CSE 6140/ CX 4140:

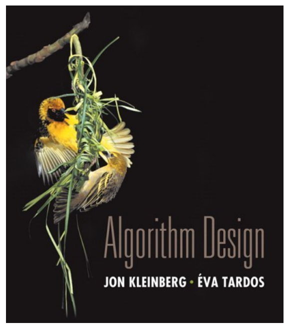
Computational Science and Engineering ALGORITHMS

Instructor: Anne Benoit

Visiting Associate Professor, CSE

Based on slides by Bistra Dilkina

KT4.5 Minimum Spanning Trees



Slides by Kevin Wayne.

Trees

Def. A path is a sequence edges which connects a sequence of nodes.

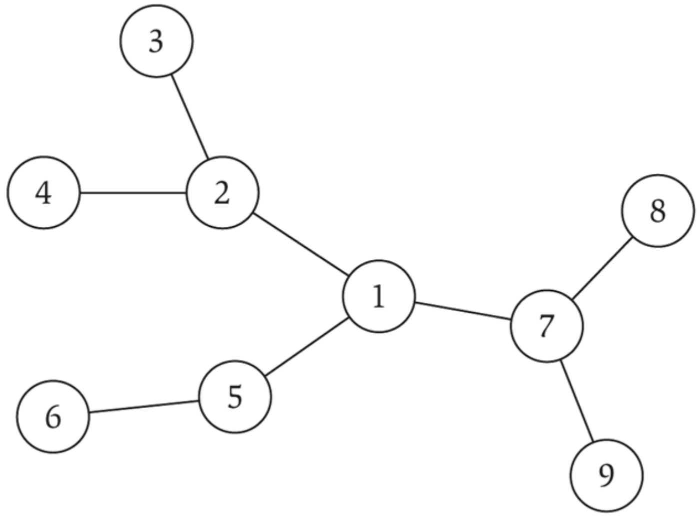
Def. A cycle is a path with no repeated nodes or edges other than the start and end nodes.

Def. An undirected graph is a tree if it is connected and does not contain a cycle.

Def. An undirected graph is a spanning tree if it is a tree and touches every vertex in G.

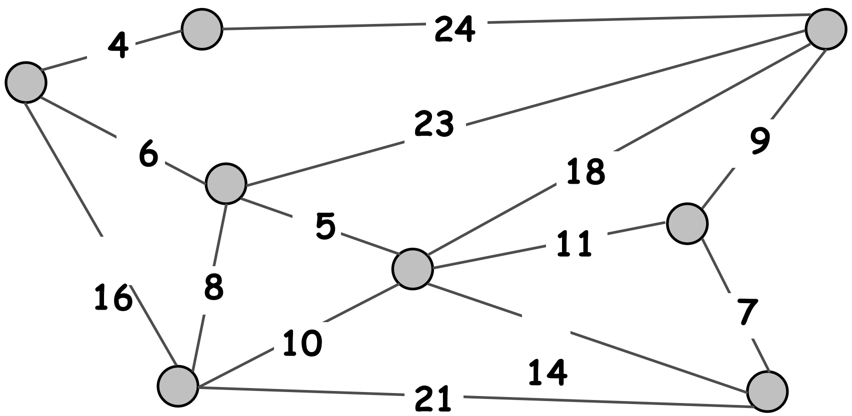
Theorem. Let G be an undirected graph on n nodes. Any two of the following statements imply the third.

- G is connected.
- G does not contain a cycle.
- G has $n-1$ edges.

