CHAPTER

TEN

Graph Algorithms

10.1 Introduction

In this section, we consider simple graphs without loops or multiple edges.

Proposition 10.1. *Let $\mathbf{G} = (V, E)$ be a graph on n vertices, and let $F = (V, S)$ be a spanning forest. Then $0 \leq |S| \leq n - 1$. Futhermore, if $|S| = n - k$, then F has k components. In particular, F is a spanning tree if and only if it contains $n - 1$ edges.*

Proposition 10.2. *Let $T = (V, S)$ be spanning tree in a graph $\mathbf{G}$, and let $e = xy$ be an edge of $\mathbf{G}$ which does not belong to T. Then*

1. *There is a* unique *path $P = (x_0, x_1, x_2, \ldots, x_t)$ with (a) $x = x_0$; (b) $y = x_y$; and (c) $x_i x_{i+1} \in S$ for each $i = 0, 1, 2, \ldots, t - 1$.*
2. *For each $i = 0, 1, 2, \ldots, t - 1$, let $f_i = x_i x_{i+1}$ and then set*

$$S_i = \{e\} \cup \{g \in S : g \neq f\},$$

i.e., we exchange *edge f for edge e. Then $T_i = (V, S_i)$ is a spanning tree of $\mathbf{G}$.*

10.2 Minimum Weight Spanning Trees

In this section, we consider pairs (G, w) where $\mathbf{G} = (V, E)$ is a connected graph and $w : E \to \mathbb{N}_0$. For each edge $e \in E$, the quantity $w(e)$ is called the *weight* of e. Given a set S of edges, we define the *weight* of S, denoted $w(S)$, by setting $w(S) = \sum_{e \in S} w(e)$. In particular, the weight of a spanning tree T is just the sum of the weights of the edges in T.

Problem. Find a minimum weight spanning tree T of $\mathbf{G}$.

To solve this problem, we will develop *two* efficient graph algorithms, each having certain computational advantages and disadvantages. In each case, the argument to show that the algorithm is optimal rests on the following technical lemma. To avoid trivialities, we assume $n \geq 3$.

Lemma 10.3. *Let F be a spanning forest of $\mathbf{G}$ and let C be a component of F. Also, let $e = xy$ be an edge of minimum weight among all edges with one endpoint in C and the other not in C. The among all spanning trees of $\mathbf{G}$ that contain the forest F, there is one of minimum weight that contains the edge e.*

Proof. Let $T = (V, S)$ be any spanning tree of minimum weight among all spanning trees that contain the forest F, and suppose that $e = xy$ is not an edge in T. Then let $P = (x_0, x_1, x_2, \ldots, x_t)$ be the unique path with (a) $x = x_0$; (b) $y = x_t$; and (c) $x_i x_{i+1} \in S$ for each $i = 0, 1, 2, \ldots, t-1$. Without loss of generality, we may assume that $x = x_0 \in C$ while $y = x_t \notin C$. Then there is a least non-negative integer i for which $x_i \in C$ and $x_{i+1} \notin C$. It follows that $x_j \in C$ for all j with $0 \leq j \leq i$.

The edge e has minimum weight among all edges with one endpoint in C and the other not in C, so $w(e) \leq w(f)$. Then let T_i be the tree obtained by exchanging the edge f for edge e. It follows that $w(T_i) = w(T) - w(f) + w(e) \leq w(T)$. Furthermore, T_i contains the spanning forest F as well as the edge e. □

10.2.1 Kruskal's Algorithm

In this secton, we develop one of the best known algorithms for finding a minimum weight spanning tree. It is known as Kruskal's Algorithm, although some prefer the descriptive label: *Avoid Cycles.*

To start the algorithm, we sort the edges according to weight. To be more precise, let q denote the number of edges in $\mathbf{G}$. Then label the edges as $e_1, e_2, e_3, \ldots, e_q$ so that $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_q)$.

Initialization. Set $S = \emptyset$ and $i = 0$.

