闯迷宫游戏

作 者 姓 名： 王凌

学 号： 1951504

指 导 教 师： 张颖

学院、 专业： 软件工程

同济大学

Tongji University

目录

[1 分析 2](#_Toc32441)

[1.1 项目简介 2](#_Toc23475)

[1.2 功能要求 2](#_Toc5639)

[2 设计 3](#_Toc22195)

[2.1 Stack设计 3](#_Toc23395)

[2.2 主程序设计 3](#_Toc30762)

[3 实现 4](#_Toc25727)

[3.1 Stack类实现 4](#_Toc19954)

[3.1.1 Stack()构造函数实现 4](#_Toc25338)

[3.1.2 push()压栈函数实现 4](#_Toc22955)

[3.1.3 pop()出栈函数实现 4](#_Toc5735)

[3.1.4 top()，empty()函数的实现 4](#_Toc30517)

[3.1.5 isFull()，clear()函数的实现 5](#_Toc25463)

[3.2 主函数功能的实现 5](#_Toc14238)

[3.2.1 isValid()函数的实现 5](#_Toc24986)

[3.2.2 print()输出路径函数 6](#_Toc23748)

[3.2.3 dfs()深搜函数的实现 6](#_Toc31383)

[3.2.4 主函数的实现 7](#_Toc17174)

[4 测试 9](#_Toc21479)

[4.1 功能测试 9](#_Toc12796)

[4.1.1测试1 9](#_Toc14291)

[4.1.2 测试2 10](#_Toc4346)

[4.1.3 测试3 11](#_Toc9388)

[4.2 边界测试 13](#_Toc11923)

[4.2.1 没有路径 13](#_Toc6803)

[4.2.2 入口、出口为墙 14](#_Toc28518)

[4.2.3 入口、出口越界 15](#_Toc17575)

# 1 分析

## 1.1 项目简介

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 功能要求

迷宫问题的求解可以根据需求采用深搜和宽搜两种方法。当求最短路径长度而不要求输出路径的时候，一般采用宽搜；但是如果要记录输出路径的话，一般采用深搜更方便。深搜中的回溯法指的是若无法再前进，则return回上一层，消除影响之后选择其它方向继续搜索。从入口出发，向能走到的所有方向搜索，若能走通，则到达新点再向所有方向搜索；若所有的方向均没有通路，则沿原路返回上一个点，换下一个方向再继续试探，直到所有可能的道路都探索到。

为了记录迷宫的路径，我们将所经过的点坐标放进栈内。

# 2 设计

## 2.1 Stack设计

包括两个私有成员变量list数组和栈顶位置top0。

公有成员函数如下：

Stack();//构造函数，初始化栈

void push(const T &item);//将元素item压入栈

T pop();//将栈顶元素弹出栈

void clear();//将栈清空

const T &top() const;//访问栈顶元素

bool empty() const;//测试是否栈满

bool isFull() const;//测试是否栈空

## 2.2 主程序设计

由用户输入迷宫的规模以及迷宫具体地形，默认入口为（1，1），出口在地图右下角处。按要求在输出界面展示迷宫，通过深搜寻找路径，并且用一个栈存储路径，如果找到了从栈中取出路径输出，如果没找到给出没有路径的提示。

# 3 实现

## 3.1 Stack类实现

### 3.1.1 Stack()构造函数实现

栈的自定义构造函数，栈顶初始化为-1。

1. **template** <**class** T,**int** SIZE>
2. Stack<T,SIZE>::Stack():top0(-1){}//构造函数，栈顶初始化为-1

### 3.1.2 push()压栈函数实现

如果栈满了，则报错；否则将新元素压入栈顶

1. **template** <**class** T,**int** SIZE>
2. **void** Stack<T,SIZE>::push(**const** T &item)//将元素item压入栈
3. {
4. assert(!isFull());//如果栈满了，则报错
5. list[++top0]=item;//将新元素压入栈顶
6. }

### 3.1.3 pop()出栈函数实现

如果栈为空，则报错；否则返回栈顶元素，并将其弹出

1. **template** <**class** T,**int** SIZE>
2. T Stack<T,SIZE>::pop()//将栈顶元素弹出栈
3. {
4. assert(!empty());//如果栈为空，则报错
5. **return** list[top0--];//返回栈顶元素，并将其弹出栈顶
6. }

