项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 王加炜

学 号： 2150265

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

# 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。

# 设计

## 2.1数据结构设计

如上功能分析所述，该项目每个点都需要存储各方向通向的位置及自身是否是可通过的，此外，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，故需要保存每一点的数据。对于整个迷宫，各个点之间相互独立，且需要大量的访问各点信息，无需很多的增删操作，故采用数组存储各个点，方便访问各个点的信息。走迷宫的过程采用回溯法，调用递归的函数完成操作。

## 2.2类结构设计

将迷宫信息存放在一个二维数组中，通过递归求得的通路的x坐标和y坐标信息分别存储在一个一维数组中，数组下标表示是第几步。

## 2.3成员与操作设计

### 2.3.1全局变量

int src\_x, src\_y;

int des\_x, des\_y;

char map[100][100];

int step\_x[100];

int step\_y[100];

int step\_ = 0;

int row, col;

### 2.3.2全局函数

migong函数：

bool migong(int current\_x, int current\_y)

输入函数：

void input()

输出函数：

void output()

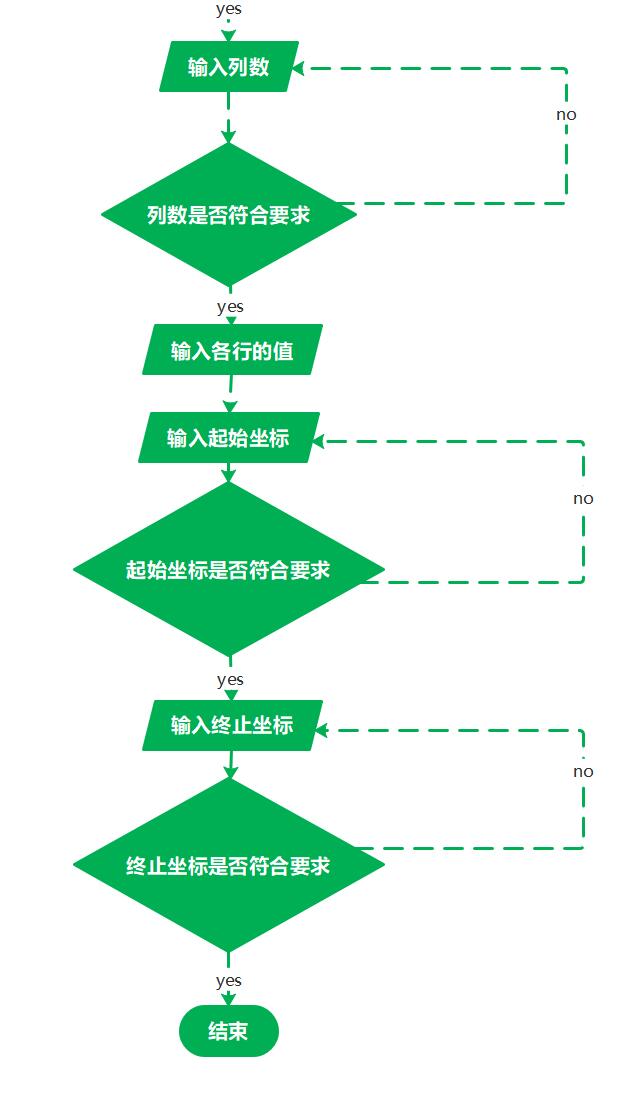
## 2.4系统设计

用户输入迷宫的行数和列数，然后手动输入一个迷宫，指定起始位置和终止位置，通过迷宫遍历算法将步数信息存储在数组中，最后输出步数信息和图形化界面。

# 三．实现

## 3.1初始化信息输入

### 3.1.1流程图



### 3.1.2文字说明

该函数作用是提示用户输入初始化信息。信息包括要建立的迷宫行 数，列数以及每行的迷宫信息，最后提示用户输入起始坐标和终止坐标。 将输入的所有信息分别存储在对应的全局变量当中。其中包含了错误 输入的情况，用户如果输入错误会提示用户重新输入。

### 3.1.3代码实现

void input()

{

//输入行号加列号信息，外加错误判断

while (1)

{

cout << "输入行数";

cin >> row;

if (row <= 0)

