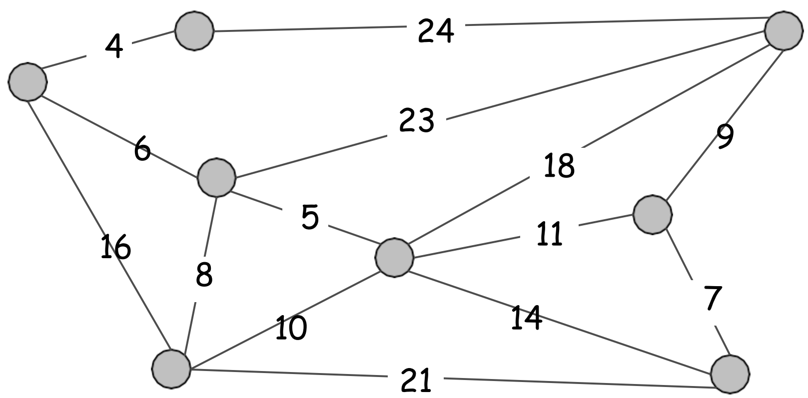
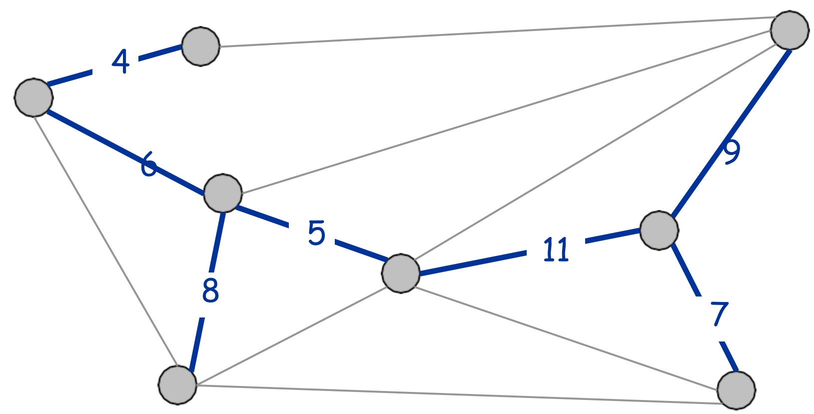
4.5 Minimum Spanning Tree

Minimum Spanning Tree

Minimum spanning tree. 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.



$G = (V, E)$



T, $\Sigma_{e \in T} c_e = 50$

Cayley's Theorem. There are n^{n-2} spanning trees of K_n.

can't solve by brute force

Applications

MST is fundamental problem with diverse applications.

- Network design.
  - telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-hard problems.
  - traveling salesperson problem, Steiner tree
- Indirect applications.
  - max bottleneck paths
  - LDPC codes for error correction
  - image registration with Renyi entropy
  - learning salient features for real-time face verification
  - reducing data storage in sequencing amino acids in a protein
  - model locality of particle interactions in turbulent fluid flows
  - autoconfig protocol for Ethernet bridging to avoid cycles in a network
- Cluster analysis.

Kruskal's algorithm. Start with $T = \phi$. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Reverse-Delete algorithm. Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

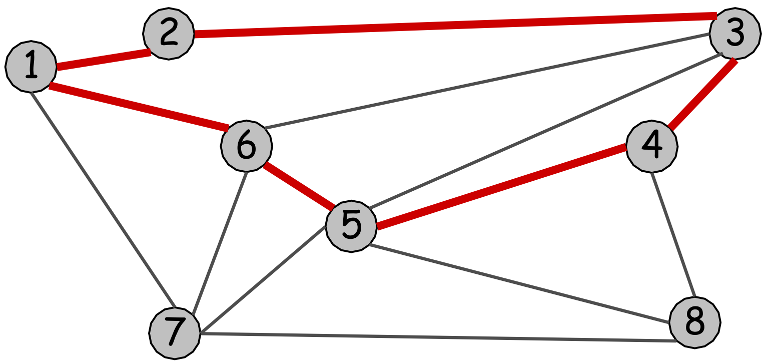
Prim's algorithm. Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T.

Greedy Algorithms

Simplifying assumption. All edge costs c_e are distinct.

Cycles and Cuts

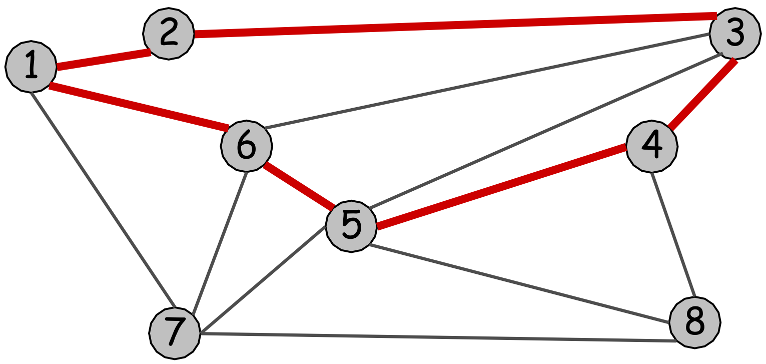
Cycle. Set of edges of the form a-b, b-c, c-d, …, y-z, z-a.



Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1

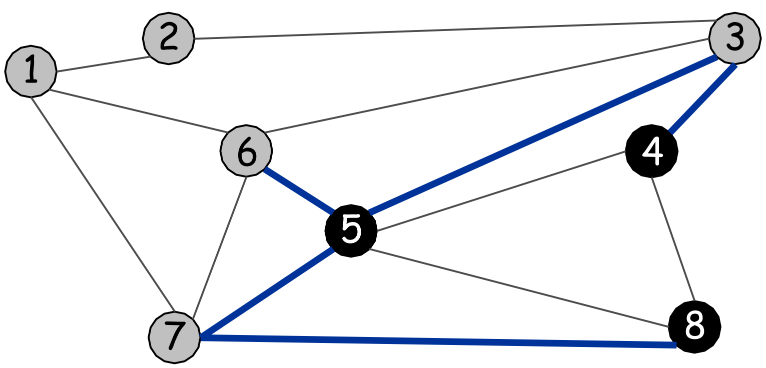
Cycles and Cuts

Cycle. Set of edges of the form a-b, b-c, c-d, …, y-z, z-a.



Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1

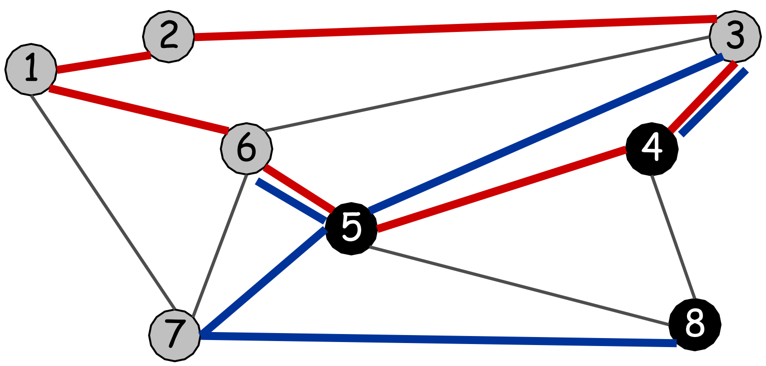
Cutset. A ***cut*** is a subset of nodes S. The corresponding ***cutset*** D is the subset of edges with exactly one endpoint in S.



Cut S = { 4, 5, 8 }
Cutset D = 5-6, 5-7, 3-4, 3-5, 7-8

Cycle-Cut Intersection

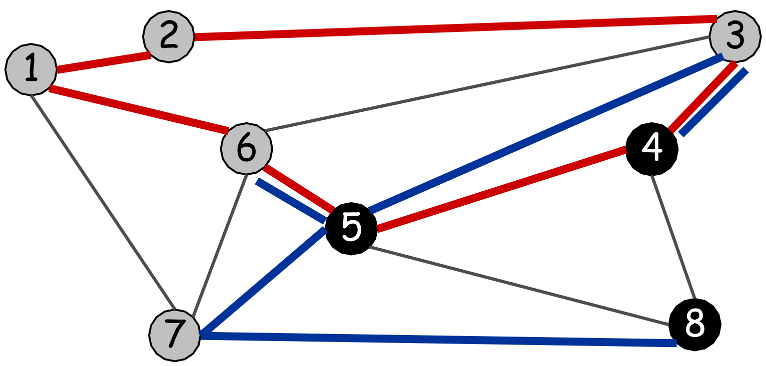
Claim. A cycle and a cutset intersect in an even number of edges.



Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1
Cutset D = 3-4, 3-5, 5-6, 5-7, 7-8
Intersection = 3-4, 5-6

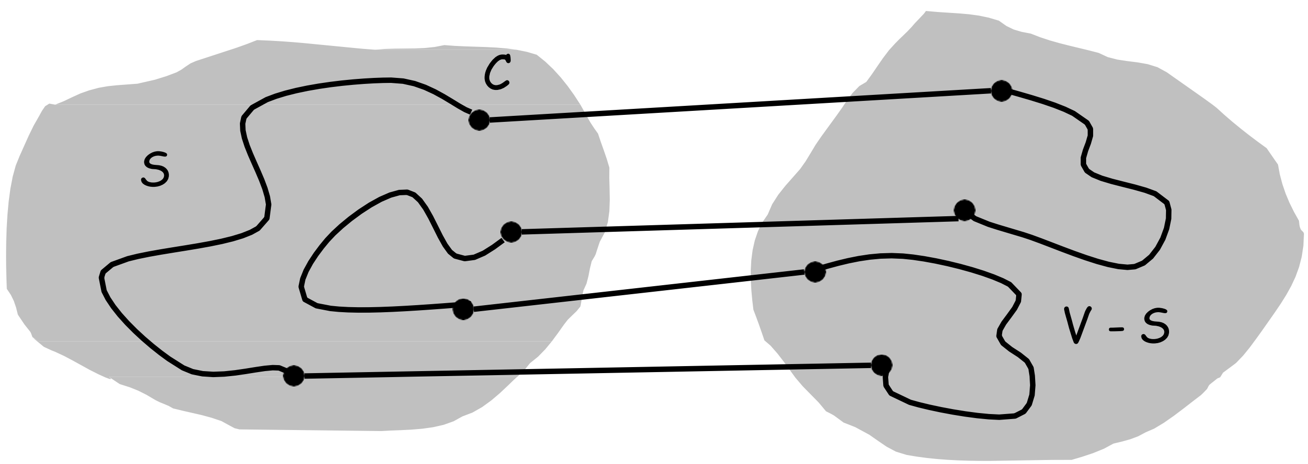
Cycle-Cut Intersection

Claim. A cycle and a cutset intersect in an even number of edges.



Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1
Cutset D = 3-4, 3-5, 5-6, 5-7, 7-8
Intersection = 3-4, 5-6

Pf sketch. Consider a cut S (recall S is a set of nodes) and a cycle C.
The corresponding cutset D is the subset of edges with exactly one endpoint in S.



Simplifying assumption. All edge costs c_e are distinct.

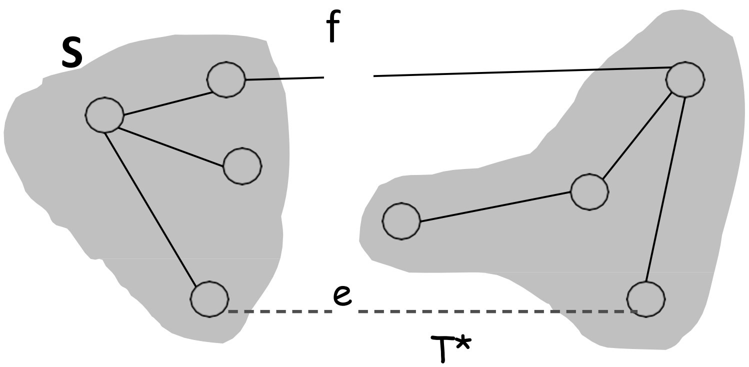
Cut property. Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then the MST T* contains e.

