项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 郑柯凡

学 号： 1950072

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 2](#_Toc1166)

[1.1背景分析 2](#_Toc25957)

[1.2功能分析 3](#_Toc21146)

[2 设计 3](#_Toc4669)

[2.1思路设计 3](#_Toc23109)

[2.2数据结构的选择 3](#_Toc22343)

[2.2.1递归方式 3](#_Toc25375)

[2.2.2非递归方式 3](#_Toc28777)

[2.3类的设计 4](#_Toc18888)

[2.3.1递归方式 4](#_Toc16101)

[2.3.2非递归方式 5](#_Toc3342)

[3 功能实现 8](#_Toc18051)

[3.1 个性化初始化迷宫功能的实现（递归方式与非递归方式一致） 8](#_Toc26886)

[3.1.1个性化初始化迷宫功能流程图 9](#_Toc9412)

[3.1.2个性化初始化迷宫功能核心代码 9](#_Toc6118)

[3.1.3个性化初始化迷宫功能截屏示例 10](#_Toc23257)

[输入不规范 10](#_Toc20135)

[输入规范 11](#_Toc18491)

[3.2 打印迷宫功能的实现（递归方式与非递归方式一致） 11](#_Toc31670)

[3.2.1打印迷宫功能流程图 11](#_Toc25362)

[3.2.2打印迷宫功能核心代码 11](#_Toc25812)

[3.2.3打印迷宫功能截屏示例 12](#_Toc8426)

[3.3 搜索路径功能的实现（递归方式） 12](#_Toc5303)

[3.3.1搜索路径功能流程图 12](#_Toc4305)

[3.3.2搜索路径功能核心代码 12](#_Toc26356)

[3.3.3搜索路径功能截屏示例 13](#_Toc21093)

[3.4 搜索路径功能的实现（非递归方式） 13](#_Toc7922)

[3.4.1搜索路径功能流程图 13](#_Toc17250)

[3.4.2搜索路径功能核心代码 14](#_Toc21439)

[3.4.3搜索路径功能截屏示例 14](#_Toc5584)

[4测试 15](#_Toc29077)

[4.1 7\*7大小迷宫测试 15](#_Toc17935)

[4.2 10\*10大小迷宫测试 16](#_Toc26678)

[4.3 7\*7大小无通路迷宫测试 16](#_Toc24113)

**1 分析**

**1.1背景分析**

人类建造迷宫已有5000年的历史。在世界的不同文化发展时期，这些奇特的建筑物始终吸引人们沿着弯曲的墙壁不断探索，寻找真相。因此，迷宫类小游戏也应运而生。在游戏中，玩家通常会给定一个入口和一个出口，在不断的推进和试错中找到迷宫的出口，最终走出迷宫。

**1.2功能分析**

在此项目中，给定入口和出口，需要从入口出发，通过一种方式找到一条通到终点的路，并将其输出。或没有一条这样的路，输出此迷宫无解。

**2 设计**

**2.1思路设计**

迷宫问题的一般求解过程就是采用回溯的方法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。

具体来说，先从起点开始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，当找到出口时就结束了，否则无解。

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，能正确返回前一个点以便从下一个方向向前试探，需要在试探过程中保存所达到的每个点的位置以及该点前进的方向。可以用递归的方法实现，也可以用一个后进先出的栈来储存经过的点信息。

**2.2数据结构的选择**

**2.2.1递归方式**

递归方式中，程序通过不断地调用自身来实现搜索中的前进，执行完毕就回溯。因此不需要其他的存储结构，只需要一个二维数组来存储地图的坐标信息（是否是障碍物），以及一个二维数组来存储每个坐标点的访问信息。

**2.2.2非递归方式**

非递归方式中，除了需要两个二维数组来存储地图坐标的可否通过信息和访问信息之外，还需要一个栈来存储经过的每个坐标信息（横纵坐标，以及上次前进的方向），利用栈后进先出的特性，在回溯时弹出最后压入的坐标点，然后从下一个方向继续搜索。同时为了避免数据溢出的情况，采用链式栈的结构。

**2.3类的设计**

**2.3.1递归方式**

坐标类：私有成员包括横坐标x和纵坐标y。

1. **class** Point {
2. **friend** **class** Maze;
3. **private**:
4. **int** x, y;
5. **public**:
6. Point() { x = y = 0; }
7. ~Point() {};
8. };

迷宫类：私有成员包括二维字符数组存储的迷宫地图，二维布尔数组存储的访问信息，上下左右四个方向向量，以及起点和终点。公有函数包括初始化的构造函数，析构函数，个性化初始化函数，获取起点坐标函数以及搜索通路函数。

