CSE 421

Greedy Algorithms / Minimum Spanning Tree

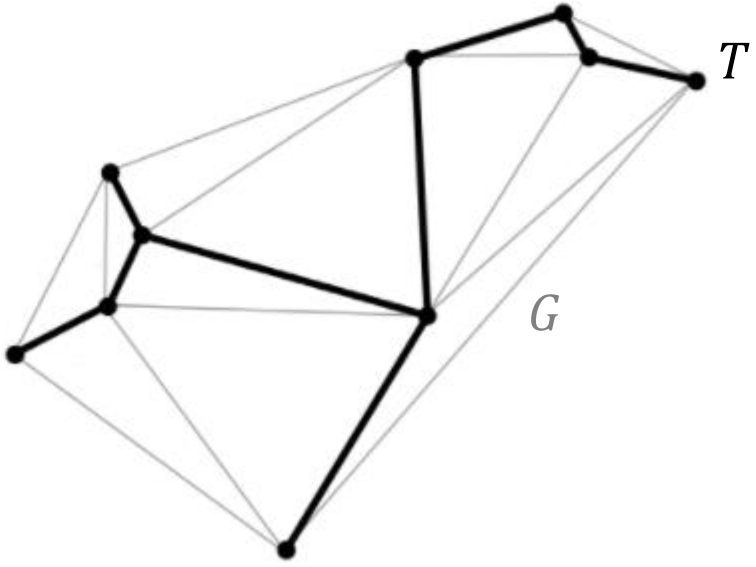
Yin Tat Lee

Spanning Tree

Given a connected undirected graph $G = (V, E)$.

We call T is a spanning tree of G if

- All edges in T are from E.
- T includes all of the vertices of G.



Why spanning tree?

Many problems is easy for tree.

General framework:

- Approximate the graph by a tree.
- Solve the problem on a tree.

We have covered different tree:

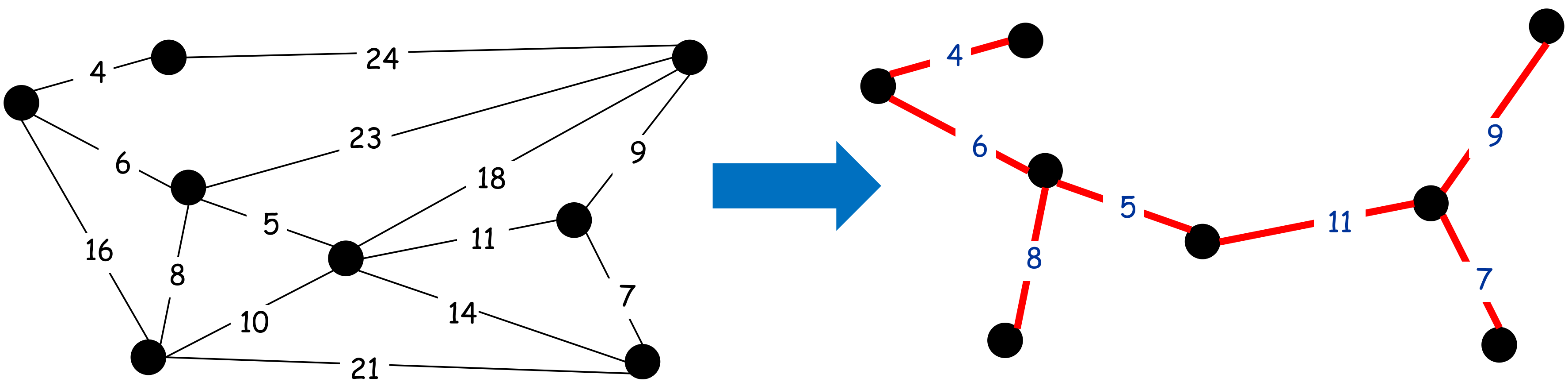
- BFS tree / Dijkstra tree
  - Remember all shortest paths from s.
- DFS tree (see Trémaux tree in wiki)
  - Every two adjacent vertices in G are related to each other as an ancestor and descendant in the tree

There are many other different trees depending on applications.

Minimum Spanning Tree (MST)

Given a connected undirected graph $G = (V, E)$ with real-valued edge weights $c_e \geq 0$.