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 MST T* contains e.

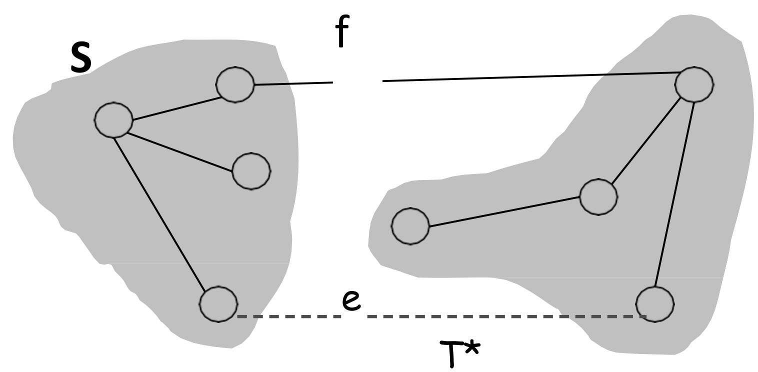
Pf. (exchange argument)
- Suppose e does not belong to T*, and let's see what happens.



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 MST T* contains e.

Pf. (exchange argument)
- Suppose e does not belong to T*, and let's see what happens.
- Adding e to T* creates a cycle C in T*.

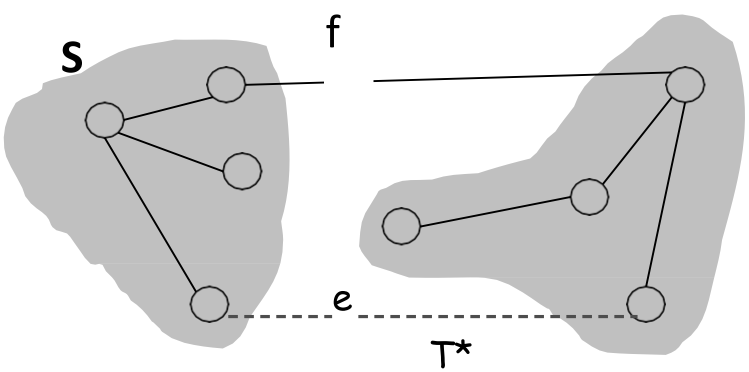


Simplifying assumption. All edge costs c_e are distinct.

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 MST T* contains e.

Pf. (exchange argument)

- Suppose e does not belong to T*, and let's see what happens.
- Adding e to T* creates a cycle C in T*.
- Edge e is both in the cycle C and in the cutset D corresponding to S
  $\Rightarrow$ there exists another edge, say f, that is in both C and D.

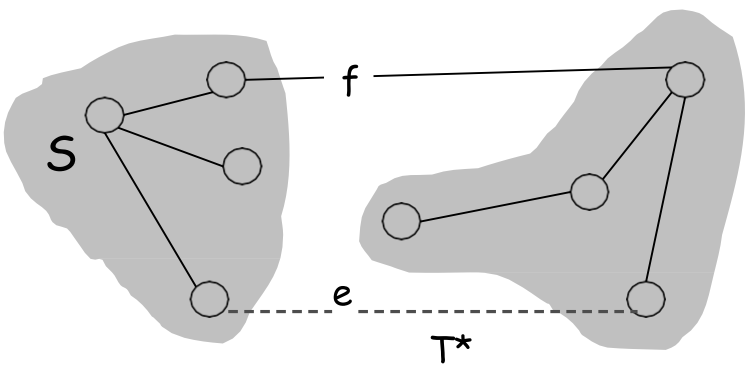


Simplifying assumption. All edge costs c_e are distinct.

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 MST T* contains e.

Pf. (exchange argument)

- Suppose e does not belong to T*, and let's see what happens.
- Adding e to T* creates a cycle C in T*.
- Edge e is both in the cycle C and in the cutset D corresponding to S $\Rightarrow$ there exists another edge, say f, that is in both C and D.
- $T' = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.

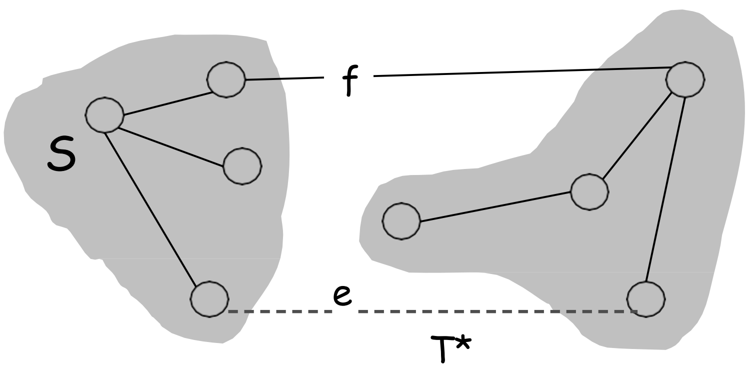


Simplifying assumption. All edge costs c_e are distinct.

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 MST T* contains e.

Pf. (exchange argument)

- Suppose e does not belong to T*, and let's see what happens.
- Adding e to T* creates a cycle C in T*.
- Edge e is both in the cycle C and in the cutset D corresponding to S
  $\Rightarrow$ there exists another edge, say f, that is in both C and D.
- $T' = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.
- Since $c_e < c_f$, $cost(T') < cost(T^*)$.

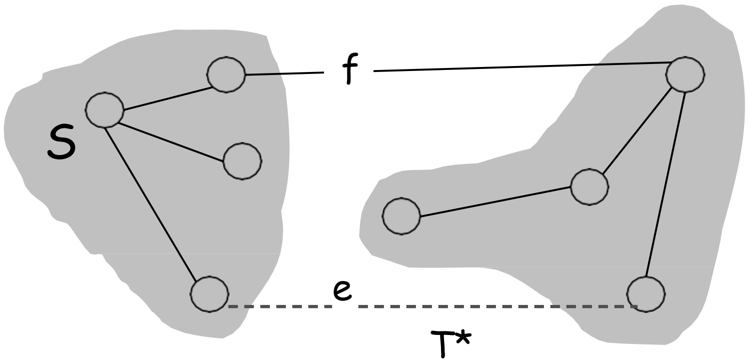


Simplifying assumption. All edge costs c_e are distinct.