1. **class** Maze {
2. **private**:
3. //迷宫地图
4. **char** map[row][column]{};
5. //访问数组
6. **bool** visited[row][column]{};
7. //4个方向向量
8. Point dir[4];
9. //起点终点
10. Point start;
11. Point end;
12. **public**:
13. //构造函数，构造迷宫
14. Maze()
15. {
16. //初始化地图和访问数组
17. **for** (**int** i = 0; i < row; i++)
18. {
19. **for** (**int** j = 0; j < column; j++)
20. {
21. map[i][j] = '0';
22. visited[i][j] = 0;
23. }
24. }
25. //设置障碍物
26. **int** barnum = 30;
27. **int** bar[row \* column][2] = {
28. {0,0},{0,1},{0,2},{0,3},{0,4},{0,5},{0,6},{1,2},{1,6},{2,0},{2,2},{2,4},{2,5},{2,6},{3,0},
29. {3,4},{3,6},{4,0},{4,2},{4,6},{5,0},{5,2},{5,4},{5,6},{6,0},{6,1},{6,2},{6,3},{6,4},{6,6}
30. };
31. **for** (**int** i = 0; i < barnum; i++)
32. {
33. map[bar[i][0]][bar[i][1]] = '#';
34. }
35. //设置四个方向向量和起点终点坐标
36. dir[0].x = -1; dir[0].y = 0;
37. dir[1].x = 0; dir[1].y = 1;
38. dir[2].x = 1; dir[2].y = 0;
39. dir[3].x = 0; dir[3].y = -1;
40. start.x = 0; start.y = 0;
41. end.x = 0; end.y = 0;
42. };
43. //析构函数，清除迷宫
44. ~Maze() {};
45. //个性化初始化迷宫
46. **bool** Init();
47. //输出迷宫
48. **void** Display();
49. //获取起点信息
50. **int** Getstartx() { **return** start.x; }
51. **int** Getstarty() { **return** start.y; }
52. //寻找通路
53. **bool** SeekPath(**int** x,**int** y);
54. };

**2.3.2非递归方式**

坐标类：大体和递归方式一致，多加了一个私有成员dir记录方向。

1. //坐标类
2. **class** Point {
3. **friend** **class** Maze;
4. **friend** **class** Node;
5. **friend** **class** LinkStack;
6. **private**:
7. **int** x, y;
8. **int** dir;
9. **public**:
10. Point() { x = y = dir = 0; }
11. ~Point() {};
12. };

迷宫类：大体和递归方式一致。删除了获取起点坐标函数，因为非递归方式中不需要在成员函数之外使用起点坐标。搜索通路函数构造略有不同，下文详解。

1. **class** Maze {
2. **private**:
3. //迷宫地图
4. **char** map[row][column]{};
5. //访问数组
6. **bool** visited[row][column]{};
7. //四个方向向量
8. Point dir[4]{};
9. //起点终点
10. Point start;
11. Point end;
12. **public**:
13. //构造函数，构造迷宫
14. Maze()
15. {
16. //默认初始化
17. **for** (**int** i = 0; i < row; i++)
18. {
19. **for** (**int** j = 0; j < column; j++)
20. {
21. map[i][j] = '0';
22. visited[i][j] = 0;
23. }
24. }
25. **int** barnum = 30;
26. **int** bar[row \* column][2] = {
27. {0,0},{0,1},{0,2},{0,3},{0,4},{0,5},{0,6},{1,2},{1,6},{2,0},{2,2},{2,4},{2,5},{2,6},{3,0},
28. {3,4},{3,6},{4,0},{4,2},{4,6},{5,0},{5,2},{5,4},{5,6},{6,0},{6,1},{6,2},{6,3},{6,4},{6,6}
29. };
30. **for** (**int** i = 0; i < barnum; i++)
31. {
32. map[bar[i][0]][bar[i][1]] = '#';
33. }
34. //设置四个方向
35. dir[0].x = -1; dir[0].y = 0;
36. dir[1].x = 0; dir[1].y = 1;
37. dir[2].x = 1; dir[2].y = 0;
38. dir[3].x = 0; dir[3].y = -1;
39. //设置起点终点
40. start.x = 0; start.y = 0;
41. end.x = 0; end.y = 0;
42. };
43. //析构函数，清除迷宫
44. ~Maze() {};
45. //个性化初始化迷宫
46. **bool** Init();
47. //输出迷宫
48. **void** Display();
49. //寻找通路
50. **void** SeekPath();
51. };

（以下代码包含在MySTL头文件中）

链式栈结点类：私有成员包含一个坐标信息，以及下一个结点的地址。

1. //结点
2. **template** <**class** Type>
3. **class** Node {
4. **private**:
5. Type data;
6. Node<Type>\* next;
7. **public**:
8. Node() { next = NULL; }
9. ~Node() {};
10. //获取结点信息
11. Type GetData() { **return** data; }
12. //获取下个结点地址
13. Node<Type>\* GetNext() { **return** next; }
14. //修改数据
15. **void** ModifyData(Type tmp)
16. {
17. data = tmp;
18. };
19. //修改地址
20. **void** ModifyNext(Node<Type>\*tmp)
21. {
22. next = tmp;
23. }
24. };

