Lecture: Analysis of Algorithms (CS583 - 004)[1]

Amarda Shehu

Spring 2019

[1] Some material adapted from Kevin Wayne's Algorithm Class @ Princeton

What is the Spanning Tree of a Graph?

If $G = (V, E)$ is a graph, then any subgraph of G that (i) contains all vertices V of G and (ii) is a tree is a **spanning tree** of G.

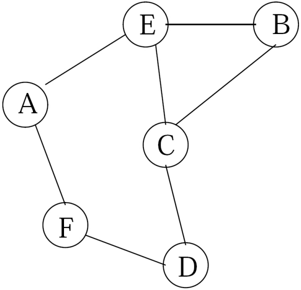


Figure: Graph $G = (V, E)$

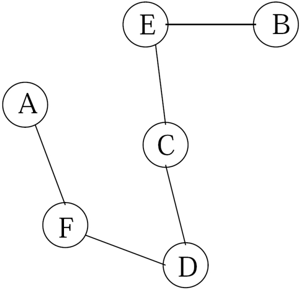


Figure: Spanning tree $T = (V, E^{'})$ of graph G

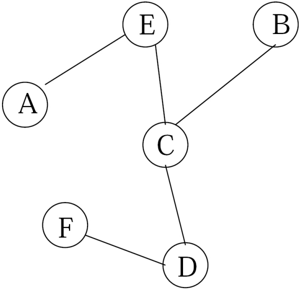


Figure: Another spanning tree of graph G

Some Spanning Trees are Better than Others

- A weighted (connected) undirected graph $G = (V, E)$
- Weight function $w : E \rightarrow R$ associates a weight with an edge
- The weight $w(T)$ of a tree T is $\sum_{(u,v) \in T} w(u, v)$
- A minimum spanning tree (MST) has the minimum $w(T)$ over all spanning trees T of a graph G

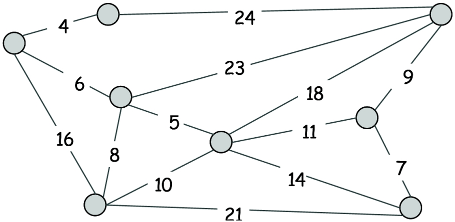


Figure: Weighted graph

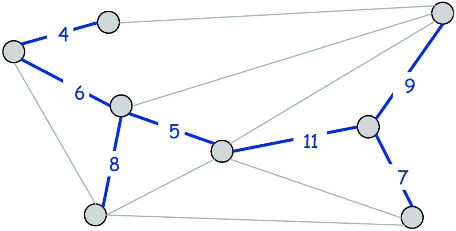


Figure: MST: $w(T) = 50$

Finding MSTs is Useful in Diverse Applications

- Network design
  - Phone, electric, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-hard problems
  - Traveling Salesman Problem, Steiner trees
- Other (indirect) applications
  - Maximum bottleneck paths
  - LDPC codes for error correction
  - Image registration with Renyi entropy
  - Learn features for real-time face verification
  - Reduce 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

Finding MSTs: Problem Statement

Problem: Given a weighted (connected) undirected graph $G = (V, E)$, find an MST of G.

Input: A connected, undirected graph $G = (V, E)$ with weight function $w : E \rightarrow R$

Output: A spanning tree T of G that is of minimum weight $w(T) = \sum_{(u,v) \in T} w(u, v)$

Algorithms to Find MSTs

There's the history:

- Boruvka [Otakar Boruvka 1926]
  - Wanted to minimize the cost of electric coverage of Moravia
- Jarnik [V. Jarnik 1930]
- Kruskal [Joseph B. Kruskal 1956]
- Prim [Run C. Prim 1957]
- Chazelle [Bernard Chazelle 2000]

And then there's us:

- Brute-force approach
- Something smarter?

Finding MSTs: Brute-force Approach

Enumerating spanning trees of a graph

- Denote the number of spanning trees of a graph G by $t(G)$
- $t(G)$ is easy to compute for special graphs
- Caylee's formula gives $t(G)$ for a complete graph on n vertices: $t(G) = n^{n-2}$ for $n > 1$
- Example: in a complete graph on 4 vertices, $t(G) = 16$
- For any graph G, $t(G)$ can be computed with Kirchhoff's matrix-free theorem: $t(G) = \frac{1}{n}\lambda_1 \cdot \ldots \cdot \lambda_{n-1}$, where λ_i are the non-zero eigenvalues of the Laplacian matrix of G
- Bottom line: Too many spanning trees to enumerate to find MST through a brute-force approach

Algorithms to Find an MST

Brute-force Approach: terribly inefficient

Greedy Approach:

- Find a key property of the MST to help determine whether an edge of G is part of the MST
- Then build up the MST one step (edge/vertex) at a time

Greedy Algorithms to Find the MST of a Graph

- Kruskal's Algorithm

  Heuristic: Select best edge for insertion

  Approach: (i) Start with $T = \emptyset$. (ii) Consider edges in ascending order of weight/cost. (iii) Insert edge e in T unless doing so creates a cycle.

- Reverse-Delete Algorithm

  Heuristic: Select worst edge for deletion

  - Approach: (i) Start with $T = E$. (ii) Consider edges in descending order of weight/cost. (iii) Delete edge e from T unless doing so disconnects T

- Prim's Algorithm

  Heuristic: Select best vertex

  Approach: (i) Start with some vertex s as root node. (ii) Greedily grow T from s outward. (iii) At each step, add cheapest edge e to T that has exactly one endpoint in T.