{

cout << "行号不符，请重新输入！！！" << endl;

}

else

{

break;

}

}

while (1)

{

cout << "输入列数";

cin >> col;

if (col <= 0)

{

cout << "列号不符，请重新输入！！！" << endl;

}

else

{

break;

}

}

//输入各行的值

cout << "输入各行的值(障碍用#表示，通路用0表示)" << endl;

for (int i = 0; i <= row + 1; i++)

{

if (i != 0 && i != row + 1)

{

cout << "第" << i << "行是：";

}

for (int j = 0; j <= col + 1; j++)

{

if (i == 0 || i == row + 1 || j == 0 || j == col + 1)

{

map[i][j] = '#';

}

else

{

cin >> map[i][j];

}

}

}

//输入开始位置和终止位置，外加错误判断信息

while (1)

{

cout << "开始位置（第x行,第y列)";

cin >> src\_x;

cin >> src\_y;

if (map[src\_x][src\_y] == '0')

{

break;

}

else

{

cout << "输入位置有误，请重新输入！！！" << endl;

}

}

while (1)

{

cout << "结束位置（第x行,第y列)";

cin >> des\_x;

cin >> des\_y;

if (map[des\_x][des\_y] == '0')

{

break;

}

else

{

cout << "输入位置有误，请重新输入！！！" << endl;

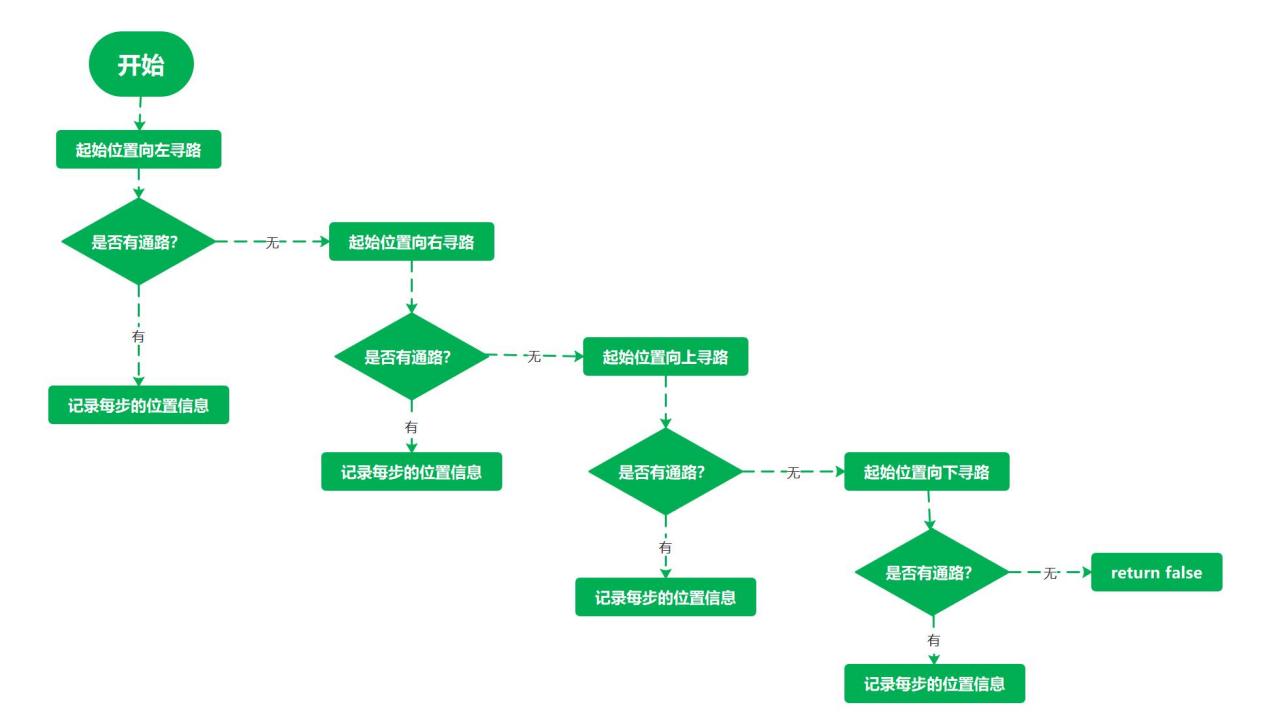
}

}

}

## 3.2migong函数进行路径查找

### 3.2.1流程图



### 3.2.2文字说明

该函数是一个递归函数，模拟了从起始坐标开始从上下左右四个方向一层层遍历，直至找到终止坐标的过程。该函数的参数分别表示当前遍历的位置的横坐标和纵坐标。返回值为true，表示找寻到通路路径；否则未找到。