### 3.1.4 top()，empty()函数的实现

访问栈顶元素，若为空则报错。

测试栈是否为空，判断栈顶指针是否为-1。

1. **template** <**class** T,**int** SIZE>
2. **const** T &Stack<T,SIZE>::top() **const**//访问栈顶元素
3. {
4. assert(!empty());//如果栈为空，则报错
5. **return** list[top0];//返回栈顶元素
6. }
7. **template** <**class** T,**int** SIZE>
8. **bool** Stack<T,SIZE>::empty() **const**//测试栈是否空
9. {
10. **return** top0==-1;
11. }

### 3.1.5 isFull()，clear()函数的实现

测试栈是否为满，判断栈顶指针是否等于SIZE-1；

清空栈则是让栈顶指针指向-1。

1. **template** <**class** T,**int** SIZE>
2. **bool** Stack<T,SIZE>::isFull() **const**//测试是否栈满
3. {
4. **return** top0==SIZE-1;
5. }
6. **template** <**class** T,**int** SIZE>
7. **void** Stack<T, SIZE>::clear()//清空栈
8. {
9. top0=-1;
10. }

## 3.2 主函数功能的实现

### 3.2.1 isValid()函数的实现

用于检查起始，终止位置合理性，是否越界，是否为墙，并且给相应的提示，如果不合法，就直接结束程序。

1. **bool** isValid(**int** sx,**int** sy,**int** tx,**int** ty)
2. {
3. **if**(sx<1 || sx>map\_a || sy<1 || sy>map\_b)
4. cout<<"起点不在迷宫里面";
5. **else** **if**(map\_bool[sx][sy]==1) cout<<"起点为墙";
6. **else** **if**(tx<1 || tx>map\_a || ty<1 || ty>map\_b)
7. cout<<"终点不在迷宫里面";
8. **else** **if**(map\_bool[tx][ty]==1) cout<<"终点为墙";
9. **else** **return** **true**;
10. }

### 3.2.2 print()输出路径函数

原本的路径在path栈之中，如果按照path栈弹出的内容输出的话，顺序是从终点到起点，所以再定义一个path\_out栈，把path栈的内容压入path\_out栈中在输出就是从起点到终点的顺序了。

1. **void** print()
2. {
3. cout<<"迷宫路径:"<<endl;
4. //path栈里的路径直接输出的话是从终点到起点
5. //所以要先把path栈里的内容先压到path\_out栈之中
6. Stack<P> path\_out;
7. **while**(!path.empty())
8. {
9. path\_out.push(path.top());//path\_out压栈path.top()元素
10. **if**(!path.empty()) path.pop();//path出栈
11. }
12. P temp=path\_out.top();
13. **int** cnt=1;
14. cout<<"<"<<temp.first<<","<<temp.second<<">";
15. path\_out.pop();
16. **while**(!path\_out.empty())//path\_out栈不为空，一直输出
17. {
18. **if**(cnt%4==0) cout<<endl;
19. cnt++;
20. temp=path\_out.top();
21. cout<<" ---> <"<<temp.first<<","<<temp.second<<">";
22. **if**(!path\_out.empty()) path\_out.pop();
23. }
24. }

### 3.2.3 dfs()深搜函数的实现

起始位置为坐标(sx,sy)，当前的位置为(x,y)，如果(x,y)不是终点位置(tx,ty)，则在上下左右四个方向中找不越界和没有墙的坐标，结点入站、标记并搜索；如果搜索到尽头还没有则不断返回搜索的上一层，消除标记、结点出栈，并向其它几个方向搜索。

如果搜索到了，则flagg变为1，根据flagg的值，递归不再搜索，输出结果。

TIPS:由于是搜索到了终点就直接输出，所以输出的路径不一定是最短路径!

1. **bool** flagg=0;
2. **void** dfs(**int** x,**int** y,**int** tx,**int** ty)//当前坐标(x,y)，终点坐标(tx,ty)
3. {
4. **if**(x==tx && y==ty)
5. {
6. print();
7. flagg=1;
8. }
9. **else** **for**(**int** i=1;i<=4;i++)
10. {
11. **int** nx=x+dx[i],ny=y+dy[i];//走的下一步坐标(nx,ny)
12. **if**(nx>=1 && nx<=map\_a && ny>=1 && ny<=map\_b && map\_bool[nx][ny]==0)
13. //判断，在不越界和没有墙的时候继续进行
14. {
15. **if**(!flagg) path.push(P(nx,ny));
16. //如果没有到达终点的话继续压栈
17. map\_bool[nx][ny]=1;//标记、染色走过的路径
18. dfs(nx,ny,tx,ty);//继续深搜
19. **if**(!flagg)//如果没有到达终点
20. {
21. map\_bool[nx][ny]=0;//回溯，消除走过的路径的标记
22. path.pop();//出栈
23. }
24. }
25. }
26. **return**;
27. }

