GRAPHS

Day 4
(DSU and MST)

Youtube link :

https://youtu.be/QHCGQ371eXs

Contents:

1. **DSU and related operations**
2. **Applications of DSU**
3. **Kruskal's Algorithm for MST**

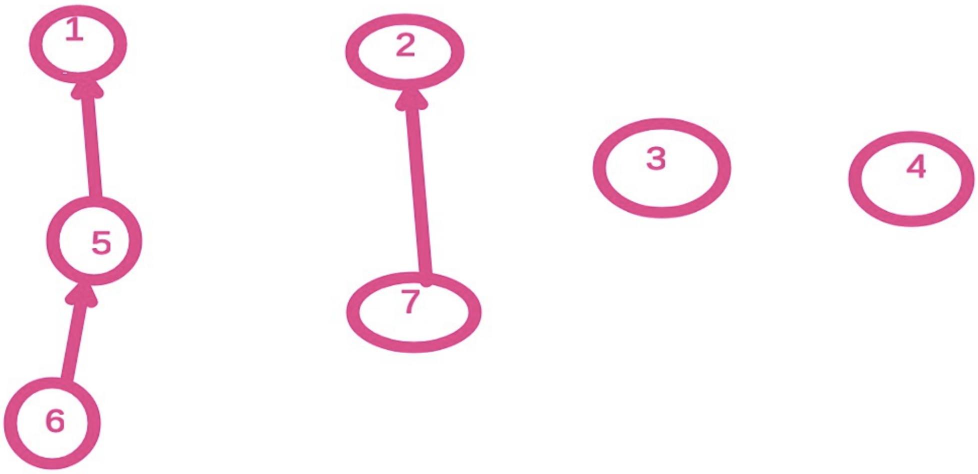
DSU (Disjoint Set Union)

- A data structure used for performing the union of disjoint sets very efficiently
- **Disjoint sets**: Any 2 sets are disjoint when their intersection is an empty set.
  Eg. A = { 1, 2, 6}
  B = {4, 5 }
  A intersection B = ϕ (empty set)
  So, A and B are disjoint sets

3 operations:

1. make_set() ; // To create DSU initially
2. find_set(); // Find the set to which an element belongs (To find representative of a set)
3. union_set(); // Take Union of 2 sets or merge 2 sets

Representative of a set is the topmost element :
(Parent of representative is that element itself)



Naive implementation

```
const int MAX=100005;
int par[MAX];

void make_set(int n)
{
	for(int i=0; i<n; i++)
	{
		par[i]=i;
	}
}
```

```
int find_set(int x)
{
  if( par[x] == x)
  {
        return x;
  }
  return find_set(par[x]);
}
```

// You can also write this in a while loop like:
(Both will work the same way, use anyone which you like)

```
int find_set(int x)
{
   while( par[x] != x)
  {
        x=par[x];
  }
  return x;
}
```

```
void union_set(int a, int b)
{
    int p1 = find_set(a);
// representative of set of a
```

```
     int p2 = find_set(b);
     // representative of set of b
    if ( p1 ! = p2)
    {
       par[p1]=p2;
    }
}
```

Time complexity: O(d)
Where d = depth (distance from the representative)

Faster Implementation

1. **Path compression in find_set():**

(Store the representative in parent directly)

```
int find_set(int x)
{
  if( par[x] == x)
  {
       return x;
  }
  par[x]=find_set(par[x]);
  return par[x];
}
```

It reduces time complexity to O (log N) approximately

2. Union by Rank in union_set() :

- Create another array for size of set. (rnk[ ])
- Make smaller set as a child of larger set.

```
int rnk[ 100005];
void make_set(int n)
{
    for(int i=0; i<n; i++)
    {
        par[i] = i;
        rnk[i] = 1;
    }
}


void union_set(int a, int b)
{
     int p1 = find_set(a);
     int p2 = find_set(b);
     if ( p1 == p2)
         return;
     if (rnk[p1] > rnk[p2])
    {
        par[p2]=p1;
        rnk[p1] = rnk[p1] + rnk[p2];
    }
else
  {
```

```
        par[p1] = p2;
        rnk[p2]  = rnk[p1] + rnk[p2];
    }
}
```

Time Complexity of find_set() and union_set(), when you use both path compression and union by rank:
O(α(N))
where α(N) = Inverse Ackermann Function
(Approximately , it is equal to O(1))

Try this problem:
https://www.hackerrank.com/challenges/merging-communities/problem

