**项目说明文档**

**数据结构课程设计**

**——勇闯迷宫游戏**

作 者 姓 名： 张梓瀚

学 号： 2051943

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 项目分析 - 3 -](#_Toc121209021)

[1.1项目背景 - 3 -](#_Toc121209022)

[1.2 项目要求 - 3 -](#_Toc121209023)

[1.2.1 功能要求 - 3 -](#_Toc121209024)

[1.2.2 输入格式 - 3 -](#_Toc121209025)

[1.2.3 输出格式 - 4 -](#_Toc121209026)

[2 项目设计 - 4 -](#_Toc121209027)

[2.1 数据结构设计 - 4 -](#_Toc121209028)

[2.2 类设计 - 4 -](#_Toc121209029)

[2.2.1 链表类（List） - 4 -](#_Toc121209030)

[2.2.2 栈类（MyStack） - 4 -](#_Toc121209031)

[2.3 勇闯迷宫游戏算法 - 5 -](#_Toc121209032)

[2.3.1 主要思想 - 5 -](#_Toc121209033)

[2.3.2代码 - 5 -](#_Toc121209034)

[3 项目测试 - 7 -](#_Toc121209035)

[3.1 一般情况 - 7 -](#_Toc121209036)

[3.2 无法到达终点的情况 - 8 -](#_Toc121209037)

# 1 项目分析

## 1.1项目背景

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。

事实上，对于迷宫问题进行求解在现实生活中有着许多的实际应用场景。例如：日常使用的百度地图就需要通过寻路算法来为我们指引正确的道路。而迷宫问题正是其问题的一个简化，能够快速有效的解决迷宫问题能够为我们未来研究更加深入的算法提供最为基础的理论依据。

## 1.2 项目要求

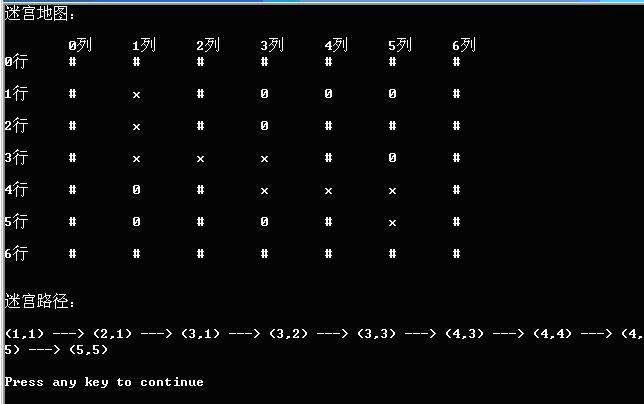
### 1.2.1 功能要求

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。

### 1.2.2 输入格式

（无）

### 1.2.3 输出格式



# 2 项目设计

## 2.1 数据结构设计

本题目选用栈作为搜索过程中的存储结构，每次取栈顶点，检查其周围各个方向的点是否未访问过且可访问（不是障碍或边界），如是，则将对应点入栈；如四个方向的点都不满足要求，则说明当前栈顶点已无法到达终点，故将其出栈。因此，需要一个存放坐标点的栈和一个记录各点是否已访问过的bool类型数组。

## 2.2 类设计

经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（ListNode）与链表类（List），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。

为了实现代码的复用性，本系统利用了上题中实现过的**链表结点类**与**链表类**，以List类为基础实现了一个链式栈。

### 2.2.1 链表类（List）

链表的结点中存储的数据有链表数据data和后继结点next。为提高代码的泛用性，本题中ListNode类采用模板类实现。

### 2.2.2 栈类（MyStack）

//基于List类实现的链式栈

template<typename T>

class myStack : private List<T> {

public:

myStack();

//向链表头部（栈顶）插入结点

void push(const T& data);

//返回第一个结点的数据

const T top();

//删除第一个结点

void pop();

//判断栈是否为空

bool empty();

//获取栈的大小

int size();

};

## 2.3 勇闯迷宫游戏算法

### 2.3.1 主要思想

本题目选用深度优先搜索（DFS）的方法，先将起点进栈，每次取栈顶点，检查其周围各个方向的点是否未访问过且可访问（不是障碍或边界），如是，则将对应点入栈；如四个方向的点都不满足要求，则说明当前栈顶点已无法到达终点，故将其出栈。若栈不为空则重复以上过程，直到抵达终点。

### 2.3.2代码

bool vis[MAP\_SIZE][MAP\_SIZE];

//坐标点类

struct Point

{

int x, y;

bool operator == (const Point& rhs)

{

return x == rhs.x && y == rhs.y;

}

Point operator + (Point& rhs)

{

Point pt = { x + rhs.x,y + rhs.y };

return pt;

}

};

Point operator + (const Point& p1, const Point& p2)