Inductive Step. While $|S| < n - 1$, let j be the least non-negative integer so that $j > i$ and there are no cycles in $S \cup \{e_j\}$. Then (using pseudo-code) set

$$i = j \quad \text{and} \quad S = S \cup \{j\}$$

The correctness of Kruskal's Algorithm follows from an elementary inductive argument. First, the set S is initialized as the empty set so there is certainly a minimum weight spanning tree containing all the edges in S. Now suppose that for some i with $0 \leq i < n$, $|S| = i$ and there is a minimum weight spanning tree containing all the edges in S. Let F be the spanning forest determined by the edges in S, and let $C_1, C_2, \ldots, C_s$ be the components of F. For each $k = 1, 2, \ldots, s$, let f_k be a minimum weight edge with

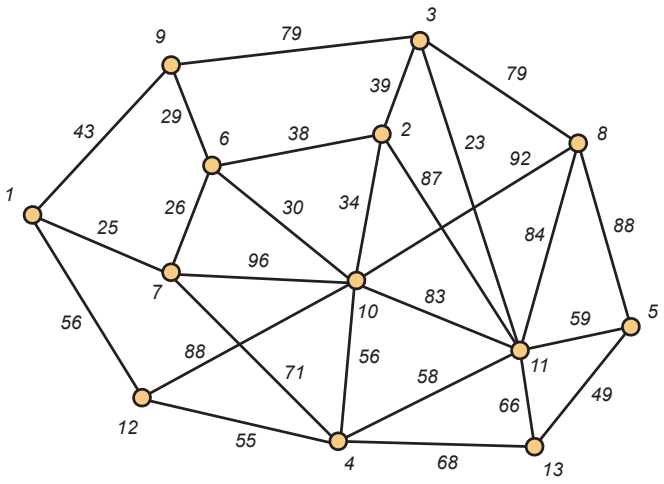

FIGURE 10.1:

one endpoint in C_k and the other not in C_k. Then the edge e added to S by Kruskal's Algorithm is just the minimum weight edge among $\{f_1, f_2, \ldots, f_s\}$.

10.2.2 Prim's Algorithm

This algorithm is also known by the more descriptive label: *Build Tree*. We begin by choosing a root vertex r.

Initialization. Set $W = \{r\}$ and $S = \emptyset$.

Inductive Step. While $|W| < n$, let e be an edge of minimum weight among all edges with one endpoint in W and the other not in W. If $e = xy$, $x \in W$ and $y \notin W$, update W and S by setting:

$$W = W \cup \{y\} \quad \text{and} \quad S = S \cup \{e\}$$

The correctness of Prim's algorithm follows immediately from Lemma 10.3.

Example 10.4. Consider the weighted graph shown in Figure 10.1.

For this graph, the following lists show the edges making up minimum weight spanning trees in the order they would be selected by the two algorithms.

Kruskal's Algorithm	Prim's Algorithm
3 11 23	1 7 25
1 7 25	6 7 26
6 7 26	6 9 29
6 9 29	6 10 30
6 10 30	2 10 34
2 10 34	2 3 39
2 3 39	3 11 23
5 13 49	4 10 56
4 12 55	4 12 55
4 10 56	5 11 59
5 11 59	5 13 49
3 8 79	3 8 79

In this example, the two algorithms have identified exactly the same set of edges—although they are listed in different order. The minimum weight of the spanning tree these edges determine is

$$504 = 23 + 25 + 26 + 29 + 30 + 34 + 39 + 49 + 55 + 56 + 59 + 77$$

10.3 Digraphs

A *digraph* $\mathbf{G}$ is a pair (V, E) where V is a vertex set and $E \subset V \times V$ with $x \neq y$ for every $(x, y) \in E$. We consider the pair (x, y) as a *directed edge* from x to y. Note that for distinct vertices x and y from V, the ordered pairs (x, y) and (y, x) are distinct, so the digraph may have one, both or neither of the directed edges (x, y) and (y, x).

Diagrams of digraphs use arrowheads on the edges to indicate direction. This is illustrated in Figure 10.2.

When $\mathbf{G}$ is a digraph, a sequence $P = (r = u_0, u_1, \ldots, u_t = x)$ of distinct vertices is called a *directed path* from r to x when $(u_i u_{i+1})$ is a directed edge in $\mathbf{G}$ for every $i = 0, 1, \ldots, t-1$. A directed path $C = (r = u_0, u_1, \ldots, u_t = x)$ is called a *directed cycle* when (u_t, u_0) is a directed edge of $\mathbf{G}$.

