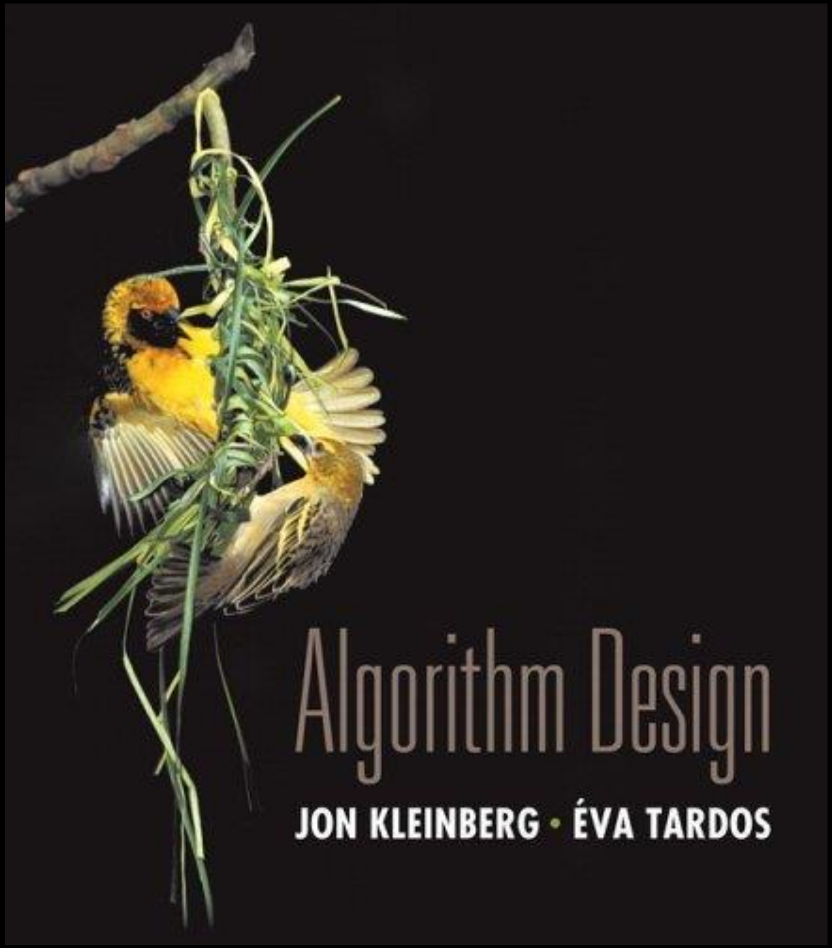


Chapter 4

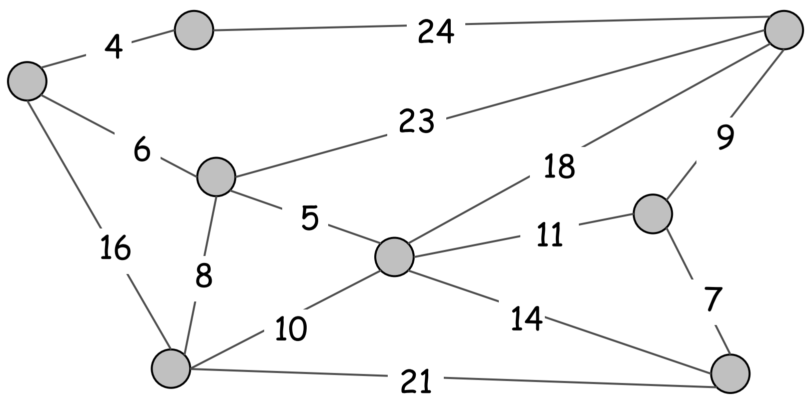
Greedy Algorithms

Slides by Kevin Wayne.

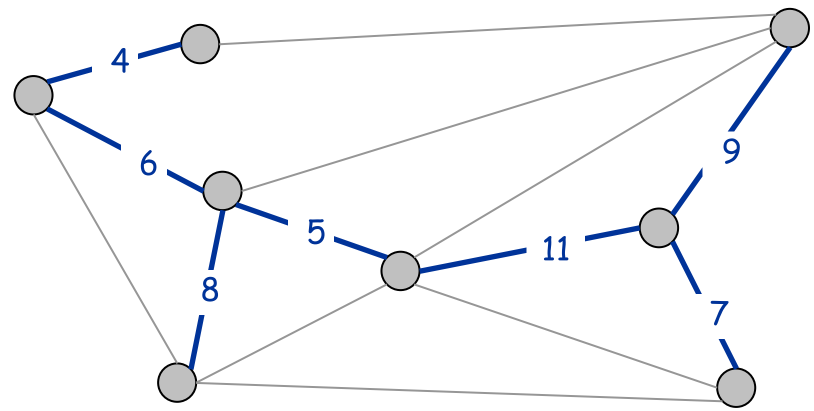
4.5 Minimum Spanning Tree

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 Minimum Spanning Tree (MST) problem

- Input:
  - a connected weighted undirected graph $G = (V, E)$ with real-valued edge weights c_e
- Feasible solution:
  - a spanning tree T of G, i.e. a tree $T=(V,F)$ with $T \subseteq E$ (reaching all vertices of G)
- measure (to minimize):
  - the weight (or cost) of T, i.e $c(T)= \Sigma_{e \in T} c_e$

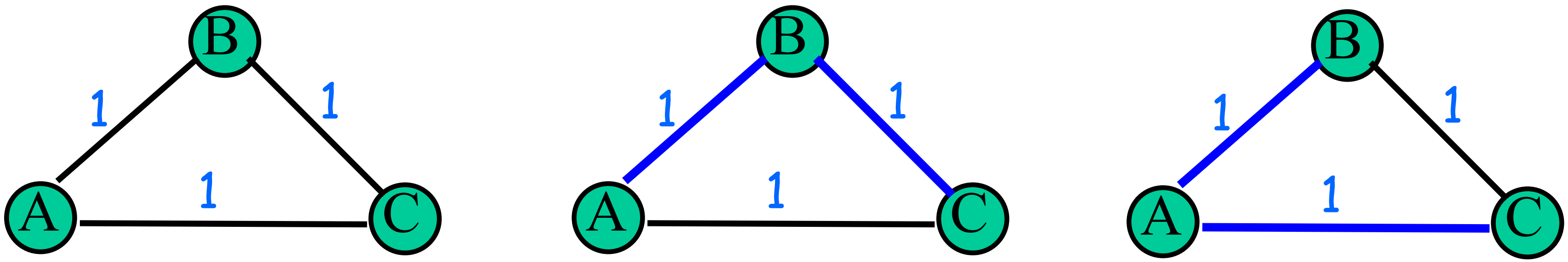
Applications

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.

Uniqueness of MST

The MST is not unique in general

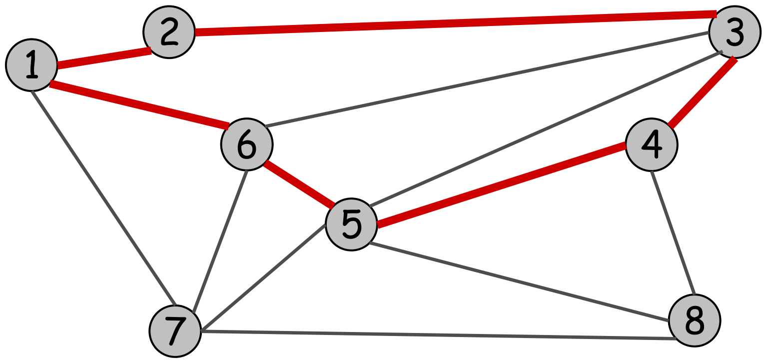


Property: If G has distinct weights then the MST is unique.

exercise: prove it.

Cycles and Cuts

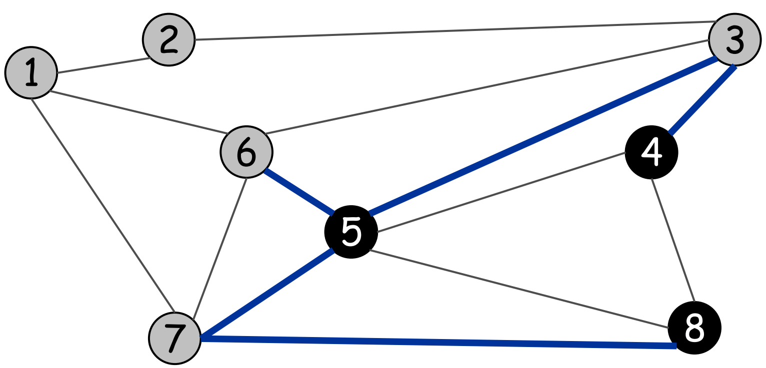
Cycle. Set of edges the form a-b, b-c, c-d, …, y-z, z-a.