{

Point pt = { p1.x + p2.x,p1.y + p2.y };

return pt;

}

bool isReachable(Point pnt)

{

return Map[pnt.x][pnt.y] == 'x';

}

const Point step[4] = { {1,0},{0,1},{-1,0},{0,-1} };

myStack<Point>stk;

//深度优先搜索

bool search(Point start, Point end)

{

bool\*\* visited = new bool\* [MAP\_SIZE + 1];

for (int i = 0; i <= MAP\_SIZE; i++)

{

visited[i] = new bool[MAP\_SIZE + 1];

memset(visited[i], 0, (MAP\_SIZE + 1) \* sizeof(bool));

}

//起点入栈

stk.push(start);

Point current;

while (!stk.empty())

{

current = stk.top();

if (current == end)

break;

bool no\_way = true;

//判断四个方向是否能到达，如能到达则将该方向上的点入栈

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

{

Point next = current + step[i];

if (isReachable(next) && !visited[next.x][next.y])

{

stk.push(next);

visited[next.x][next.y] = true;

no\_way = false;

break;

}

}

//若没有可达的方向，说明当前点没有前进可能，将其弹栈

if (no\_way)

stk.pop();

}

for (int i = 0; i <= MAP\_SIZE; i++)

delete[] visited[i];

delete[] visited;

if (stk.empty())

return false;

return true;

}

# 3 项目测试

## 3.1 一般情况

测试用例：

{'#','#','#','#','#','#','#','#'},

{'#','x','0','x','0','0','x','#'},

{'#','x','0','x','x','0','x','#'},

{'#','x','0','x','x','x','x','#'},

{'#','x','0','x','0','x','0','#'},

{'#','x','x','x','0','x','x','#'},

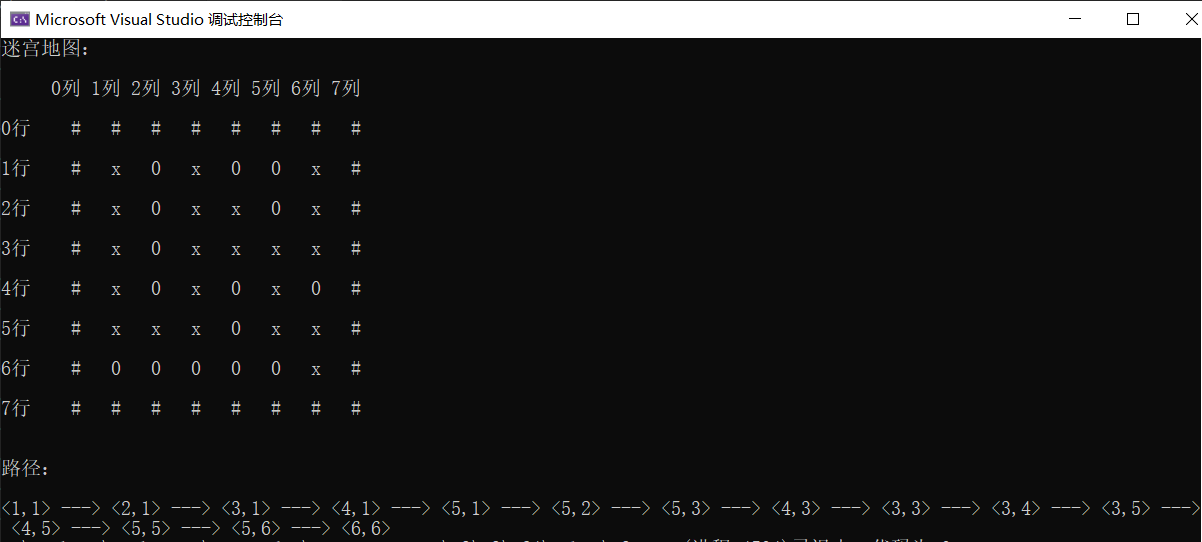
{'#','0','0','0','0','0','x','#'},

{'#','#','#','#','#','#','#','#'}

预期结果：

<1,1> ---> <2,1> ---> <3,1> ---> <4,1> ---> <5,1> ---> <5,2> ---> <5,3> ---> <4,3> ---> <3,3> ---> <3,4> ---> <3,5> ---> <4,5> ---> <5,5> ---> <5,6> ---> <6,6>

实验结果：



## 3.2 无法到达终点的情况

测试用例：

{'#','#','#','#','#','#','#','#'},

{'#','x','0','x','0','0','x','#'},

{'#','x','0','x','x','0','x','#'},

{'#','x','0','x','x','x','x','#'},

{'#','x','0','x','0','x','0','#'},

{'#','x','x','x','0','x','0','#'},

{'#','0','0','0','0','0','x','#'},

{'#','#','#','#','#','#','#','#'}

预期结果：没有可达路径

实验结果：