A Generic Algorithmic Template for Finding MSTs

Generic-MST(G, w)

1: $T \leftarrow \{\}$
2: **while** T does not form a spanning tree **do**
3:     find an edge in E that is safe for T
4:     $T \leftarrow T \cup \{u, v\}$
5: **return** T

Taking care of some implementation and correctness details:

- line 2: when do we know T forms a spanning tree?
- line 3: what does it mean to add a safe edge to T?
- lines 3-4: safeness has to address both low cost and no cycles

Cycles and Cuts

Cycle: Set of edges $\{(v_1, v_2), \ldots, (v_k, v_1)\}$

Cut: A subset S of vertices V
Cutset: Subset D of edges with exactly one endpoint in S

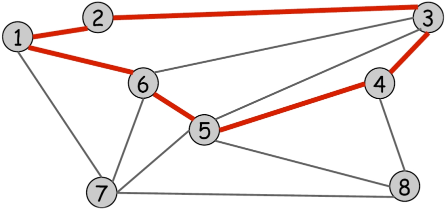


Figure: Cycle $C = \{(1,2), \ldots, (6,1)\}$

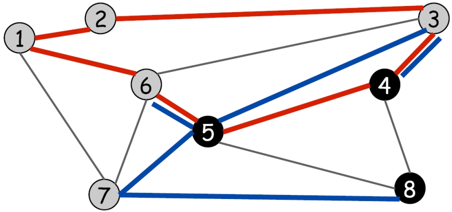


Figure: Cut $S = (4,5,8)$. Cutset $D = \{(5,6), \ldots, (7,8)\}$

Greedy Algorithms for MSTs Exploit Certain Properties

- Simplifying assumption: All edge costs/weights are distinct
- Cut property: Let S be any subset of vertices V in the graph $G = (V, E)$. Let $e \in E$ be the minimum weight edge with exactly one endpoint in S. Then, the MST of G contains e.
- Cycle property: Let C be any cycle, and let f be the maximum weight edge in C. Then, the MST does not contain f.

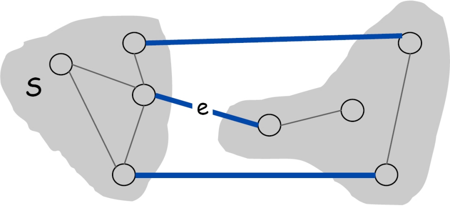


Figure: e is in the MST

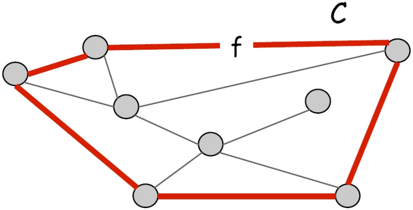
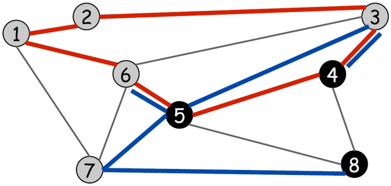


Figure: f is not in the MST

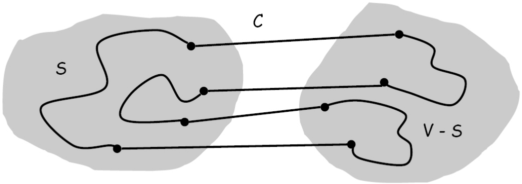
Cycle-Cut Intersection

Lemma: A cycle and a cutset intersect in an even number of edges



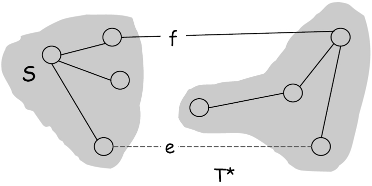
Cycle $C = \{(1,2),(2,3),\ldots,(6,1)\}$
Cutset $D = \{(3,4),(3,5),\ldots,(7,8)\}$
Intersection $I = \{(3,4),(5,6)\}$

Proof: Argument built from picture below



Cut Property: Proof

Cut Property Lemma: Let S be any subset of vertices V of $G = (V, E)$. Let $e \in E$ be the minimum weight edge with exactly one endpoint in S. Then, the MST T^* of G contains e.

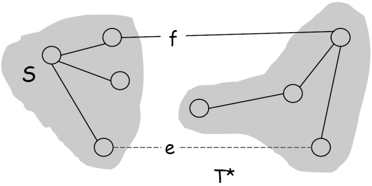


Proof: (cut-and-paste argument)

- Suppose $e \notin E(T^*)$. We are given that $e = (u, v)$, where $u \in S$ and $v \in V - S$. So, $e \in D$, the cutset corresponding to S.
- As a spanning tree, T^* contains a unique path from u to v without e in it
- Adding e to T^* would create a cycle C in T^*. So, $e \in C \cap D$.
- Since $C \cap D$ contains an even number of edges, $\exists f \in C \cap D$.
- Create $T^{'} = T^* \cup \{e\} - \{f\}$. Since $w(e) < w(f) \Rightarrow w(T^{'}) < w(T^*)$
- $T^{'}$ is more optimal than $T^* \Rightarrow$ proof achieved by contradiction.

Cycle Property: Proof

Cycle Property Lemma: Let C be any cycle in $G = (V, E)$. Let f be the maximum weight edge in C. Then, the MST T^* of G does not contain f.



