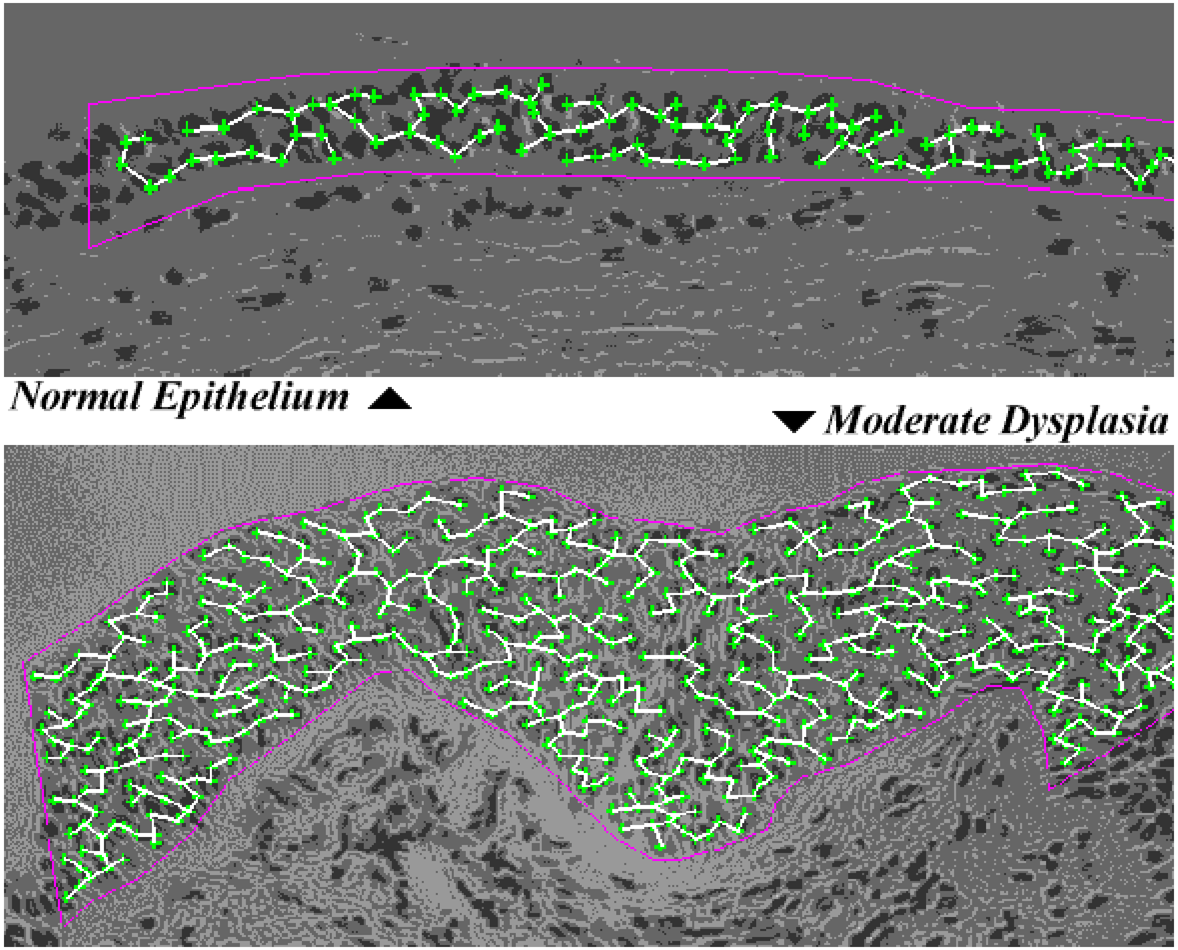
4.5 Minimum Spanning Tree

- Minimum Spanning Tree Problem (and applications)
- Cut-property and Cycle-property (inc. proof)
- MST algorithms:
  - Prim
  - Kruskal and Union-Find
  - Reverse-Delete

4.5 Minimum Spanning Tree

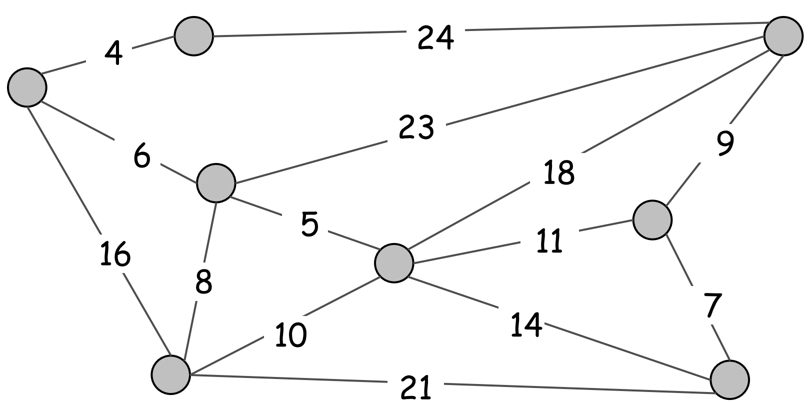


http://www.bccrc.ca/ci/ta01_archlevel.html
(see also Blackboard – External Links)