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

A Cut. A cut is a subset of nodes S. (Sometime defined as a partition of V into S and V\S.)

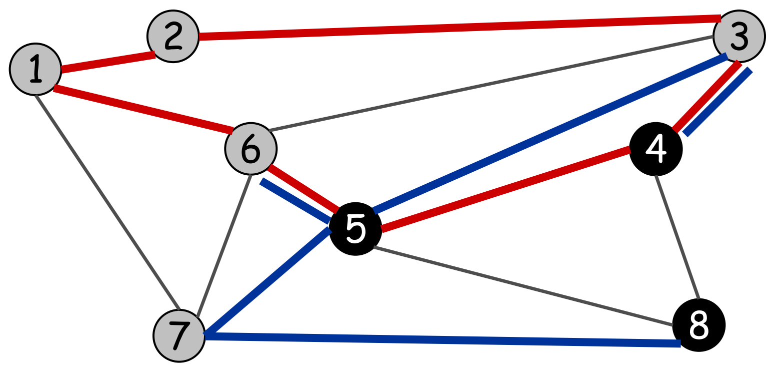
Cutset. The corresponding cutset D of a cut S is the subset of edges with exactly one endpoint in S.



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

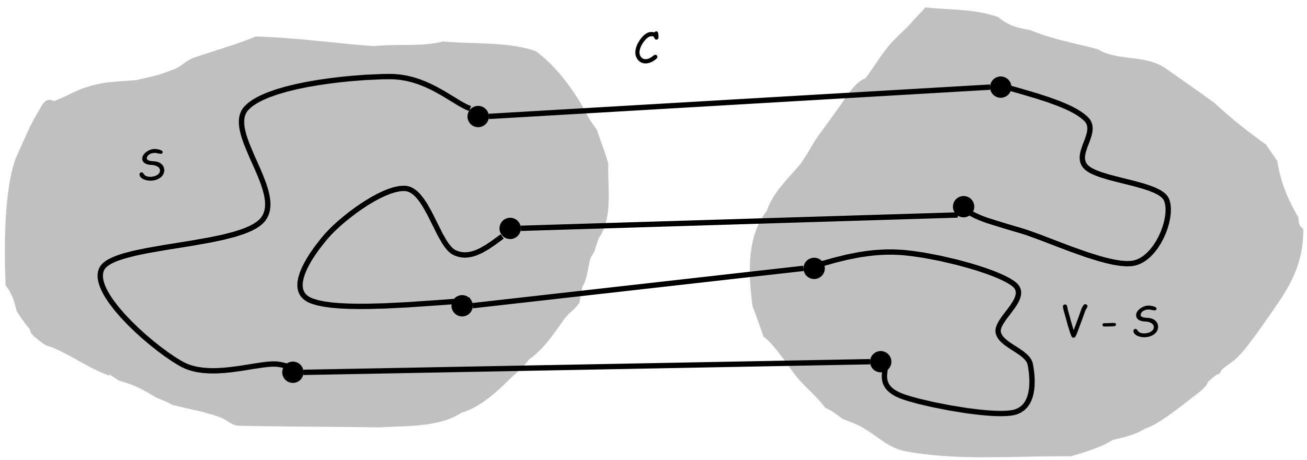
Cycle-Cut Intersection

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



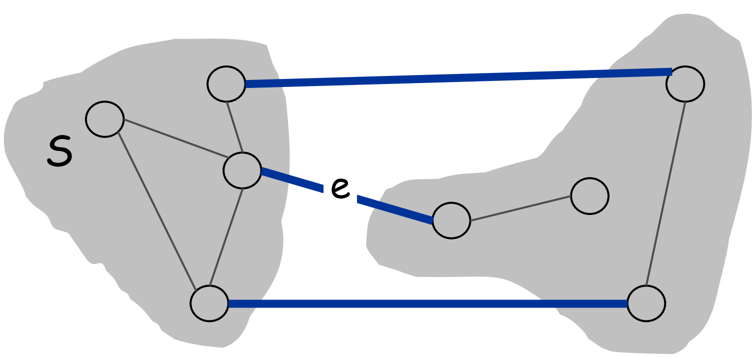
Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1
Cutset D = 3-4, 3-5, 5-6, 5-7, 7-8
Intersection = 3-4, 5-6

Pf. (by picture)

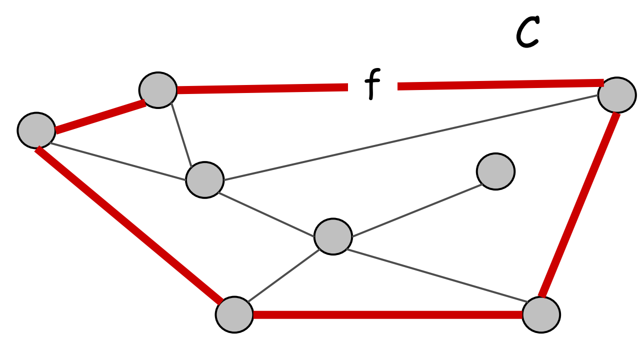


Cut property. Let S be any subset of nodes, and let e be a min cost edge with exactly one endpoint in S. Then there exists an MST containing e.

Cycle property. Let C be any cycle, and let f be a max cost edge belonging to C. Then there exists an MST that does not contain f.



e is in some MST

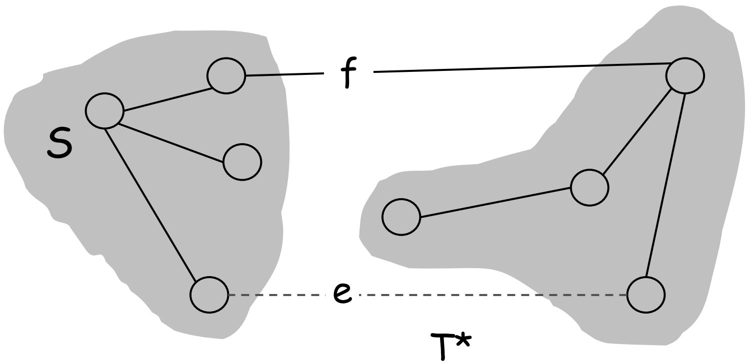


f is not in some MST

Cut property. Let S be any subset of nodes, and let e be a min cost edge with exactly one endpoint in S. Then there exists an MST T* containing e.

Pf. (exchange argument)

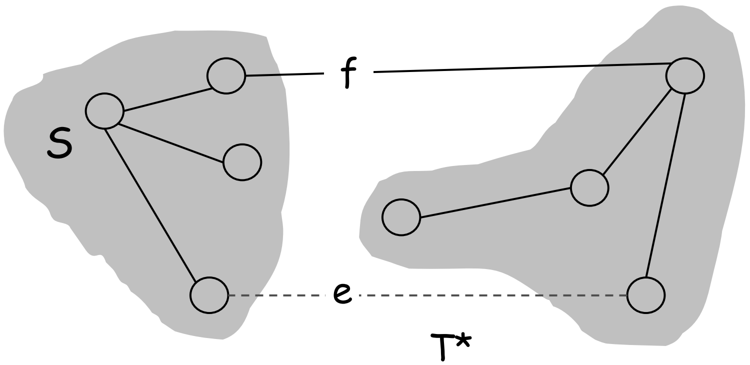
- Suppose e does not belong to T*.
- Adding e to T* creates a cycle C in T*.
- Edge e is both in the cycle C and in the cutset D corresponding to S $\Rightarrow$ there exists another edge, say f, that is in both C and D.
- $T' = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.
- Since $c_e \leq c_f$, $\text{cost}(T') \leq \text{cost}(T^*)$.
- Then T' is an MST containing e ▪



Cycle property. Let C be any cycle in G, and let f be a max cost edge belonging to C. Then there exists an MST T* that does not contain f.

Pf. (exchange argument)

- Suppose f belongs to T*.
- 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 $c_e \leq c_f$, cost(T') $\leq$ cost(T*).
- Then T' is an MST that does not contain f. ▪



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.

Kruskal's algorithm

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.

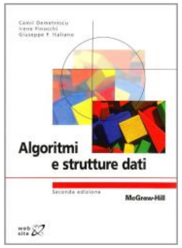
Remark.