An MST T is a spanning tree whose sum of edge weights is minimized.



$$G = (V, E)$$

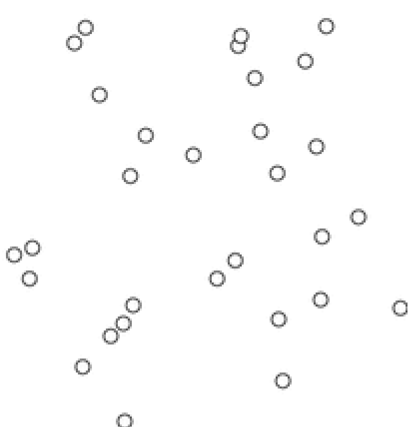
$$c(T) = \sum_{e \in T} c_e = 50$$

See wiki for applications

Kruskal's Algorithm [1956]

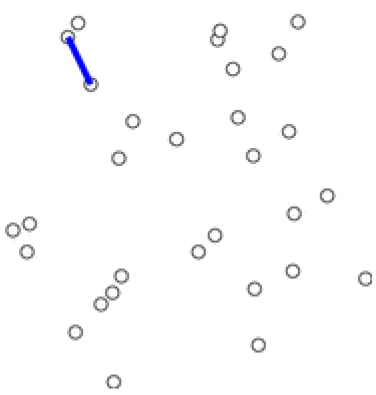
```
Kruskal(G, c) {
    Sort edges weights so that c_1 ≤ c_2 ≤ ⋯ ≤ c_m .
    T ← ∅

    foreach (u ∈ V) make a set containing singleton {u}

    for i = 1 to m
        Let (u,v) = e_i
        if (u and v are in different sets) {
            T ← T ∪ {e_i}
            merge the sets containing u and v
        }
    return T
}
```

Kruskal



Sort edges weight.
Add edges whenever it does not create cycle.

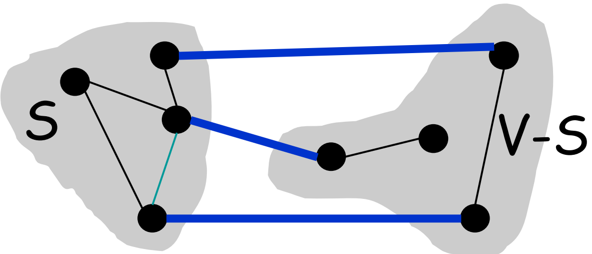
Prim



add the cheapest edge from the tree to another vertex.

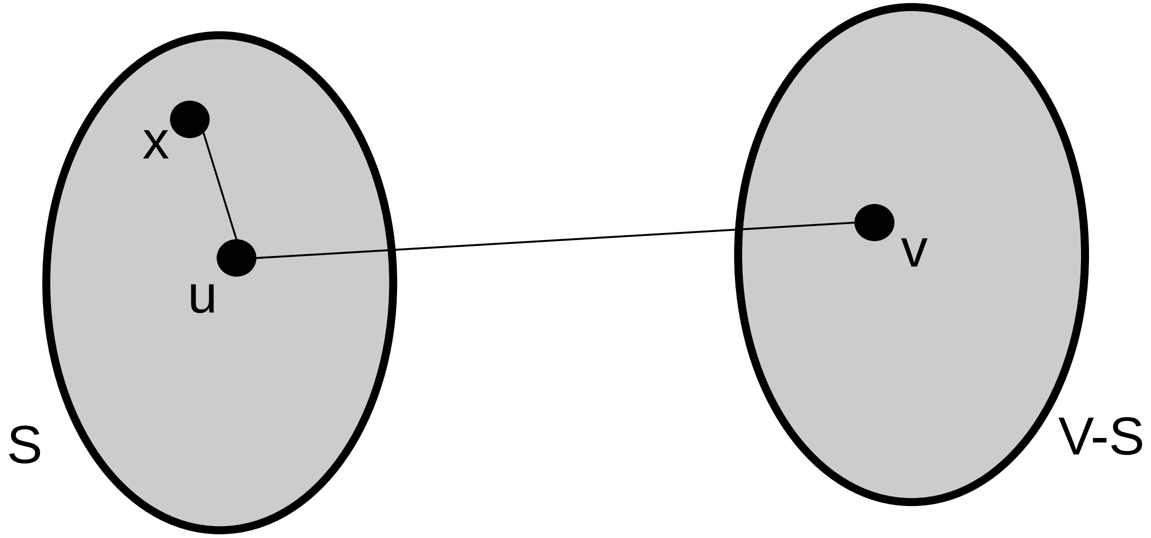
The proof is easier.
Exercise.