该递归函数的终止条件是当前位置和终止坐标重合，表示遍历已经完成，直接return true。如果当前位置不是终止坐标的位置，则通过递归遍历当前位置的上下左右四个位置。如果四个位置的递归函数返回值均为false，说明最终通路路径没有经过当前位置；而只要其中一个位置的递归函数返回true，则说明最终通路经过了当前位置，将当前位置的坐标信息记录到step数组中去，以便后续按顺序输出通路路径使用。

需要注意的是，如果当前位置已经被访问，则后续遍历时不会重新访问该位置。这对于降低该算法的时间复杂度有积极作用。同时也保证了在遍历的过程中不会重复经过相同位置。

### 3.2.3代码实现

bool migong(int current\_x, int current\_y)

{

if (current\_x == des\_x && current\_y == des\_y)

{

return true;

}

else

{

if (map[current\_x][current\_y] == '0')//该位置未被遍历

{

map[current\_x][current\_y] = 'x';//表示该位置已经遍历过

if (migong(current\_x + 1, current\_y))//沿该方向遍历可以找到到终点的通路

{

step\_x[++step\_] = current\_x + 1;

step\_y[step\_] = current\_y;

return true;

}

else if (migong(current\_x - 1, current\_y))//沿该方向遍历可以找到到终点的通路

{

step\_x[++step\_] = current\_x - 1;

step\_y[step\_] = current\_y;

return true;

}

else if (migong(current\_x, current\_y + 1))//沿该方向遍历可以找到到终点的通路

{

step\_x[++step\_] = current\_x;

step\_y[step\_] = current\_y + 1;

return true;

}

else if (migong(current\_x, current\_y - 1))//沿该方向遍历可以找到到终点的通路

{

step\_x[++step\_] = current\_x;

step\_y[step\_] = current\_y - 1;

return true;

}

else

{

map[current\_x][current\_y] = '$';//表示该位置无通路

return false;

}

}

else

{

return false;