Solution:

```
const int MAX=100005;
int par[MAX];
int rnk[MAX];
void make_set(int n)
{
    for(int i=1; i<=n; i++)
    {
        par[i] = i;
        rnk[i] = 1;
    }
}
```

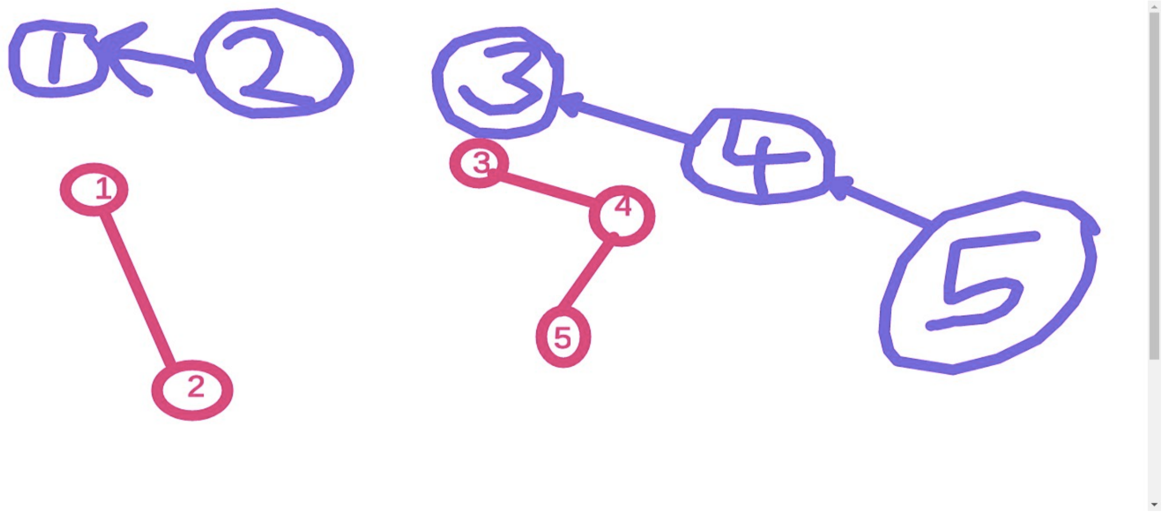
```
int find_set(int x)
{
  if( par[x] == x)
  {
       return x;
  }
  par[x]=find_set(par[x]);
  return par[x];
}

void union_set(int a, int b)
{
     int p1 = find_set(a);
     int p2 = find_set(b);
     if ( p1 == p2)
         return;
     if (rnk[p1] > rnk[p2])
    {
        par[p2]=p1;
        rnk[p1] = rnk[p1] + rnk[p2];
    } else
  {
       par[p1] = p2;
       rnk[p2]  = rnk[p1] + rnk[p2];
   }
}
```

```cpp
int main()
{
   int n,q;
    cin>> n >> q;
    make_set(n);
    while(q--)
   {
     char ch;
    cin>>ch;
   if(ch=='M')
   {
   int a,b;
   cin>>a>>b;
   union_set(a,b);
    }
else {
   int a;
   cin>>a;
   int representative = find_set(a);
    cout<<rnk[representative]<<'\n';
   }
 }
return 0;
}
```

Applications of DSU

1. Finding the number of components in a graph



Try:
https://cses.fi/problemset/task/1676

Solution:

```
int mx=1;

void union_set(int a, int b)
{
     int p1 = find_set(a);
     int p2 = find_set(b);
     if ( p1 == p2)
         return;
     if (rnk[p1] > rnk[p2])
    {
        par[p2]=p1;
        rnk[p1] = rnk[p1] + rnk[p2];
        mx=max(mx,rnk[p1]);
    } else
```

```
  {
       par[p1] = p2;
       rnk[p2]  = rnk[p1] + rnk[p2];
        mx=max(mx,rnk[p2]);
   }
}

int components = n;
cin>> n >> m;
make_set(n);
int u,v;
while(m--)
{
cin>>u>>v;
u--;
v--;
if( find_set(u) == find_set(v) )
{
    cout<<components<< " "<<mx<<'\n';
    continue;
}
union_set(u,v);
components--;
    cout<<components<< " "<<mx<<'\n';
}
```