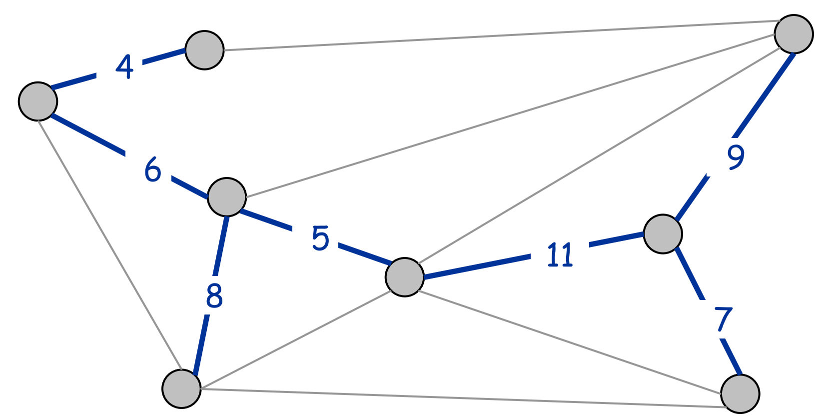
Minimum Spanning Tree

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

- T is a tree
- T connects all vertices, and
- the sum of edge weights is minimized



$G = (V, E)$



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

Cayley's Formula. There are n^{n-2} spanning trees of a fully connected graph.

can't solve by brute force

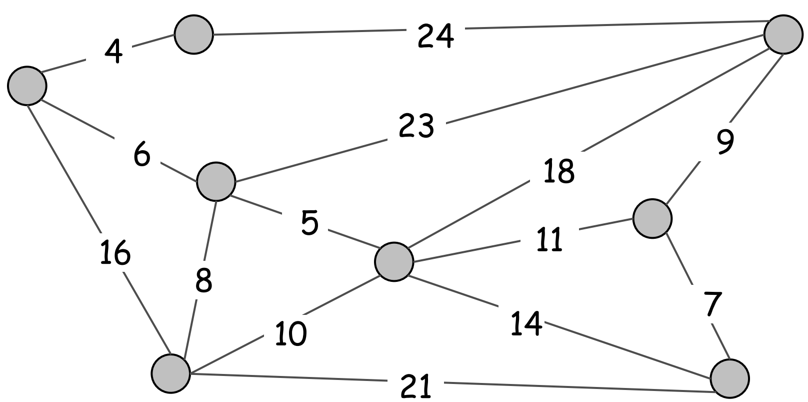
Applications

MST is fundamental problem with diverse applications.

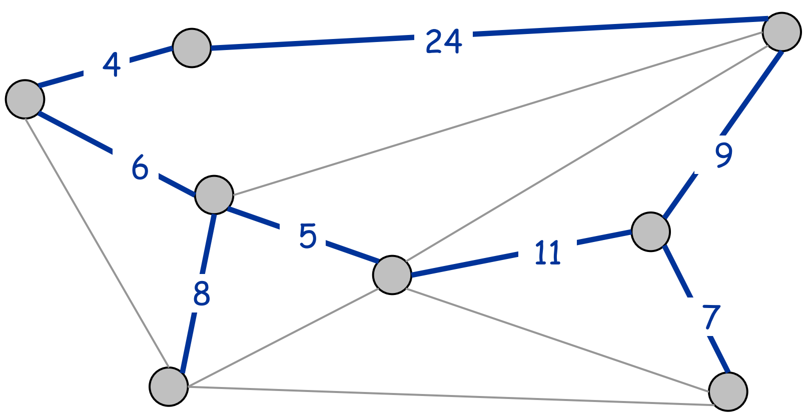
- Network design.
  - telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-hard problems.
  - traveling salesperson problem, Steiner tree
- Indirect applications.
  - max bottleneck paths
  - LDPC codes for error correction
  - image registration with Renyi entropy
  - 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.

Minimum Spanning Tree

Q. Is T a minimum spanning tree?



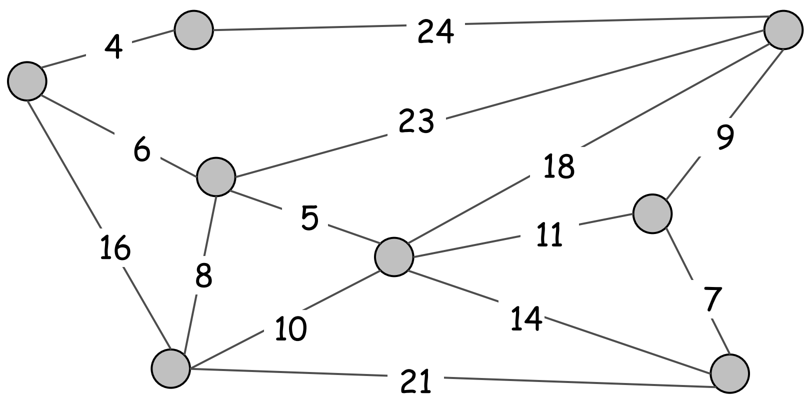
G = (V, E)