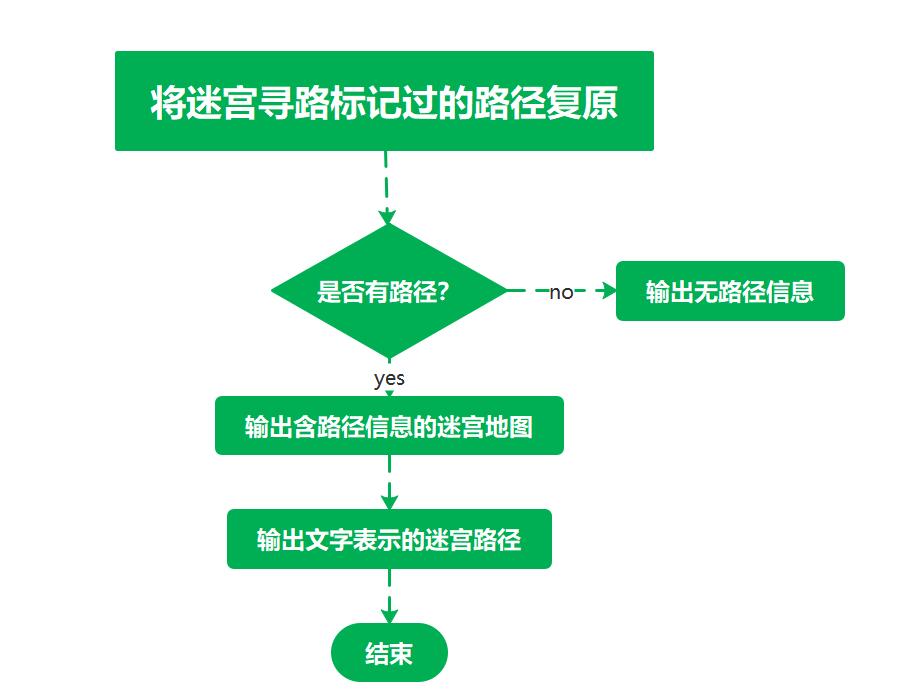
}

}

}

## 3.3输出函数

### 3.3.1流程图



### 3.3.2文字说明

该函数实现新地图的打印显示，其中要经过的路径位置用不同符号加以区别。首先将递归函数执行时判断出该位置不可通但是其实是可走 位置的坐标还原，最后通过坐标的二元组形式将从起点到终点的移动路径完整地表示出来。如果migong算法未找到该路径，不打印路径，并打印”未找到相关路径“；若找到，则正常显示该路径。

需要注意的是，由于递归函数存放step数组的过程是通过栈逆序执行的，即最后经过的通路位置信息会在step数组的最前面。所以在输出路径信息的时候需要从step数组的末位置开始往前输出。

### 3.3.3代码实现

void output()

{

//输出整个迷宫的可走路径图

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

{

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

{

if (map[i][j] == '$')

{

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

}

}

}

map[des\_x][des\_y] = 'x';

cout << "迷宫地图：" << endl;

cout << " ";

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

{

cout << setw(4) << j << "列";

}

cout << endl;

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

{

cout << setiosflags(ios::left) << i << setw(4) << "行";

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

{

cout << setiosflags(ios::left) << setw(5) << map[i][j] << " ";

}

cout << endl;

}

cout << endl;

cout << "迷宫路径：" << endl;

cout << "<" << src\_x << "," << src\_y << "> -->";

for (int i = step\_; i > 0; i--)

{

if (i > 1)

cout << " <" << step\_x[i] << "," << step\_y[i] << "> -->";

else

cout << " <" << step\_x[i] << "," << step\_y[i] << "> ";

}

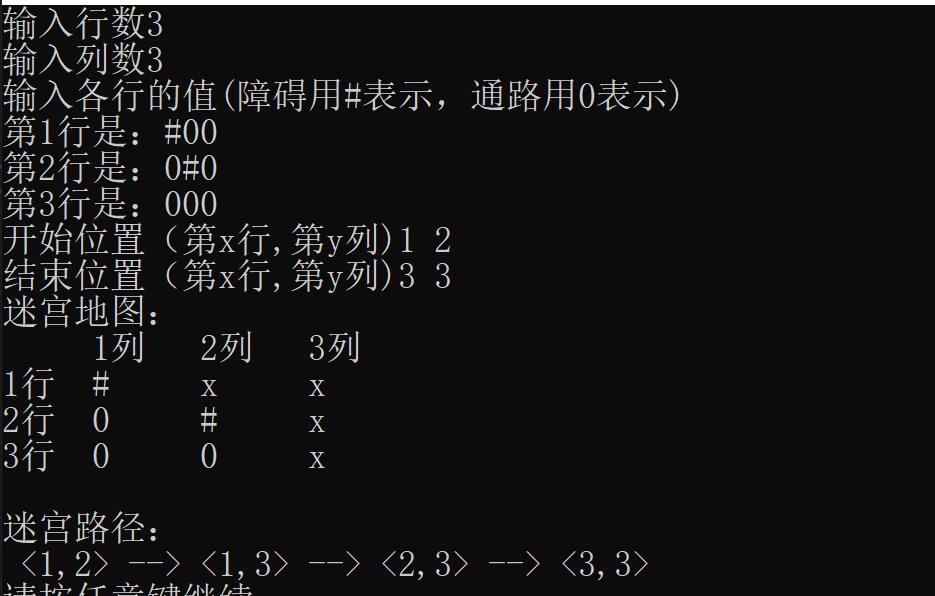
cout << endl;

