CSE 421

Greedy Algorithms

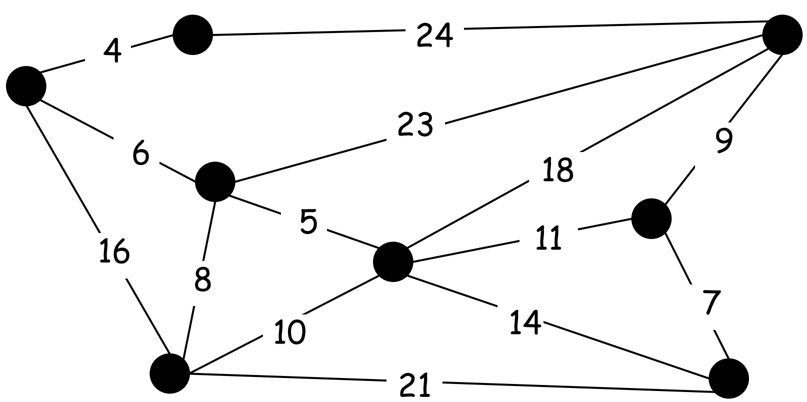
Shayan Oveis Gharan

Minimum Spanning Tree Problem

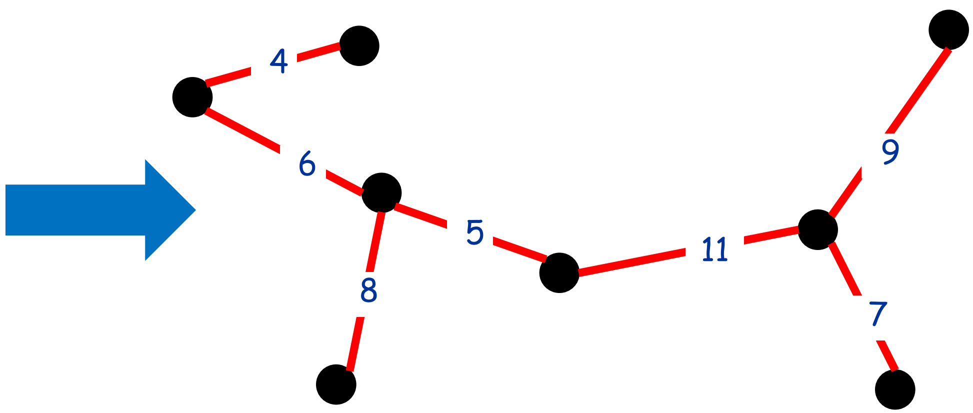
Minimum Spanning Tree (MST)

Given a connected graph $G = (V, E)$ with real-valued edge weights c_e, an MST is a subset of the edges $T \subseteq E$ such that T is a spanning tree whose sum of edge weights is minimized.

↳ all vertices



$G = (V, E)$



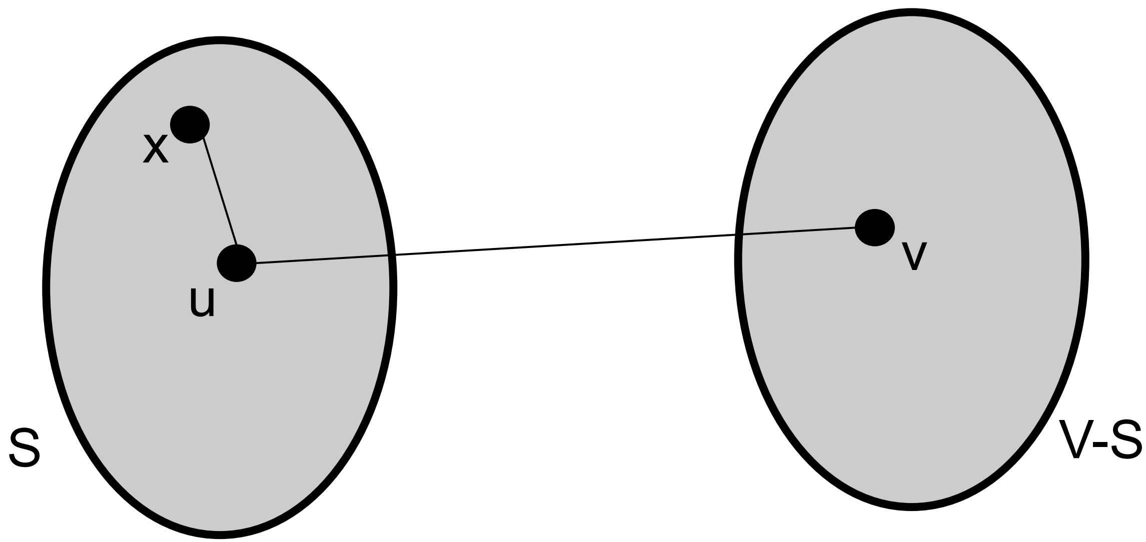
$$c(T) = \sum_{e \in T} c_e = 50$$

Cuts

A graph has $2^{n-1}-1$ many cuts

In a graph $G = (V, E)$ a cut is a bipartition of V into 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.



