# Computer Science 331

## Computation of Minimum-Cost Spanning Trees — Prim's Algorithm

Mike Jacobson

Department of Computer Science
University of Calgary

Lecture #33



# Outline







# Computation of Min-Cost Spanning Trees

**Motivation:** Given a set of sites (represented by vertices of a graph), connect these all as cheaply as possible (using connections represented by the edges of a weighted graph).

**Goals for Today:**

- presentation of the definitions needed to formally define a problem motivated by the above
- presentation of an algorithm (Prim's Algorithm) for solving the problem

**Reference:**

- *Introduction to Algorithms*, Chapter 23
- Text, Section 13.6 (problem), 13.6.2 (Prim's Algorithm)





# Costs of Spanning Trees in Weighted Graphs

Recall that if $G = (V, E)$ is a connected, undirected graph, then a *spanning tree* of $G$ is a subgraph $\widehat{G} = (\widehat{V}, \widehat{E})$ such that

- $\widehat{V} = V$ (so $\widehat{G}$ includes all the vertices in $G$)
- $\widehat{G}$ is a tree

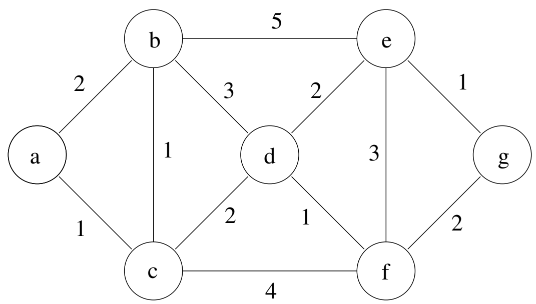
Suppose now that $G = (V, E)$ is a connected *weighted* graph with weight function $w : E \mapsto \mathbb{N}$, and that $G_1 = (V_1, E_1)$ is a spanning tree of $G$

The *cost* of $G_1$, $w(G_1)$, is the sum of the weights of the edges in $G_1$, that is,

$$w(G_1) = \sum_{e \in E_1} w(e).$$

## Example

Suppose $G$ is a weighted graph with weights as shown below.

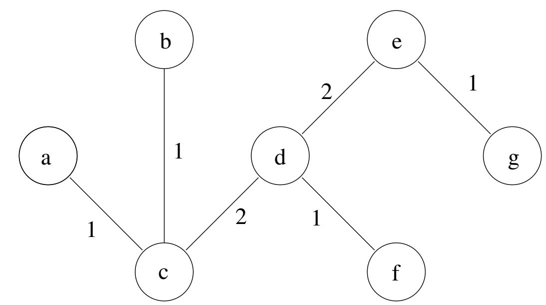







## Example

The cost of the following spanning tree, $G_1 = (V_1, E_1)$, is 8.

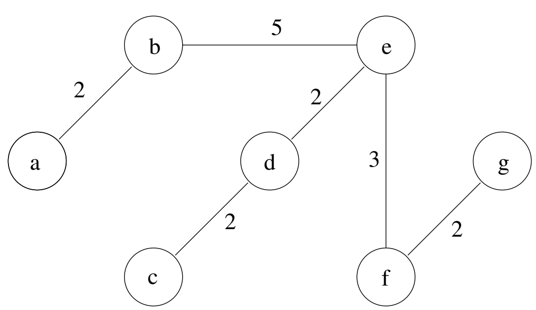







## Example

The cost of the following spanning tree, $G_2 = (V_2, E_2)$, is 16.








## Minimum-Cost Spanning Trees

Suppose $(G, w)$ is a weighted graph.

A subgraph $G_1$ of $G$ is a *minimum-cost spanning tree* of $(G, w)$ if the following properties are satisfied.

1. $G_1$ is a spanning tree of $G$.
2. $w(G_1) \leq w(G_2)$ for every spanning tree $G_2$ of $G$.

**Example:** In the previous example, $G_2$ is clearly *not* a minimum-cost spanning tree, because $G_1$ is a spanning tree of $G$ such that $w(G_2) > w(G_1)$.

- It can be shown that $G_1$ *is* a minimum-cost spanning tree of $(G, w)$.

