项目说明文档

数据结构课程设计

——勇闯迷宫问题

作 者 姓 名：　罗吉皓

学　　号：　1652792

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

I. 项目分析

1.1 项目名称： 勇闯迷宫问题

1.2 项目背景

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。

1.3 项目功能分析

作为一个简易的迷宫路径搜索问题，基本的功能实现是

1.记录所输入的迷宫并且可以予以显示

2.记录起始点及终止位置

3.对迷宫进行搜索，寻找从起始点到达终止点的可能路径

4.用坐标形式输出路径，并在地图上显示

综上所述，一个迷宫搜索项目至少应该具有输入、输出、搜索的功能。

II. 项目设计

2.1 数据结构设计

如上功能分析所述，该系统要求大量的搜索，回溯操作，因此考虑使用堆栈或者队列这两种数据结构之一。本项目使用的数据结构为队列，使用队列实现迷宫搜索算法的设计与使用堆栈实现迷宫搜索算法的设计的不同在文档的最后会有所提及。

2.2 数据结构类的设计

由于cpp STL库里自带了队列的类，在本项目中直接引用了<queue>的头文件

2.3 系统设计

系统首先提示用户输入迷宫的行数和列数，并且用字符串数组记录迷宫的分布（0表示路径，#表示墙）。最后提示用户输入起始位置（xStart,yStart)及终止位置(xEnd,yEnd)，至此所有的输入结束。（迷宫的坐标从（*0*，*0*）开始）

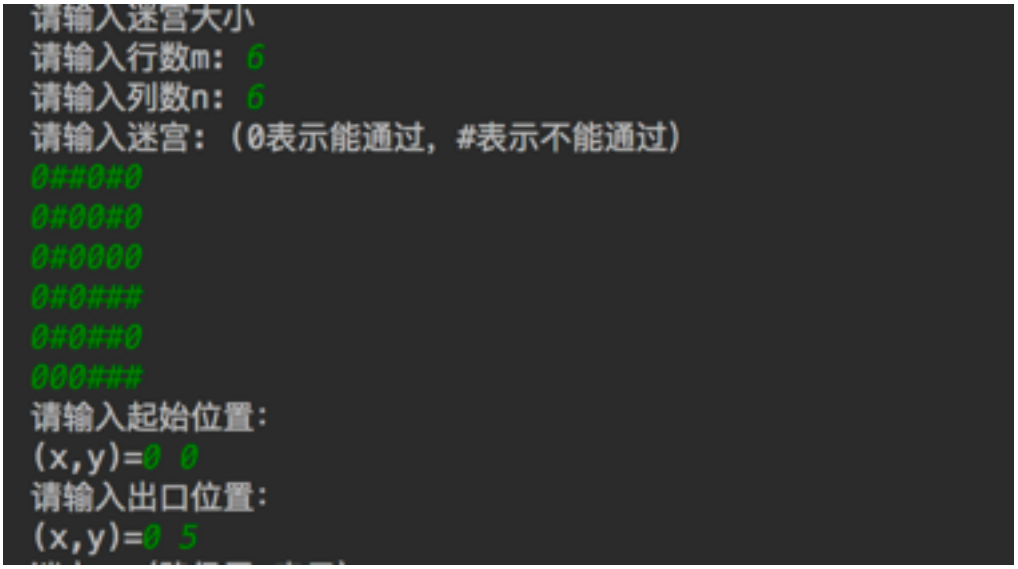
```
{
    cout << "请输入迷宫大小" << endl << "请输入行数m: ";
    cin >> m;
    cout << "请输入列数n: ";
    cin >> n;
    cout << "请输入迷宫: (0表示能通过, #表示不能通过) " << endl;
    //用0#的方式来代表迷宫的能走的位置及墙的位置
    while (true) {
        int flag=0;
        for (i = 0; i < m; i++)
            for (j = 0; j < n; j++) {
                cin >> Map[i][j];
                if (Map[i][j] != '0' && Map[i][j] != '#') {
                    cout << "输入错误! 请重新输入! " << endl;
                    flag=1;
                    cout << "请输入迷宫: (0表示能通过, #表示不能通
过) " << endl;
                    break;
                }
            }
        if (flag==0) break;
    }
    cout << "请输入起始位置: \n(x,y)=";
    cin  >> xStart >> yStart ;
    cout << "请输入出口位置: \n(x,y)=";
    cin  >> xEnd >> yEnd ;
}
```