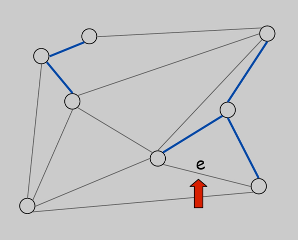
Proof: (cut-and-paste argument)

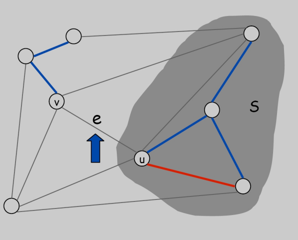
- Suppose $f \in E(T^*)$. Deleting f from T^* creates a cut S in T^*. So $f \in D$, the cutset corresponding to S.
- Edge $f \in C$ as well, so $f \in C \cap D$
- Since $C \cap D$ contains an even number of edges, $\exists e \in C \cap D$.
- Create $T^{'} = T^* \cup \{e\} - \{f\}$. Since $w(e) < w(f) \Rightarrow w(T^{'}) < w(T^*)$
- $T^{'}$ is more optimal than $T^* \Rightarrow$ proof achieved by contradiction.

Kruskal's Algorithm: One Edge at a Time

Kruskal's Algorithm [Kruskal, 1956]

- Start with $E(T) \leftarrow \emptyset$
- Consider edges in $E(G)$ in ascending order of weight
- Case 1: If adding e to $E(T)$ creates a cycle, discard e (cycle property)
- Case 2: Else, insert $e = (u, v)$ in $E(T)$, where S is the set of vertices in $u's$ connected component (cut property)





Kruskal's Algorithm: Implementation and Analysis

Kruskal-MST(G = (V, E), w)

```
1: sort the edges of G in ascending order of weights
2: V(T) ← V(G), E(T) ← ∅
3: for each edge e = (u, v) ∈ E in sorted order do
4:    if u and v are in different connected components then
5:       E(T) ← E(T) ∪ {e}
6: return T
```

Analysis:

- Sorting $\Rightarrow O(|E| \cdot lg(|E|))$ time in the worst-case
- For loop iterates over all $|E|$ edges in sorted order
- Potentially, line 4 could be slow. How can one find quickly whether the endpoints of e are disconnected in S?
- Line 4 can be performed in $O(1)$ time through the union-find operation on a disjoint-set data structure
- Short detour...

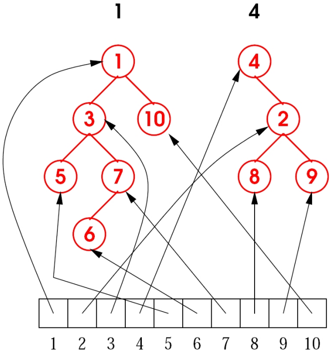
Disjoint-set Data Structure

- Maintains a collection of disjoint dynamic sets $\{S_1, \ldots, S_k\}$
- Each S_i can be represented as a linked list or tree
- The unique "key" of a set can be stored at root

Operations:

- Make-Set(x): create $\{x\}$
- Find-Set(x): find set that contains x
- Union(x,y): merge sets that contain x and y

A sequence of $O(m)$ Union and Find-Set operations on m elements can be performed in $O(m \cdot lgm)$ time.



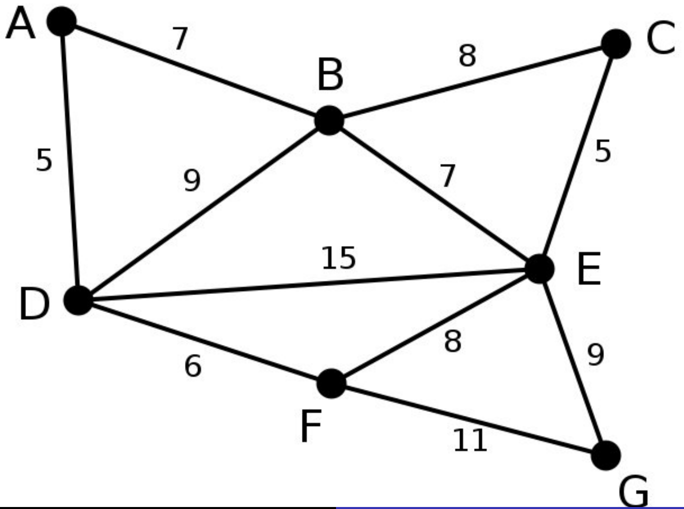
Back to Kruskal's Algorithm: Implementation and Analysis

Kruskal-MST(G, w)

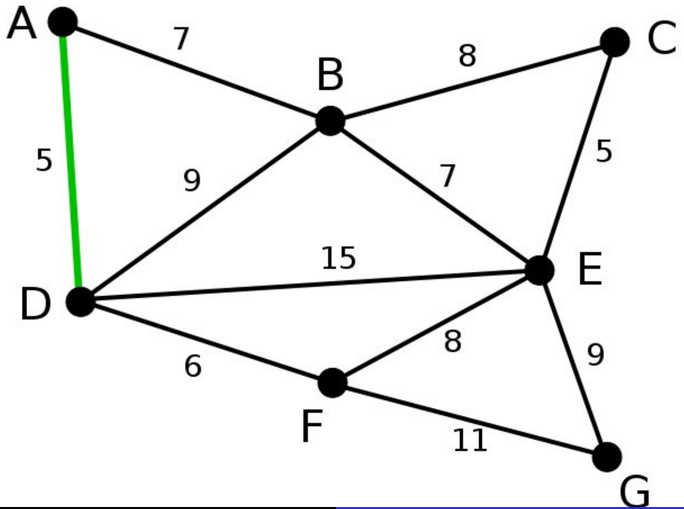
1: $S \leftarrow \{\}$
2: **for** each vertex $v \in V(G)$ **do**
3:     Make-Set(v)
4: sort the edges of G in ascending order of weights
5: **for** each edge $e = (u, v)$ in sorted order **do**
6:     **if** Find-Set(u) $\neq$ Find-Set(v) **then**
7:         $S \leftarrow S \cup \{(u, v)\}$
8:         Union(u,v)
9: **return** S

Analysis: Lines 5-8 contain $O(E)$ Find-Set and Union operations. Along with $|V|$ Make-Set, these take $O((V + E) \cdot \alpha(V))$, where α is a slowly growing function. Total running time is $O(E \cdot lg(E))$, since $E \geq |V| - 1$ in a connected graph (equiv. $O(E \cdot lg(V))$).