链式栈类：私有成员包括栈顶元素地址，以及栈的大小。公有成员包括压入函数，弹出并获取栈顶元素函数，以及判断栈是否为空的函数。

其中，压入函数实现如下：利用一个辅助结点，使其与输入结点相等，再将其放至栈顶链接原来的栈顶元素，最后修改栈顶地址。栈的大小加1。

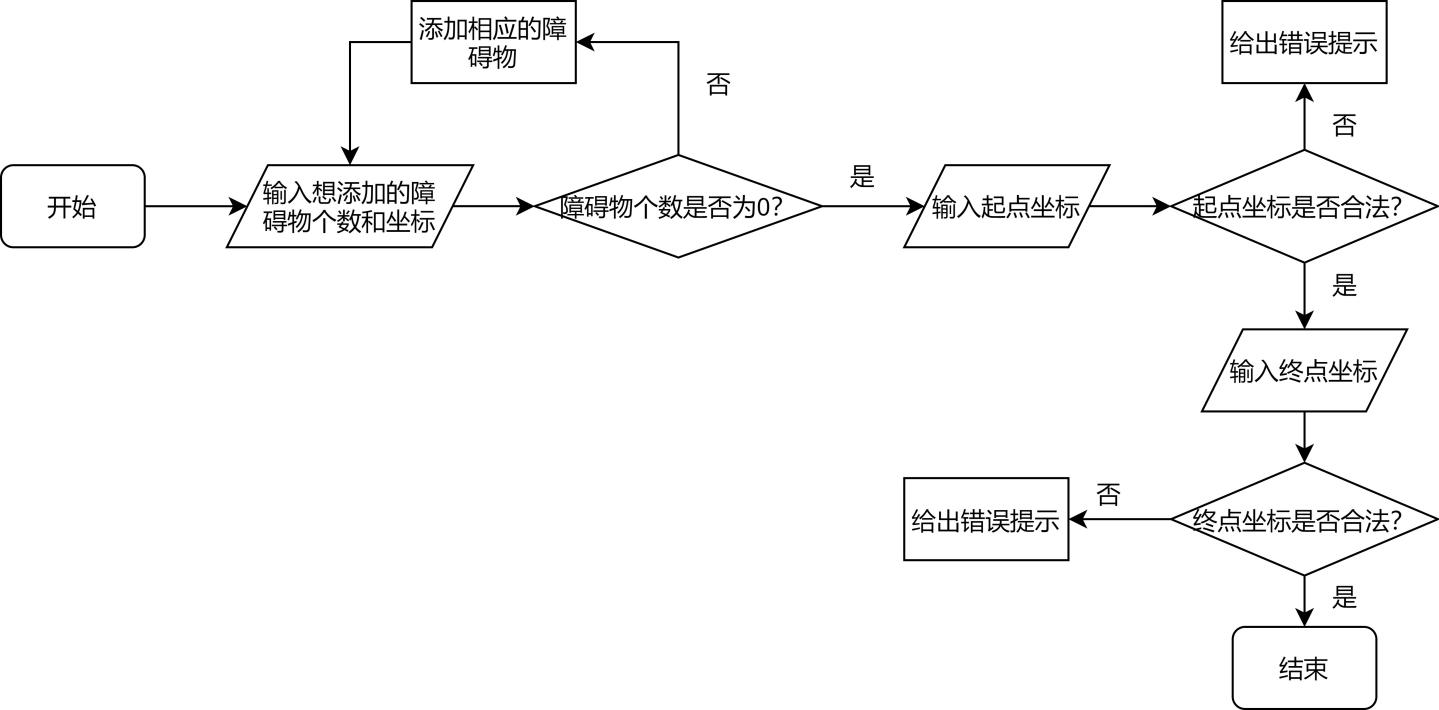
弹出函数实现如下：利用一个辅助结点储存栈顶元素信息，并将其输出到目标结点中，最后修改栈顶元素地址并delete辅助结点。

1. //链式栈
2. **template** <**class** Type>
3. **class** LinkStack {
4. **private**:
5. //栈顶元素地址
6. Node<Type>\* top;
7. //栈的大小
8. **int** size;
9. **public**:
10. //链式栈的构造函数(置空栈)
11. LinkStack() {
12. **this**->top = NULL;
13. **this**->size = 0;
14. };
15. //析构函数
16. ~LinkStack() {};
17. //将坐标放入栈顶
18. **void** Push(Type tmp)
19. {
20. Node<Type>\* temp = **new** Node<Type>;
21. temp->ModifyData(tmp);
22. temp->ModifyNext(**this**->GetTop());
23. **this**->top = temp;
24. **this**->size++;
25. };
26. //将栈顶的坐标弹出
27. **void** Pop(Type& tmp)
28. {
29. Node<Type>\* temp = **this**->GetTop();
30. tmp = temp->GetData();
31. top = top->GetNext();
32. size--;
33. **delete** temp;
34. };
35. //判断栈是否为空
36. **bool** IsEmpty() {
37. **if** (**this**->size > 0)
38. {
39. **return** 0;
40. }
41. **else**
42. {
43. **return** 1;
44. }
45. }
46. //获取栈顶元素地址
47. Node<Type>\* GetTop() { **return** top; };
48. //获取栈的大小
49. **int** GetSize() { **return** size; }
50. };

