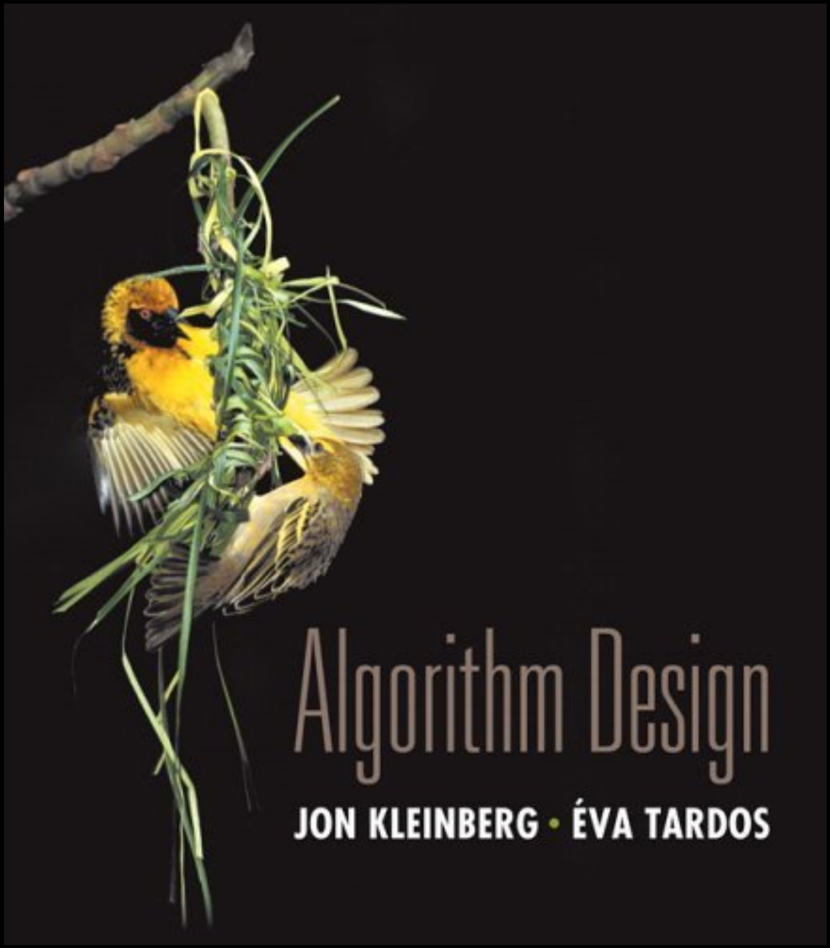


Chapter 4

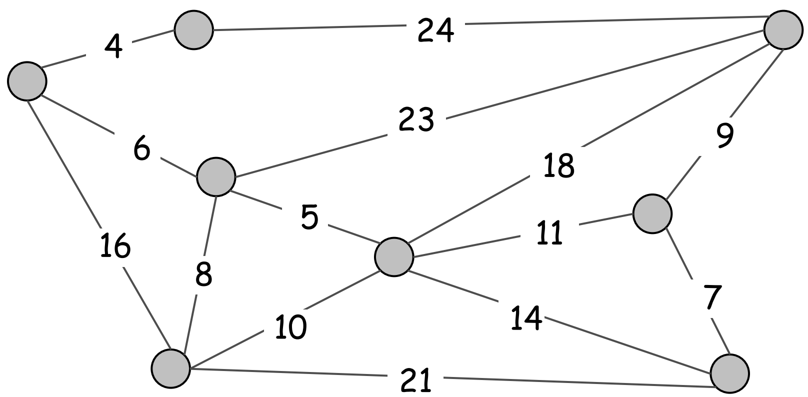
Greedy Algorithms

Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

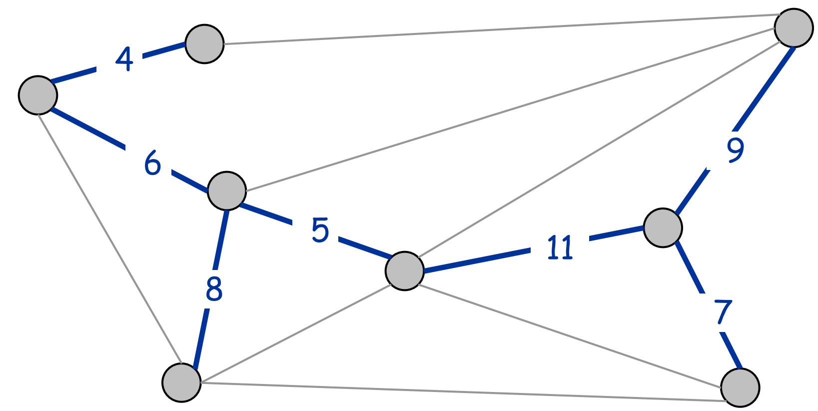
4.5 Minimum Spanning Tree

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

The MST Problem: Formal Definition

Input

- Symmetric, connected weighted graph **$G=(V,E,w)$**, where:
- **V** = Set of nodes, **E** = Set of edges, edge cost function **$c: E \rightarrow R^+$**

Feasible solutions

- Any Spanning Tree of **G**: **F** with **$F \subseteq E$**

Cost of feasible solutions:

- Cost (to minimize) $c(T) = \sum_{e \in T} c(e)$

- Check your **learning** level by answering the following questions:
  - How many bits for the input representation?