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 MST T* contains e.

Pf. (exchange argument)

- Suppose e does not belong to T*, and let's see what happens.
- Adding e to T* creates a cycle C in T*.
- Edge e is both in the cycle C and in the cutset D corresponding to S
  $\Rightarrow$ there exists another edge, say f, that is in both C and D.
- $T' = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.
- Since $c_e < c_f$, cost(T') < cost(T*).
- This is a contradiction. ▪



Algorithm based on the cut property

Task: design an algorithm using this observation (the cut property).

Prim's Algorithm: Proof of Correctness

Prim's algorithm. [Jarník 1930, Dijkstra 1957, Prim 1959]

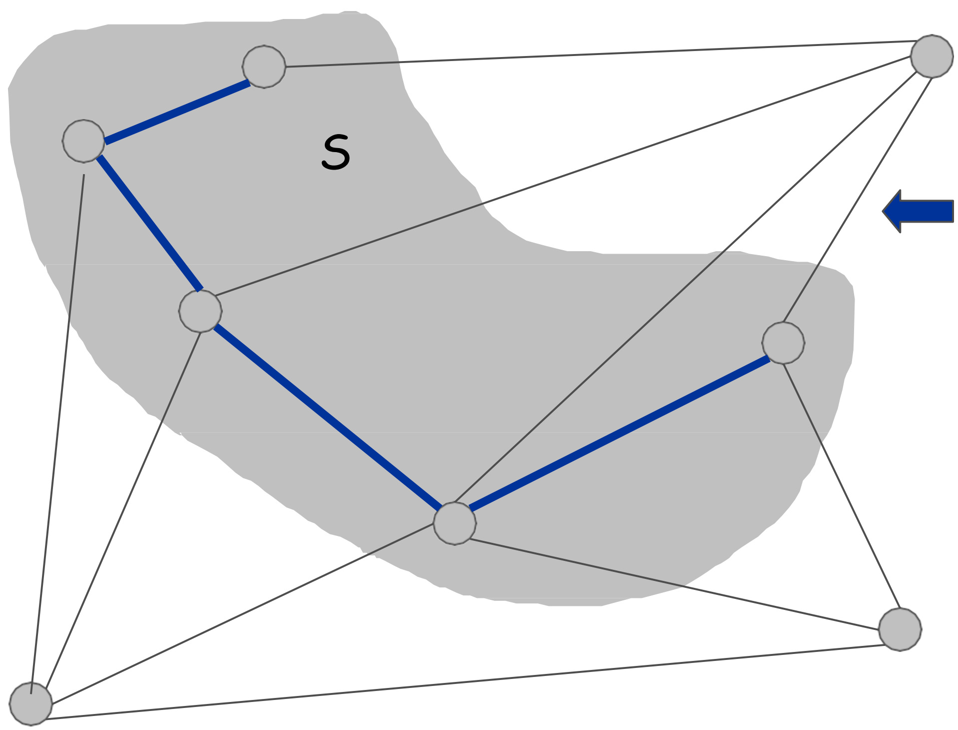
Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T.

Prim's Algorithm: Proof of Correctness

Prim's algorithm. [Jarník 1930, Dijkstra 1957, Prim 1959]

Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T.

- Initialize S = any node.
- Apply the cut property to S. *(Recall it: Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then the MST T* contains e.)*
- Add min cost edge in cutset corresponding to S to T, and add one new explored node u to S.



Implementation. Use a priority queue as in Dijkstra.

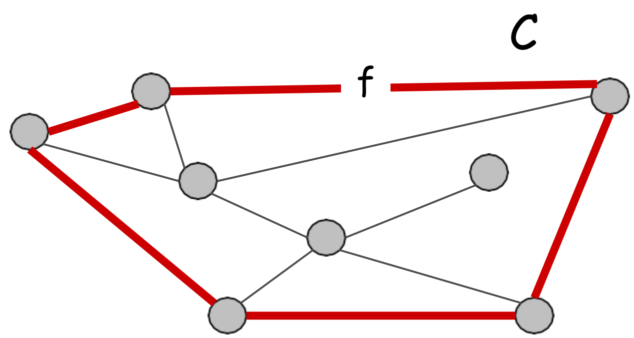
- Maintain set of explored nodes S.
- For each unexplored node v, maintain attachment cost a[v] = cost of cheapest edge v to a node in S.
- $O(n^2)$ with an array; $O(m \log n)$ with a binary heap.

```
Prim(G, c) {
   foreach (v ∈ V) a[v] ← ∞
   Initialize an empty priority queue Q
   foreach (v ∈ V) insert v onto Q
   Initialize set of explored nodes S ← ϕ

   while (Q is not empty) {
      u ← delete min element from Q
      S ← S ∪ { u }
      foreach (edge e = (u, v) incident to u)
          if ((v ∉ S) and (ce < a[v]))
             decrease priority a[v] to ce}
```

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.



f is not in the MST

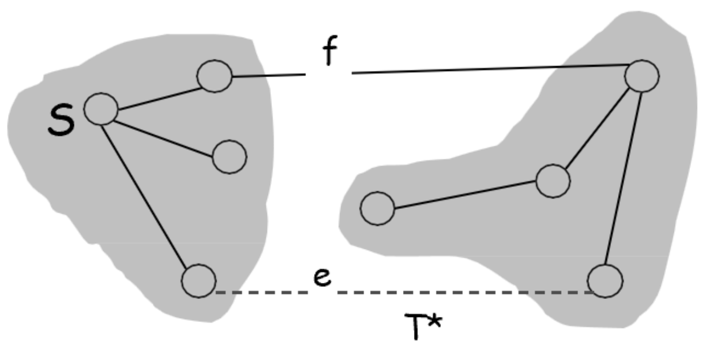
Proof?