输入的迷宫:

```
0##0#0
0#00#0
0#0000
0#0###
0#0##0
000###
```

起始位置: (0, 0) 终止位置 (0, 5)

输入功能截屏示例：

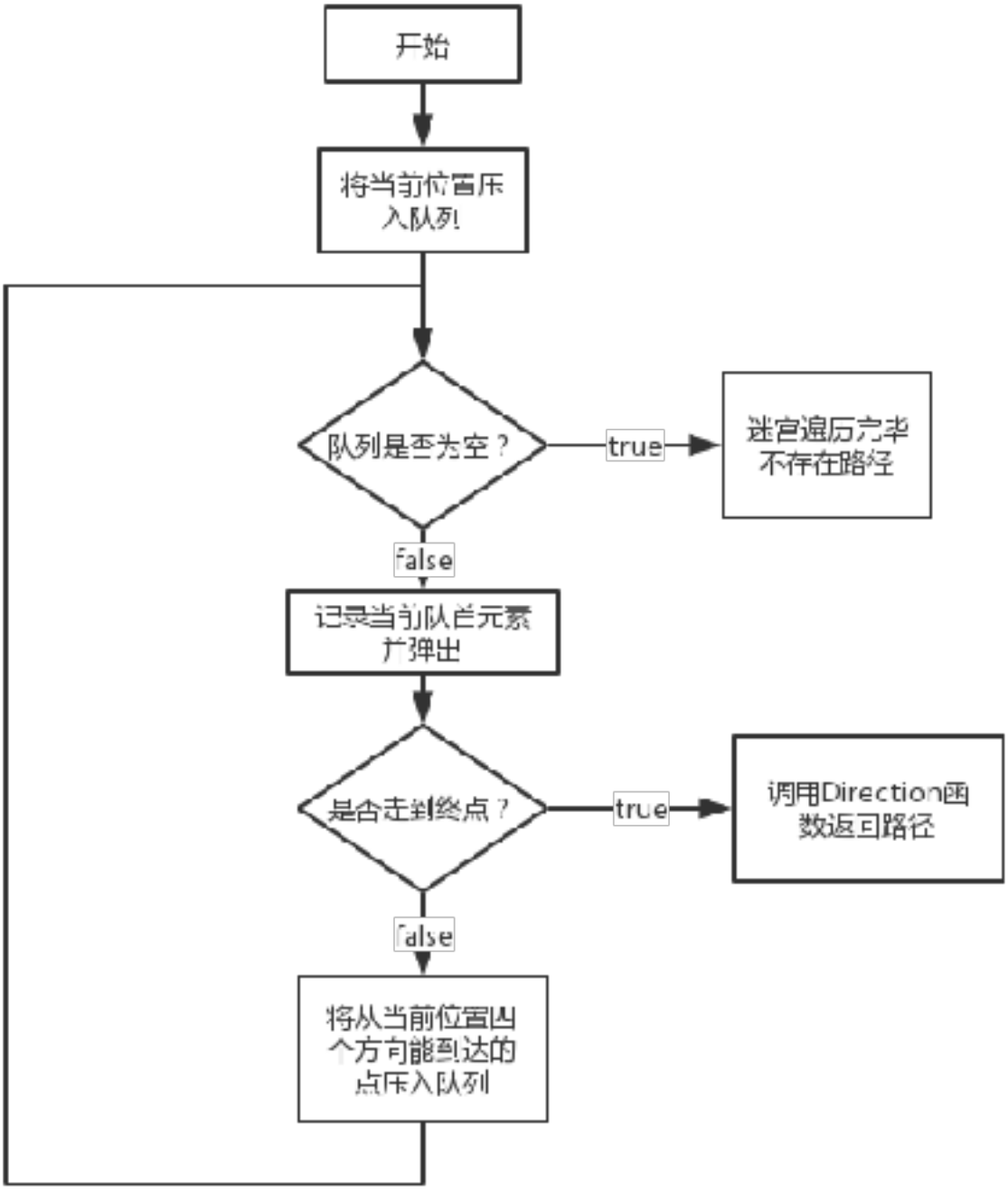


III 实现

```
自定义node类型 分别用来存储走到该点所用的步数step以及该点的位置(x,y)
struct node
{
    int x,y,step;
};
```

3.1 广度优先搜索算法的实现

3.1.1 搜索算法流程图


开始
将当前位置压入队列
队列是否为空？
true
迷宫遍历完毕不存在路径
false
记录当前队首元素并弹出
是否走到终点？
true
调用Direction函数返回路径
false
将从当前位置四个方向能到达的点压入队列

3.1.2 思路分析

在广度优先搜索过程中，设x和y是两个相继要被访问的未访问过的顶点。它们的邻接点分别记为x1，x2，…，xs和y1，y2，…，yt。

为确保先访问的顶点其邻接点亦先被访问，在搜索过程中使用FIFO队列来保存已访问过的顶点。当访问x和y时，这两个顶点相继入队。此后，当x和y相继出队时，我们分别从x和y出发搜索其邻接点x1，x2，…，xs和y1，y2，…，yt，对其中未访者进行访问并将其入队。这种方法是将每个已访问的顶点入队，故保证了每个顶点至多只有一次入队。

对于迷宫问题当然，需要注意的是不能重复访问已经访问过的结点，在这里我修改了原来迷宫数组Map，将原来的***'0'***修改成***'$'***，来标记已经访问过。

当我们的终点入列时，代表了我们已经搜索到了终点，此时我们便需要调用Direction函数进行相应的回溯操作，来输出好路径。如果队列已空，代表了所有的点已经被访问过，但是还是没有找到路径，此时我们便需要输出“不存在路径”。