## Existence of a Minimum-Cost Spanning Tree

**Lemma 1**

*Let $G$ be a weighted graph with weight function $w$ If $G$ is connected then $G$ has a minimum-cost spanning tree (which is not necessarily unique).*

**Proof.**

$G$ has at least *one* spanning tree, because:

- 

$G$ only has a *finite number* of spanning trees, because:

- 

Thus, there must exist a spanning tree of $G$ whose cost is less than or equal to that of any other spanning tree. □





## Building a Minimum-Cost Spanning Tree

To construct a minimum-cost spanning tree of $G = (V, E)$:

1. Start with $\widehat{G} = (\widehat{V}, \widehat{E})$, where $\widehat{V} \subseteq V$ and $\widehat{E} = \emptyset$.

   **Note:** $\widehat{G}$ is a subgraph of some minimum-cost spanning tree of $(G, w)$.
2. Repeatedly add vertices (if necessary) and edges — ensuring that $\widehat{G}$ is still a subgraph of a minimum-cost spanning tree as you do so.

   Continue doing this until $\widehat{V} = V$ and $|\widehat{E}| = |V| - 1$ (so that $\widehat{G}$ is a spanning tree of $\widehat{G}$).





## Building a Minimum-Cost Spanning Tree

**Additional Notes:**

- This can be done in several different ways, and there are at least two different algorithms that use this approach to solve this problem.

  The algorithm to be presented here begins with $\widehat{V} = \{s\}$ for some vertex $s \in V$, and makes sure that $\widehat{G}$ is always a *tree*.
- As a result, this algorithm is structurally very similar to *Dijkstra's Algorithm* to compute minimum-cost paths (which we have already discussed in class).





## Specification of Requirements

**Pre-Condition**

- $G = (V, E)$ is a **connected** graph with weight function $w$

**Post-Condition:**

- $\pi$ is a function $\pi : V \to V \cup \{\text{NIL}\}$
- If
  $$\widehat{E} = \{(\pi(v), v) \mid v \in V \text{ and } \pi(v) \neq \text{NIL}\}$$
  then $(V, \widehat{E})$ is a minimum-cost spanning tree for $G$
- The graph $G = (V, E)$ and its weight function have not been changed

## Data Structures

The algorithm (to be presented next) will use a **priority queue** to store information about weights of edges that are being considered for inclusion

- The priority queue will be a *MinHeap*: the entry with the *smallest* priority will be at the top of the heap
- Each node in the priority queue will store a *vertex* in $G$ and the *weight* of an edge incident to this vertex
- The *weight* will be used as the vertex's priority
- An array-based representation of the priority queue will be used

A second array will be used to locate each entry of the priority queue for a given node in constant time

**Note:** The data structures will, therefore, look very much like the data structures used by Dijkstra's algorithm.





## Pseudocode

**MST-Prim**$(G, w, s)$
  **for** $v \in V$ **do**
    $colour[v] = \text{white}$
    $d[v] = +\infty$
    $\pi[v] = \text{NIL}$
  **end for**
  Initialize an empty priority queue $Q$
  $colour[s] = \text{grey}$
  $d[s] = 0$
  add $s$ with priority 0 to $Q$





## Pseudocode, Continued

**while** ($Q$ is not empty) **do**
  $(u, c) = \text{extract-min}(Q)$ {Note: $c = d[u]$}
  **for** each $v \in Adj[u]$ **do**
    **if** $(colour[v] == \text{white})$ **then**
      $d[v] = w((u, v))$
      $colour[v] = \text{grey}$; $\pi[v] = u$
      add $v$ with priority $d[v]$ to $Q$
    **else if** $(colour[v] == \text{grey})$ **then**
      Update information about $v$ (Shown on next slide)
    **end if**
  **end for**
  $colour[u] = \text{black}$
**end while**
**return** $\pi$