An efficient implementation of Kruskal's algorithm uses a Union-Find data structure:

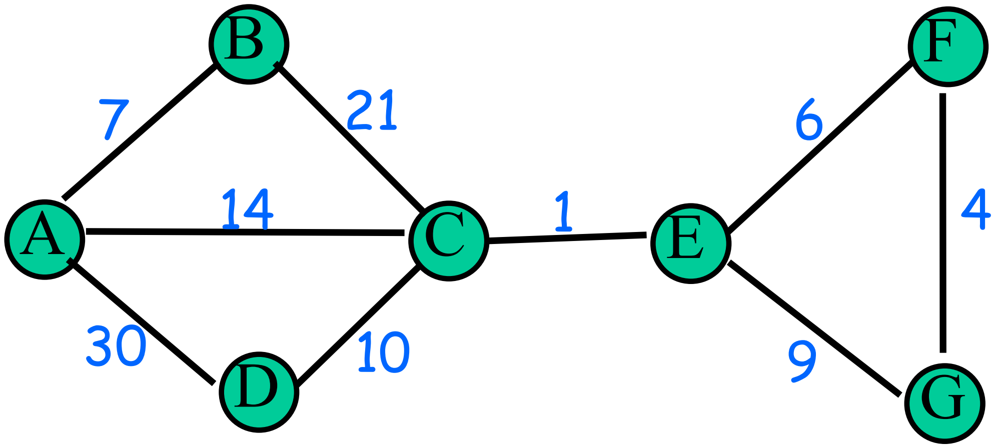
- to maintain the connected components of the current solutions
- to check whether the current edge forms a cycle (with the current solution)

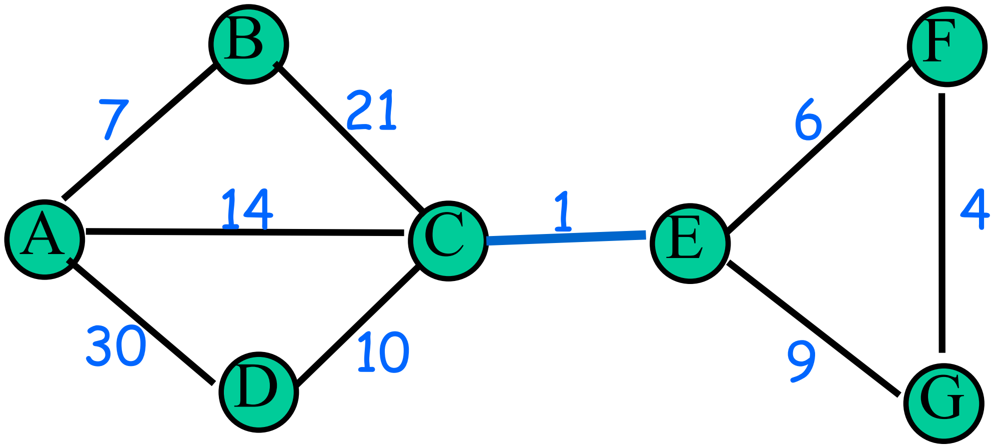
A pseudocode

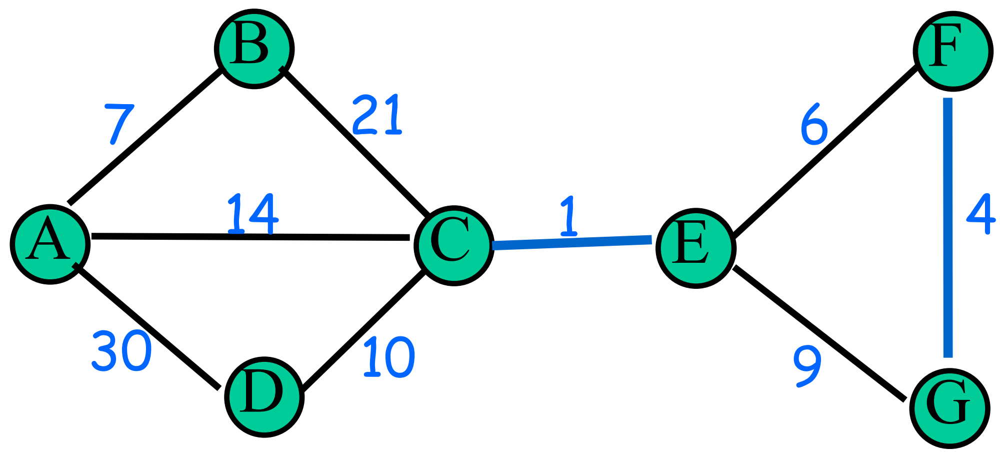
taken from

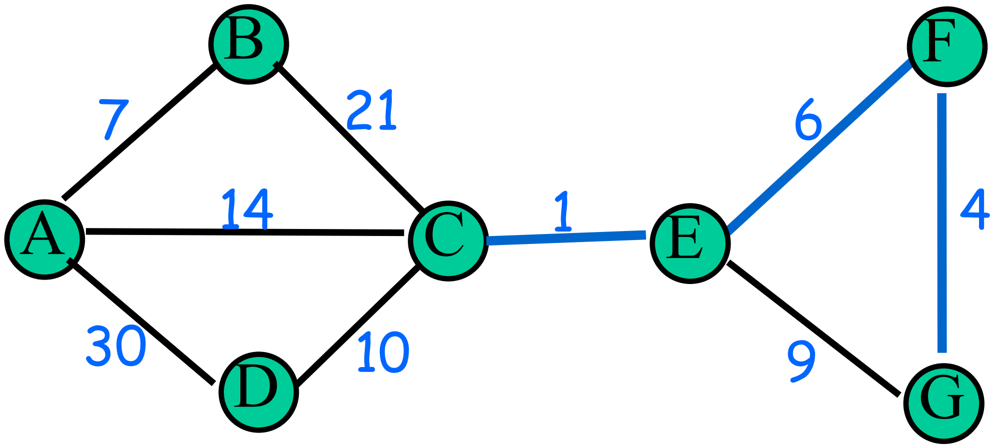


algorithm Kruskal (graph G=(V,E,c))
    UnionFind UF
    $T=\varnothing$
    sort the edges in ascending order of costs
    for each vertex v **do** UF.makeset(v)
    for each edge (x,y) in order **do**
        T_x=UF.find(x)
        T_y=UF.find(y)
        if $T_x \neq T_y$ **then**
            UF.union(T_x,T_y)
            add edge (x,y) to T
    return T