  - What is a Spanning Tree (ST) of a connected graph?
  - Do you know any algorithm to compute a generic ST?

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.

GENERAL APPROACH: Greedy Algorithms

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

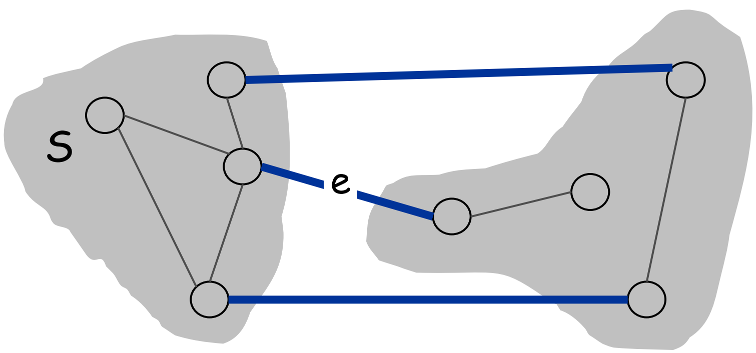
Reverse-Delete algorithm. Start with **T** = **E**. Consider edges in **descending** order of cost. **Delete** edge **e** from **T** unless doing so would **disconnect T**.

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

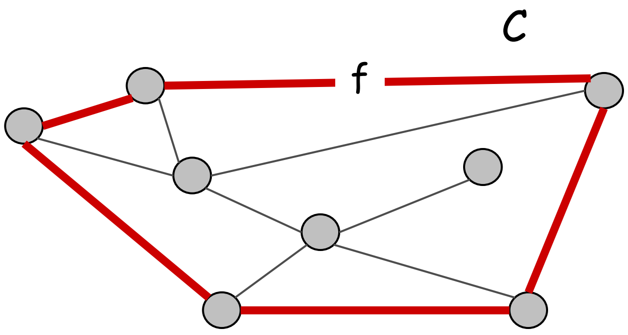
MAIN THEOREM (Informal Statement). ***All*** three algorithms produce an ***MST***.

PROOF of the MAIN THEOREM.

We will use TWO GENERAL PROPERTIES OF GRAPHS:



e is in the MST



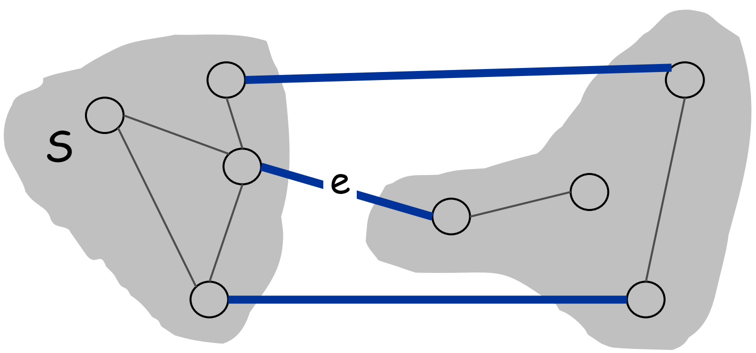
f is not in the MST

Greedy Algorithms: Analysis

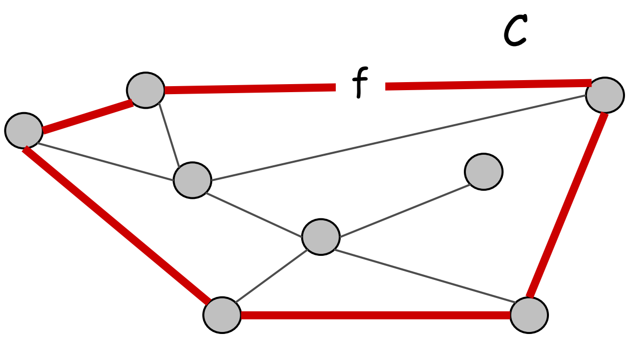
Simplifying assumption. **All edge costs c(e) are distinct.**

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 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.



e is in the MST



f is not in the MST

OSSERVA: Regola del CUT-SET (Taglio) e Regola del CYCLE (CICLO)

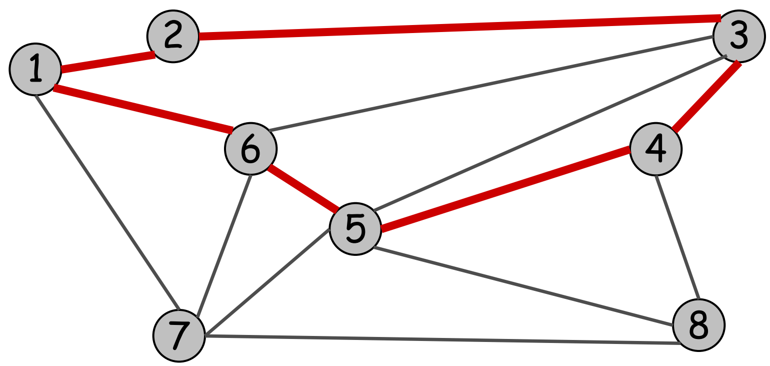
CUT RULE: *Scegli una qualsiasi partizione* **(S,V-S)** di **V** che non è attraversata da **archi blu**. Tra tutti gli archi non ancora colorati che sono nel **cut-set E(S,V-S)** , sceglione uno di costo minimo e coloralo di blu (cioè, **aggiungilo alla soluzione T**).

CYCLE RULE: Scegli un ciclo nel grafo che non contiene **archi rossi**. Tra tutti gli archi non ancora colorati del ciclo, scegline uno di costo massimo e coloralo di rosso (cioè, **scartalo per sempre**).

To prove the two properties, we use another simple property:

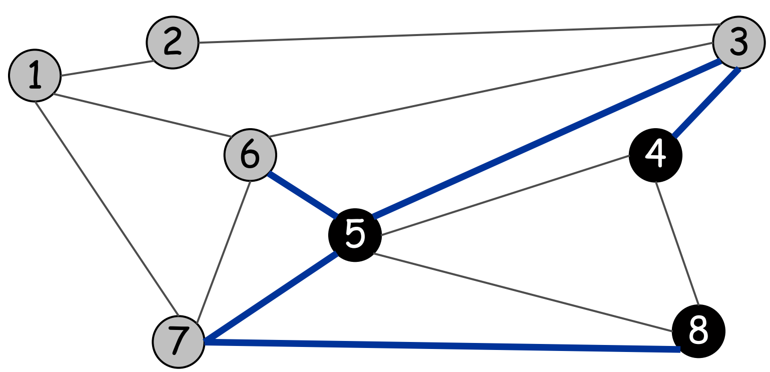
Cycles and Cuts

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

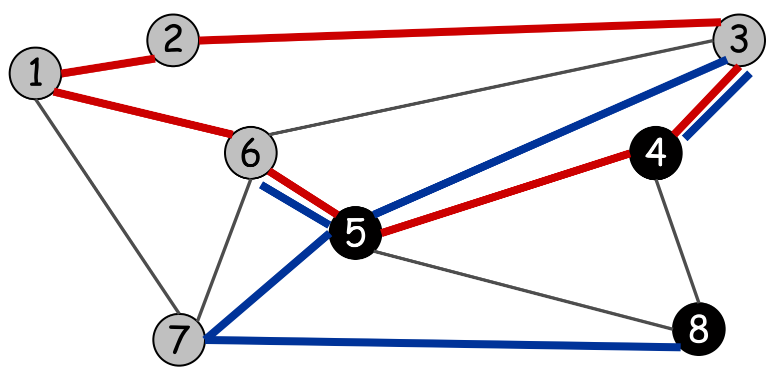
Cutset. A **CUT** is determined by some S. The corresponding **CUTSET D** is the subset of edges with exactly one endpoint in **S**.