**3 功能实现**

**3.1 个性化初始化迷宫功能的实现（递归方式与非递归方式一致）**

**3.1.1个性化初始化迷宫功能流程图**

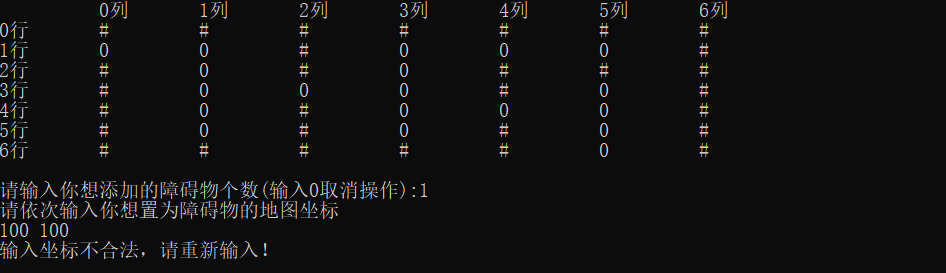


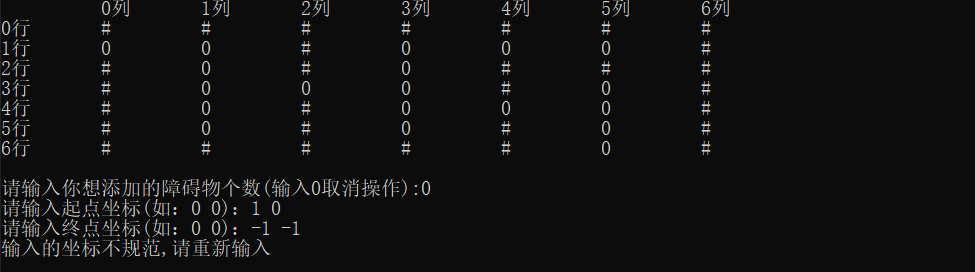
**3.1.2个性化初始化迷宫功能核心代码**

1. **bool** Maze::Init()
2. {
3. **while** (1)
4. {
5. //个性化初始化
6. **int** num, row2, column2;
7. **while** (1)
8. {
9. cout << "请输入你想添加的障碍物个数(输入0取消操作):";
10. cin >> num;
11. **if** (num <= 0)
12. {
13. **break**;
14. }
15. cout << "请依次输入你想置为障碍物的地图坐标" << endl;
16. **for** (**int** i = 0; i < num; i++)
17. {
18. cin >> row2 >> column2;
19. **if** (row2 >= 0 && row2 < row && column2 >= 0 && column2 < column)
20. {
21. map[row2][column2] = '#';
22. }
23. **else**
24. {
25. cout << "输入坐标不合法，请重新输入！" << endl << endl;
26. }
27. }
28. **this**->Display();
29. }
30. //起点终点坐标互换，便于回溯时输出路径
31. **int** x, y;
32. cout << "请输入起点坐标(如：0 0)："; cin >> x >> y;
33. **if** (x >= 0 && x < row && y >= 0 && y < column && map[x][y] == '0')
34. {
35. end.x = x; end.y = y;
36. }
37. **else**
38. {
39. cout << "输入的坐标不规范,请重新输入" << endl;
40. **return** 0;
41. }
42. cout << "请输入终点坐标(如：0 0)："; cin >> x >> y;
43. **if** (x >= 0 && x < row && y >= 0 && y < column && map[x][y] == '0')
44. {
45. start.x = x; start.y = y;
46. }
47. **else**
48. {
49. cout << "输入的坐标不规范,请重新输入" << endl;
50. **return** 0;
51. }
52. **break**;
53. }
54. **return** 1;
55. }

