HDU3085

题目描述（HDU3085）：小明做了一个可怕的噩梦，梦见他和他的朋友分别被困在一个大迷宫里。更可怕的是，在迷宫里有两个魔鬼，他们会杀人。小明想知道他能否在魔鬼找到他们之前找到他的朋友。小明和他的朋友可以朝四个方向移动。在每秒内，小明都可以移动 3 步，他的朋友可以移动 1 步。魔鬼每秒都会分裂成几部分，占据两步之内的网格，直到占据整个迷宫。假设魔鬼每秒都会先分裂，然后小明和他的朋友开始移动，如果小明或他的朋友到达一个有魔鬼的格子，就会死（新的魔鬼也可以像原来的魔鬼一样分裂）。

输入：输入以整数 T 开头，表示测试用例的数量。每个测试用例的第 1 行都包含两个整数 n 和 m（1<n,m<800），表示迷宫的行数和列数。接下来的 n 行，每行都包含 m 个字符，字符“.”表示一个空地方，所有人都可以走；“X”表示一堵墙，只有人不能走；“M”表示小明；“G”表示小明的朋友；“Z”表示魔鬼，保证包含一个字母 M、一个字母 G 和两个字母 Z。

输出：如果小明和他的朋友能够见面，则单行输出见面的最短时间，否则输出-1。

输入样例

```
3
5 6
XXXXXX
XZ..ZX
XXXXXX
M.G...
......
5 6
XXXXXX
XZZ..X
XXXXXX
M.....
..G...

10 10
..........
..X.......
..M.X...X.
X.........
.X..X.X.X.
.........X
..XX....X.
X....G...X
...ZX.X...
...Z..X..X
```

输出样例

```
1
1
-1
```

题解：已知起点（小明）、终点（小明的朋友），两者在中间遇到即见面成功，可以采用双向广度优先搜索。使用双向广度优先搜索时需要创建两个队列，分别从小明的初始位置、小明的朋友的初始位置开始，轮流进行广度优先搜索。在本题中，小明每次都可以移动 3 步，小明的朋友每次都可以移动 1 步，因此在每一轮循环中，小明都扩展 3 层，小明的朋友都扩展 1 层。在扩展时，需要检查与魔鬼的距离，判断该节点是否会被魔鬼波及。

1. 算法设计

（1）定义两个队列 q[0]、q[1]，分别将小明的起始位置 mm 和小明的朋友的起始位置 gg 入队，秒数 step=0。

（2）如果两个队列均不空，则执行步骤 3，否则返回-1。

（3）step++，小明扩展 3 步，如果搜索到小明的朋友的位置，则返回 true；否则小明的朋友扩展 1 步，如果搜索到小明的位置，则返回 true。如果在两个方向搜索时发现有一个方向为 true，则返回秒数 step，否则执行步骤 2。

2. 算法实现

```
int solve(){
    while(!q[0].empty()) q[0].pop();//清空队列
    while(!q[1].empty()) q[1].pop();
    q[0].push(mm);
    q[1].push(gg);
    step=0;
    while(!q[0].empty()&&!q[1].empty()){
        step++;
        if(bfs(0,3,'M','G')||bfs(1,1,'G','M'))
            return step;
    }
    return -1;
}

bool check(int x,int y){
    if(x<0||y<0||x>=n||y>=m||mp[x][y]=='X') return false;
    for(int i=0;i<2;i++)
        if(abs(x-zz[i].x)+abs(y-zz[i].y)<=2*step) return false;
    return true;
}
int bfs(int t,int num,char st,char ed){
    queue<node> que=q[t];
    for(int k=0;k<num;k++){
        while(!que.empty()){
            node now=que.front();
            que.pop();
            q[t].pop();
            if(!check(now.x,now.y)) continue;
            for(int j=0;j<4;j++){
                int fx=now.x+dir[j][0];
                int fy=now.y+dir[j][1];
                if(!check(fx,fy)||mp[fx][fy]==st) continue;
                if(mp[fx][fy]==ed)
                    return true;
                mp[fx][fy]=st;
                q[t].push(node(fx,fy));
            }
        }
        que=q[t];
    }
    return false;
}
```