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. (exchange argument)

- Suppose f belongs to T*, and let's see what happens.

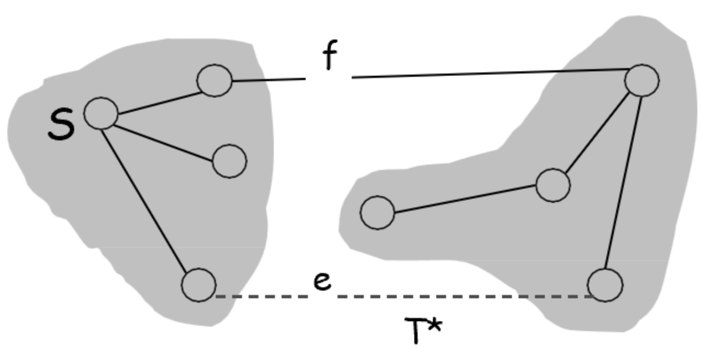


Simplifying assumption. All edge costs c_e are distinct.

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. (exchange argument)

- Suppose f belongs to T*, and let's see what happens.
- Deleting f from T* creates a cut S in T*.

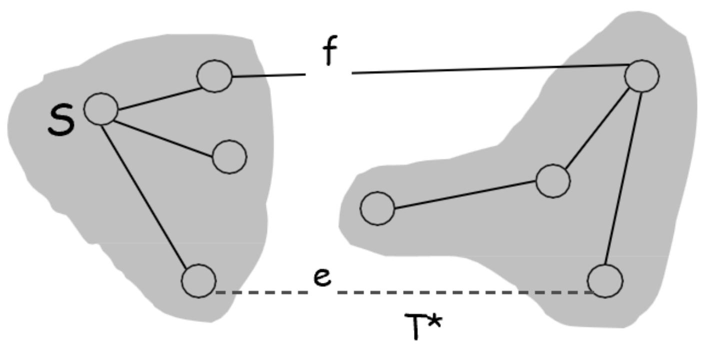


Simplifying assumption. All edge costs c_e are distinct.

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. (exchange argument)

- Suppose f belongs to T*, and let's see what happens.
- Deleting f from T* creates a cut S in T*.
- Edge f is both in the cycle C and in the cutset D corresponding to S
  $\Rightarrow$ there exists another edge, say e, that is in both C and D.

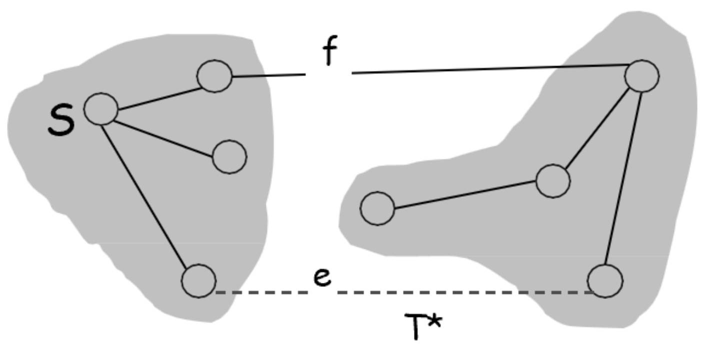


Simplifying assumption. All edge costs c_e are distinct.

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. (exchange argument)

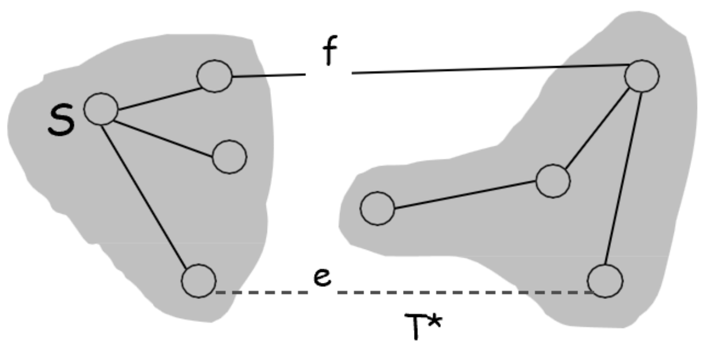
- Suppose f belongs to T*, and let's see what happens.
- Deleting f from T* creates a cut S in T*.
- Edge f is both in the cycle C and in the cutset D corresponding to S $\Rightarrow$ there exists another edge, say e, that is in both C and D.
- $T' = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.



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. (exchange argument)
- Suppose f belongs to T*, and let's see what happens.
- Deleting f from T* creates a cut S in T*.
- Edge f is both in the cycle C and in the cutset D corresponding to S
  $\Rightarrow$ there exists another edge, say e, that is in both C and D.
- $T' = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.
- Since $c_e < c_f$, $\text{cost}(T') < \text{cost}(T^*)$.

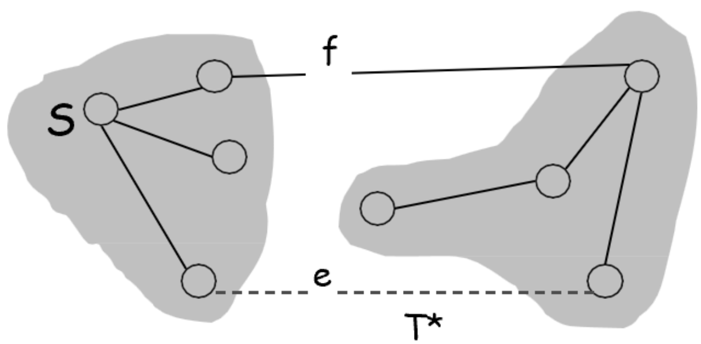


Simplifying assumption. All edge costs c_e are distinct.