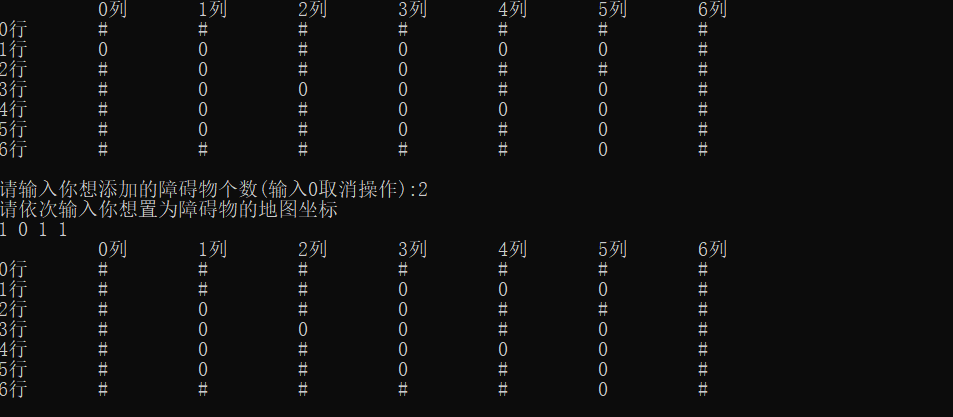
**3.1.3个性化初始化迷宫功能截屏示例**

**输入不规范**





**输入规范**



**3.2** 打印迷宫功能的实现（递归方式与非递归方式一致）

**3.2.1打印迷宫功能流程图**

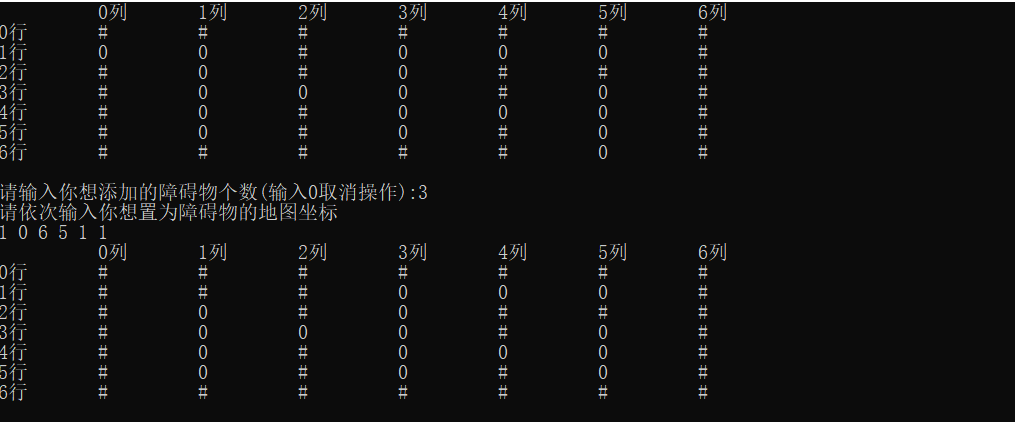
****

%3CmxGraphModel%3E%3Croot%3E%3CmxCell%20id%3D%220%22%2F%3E%3CmxCell%20id%3D%221%22%20parent%3D%220%22%2F%3E%3CmxCell%20id%3D%222%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BexitX%3D1%3BexitY%3D0.5%3BexitDx%3D0%3BexitDy%3D0%3BentryX%3D0%3BentryY%3D0.5%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%223%22%20target%3D%226%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%223%22%20value%3D%22%E5%BC%80%E5%A7%8B%22%20style%3D%22rounded%3D1%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22130%22%20y%3D%22200%22%20width%3D%2270%22%20height%3D%2240%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%224%22%20value%3D%22%E7%BB%93%E6%9D%9F%22%20style%3D%22rounded%3D1%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22450%22%20y%3D%22200%22%20width%3D%2270%22%20height%3D%2240%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%225%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BexitX%3D1%3BexitY%3D0.5%3BexitDx%3D0%3BexitDy%3D0%3BentryX%3D0%3BentryY%3D0.5%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%226%22%20target%3D%224%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%226%22%20value%3D%22%E9%81%8D%E5%8E%86%E5%9C%B0%E5%9B%BE%E6%95%B0%E7%BB%84%26lt%3Bbr%26gt%3B%E5%B0%86%E5%86%85%E5%AE%B9%E8%BE%93%E5%87%BA%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22270%22%20y%3D%22200%22%20width%3D%22100%22%20height%3D%2240%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3C%2Froot%3E%3C%2FmxGraphModel%3E

