项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 杨煜

学 号： 1850217

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](#_Toc26871600)

[1.1 项目内容 1](#_Toc26871601)

[1.2 项目要求 1](#_Toc26871602)

[2 设计 1](#_Toc26871603)

[2.1 算法分析 1](#_Toc26871604)

[2.2 递归思想的本质 2](#_Toc26871605)

[2.3 迷宫构建 2](#_Toc26871606)

[3 实现 3](#_Toc26871607)

[3.1 深度优先搜索的实现 3](#_Toc26871608)

[3.1.1 dfs函数 3](#_Toc26871609)

[3.1.2 函数核心代码 3](#_Toc26871610)

[3.2 总体流程的实现 5](#_Toc26871611)

[3.2.1 总体流程 5](#_Toc26871612)

[3.2.2 总体系统核心代码 5](#_Toc26871613)

[3.2.3 总体系统流程图 8](#_Toc26871614)

[4 测试 9](#_Toc26871615)

[4.1 功能测试 9](#_Toc26871616)

[4.1.1 一般情况 9](#_Toc26871617)

# 1 分析

## 1.1 项目内容

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。

## 项目要求

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。

# 2 设计

## 2.1 算法分析

如上述项目要求所述，该程序要对一个迷宫进行求解，通过入口进入找到出口。可以采用回溯法的方法，即深度优先搜索的方法。通过对某一个方向不停的迭代加深，如果找到出口，则该深度搜索结束，函数返回值为1，如果没有方向可以继续前进，则向上一层回溯，达到上一个节点重新判别下个方向。如果都找不到则函数返回为零，如果找到则输出路径（不一定是最短路）。如果要输出最短路径，则可以使用广度优先的搜索方法，可以最快速的找到所需要的最短路。该程序使用的深度优先算法的时间复杂度为O(4^n),时间开销巨大。

## 2.2 递归思想的本质

递归算法（英语：recursion algorithm）在计算机科学中是指一种通过重复将问题分解为同类的子问题而解决问题的方法。递归式方法可以被用于解决很多的计算机科学问题，因此它是计算机科学中十分重要的一个概念。绝大多数编程语言支持函数的自调用，在这些语言中函数可以通过调用自身来进行递归。计算理论可以证明递归的作用可以完全取代循环。

为了保证递归过程的每次调用和返回的正确执行，在高级语言中，是利用一个递归工作栈来进行处理的。在每进入一层递归时，系统就要建立一个新的工作记录，吧项目登入，加到递归工作栈的栈顶。

## 2.3 迷宫构建

将样例植入程序

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

{

maze[0][i] = '#';

maze[i][0] = '#';

maze[6][i] = '#';

maze[i][6] = '#';

}

maze[1][2] = '#';

maze[2][2] = '#';

maze[2][4] = '#';

maze[2][5] = '#';

maze[3][4] = '#';

maze[4][2] = '#';

maze[5][2] = '#';

maze[5][4] = '#';

for (int i = 1; i < 6; i++)

{

for (int j = 1; j < 6; j++)

{

if (maze[i][j] != '#')

{

maze[i][j] = '0';

}

}

}

# 3 实现

## 3.1 深度优先搜索的实现

### 3.1.1 dfs函数

dfs函数即深度优先搜索函数，接受两个参数i，j，这两个参数代表了当前的节点的位置。首先，判断节点的坐标是否为终止坐标，如果是，则将其标记为X然后步长加一，记录路径。如果不是终止调节则进行正常的判读，如果它的左方有可以行走的通路0，则将该点标记为@，然后递归调用dfs（i,j-1），如果能够达到最终的出口节点，则将该点标记为X，并记录下节点位置，以及步长加一。如果不能则不记录。如果该方向不能经过则查找下一方向，以此类推。以上，该函数使用了填充其他符号的方式避免了死循环的产生。

### 3.1.2 函数核心代码

bool dfs(int i, int j)

{

if (i == 5 && j == 5)//判断是否为终止条件

{

maze[i][j] = 'X';//记录节点

number1[path] = i;

number2[path] = j;

path++;

return 1;