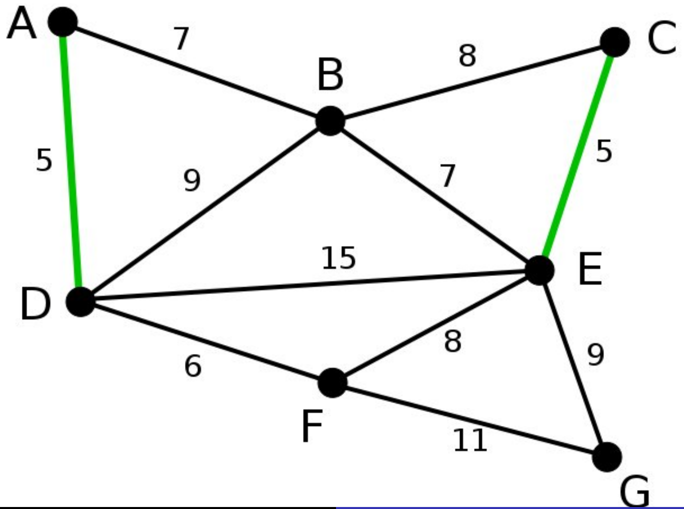
Kruskal's Algorithm in Action



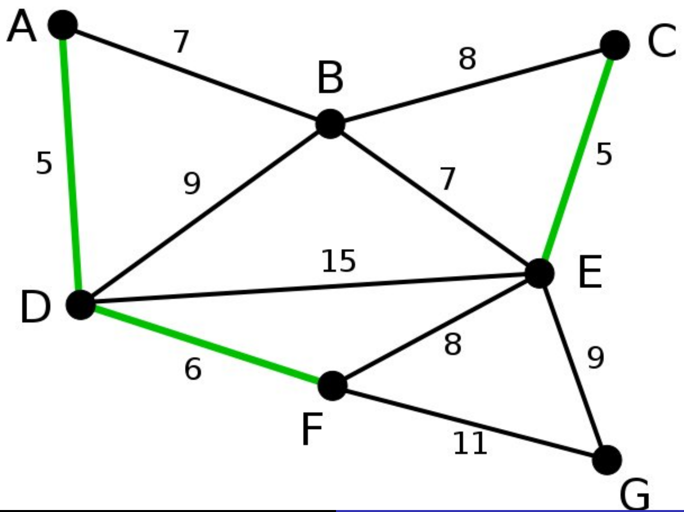
Kruskal's Algorithm in Action



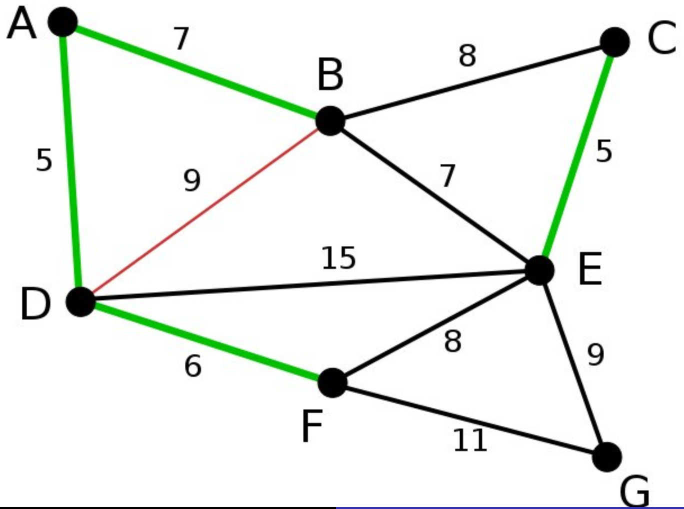
Kruskal's Algorithm in Action



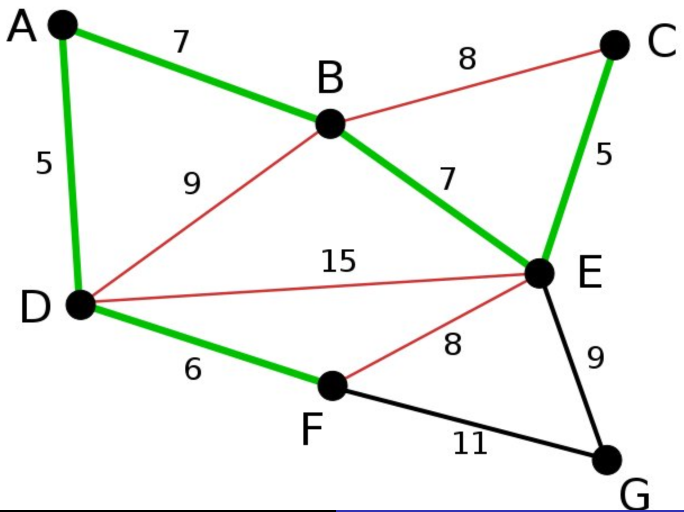
Kruskal's Algorithm in Action



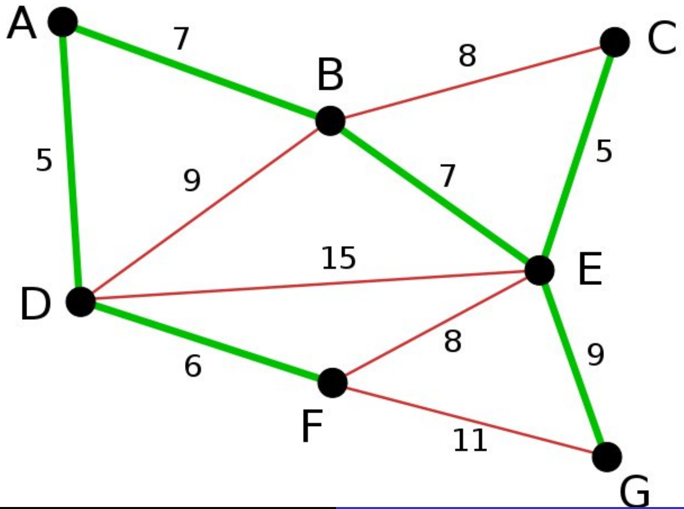
Kruskal's Algorithm in Action



Kruskal's Algorithm in Action



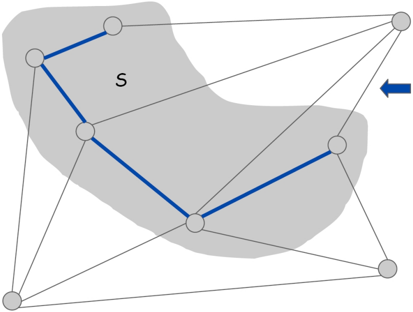
Kruskal's Algorithm in Action



Prim's Algorithm: One Vertex at a Time

Prim's Algorithm [Jarnik 1930, Dijkstra 1957, Prim 1959]

- Initialize S to be any vertex of G
- Apply cut property to S
- Add minimum weight edge $e = (u, v)$ in cutset D corresponding to S to the growing MST and add new v to S



Prim's Algorithm

Prim-MST(G, w)

1: let T contain first an arbitrary vertex $s \in V$
