1.题目名称

迷宫问题

2.代码行数

134行

3.算法思想

1.迷宫存储

使用了邻接矩阵对地图进行存储

2.寻找出路

这里给出了两种寻找路线算法，广度优先遍历以及深度优先遍历。

分别利用了数据结构中的队列和栈。

4.主要/核心函数分析

Read_File

```
void Read_File(const string& filename){}
```

该函数用于从文件中读取地图数据，并将数据存储在`Maze`数组中。时间复杂度为$O(n^2)$，其中n是`Maze`数组的大小。

generate

```
void generate(){}
```

该函数用于生成起点和终点。它使用随机数生成起点和终点的坐标，并确保它们都是可行的路径（值为1）。时间复杂度为O(1)。

Bfs

```
void Bfs(){                             //进行广度优先搜索
    queue<Node> NodeQueue;
    NodeQueue.push(input);
    queue<string> Path;
    Path.push(ToString(input));
    Maze[input.x][input.y]=0;
    Node p;
    string pt,ct;
    while(!NodeQueue.empty()){
        p.x=NodeQueue.front().x;
        p.y=NodeQueue.front().y;
        pt=Path.front();
        NodeQueue.pop();
        Path.pop();
        for(int i=0;i<4;i++){
```

```
            p.x+=Direction[i].x;
            p.y+=Direction[i].y;
            if(p.x>=0&&p.x<30&&p.y>=0&&p.y<30&&Maze[p.x][p.y]==1){     //判断是否越界
                NodeQueue.push(p);
                Maze[p.x][p.y]=0;
                ct=pt+"->"+ ToString(p);
                Path.push(ct);
                if(p.x==output.x&&p.y==output.y){
                    cout<<ct<<endl;
                    return;
                }
            }
            p.x-=Direction[i].x;
            p.y-=Direction[i].y;
        }
    }
}
```

该函数实现了广度优先搜索算法，用于找到起点到终点的最短路径。它使用队列来进行搜索，并在找到终点时输出路径。时间复杂度为O(n^2)，其中n是`Maze`数组的大小。

Dfs

```
void Dfs(Node in){            //进行深度搜索算法
    Node p;
    if(in.x==input.x&&in.y==input.y){
        flag=1;
        string pt;
        while(!PathT.empty()){
            if(PathT.size()!=1)
                cout<<PathT.top()<<"->";
            else
                cout<<PathT.top();
            PathT.pop();
        }
        return;
    }
    p.x=in.x,p.y=in.y;
    for(int i=0;i<4;i++){     //遍历四个方向
        p.x+=Direction[i].x;
        p.y+=Direction[i].y;
        if(p.x>=0&&p.x<30&&p.y>=0&&p.y<30&&Maze[p.x][p.y]==1){
            Maze[p.x][p.y]=0;
            PathT.push(ToString(p));
            Dfs(p);
            if(flag!=1){
                Maze[p.x][p.y]=1;
                PathT.pop();
            }
        }
        p.x-=Direction[i].x;
        p.y-=Direction[i].y;
    }
}
```

该函数实现了深度优先搜索算法，用于找到起点到终点的路径。它使用递归来进行搜索，将经过的路径存进栈中，并在找到路径时输出。时间复杂度不确定，因为它是一个递归函数，其执行时间取决于迭代的次数，最坏的情况下是O(n^2)。

5.测试数据(规模,测试次数)