2. Finding a cycle in undirected graph

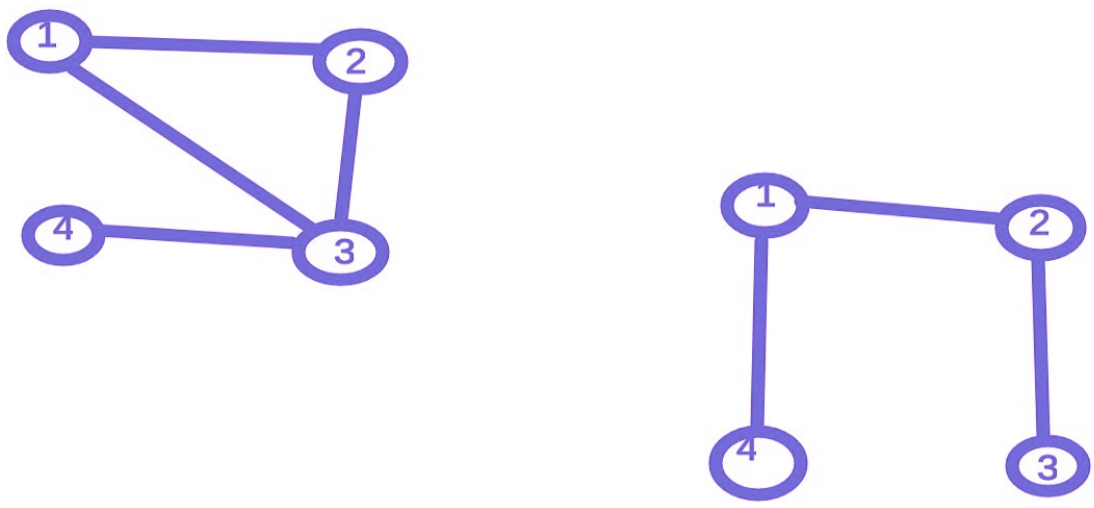
Code:

```
make_set(n);
for( auto e: edges)
{
     int u=e.first;
     int v=e.second;
    if ( find_set(u) == find_set(v) )
     {
        cout<<"Cycle found";
        break;
     }
      union_set(u,v);
}
```

MST (Minimum Spanning Tree)

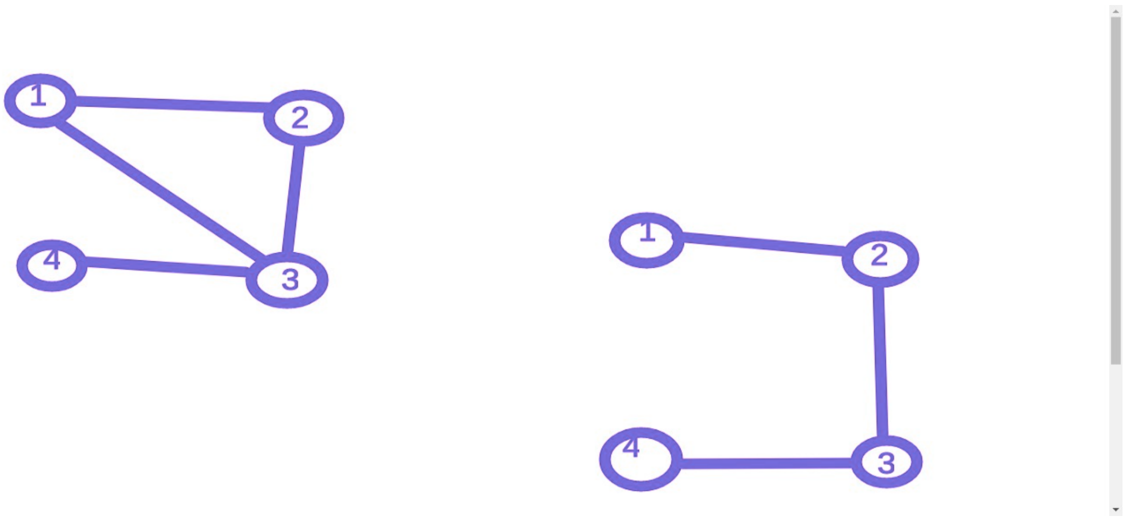
Spanning Tree of a graph :
A spanning tree is a tree containing all nodes of the given graph and is a subgraph of the given graph.