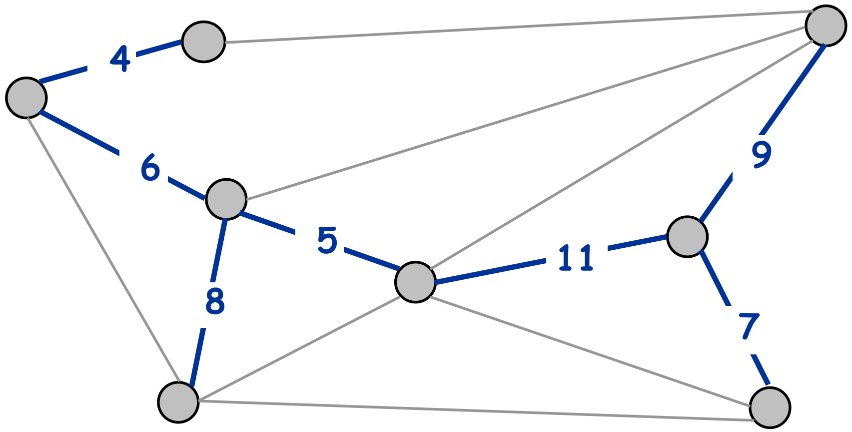


Minimum Spanning Tree

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.



$G = (V, E)$



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

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
  - 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
- Cluster analysis.

Greedy Algorithms

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.

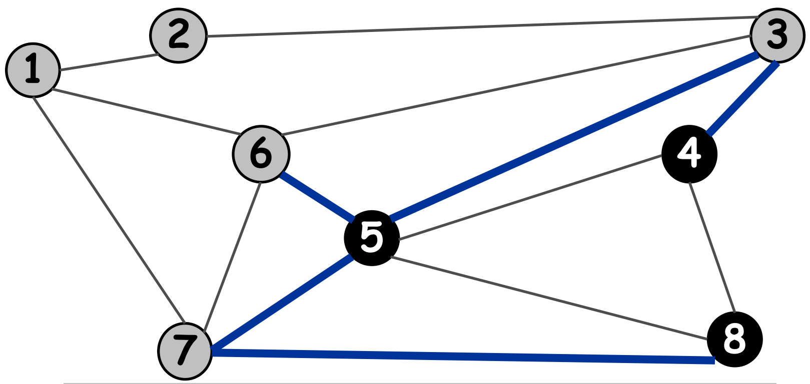
Remark. All three algorithms produce an MST.

Def. A cut is a partition of the nodes into two nonempty subsets *S* and *V* – *S*.

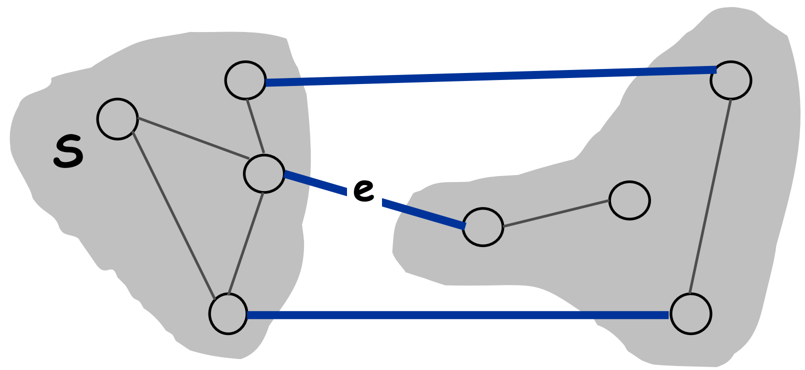
Def. The cutset of a cut *S* is the set of edges with exactly one endpoint in *S*.

Simplifying assumption. All edge costs c_e are distinct.

Cut property. Let S be any subset of nodes, and let e be <u>the</u> min cost edge with exactly one endpoint in S. Then every MST contains e.



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



e is in the MST

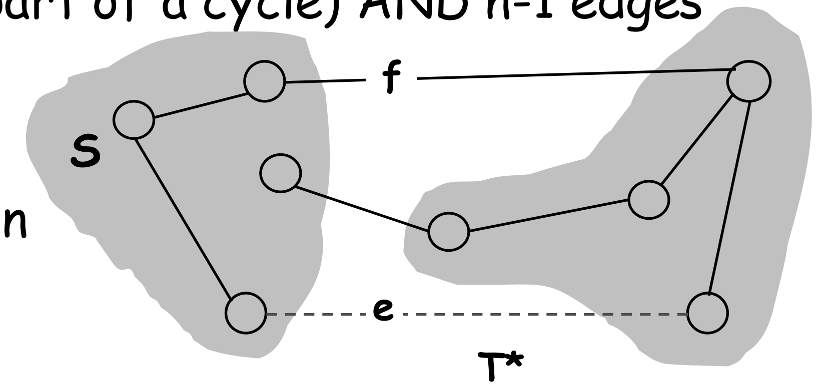
Cut Property

Cut property. Let S be any subset of nodes, and let $e=(u,v)$ be the min cost edge with exactly one endpoint in S. Then every MST T^* contains e.

