Mr. Savior

Limits: 1s, 1.0 GB

"**Stop War, Save Life**" - a life-changing motto proposed by the "**Peace Want Society**".
The head of this society named **Mr. Savior** wants to establish this motto all over the world. He takes the responsibility. The evil guys set up a challenge for Mr. Savior. Those evil guys provide a map. The map is actually a grid. Each of the cells is either empty or contains an obstacle. Some of the obstacles are made of sands and some of them are made of concrete. Obstacles made of sands can easily be breakable whereas obstacles made of concrete cannot be breakable. Those evil guys throw Mr. Savior in an empty cell within the grid. He has to get out from that grid in order to save the war affected people. Mr. Savior can move towards left, right, top, bottom from the current cell.

He asked you to figure out minimum number of obstacles made by sands he needs to break in order to get out from the grid or determine if it is impossible.

Input

Inputs start with an integer **T** denotes the number of test cases you have to perform.
Each test case consists of two integers **N, M** that denotes the size of the grid. Each of the next N lines contains M characters. There are **4**types of characters -

' . ' – an empty cell.
'#' – Obstacle made of sands.
'*' – Obstacle made of concrete.
'$' - The location where the evil guys throw Mr. Savior. This symbol will present in the grid exactly once.

Constraints

1 ≤ **T** ≤ 5000
1 ≤ **M,N** ≤ 100

Output

For each test case, you need figure out the minimum number of obstacles made by sands he needs to break in order to get out from the grid or report him it's impossible. If it is impossible you have to print "**Impossible**". Printing format will be "**Case X: Y**" where **X** is the case number and **Y** is the desired result. See samples for further clarification.

Samples

Input	Output
1 5 5 *.*** *#*.* *.$.* *.*#* ***.*	

```cpp
/// https://toph.co/p/mr-savior
#include<bits/stdc++.h>
using namespace std;
char s[105][105];
int fx[] = {-1,+0,+0,+1};
int fy[] = {+0,-1,+1,+0};
int n,m,cost[105][105], vis[105][105];
struct dt{
    int x,y,w;
    bool operator < (const dt&p)const{
        return p.w<w;
    }
};
int BFS(){
    for(int i=0; i<101; i++){
        for(int j=0; j<101; j++){
            cost[i][j] = 10000000; vis[i][j] = 0;
        }
    }
    int sx,sy;
    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            if(s[i][j]=='$'){
                sx = i, sy = j;
                break;
            }
        }
    }
    priority_queue<dt>qq;
    dt p; p.x = sx, p.y = sy, p.w = 0;
    qq.push(p);
    cost[sx][sy] = 0;

    while(!qq.empty()){
        dt u; u = qq.top();
        qq.pop();
        int x,y,w; x = u.x, y = u.y, w = u.w;
        if(vis[x][y]==1)continue;

        for(int k=0; k<4; k++){
            int tx = x+fx[k];
            int ty = y+fy[k];
            if(tx<0 || tx>=n || ty<0 || ty>=m){
                return w;
            }else{
                if(s[tx][ty]!='*'){
                    int nv=w+1;
                    if(s[tx][ty]=='.')nv=w;
                    if(s[tx][ty]=='#')nv=w+1;
                    if(nv<cost[tx][ty]){
                        dt z; z.x = tx, z.y = ty, z.w = nv;
                        cost[tx][ty] = nv;
                        qq.push(z);
                    }
                }
            }
        }
        vis[x][y]=1;
    }
    return -1;
}
int main(){
    int t; scanf("%d",&t);
    for(int ks=1; ks<=t; ks++){
        scanf("%d %d",&n,&m);
        for(int i=0; i<n; i++) scanf("%s",s[i]);
        int ans = BFS();
        if(ans==-1)printf("Case %d: Impossible\n",ks);
        else printf("Case %d: %d\n",ks,ans);
    }
    return 0;
}
```