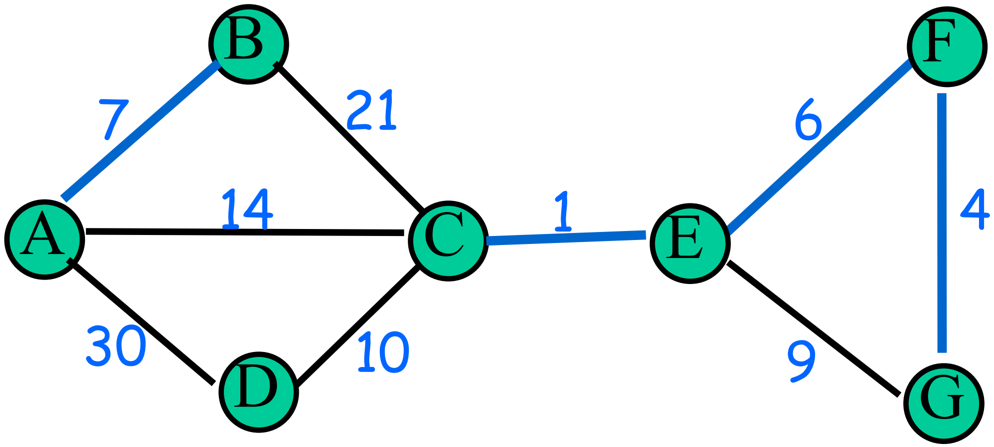

A
B
C
D
E
F
G
7
21
14
30
10
1
6
4
9

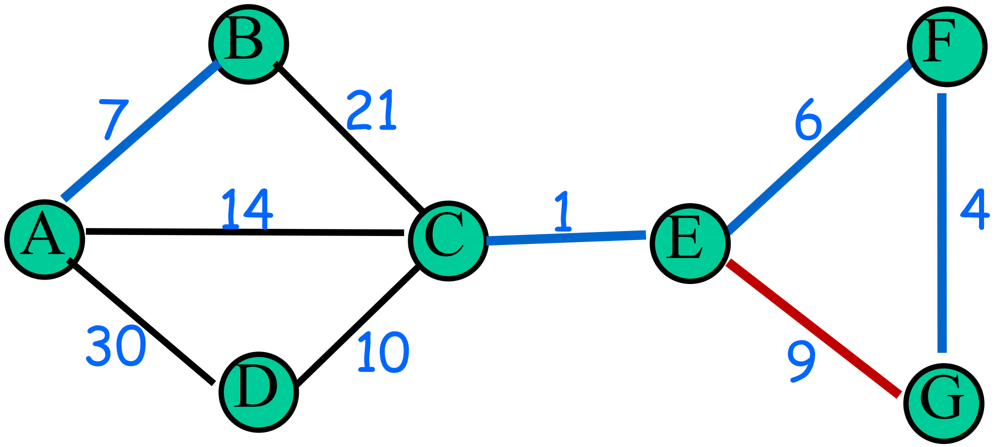

A
B
C
D
E
F
G
7
21
14
1
30
10
6
4
9


A
B
C
D
E
F
G
7
21
14
1
30
10
6
4
9

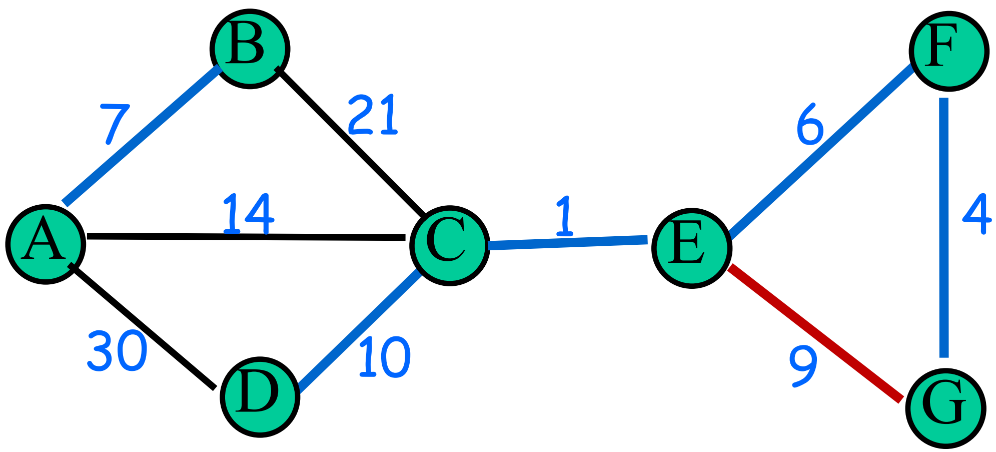


B
7
21
A
14
C
1
E
6
F
4
30
10
9
D
G

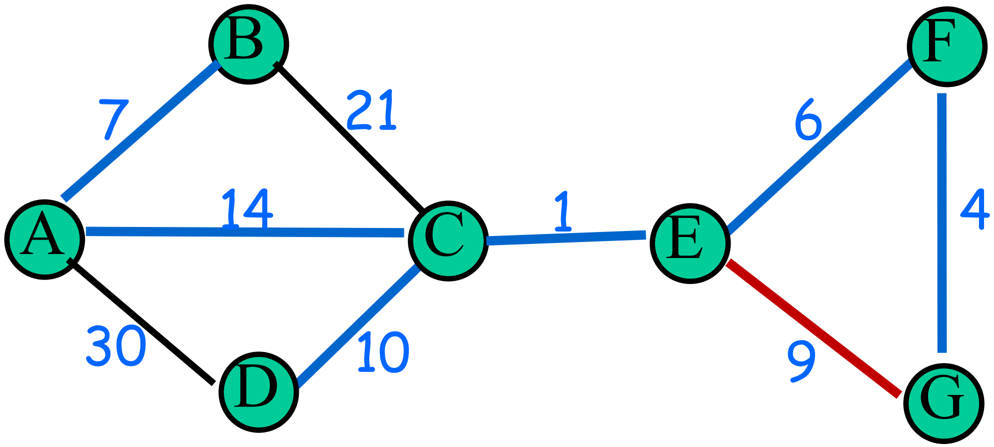

A
B
C
D
E
F
G
7
21
14
1
30
10
6
4
9

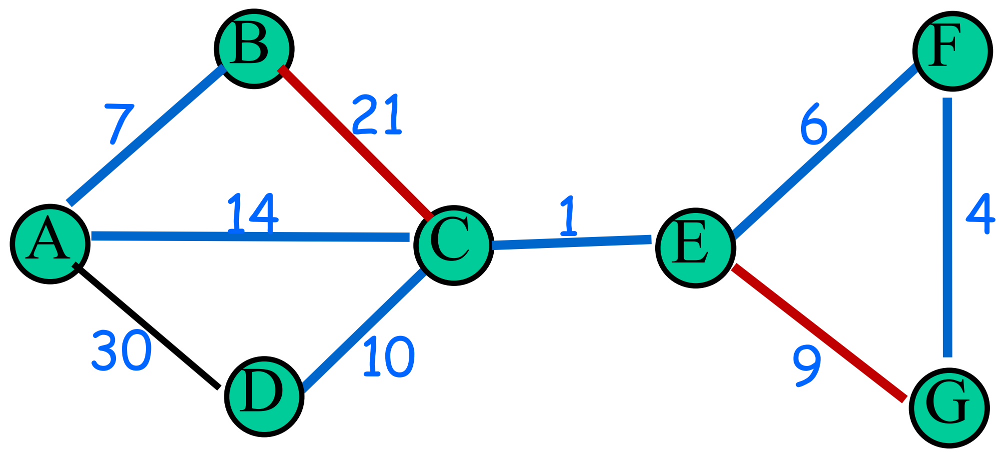


A
B
C
D
E
F
G
7
21
14
1
30
10
6
4
9

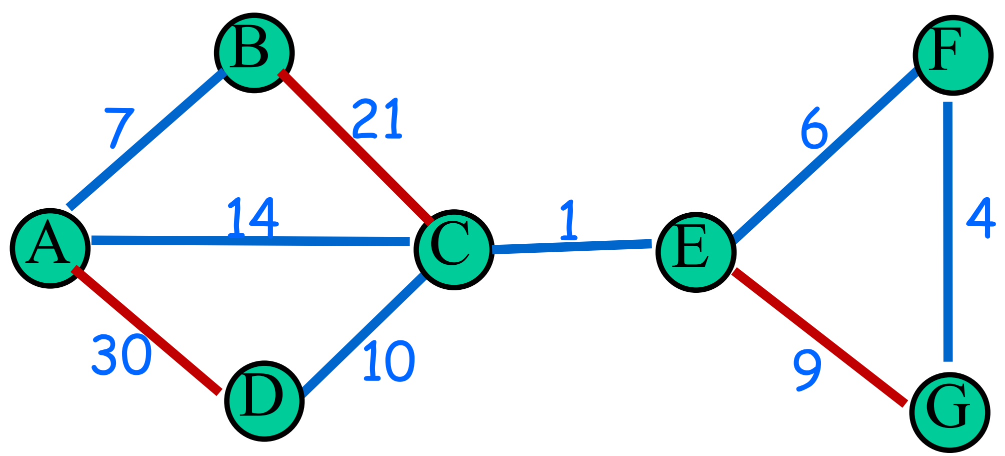


A
B
C
D
E
F
G
7
21
14
1
30
10
6
4
9

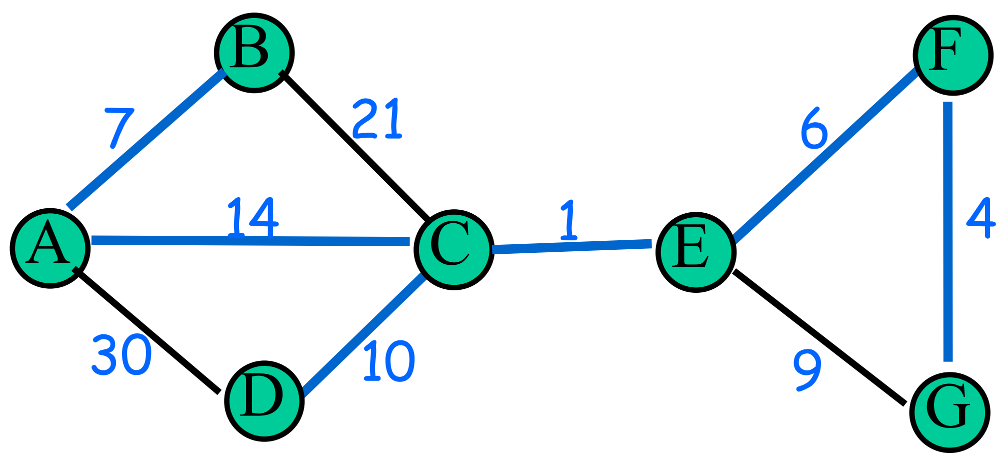

A
B
C
D
E
F
G
7
21
14
1
30
10
6
4
9



A
B
C
D
E
F
G
7
21
14
1
30
10
6
4
9



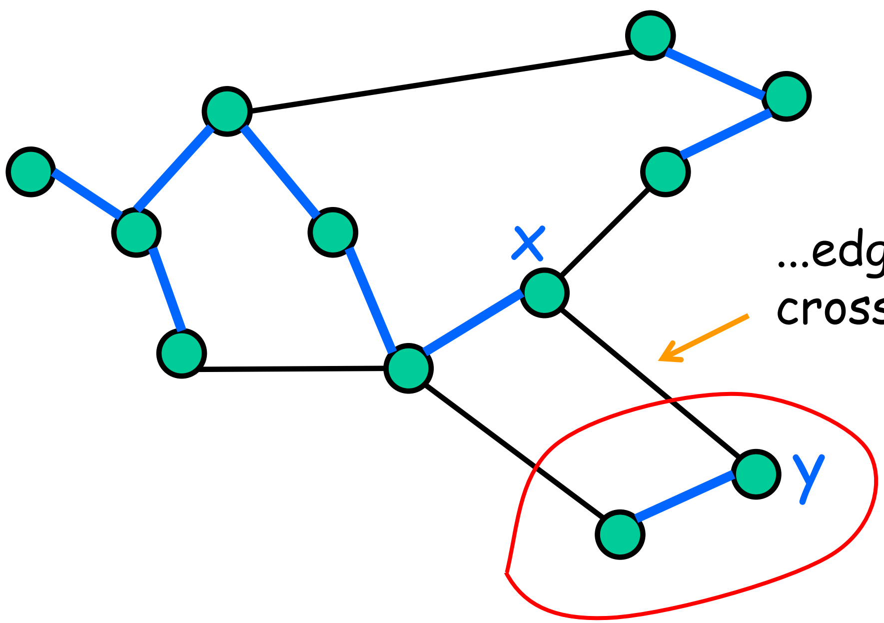
A
B
C
D
E
F
G
7
14
21
30
10
1
6
4
9



A
B
C
D
E
F
G
7
21
14
1
30
10
6
4
9

Correctness of Kruskal's algorithm

when the algorithm decides to add edge (x,y) to the solution



since the algorithm look at the edges in increasing order of costs...

...edge of minimum cost crossing the cut S,V\S

consider the set S of vertices belonging to the same connected component of y

Running time of Kruskal's algorithm

algorithm Kruskal (graph G=(V,E,c))
  UnionFind UF
  T=∅
  sort the edges in ascending order of costs
  for each vertex v **do** UF.makeset(v)
  for each edge (x,y) in order **do**
    T_x=UF.find(x)