Simplifying assumption. All edge costs c_e are distinct.

Pf. (exchange argument)

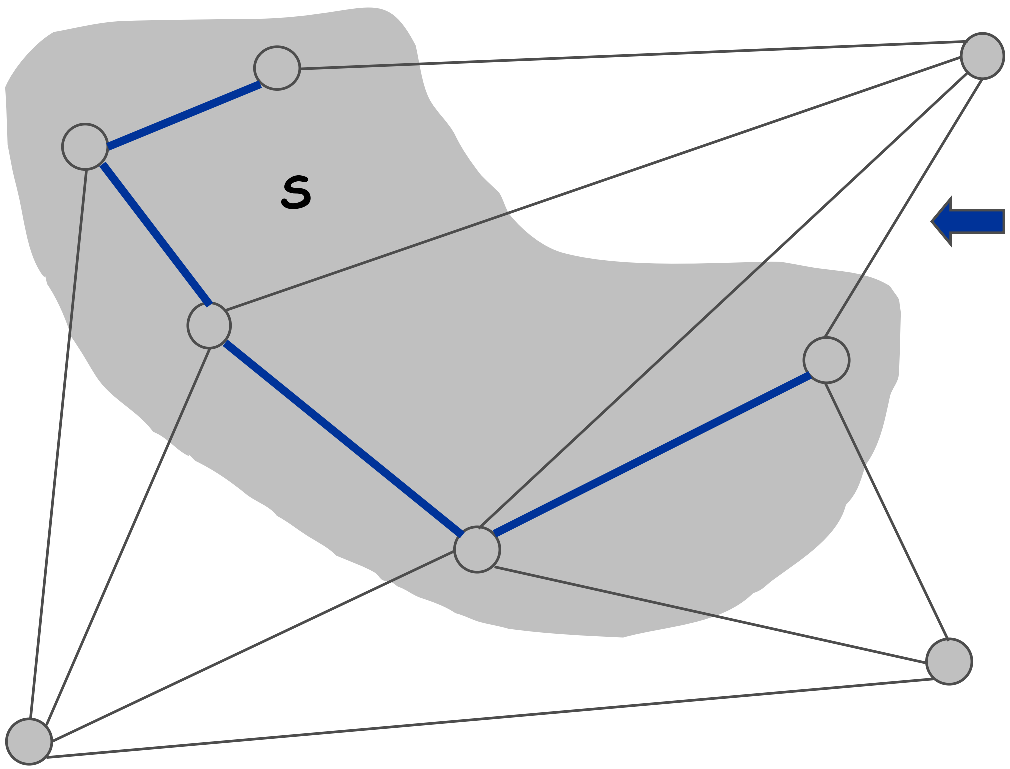
- Given: e is the min cost edge in the cutset for a set S, T^* is an MST
- Suppose e does not belong to T^*, and let's see what happens.
- The endpoints u and v of e must be connected by a path in T^* (spanning)
- => Adding e to T^* creates a cycle C in T^*
- Edge e is both in the cycle C and in the cutset corresponding to S $\Rightarrow$ there exists another edge, say f, that is in both C and cutset of S.
- $T' = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree, because:
  - All n nodes are connected (f,e were part of a cycle) AND n-1 edges
- Since $c_e < c_f$, $cost(T') < cost(T^*)$.
- This is a contradiction since T^* is an optimal spanning tree, our assumption about e not being in T^* must be wrong.



