数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 李佳诺

学 号： 1751188

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 4

1.1背景分析 4

1.2 功能分析 4

2 设计 4

2.1 算法设计 4

2.2 数据结构设计 5

2.3 成员与操作设计 5

地图结点（Position） 5

方向（direction） 5

路径保存（Route） 5

出口保存（ExitPostion） 5

地图保存（Maze） 5

3 实现 6

3.1 地图（Maze） 6

3.2 搜索函数（run） 7

深度优先搜索流程图 7

3.3 主函数main() 9

主函数流程图 9

4 测试 11

4.1 功能性测试 11

4.2 出错性测试 13

# 1 分析

## 1.1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

迷宫问题的求解过程可以采用回溯法即在一定的约束条件下试探地搜索前进，若前进中受阻，则及时回头纠正错误另择通路继续搜索的方法。从入口出发，按某一方向向前探索，若能走通，即某处可达，则到达新点，否则探索下一个方向；若所有的方向均没有通路，则沿原路返回前一点，换下一个方向再继续试探，直到所有可能的道路都探索到，或找到一条通路，或无路可走又返回入口点。在求解过程中，为了保证在达到某一个点后不能向前继续行走时，能正确返回前一个以便从下一个方向向前试探，则需要在试探过程中保存所能够达到的每个点的下标以及该点前进的方向，当找到出口时试探过程就结束了。

# 2 设计

## 2.1 算法设计

我们采用思路清晰明了的递归算法。对于本题来说，首先要确定的是如何找到出口，在这里，我们采用的是深度优先搜索，即只要下一步可以走就一直走，如果不能走就返回上一步。主要运用的是回溯的思想。

## 2.2 数据结构设计

回溯的思想的实现依赖于堆栈这一数据结构，用于保存上一搜索过的结点信息。然而考虑到本题之后要输出路径信息，是从头至尾的，由于栈“先进后出”的特性，无法快速输出。于是借助了STL里的vector（可以直接在尾部push），既能实现尾部删改的堆栈，又可以快速从头输出结果。

## 2.3 成员与操作设计

### 地图结点（Position）

//位置结点

struct Position

{

int x; //x表横坐标

int y; //y表纵坐标

};

### 方向（direction）

//定义四个方向

Position direction[4] =

{

{0,1},{0,-1},{-1,0},{1,0}

};

### 路径保存（Route）

vector<Position> Route; //用来存储通关路径

### 出口保存（ExitPostion）

Position ExitPostion = { 5,5 }; //出口坐标

### 地图保存（Maze）

char Maze[8][8] //地图

# 3 实现

## 3.1 地图（Maze）

在本题中支持用户自己输入任意7×7的地图,由char类型的二维数组Maze保存。其核心代码如下：

cout << "请输入地图（7\*7）：" << endl;

char temp;

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

{

cout << "请输入第"<<i<<"行：";

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

{

cin >> temp;

Maze[i][j] = temp;

}

}

## 3.2 搜索函数（run）

### 深度优先搜索流程图

搜索相邻的位置

开 始

删除结点

current==exit?

下一步可走？

成 功

是否搜索完毕？

结束

是

是

否

是

否

否

失 败

本函数的主要功能就是从出发点搜索通往终点的路径，运用了深度优先搜索算法。核心代码如下：

void run(int x, int y)

{

if (ExitPostion.x == x&&ExitPostion.y == y)

{

Isfound = true;

Maze[x][y] = 'X';

Route.push\_back(ExitPostion);

return;

}

for (int i = 0; i < 4; i++)//四个方向

{

//如果下一步可以走的话将当前位置放入Route，并标记地图

if (Maze[x + direction[i].x][y + direction[i].y] == '0')

{

Position temp = { x,y };

Route.push\_back(temp);

Maze[x][y] = 'X';

run(x + direction[i].x, y + direction[i].y);

if (Isfound) { return; }

//如果这条路不通，则在路径中删除，并返回上一步

Route.erase(Route.end() - 1);

Maze[x][y] = '0';

}

}

}

## 3.3 主函数main()

### 主函数流程图

开始

失败

搜索路径

输入起始点

输入地图

Isfound？?