Cuts



In a graph $G = (V, E)$, a cut is a bipartition of V into disjoint sets $S, V - S$ for some $S \subseteq V$. We show it by $(S, V - S)$.

An edge $e = \{u, v\}$ is in the cut $(S, V - S)$ if exactly one of u, v is in S.

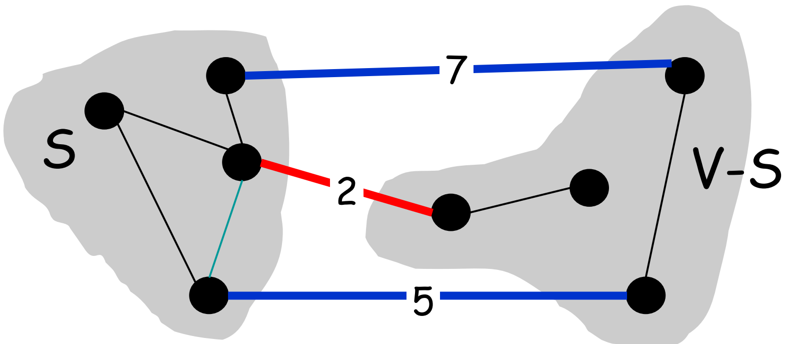


Properties of the OPT

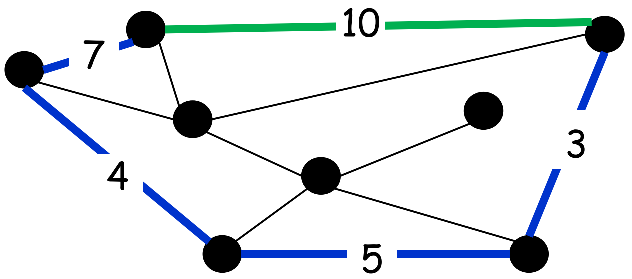
Simplifying assumption: All edge costs c_e are distinct.

Cut property: Let S be any subset of nodes (called a cut), and let e be the min cost edge with exactly one endpoint in S. Then every MST contains e.

Cycle property. Let C be any cycle, and let f be the max cost edge belonging to C. Then no MST contains f.



red edge is in the MST

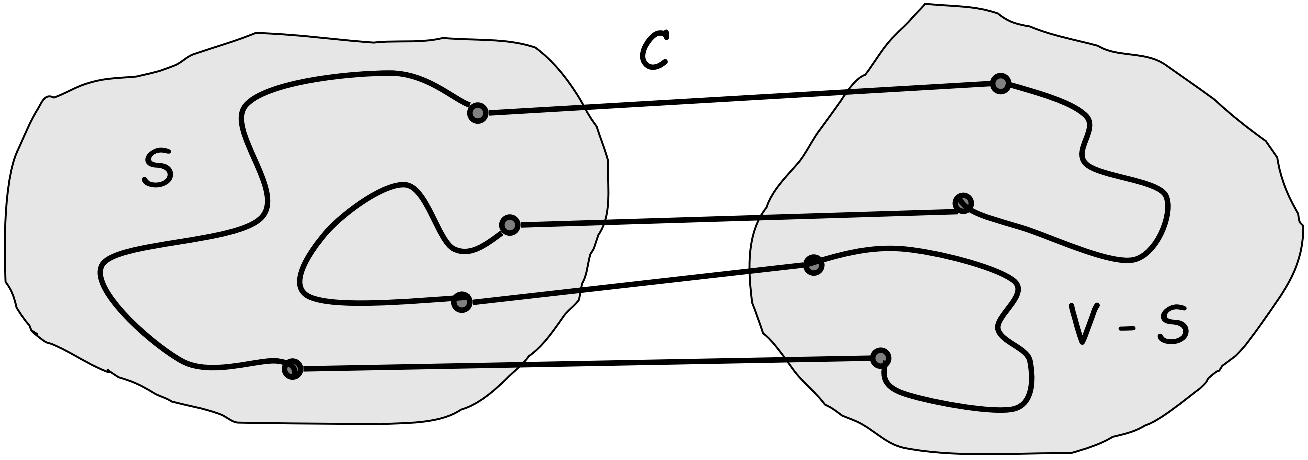


Green edge is not in the MST

Cycles and Cuts

Claim. A cycle crosses a cut (from S to $V - S$) an even number of times.

