项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 王星洲

学 号： 1652977

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](#_Toc534066812)

[1.1 背景分析 1](#_Toc534066813)

[1.2 功能分析 1](#_Toc534066814)

[2 设计 1](#_Toc534066815)

[2.1 数据结构设计 1](#_Toc534066816)

[2.2 流程设计 2](#_Toc534066817)

[3 实现 2](#_Toc534066818)

[3.1 初始化及声明 2](#_Toc534066819)

[3.1.1 初始化核心代码 2](#_Toc534066820)

[3.1.2 初始化说明 3](#_Toc534066821)

[3.2 搜索路径功能的实现 4](#_Toc534066822)

[3.2.1 搜索路径功能流程图 4](#_Toc534066823)

[3.2.2 搜索路径功能核心代码 5](#_Toc534066824)

[3.2.3 搜索路径功能说明 6](#_Toc534066825)

[3.3 输出结果功能的实现 7](#_Toc534066826)

[3.3.1 输出结果功能流程图 7](#_Toc534066827)

[3.3.2 输出结果功能核心代码 8](#_Toc534066828)

[3.3.3 输出结果功能说明 8](#_Toc534066829)

[3.4 打印地图功能的实现 9](#_Toc534066830)

[3.4.1 打印地图流程图 9](#_Toc534066831)

[3.4.2 打印地图功能核心代码 10](#_Toc534066832)

[4 测试 10](#_Toc534066833)

[4.1 功能测试 10](#_Toc534066834)

[4.1.1 一般功能测试 10](#_Toc534066835)

[5 亮点 11](#_Toc534066836)

[5.1 模块化程度高 11](#_Toc534066837)

[5.2 使用了栈作为存储，避免递归算法 12](#_Toc534066838)

# 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 数据结构设计

如上述功能分析描述，本题需要使用回溯法解决问题，然而使用回溯法有很多方式，比如递归等，但考虑到递归的效率较低，成本较高，我们采用一个栈stack来存储在试探过程中所走过的路径，一旦需要回退，可以从栈中取得刚才走过位置的坐标和前进方向。栈中需要保存一系列三元组以记录这些信息。这个三元组我们使用结构struct来实现，为了避免走重复的路，我们使用了一个数组mark[][]来记录走过的路径。最后，为了按照题目要求输出结果，我们使用maze[][]来判断每一个点对应的输出是x # 或 0，并且使用两个数组vector实现将栈中的结构按照要求倒序输出。

## 2.2 流程设计

本程序没有人机交互内容，在计算机内定义一个题目中的初始数组

int maze[7][7] =

{

1,1,1,1,1,1,1,

1,0,1,0,0,0,1,

1,0,1,0,1,1,1,

1,0,0,0,1,0,1,

1,0,1,0,0,0,1,

1,0,1,0,1,0,1,

1,1,1,1,1,1,1

};

接下来经过回溯算法分析走的路径，并且将对应的路径存入栈中，将maze中对应位置赋值为2.接下来对于maze中0/1/2输出不同的符号。然后取出栈中的值，按照格式倒序输出即可。

# 3 实现

## 3.1 初始化及声明

### 3.1.1 初始化核心代码

int maze[7][7] = //迷宫地图 0为路径 1为围墙

{

1,1,1,1,1,1,1,

1,0,1,0,0,0,1,

1,0,1,0,1,1,1,

1,0,0,0,1,0,1,

1,0,1,0,0,0,1,

1,0,1,0,1,0,1,

1,1,1,1,1,1,1

};

int mark[7][7] = { 0 }; //标记数组，记录某处是否已经经过

struct path { //三元组

int x;

int y;

int dir;

};

struct offsets{ //方向

int a, b;

};

void setOffsets(offsets move[]) //设置每一步前进方向的四种可能

{

move[0].a = 0; move[0].b = 1;//上

move[1].a = 0; move[1].b = -1;//下

move[2].a = 1; move[2].b = 0;//右

move[3].a = -1; move[3].b = 0;//左

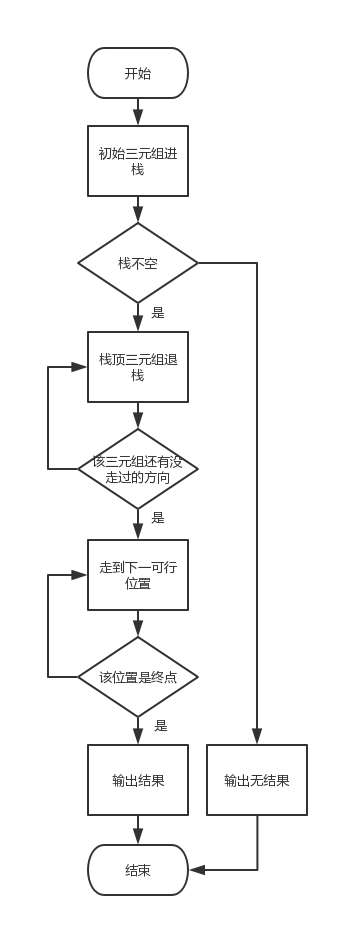
}

### 3.1.2 初始化说明

本题需要的准备量比较大，首先是根据题目录入初始的地图，接下来创建一个标记数组，该数组在之后的判断路径重复上将会起作用，再创建一个之后栈中需要储存的数据类型——三元组，储存位置x，y坐标和前进方向dir。再将每一步可以走的四种方向进行赋值，则初始化工作完成。