2: **while** T has fewer than $|V|$ vertices **do**
3: find the lightest edge connecting T to $G - T$
4: add it to T
5: **return** T

- Maintain set of explored vertices (that are already nodes in the tree) in S
- For each unexplored vertex $v \in V - S$, maintain the attachment cost $d[v] =$ weight of lightest edge connecting v to a node in S
- Key to a fast implementation: maintain $V - S$ as a priority queue, where the key of each unexplored vertex is the attachment cost, the weight of the lightest edge connecting v to S

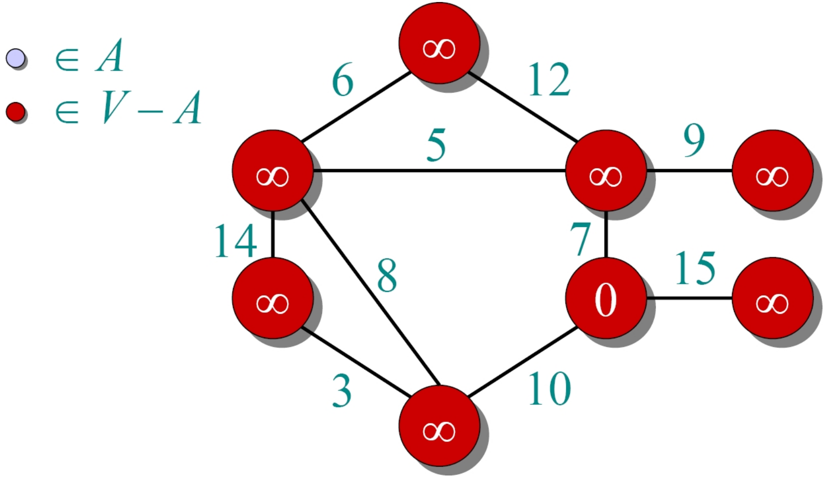
Implementing Prim's Algorithm

Remember: Maintain $V - S$ as a priority queue Q. The key of each vertex v in Q is the weight of the lightest edge connecting v to S

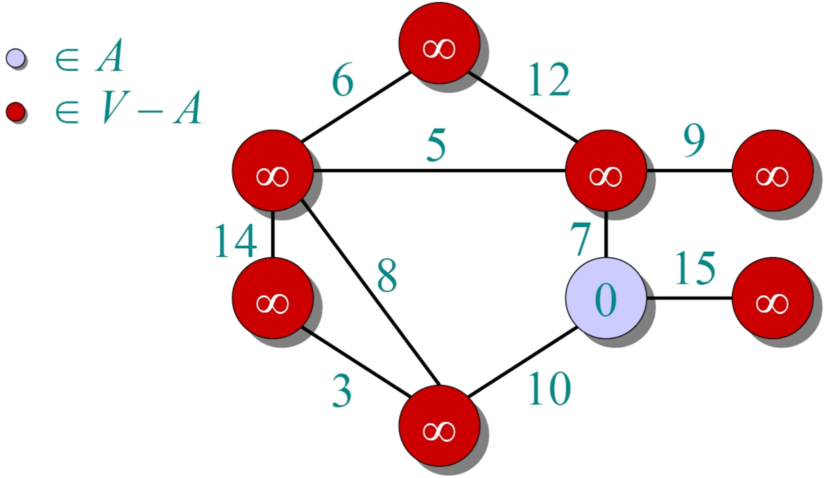
Prim-MST(G, w)