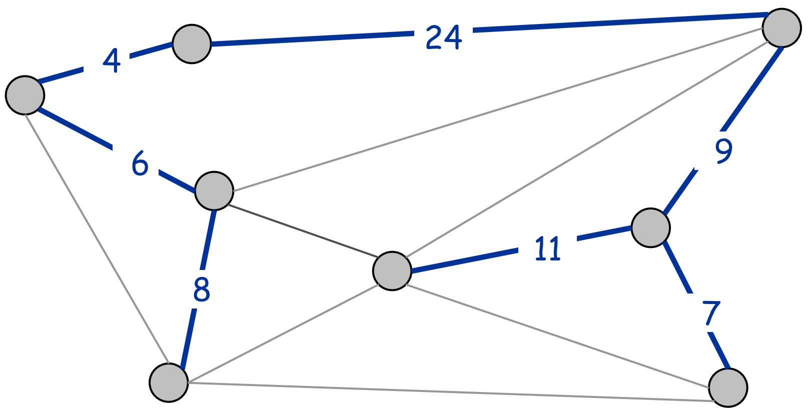
T

Minimum Spanning Tree

Q. Is T a minimum spanning tree?



$G = (V, E)$

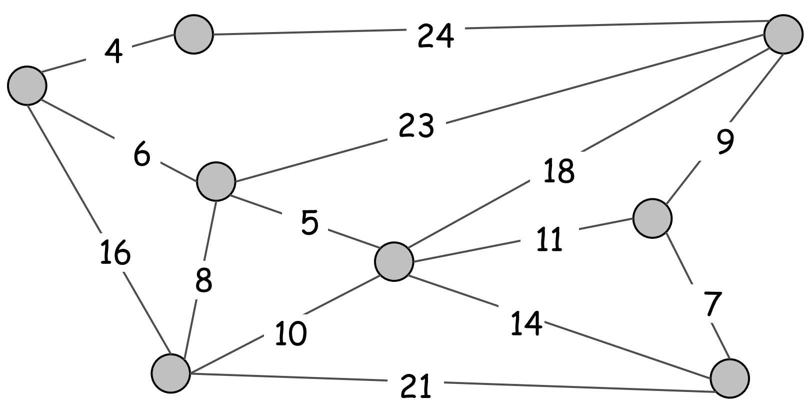


T

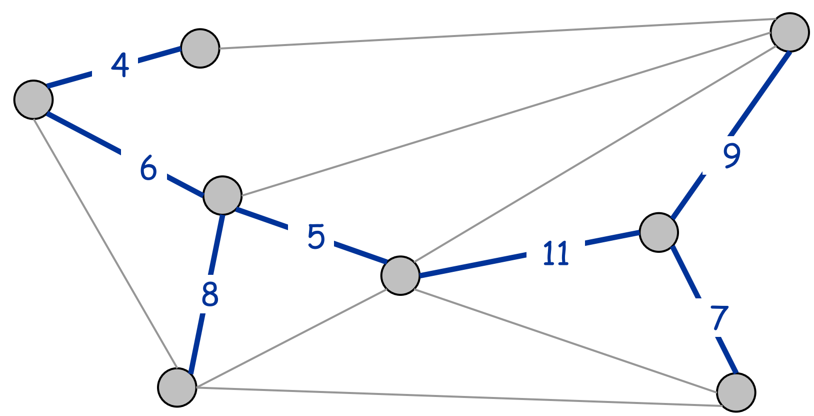
Minimum Spanning Tree

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

- T is a tree
- T connects all vertices, and
- the sum of edge weights is minimized



$G = (V, E)$



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

Q. How to find such a minimum spanning tree greedily? (1 min)

Greedy Algorithms

Kruskal's algorithm. Start with T = $\varnothing$. 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.

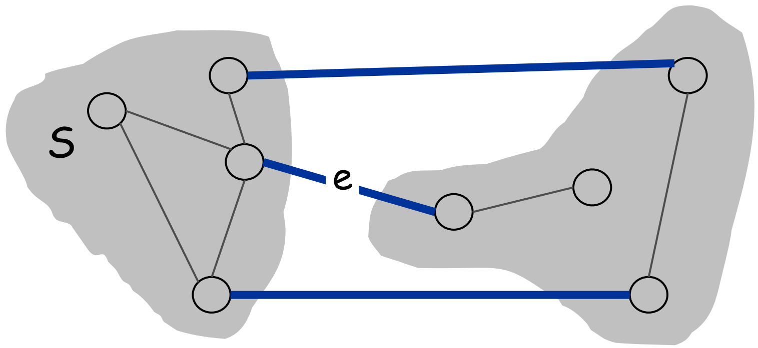
(Boruvka, 1926). Was first. (For each vertex add cheapest edge.)

Remark. All algorithms produce an MST. We will prove this for the first three above using two general properties: the cut property and the cycle property.

Simplifying assumption. All edge costs c_e are distinct.

Q. Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Should e be in every MST?

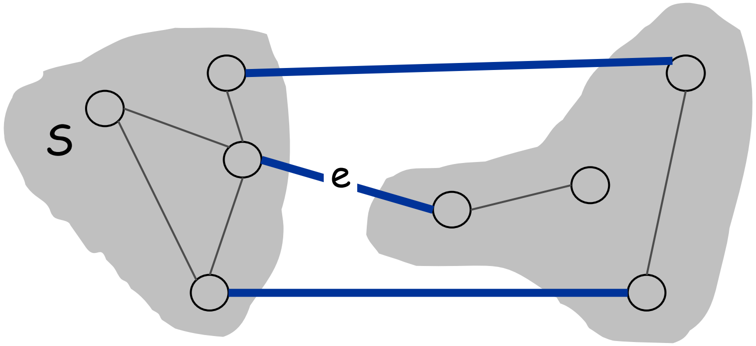
A.



Simplifying assumption. All edge costs c_e are distinct.

Q. Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Should e be in every MST?

A. Yes ➔ cut property

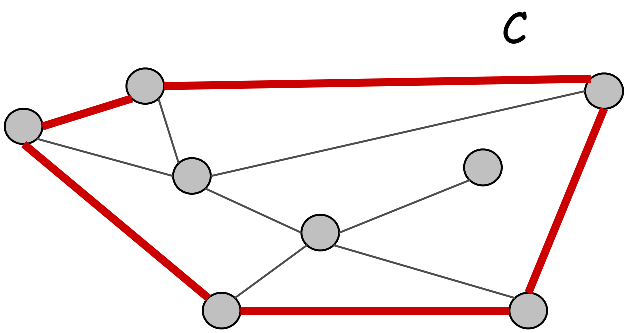


e is in every MST

Simplifying assumption. All edge costs c_e are distinct.