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. (exchange argument)

- Suppose f belongs to T*, and let's see what happens.
- Deleting f from T* creates a cut S in T*.
- Edge f is both in the cycle C and in the cutset D corresponding to S $\Rightarrow$ there exists another edge, say e, that is in both C and D.
- $T' = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.
- Since $c_e < c_f$, $cost(T') < cost(T^*)$.
- This is a contradiction. ▪



Algorithm using the cycle property

Task: design an algorithm using this observation (the cycle property).

Kruskal's Algorithm

Kruskal's algorithm. [Kruskal, 1956]

Start with T = ∅. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Kruskal's Algorithm: Proof of Correctness

Kruskal's algorithm. [Kruskal, 1956]

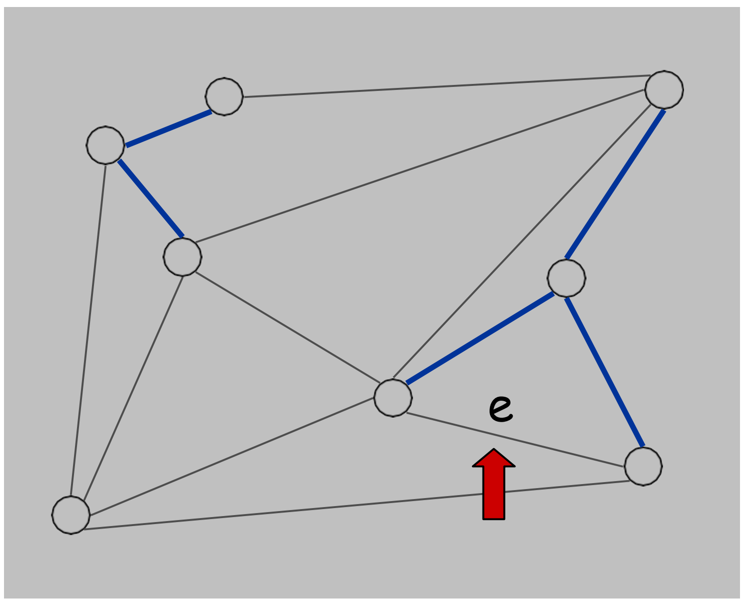
Start with T = ϕ. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Kruskal's Algorithm: Proof of Correctness

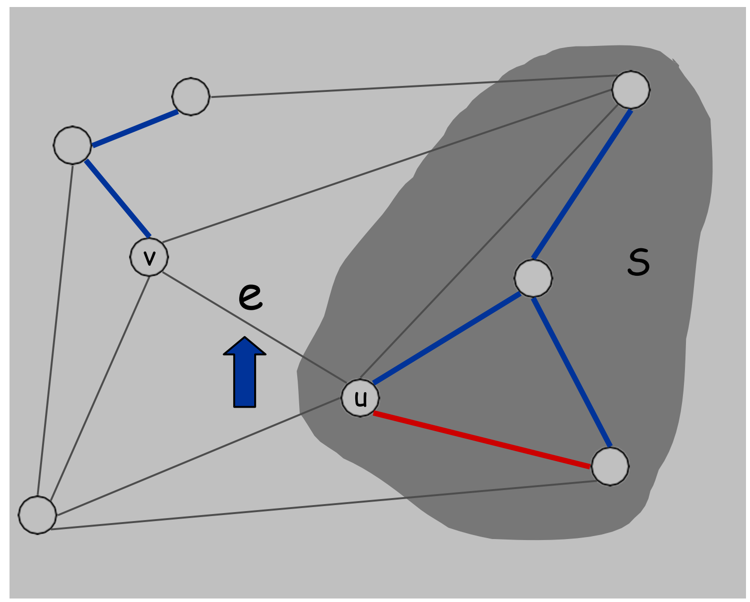
Start with T = ϕ. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Kruskal's algorithm. [Kruskal, 1956]

- Consider edges in ascending order of weight.