**3.2.2打印迷宫功能核心代码**

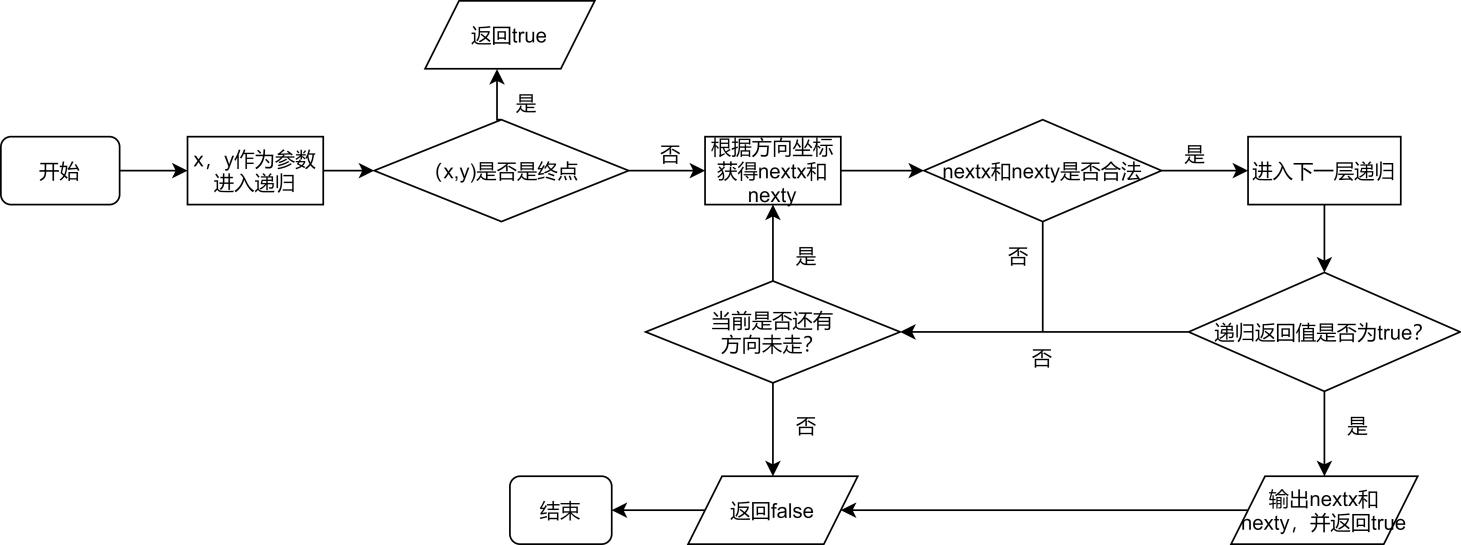
1. **void** Maze::Display()
2. {
3. cout << setiosflags(ios::left);
4. cout << setw(10) << " ";
5. **for** (**int** j = 0; j < column; j++)
6. {
7. string temp = "0列";
8. temp[0] = j + 48;
9. cout << setw(10) << temp;
10. }
11. cout << endl;
12. **for** (**int** i = 0; i < row; i++)
13. {
14. string temp = "0行";
15. temp[0] = i + 48;
16. cout << setw(10) << temp;
17. **for** (**int** j = 0; j < column; j++)
18. {
19. cout << setw(10) << map[i][j];
20. }
21. cout << endl;
22. }
23. cout << endl;
24. }