$Q \leftarrow V$
key[v] $\leftarrow \infty$ and $\pi[v] \leftarrow \infty$ for all $v \in V$
key[s] $\leftarrow$ 0 for an arbitrary $s \in V$
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ Extract-Min(Q)
    for each $v \in \mathrm{Adj(u)}$ **do**
        if $v \in Q$ and $w(u, v) < \mathrm{key[v]}$ **then**
            $\mathrm{key[v] \leftarrow w(u, v)}$
            $\mathrm{\pi(v) \leftarrow u}$
return $(v, \pi(\mathrm{v}))$ as the MST in the end

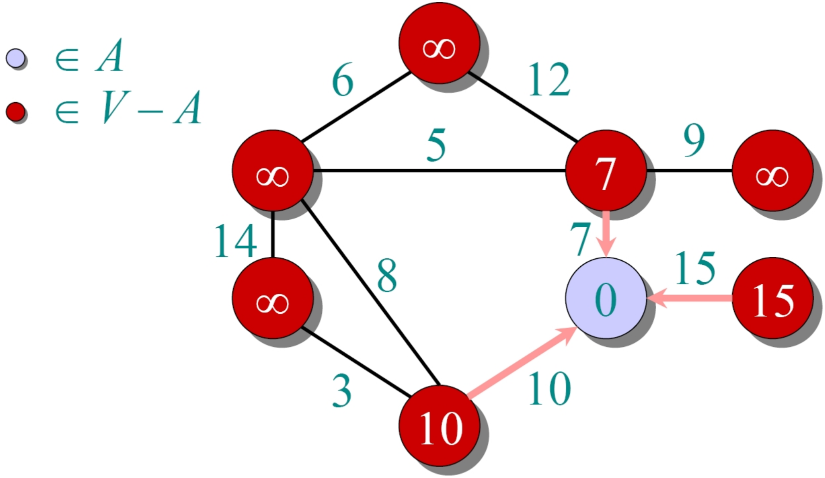
Prim's Algorithm in Action [explored vertices in A]



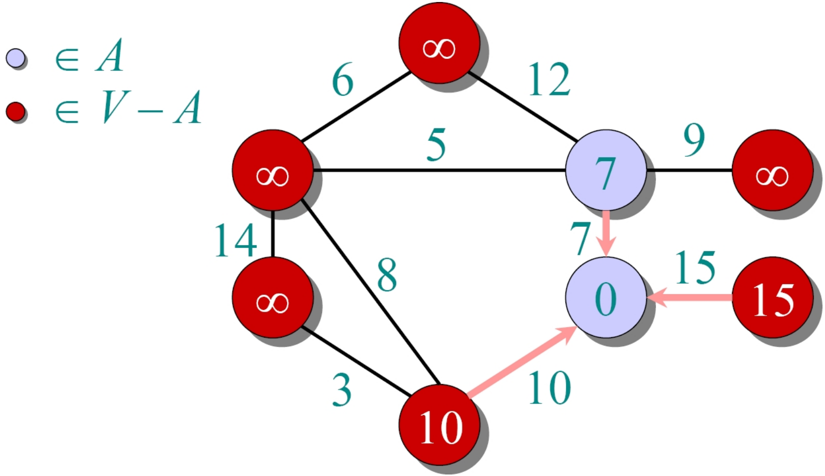
Prim's Algorithm in Action [explored vertices in A]



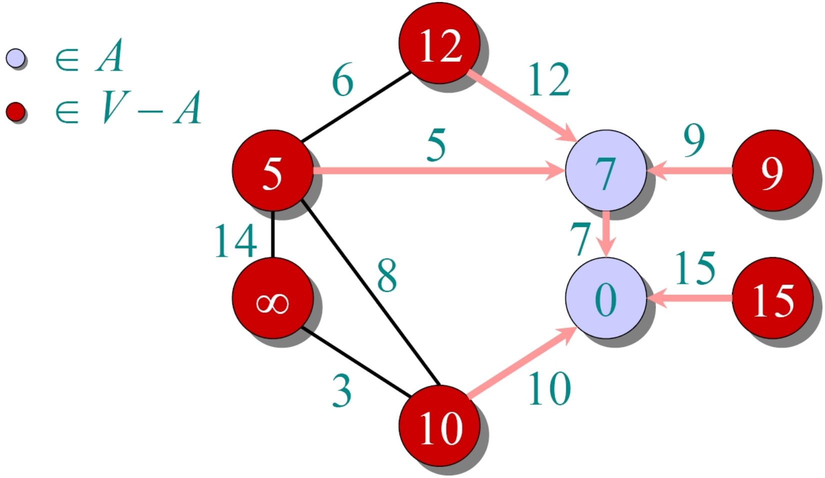
Prim's Algorithm in Action [explored vertices in A]



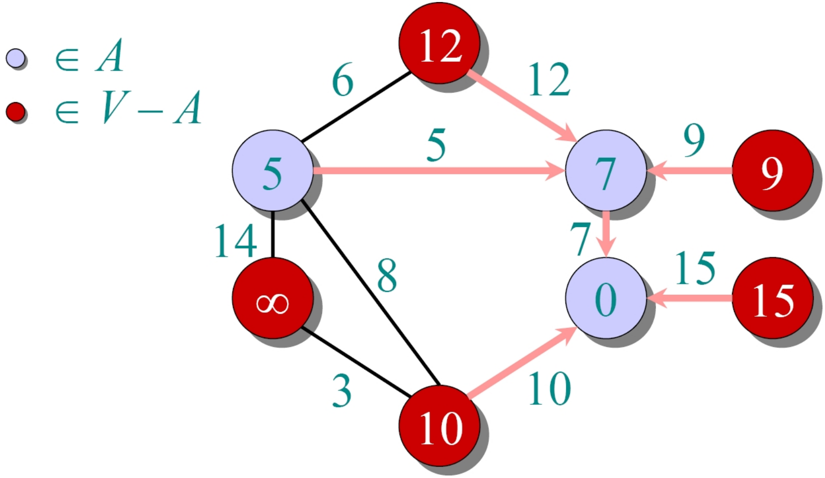
Prim's Algorithm in Action [explored vertices in A]