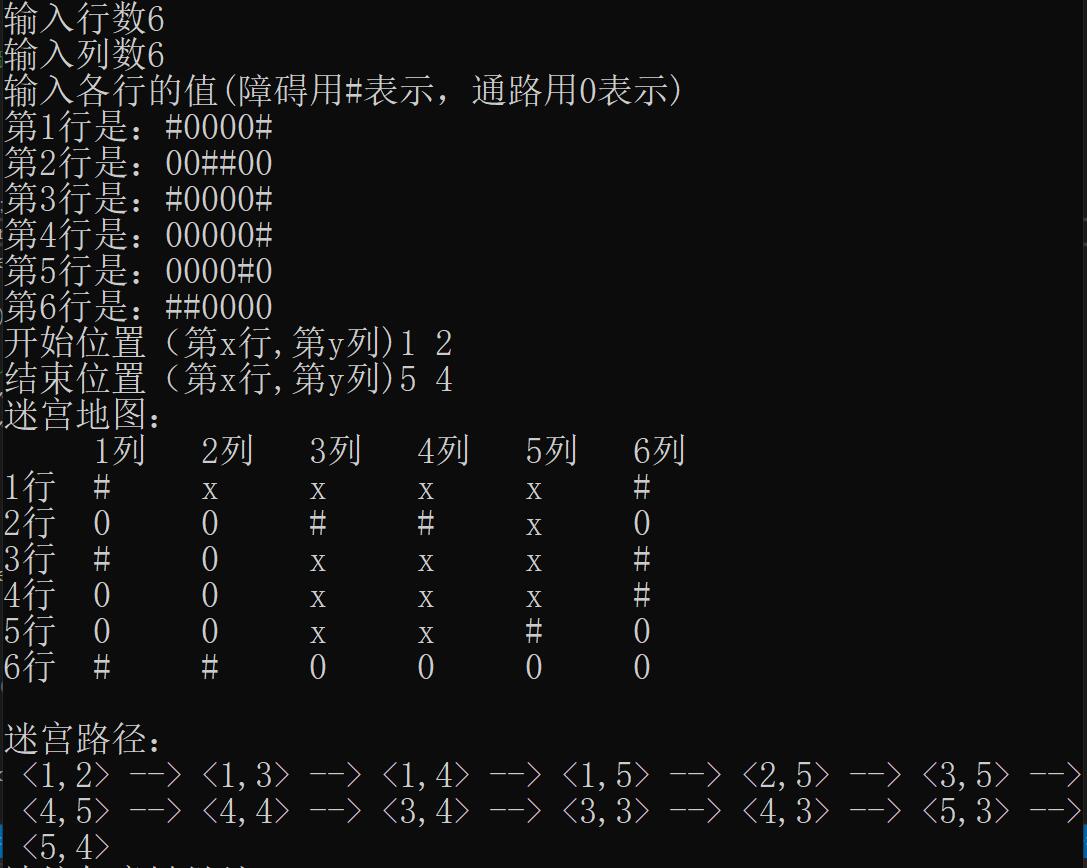
}

# 四．测试

## 4.1功能测试

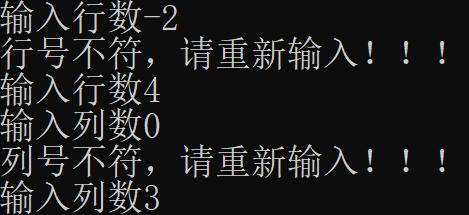
### 4.1.1正常输入的情况：



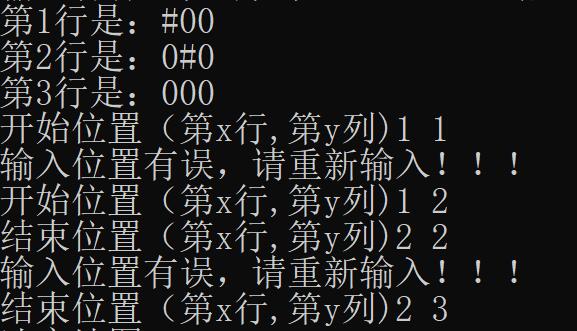


## 4.2错误测试

### 4.2.1行号和列号不按照规定输入：



### 4.2.2起始坐标和终止坐标处不可达:



### 4.2.3起始坐标和终止坐标之间没有可达路径：