Proof. (by picture)



Every time the cycle crosses a cut, it goes from S to $V - S$ or from $V - S$ to S.

Cut Property: Proof

Simplifying assumption: All edge costs c_e are distinct.

Cut property. Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then the T^* contains e.

Proof. By contradiction

Suppose $e = \{u, v\}$ does not belong to T^*.

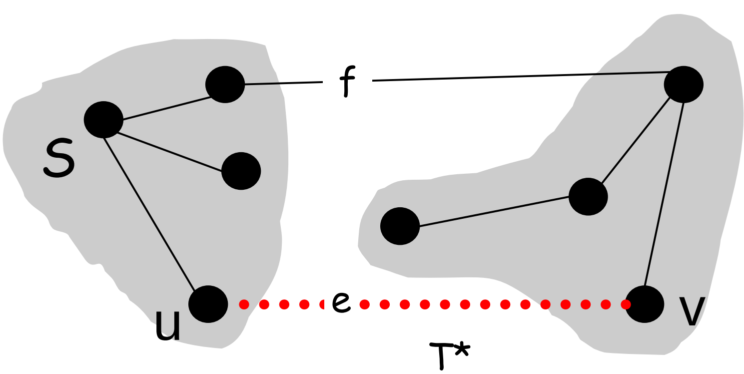
Adding e to T^* creates a cycle C in T^*. (coz all tree has $n - 1$ edges)

There is a path from u to v in T^* $\Rightarrow$ there exists another edge, say f, that leaves S.

$T = T^* \cup \{e\} - \{f\}$ is also a spanning tree.

Since $c_e < c_f$, $c(T) < c(T^*)$.

This is a contradiction.



Cycle Property: Proof

Simplifying assumption: All edge costs c_e are distinct.

Cycle property: Let C be any cycle in G, and let f be the max cost edge belonging to C. Then the MST T^* does not contain f.

Every connected graph has a spanning tree. Hence it has at least $n - 1$ edges.

Proof. By contradiction

Suppose f belongs to T^*.

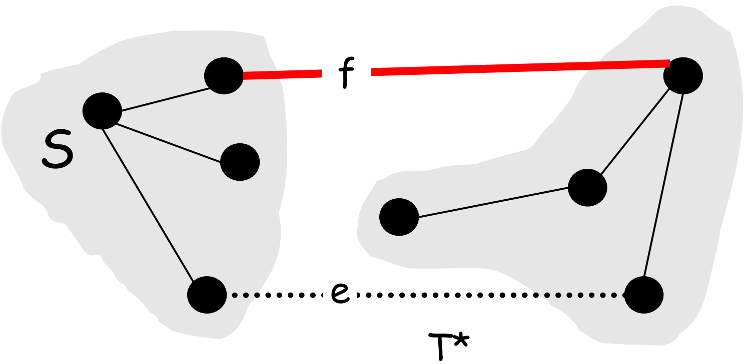
Deleting f from T^* cuts T^* into two connected components.

There exists another edge, say e, that is in the cycle and connects the components.

$T = T^* \cup \{e\} - \{f\}$ is also a spanning tree.

Since $c_e < c_f$, $c(T) < c(T^*)$.

This is a contradiction.



Proof of Correctness (Kruskal)

Consider edges in ascending order of weight.