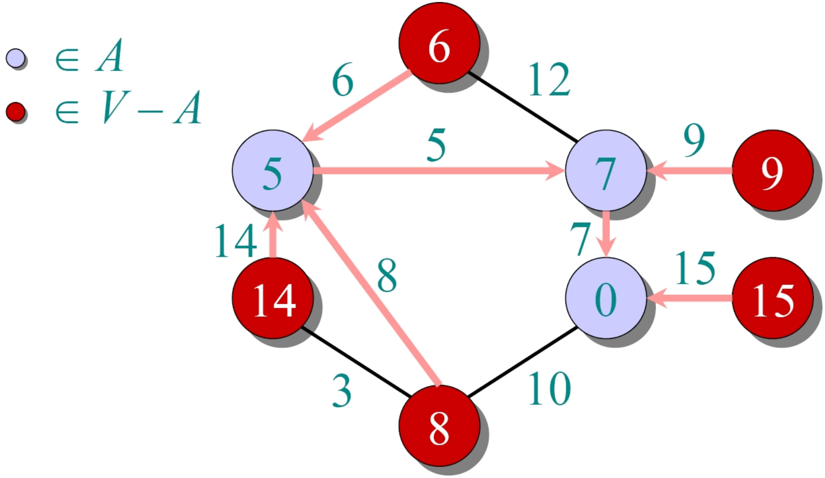
Prim's Algorithm in Action [explored vertices in A]



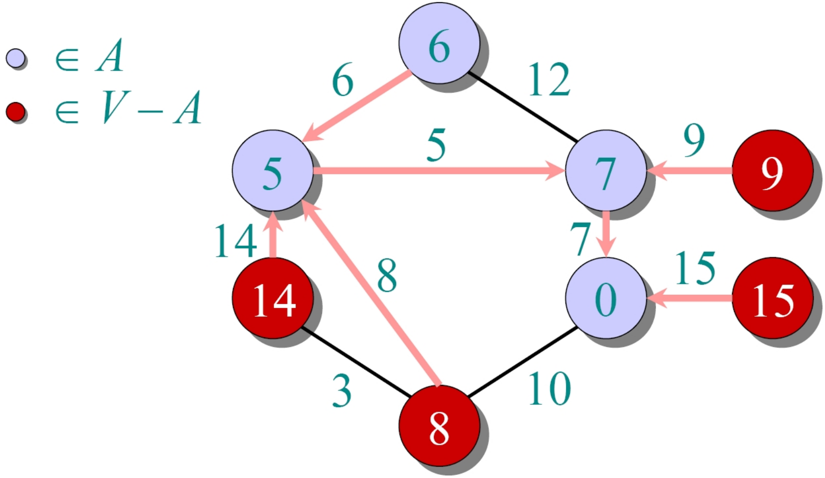
Prim's Algorithm in Action [explored vertices in A]



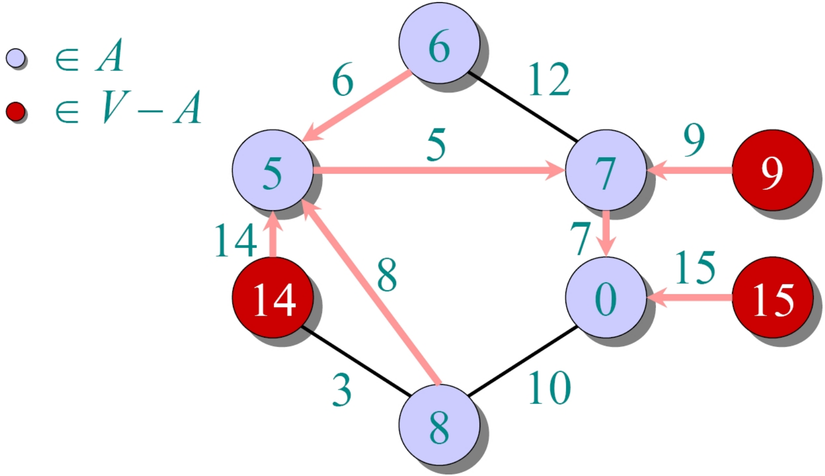
Prim's Algorithm in Action [explored vertices in A]



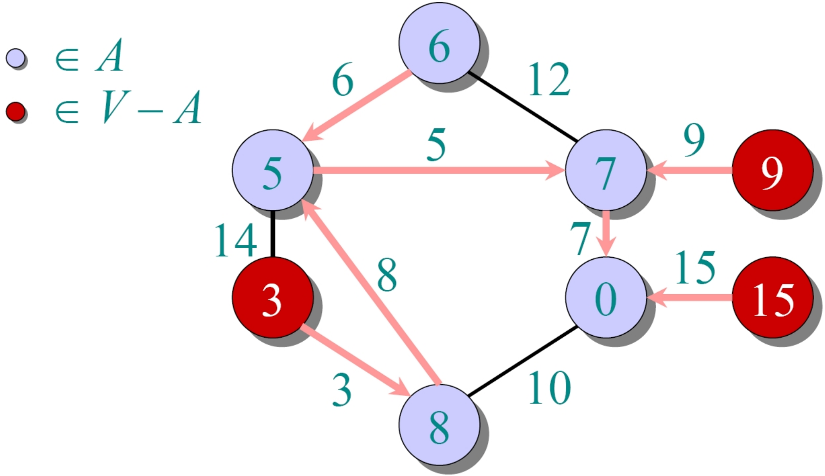
Prim's Algorithm in Action [explored vertices in A]