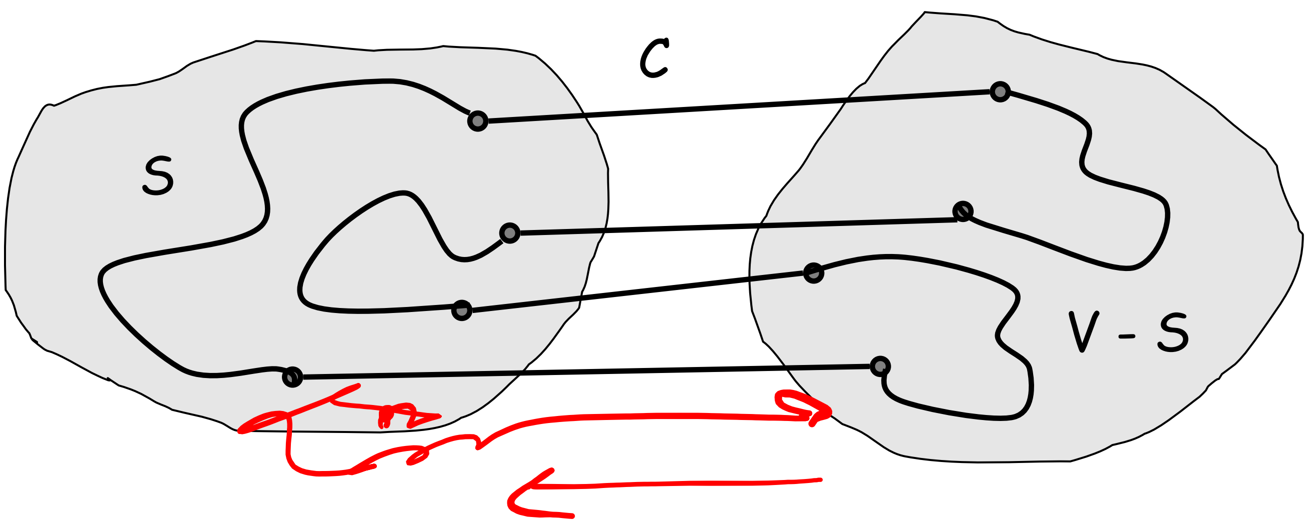
Obs: If G is connected then there is at least one edge in every cut.

If G not conn $\Rightarrow \exists\ (S, V-S)$ no edges

Cycles and Cuts

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

Pf. (by picture)

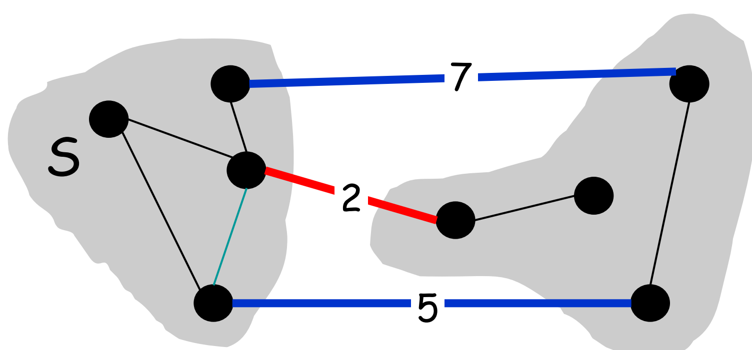


Properties of the OPT

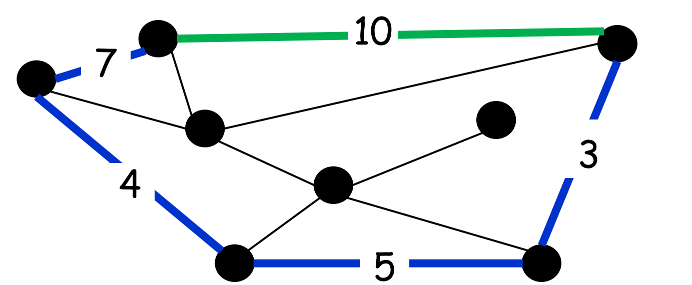
Simplifying assumption: All edge costs c_e are distinct.