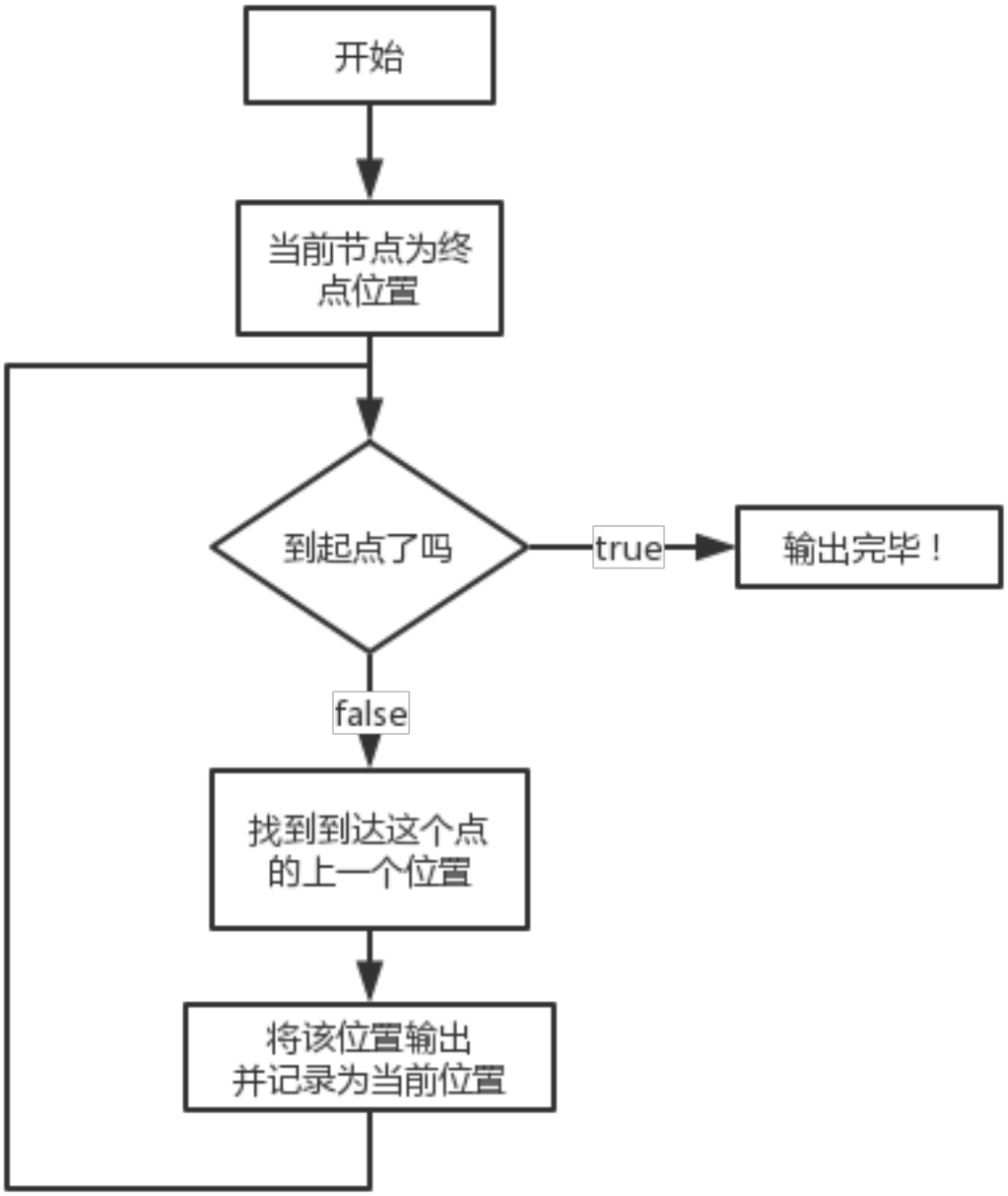
3.1.3 具体实现

核心代码展示：

```
{
    int dx[4]={-1, 0, 0, 1};
    int dy[4]={ 0, 1,-1, 0};
//dx dy分别为两个方向数组 {-1, 0} {0, 1} {0, -1} {1, 0} 代表四个方向
    while(!Que.empty())
    //当Que队列为空时，代表所有能走的点已经全部遍历过了
    {
        cur=Que.front();
        Que.pop();
        if(cur.x==xEnd && cur.y==yEnd)
        //如果当前已经遍历到了终点，调用Direction函数来回溯路径
        {
            Direction(m,n,xStart,yStart,xEnd,yEnd,ans,Map);
            return;
        }
        for(int i=0;i<4;i++)
        //i从0到3代表了下一步能走的四个方向
        {
            next.x=cur.x+dx[i];
            next.y=cur.y+dy[i];
            //next记录了从cur开始下一步将走到的位置
            if(next.x>=0 && next.x<m &&
               next.y>=0 && next.y<n &&
               Map[next.x][next.y]== '0')
  //判断条件 下一步的位置（next）应该在合理的范围内 并且下一步是可走的路径
            {
                Map[next.x][next.y]='s';
                //用s表示该点已经被遍历过了 方便遍历
                next.step=cur.step+1;
                //下一步(next)是这一步(cur)的步数加一
                ans[next.x][next.y].x=cur.x;
                ans[next.x][next.y].y=cur.y;
                //ans数组记录走到这一位置的上一位置 方便最后的回溯
                Que.push(next);
                //将next压入队列
            }
        }
    }
}
```