规模: 30*30的迷宫

测试次数:1

测试用例:见测试文件

6.运行结果

```
F:\data_structure\Must\question_2\cmake-build-debug\question_2.exe
1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 0
0 0 1 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 0 1 0 1 1 0 1 1 1 1
1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 0 1 1 0 0 1
1 0 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 0 0 0 0 0 1 1 0 0 0 1
1 1 0 1 0 1 1 0 0 1 0 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1
1 0 1 1 0 1 0 1 1 1 1 1 0 1 0 1 0 1 0 0 0 1 0 1 1 0 1 0 1 0
0 1 1 0 1 0 1 1 0 0 0 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 1 1 1
1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 0
1 0 1 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 1 0 1 1 0 1 1 0 1 0 1 1
1 0 1 1 0 1 0 1 0 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1 0 0 1 1 0 1
1 0 0 1 1 1 1 1 0 1 0 0 0 0 1 0 0 1 1 1 0 0 1 1 0 1 1 1 0 1 1
1 1 1 0 1 0 1 0 1 1 1 1 1 0 0 1 1 0 1 0 1 1 0 1 1 1 0 0 1 1 0
0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 0 0 1 0 1 0 1 0 0 0 1 1 1 0 1
1 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 1 1 1
1 1 0 0 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0 0 0 1 1 1 1 0 1
1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1
1 1 1 0 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1
0 0 1 1 0 0 1 1 1 1 0 1 0 1 1 0 0 1 0 0 1 0 1 1 0 0 1 0 1 0 0
1 1 1 0 1 1 1 0 0 0 1 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 1 0 1 1 1
1 0 0 1 1 0 0 1 1 1 0 1 1 0 0 1 1 0 1 0 0 1 1 1 1 0 1 1 0 1 0
1 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 0 1 1 1 0 0 1 0 1 0 0 0 1 1
0 1 1 1 0 0 1 1 1 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 1 1 1 0 1
1 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1
0 1 1 0 0 1 0 1 1 0 1 1 0 0 1 0 1 1 1 0 0 1 1 1 1 0 0 0 1 1 0
1 0 1 1 1 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1
1 1 0 0 0 1 0 1 1 0 1 1 1 0 0 1 1 1 0 0 1 0 0 0 1 0 1 0 0 1
1 0 0 1 0 1 0 1 0 1 0 0 0 1 1 0 0 1 1 0 1 1 0 1 0 0 1 1 1 1
1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 1 0 1 0 1
1 0 0 0 0 1 0 1 0 1 1 0 0 1 1 1 0 0 1 1 0 1 1 0 0 1 1 0 0 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 0 0 0 0 1 1 1 1
起点为(9,11),终点为(2,23)
Bfs
```

```
(9,11)->(9,12)->(9,13)->(9,14)->(9,15)->(8,15)->(7,15)->(6,15)->(5,15)->
(4,15)->(4,16)->(3,16)->(3,17)->(3,18)->(4,18)->(4,19)->(4,20)->(4,21)->
(4,22)->(4,23)->(5,23)->(5,24)->(6,24)->(6,25)->(7,25)->(7,26)->(8,26)->
(9,26)->(10,26)->(10,25)->(10,24)->(11,24)->(11,23)->(11,22)->(10,22)->
(9,22)->(9,23)->(8,23)->(7,23)->(7,22)->(7,21)->(8,21)->(8,20)->(9,20)->
(9,19)->(9,18)->(10,18)->(10,17)->(11,17)->(12,17)->(13,17)->(13,16)->
(14,16)->(14,15)->(15,15)->(16,15)->(16,16)->(16,17)->(16,18)->(16,19)->
(17,19)->(18,19)->(18,18)->(19,18)->(20,18)->(20,19)->(20,20)->(21,20)->
(22,20)->(22,21)->(23,21)->(23,22)->(23,23)->(24,23)->(24,24)->(24,25)->
(24,26)->(25,26)->(26,26)->(26,27)->(26,28)->(26,29)->(25,29)->(24,29)->
(24,28)->(23,28)->(23,27)->(22,27)->(21,27)->(21,26)->(21,25)->(20,25)->
(19,25)->(18,25)->(17,25)->(16,25)->(16,24)->(16,23)->(15,23)->(15,22)->
(15,21)->(15,20)->(14,20)->(13,20)->(13,21)->(13,22)->(13,23)->(13,24)->
(14,24)->(14,25)->(14,26)->(14,27)->(13,27)->(12,27)->(11,27)->(11,28)->
(10,28)->(10,29)->(9,29)->(8,29)->(8,28)->(7,28)->(6,28)->(5,28)->(4,28)->
(4,29)->(3,29)->(2,29)->(1,29)->(1,28)->(1,27)->(1,26)->(0,26)->(0,25)->
(0,24)->(1,24)->(1,23)->(2,23)
Dfs
(9,11)->(9,12)->(9,13)->(9,14)->(9,15)->(8,15)->(7,15)->(6,15)->(5,15)->
(4,15)->(4,16)->(3,16)->(3,17)->(3,18)->(4,18)->(4,19)->(4,20)->(4,21)->
(4,22)->(4,23)->(5,23)->(5,24)->(6,24)->(6,25)->(7,25)->(7,26)->(8,26)->
(9,26)->(10,26)->(10,25)->(10,24)->(11,24)->(11,23)->(11,22)->(10,22)->
(9,22)->(9,23)->(8,23)->(7,23)->(7,22)->(7,21)->(8,21)->(8,20)->(9,20)->
(9,19)->(9,18)->(10,18)->(10,17)->(11,17)->(12,17)->(13,17)->(13,16)->
(14,16)->(14,15)->(15,15)->(16,15)->(16,16)->(16,17)->(16,18)->(16,19)->
(17,19)->(18,19)->(18,18)->(19,18)->(20,18)->(20,19)->(20,20)->(21,20)->
(22,20)->(22,21)->(23,21)->(23,22)->(23,23)->(24,23)->(24,24)->(24,25)->
(24,26)->(25,26)->(26,26)->(26,27)->(26,28)->(26,29)->(25,29)->(24,29)->
(24,28)->(23,28)->(23,27)->(22,27)->(21,27)->(21,26)->(21,25)->(20,25)->
(19,25)->(18,25)->(17,25)->(16,25)->(16,24)->(16,23)->(15,23)->(15,22)->
(15,21)->(15,20)->(14,20)->(13,20)->(13,21)->(13,22)->(13,23)->(13,24)->
(14,24)->(14,25)->(14,26)->(14,27)->(13,27)->(12,27)->(11,27)->(11,28)->
(10,28)->(10,29)->(9,29)->(8,29)->(8,28)->(7,28)->(6,28)->(5,28)->(4,28)->
(4,29)->(3,29)->(2,29)->(1,29)->(1,28)->(1,27)->(1,26)->(0,26)->(0,25)->
(0,24)->(1,24)->(1,23)->(2,23)
进程已结束,退出代码0

