§ 8.8 Graph Algorithms

algorithm *n.* Any peculiar method of computing.
The American College Dictionary

Our study of digraphs and graphs has led us to a number of concrete questions. Given a digraph, what is the length of a shortest path from one vertex to another? If the digraph is weighted, what's the minimum or maximum weight of such a path? Is there any path at all? What are the components of a graph? Does removing an edge increase the number of components? Does a given edge belong to a cycle? Do any edges belong to cycles?

This section describes some algorithms for answering these questions and others, algorithms which can be implemented on computers as well as used to organize hand computations. The algorithms we have chosen are reasonably fast, and their workings are comparatively easy to follow. For a more complete discussion we refer the reader to books on the subject, such as *Data Structures and Algorithms* by Aho, Hopcroft and Ullman.

We concentrate first on digraph problems, and deal later with modifications for the undirected case. As we noted in § 8.3, any min-weight algorithm can be used to get a shortest path-length algorithm simply by giving all edges weight 1. We can use such an algorithm to see if there is any path at all from u to v in G by creating fictitious edges of enormous weights between vertices which are not joined by edges in G. If the min-path from u to v in the enlarged graph has an enormous weight, it must be because no path exists made entirely from edges of G.

The min-weight problem is essentially a question about digraphs without loops or parallel edges, so we limit ourselves to that setting. Hence $E(G) \subseteq V(G) \times V(G)$, and we can describe the digraph with a table of the edge-weight function $W(u, v)$, as we did in § 8.3.

The min-weight algorithms we will consider all begin by looking at paths of length 1, i.e., single edges, and then systematically consider longer and longer paths between vertices. As they proceed, the algorithms find smaller and smaller path weights between vertices, and when they stop the weights are the best possible.

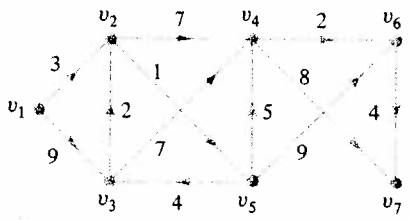
Our first algorithm just finds min-weights from a selected vertex to the other vertices in the digraph G. To describe how it works, it will be convenient to suppose that $V(G) = \{1, \ldots, n\}$ and that 1 is the selected vertex. Starting with 1, the algorithm looks at additional vertices one at a time, always choosing a new vertex w whose known best path weight from 1 is as small as possible, and updating best path weights from 1 to other vertices by considering paths through w. It keeps track of the set of vertices which it has looked at, by putting them in a set L, and it doesn't look at them again. At any given time, $D(j)$ is the smallest weight of a path from 1 to j whose vertices lie in L. Here is the recipe.

Set $L = \{1\}$
For $i = 1$ to n
    Set $D(i) = W(1, i)$
End for
While $V(G)\backslash L \neq \varnothing$
    Choose k in $V(G)\backslash L$ with $D(k)$ as small as possible
    Put k in L
    For each j in $V(G)\backslash L$
        If $D(j) > D(k) + W(k,j)$
            Replace $D(j)$ by $D(k) + W(k,j)$
    End for
End while
End ▮

We have written this algorithm in a style similar to a computer program, rather than in our usual "Step 1, Step 2, ..." format, to make the looping clearer. Each time the algorithm runs through the While loop, it goes back and checks to see if $V(G)\backslash L$ is empty yet. If it is, the algorithm stops. If not, the algorithm goes through the loop again, with a new k.

We need to check that this algorithm stops, and that the final value of $D(j)$ is $W^*(1, j)$ for each j. We also want to get some estimate of the time it takes the algorithm to run. First, though, let's look at how it works.

MPLE 1 Consider the weighted digraph G shown in Figure 1(a). Its edge-weight table is given in Figure 1(b).



(a)

W	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_1	∞	3	9	∞	∞	∞	∞
v_2	∞	∞	∞	7	1	∞	∞
v_3	∞	2	∞	7	∞	∞	∞
v_4	∞	∞	∞	∞	∞	2	8
v_5	∞	∞	4	5	∞	9	∞
v_6	∞	∞	∞	∞	∞	∞	4
v_7	∞	∞	∞	∞	∞	∞	∞

(b)

The table in Figure 2 shows how the values of L and of $D(2), \ldots, D(7)$ change as the algorithm progresses, starting with 1.

Notice that the values in the columns decrease with time, and that once j gets into L the value of $D(j)$ doesn't change. ▮

L	$D(1)$	$D(2)$	$D(3)$	$D(4)$	$D(5)$	$D(6)$	$D(7)$	Comment
$\{1\}$	∞	3	9	∞	∞	∞	∞	Initial data
$\{1, 2\}$	∞	3	9	10	4	∞	∞	Found $v_1v_2v_4$ and $v_1v_2v_5$
$\{1, 2, 5\}$	∞	3	8	9	4	13	∞	Found $v_1v_2v_5v_3$, $v_1v_2v_5v_4$ and $v_1v_2v_5v_6$
$\{1, 2, 5, 3\}$	∞	3	8	9	4	13	∞	No improvement
$\{1, 2, 5, 3, 4\}$	∞	3	8	9	4	11	17	Found $v_1v_2v_5v_4v_6$ and $v_1v_2v_5v_4v_7$
$\{1, 2, 5, 3, 4, 6\}$	∞	3	8	9	4	11	15	Found $v_1v_2v_5v_4v_6v_7$

Figure 2

Theorem 1 If the edge weights in G are nonnegative, then DIJKSTRA'S algorithm stops with $D(j) = W^*(1, j)$ for $j = 2, \ldots, n$.

