1254：走出迷宫

题目描述

当你站在一个迷宫里的时候，往往会被错综复杂的道路弄得失去方向感，如果你能得到迷宫地图，事情就会变得非常简单。

假设你已经得到了一个n×m的迷宫的图纸，请你找出从起点到出口的最短路。

输入格式

第一行是两个整数n和m(1≤n,m≤100)，表示迷宫的行数和列数。

接下来n行，每行一个长为m的字符串，表示整个迷宫的布局。字符‘.’表示空地，‘#’表示墙，‘S’表示起点，‘T’表示出口。

输出格式

输出从起点到出口最少需要走的步数。

输入样例

```
3 3
S#T
.#.
...
```

输出样例

```
6
```

解析

二维地图，最短路径，套用广搜模板。

编码

```
#include<bits/stdc++.h>

using namespace std;

struct Node {
    int Row; //行数
    int Col; //列数
    int Step; //到达当前节点时，一共走了多少步
```

```
    //有参数的构造函数
    Node(int row, int col, int step) {
        this->Row = row;
        this->Col = col;
        this->Step = step;
    }

    //有参数的构造函数
    Node(int row, int col) {
        this->Row = row;
        this->Col = col;
    }

    //无参数的构造函数
    Node() {};
};

void BfsByQueue(Node start, Node end);

//地图表
char Map[101][101];
//访问记录表，所有存在这里的点，都不可以被二次使用
bool Vis[101][101];
//搜索队列，数组版
Node SearchArray[101 * 101];
//搜索队列，队列版
queue<Node> SearchQueue;
//地图的行列边界
int Row, Col;
//当前有多少个方向
const int FORWARD = 4;

int Forward[4][2] = {
        //x描述的纵向变化，y描述的是横向变化
       {-1, 0}, //上
       {1,  0}, //下
       {0,  -1}, //左
       {0,  1}, //右
};

//读取地图
void ReadMap() {
    Node start;
    Node end;
    for (int i = 0; i < Row; ++i) {
        cin >> Map[i];
    }
    for (int i = 0; i < Row; ++i) {
```

```
            for (int j = 0; j < Col; ++j) {
                if (Map[i][j] == 'S') {
                    start = Node(i, j, 0);
                }
                if (Map[i][j] == 'T') {
                    end = Node(i, j, 0);
                }
            }
        }
        BfsByQueue(start, end);
}

//检测节点是否有效，可以被存储
bool CheckNode(int row, int col) {
        //1、是否越界
        if (row >= 0 && col >= 0 && row < Row && col < Col) {
            //2、是否被访问过
            if (!Vis[row][col]) {
                if (Map[row][col] == '.' || Map[row][col] == 'T') {
                    return true;
                }
            }
        }
        return false;
}

//检测是否为终点
bool CheckEnd(int row, int col, Node end) {
        if (row == end.Row && col == end.Col) {
            return true;
        }
        return false;
}

//广度优先搜索（通过队列）
void BfsByQueue(Node start, Node end) {
        memset(Vis, false, sizeof(Vis));
        //对列中有值,清空队列
        while (!SearchQueue.empty()) {
            SearchQueue.pop();
        }
        //将起点放到队列
        SearchQueue.push(start);
        //启动搜索循环
        while (!SearchQueue.empty()) {
            //取出第一个点
            Node curNode = SearchQueue.front();
            //从队列中删除
            SearchQueue.pop();
```

```
            for (int i = 0; i < FORWARD; ++i) {
                int newRow = curNode.Row + Forward[i][0];
                int newCol = curNode.Col + Forward[i][1];
                if (CheckNode(newRow, newCol)) {
                    //将当前节点存入队列
                 SearchQueue.push(Node(newRow, newCol, curNode.Step + 1)
                    Vis[newRow][newCol] = true;
                }
                if (CheckEnd(newRow, newCol, end)) {
                    cout << curNode.Step + 1 << endl;
                    return;
                }
            }
        }
}

int main() {
    cin >> Row >> Col;
    ReadMap();
    return 0;
}
```