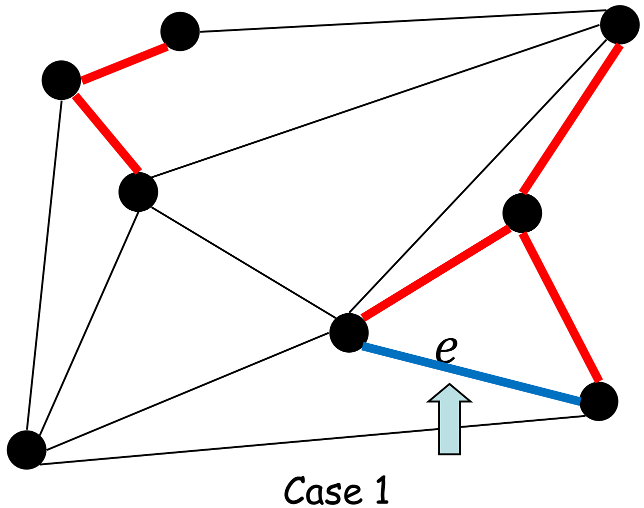
Case 1: adding e to T creates a cycle,

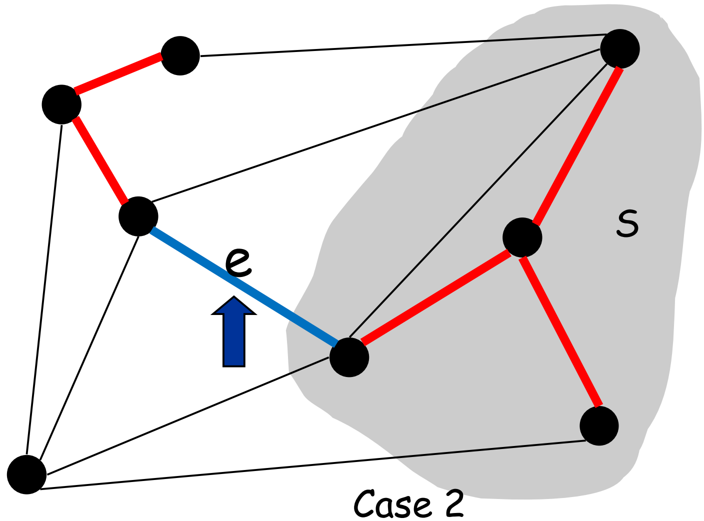
e is the maximum weight edge in that cycle.

cycle property show e is not in any minimum spanning tree.

Case 2: $e = (u, v)$ is the minimum weight edge in the cut S where S is the set of nodes in u's connected component.

So, e is in all minimum spanning tree.





This proves MST is unique if weights are distinct.

Implementation: Kruskal's Algorithm

Implementation. Use the union-find data structure.

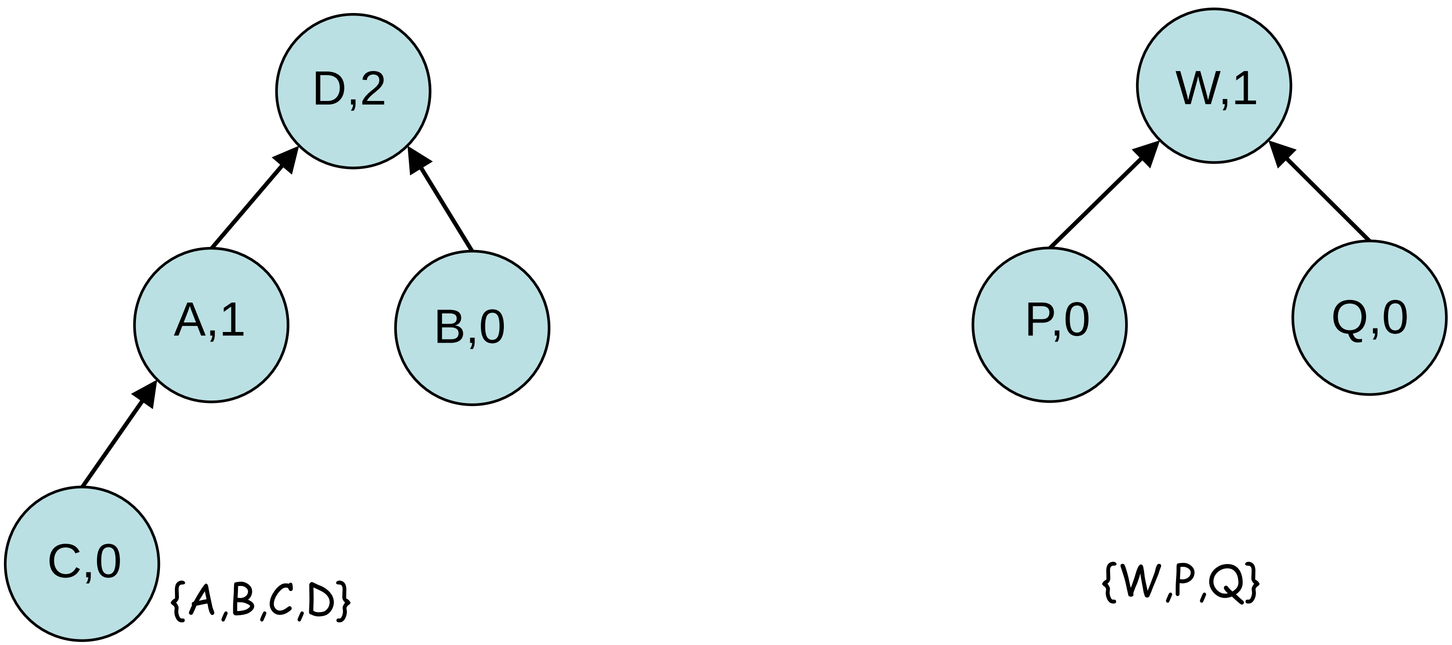
- Build set T of edges in the MST.
- Maintain a set for each connected component.
- O($m \log n$) for sorting and O($m \log n$) for union-find

```
Kruskal(G, c) {
   Sort edges weights so that c_1 ≤ c_2 ≤ ··· ≤ c_m.
   T ← ∅

   foreach (u ∈ V) make a set containing singleton {u}

   for i = 1 to m
      Let (u,v) = e_i
      if (u and v are in different sets) {
         T ← T ∪ {e_i}
         merge the sets containing u and v
      }
   return T
}
```