Proof. Since each pass through the While loop in the algorithm adds one more vertex to L, the algorithm makes $n - 1$ trips through the loop and then stops. For convenience, denote the final value of $D(j)$ by $D^*(j)$. It is not at all obvious that $D^*(j) = W^*(1, j)$, because the algorithm makes "greedy" choices of new vertices whenever it gets a chance. Greed does not always pay—consider trying to get 40 cents out of a pile of dimes and quarters by picking a quarter first—but it works this time.

Notice first that $W^*(1, j) \leqq D(j)$ at all times and for all j, because $D(j) = W(1, j)$ at the start, and any replacement value is the weight of some path from 1 to j. The following lemma displays the key property of DIJKSTRA'S algorithm.

Lemma If j is chosen before k, then $D^*(j) \leqq D^*(k)$.

Proof. Suppose the lemma is false. Among all pairs $\langle j, k \rangle$ with j chosen before k and with $D^*(j) > D^*(k)$, take a pair with the time interval between the two choices as small as possible.

When j is chosen, k is not yet in L, so that $D^*(j) = D(j) \leqq D(k)$. At the end $D^*(k) < D^*(j)$, so there must be an m chosen between j and k for which $D(k)$ is replaced by $D(m) + W(m, k)$ with $D(m) + W(m, k) < D^*(j)$. Then also $D^*(m) \leqq D(m) \leqq D(m) + W(m, k)$, since $W(m, k) \geqq 0$ by the hypothesis of Theorem 1. But this means that $D^*(m) < D^*(j)$ with the time interval between the choices of j and m less than that between j and k, a contradiction. ∎

We return to the proof of Theorem 1. We know that $W^*(1, j) \leqq D^*(j)$ for every j, and we want to show equality holds. Suppose not, and among values of j with $W^*(1, j) < D^*(j)$ choose one with $W^*(1, j)$ as small as possible.

Consider a path $1 \cdots kj$ of weight $W^*(1, j)$. For each vertex m in this path, $W^*(1, m)$ is the weight of the initial segment $1 \cdots m$ by the Proposition of § 8.3. Hence, in particular, $W^*(1, m) \leqq W^*(1, j)$, and we may suppose that j is the first vertex on the path for which $W^*(1, j) < D^*(j)$. If $k = 1$, i.e., if the path has length 1, then $D^*(j) > W^*(1, j) = W(1, j) = D(j)$ at the start, which is absurd. So $k \neq 1$.

Since k precedes j on the path, $W^*(1, k) = D^*(k)$, so

$$D^*(k) \leqq D^*(k) + W(k, j) = W^*(1, k) + W(k, j) = W^*(1, j) < D^*(j).$$

By the lemma, this means that k is chosen before j. So at the time k is chosen it must be true that $W^*(1, k) = D^*(k) = D(k)$. At the end of that pass through the loop,

$$\begin{aligned} W^*(1, j) = W^*(1, k) + W(k, j) &= D(k) + W(k, j) \\ &\geqq D(j) \qquad \text{[by replacement, if needed]} \\ &\geqq D^*(j), \end{aligned}$$

a final contradiction. ∎

How long does DIJKSTRA'S algorithm take for a digraph with n vertices? The largest part of the time is spent going through the While loop, removing one of the original n vertices at each pass. Time to find the smallest $D(k)$ is at worst $O(n)$ if we simply examine the vertices in $V(G)$ one by one, and in fact there are sorting algorithms which will do this faster. For each chosen vertex k there are at most n comparisons and replacements, so the total time for one pass through the loop is at most $O(n)$. All told, the algorithm makes n loops, so it takes total time $O(n^2)$.

If the digraph is presented in terms of successor lists, the algorithm can be rewritten so that the replacement/update step only looks at successors of k. During the total operation, each edge is then considered just once in an update step. Such a modification speeds up the overall performance if $|E(G)|$ is much less than n^2.

DIJKSTRA'S algorithm finds the weights of min-paths from a given vertex. To find $W^*(v_i, v_j)$ for all choices of vertices v_i and v_j we could just apply the algorithm n times, starting from each of the n vertices. There is another algorithm, originally due to Warshall and refined by Floyd, which produces all of the values $W^*(v_i, v_j)$ at the end, and which is easy to program. Like DIJKSTRA'S algorithm, it builds an expanding list of examined vertices and looks at paths through vertices on the list.

Suppose that $V(G) = \{v_1, \ldots, v_n\}$. WARSHALL'S algorithm works with an $n \times n$ matrix $\mathbf{W}$, which at the beginning is the edge-weight matrix $\mathbf{W}_0$ with $\mathbf{W}_0[i, j] = W(v_i, v_j)$ for all i and j, and at the end is the min-weight matrix $\mathbf{W}_n = \mathbf{W}^*$ with $\mathbf{W}^*[i, j] = W^*(v_i, v_j)$.