上班之路

知识点BFS 搜索广搜
时间限制：1s 空间限制：256MB 限定语言：不限

题目描述：

Jungle生活在美丽的蓝鲸城，大马路都是方方正正，但是每天马路的封闭情况都不一样。

地图由以下元素组成：

1）"." — 空地，可以达到;

2）"*" — 路障，不可达到;

3）"S" — Jungle的家;

4）"T" — 公司.

其中我们会限制Jungle拐弯的次数，同时Jungle可以清除给定个数的路障，现在你的任务是计算Jungle是否可以从家里出发到达公司。

输入描述：

输入的第一行为两个整数t,c（0<=t,c<=100），t代表可以拐弯的次数，c代表可以清除的路障个数。
输入的第二行为两个整数n,m（1<=n,m<=100）,代表地图的大小。
接下来是n行包含m个字符的地图。n和m可能不一样大。
我们保证地图里有S和T。

输出描述：

输出是否可以从家里出发到达公司，是则输出YES，不能则输出NO。

示例1

输入:

2 0
5 5
..S..
****.
T....
****.
.....

输出:

YES

示例2

输入:

1 2
5 5
.*S*.
*****
..*..
*****
T....

输出:

NO

说明:

该用例中，至少需要拐弯1次，清除3个路障，所以无法到达

解题思路:

通过回溯法求出所有可能的路径。

```
public class Main{

    public static char[][] map;        //地图
    public static int t;       //转弯次数
    public static int c;       //路障个数
    public static int n;       //地图行数
    public static int m;       //地图列数
```

```
public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        t = sc.nextInt();
        c = sc.nextInt();
        n = sc.nextInt();
        m = sc.nextInt();
        sc.nextLine();

        map = new char[n][m];
        char[][] mapCopy = new char[n][m];

        int x = 0;
        int y = 0;
        for(int i=0; i<n; i++){
            String string = sc.nextLine();
            for(int j=0; j<m; j++){
                map[i][j] = string.charAt(j);
                mapCopy[i][j] = map[i][j];
                if(map[i][j] == 'S'){
                    x = i;
                    y = j;
                }
            }
        }

        if(toCompany( mapCopy, x, y, new ArrayList<>(), 0, 0) == 1){
            System.out.println("YES");
        }else {
            System.out.println("NO");
        }

}

/**
 *
 * @param newMap            地图，用来记录走过的行程
 * @param x                 横坐标
 * @param y                 纵坐标
 * @param list              走过的坐标集合
 * @param turn              转弯的次数
 * @param barricade         路过路障的次数
 * @return
```

```
 */
    public static int toCompany(char[][] newMap, int x, int y, List<int[]> list, int turn, int barricade){

        if(list.size() > 1){       //至少走过两个格子才能判断是否转弯
            int[] ints = list.get(list.size()-2);     //获取路过的倒数第二个格子
            if(ints[0] != x && ints[1] != y){     //如果横纵坐标没有相同的，则表示转过弯
                turn ++;
            }
        }

        list.add(new int[]{x, y});          //走过的格子

        if(turn > t){           //转弯次数大于 t，则不符合，返回
            return 0;
        }

        if(newMap[x][y] == '*'){       //记录路障的个数
            barricade ++;
        }

        if(barricade > c){          //路障的个数大于 c，则不符合，返回
            return 0;
        }

        if(newMap[x][y] == 'T'){       //到达公司完成路程
            return 1;
        }

        newMap[x][y] = 'X';         //走过的地方记录为 X

        if(x>0){       //向上
            if(newMap[x-1][y] != 'X'){          //走过的格子不再走
                if(toCompany( newMap,x-1, y, list, turn, barricade) == 1){
                    return 1;
                }else {
                    newMap[x-1][y] = map[x-1][y];     //不符合要求的路程需要恢复
                    list.remove(list.size()-1);         //走过的格子需要剔除
                }
            }
        }

        if(x<n-1){          //向下
            if(newMap[x+1][y] != 'X'){          //走过的格子不再走
```

```
                if(toCompany( newMap,x+1, y, list, turn, barricade) == 1){
                    return 1;
                }else {
                    newMap[x+1][y] = map[x+1][y];     //不符合要求的路程需要恢复
                    list.remove(list.size()-1);         //走过的格子需要剔除
                }
            }
        }

        if(y>0){              //向左
            if(newMap[x][y-1] != 'X'){          //走过的格子不再走
                if(toCompany( newMap, x, y-1, list, turn, barricade) == 1){
                    return 1;
                }else {
                    newMap[x][y-1] = map[x][y-1];     //不符合要求的路程需要恢复
                    list.remove(list.size()-1);         //走过的格子需要剔除
                }
            }
        }

        if(y<m-1){          //向右
            if(newMap[x][y+1] != 'X'){          //走过的格子不再走
                if(toCompany( newMap, x, y+1, list, turn, barricade) == 1){
                    return 1;
                }else {
                    newMap[x][y+1] = map[x][y+1];     //不符合要求的路程需要恢复
                    list.remove(list.size()-1);         //走过的格子需要剔除
                }
            }
        }

        return 0;
    }

}
```