Example of not a spanning tree:



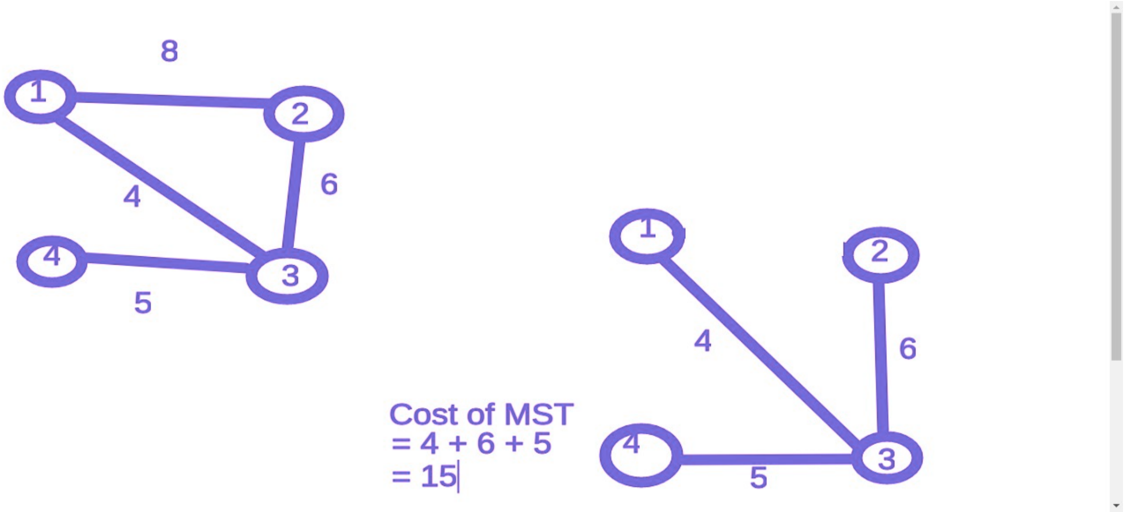
(Since edge (1,4) is not present in the original graph)

Example of a Spanning Tree:



Minimum Spanning Tree:

We need to find a spanning tree whose sum of edges is minimum.



Kruskal’s Algorithm

(A Greedy Algorithm)

1. Sort edges in increasing order of weights
2.

```
 make_set(n);
int sum=0;
int cnt=0;
for (auto e: edges)
{
    int u=e.first;
           int v=e.second.first;
           int w=e.second.second;
           if ( find_set(u) == find_set(v) )
           {
       continue;
           }
           union_set(u,v);
           sum = sum + w;
```

```
            cnt++;
    }

if( cnt == n-1)
    cout<<sum<<'\n';
else
{
     cout<<"NO MST Possible";
}
```

Try to solve:

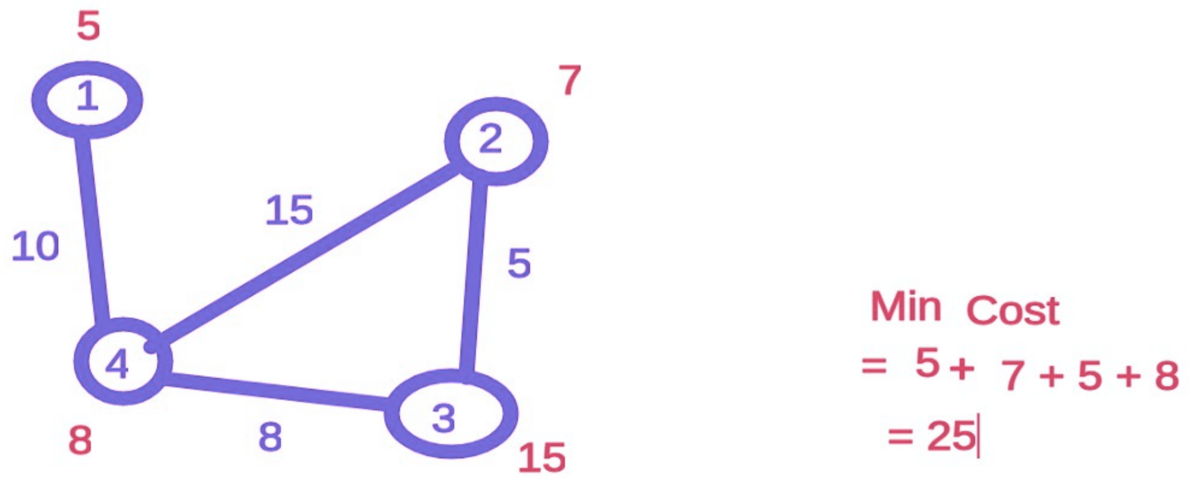
https://cses.fi/problemset/task/1675

Q. Find a spanning tree of a given graph with minimum products of weights?

Solution:

- Find MST with Kruskal's algorithm and print the products of its weights.

Q. Given n cities and m pipe connections between those cities . For digging a well in a city i, it costs C[i]. Find minimum cost to provide water supply to all.
Example:



Solution Approach:

Consider a imaginary node 0 and connect it to all remaining nodes with a edge cost of C[i].

Now, find the MST of this new graph.