打印路径

结束

主函数通过与用户交互，获得地图和起始点信息，然后用搜索函数进行搜索，若成功则打印路径信息，若失败则返回报错信息。其核心代码如下：

cout << "请输入地图（7\*7）：" << endl;

char temp;

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

{

cout << "请输入第"<<i<<"行：";

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

{

cin >> temp;

Maze[i][j] = temp;

}

}

Position pos = { 0,0 };

cout << "请输入起点(x,y),以空格隔开：";

cin >> pos.x >> pos.y;

cout << "请输入终点(x,y),以空格隔开：";

cin >> pos.x >> pos.y;

run(1, 1);

if (Isfound)

{

cout << "迷宫地图:\n\t";

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

cout << i << "列" << "\t\t";

cout << endl;

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

{

cout << i << "行" << "\t";

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

{

cout << Maze[i][j] << "\t\t";

}

cout << endl << endl;

}

cout << "迷宫路径：\n\t";

for (auto iter = Route.begin(); iter != Route.end() - 1; iter++)

cout << "<" << iter->x << "," << iter->y << "> " << "---> ";

cout << "<" << (Route.end() - 1)->x << "," << (Route.end() - 1)->y << ">" << endl;

}

else cout << "Not Found!" << endl;

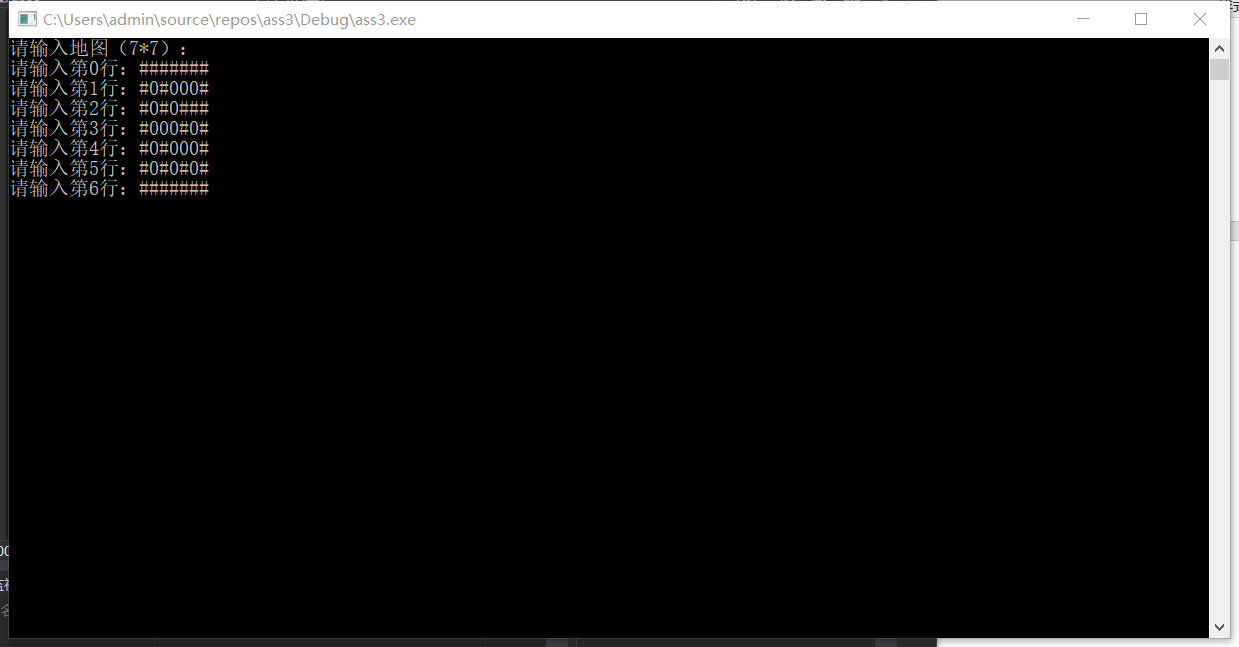
# 4 测试

## 4.1 功能性测试

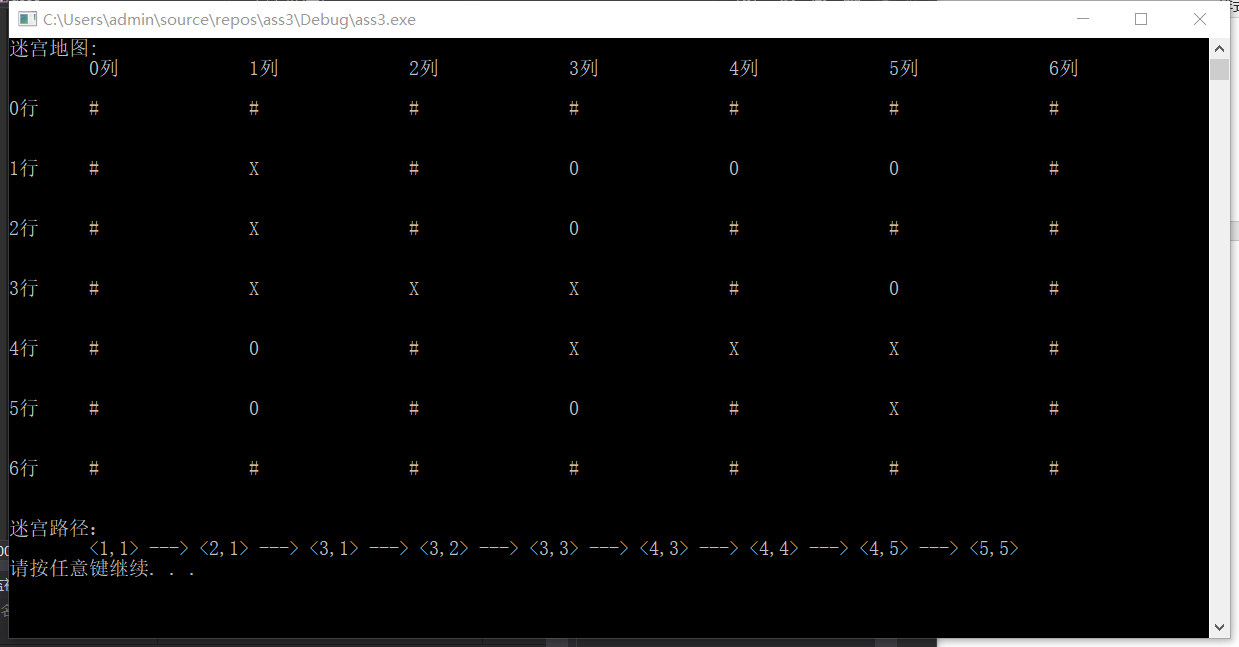