10.4 Dijkstra's Algorithm for Shortest Paths

In this section, we consider a pair $(\mathbf{G}, w)$ where $\mathbf{G} = (V, E)$ is a digraph and $w : E \to \mathbb{N}_0$ is a function assigning to each directed edge (x, y) a non-negative weight $w(x, y)$. However, in this section, we interpret weight as *distance* so that $w(x, y)$ is now called the *length* of the edge (x, y). So if $P = (r = u_0, u_1, \ldots, u_t = x)$ is a directed path from r to

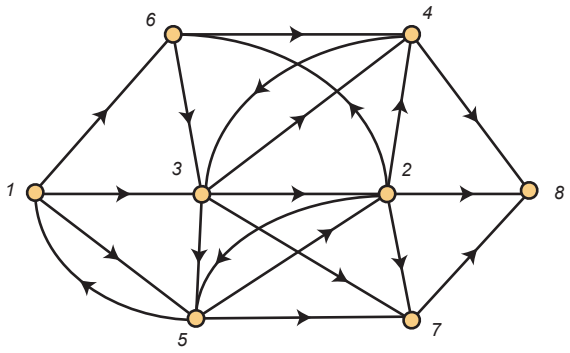

FIGURE 10.2: A DIGRAPH

x, then the *length* of the path P is just $\sum_{i=0}^{t-1} w(u_i u_{i+1})$. The *distance* from r to x is then defined to be the minimum length of a directed path from r to x.

Problem. For each vertex x, find the distance from r to x. Also, find a shortest path from r to x.

10.4.1 Description of the Algorithm

First, extend w by setting $w(x, y) = \infty$ when $x \neq y$ and (x, y) is not a directed edge of $\mathbf{G}$. Then let $n = |V|$.

At Step i, where $1 \leq i \leq n$, we will have determined:

1. A sequence $\sigma = (v_1, v_2, v_3, \ldots, v_i)$ of distinct vertices from $\mathbf{G}$ with $r = v_1$. These vertices are called *permanent* vertices, while the remaining vertices will be called *temporary* vertices.

2. For each vertex $x \in V$, we will have determined a number $d(x)$ and a path $P(x)$ from r to x of length $d(x)$.

Initialization: Step 1. Set $d(r) = 0$ and let $P(r) = (r)$ be the trivial one point path. Also, set $\sigma = (r)$. For each $x \neq r$, set $d(x) = w(r, x)$ and $P(x) = (r, x)$.

Inductive Step. If $i < n$, then for each temporary x, let

$$d(x) = \min\{d(x), d(v_i) + w(v_i, x)\}.$$

If this assignment results in a reduction in the value of $d(x)$, let $P(x)$ be the path obtained by adding x to the end of $P(v_i)$.

Let x be a temporary vertex x for which $d(x)$ is minimum. Set $v_{i+1} = x$, and update σ by appending v_{i+1} at the end.

10.4.2 The Correctness of Dijkstra's Algorithm

First, we have the following two elementary propositions.

Proposition 10.5. *Let x be a vertex and let $P = (r = u_0, u_1, \ldots, u_t = x)$ be a shortest path from r to x. Then for every integer j with $0 < j < t$, $(u_0, u_1, \ldots, u_j)$ is a shortest path from r to u_j and $(u_j, u_{j+1}, \ldots, u_t)$ is a shortest path from u_j to u_t*

Proposition 10.6. *When the algorithm halts, let $\sigma = (v_1, v_2, v_3, \ldots, v_n)$. Then $d(v_1) \leq d(v_2) \leq \cdots \leq d(v_n)$.*

We are now ready to prove the correctness of the algorithm.

Theorem 10.7. *Dijkstra's Algorithm yields shortest paths for every vertex x in $\mathbf{G}$, i.e., for each $x \in V$, the value $d(x)$ is the distance from r to x and $P(x)$ is a shortest path from r to x.*

Proof. The theorem holds trivially when $x = r$. So we consider the case where $x \neq r$. We argue that $d(x)$ is the distance from r to x and that $P(x)$ is a shortest path from r to x by induction on the minimum number k of edges in a shortest path from r to x. When $k = 1$, the edge (r, x) is a shortest path from r to x. Since $v_1 = r$, we will set $d(x) = w(r, x)$ at Step 1.