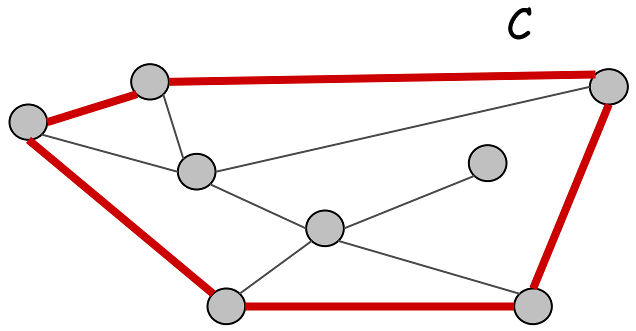
Q. Let C be any cycle, does a MST exist that has all of C's edges?

A.



Simplifying assumption. All edge costs c_e are distinct.

Q. Let C be any cycle, does a MST exist that has all of C's edges?
A. No.
Q. Which one should be not in the MST?
A.



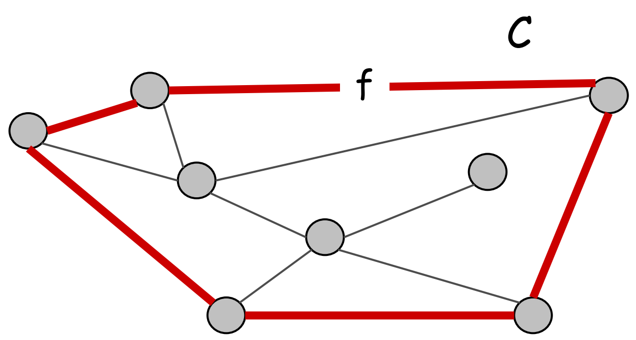
Simplifying assumption. All edge costs c_e are distinct.

Q. Let C be any cycle, does a MST exist that has all of C's edges?

A. No.

Q. Which one should be not in the MST?

A. The max cost ➔ cycle property



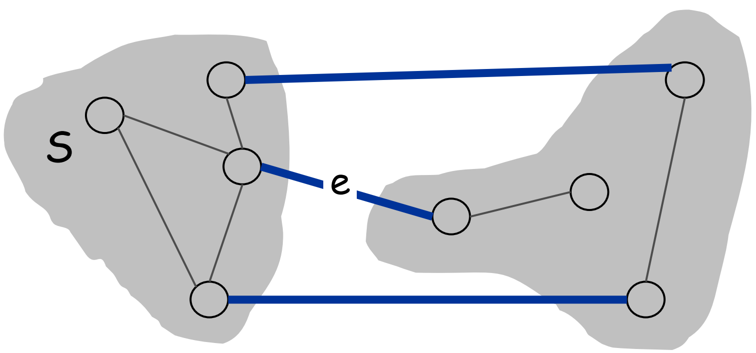
f is not in the MST

Simplifying assumption. All edge costs c_e are distinct.

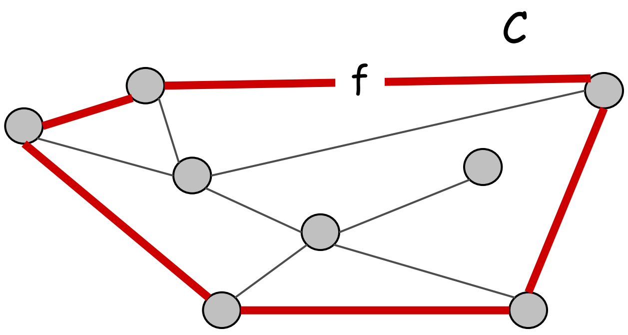
Cut property. Let S be any cut, and let e be the min cost edge with exactly one endpoint in S. Then the MST contains e.

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

Q. How to prove this? ...



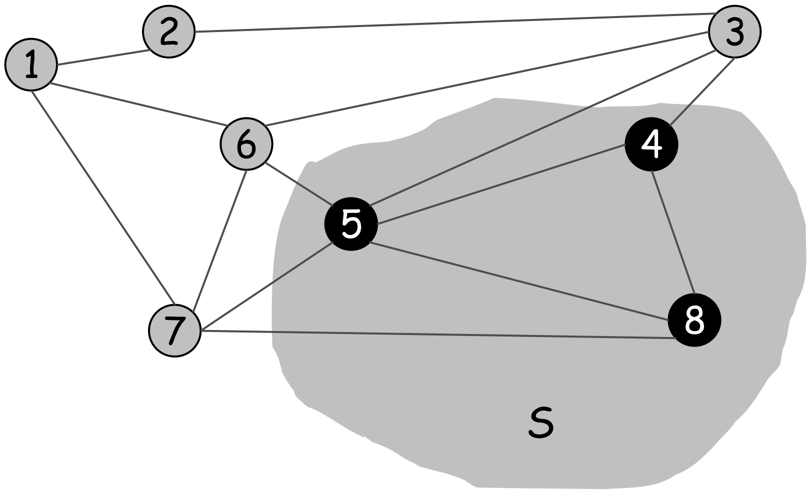
e is in the MST



f is not in the MST

Cut and cutset

Cut. A cut is a subset of nodes S.

