项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 刘畅

学 号： 2054164

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](#_Toc84771247)

[1.1背景分析 1](#_Toc84771248)

[1.2功能分析 1](#_Toc84771249)

[2 设计 1](#_Toc84771250)

[2.1 算法设计 1](#_Toc84771251)

[2.2 结构设计 1](#_Toc84771252)

[2.3 成员与操作设计 2](#_Toc84771253)

[2.4 系统设计 2](#_Toc84771254)

[3 实现 2](#_Toc84771255)

[3.1 输入函数的实现 2](#_Toc84771256)

[3.1.1 核心代码 2](#_Toc84771257)

[3.2 输出函数的实现 3](#_Toc84771258)

[3.2.1 核心代码 3](#_Toc84771259)

[3.3 递归搜索函数的实现 3](#_Toc84771260)

[3.3.1 核心代码 3](#_Toc84771261)

[3.5 统计功能的实现 4](#_Toc84771262)

[3.5.1 统计功能流程图 4](#_Toc84771263)

[3.5.2 统计功能核心代码 4](#_Toc84771264)

[3.5.3 统计功能截屏示例 4](#_Toc84771265)

[4 测试 4](#_Toc84771266)

[4.1 测试1 4](#_Toc84771267)

[4.2 测试2 5](#_Toc84771268)

[4.3 测试3 5](#_Toc84771269)

[4.4 测试4 6](#_Toc84771270)

# 1 分析

## 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。

## 功能分析

对本题建立如下数学模型：有一个只包含0和1的维矩阵，1代表迷宫中的障碍，0表示可以通过，现给定和，有一个人从左上角即(1,1)点出发，只能通过上下左右移动，且不能离开矩阵，问到达最短距离？

将矩阵建立坐标系，行数作为横坐标，列数作为纵坐标，这样最后的路径就能用一连串坐标表达。若能找到这样的一条路径，则输出路径；若无法找到，则输出“无法到达终点！”。

# 2 设计

## 2.1 算法设计

如上功能分析所述，迷宫问题的求解过程可以采用深度优先搜索（或者广度优先搜索），即在一定的约束条件下试探地探索前进（即不能越过矩阵边界），若前进过程中受阻，则及时回头纠正错误并另择通路继续探索。

深度优先搜索采用递归实现，对于现在所处的某个点，判断其四个方向是否能走，若能走，则前进；终止条件是现在所处的某个点即为终点，此时更新最短距离，并保存路径；若当前已经走过的距离超过了最短距离，则没有必要再继续走了，这里可以认为是一个简单的剪枝。

## 2.2 结构设计

将“点”封装为Point类，方便路径的存储和输出，创建2个路径数组，一个为过程性的，在递归过程中会被不断被更改，另一个是结果性的，存储最短路径；设置一个curLength=0和minLength=max，当套用一次递归公式时，就让curLength++，当回溯时就让curLength--，这样就能时刻知道在迷宫中进过了多少距离。对于迷宫的大小，为了简化起见，设置为定值20\*20，这样路径的最大值也就是20\*20，所以用来存储路径的Point数组也只需要开2\*20\*20，若要进行动态分配，则需要手写一个动态分配的vector（当然也可以直接用标准stl的vector类），较为麻烦，此处作简化处理。

代码结构采用递归方式实现，递归函数为search\_shortest\_route(int x, int y)，其中x和y是当前状态下所在的位置；在递归函数中，首先判断当前路径长度是否超过已有的最短路径，若已经超过则return；不然，则判断当前所处位置是否为终点，若为终点，则将当前路径存储到最短路径，同时更新minLength；当递归完毕，则输出最短路径即可。

## 2.3 成员与操作设计

**点类（Point）**

**public:**

int \_x, \_y;

Point(int x = 0, int y = 0) :\_x(x), \_y(y) {}

## 2.4 系统设计

系统首先输出提示词，然后接受n和m，再接受一个的01矩阵；然后进行运算，最后输出结果路径。

# 3 实现

## 3.1 输入函数的实现

### 3.1.1 核心代码

void input() {

//起点(1,1);终点(n,m)

cin >> n >> m;

startPoint = Point(1, 1);

endPoint = Point(n, m);

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= m; j++) {

cin >> map[i][j];

}

}

}

## 3.2 输出函数的实现

### 3.2.1 核心代码

void output() {

if (minLength != maxn \* maxm) {

cout << "(" << shortest\_route[0].\_x << "," << shortest\_route[0].\_y << ")";

for (int i = 1; i <= minLength; i++) {

cout << "--->(" << shortest\_route[i].\_x << "," << shortest\_route[i].\_y << ")";