### 3.2.4 主函数的实现

主要是实现了输出字符地图与人机交互。

1. **int** main()
2. {
3. cout<<"输入地图的长和宽:"<<endl;
4. cin>>map\_a>>map\_b;
5. cout<<"输入地图,用0代表路、1代表墙"<<endl;
6. **for**(**int** i=1;i<=map\_a;i++)
7. **for**(**int** j=1;j<=map\_b;j++) cin>>map\_bool[i][j];
8. //转化为字符数组地图
9. **for**(**int** i=0;i<=map\_a+1;i++)
10. **for**(**int** j=0;j<=map\_b+1;j++)
11. {
12. **if**(i==0||i==map\_a+1||j==0||j==map\_b+1) mapp[i][j]='#';
13. **else** **if**(map\_bool[i][j]==1) mapp[i][j]='#';
14. **else** **if**(map\_bool[i][j]==0) mapp[i][j]='0';
15. }
16. //输出地图
17. cout<<"迷宫地图:"<<endl;
18. printf("\t");
19. **for**(**int** j=0;j<=map\_b+1;j++) printf("%d列\t",j);
20. printf("\n");
21. **for**(**int** i=0;i<=map\_a+1;i++)
22. {
23. printf("%d行\t",i);
24. **for**(**int** j=0;j<=map\_b+1;j++) printf("%c\t",mapp[i][j]);
25. printf("\n");
26. }
27. printf("\n");
29. **int** sx,sy,tx,ty;
30. cout<<"输入起点（sx,sy）";
31. cin>>sx>>sy;
32. cout<<endl;
33. cout<<"输入终点（tx,ty）";
34. cin>>tx>>ty;
35. cout<<endl;
36. **if**(isValid(sx,sy,tx,ty)==0) **return** 0;
37. path.push(P(sx,sy));
38. map\_bool[sx][sy]=1;
39. dfs(sx,sy,tx,ty);
40. if(flagg==0) cout<<"没有路径！"<<endl;
41. **return** 0;
42. }