Prim's Algorithm

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

- Initialize S = any node.
- (Apply cut property to S.)
- Add to tree the min cost edge (u,v) in cutset corresponding to S, and add one new explored node u to S.



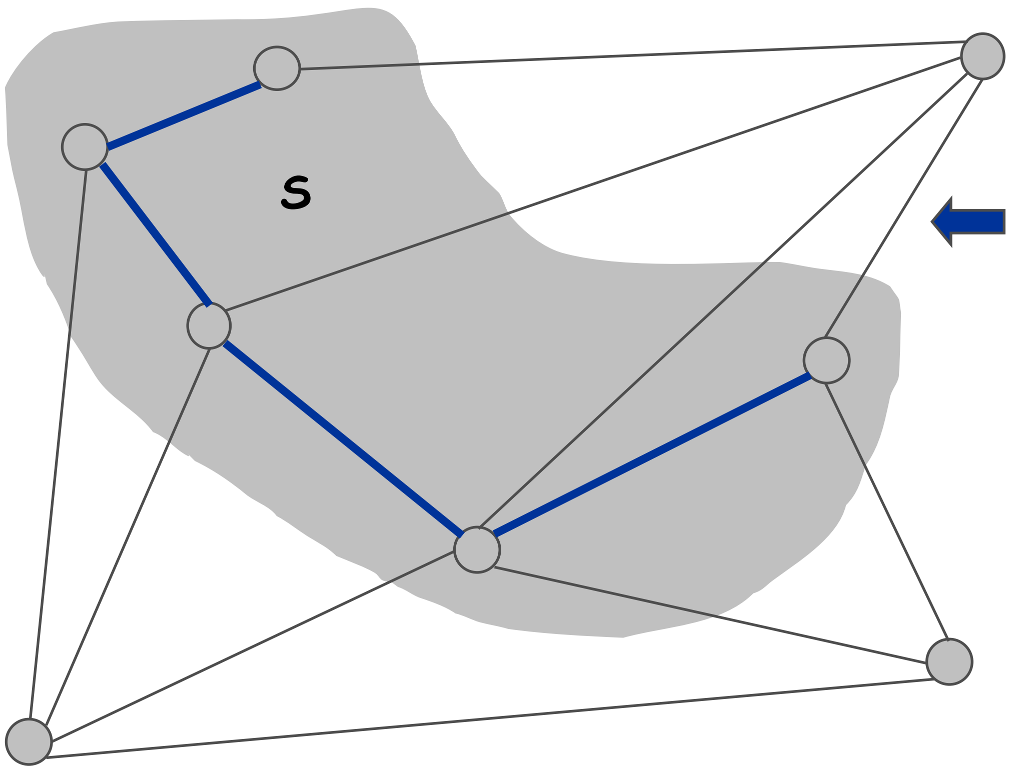
Prim's Algorithm: Proof of Correctness

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

- Initialize S = any node.
- (Apply cut property to S.)
- Add to tree the min cost edge (u,v) in cutset corresponding to S, and add one new explored node u to S.

1) Every edge added satisfies the Cut Property: by design

2) It produces a spanning tree: alg stops when S=V



Implementation: Prim's Algorithm

Implementation. Use a priority queue (as for 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.

Shortest edge between v and a node in explored set S

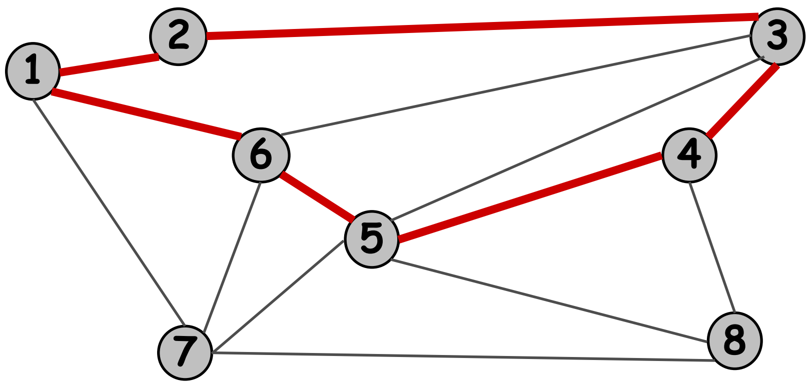
```
Prim(G, c) {
   foreach (v ∈ V) a[v] ← ∞
   Initialize an empty priority queue Q
   foreach (v ∈ V) insert v onto Q
   Initialize set of explored nodes S ← φ

   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
}
```