## 3.2 搜索路径功能的实现

### 3.2.1 搜索路径功能流程图



### 3.2.2 搜索路径功能核心代码

void findPath(int m, int p, offsets move[]) //搜索路径函数

{

int i, j, d;//当前x，y，dir

int g, h;//改变后x，y

mark[1][1] = 1;//入口

stack<path> st;//工作栈及工作元

path tmp;

tmp.x = 1;

tmp.y = 1;

tmp.dir = 0;

st.push(tmp);//初始坐标方向三元组进栈

while (st.empty() == false)//栈不空，持续走下去

{

tmp = st.top();

st.pop();//退栈

i = tmp.x;

j = tmp.y;

d = tmp.dir;

while (d < 4)//还有没进行完的移动

{

g = i + move[d].a;

h = j + move[d].b;//下一位置

if (g == m && h == p) {//找到出口

tmp.x = i;

tmp.y = j;

st.push(tmp);//进栈

result(st);

return;

}

if (maze[g][h] == 0 && mark[g][h] == 0)//有新的位置可通

{

mark[g][h] = 1;//标记为访问过

tmp.x = i;

tmp.y = j;

tmp.dir = d;

st.push(tmp);//进栈

i = g;//开始试探

j = h;

d = 0;

}

else

{

d++;//试探下一个方向

}

}

cout << "no path in Maze" << endl;

}

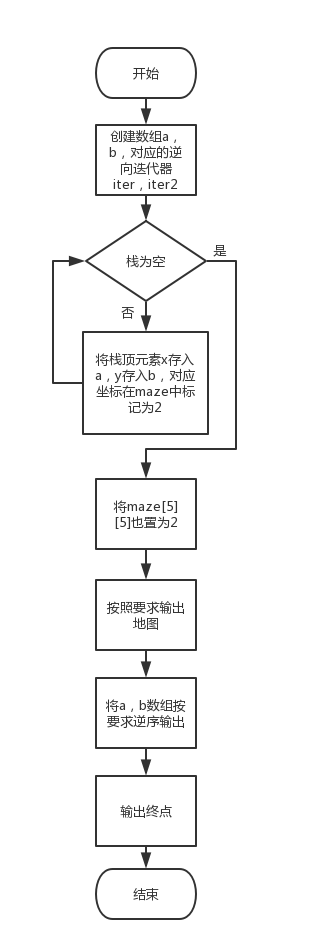
}

### 3.2.3 搜索路径功能说明

搜先创建一个栈和初始的三元组，这个三元组存储起点x，y及0方向，接下来将其压入栈中，接下来从栈中取出初始三元组，并且开始从初始位置探寻四个方向查看是否路径，如果存在路径，那么将其三元组入栈，将对应位置的mark置为1，再向后查询，如果有某处四个方向全都搜索过或被墙挡住，即没有落脚点，那么就依次将三元组出栈，直到发现某点可以更换方向继续前进。每次到达一个新点时，与终点比较，如果已经到达终点，那么调用输出结果的函数，如果没有到达终点，那么继续搜索。

## 3.3 输出结果功能的实现

### 3.3.1 输出结果功能流程图



### 3.3.2 输出结果功能核心代码

void result(stack<path> st) //根据栈中内容输出结果

{

vector<int> a, b;

while (st.empty() == false)

{

path t = st.top();

maze[t.x][t.y] = 2;

a.push\_back(t.x);

b.push\_back(t.y);

st.pop();

}

maze[5][5] = 2;

printMap(maze);

cout << endl << "迷宫路径：" << endl << endl;

auto iter = a.rbegin();

auto iter2 = b.rbegin();

for (iter; iter != a.rend(); iter++, iter2++)

{

cout << "<" << \*iter << "," << \*iter2 << "> ---> ";

}

cout << "<5,5>" << endl;