Now assume that for some positive integer k, $d(x)$ is the distance from r to x and $P(x)$ is a shortest path from r to x whenever the minimum number of edges in a shortest path from r to x is at most k.

Then let x be a vertex for which the minimum number of edges in a shortest path from r to x is $k + 1$. Let $P = (u_0, u_1, u_2, \ldots, u_{k+1})$ be a shortest path from $r = u_0$ to $x = u_{k+1}$. Then $Q = (u_0, u_1, \ldots, u_k)$ is a shortest path from r to u_k.

By the inductive hypothesis, $d(u_k)$ is the distance from r to u_k, and $P(u_k)$ is a shortest path from r to u_k. Note that $P(u_k)$ need not be the same as path Q. However, if distinct, the two paths will have the same length, namely $d(u_k)$. Also, the distance from r to x is $d(u_k) + w(u_k, x) \geq d(u_k)$.

Let i and j be the unique integers for which $u_k = v_i$ and $x = v_j$. If $j < i$, then

$$d(x) = d(v_j) \leq d(v_i) = d(u_k) \leq d(u_k) + w(u_k).$$

Therefore the algorithm has found a path $P(x)$ from r to x having length $d(x)$ which is at most the distance from r to x. Clearly, this implies that $d(x)$ is the distance from r to x and that $P(x)$ is a shortest path.

On the other hand, if $j > i$, then the inductive step at Step i results in

$$d(x) \leq d(v_i) + w(v_i, y) = d(u_k) + w(u_k, x).$$

As before, this implies that $d(x)$ is the distance from r to x and that $P(x)$ is a shortest path. □

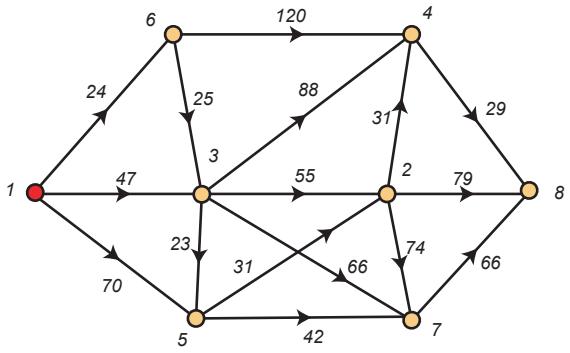

FIGURE 10.3:

Example. Consider the digraph **G** shown in Figure 10.3. For visual clarity, we have chosen a digraph which is an *oriented graph*, i.e., for each distinct pair x, y of vertices, the graph contains at most one of the two possible directed edges (x, y) and (y, x).

For this graph, here are the results of applying Dijkstra's Algorithm with the root vertex r being the vertex labeled 1:

Initialization

$\sigma = (1)$.
$d(1) = 0$; $P(1) = (1)$.
$d(2) = \infty$; $P(2) = (1, 2)$
$d(3) = 47$; $P(3) = (1, 3)$
$d(4) = \infty$; $P(4) = (1, 4)$
$d(5) = 70$; $P(5) = (1, 5)$
$d(6) = 24$; $P(6) = (1, 6)$
$d(7) = \infty$; $P(7) = (1, 6)$
$d(8) = \infty$; $P(8) = (1, 8)$

Step 1. Scan from vertex 6.

$\sigma = (1, 6)$.
$d(1) = 0$; $P(1) = (1)$.
$d(2) = \infty$; $P(2) = (1, 2)$
$d(3) = 47$; $P(3) = (1, 3)$
$d(4) = 144 = 24 + 120 = d(6) + w(6, 4)$; $P(4) = (1, 6, 4)$ updated
$d(5) = 70$; $P(5) = (1, 5)$
$d(6) = 24$; $P(6) = (1, 6)$

$d(7) = \infty$; $P(7) = (1, 6)$
$d(8) = \infty$; $P(8) = (1, 8)$

Step 2. Scan from vertex 3.