    T_y=UF.find(y)
    if $T_x \neq T_y$ **then**
      UF.union(T_x,T_y)
      add edge (x,y) to T
  return T

- sorting the edges: O(m log m)=O(m log n)
- Union-Find operations:
  - n makeset ops
  - n-1 union ops
  - m find ops

⇒ O(m log n + UF(m,n))

-using QuickFind with *union by size*
 O(m log n + m + n log n)=O(m log n)

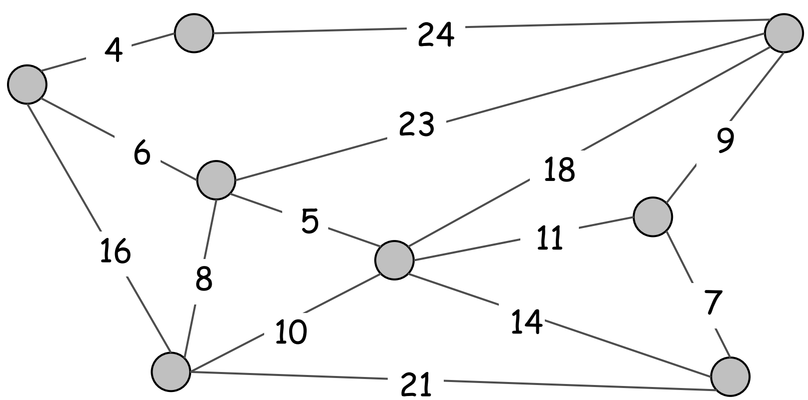
-using QuickUnion with *union by size*
 O(m log n + m log n + n)=O(m log n)

$\boxed{O(m \log n)}$

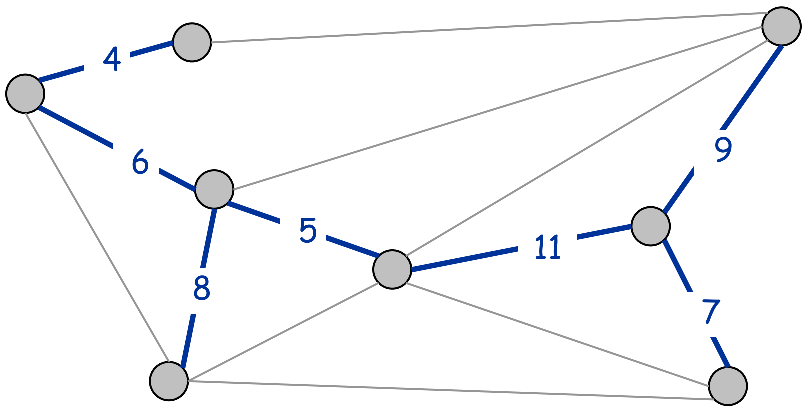
Prim's algorithm

The Minimum Spanning Tree (MST) problem

- Input:
  - a connected weighted undirected graph $G = (V, E)$ with real-valued edge weights c_e
- Feasible solution:
  - a spanning tree T of G, i.e. a tree $T=(V,F)$ with $T \subseteq E$ (reaching all vertices of G)
- measure (to minimize):
  - the weight (or cost) of T, i.e $c(T)= \Sigma_{e\in T}\, c_e$



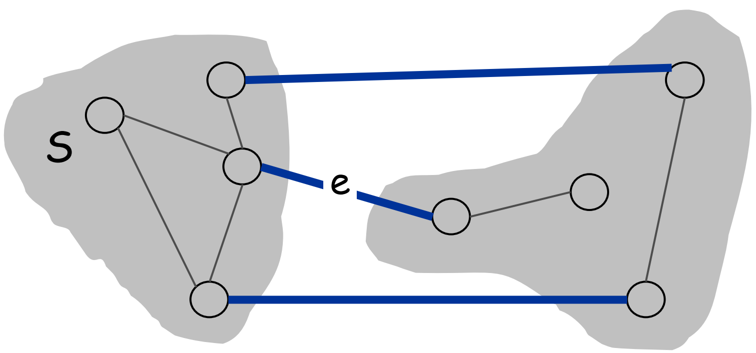
$G = (V, E)$



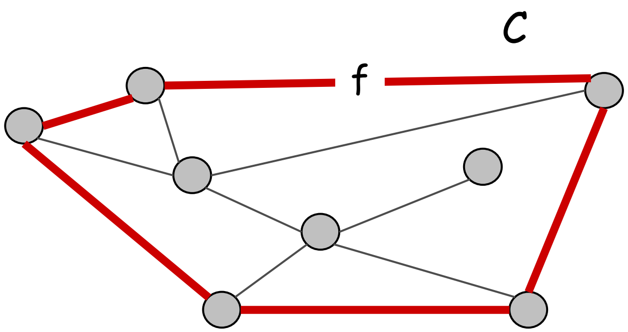
$T, \ \Sigma_{e\in T}\, c_e = 50$

Cut property. Let S be any subset of nodes, and let e be a min cost edge with exactly one endpoint in S. Then there exists an MST containing e.

Cycle property. Let C be any cycle, and let f be a max cost edge belonging to C. Then there exists an MST that does not contain f.