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

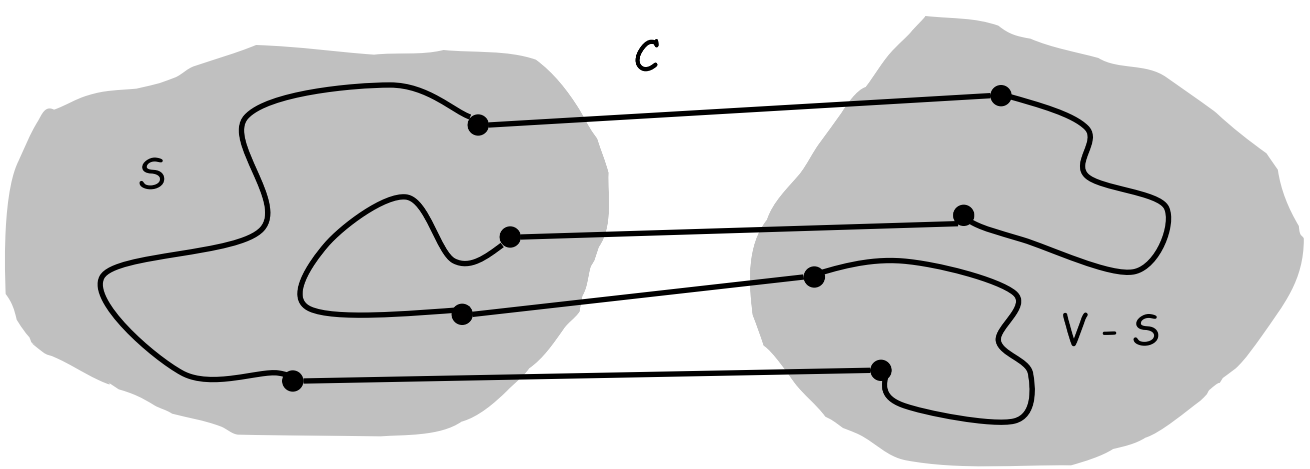
PROPERTY: Cycle-Cut Intersection

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. (by picture)

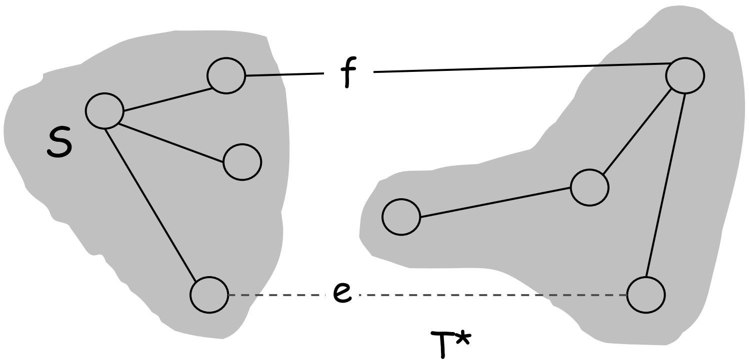


CUT PROPERTY: PROOF

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 **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**.
- Consider **T'** = **T*** $\cup$ **{ e }** - **{ f }**: it is also a **spanning tree!**
- Since $c_e < c_f \rightarrow$ cost(T') < cost(T*).
- This is a contradiction. ▪

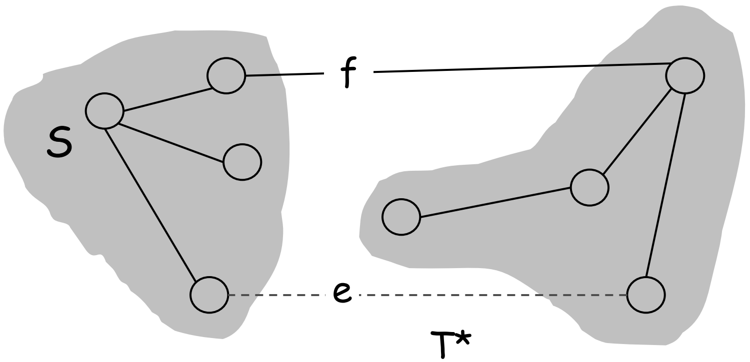


Simplifying assumption. All edge costs c_e are distinct.

