[1579] Remove Max Number of Edges to Keep Graph Fully Traversable

```
class UnionFind{
    public:
      std::vector<int> parent;
      //std::vector<int> groups;
   public:
      UnionFind(int n){
         //groups.resize(n+1,1);
         parent.resize(n+1,0);
         for (int i=1 ;i<=n;++i) parent[i]=i;
      }

      int find(int p){
         int root=p;
         while(root!=parent[root])
            root= arent[root];

         while (p!=root){
            int next=parent[p];
            parent[p]=root;
            p=next;
         }

         return root;
      }

      bool unify(int p,int q){
         int parent_p=find(p);
         int parent_q=find(q);

         if (parent_p==parent_q) return false;
         parent[parent_p]=parent_q;

         /*if (groups[parent_p]<groups[parent_q]){
            groups[parent_q]+=groups[parent_p];
            groups[parent_p]=1;
            parent[parent_p]=parent_q;
         }
         else{
            groups[parent_p]+=groups[parent_q];
            groups[parent_q]=1;
            parent[parent_q]=parent_p;
         }*/
         return true;
      }
};
```

```
class Solution {
    public:
        int maxNumEdgesToRemove(int n, std::vector<std::vector<int>>& edges){

            std::sort(edges.begin(),edges.end(),
                            [](const std::vector<int>& e1,const std::vector<int>& e2){
                                    return e1[0]>e2[0];
                            }
            );

            UnionFind Alice=UnionFind(n);
            UnionFind Bob=UnionFind(n);

            int bob=0,alice=0,both=0;
            for(auto& edge: edges){
                if(edge[0]==3){
                    if(Alice.unify(edge[1],edge[2]) && Bob.unify(edge[1],edge[2])) both++;
                }
                else if(edge[0]==2){
                    if(Bob.unify(edge[1],edge[2])) bob++;
                }
                else{
                    if(Alice.unify(edge[1],edge[2])) alice++;
                }
            }

               // Graph must be traversable for Alice and bob => Graph should be connected => E=V-1
               // #edges for both+#edges of bob == n-1
               // #edges for both+#edges of Alice == n-1
               // #edges to remove=E-(#edges for both+#edges of Alice+#edges of bob)

            return both+alice==n-1&&both+bob==n-1?edges.size()-both-alice-bob:-1;
        }
};
```