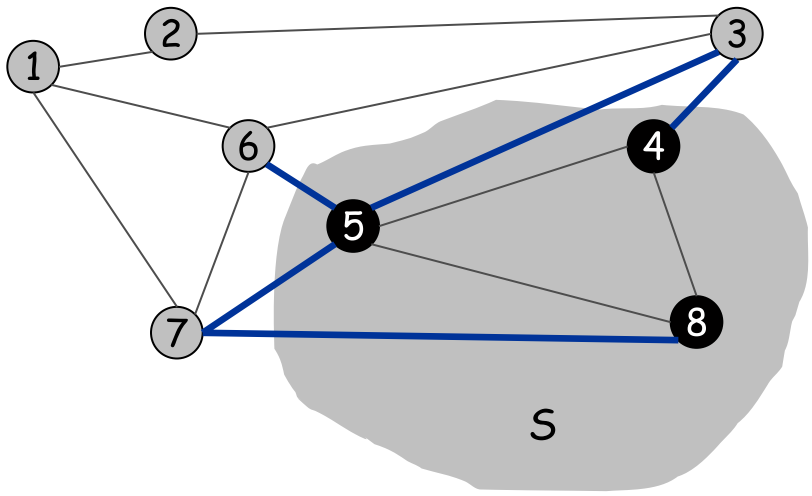


Cut S = { 4, 5, 8 }

Cut and cutset

Cut. A cut is a subset of nodes S.

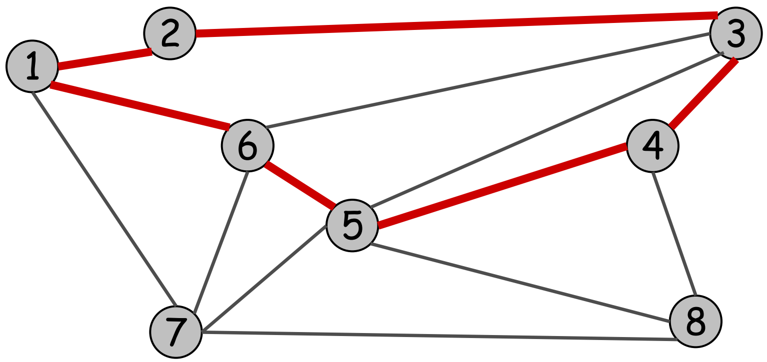
Cutset. A cutset D of a cut S is the subset of (cut)edges with exactly one endpoint in S.



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

Cycles and Cuts

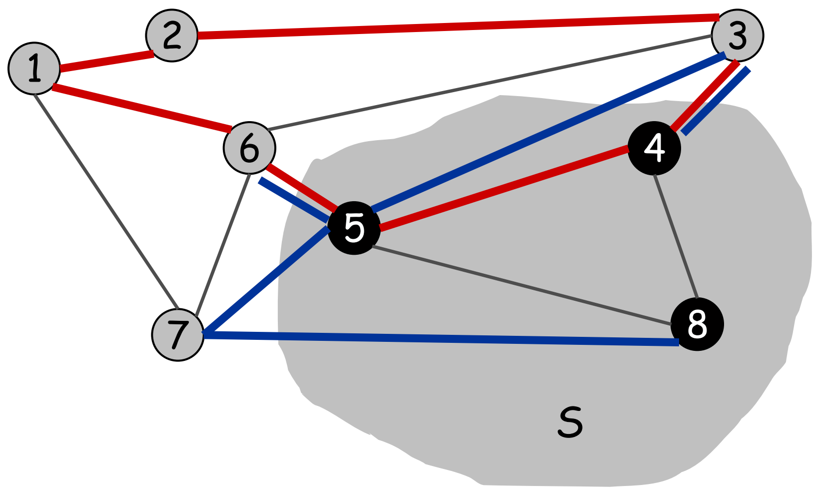
Cycle. Set of edges the form a-b, b-c, c-d, …, y-z, z-a.



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

Cycle-Cut Intersection

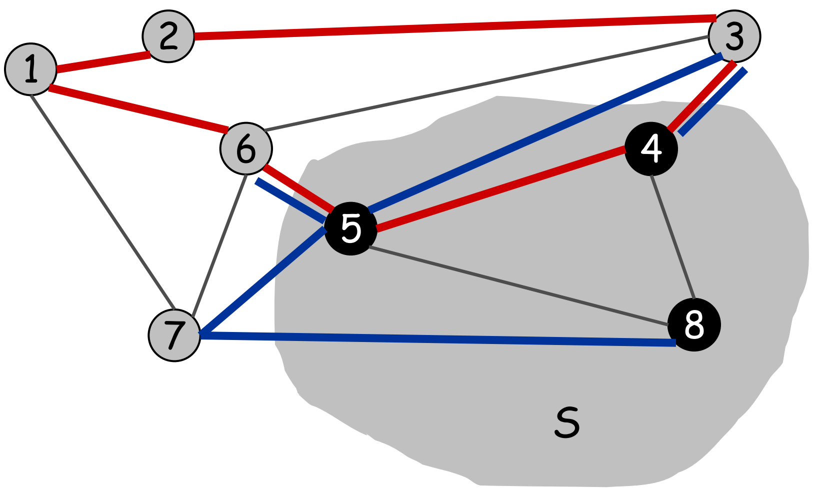
Q. Consider the intersection of a cycle and a cutset. How many edges are there in such an intersection? (1, 2, odd, even)