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

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 T* contains e.

Pf. By contradiction

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

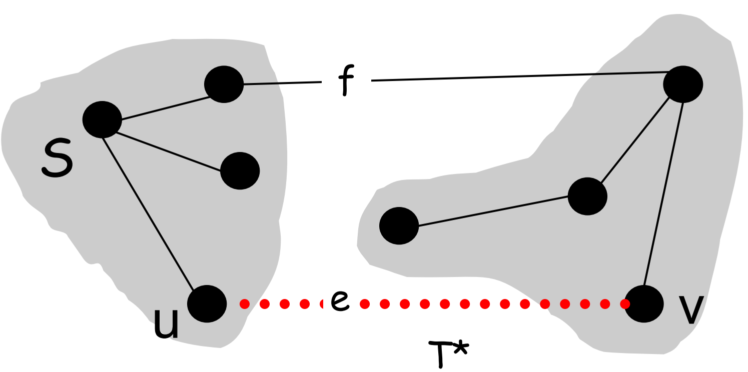
Adding e to T* creates a cycle C in T*.

C crosses S even number of times $\Rightarrow$ 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.

Pf. (By contradiction)

Suppose f belongs to T*.

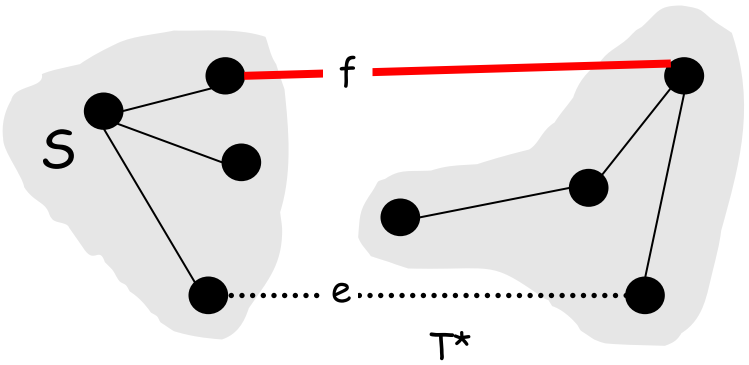
Deleting f from T* cuts T* into two connected components.

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.



Kruskal's Algorithm [1956]

