Computer Science 331

Prim's Algorithm

Mike Jacobson

Department of Computer Science
University of Calgary

Lecture #34

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).

Goal for Today: Presentation of an algorithm to compute a minimum-cost spanning tree of a graph

Reference:

- *Introduction to Algorithms,* Chapter 23
- Text, Section 12.6 (p.666-670), variation without a priority queue

Costs of Spanning Trees in Weighted Graphs

Suppose that (G, w) is a *weighted graph*.

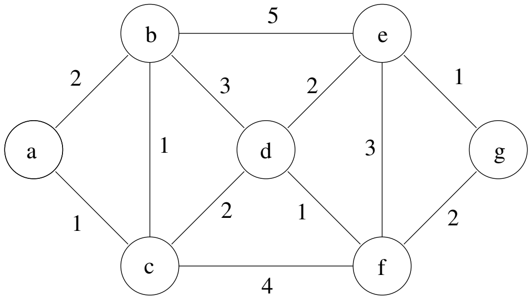
Let $G_1 = (V_1, E_1)$ be a spanning tree of the undirected graph G.

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

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

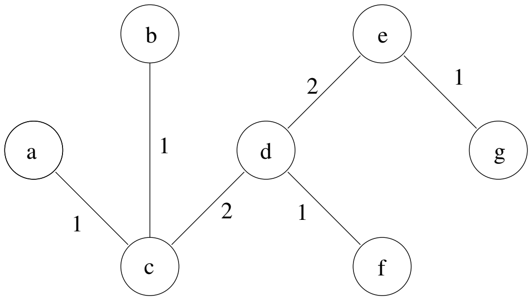
Example

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



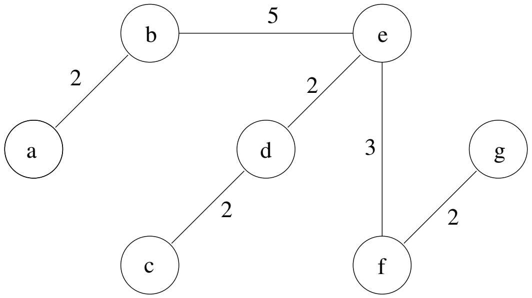
Example

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).

Building a Minimum-Cost Spanning Tree

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

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** weighted graph

Post-Condition:

- π 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 edge-weights) has 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 *node* in G and the *weight* of an edge incident on this node
- The *weight* will be used as the node'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($G$)
  for $v \in V$ do
    $colour[v] = $ white
    $d[v] = +\infty$
    $\pi[v] = $ NIL
  end for
  Initialize the priority queue $Q$ to be empty
  Choose some vertex $s \in V$
  $colour[s] = $ grey
  $d[s] = 0$
  enqueue(($s$, 0))
```

Pseudocode, Continued

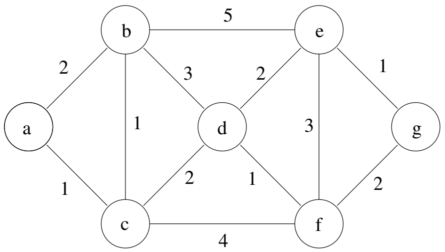
```
while ($Q$ is not empty) do
  $(u, d) = $ extract-min($Q$) {Note: $d = d[u]$}
  for each $v \in Adj[u]$ do
    if ($colour[v] == $ white) then
      $d[v] = w((u, v))$
      $colour[v] = $ grey; $\pi[v] = u$
      enqueue($v$, $d[v]$)
    else if ($colour[v] == $ grey) then
      Update information about $v$ {Shown on next slide}
    end if
  end for
  $colour[u] = $ 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-Key to replace $(v, old)$
    in $Q$ with $(v, d[v])$
  end if
```

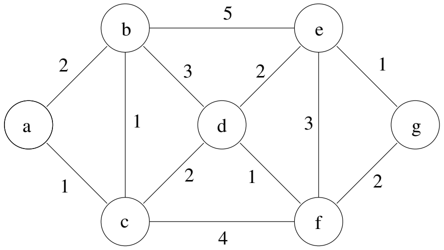
Example

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



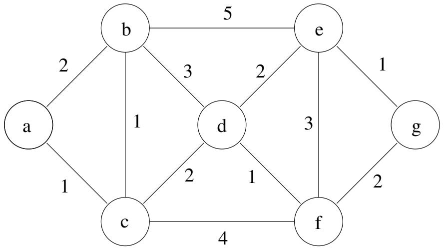
	a	b	c	d	e	f	g
d							
π							

Example (Step 1)



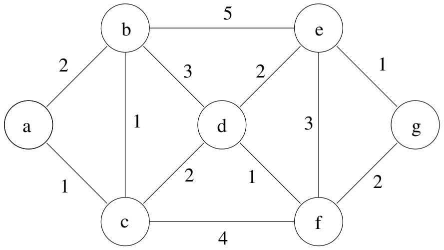
	a	b	c	d	e	f	g
d							
π							

Example (Step 2)



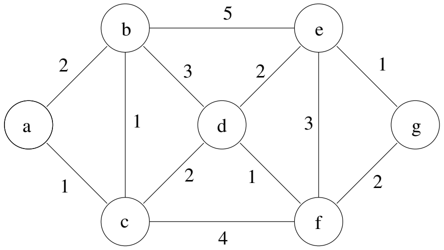
	a	b	c	d	e	f	g
d							
π							

Example (Step 3)



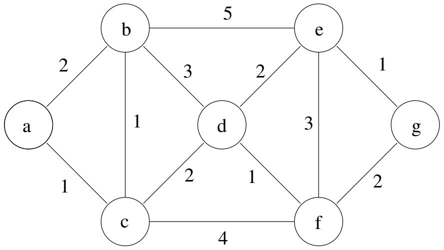
	a	b	c	d	e	f	g
d							
π							

Example (Step 4)



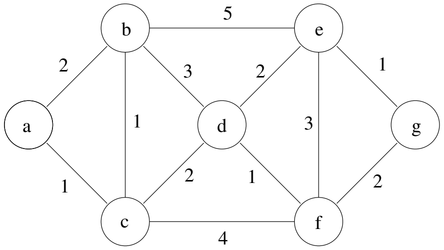
	a	b	c	d	e	f	g
d							
π							

Example (Step 5)



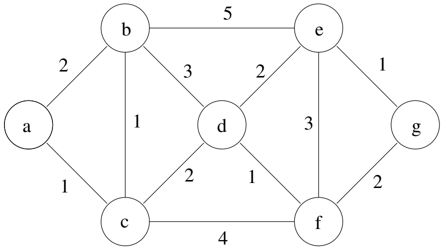
	a	b	c	d	e	f	g
d							
π							

Example (Step 6)



	a	b	c	d	e	f	g
d							
π							

Example (Step 7)



	a	b	c	d	e	f	g
d							
π							