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

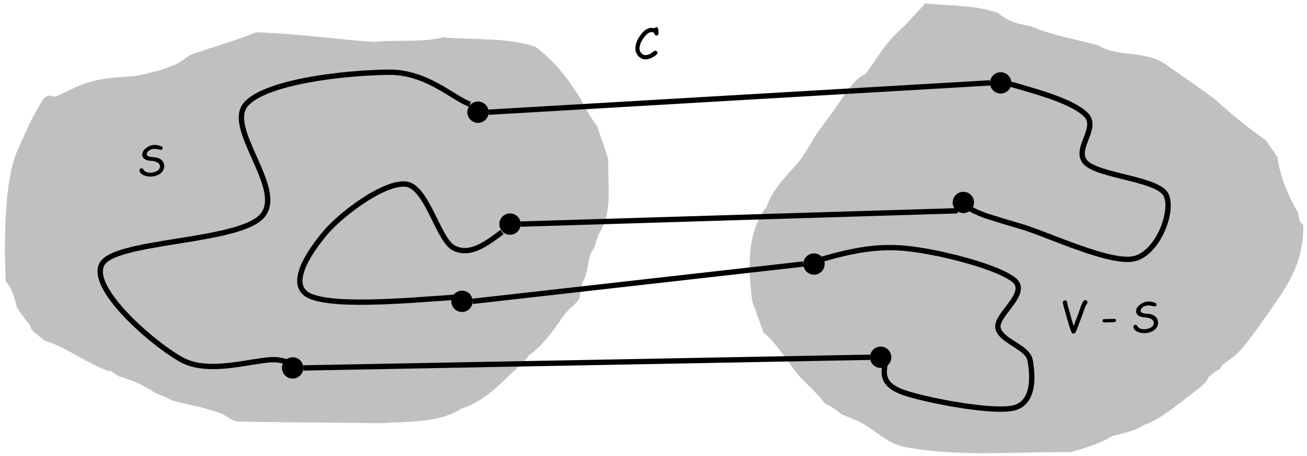
Cycle-Cut Intersection

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. Walk along cycle from a node s∈S: for every edge leaving S, there should (first) be an edge to a node in S before returning to s.



Cut property

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.

Q. What proof technique to use?

Cut property

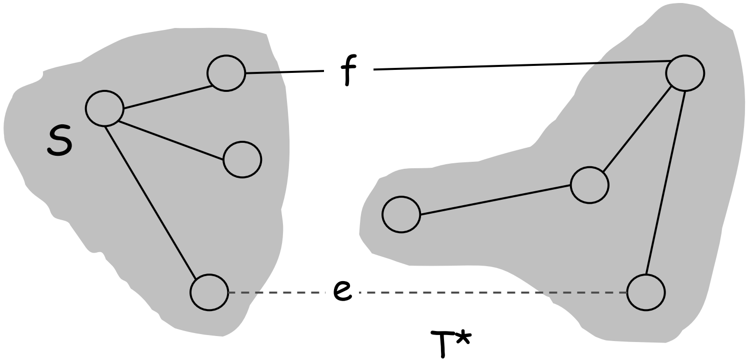
Simplifying assumption. All edge costs c_e are distinct.

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. (by contradiction)

- Suppose e does not belong to T*, and let's see what happens.

- This is a contradiction. ▪



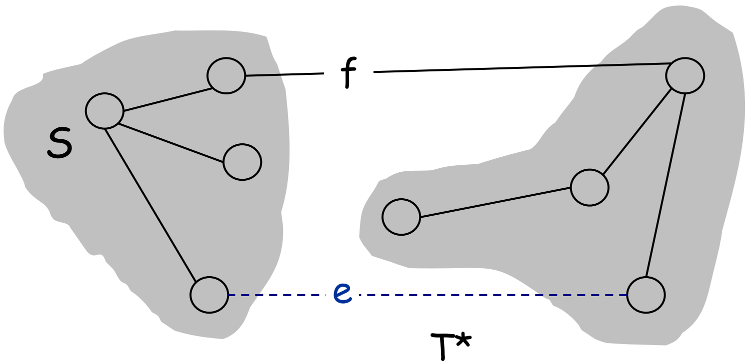
Cut property

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. (by contradiction)

- 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 $c_e < c_f$, $\text{cost}(T') < \text{cost}(T^*)$.
- This is a contradiction. ▪



This proof can be found on page 145.

Cycle property

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. (1 min)

Q. What proof technique to use?

Cycle property

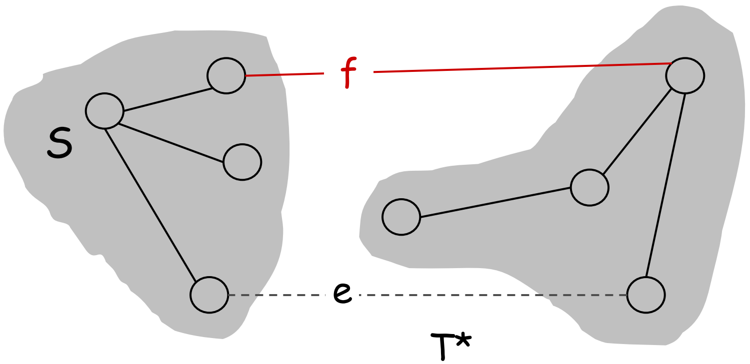
Simplifying assumption. All edge costs c_e are distinct.

