Section 6.2. Minimum-Weight Spanning Trees

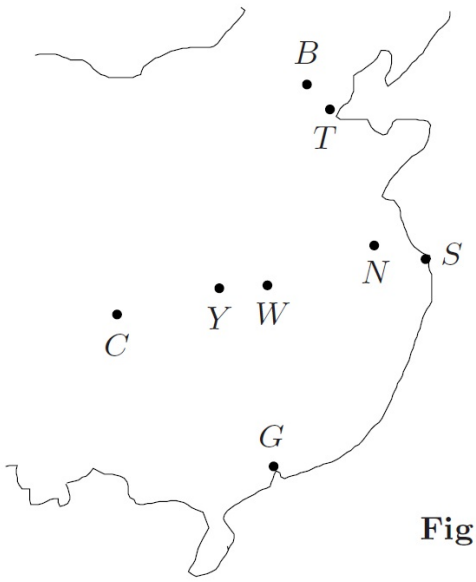
Note. In this section we consider spanning trees of edge-weighted graphs. Among all the spanning trees, we seek one of minimum sum of edge weights. You probably encountered this problem in Introduction to Graph Theory (MATH 4347/5347); see my online notes for this class on Section 7.1. Spanning Tree Algorithms. There, you likely saw Kruskal's Algorithm. We present a different algorithm in this section, but we will see Kruskal-s Algorithm in our Section 8.5. Greedy Heuristics where we refer to it as the Borůvka-Kruskal Algorithm (Algorithm 8.22). That algorithm is also encountered in Introduction to to Operations Research 1 (not an official ETSU class; see my online notes for the class on Section 6.2. Minimal Spanning Tree Algorithm). We motivate the problem with an application.

Note. Bondy and Murty describe a problem of creating an electrical grid between right locations in China, denoted B, C, G, N, S, T, W, Y. See Figure 6.5 below where the distances between the locations are given (in kilometers). The problem is to create the electrical grid with the minimum total connecting distance. We solve the problem by considering an edge weighted complete graph K_8 with vertices corresponding to locations and edge weights representing distances. This is a special case of the following more general problem.

Problem 6.8. Minimum-Weight Spanning Tree.

Given: a (edge) weighted connected graph G.

Find: a minimum weight spanning tree T in G.



	C	G	N	S	T	W	Y
B	1457	1892	901	1078	111	1057	1117
C	–	978	1199	1430	1442	750	473
G	–	–	1133	1197	1820	837	867
N	–	–	–	267	800	459	727
S	–	–	–	–	970	681	962
T	–	–	–	–	–	988	1080
W	–	–	–	–	–	–	285

Fig. 6.5. The China hydro-electric grid problem

Definition. For a given weighted connected graph G, a tree T solving the Minimum-Weight Spanning Tree problem is an *optimal tree*.

Note. The algorithm we present below is due to Vojtěch Jarnik and R. C. Prim. Jarnik's work appears in "O jistém problému minimálním [About a Certain Minimal Problem]" *Práce Moravské Prírodovedecké Spolecnosti*, **6**, 57–63 (1930); this work was originally in a letter to Czech mathematician Otakar Borůvka following Borůvka's publication of his algorithm. You can see Jarnik's paper online on the Czech Digital Mathematics Library. Prim's work appears rather later in "Shortest Connection Networks and Some Generalizations," *The Bell System Technical Journal*, **36**(6), 1389–1401 (1957); it seems challenging to find a free copy of this online. An arbitrary vertex r is selected as the root of tree T. At each stage the edge added to the current tree T is any edge of least weight in the edge cut associated with

T. The vertices are coloured black as they are added to T. Each vertex is assigned an infinite cost, $c(v) = \infty$. This requires us to impose the condition "$x < \infty$" for all $x \in \mathbb{R}$ on the inequality symbol (this is standard interpretation in analysis when dealing with the extended real numbers, $\mathbb{R} \cup \{\infty\}$). Each uncoloured vertex is assigned a "provisional" cost $c(v)$ that is the least weight of an edge linking v to some black vertex u (i.e., some u already in T), if such u exists. Vertex u is the named the provisional predecessor of v, $p(v)$. These provisional labels are updated at each stage of the algorithm.

Algorithm 6.9. The Jarník-Prim Algorithm.

Input: a weighted connected graph (G, w)

Output: an optimal tree T of G with predecessor function p, and its weight $w(T)$

1. *set* $p(v) := \varnothing$ *and* $c(v) := \infty$ *for* $v \in V$, *and* $w(T) := 0$
2. *choose any vertex* r *(as the root)*
3. *replace* $c(r)$ *by 0*
4. **while** *there is an uncoloured vertex* **do**
5. *choose such a (uncoloured) vertex* u *of minimum cost* $c(u)$
6. *colour* u *black*
7. **for** *each uncoloured vertex* v *such that* $w(uv) < c(v)$ **do**
8. *replace* $p(v)$ *by* u *and* $c(v)$ *by* $w(uv)$
9. **end for**
10. *replace* $w(T)$ *by* $w(T) + c(u)$
11. **end while**
12. *return* $(p, w(T))$.

