# 959. Regions Cut By Slashes

Runtime

4 ms | Beats 84.82%

Memory

12.12 MB | Beats 66.40%

```
/*
   Euler's theorem for planar graphs
   Time complexity: O(α(n^2)n^2)
   Space complexity: O(n^2)
*/
class DSU{
    public:
        int nb_dots;
        int V;
        int E;
        std::vector<int> parent;
        std::vector<int> group;
    public:
        DSU(int n){
            E=4*n;
            nb_dots=n+1;
            V=nb_dots*nb_dots;
            parent.resize(V);
            group.resize(V,1);
            for(int i=0;i<V;++i) parent[i]=i;
        }
        int find(int p){
            int root=p;
            while(root!=parent[root]) root=parent[root];

            // Path compression
            while(p!=root){
                int next=parent[p];
                parent[p]=root;
                p=next;
            }
            return root;
        }
        int find_rec(int p){
            if(p==parent[p]) return p;
            return parent[p]=find_rec(parent[p]); // Path compression
        }
        void unify(int p,int q){
            int parent_p=find(p);
            int parent_q=find(q);
            if(parent_p==parent_q) return;

            if(group[parent_p]<group[parent_q]){
                group[parent_q]+=group[parent_p];
                group[parent_p]=1;
                parent[parent_p]=parent_q;
            }
            else{
                group[parent_p]+=group[parent_q];
                group[parent_q]=1;
                parent[parent_q]=parent_p;
            }
        }
};
```

```
class Solution {
public:
    int regionsBySlashes(std::vector<std::string>& grid) {
        int n=grid.size();

        DSU dsu=DSU(n);

        int nb_dots=dsu.nb_dots;
        int V=dsu.V;
        int E=dsu.E;
        for (int i=0;i<nb_dots;i++){
            for (int j=0;j<nb_dots;j++){
                if (i==0 || j==0 || i==nb_dots-1 || j==nb_dots-1){
                    int point=i*nb_dots+j;
                    if(point!=0) dsu.unify(0,point);
                }
            }
        }
        for(int i=0;i<n;++i){
            std::string s=grid[i];
            for(int j=0;j<s.size();++j){
                if(s[j]==' ') continue;
                int point1,point2;
                if(s[j]=='/'){
                    point1=i*nb_dots+(j+1);
                    point2=(i+1)*nb_dots+j;
                }
                else if(s[j]=='\\'){
                    point1=i*nb_dots+j;
                    point2=(i+1)*nb_dots+(j+1);
                }
                E++;
                dsu.unify(point1,point2);
            }
        }

        //C1: The number of connected components formed by the X vertices
        int C1=0,X=0;
        for(auto& g: dsu.group){
            if(g>1) {
                C1++;
                X+=g;
            }
        }
        // C: Total number of components
        // -1: Note that if G is connected, then C=1 (already computed)
        int C=C1+(V-X)-1;

        //Formula: V-E+F=C+1
        int F=E-V+C+1;

        return F;
    }
};
```