# 4 测试

## 4.1 功能测试

### 4.1.1测试1

**测试用例**：

7 7

0 0 1 1 1 1 1

1 0 0 0 1 0 1

0 0 1 0 1 1 1

1 0 0 1 1 1 1

0 0 1 1 0 0 0

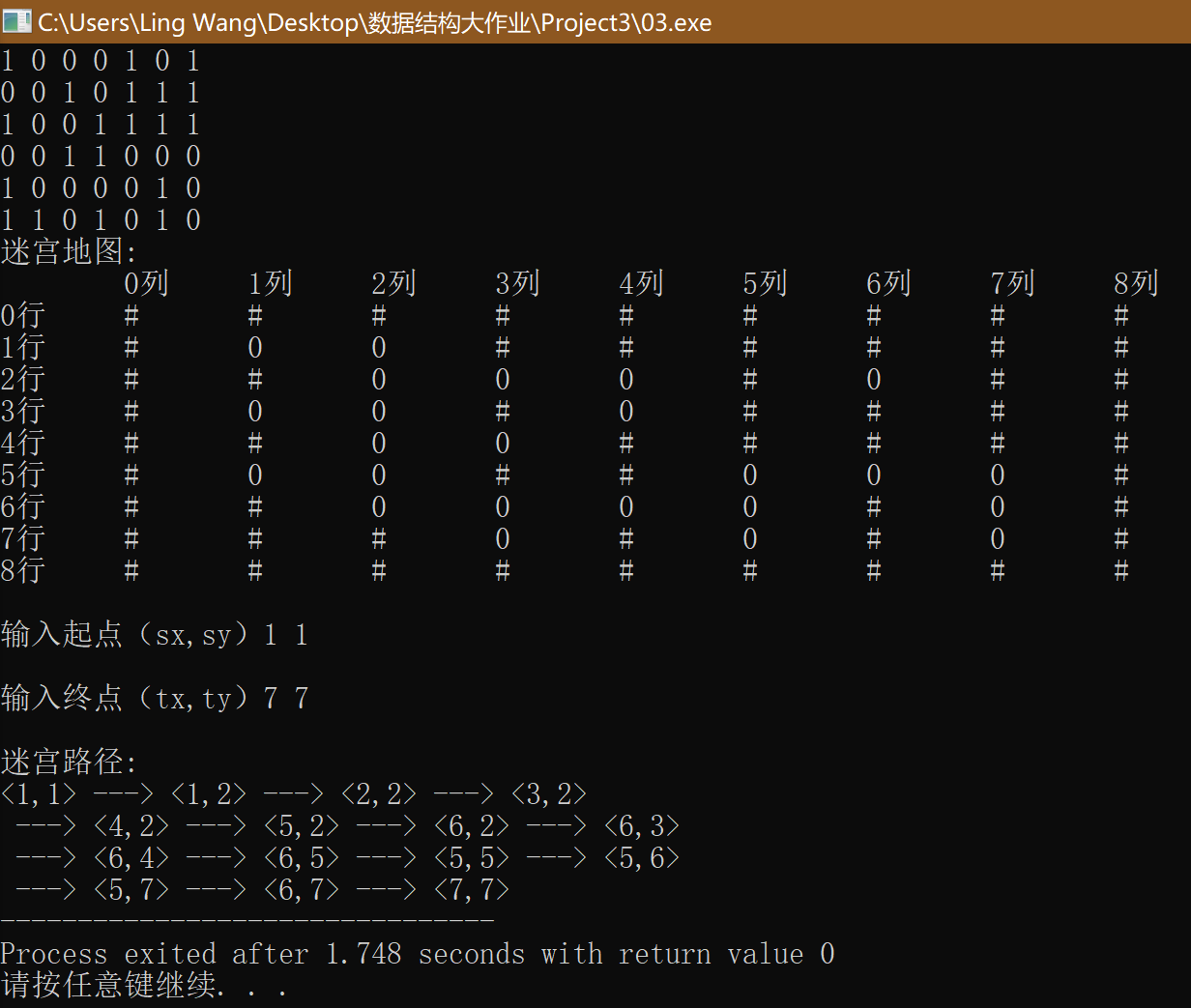
1 0 0 0 0 1 0

1 1 0 1 0 1 0

1 1

7 7

**实验结果**



### 4.1.2 测试2

**测试用例：**

7 7

0 0 1 1 1 1 1

1 0 0 0 1 0 1