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) (1 min)

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

- This is a contradiction. ▪



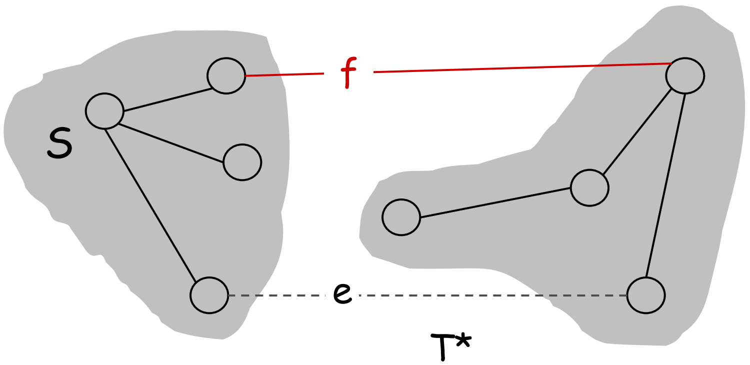
Cycle property

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*, 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 $c_e < c_f$, $\text{cost}(T') < \text{cost}(T^*)$.
- This is a contradiction. ▪



This proof can be found on page 147-148.

Generic MST Algorithm (blue rule, red rule)

Blue rule: 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. Color e blue.

Red rule: 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. Color f red.

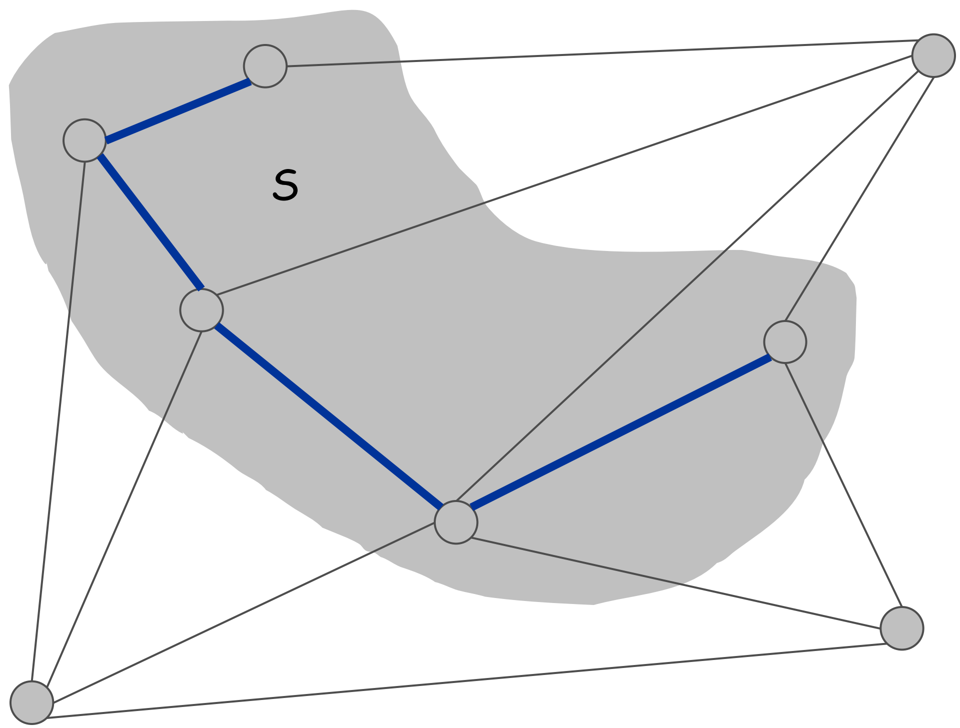
Generic greedy algorithm.
Apply these rules until all edges are colored.

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 min cost edge in cutset corresponding to S to T, and add one new explored node u to S.

Q. Implementation is similar to which algorithm you have already seen?



Implementation: Prim's Algorithm

Implementation. Use a priority queue a la Dijkstra.

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

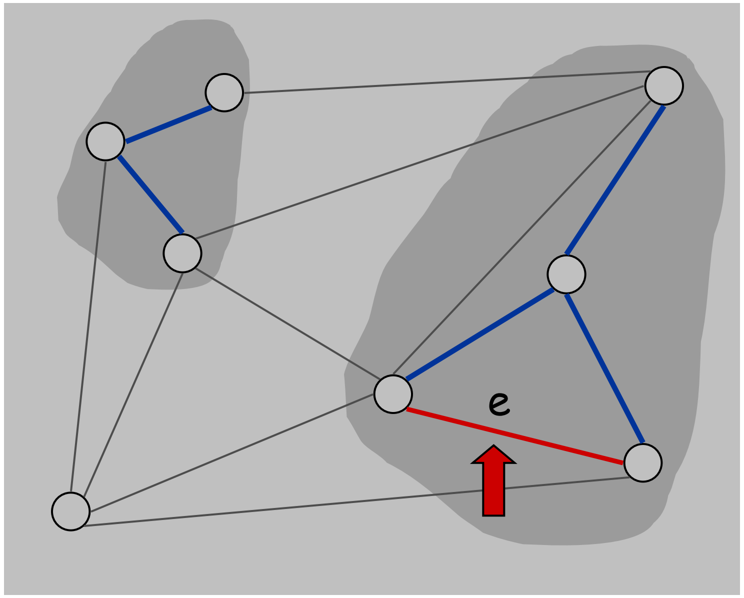
```
Prim(G, c) {
   foreach (v ∈ V) a[v] ← ∞; e[v] ← φ
   foreach (v ∈ V) insert v into Q
   Initialize set of explored nodes S ← φ, T ← φ

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

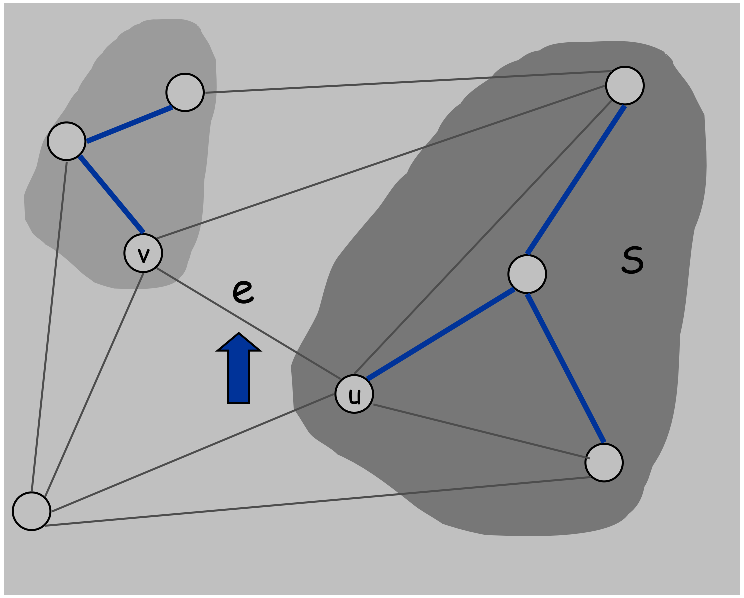
Kruskal's Algorithm: Proof of Correctness

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