Union Find Data Structure

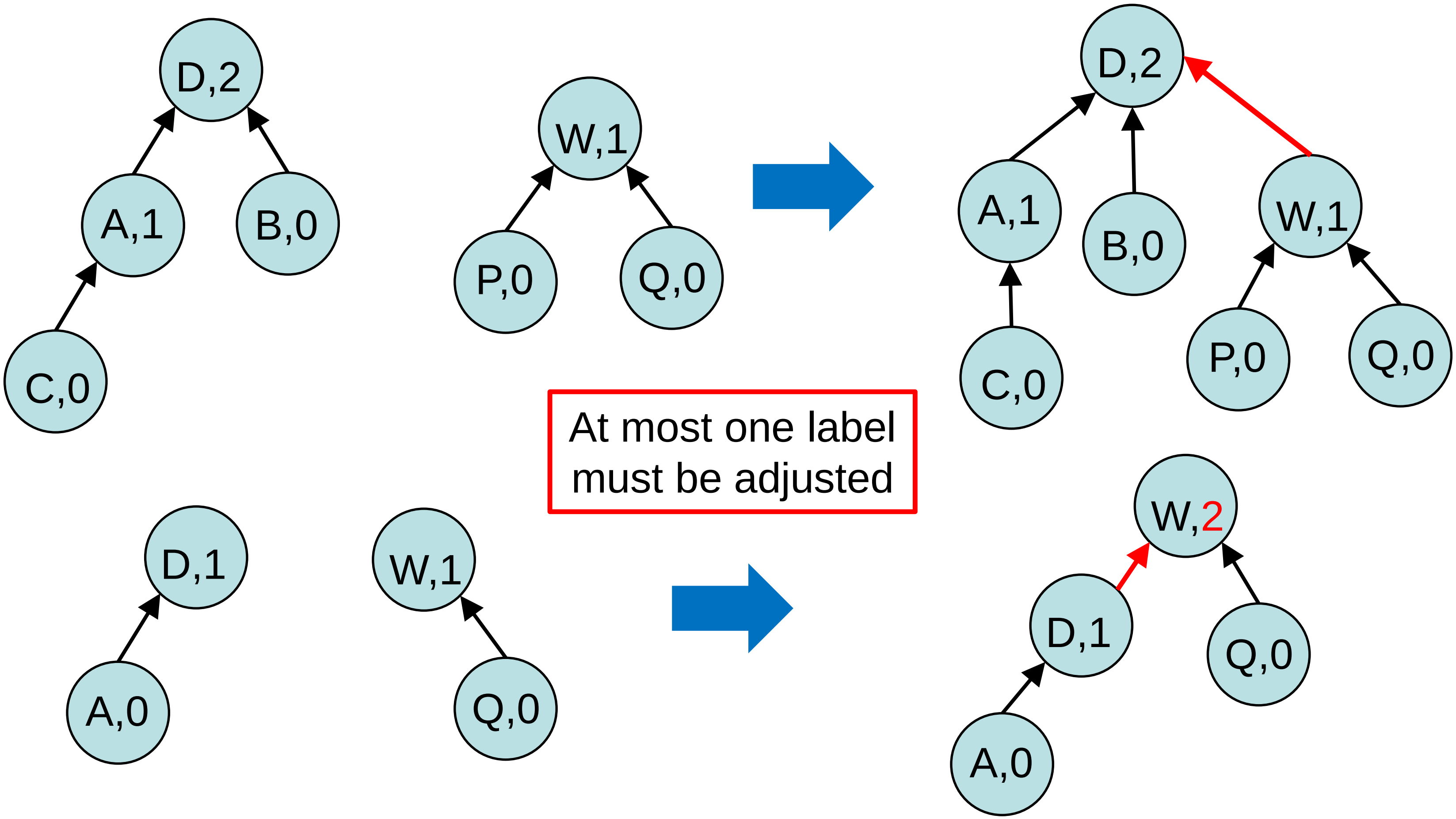
Each set is represented as a tree of pointers, where every vertex is labeled with longest path ending at the vertex