}

if (maze[i][j - 1] == '0')//左方是否有通路

{

maze[i][j] = '@';

if (dfs(i, j - 1))//递归

{

maze[i][j] = 'X';

number1[path] = i;

number2[path] = j;

path++;

return 1;

}

}

if (maze[i - 1][j] == '0')//判断上方

{

maze[i][j] = '@';

if (dfs(i - 1, j))

{

number1[path] = i;

number2[path] = j;

path++;

maze[i][j] = 'X';

return 1;

}

}

if (maze[i][j + 1] == '0')//判断右方

{

maze[i][j] = '@';

if (dfs(i, j + 1))

{

number1[path] = i;

number2[path] = j;

path++;

maze[i][j] = 'X';

return 1;

}

}

if (maze[i + 1][j] == '0')//判断下方

{

maze[i][j] = '@';

if (dfs(i + 1, j))

{

number1[path] = i;

number2[path] = j;

path++;

maze[i][j] = 'X';

return 1;

}

}

return 0;//没有通路，返回false

}

## 3.2 总体流程的实现

### 3.2.1 总体流程

总体首先完成了迷宫的建立，本程序内置有通路的迷宫。然后在入口处开始进行dfs的深度优先搜索，也就是使用回溯法，找到迷宫正确的路径。然后输出迷宫时注意到要将填充的@符号改回0符号，然后逆序输出路径即可。

### 3.2.2 总体系统核心代码

int main()

{

for (int i = 0; i < 7; i++)//建立迷宫

{

maze[0][i] = '#';

maze[i][0] = '#';

maze[6][i] = '#';

maze[i][6] = '#';

}

maze[1][2] = '#';

maze[2][2] = '#';

maze[2][4] = '#';

maze[2][5] = '#';

maze[3][4] = '#';

maze[4][2] = '#';

maze[5][2] = '#';

maze[5][4] = '#';

for (int i = 1; i < 6; i++)

{

for (int j = 1; j < 6; j++)

{

if (maze[i][j] != '#')

{

maze[i][j] = '0';

}

}

}

dfs(1, 1);//从入口开始搜索

cout << " 0列 1列 2列 3列 4列 5列 6列" << endl;

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

{

printf("%d行 ", i);

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

{

if (maze[i][j] == '@')//去除标记

{

cout << "0" << " ";

}

else

{

cout << maze[i][j] << " ";

}

}

cout << endl;

}

cout << "迷宫路径：";

for (int i = path; i >= 0; i--)//逆序输出路径

{

printf("<%d,%d> ", number1[i], number2[i]);

if (i != 0)

{

cout << "--> ";

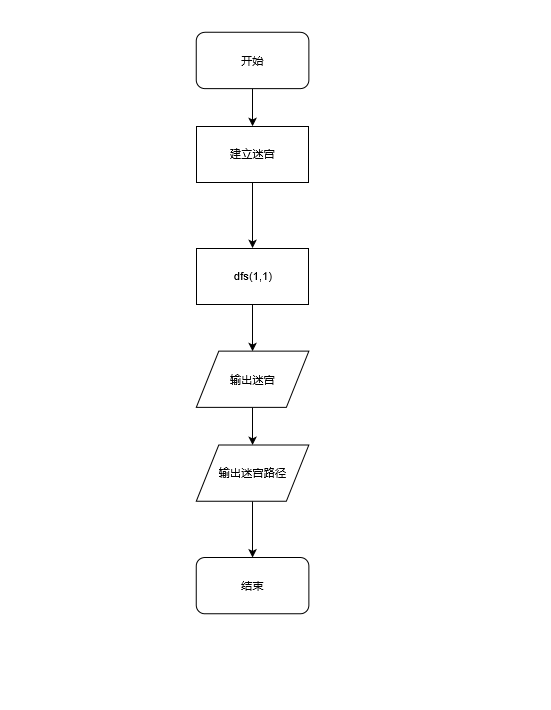
}

}

return 0;

}

### 3.2.3 总体系统流程图



# 4 测试

## 4.1 功能测试

### 4.1.1 一般情况