0 0 1 0 1 1 1

1 0 0 1 1 1 1

0 0 1 1 0 0 0

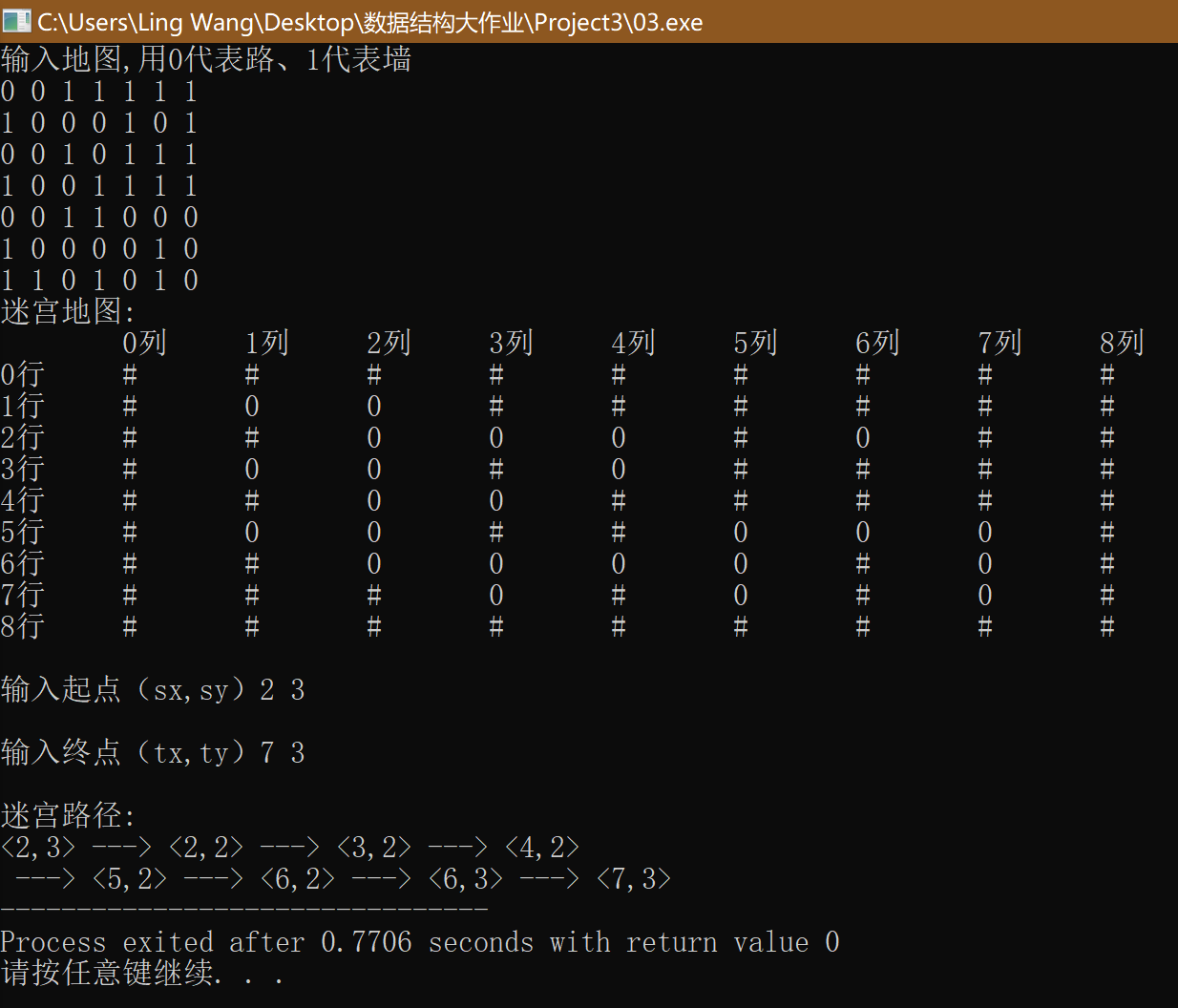
1 0 0 0 0 1 0

1 1 0 1 0 1 0

2 3

7 3

**实验结果：**



### 4.1.3 测试3

**测试用例：**

8 9

0 0 1 1 1 1 1 0 1

0 0 0 0 0 0 1 0 1

0 0 1 0 1 0 0 0 0

1 0 0 1 1 1 1 1 0

0 0 1 1 0 0 0 1 0