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



Implementation: Prim's Algorithm

Implementation. Use a priority queue ala 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

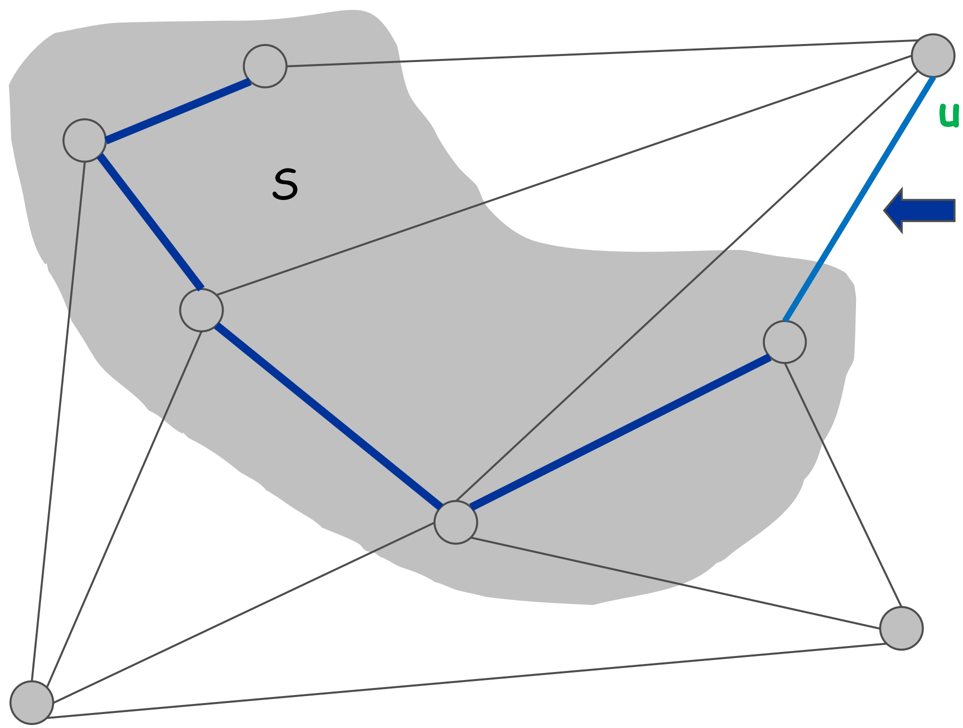
```
Prim(G, c) {
   foreach (v ∈ V) a[v] ← ∞
   Initialize an empty priority queue Q
   foreach (v ∈ V) insert v onto Q
   Initialize set of explored nodes S ← φ

   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 (c_e < a[v]))
             decrease priority a[v] to c_e
}
```

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 D(S) to **T**, and add **one** new explored node u to S.



Key Facts of the Proof you need to learn:

What is S ? How growes?

- At each round, a new unexplored node is inserted in **S**. So, at each round of the WHILE loop, **|S|** increases by 1: $S_0 = \{u_1\}$, … $S_t = \{u_1, u_2, \ldots, u_t\}$. . , $S_{n-1} = V$,

- **Connectivity of G** → The algorithm **terminates**! AND when it terminates, T **spans all nodes** of **V** (It is a GRAPH SEARCH!).

- **Why T is an *MST*?** At each WHILE loop, apply the **CUT property**! Where? On the (current) CUT :

- (S_t={explored nodes till round t}, **V** - S_t) , t= 1, …, n-1

Time complexity of Prim's Algorithm

THM: $O(n^2)$ with an array; $O(m \log n)$ with a binary heap.

Proof: DO AS EXERCISE!

SUGGESTIONS: give answers to :

- How many times a node is explored ?
- How do you represent Q?
- Which operations on Q for every new explored node? How many? How can you implement them?