3.2 回溯算法实现

3.2.1 回溯算法流程图



3.2.2 思路分析

回溯算法的整个过程从终点位置开始，不断寻找走到当前位置的前一步所在的坐标并记录下来，直至走到起始位置为止。

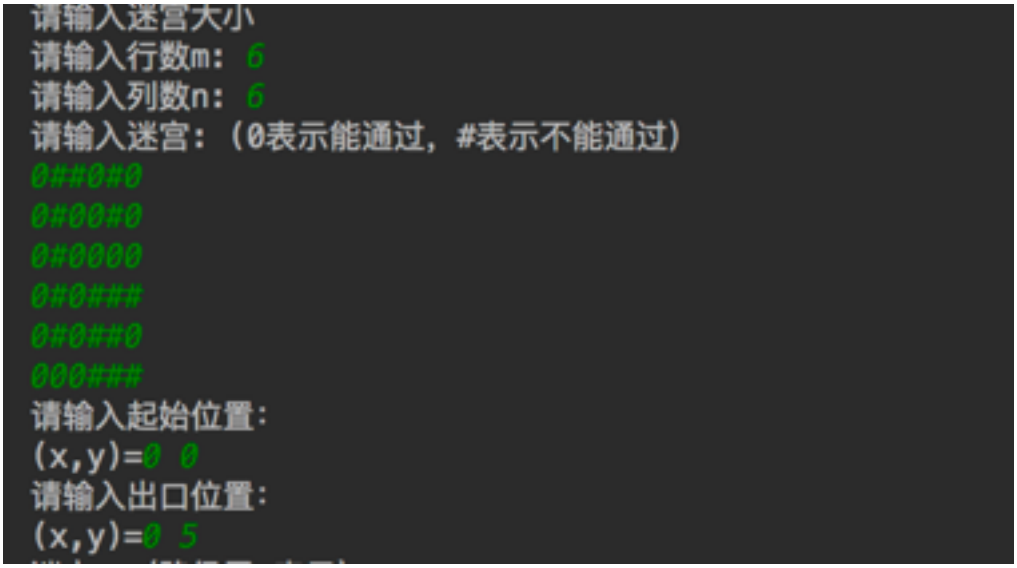
3.2.3 具体实现

核心代码如下：

```
{
    node path[MAXN];
    //path数组 用来记录最后的路径
    int x=xEnd,y=yEnd;
    int k=1,xTemp,yTemp;
    //xTemp与yTemp分别
    Map[xStart][yStart]='*';
    Map[xEnd][yEnd]='*';
    while(x!=xStart || y!=yStart)//通过回溯的过程得到我走过的路径
    {
        xTemp=x;
        yTemp=y;
        path[k].x=ans[x][y].x;
        path[k].y=ans[x][y].y;
        //path记录路径
        x=ans[xTemp][yTemp].x;
        y=ans[xTemp][yTemp].y;
        //用ans数组回溯上一步经过的点
        k++;
    }
}
```

c++

3.3 总体系统截屏示例



```
迷宫：（路径用*表示）
      0列     1列     2列     3列     4列     5列
0行     *       #       #       0       #       *
1行     *       #       0       0       #       *
2行     *       #       *       *       *       *
3行     *       #       *       #       #       #
4行     *       #       *       #       #       0
5行     *       *       *       #       #       #
路径：<0,0> ---> <1,0> ---> <2,0> ---> <3,0> ---> <4,0> ---> <5,0> ---> <5,1> ---> <5,2> ---> <4,2>
  ---> <3,2> ---> <2,2> ---> <2,3> ---> <2,4> ---> <2,5> ---> <1,5> ---> <0,5>
```