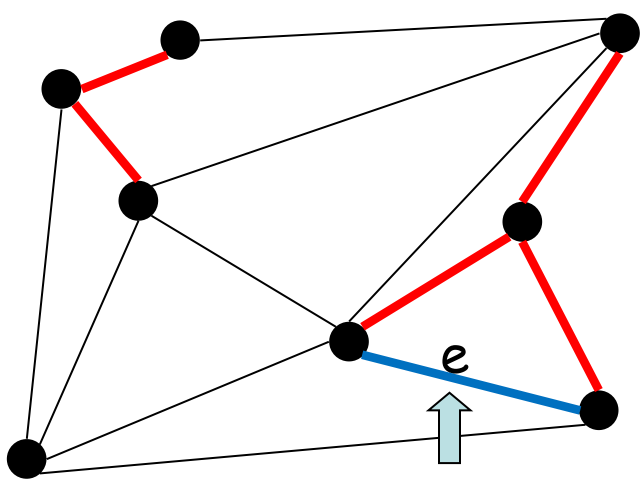
```
Kruskal(G, c) {
   Sort edges weights so that c1 ≤ c2 ≤ ... ≤ cm.
   T ← ∅

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

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

Kruskal's Algorithm: Pf of Correctness

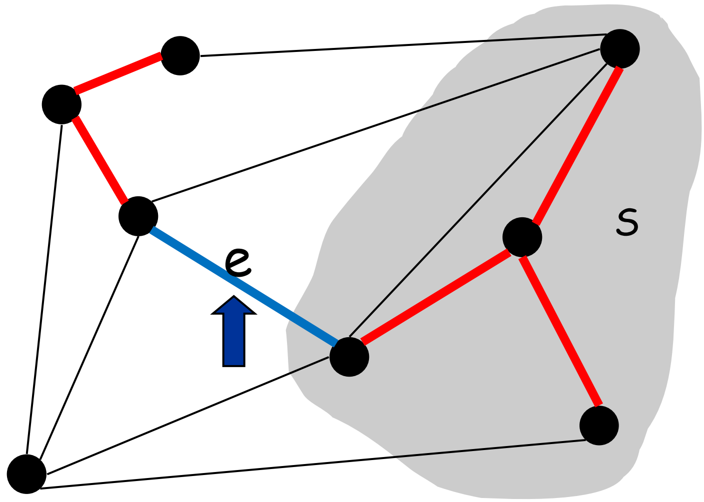
Consider edges in ascending order of weight.

Case 1: If adding e to T creates a cycle, discard e according to cycle property.

Case 2: Otherwise, insert e = (u, v) into T according to cut property where S = set of nodes in u's connected component.



Case 1



Case 2

Implementation: Kruskal's Algorithm

Implementation. Use the union-find data structure.

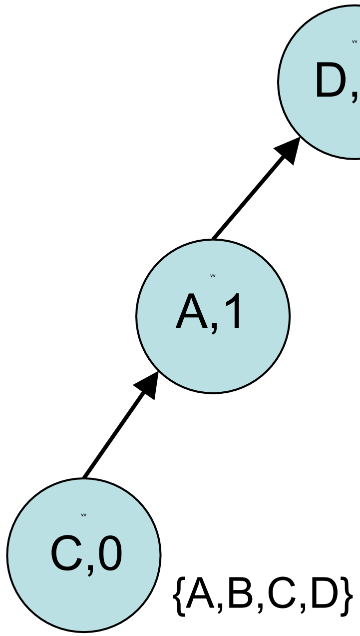
- Build set T of edges in the MST.
- Maintain a set for each connected component.
- O(m log n) for sorting and O(m log n) for union-find

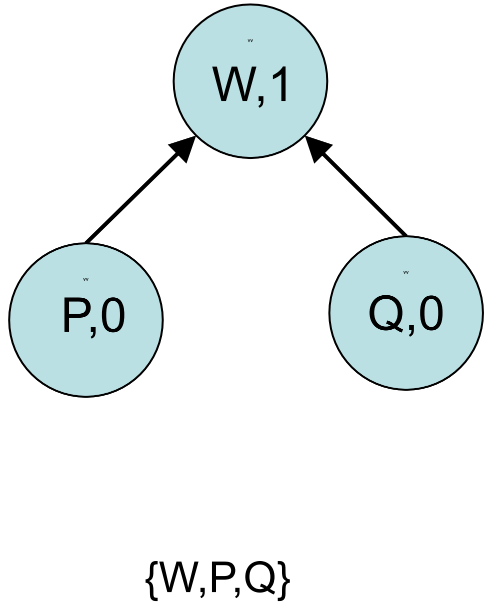
```
Kruskal(G, c) {
   Sort edges weights so that c₁ ≤ c₂ ≤ ... ≤ cₘ.
   T ← ∅

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

   for i = 1 to m
      Let (u,v) = eᵢ
      if (u and v are in different sets) {
         T ← T ∪ {eᵢ}
         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