**3.2.3打印迷宫功能截屏示例**



**3.3 搜索路径功能的实现（递归方式）**

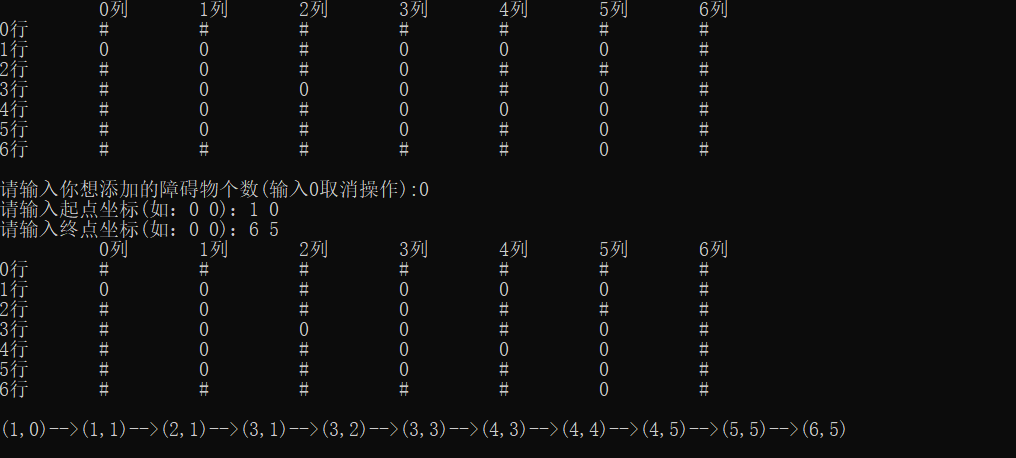
**3.3.1搜索路径功能流程图**

****

**3.3.2搜索路径功能核心代码**

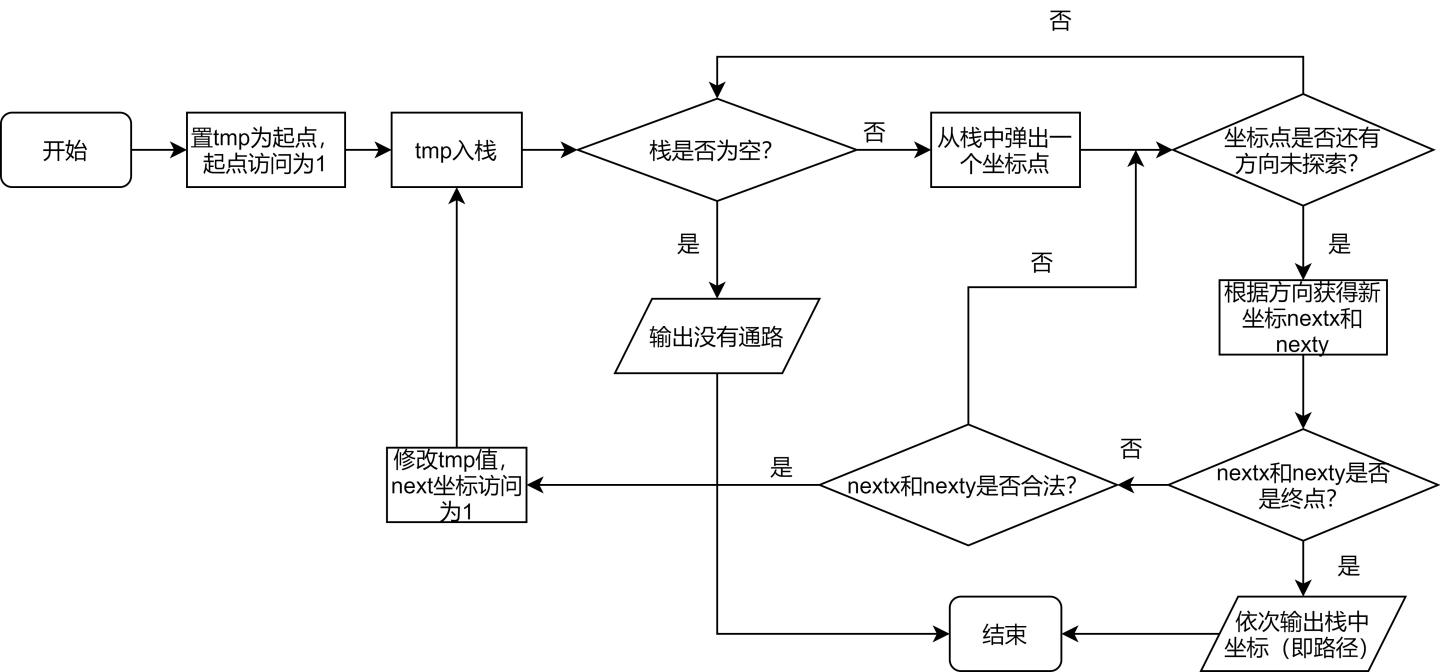
1. **bool** Maze::SeekPath(**int** x,**int** y)
2. {
3. **int** nextx, nexty, i;
4. **if** (x == end.x && y == end.y) { **return** 1; }
5. **for** (i = 0; i < 4; i++)
6. {
7. nextx = x + dir[i].x;
8. nexty = y + dir[i].y;
9. **if** (nextx < 0 || nextx >= row || nexty < 0 || nexty >= column)
10. {
11. **continue**;
12. }
13. **if** (map[nextx][nexty] == '0' && visited[nextx][nexty] == 0)
14. {
15. visited[nextx][nexty] = 1;
16. **if** (SeekPath(nextx, nexty)==1)
17. {
18. cout << "(" << nextx << "," << nexty << ")-->";
19. **return** 1;
20. }
21. }
22. }
23. **if**(x==start.x&&y==start.y)
24. {
25. cout << "NO PATH!" << endl;
26. **return** 0;
27. }
28. **return** 0;
29. }