```
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
}
```

=find(v)?

Union-Find

Union-Find.

Efficient data structure to do two operations on

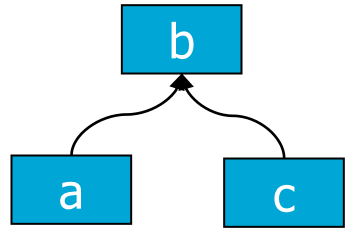
- Union: merge two components
- Find: give the representative of the component

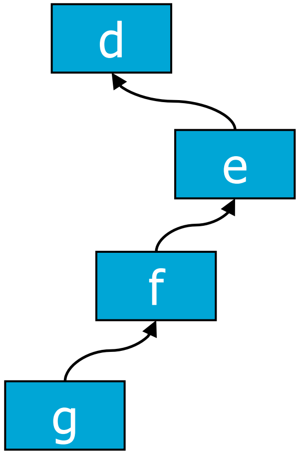
Q. How to implement efficiently?

Union-Find

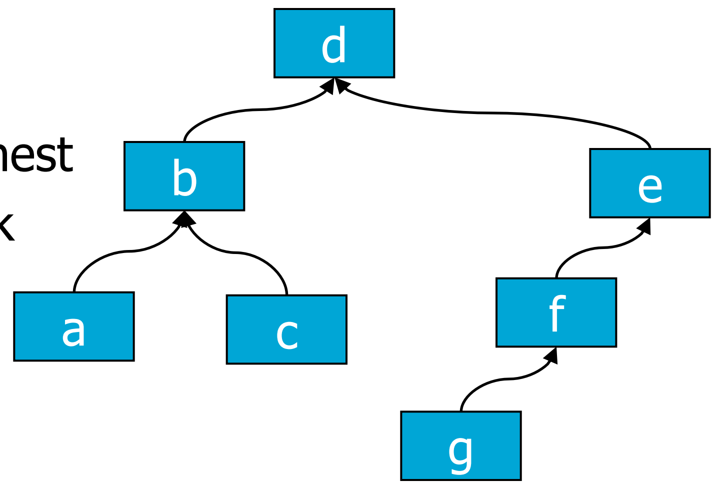
Union-Find.

- Represent component by tree

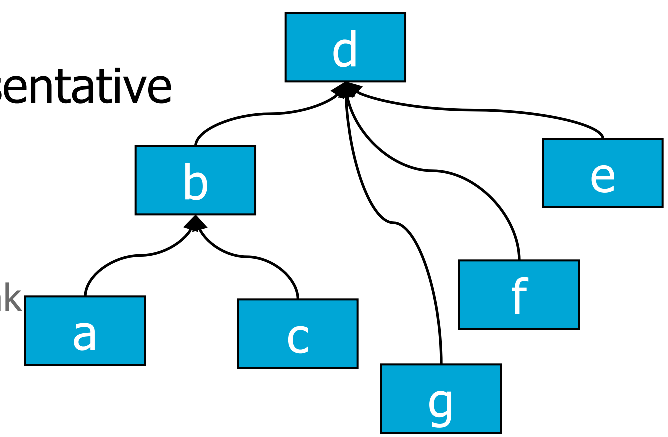




- Union: merge two components
  - assign each node a rank
  - place root with lowest rank under highest
  - increase rank of new root if equal rank



- Find: give the representative
  - path compression
    (eg find(g))
  - btw, do not update rank



TU Delft

Implementation: Kruskal's Algorithm

Implementation. Using the union-find data structure.

- $O(m \log n)$ 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
      u_root ← find(u)
      v_root ← find(v)
      if (u_root != v_root) {
         T ← T ∪ {eᵢ}
         union( u_root, v_root )
      }
   return T
}
```

$O(m)$

$O(\alpha(m, n))$

$O(1)$

Lexicographic Tiebreaking

Q. How to remove the assumption that all edge costs are distinct?

A.

Lexicographic Tiebreaking

Q. How to remove the assumption that all edge costs are distinct?
A_1. Perturb all edge costs by tiny amounts to break any ties.
A_2. Break ties using index.

A_1. 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

A_2. 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
}
```