Note. I have interchanged Steps 9 and 10 from the book's version of the algorithm. This is because the updating of the weight of the tree *must* be increased by $c(u)$ when u is added to T. The book has this inside the **for** loop and this can cause a problem in two ways. If there are for uncoloured vertices v as describe, the the weight $w(T)$ will not be updated. Second, if there are multiple such uncoloured vertices v then $w(T)$ is updated multiple times! One vertex is added to T (vertex u) and so we need to replace $w(T)$ with $w(T) + c(u)$ exactly once. In the example we work you will see that both problems can occur (namely (1) no vertices v with $w(uv) < c(v)$ and (2) multiple vertices v with $w(uv) < c(v)$).

Definition. A rooted spanning tree output by the Jarník-Prim Algorithm is a *Jarník-Prim tree*.

Example 6.2.A. We now apply Algorithm 6.9 to the Chinese electrical grid problem. We start (in Step 2) with choosing Y as the root, and so (Step 3) $c(Y) = 0$ and $c(v) = \infty$ for all other vertices. As we start the **while** loop, Step 5 sets $u = Y$ and Y is coloured black in Step 6. Next, in Steps 7–9 (in my numbering scheme) for the seven uncoloured vertices v adjacent to $u = Y$ (i.e., all vertices except Y since we have $G = K_8$) we look for edges between $u = Y$ and v of weight less than $c(v) = \infty$. This is all seven other vertices and so (Step 8) we set $p(v) = u$ and set $c(v) = w(uv)$. This gives $u = Y$ as the predecessor of all other vertices (so at this stage the tree is a star with center Y) and the costs are simply the distances

between Y and v (given in the last column of the array in Figure 6.5):

$$c(B) = 1117, \quad c(C) = 473, \quad c(G) = 867, \quad c(N) = 727,$$

$$c(S) = 962, \quad c(T) = 1080, \quad c(W) = 285.$$

This completes the **for** steps. Since $c(u) = c(Y) = 0$. Step 10 gives $w(T) = 0$.

On the second pass through the **while** loop, Step 5 sets $u = W$ and W is coloured black in Step 6. In Steps 7–9 for the six uncoloured vertices v adjacent to $u = W$ we look for edges between $u = W$ and v of weight less than $c(v)$. Thus we consider

v	$c(v)$	$w(uv) = w(Wv)$	$w(uv) < c(v)$?
B	1117	1057	YES
C	473	750	NO
G	867	837	YES
N	727	459	YES
S	962	681	YES
T	1080	988	YES

Notice that $w(uv)$ is the second-to-the-last column of the array in Figure 6.5. So Step 8 is processed for $v \in \{B, G, N, S, T\}$ to give new costs and a new predecessor $(u = W)$ for these vertices resulting in

$$p(B) = W, \;\; p(C) = Y, \;\; p(G) = W, \;\; p(N) = W, \;\; p(S) = W, \;\; p(T) = W$$

$$c(B) = 1057, \quad c(C) = 473, \quad c(G) = 837,$$

$$c(N) = 459, \quad c(S) = 681, \quad c(T) = 988.$$

On the third pass through the **while** loop, Step 5 sets $u = N$ and N is coloured black in Step 6. In Steps 7–9 for the five uncoloured vertices v adjacent to $u = N$

we look for edges between $u = N$ and v of weight less than $c(v)$. Thus we consider:

v	$c(v)$	$w(uv) = w(Wv)$	$w(uv) < c(v)$?
B	1057	901	YES
C	473	1199	NO
G	837	1133	NO
S	681	267	YES
T	988	800	YES

So Step 8 is processed for $v \in \{B, S, T\}$ to give new costs and a new predecessor $(u = N)$ for these vertices resulting in:

$$p(B) = N, \;\; p(C) = Y, \;\; p(G) = W, \;\; p(S) = N, \;\; p(T) = N$$

$$c(B) = 901, \quad c(C) = 473, \quad c(G) = 837, \quad c(S) = 267, \quad c(T) = 800.$$

Since $c(u) = c(N) = 459$, Step 10 gives $w(T) = 285 + 459 = 744$.