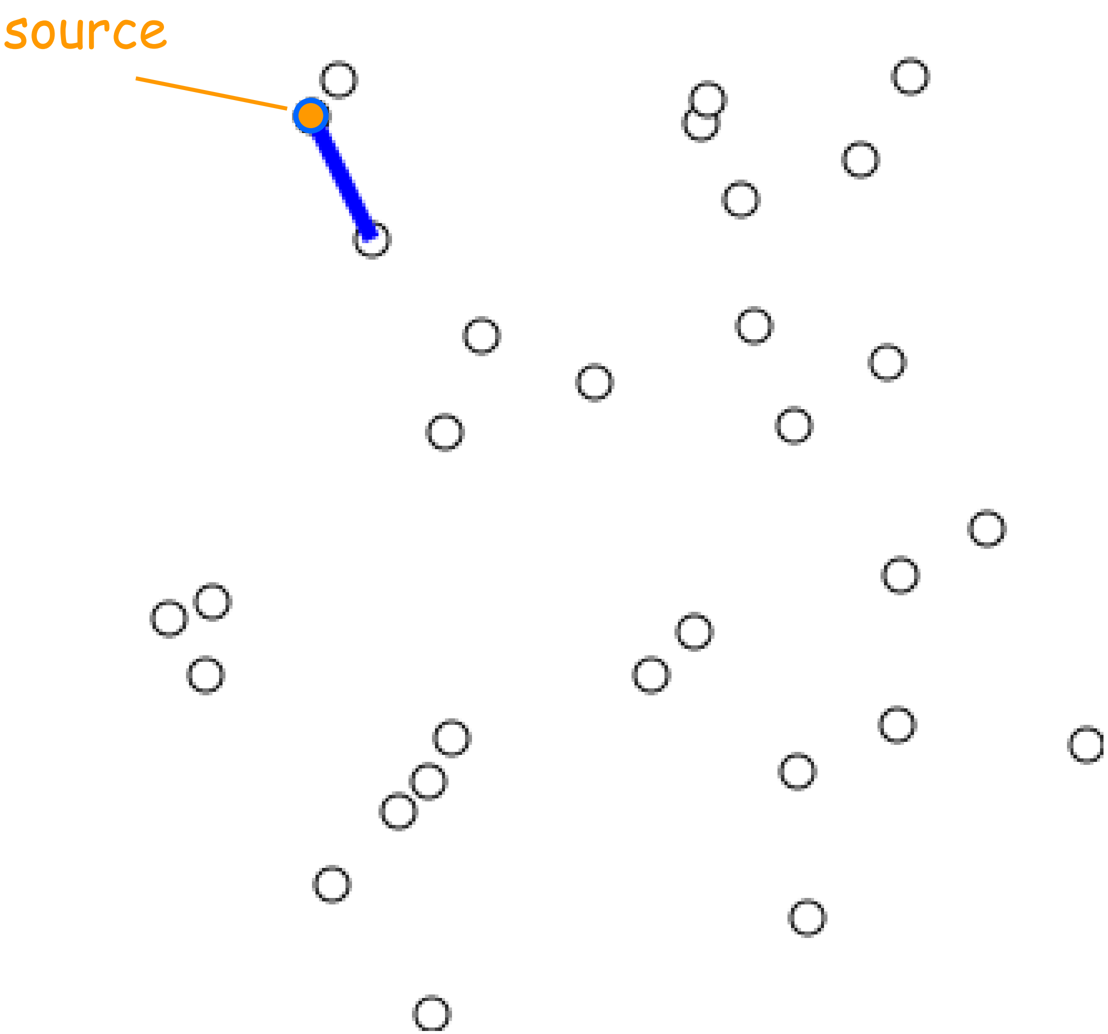
e is in some MST



f is not in some MST

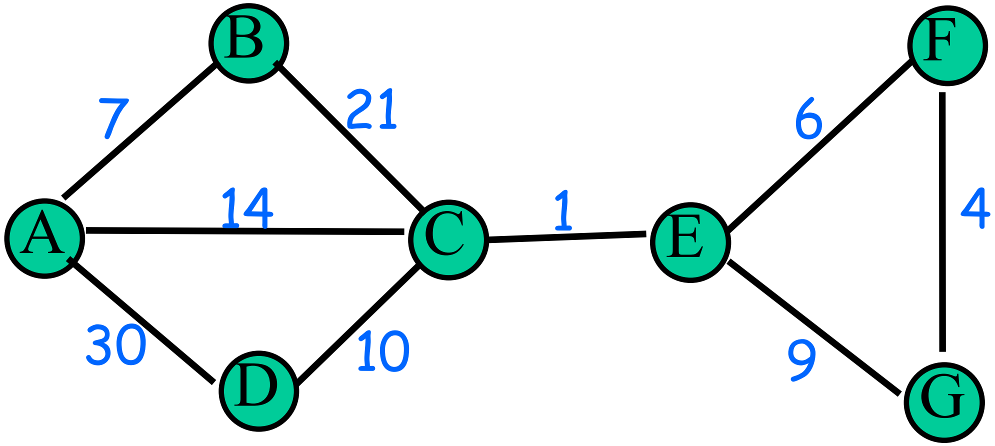
Prim's algorithm [Jarník 1930, Dijkstra 1957, Prim 1959].
Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T.

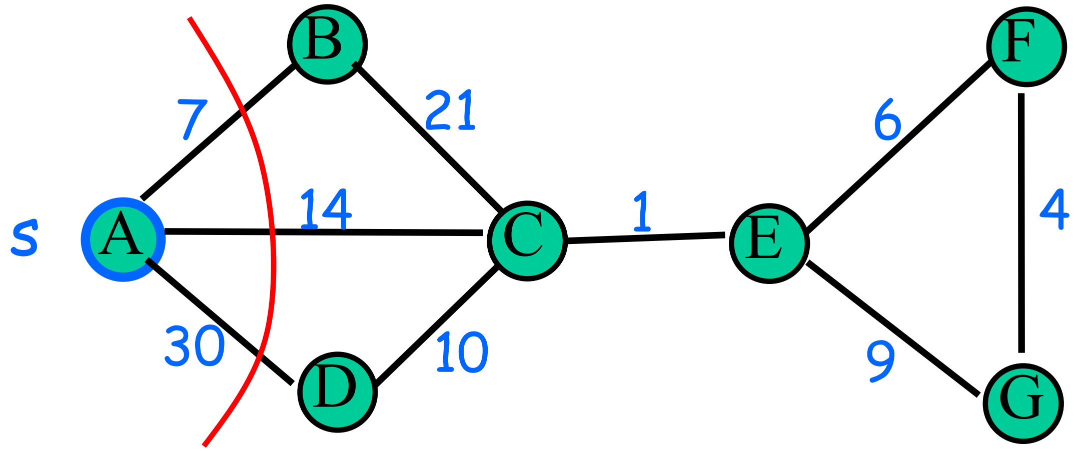
Correctness.
Immediate consequence of the cut property, used exactly n-1 times.



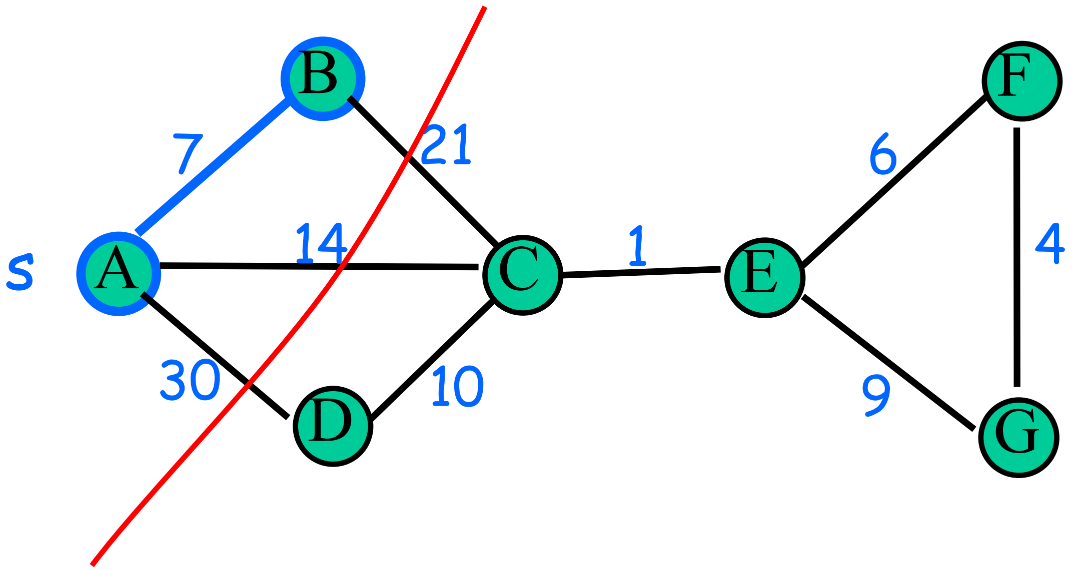
Euclidean complete graph

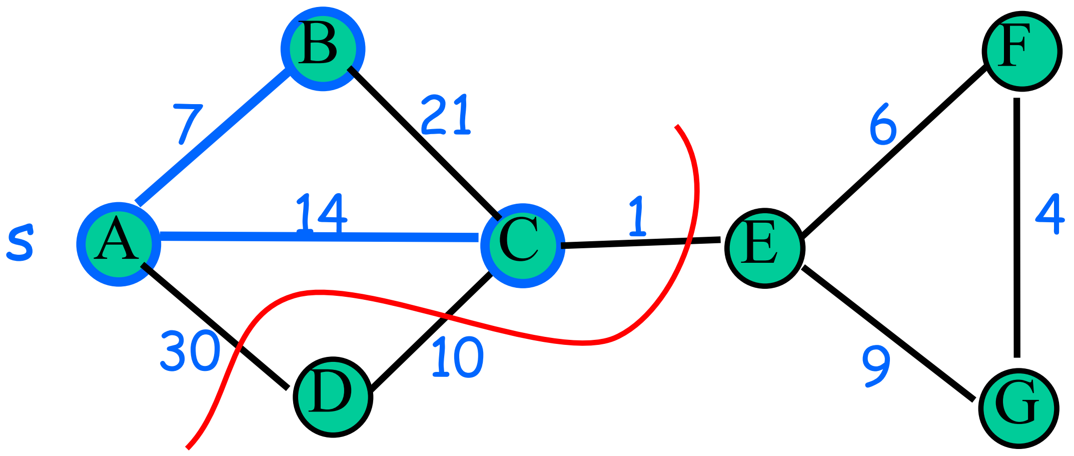
- vertices placed on the plane
- for each pair of vertices u and v the cost of edge (u,v) is the Euclidean distance between u and v.


