```
#include<iostream>
#include<vector>
#include<algorithm>
#define ll long long int
#define rep(n) for(int i=0;i<n;i++)

//(1)
class UnionFindTree
{
  public:
    std::vector<ll> par;
    std::vector<ll> rank;
    std::vector<ll> size;
    //(2)
    UnionFindTree(ll n)
    {
      par.resize(n);
      rep(n)
      {
        par[i]=i;
      }
      rank.resize(n);
      rep(n)
      {
        rank[i]=0;
      }
      size.resize(n);
      rep(n)
        {
        size[i]=1;
      }
      }
    //(3)
    ll Find(ll x)
    {
      if(par[x]==x)
      {
        return x;
      }
      par[x]=Find(par[x]);
      return par[x];
    }
    //(4)
    void Union(ll x,ll y)
    {
      x=Find(x);
      y=Find(y);
      if(x==y)
      {
        return;
      }
      //(5)
      if(rank[x]<rank[y])
      {
        par[x]=y;
        size[y]+=size[x];
      }
      else
      {
        par[y]=x;
        size[x]+=size[y];
        if(rank[x]==rank[y])
        {
          rank[x]++;
        }
      }
    }
    //(6)
    bool Same(ll x,ll y)
    {
```

```cpp
    return Find(x)==Find(y);
  }

  //(7)
  ll Size(ll x)
  {
    return size[Find(x)];
  }
};

int main()
{
  ll n,m;
  std::cin>>n>>m;
  UnionFindTree *uft=new UnionFindTree(n);
  rep(m)
  {
    ll x,y;
    std::cin>>x>>y;
    x--;
    y--;
    uft->Union(x,y);
  }
  //(8)
  std::vector<ll> group(n);
  rep(n)
  {
    group[i]=uft->Find(i);
  }
  //(9)
  std::sort(group.begin(),group.end());
  group.erase(std::unique(group.begin(),group.end()),group.end());
  ll group_num=group.size();
  //(10)
  ll group_max=0;
  rep(group_num)
  {
    ll val=uft->Size(group[i]);
    if(val>group_max)
    {
      group_max=val;
    }
  }
  std::cout<<group_num<<" "<<group_max<<std::endl;
}
```