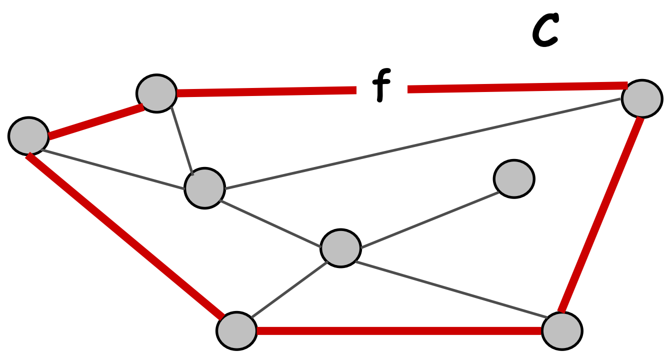
Simplifying assumption. All edge costs c_e are distinct.

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

Cycle property. Let C be any cycle, and let f be the max cost edge belonging to C. Then every MST does not contain f.



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



f is not in the MST

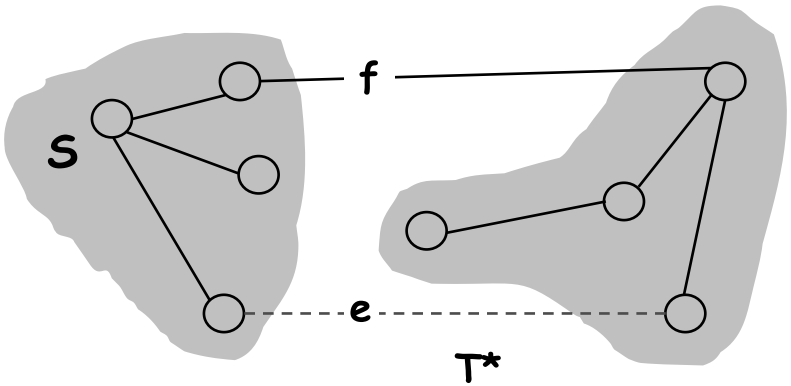
Cycle Property

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

Simplifying assumption. All edge costs c_e are distinct.

Pf. (exchange argument)

- Given: f is the max cost edge in a cycle C, T* is an MST
- Suppose f belongs to T*, and let's see what happens.
- Deleting f from T* disconnects T* and 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 (same argument as before).
- Since $c_e < c_f$, cost(T') < cost(T*).
- This is a contradiction since T* is an optimal spanning tree, hence our assumption about f is wrong.



Reverse-Delete Algorithm: Proof of Correctness

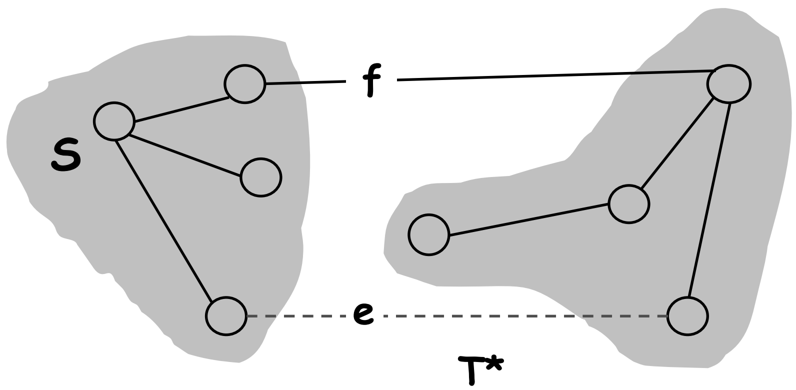
The algorithm. Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

Th. This algorithm produces an MST of G.