**3.3.3搜索路径功能截屏示例**



**3.4 搜索路径功能的实现（非递归方式）**

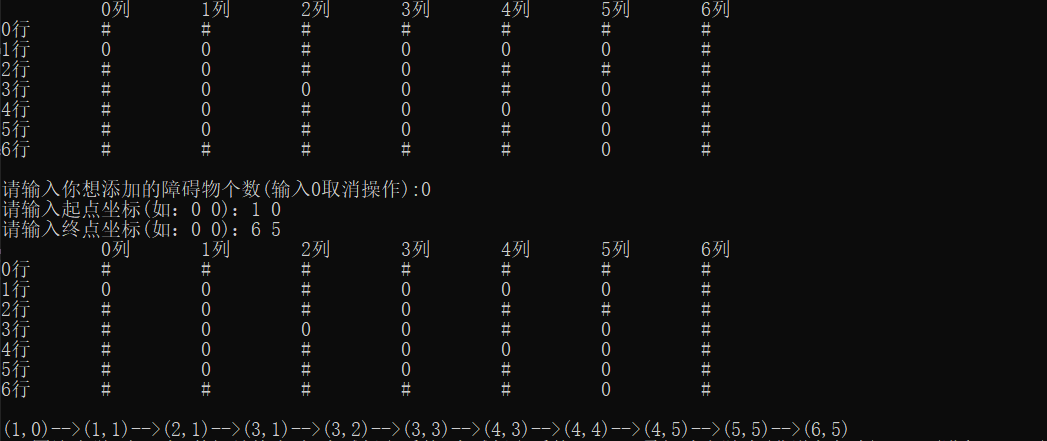
**3.4.1搜索路径功能流程图**

****

**3.4.2搜索路径功能核心代码**

1. **void** Maze::SeekPath()
2. {
3. **int** nextx, nexty, curx, cury, d;
4. visited[start.x][start.y] = 1;
5. LinkStack<Point> st; Point tmp;
6. tmp.x = start.x; tmp.y = start.y;
7. st.Push(tmp);
8. **while** (st.IsEmpty() == 0)
9. {
10. st.Pop(tmp);
11. curx = tmp.x; cury = tmp.y; d = tmp.dir;
12. **while** (d < 4)
13. {
14. nextx = curx + dir[d].x;
15. nexty = cury + dir[d].y;
16. **if** (nextx < 0 || nextx >= row || nexty < 0 || nexty >= column)
17. {
18. d++;
19. **continue**;
20. }
21. **if** (nextx == end.x && nexty == end.y)
22. {
23. cout << "(" << nextx << "," << nexty << ")";
24. cout << "-->(" << curx << "," << cury << ")";
25. Node<Point>\* temp = st.GetTop();
26. **while** (temp != NULL)
27. {
28. cout << "-->(" << temp->GetData().x << "," << temp->GetData().y << ")";
29. temp = temp->GetNext();
30. }
31. cout << endl;
32. **return**;
33. }
34. **if** (map[nextx][nexty] == '0' && visited[nextx][nexty] == 0)
35. {
36. visited[nextx][nexty] = 1;
37. tmp.x = curx; tmp.y = cury; tmp.dir = d;
38. st.Push(tmp);
39. curx = nextx; cury = nexty; d = 0;
40. }
41. **else**
42. {
43. d++;
44. }
45. }
46. }
47. cout << "NO PATH!" << endl;
48. }

**3.4.3搜索路径功能截屏示例**



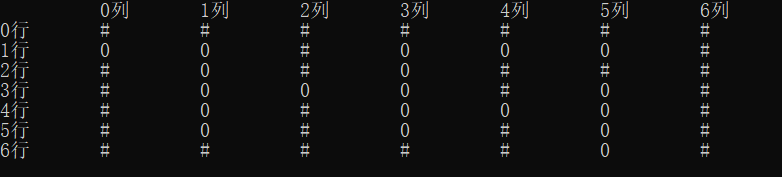
**4测试**

**4.1 7\*7大小迷宫测试**

测试用例：

Start:(1,0)

End:(6,5)



预期结果：(1,0)->(1,1)->(2,1)->(3,1)->(3,2)->(3,3)->(4,3)->(4,4)->(4,5)->(5,5)->(6,5)

实验结果：

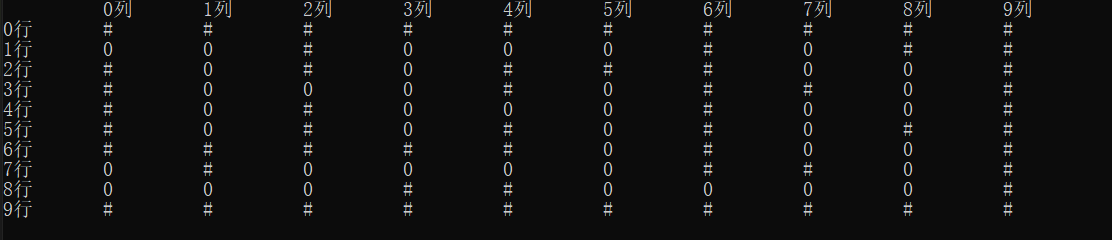


**4.2 10\*10大小迷宫测试**

测试用例：

Start:(1,0)

End:(1,7)



预期结果：(1,0)->(1,1)->(2,1)->(3,1)->(3,2)->(3,3)->(4,3)->(4,4)->(4,5)->(5,5)->(6,5)->(7,5)->(8,5)->(8,6)->(8,7)->(8,8)->(7,8)->(6,8)->(6,7)->(5,7)->(4,7)->(4,8)->(3,8)->(2,8)->(2,7)->(1,7)

实验结果：

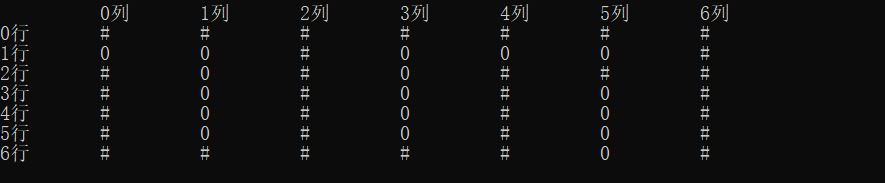


**4.3 7\*7大小无通路迷宫测试**

测试用例：

Start:(1,0)

End:(6,5)



预期结果：NO PATH!

实验结果：