To check whether A,Q are in same connected component, follow pointers and check if root is the same.



Union Find Data Structure

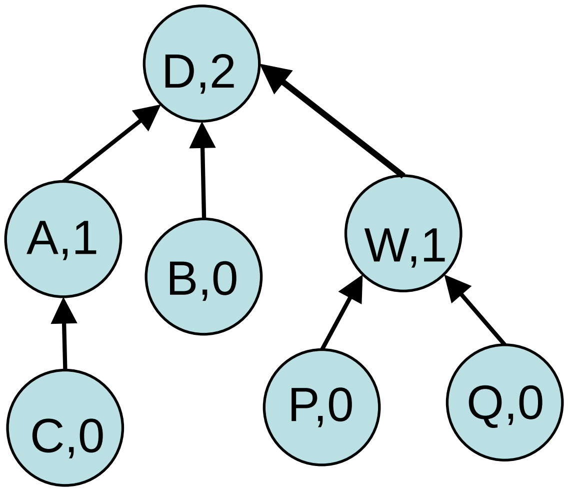
Merge: To merge two connected components, make the root with the smaller label point to the root with the bigger label (adjusting labels if necessary). Runs in $O(1)$ time



Depth vs Size

Claim: If the label of a root is k, there are at least 2^k elements in the set.

Therefore, the depth of any tree in algorithm is at most $\log_2 n$



So, we can check if u, v are in the same component in time $O(\log n)$

Depth vs Size: Correctness

Claim: If the label of a root is k, there are at least 2^k elements in the set.

Proof: By induction on k.

Base Case ($k = 0$): this is true. The set has size 1.

Inductive Step: If we merge roots with labels $k_1 > k_2$, the number of vertices only increases while the label stays the same.

If $k_1 = k_2$, the merged tree has label $k_1 + 1$,

and by induction, it has at least

$$2^{k_1} + 2^{k_2} = 2^{k_1+1}$$

elements.

Kruskal's Algorithm with Union Find

Implementation. Use the union-find data structure.

- Build set T of edges in the MST.
- Maintain a set for each connected component.
- O(m log n) for sorting and O(m log n) for union-find

```
Kruskal(G, c) {
   Sort edges weights so that c_1 ≤ c_2 ≤ ⋯ ≤ c_m.
   T ← ∅

   foreach (u ∈ V) make a set containing singleton {u}

   for i = 1 to m
      Let (u,v) = e_i
      if (u and v are in different sets) {
         T ← T ∪ {e_i}
         merge the sets containing u and v
      }
   return T
}
```

Find roots and compare

Merge at the roots