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

Pf.

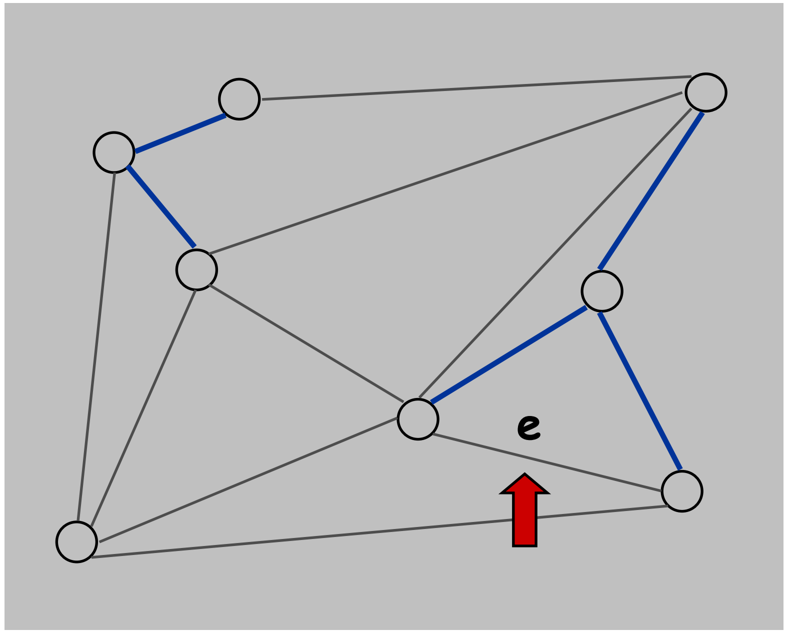
- Edges removed do not belong to any MST.
- The output is a spanning tree: connected & no cycle.



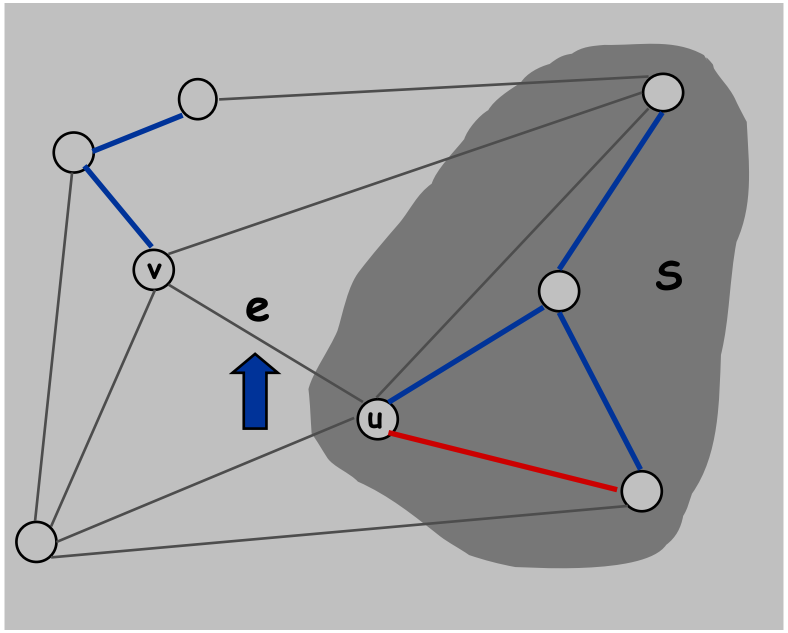
Kruskal's Algorithm: Proof of Correctness

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: because of sorted order, e must be maxcost in cycle).
- Case 2: Otherwise, insert e = (u, v) into T (according to cut property: for set S = set of nodes in u's connected component in current set T).



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 $O(m \underbrace{\alpha(m, n)}_{\text{essentially a constant}})$ for union-find.

$m \le n^2 \Rightarrow \log m$ 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

Simplifying assumption. All edge costs c_e are distinct.

Impact. Kruskal and Prim only interact with costs via pairwise comparisons.

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

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

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