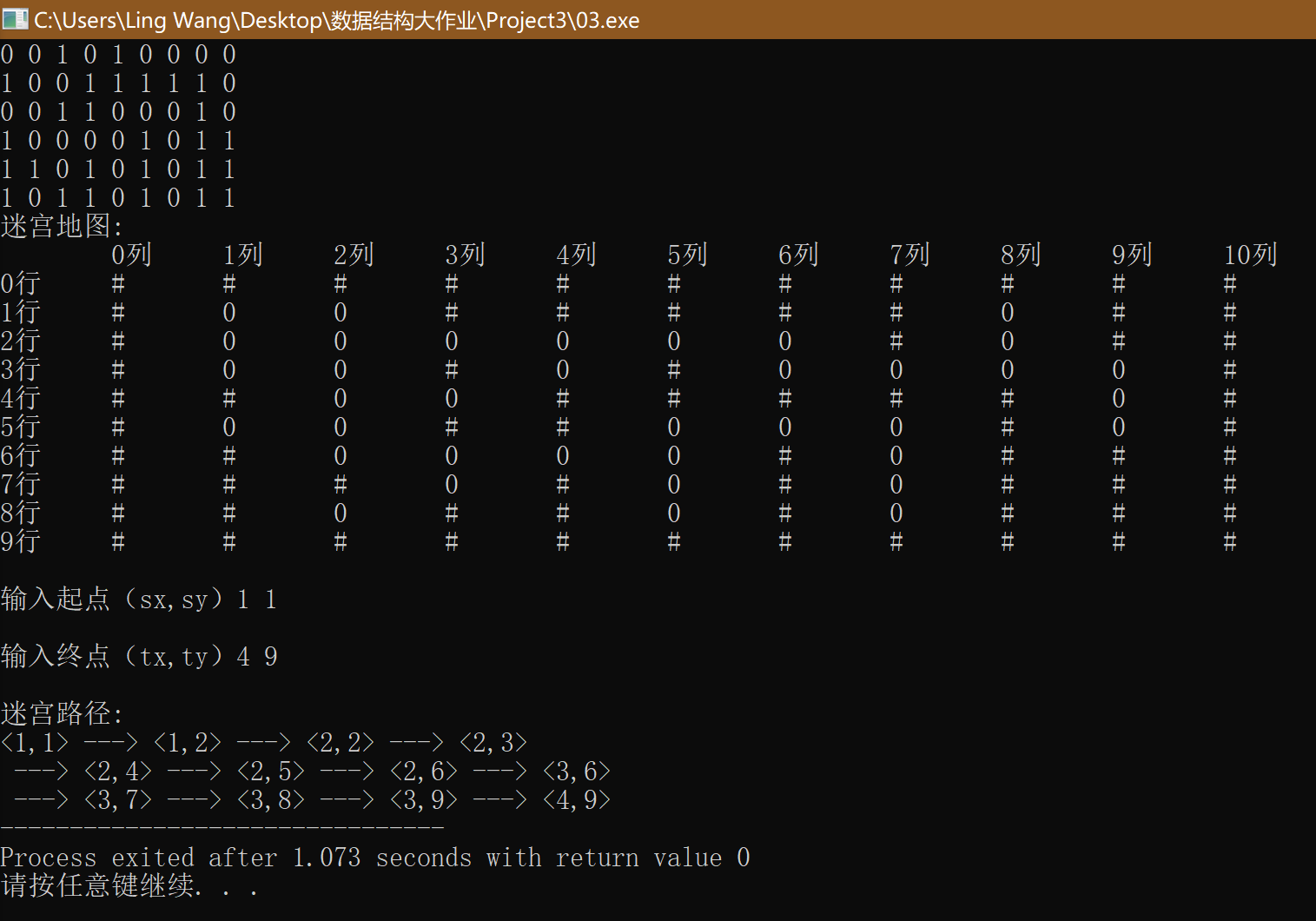
1 0 0 0 0 1 0 1 1

1 1 0 1 0 1 0 1 1

1 0 1 1 0 1 0 1 1

1 1

4 9

**实验结果：** 

## 4.2 边界测试

### 4.2.1 没有路径

**测试用例：**

5 5

0 1 1 1 1

1 1 1 1 1

1 1 1 1 1

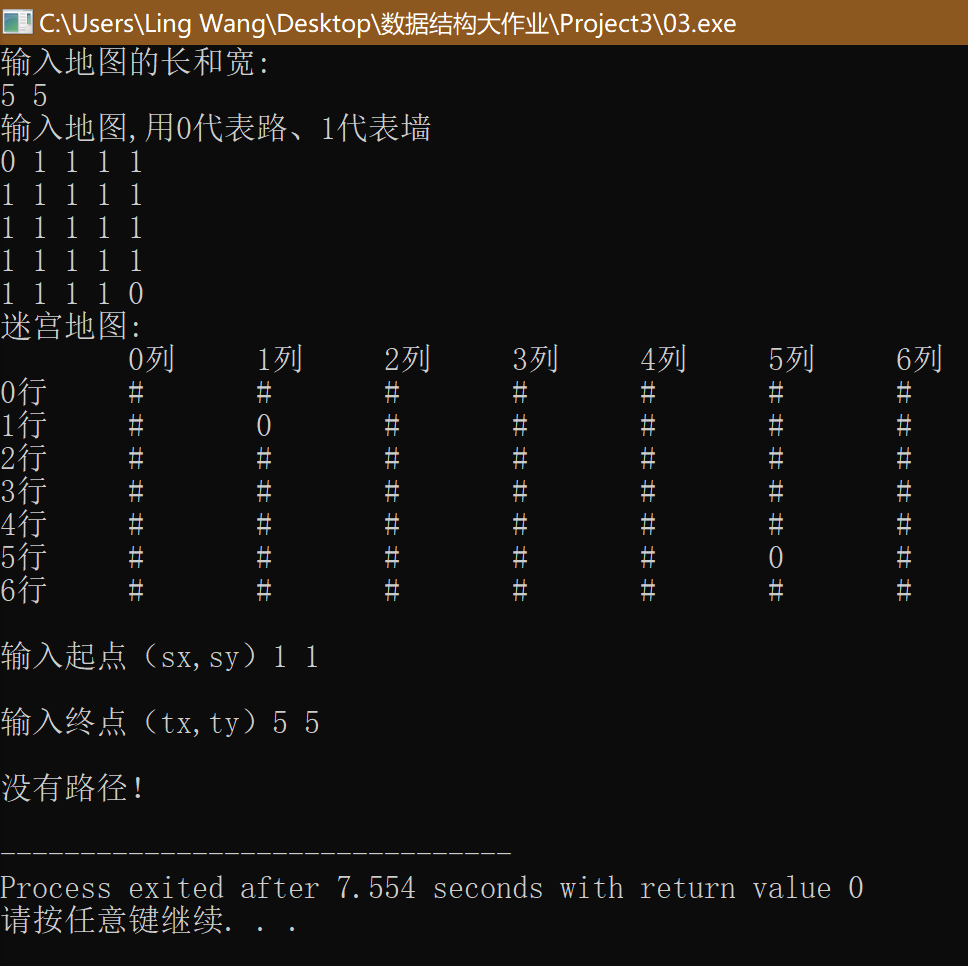
1 1 1 1 1

1 1 1 1 0

1 1

5 5