On the fourth pass through the **while** loop, Step 5 sets $u = S$ and S is coloured black in Step 6. In Steps 7–9 for the four uncoloured vertices adjacent to $u = S$ we look for edges between $u = S$ and v of weight less than $c(v)$. Thus we consider:

v	$c(v)$	$w(uv) = w(Wv)$	$w(uv) < c(v)$?
B	901	1078	NO
C	473	1430	NO
G	837	1197	NO
T	800	970	NO

So Step 8 is not processed. Since $c(u) = c(S) = 267$, Step 10 gives $w(T) = 744 + 267 = 1011$.

On the fifth pass through the **while** loop, Step 5 sets $u = C$ and C is coloured black in Step 6. In Steps 7–9 for the three uncoloured vertices adjacent to $u = C$ we look for edges between $u = C$ and v of weight less than $c(v)$. Thus we consider

v	$c(v)$	$w(uv) = w(Wv)$	$w(uv) < c(v)$?
B	901	1457	NO
G	837	978	NO
T	800	1442	NO

So Step 8 is not processed. Since $c(u) = c(C) = 473$, Step 10 gives $w(T) = 1011 + 473 = 1484$.

On the sixth pass through the **while** loop, Step 5 sets $u = T$ and T is coloured black in Step 6. In Steps 7–9 for the three uncoloured vertices adjacent to $u = T$ we look for edges between $u = T$ and v of weight less than $c(v)$. Thus we consider

v	$c(v)$	$w(uv) = w(Wv)$	$w(uv) < c(v)$?
B	901	111	YES
G	837	1820	NO

So Step 8 is processed for $v = B$ to give a new cost and a new predecessor for B: $p(B) = T$ and $c(B) = 111$. Since $c(u) = c(T) = 800$, Step 10 gives $w(T) = 1484 + 800 = 2284$.

On the seventh pass through the **whole** loop, Step 5 sets $u = B$ and B is coloured block in Step 6. In Steps 7–9 for the one uncoloured vertex adjacent to C, we consider the edge between $u = B$ and $v = G$. Again we consider:

v	$c(v)$	$w(uv) = w(Wv)$	$w(uv) < c(v)$?
G	837	1892	NO

So Step 8 is not processed. Since $c(u) = c(B) = 111$ Step 10 gives $w(T) = 2284 + 111 = 2395$.

On the eighth and final pass through the **while** loop, Step 5 sets $u = G$ and G is coloured black in Step 6. Since there are no uncoloured vertices, Step 8 is not processed. Step 10 gives $w(T) = 2395 + 837 = 3232$.

With the algorithm complete, we have the predecessors and costs:

$$Y = \text{root}, \;\; p(B) = T, \;\; p(C) = Y, \;\; p(G) = W,$$

$$p(N) = W, \;\; p(S) = N, \;\; p(T) = N, \;\; p(W) = Y.$$

Figure 6.6 below shows that tree T, with the edges numbered in the order in which they were added to T. Notice that we have:

Edge #	Edge	Distance
1	YW	285
2	WN	459
3	NS	267
4	YC	473
5	NT	800
6	TB	111
7	WG	837

The distances sum to 3232 (kilometers), as stated above. The next theorem guarantees that the Jarník-Prim Algorithm in fact yields an optimal tree.

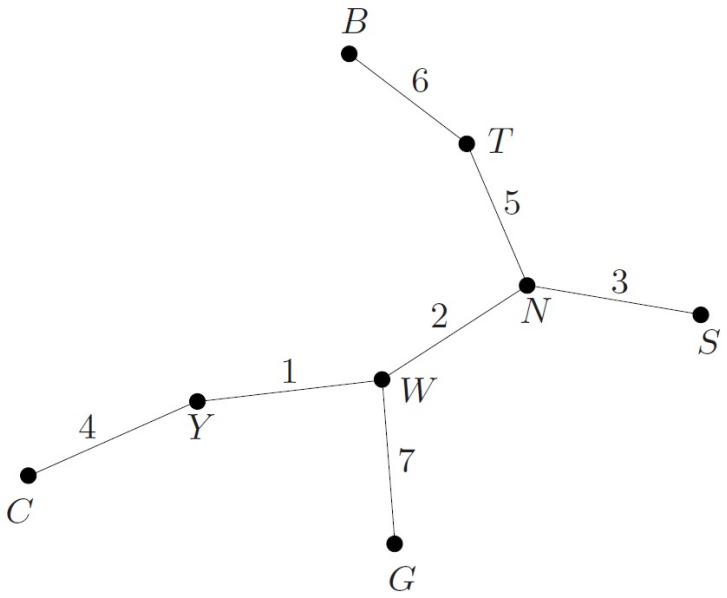


Fig. 6.6. An optimal tree returned by the Jarník–Prim Algorithm

Theorem 6.10. Every Jarník-Prim tree is an optimal tree.

Revised: 6/5/2022