$\sigma = (1, 6, 3)$.
$d(1) = 0$; $P(1) = (1)$.
$d(2) = 102 = 47 + 55 = d(3) + w(3, 2)$; $P(2) = (1, 3, 2)$ updated
$d(3) = 47$; $P(3) = (1, 3)$
$d(4) = 135 = 47 + 88 = d(3) + w(3, 4)$; $P(4) = (1, 3, 4)$ updated
$d(5) = 70$; $P(5) = (1, 5)$
$d(6) = 24$; $P(6) = (1, 6)$
$d(7) = 113 = 47 + 66 = d(3) + w(3, 7)$; $P(7) = (1, 3, 7)$ updated
$d(8) = \infty$; $P(8) = (1, 8)$

Step 3. Scan from vertex 5.

$\sigma = (1, 6, 3, 5)$.
$d(1) = 0$; $P(1) = (1)$.
$d(2) = 101 = 70 + 31 = d(5) + w(5, 2)$; $P(2) = (1, 5, 2)$ updated
$d(3) = 47$; $P(3) = (1, 3)$
$d(4) = 135$; $P(4) = (1, 3, 4)$
$d(5) = 70$; $P(5) = (1, 5)$
$d(6) = 24$; $P(6) = (1, 6)$
$d(7) = 112 = 70 + 42 = d(5) + w(5, 7)$; $P(7) = (1, 5, 7)$ updated
$d(8) = \infty$; $P(8) = (1, 8)$

Step 4. Scan from vertex 2.

$\sigma = (1, 6, 3, 5, 2)$.
$d(1) = 0$; $P(1) = (1)$.
$d(2) = 101$; $P(2) = (1, 5, 2)$
$d(3) = 47$; $P(3) = (1, 3)$
$d(4) = 132 = 101 + 31 = d(2) + w(2, 4)$; $P(4) = (1, 5, 2, 4)$ updated
$d(5) = 70$; $P(5) = (1, 5)$
$d(6) = 24$; $P(6) = (1, 6)$
$d(7) = 112$; $P(7) = (1, 5, 7)$
$d(8) = 180 = 101 + 79 = d(2) + w(2, 8)$; $P(8) = (1, 5, 2, 8)$ updated

Step 5. Scan from vertex 7.

$\sigma = (1, 6, 3, 5, 2, 7)$.
$d(1) = 0$; $P(1) = (1)$.
$d(2) = 101$; $P(2) = (1, 5, 2)$
$d(3) = 47$; $P(3) = (1, 3)$
$d(4) = 132$; $P(4) = (1, 5, 2, 4)$
$d(5) = 70$; $P(5) = (1, 5)$
$d(6) = 24$; $P(6) = (1, 6)$
$d(7) = 112$; $P(7) = (1, 5, 7)$
$d(8) = 178 = 112 + 66 = d(7) + w(7, 8)$; $P(8) = (1, 5, 7, 8)$ updated

Step 6. Scan from vertex 4.

$\sigma = (1, 6, 3, 5, 2, 7, 4)$.
$d(1) = 0$; $P(1) = (1)$.
$d(2) = 101$; $P(2) = (1, 5, 2)$
$d(3) = 47$; $P(3) = (1, 3)$
$d(4) = 132$; $P(4) = (1, 5, 2, 4)$
$d(5) = 70$; $P(5) = (1, 5)$
$d(6) = 24$; $P(6) = (1, 6)$
$d(7) = 112$; $P(7) = (1, 5, 7)$
$d(8) = 161 = 132 + 29 = d(4) + w(4, 8)$; $P(8) = (1, 5, 2, 4, 8)$ updated

FINAL RESULTS

$\sigma = (1, 6, 3, 5, 2, 7, 4, 8)$.
$d(1) = 0$; $P(1) = (1)$.
$d(2) = 101$; $P(2) = (1, 5, 2)$
$d(3) = 47$; $P(3) = (1, 3)$
$d(4) = 132$; $P(4) = (1, 5, 2, 4)$
$d(5) = 70$; $P(5) = (1, 5)$
$d(6) = 24$; $P(6) = (1, 6)$
$d(7) = 112$; $P(7) = (1, 5, 7)$
$d(8) = 161$; $P(8) = (1, 5, 2, 4, 8)$