}

}

else {

cout << "没有通向终点的路径！" << endl;

}

}

## 3.3 递归搜索函数的实现

### 3.3.1 核心代码

void search\_shortest\_route(int x, int y) {

//判断当前路径长度是否大于等于最短路径

if (curLength >= minLength) {

//若是，则直接返回

return;

}

else if (x == endPoint.\_x && y == endPoint.\_y) {

//若小于，且当前已是终点，则复制路径，并更新最短路径，注意这个地方最短路径应该要减1

for (int i = 0; i < curLength; i++) {

shortest\_route[i].\_x = route[i].\_x;

shortest\_route[i].\_y = route[i].\_y;

}

minLength = curLength-1;

return;

}

int tx, ty;

//对于当前的位置，试着向上下左右四个方向移动

//移动之前判断目标位置是否合理，即是否越界、已经走过、有障碍物

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

tx = x + step[i][0];

ty = y + step[i][1];

if (tx >= 1 && ty >= 1 && tx <= n && ty <= m && !map[tx][ty] && !b[tx][ty]) {

//走之前将其标记为走过，并当前路径长度加一，加入到当前路径之中

b[tx][ty] = 1;

route[curLength] = Point(tx, ty);

curLength++;

search\_shortest\_route(tx, ty);

//走之后路径长度减一，并将其标记为未走过

curLength--;

b[tx][ty] = 0;

}

}

}

# 4 测试

## 4.1 测试1

**测试用例：**

**8 9**

**0 1 1 1 1 0 0 0 0**

**0 0 0 0 0 0 1 1 0**

**1 0 1 1 1 0 0 0 0**

**0 0 0 0 1 1 0 1 0**

**0 1 1 0 0 0 0 1 0**

**0 1 0 0 1 0 1 0 0**

**1 0 1 0 1 0 1 0 0**

**0 0 0 0 0 0 1 0 0**

实验结果：

(1,1)--->(2,1)--->(2,2)--->(2,3)--->(2,4)--->(2,5)--->(2,6)--->(3,6)--->(3,7)--->(3,8)--->(3,9)--->(4,9)--->(5,9)--->(6,9)--->(7,9)--->(8,9)

## 4.2 测试2

**测试用例：**

**20 20**

**0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 0 1 1**

**1 0 0 1 0 0 1 0 0 0 0 1 0 1 1 1 1 0 0 1**

**0 0 0 0 0 0 0 1 0 1 0 1 0 0 1 1 1 1 0 1**

**0 1 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 1 1**

**1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 1 0 1**

**0 0 1 0 1 1 1 0 0 1 0 1 1 1 1 1 0 1 0 1**

**1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 1 1 0**

**1 0 1 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 1 1**

**0 0 1 0 0 1 1 1 0 0 1 0 1 1 1 1 0 0 1 1**

**1 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0**

**0 0 1 0 1 1 0 0 0 0 0 0 0 0 1 1 0 1 0 1**

**0 1 1 0 0 0 0 1 1 0 0 1 0 1 0 1 0 0 1 0**

**0 0 0 1 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0**

**0 0 0 1 0 1 1 0 1 0 1 0 1 0 0 0 0 1 0 1**

**1 1 0 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 1**

**0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1**

**1 1 1 0 0 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0**

**0 0 1 1 0 1 0 1 0 1 1 0 0 0 0 1 0 0 0 1**

**1 0 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 0 1**

**1 1 1 1 0 1 1 0 0 1 1 1 0 1 1 1 0 0 0 0**

实验结果：

(1,1)--->(1,2)--->(2,2)--->(3,2)--->(3,3)--->(4,3)--->(5,3)--->(5,4)--->(6,4)--->(7,4)--->(8,4)--->(9,4)--->(9,5)--->(10,5)--->(10,6)--->(10,7)--->(11,7)--->(12,7)--->(13,7)--->(13,8)--->(14,8)--->(15,8)--->(15,9)--->(15,10)--->(15,11)--->(15,12)--->(15,13)--->(16,13)--->(16,14)--->(17,14)--->(17,15)--->(17,16)--->(17,17)--->(18,17)--->(18,18)--->(19,18)--->(20,18)--->(20,19)--->(20,20)

## 4.3 测试3

**测试用例：**

**1 1**

**0**

实验结果：

(1,1)

## 4.4 测试4

**测试用例：**

**3 3**

**0 0 1**

**0 1 0**

**1 0 0**

实验结果：

没有通向终点的路径！