IV 测试

4.1搜索功能测试

```
*****************************测试用例*****************************
请输入行数m： 4
请输入列数n： 4
请输入迷宫：（0表示能通过，#表示不能通过）
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
请输入起始位置：
(x,y)=0 0
请输入出口位置：
(x,y)=3 3
```

```
*****************************预期结果*****************************
迷宫：（路径用*表示）
      0列     1列     2列     3列
0行     *       *       *       *
1行     0       0       0       *
2行     0       0       0       *
3行     0       0       0       *
路径：<0,0> ---> <0,1> ---> <0,2> ---> <0,3> ---> <1,3> ---> <2,3> ---> <3,3>
```

实际结果

```
迷宫：（路径用*表示）
      0列     1列     2列     3列
0行     *       *       *       *
1行     0       0       0       *
2行     0       0       0       *
3行     0       0       0       *
路径：<0,0> ---> <0,1> ---> <0,2> ---> <0,3> ---> <1,3> ---> <2,3> ---> <3,3>
```

4.2.健壮性实验

4.2.1 输入健壮性判断

```
*****************************测试用例*****************************
请输入迷宫大小
请输入行数m: 4
请输入列数n: 4
请输入迷宫：(0表示能通过，#表示不能通过)
1 2 3 4
```

```
*****************************预期结果*****************************
输入错误！请重新输入！
```

实际结果

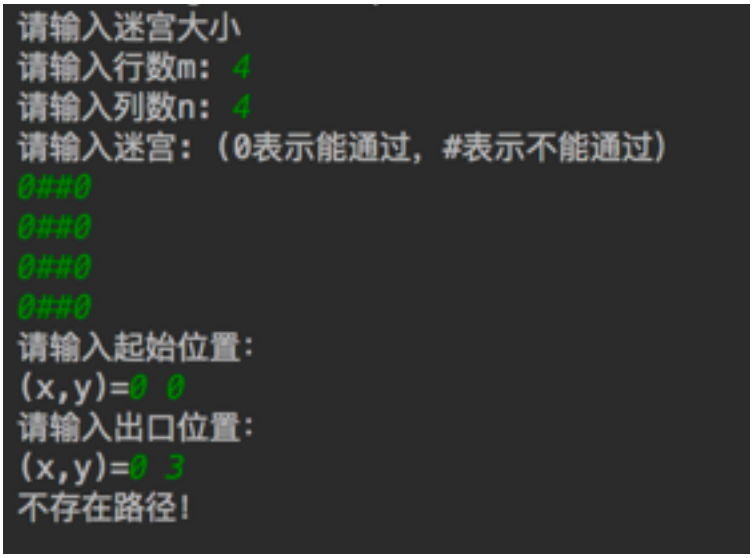
```
请输入迷宫大小
请输入行数m: 4
请输入列数n: 4
请输入迷宫：(0表示能通过，#表示不能通过)
1 2 3 4
输入错误！请重新输入！
```

4.2.2 输入迷宫没有路径

```
*****************************测试用例*****************************
请输入迷宫大小
请输入行数m: 4
请输入列数n: 4
请输入迷宫：(0表示能通过，#表示不能通过)
0##0
0##0
0##0
0##0
请输入起始位置:
(x,y)=0 0
请输入出口位置:
(x,y)=0 3
```

```
*****************************预期结果*****************************
不存在路径！
```

实际结果



V 总结

迷宫项目的基本功能至此已经全部实现了，在整个实验中,我学会利用了广度优先搜索的方法来实现迷宫问题的求解。

在本次项目中，在深度优先搜索（堆栈）与广度优先搜索（队列）中，我选择了广度优先搜索，原因如下：

1. 深度优先搜索采用的是递归的方式尝试下一步，而广度优先搜索采用的是循环结构去遍历，时间复杂度的考虑 广度优先搜索要远远优于深度优先搜索
2. 在有多条路径的情况下，深度优先搜索会随机的选择其中一条路径，而广度优先搜索输出的一定是最优的（最短的）路径

当然，如果题目要求输出所有的路径，我会选择深度优先搜索。

VI 参考文献

1.http://blog.csdn.net/qq_25044847/article/details/50424976

2.数据结构课本