Case 1



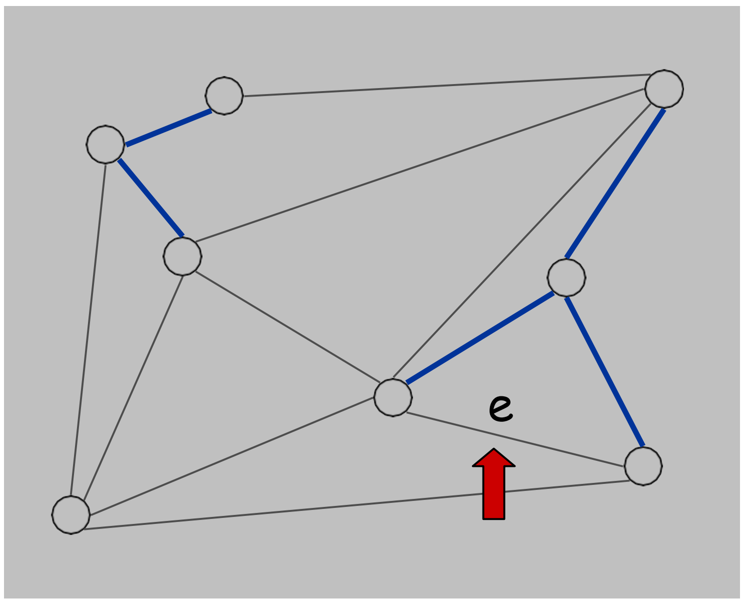
Case 2

Kruskal's Algorithm: Proof of Correctness

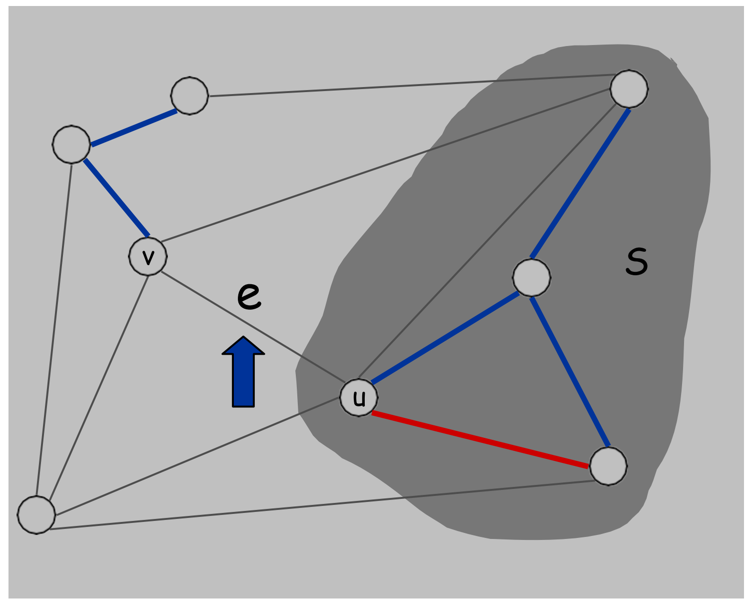
Start with T = ϕ. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Kruskal's algorithm. [Kruskal, 1956]

- Consider edges in ascending order of weight.
- Case 1: If adding e to T creates a cycle, then …



Case 1



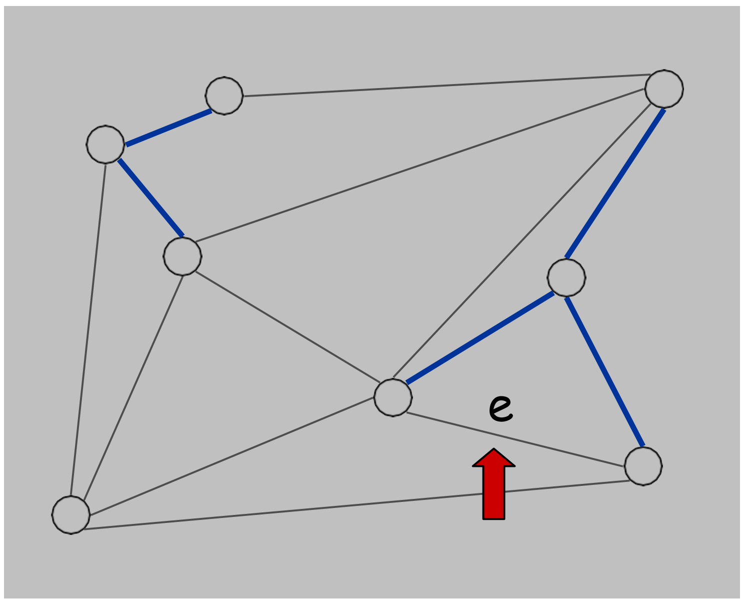
Case 2

Kruskal's Algorithm: Proof of Correctness

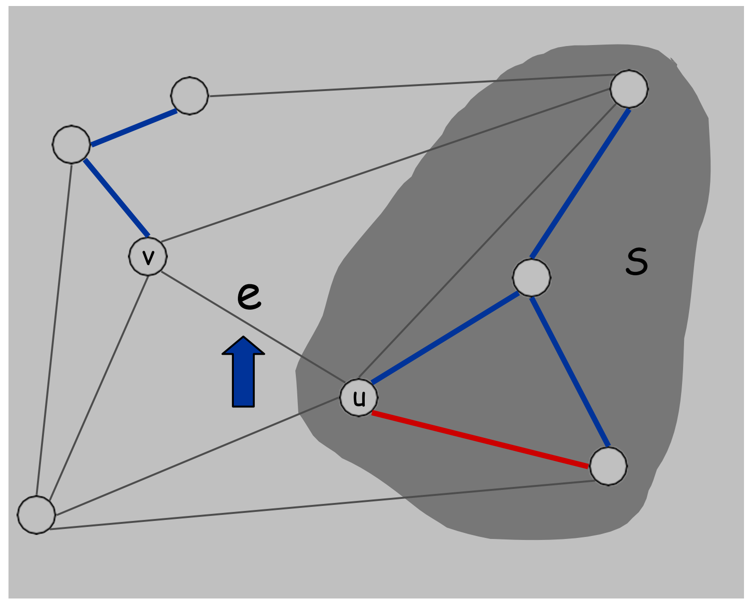
Start with T = ϕ. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Kruskal's algorithm. [Kruskal, 1956]

- Consider edges in ascending order of weight.
- Case 1: If adding e to T creates a cycle, discard e according to cycle property.