A
B
C
D
E
F
G
7
21
14
1
30
10
6
4
9

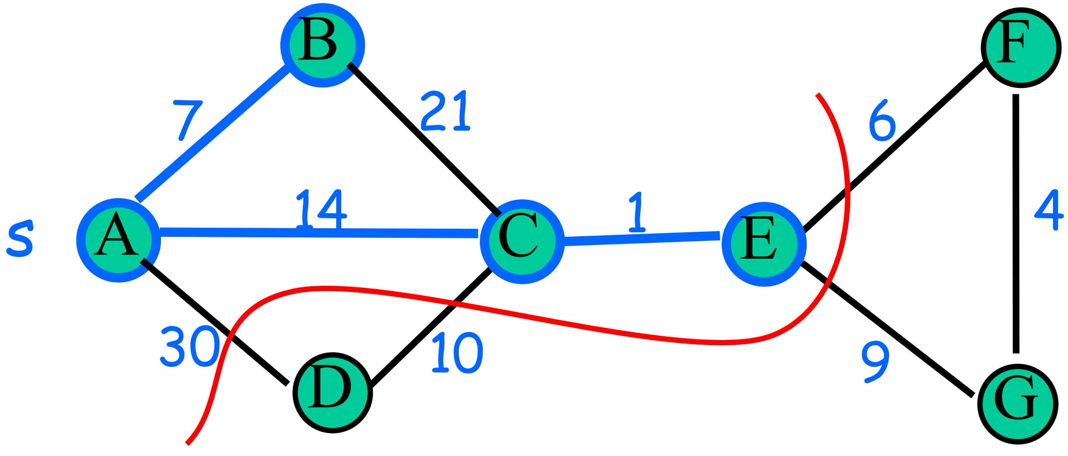


s
A
B
C
D
E
F
G
7
14
30
21
10
1
6
9
4

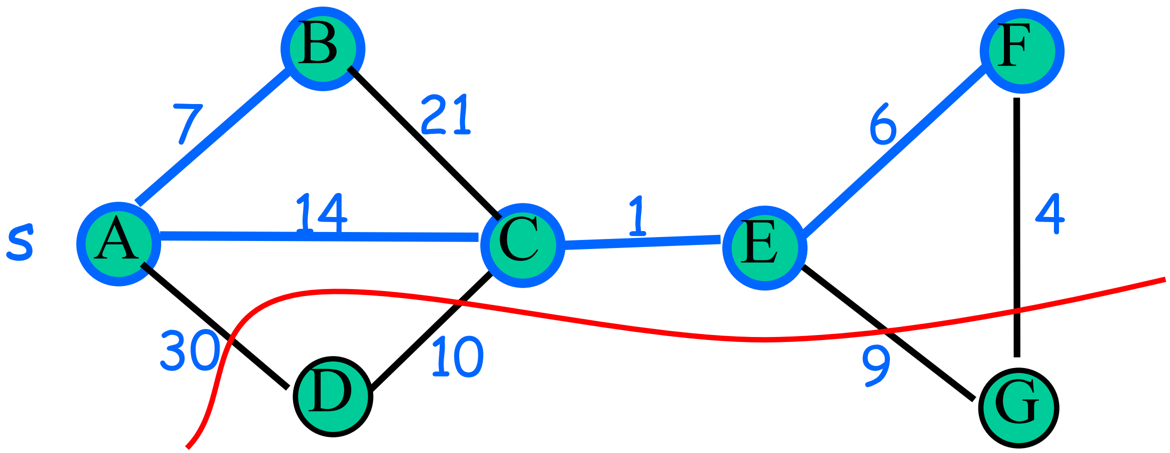

s
A
B
C
D
E
F
G
7
21
14
1
30
10
6
4
9

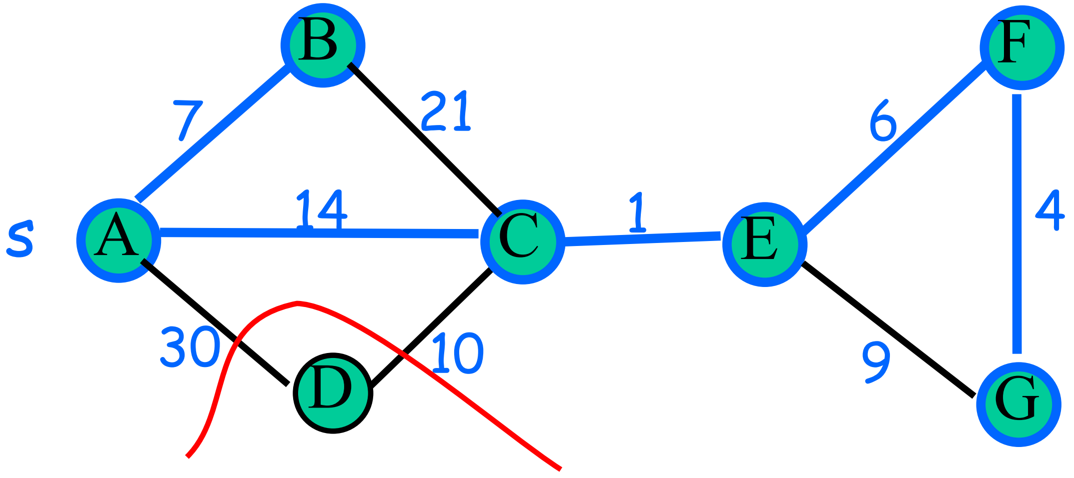


s
A
B
C
D
E
F
G
7
21
14
1
30
10
6
4
9

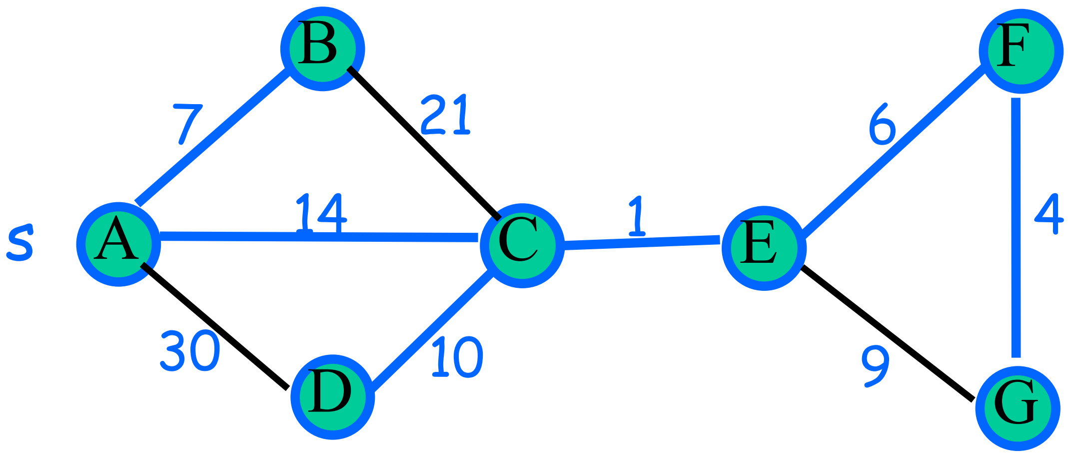


s
A
B
C
D
E
F
G
7
21
14
1
6
4
30
10
9


s
A
B
C
D
E
F
G
7
14
21
30
10
1
6
4
9



s
A
B
C
D
E
F
G
7
21
14
1
6
4
30
10
9


s
A
B
C
D
E
F
G
7
21
14
1
30
10
6
4
9

Running time

A simple (and inefficient) implementation:

For n-1 times, find a cheapest edge crossing the cut induced by the current partial tree in O(m) time.

Total running time: O(mn).

A much faster implementation:

- Maintain set of explored nodes S.
- Use a priority queue to maintain unexplored nodes.
- For each unexplored node v, the priority is the attachment cost a[v] = cost of a cheapest edge incident in v having the other endpoint in S.

```
Prim(G, s) {
   foreach (v ∈ V) a[v] ← ∞
   a[s] ← 0
   Initialize an empty priority queue Q
   foreach (v ∈ V) insert v onto Q with priority a[v]
   Initialize set of explored nodes S ← ϕ
   Initialize T to the tree containing only s.

   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]))
             make u parent of v in T
             decrease priority a[v] to c_e
   return T
}
```

Running time.