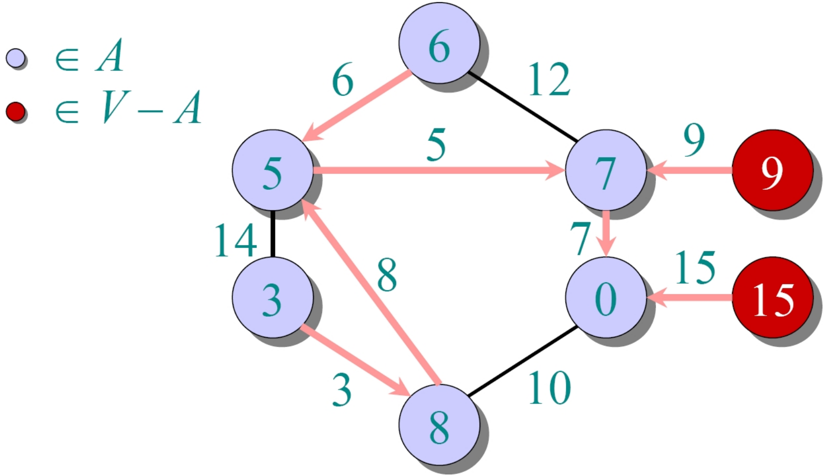
Prim's Algorithm in Action [explored vertices in A]



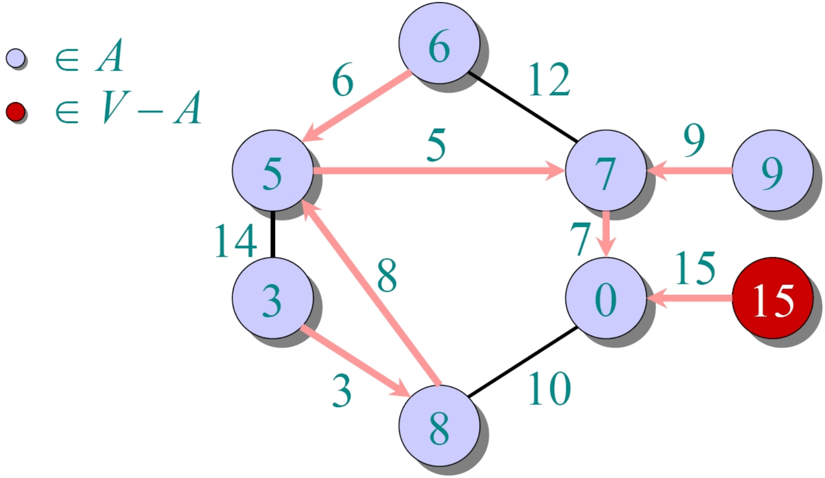
Prim's Algorithm in Action [explored vertices in A]



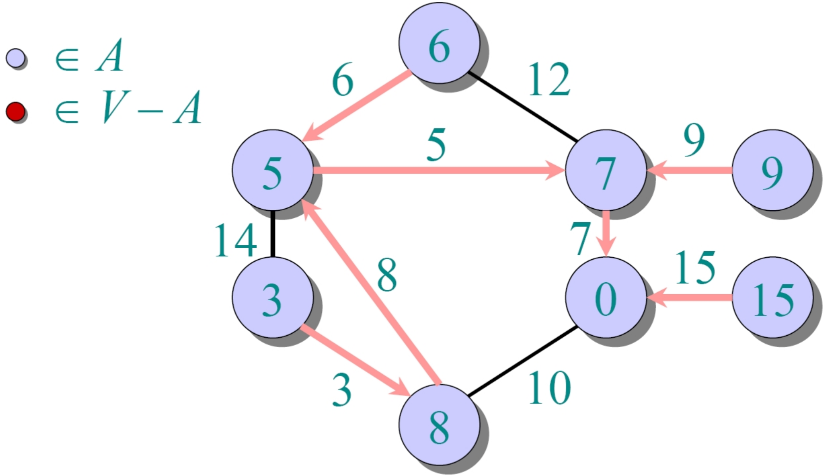
Prim's Algorithm in Action [explored vertices in A]



Prim's Algorithm in Action [explored vertices in A]



Prim's Algorithm in Action [explored vertices in A]



Analysis of Prim's Algorithm

$\Theta(V)$ total:

$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$

$|V|$ times:

while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)

$degree(u)$ times:

    for each $v \in Adj[u]$
      do if $v \in Q$ and $w(u, v) < key[v]$
        then $key[v] \leftarrow w(u, v)$
          $\pi[v] \leftarrow u$

Analysis of Prim's Algorithm

$$\text{Time} = \theta(V) \cdot T(\mathrm{Extract-Min}) + \theta(\mathrm{E}) \cdot \mathrm{T}(\mathrm{Decrease-Key})$$

Q	T(Extract-Min)	T(Decrease-Key)	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(1)$	$O(lgV)$	$O(E \cdot lgV)$
Fibonacci heap	$O(lgV)$	$O(1)$	$O(E + V \cdot lgV)$