Case 1



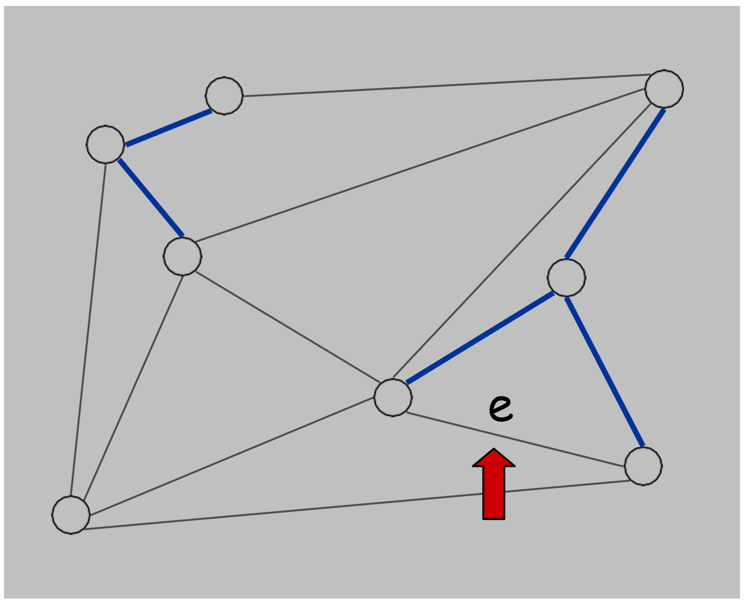
Case 2

Kruskal's Algorithm: Proof of Correctness

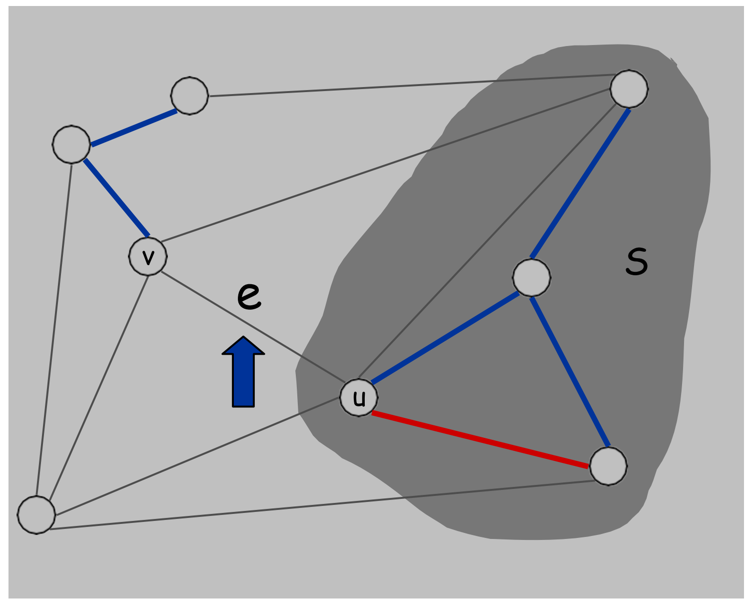
Start with T = ϕ. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Kruskal's algorithm. [Kruskal, 1956]

- 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: Else:



Case 1



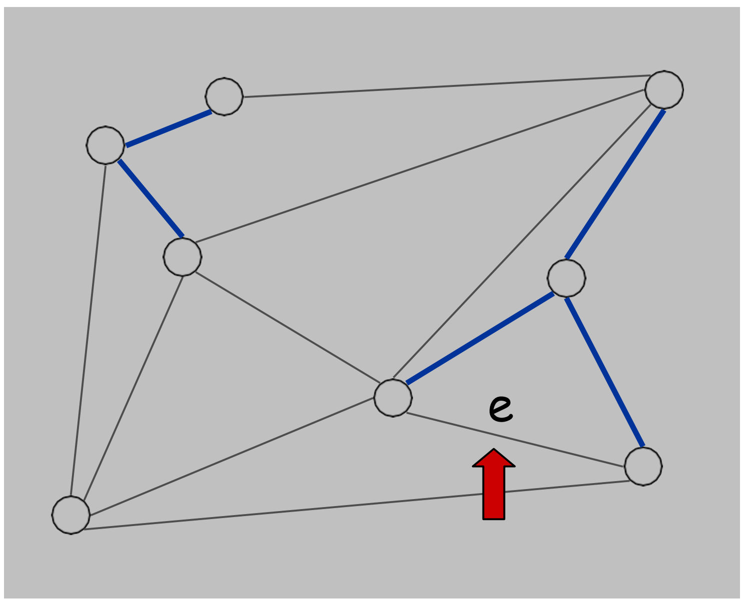
Case 2

Kruskal's Algorithm: Proof of Correctness

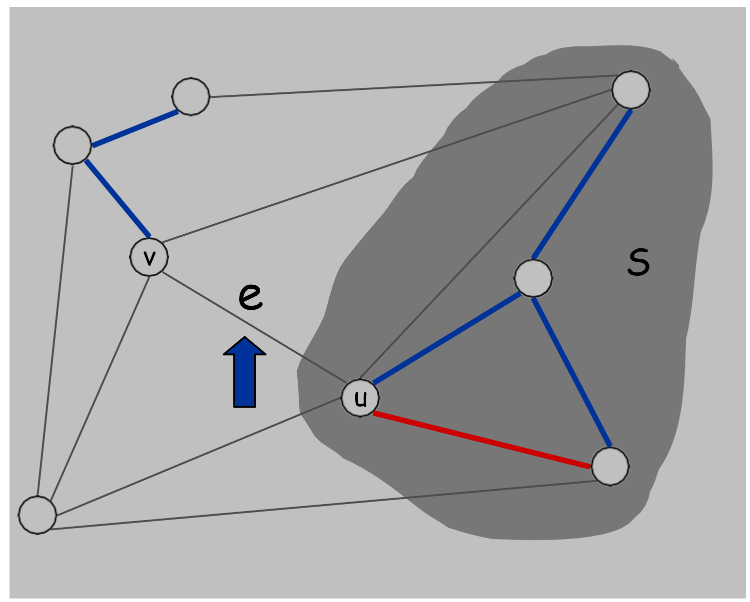
Start with T = ϕ. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Kruskal's algorithm. [Kruskal, 1956]

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



Case 1



Case 2

Recall: 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 MST T* contains e.

Implementation: Kruskal's Algorithm

Implementation. very efficient: use the "union-find" data structure.

- Build set T of edges in the MST.
- Maintain set for each connected component.
- O(m log n) for sorting and $O(m \underbrace{\alpha(m, n)}_{\text{essentially a constant}})$ for union-find.

$m \leq n^2 \Rightarrow \log m$ is $O(\log n)$

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

   foreach (u ∈ V) make a set containing singleton u
   for i = 1 to m
      (u,v) = eᵢ
      if (u and v are in different sets) {
         T ← T ∪ {eᵢ}
         merge the sets containing u and v
      }
   return T }
```

are u and v in different connected components?

merge two components

Lexicographic Tiebreaking

To remove the assumption that all edge costs are distinct: perturb all edge costs by tiny amounts to break any ties.

Impact. Kruskal and Prim only interact with costs via pairwise comparisons. If perturbations are sufficiently small, MST with perturbed costs is MST with original costs.

e.g., if all edge costs are integers, perturbing cost of edge e_i by i / n^2

Implementation. Can handle arbitrarily small perturbations implicitly by breaking ties lexicographically, according to index.

```
boolean less(i, j) {
   if      (cost(e_i) < cost(e_j)) return true
   else if (cost(e_i) > cost(e_j)) return false
   else if (i < j)                 return true
   else                            return false
}
```