- $O(m+n)$ time plus the cost of the priority queue operations
- n inserts, n delete min ops, m decrease key ops
- $O(n^2)$ with an array; $O(m \log n)$ with a binary heap; $O(m + n \log n)$ with Fibonacci's heaps

```
Prim(G, s) {
   foreach (v ∈ V) a[v] ← ∞
   a[s] ← 0
   Initialize an empty priority queue Q
   foreach (v ∈ V) insert v onto Q with priority a[v]
   Initialize set of explored nodes S ← φ
   Initialize T to the tree containing only s.

   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]))
             make u parent of v in T
             decrease priority a[v] to c_e
   return T
}
```

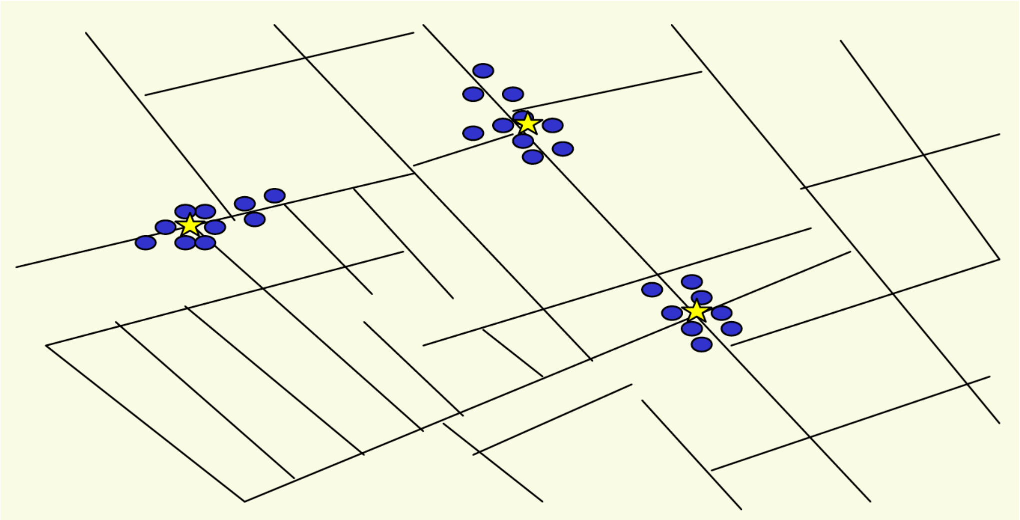
Running time.

- $O(m+n)$ time plus the co rations
- n inserts, n delete min
- $O(n^2)$ with an array; $O($

  $O(m + n \log n)$ with Fibonacci heaps

$O(m + n \log n)$

4.7 Clustering



Outbreak of cholera deaths in London in 1850s.
Reference: Nina Mishra, HP Labs

Clustering

Clustering. Given a set U of n objects labeled $p_1, \ldots, p_n$, classify into coherent groups.

photos, documents. micro-organisms

Distance function. Numeric value specifying "closeness" of two objects.

number of corresponding pixels whose intensities differ by some threshold

Fundamental problem. Divide into clusters so that points in different clusters are far apart.

- Routing in mobile ad hoc networks.
- Identify patterns in gene expression.
- Document categorization for web search.
- Similarity searching in medical image databases
- Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.

Clustering of Maximum Spacing

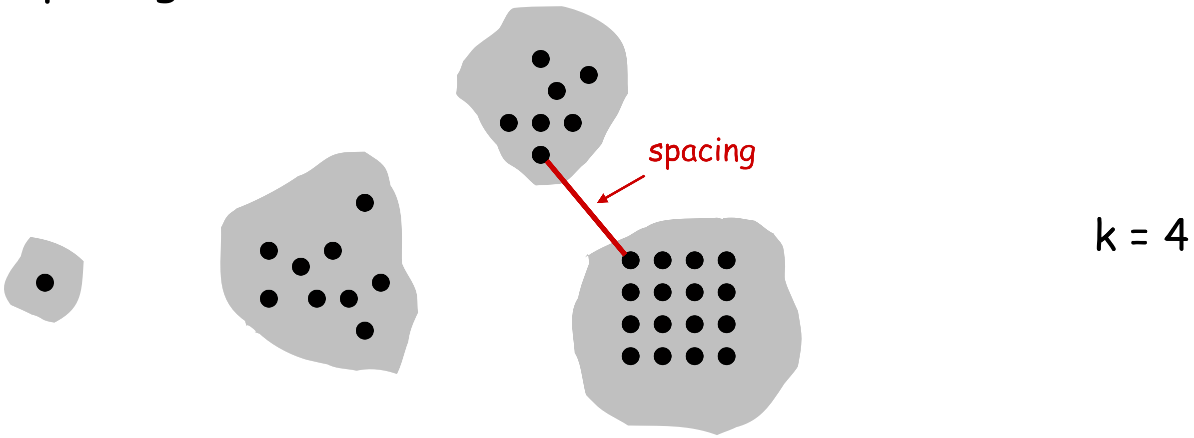
k-clustering. Divide objects into k non-empty groups.

Distance function. Assume it satisfies several natural properties.

- $d(p_i, p_j) = 0$ iff $p_i = p_j$ (identity of indiscernibles)
- $d(p_i, p_j) \geq 0$ (nonnegativity)
- $d(p_i, p_j) = d(p_j, p_i)$ (symmetry)

Spacing. Min distance between any pair of points in different clusters.

Clustering of maximum spacing. Given an integer k, find a k-clustering of maximum spacing.



Greedy Clustering Algorithm

Single-link k-clustering algorithm.

- Form a graph on the vertex set U, corresponding to n clusters.
- Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.
- Repeat n-k times until there are exactly k clusters.

Key observation. This procedure is precisely Kruskal's algorithm (except we stop when there are k connected components).

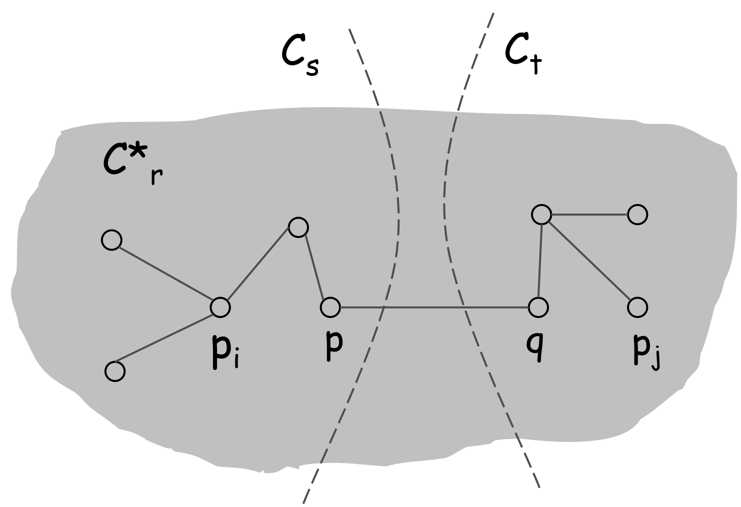
Remark. Equivalent to finding an MST and deleting the k-1 most expensive edges.

Greedy Clustering Algorithm: Analysis

Theorem. Let C^* denote the clustering $C^*_1, \ldots, C^*_k$ formed by deleting the k-1 most expensive edges of a MST. C^* is a k-clustering of max spacing.

Pf. Let C denote some other clustering $C_1, \ldots, C_k$.

- The spacing of C^* is the length d^* of the $(k-1)^{st}$ most expensive edge of the MST.
- Let p_i, p_j be in the same cluster in C^*, say C^*_r, but different clusters in C, say C_s and C_t.
- Some edge (p, q) on p_i-p_j path in C^*_r spans two different clusters in C.
- All edges on p_i-p_j path have length $\leq d^*$ since Kruskal chose them.
- Spacing of C is $\leq d^*$ since p and q are in different clusters. ▪



Extra Slides

MST Algorithms: Theory

Deterministic comparison based algorithms.

- $O(m \log n)$ [Jarník, Prim, Dijkstra, Kruskal, Boruvka]
- $O(m \log \log n)$. [Cheriton-Tarjan 1976, Yao 1975]
- $O(m\, \beta(m, n))$. [Fredman-Tarjan 1987]
- $O(m \log \beta(m, n))$. [Gabow-Galil-Spencer-Tarjan 1986]
- $O(m\, \alpha(m, n))$. [Chazelle 2000]
- $O(T^*(m,n)) = O(m\, \alpha(m, n))$. [Pettie-Ramachandran 2000]

Holy grail. $O(m)$.

Notable.

- $O(m)$ randomized. [Karger-Klein-Tarjan 1995]
- $O(m)$ verification. [Dixon-Rauch-Tarjan 1992]