1216：红与黑

题目描述

有一间长方形的房子，地上铺了红色、黑色两种颜色的正方形瓷砖。你站在其中一块黑色的瓷砖上，只能向相邻的黑色瓷砖移动。请写一个程序，计算你总共能够到达多少块黑色的瓷砖。

输入

包括多组数据。每组数据的第一行是两个整数W和H，分别表示x方向和y方向瓷砖的数量。W和H都不超过20。在接下来的H行中，每行包括W个字符。每个字符表示一块瓷砖的颜色，规则如下：

1）‘.’：黑色的瓷砖；
2）‘#’：白色的瓷砖；
3）‘@’：黑色的瓷砖，并且你站在这块瓷砖上。该字符在每组数据中唯一出现一次。

当在一行中读入的是两个零时，表示输入结束。

输出

对每组数据，分别输出一行，显示你从初始位置出发能到达的瓷砖数(记数时包括初始位置的瓷砖)。

输入样例

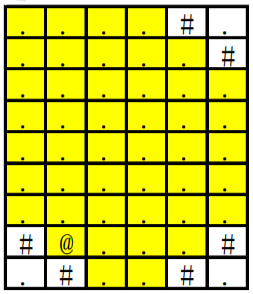
```
6 9
. . . . # .
. . . . . #
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
# @ . . . #
. # . . # .
0 0
```

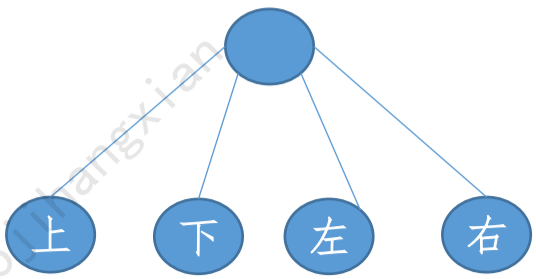
输出样例

45

分析

首先明确题意：这道题是求总共能够到达的黑色瓷砖数量。如下图所示，这道题的决策树也是非常的简单，即每一层都按照上下左右去进行移动判断即可：





图中黄色的部分就是我们所要求的答案，只要看懂了上面的这张图，解题的方法就非常清晰了。

我们只需要从起点开始，持续的调用DFS，直到无路可走，那么就肯定将全部的瓷砖走过。

特别强调，在回溯的时候，我们不需要把已经走过的瓷砖标记为未访问。因为，走过的瓷砖已经被我们统计过了，无须重复计算。

最后，如何统计走过的瓷砖数量？很简单，只需要在每次进入DFS的时候累加即可。

本题有两个注意点：

1、存在多组测试数据，注意重置初始数据。

2、行列顺序与常规题目相反

编码

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

using namespace std;
//四个移动方向
int const FORWARD_NUM = 4;
//边界
int R, S;
//地图
char Map[21][21];
//记录当前节点是否行走过
bool Vis[21][21];
//最大经过数量
int MaxNum = 0;
//四个个移动方向
int Forward[4][2] = { //四方向，二维（行，列）
        {-1, 0},//上
        {1,  0},//下
        {0,  -1},//左
        {0,  1},//右
};

//判断指定的目标点是否可以行走
bool Check(int x, int y) {
    //1、是否越界
    if (x >= 0 && y >= 0 && x < S && y < R) {
        //2、未访问过
        if (!Vis[x][y]) {
            //3、黑色瓷砖
            if (Map[x][y] == '.') {
                return true;
            }
        }
    }
    return false;
}

//深搜代码
void Dfs(int sx, int sy) {
    //每行走一个新的地板，都要记录一下。
    MaxNum++;
    for (int i = 0; i < FORWARD_NUM; ++i) {
        //新的xy坐标
        int newX = sx + Forward[i][0];
        int newY = sy + Forward[i][1];
        if (Check(newX, newY)) {
```

```
            Vis[newX][newY] = true;
            //执行深搜
            Dfs(newX, newY);
        }
    }
}

void Reset() {
    memset(Vis, false, sizeof(Vis));
    MaxNum = 0;
}

void Read() {
    //读入地图规模
    while (cin >> R >> S) {
        //重置数据
        Reset();
        if (R + S == 0)
            return;
        //读入地图
        for (int j = 0; j < S; ++j) {
            cin >> Map[j];
        }

        int startX;
        int startY;
        //寻找起点
        for (int i = 0; i < S; ++i) {
            for (int j = 0; j < R; ++j) {
                if (Map[i][j] == '@') {
                    startX = i;
                    startY = j;
                }
            }
        }
        Vis[startX][startY] = true;
        Dfs(startX, startY);
        cout << MaxNum << endl;
    }
}

int main() {
    Read();
    return 0;
}
```