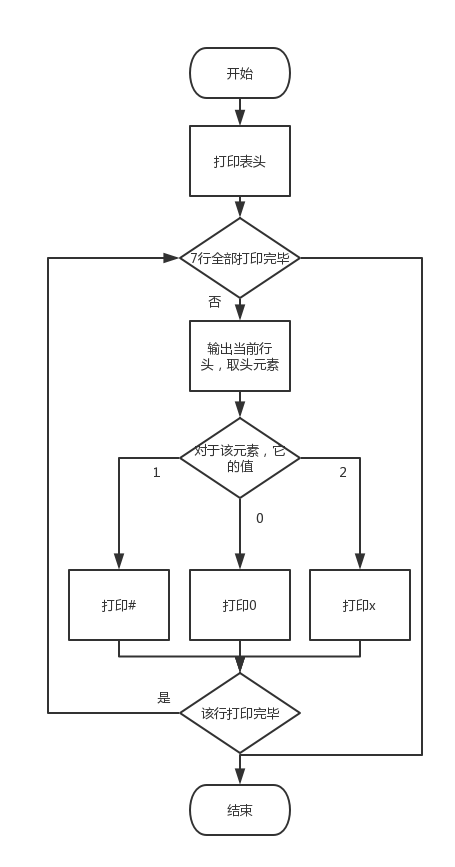
}

### 3.3.3 输出结果功能说明

输出结果功能首先要将搜索路径功能得到的数组传入，这个数组是除了终点以外所有经过路径的点的三元组集，接下来，我们栈中的所有三元组的坐标在maze内标为2，并且把终点同样标记为2，即为走过的路径。而最后迷宫路径的输出需要从起点到终点，而栈中的元素的输出顺序是后进先出的，所以我们需要将栈中的元素逆栈顺序输出，所以我采用两个数组，a和b，使用迭代器，逆序输出栈中的元素，最后将终点5，5输出，则输出完毕。

## 3.4 打印地图功能的实现

### 3.4.1 打印地图流程图



### 3.4.2 打印地图功能核心代码

void printMap(int maze[7][7]) //打印地图

{

cout << "迷宫地图：" << endl << endl;

cout << '\t' << "0列" << '\t' << "1列" << '\t' << "2列" << '\t' << "3列" << '\t' << "4列" << '\t' << "5列" << '\t' << "6列" << endl;

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

{

cout << j << "行" << '\t';

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

{

if (maze[j][i] == 1)

{

cout << "#" << '\t';

}

else if(maze[j][i] == 2)

{

cout << "x" << '\t';

}

else

{

cout << "0" << '\t';

}

}

cout << endl << endl;

}

}

# 4 测试

## 4.1 功能测试

### 4.1.1 一般功能测试

**测试用例**：

int maze[7][7] =

{

1,1,1,1,1,1,1,

1,0,1,0,0,0,1,

1,0,1,0,1,1,1,

1,0,0,0,1,0,1,

1,0,1,0,0,0,1,

1,0,1,0,1,0,1,

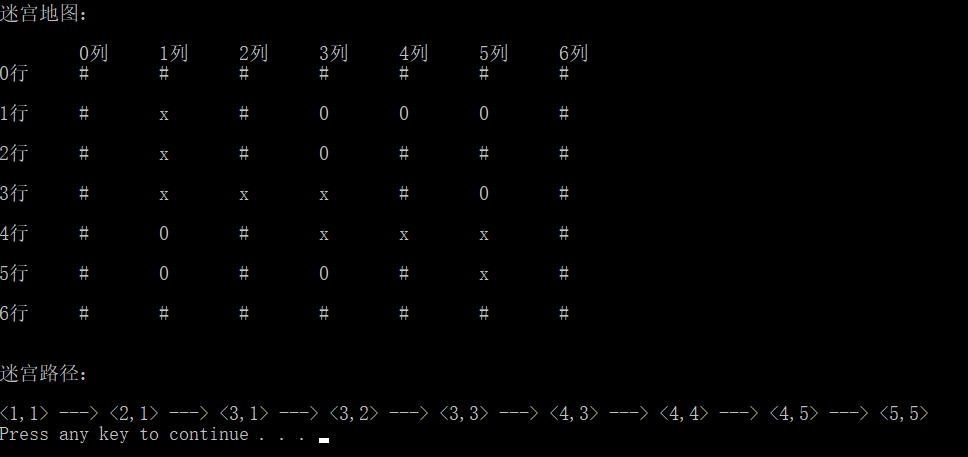
1,1,1,1,1,1,1

}

**预期结果**：

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

**实验结果**



# 5 亮点

## 5.1 模块化程度高

本题中我的所有主要功能基本都由各种函数实现，主函数内基本没有操作，使程序更模块化，更便于管理，主函数：

int main()

{

offsets move[4];

setOffsets(move);

findPath(5, 5, move);

system("pause");

return 0;

}

## 5.2 使用了栈作为存储，避免递归算法

本题中，为了使算法的效率更高，避免由于递归造成的高复杂度，我采用了栈存储加循环的方式，以空间换时间，使代码的效率增加，灵活性更强。

详情见2.1， 3.2.