**实验结果：**



### 4.2.2 入口、出口为墙

**测试用例：**

5 5

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

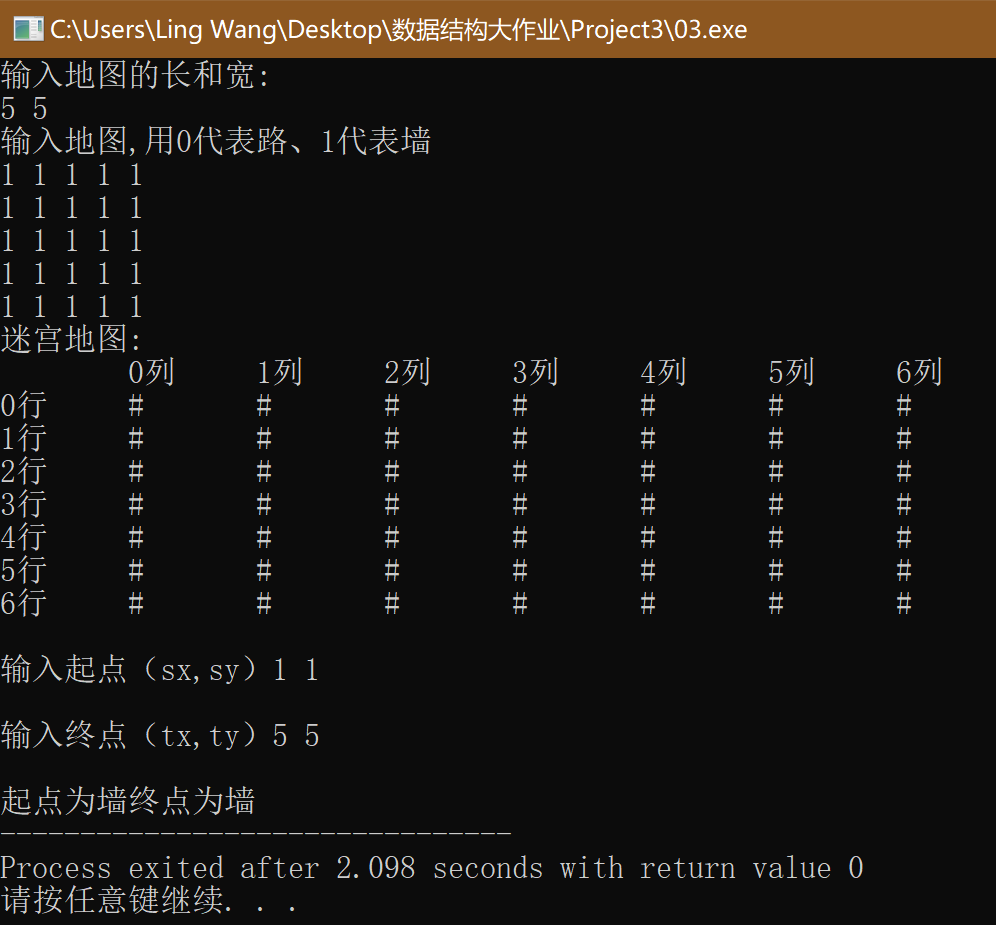
1 1 1 1 1

1 1 1 1 1

1 1

5 5

**实验结果：**



### 4.2.3 入口、出口越界

**测试用例：**

5 5

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

0 0

100 100

**实验结果：**