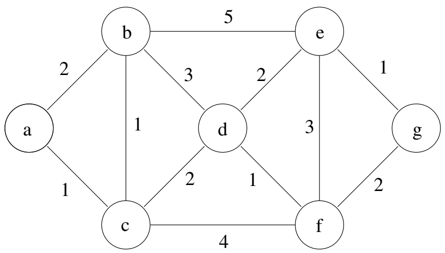


## Pseudocode, Concluded

**Updating Information About v**
  **if** $(w((u, v)) < d[v])$ **then**
    $old = d[v]$
    $d[v] = w((u, v))$
    $\pi[v] = u$
    Use `Decrease-Priority` to replace $(v, old)$ in $Q$ with $(v, d[v])$
  **end if**

# Example

Consider the execution of MST-Prim($G$, $a$):

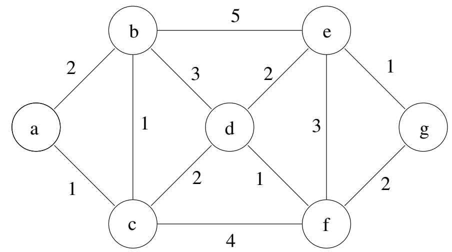



| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| d | | | | | | | |
| $\pi$ | | | | | | | |





# Example (Step 1)

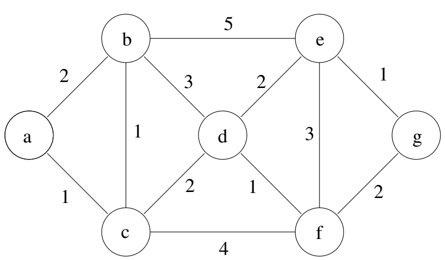



| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| d | | | | | | | |
| $\pi$ | | | | | | | |





# Example (Step 2)

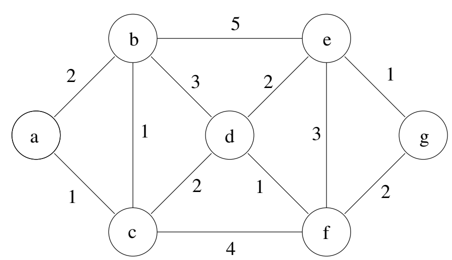



| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| d | | | | | | | |
| $\pi$ | | | | | | | |



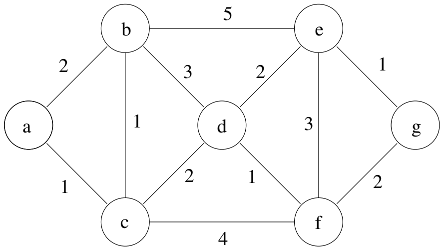


# Example (Step 3)




| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| d | | | | | | | |
| $\pi$ | | | | | | | |

# Example (Step 4)

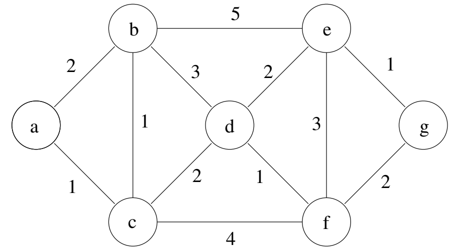



| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| d | | | | | | | |
| $\pi$ | | | | | | | |





# Example (Step 5)

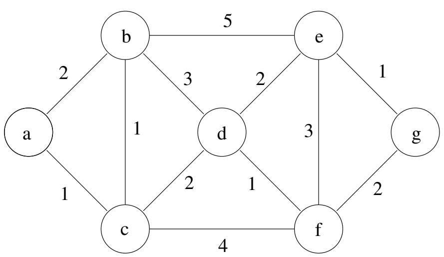



| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| d | | | | | | | |
| $\pi$ | | | | | | | |





# Example (Step 6)

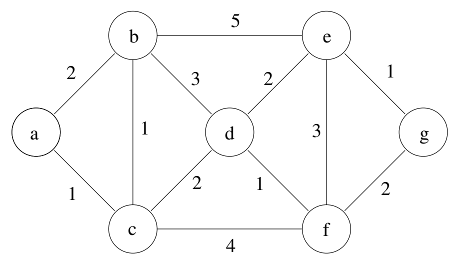



| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| d | | | | | | | |
| $\pi$ | | | | | | | |





# Example (Step 7)




| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| d | | | | | | | |
| $\pi$ | | | | | | | |