```

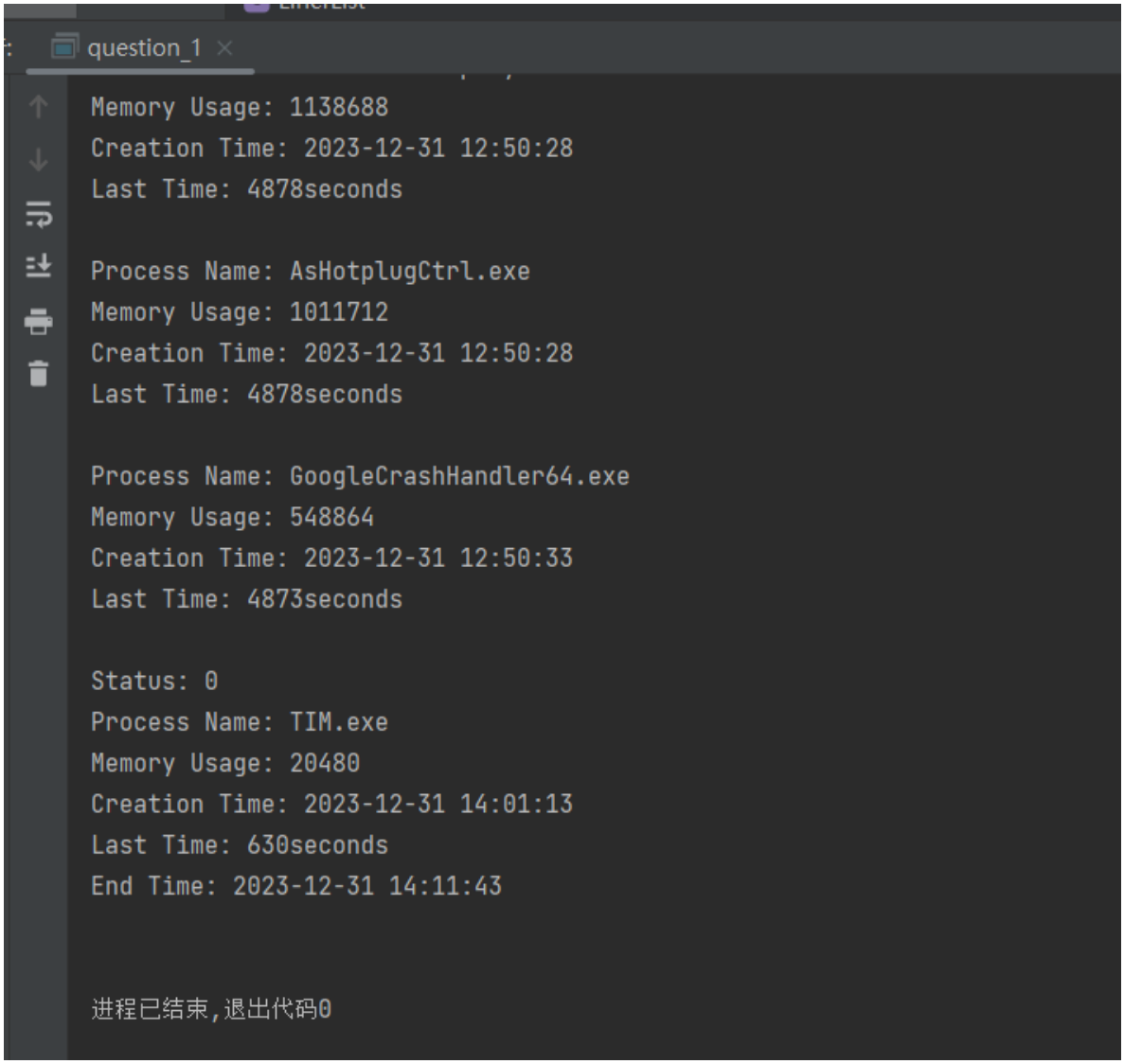
7.时间复杂度分析

Bfs时间复杂度为O(n^2)

Dfs时间复杂度最坏的情况下为O(n^2)

因此该程序时间复杂度为O(n^2)

8.结果截屏图片



9.心得体会

通过该题，对邻接矩阵的相关操作有了更深入的了解。同时，更加熟悉广度优先搜索以及深度优先搜索这两种常用的搜索算法，对栈和队列的使用更加得心应手。