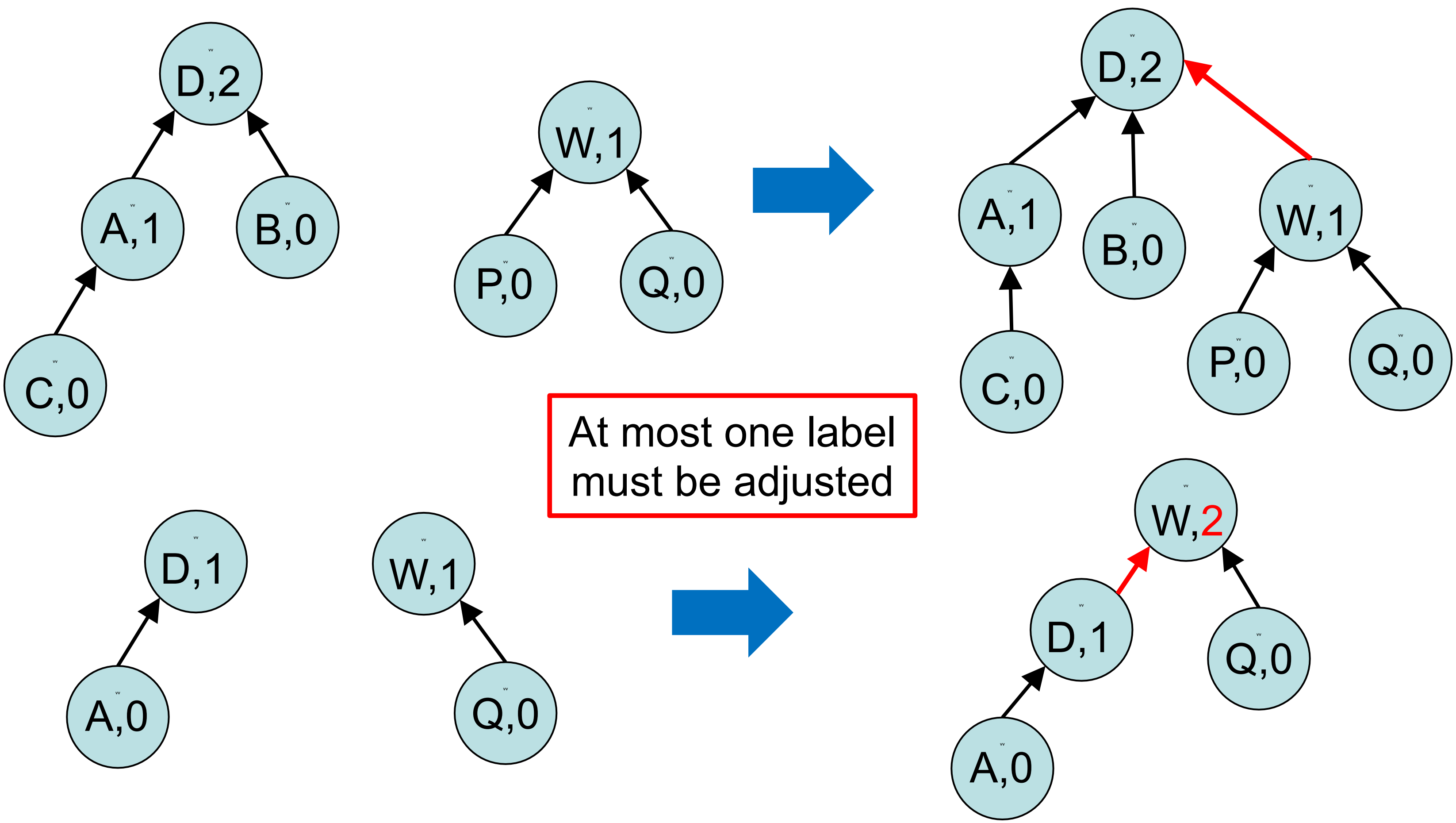
To check whether A,Q are in same connected component, follow pointers and check if root is the same.





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



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 c1 ≤ c2 ≤ ... ≤ cm.
   T ← ∅

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

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

Find roots and compare

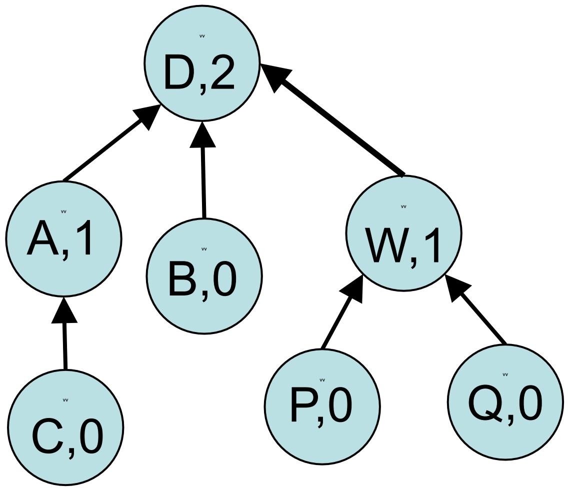
Merge at the roots

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 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.

Pf: By induction on k.

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

IH: Suppose the claim is true until some time t

IS: 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.

Removing weight Distinction Assumption

Suppose edge weights are not distinct, and Kruskal's algorithm sorts edges so

$$c_{e_1} \leq c_{e_2} \leq \cdots \leq c_{e_m}$$

Suppose Kruskal finds tree T of weight $c(T)$, but the optimal solution T^* has cost $c(T^*) < c(T)$.

Perturb each of the weights by a very small amount so that

$$c'_{e_1} < c'_{e_2} < \cdots < c'_{e_m}$$

where $c'_{e_i} = c_{e_i} + i.\epsilon$

If ϵ is small enough, $c'(T^*) \leq c(T^*) + m^2\epsilon < c(T)$.

But Kruskal's algorithm returns the same output T. This contradicts the correctness of Kruskal's algorithm, since Kruskal's algorithm is correct if all weights are distinct. ∎

Summary (Greedy Algorithms)

- Greedy Stays Ahead: Interval Scheduling, Dijkstra's algorithm
- Structural: Interval Partitioning
- Exchange Arguments: MST, Kruskal's Algorithm,
- Data Structures: Union Find