测试该程序能否正确输出相应地图的路径。实验结果如下：

地图1：

起点（1，1），终点（5，5）

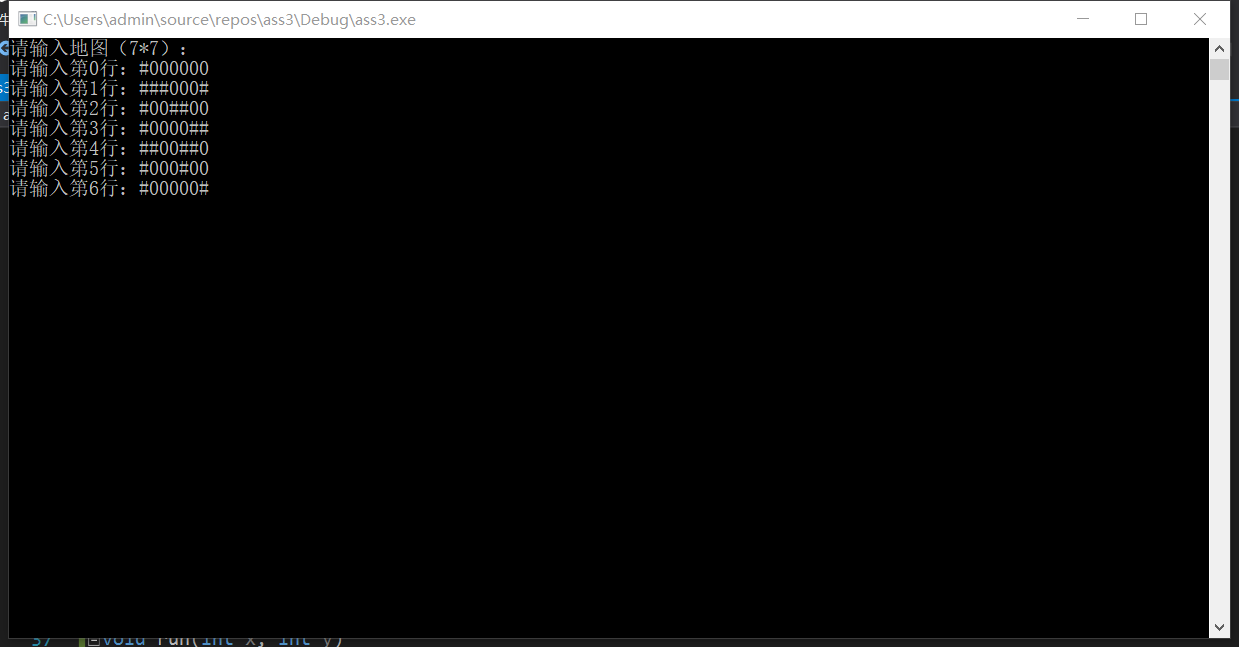


结果1：

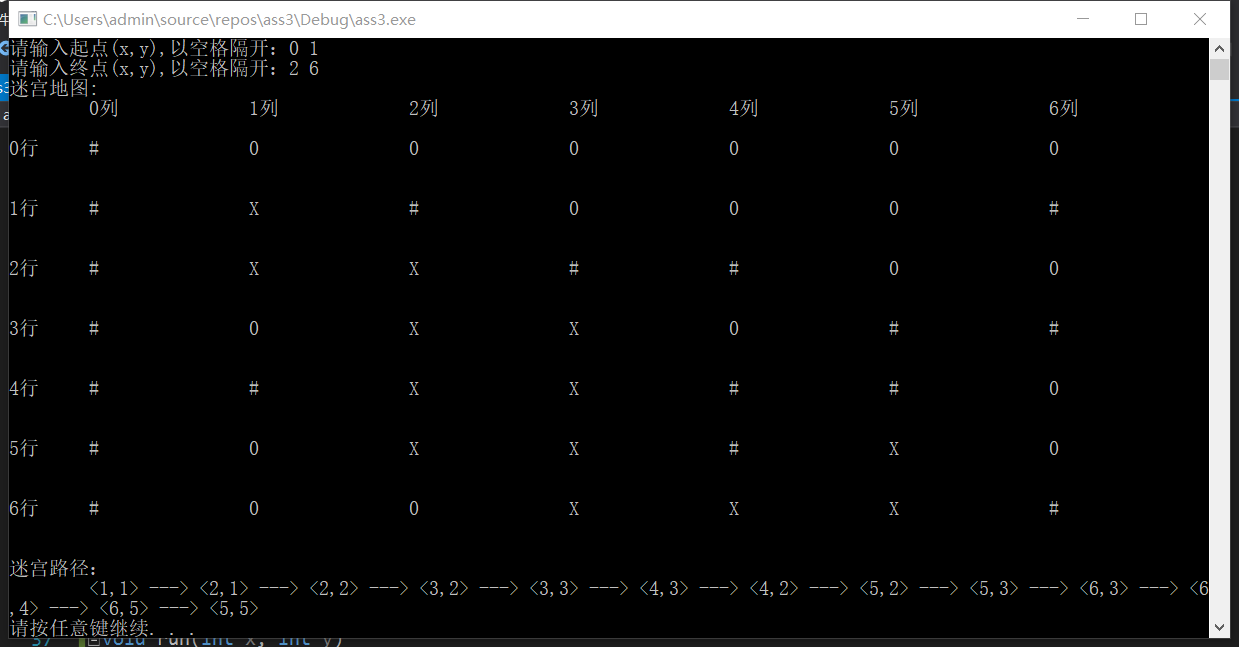
**

地图2：

起点（0，1），终点（2，6）

**

结果2：

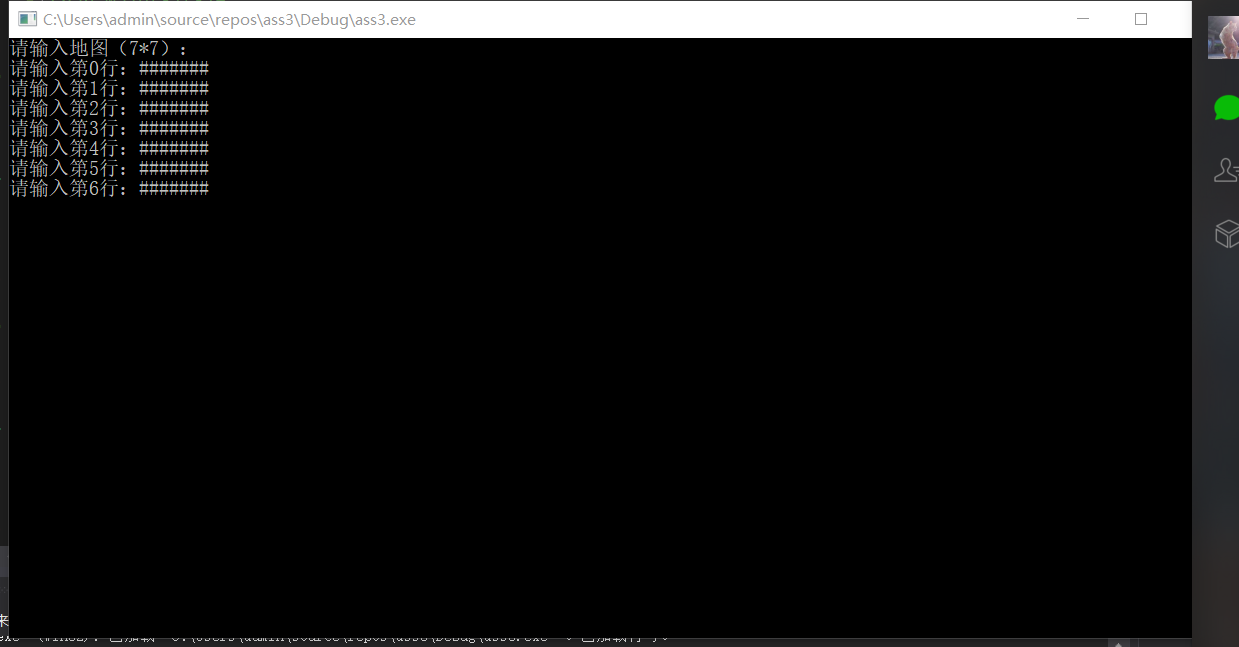
**

## 4.2 出错性测试

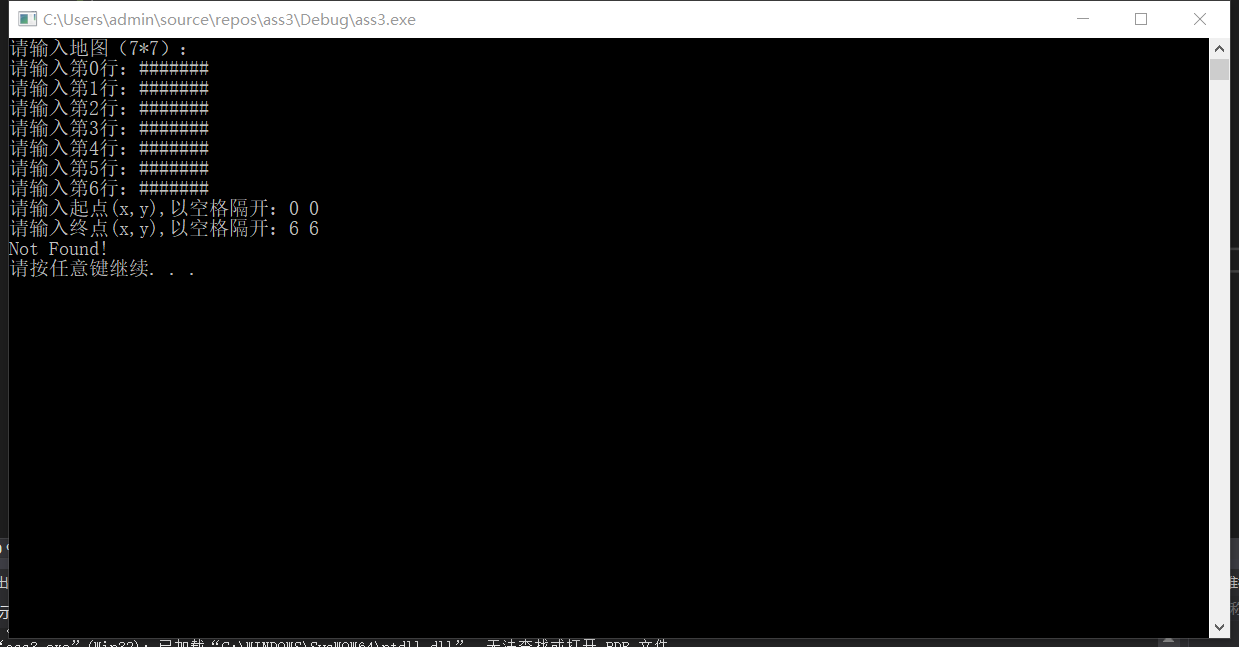
当程序遍历完所有路径后都没有找到出口，则返回信息，提示用户没找到出口

地图3：

起点（0，0），终点（6，6）

**

结果3：

**