7. [W. Miller] Show that BGraph1 computes shortest paths for directed acyclic graphs represented by adjacency lists (instead of inverse adjacency lists as in BGraph).

```
1   Algorithm BGraph1(G, n)
2   {
3       bcost[1] := 0.0;
4       for j := 2 to n do bcost[j] := ∞;
5       for j := 1 to n − 1 do
6           for each r such that ⟨j, r⟩ is an edge of G do
7               bcost[r] := min(bcost[r], bcost[j] + c[j, r]);
8   }
```

Note: There is a possibility of a floating point overflow in this function. In such cases the program should be suitably modified.

5.3 ALL-PAIRS SHORTEST PATHS

Let $G = (V, E)$ be a directed graph with n vertices. Let *cost* be a cost adjacency matrix for G such that $cost(i, i) = 0$, $1 \leq i \leq n$. Then $cost(i, j)$ is the length (or cost) of edge $\langle i, j \rangle$ if $\langle i, j \rangle \in E(G)$ and $cost(i, j) = \infty$ if $i \neq j$ and $\langle i, j \rangle \notin E(G)$. The *all-pairs shortest-path problem* is to determine a matrix A such that $A(i, j)$ is the length of a shortest path from i to j. The matrix A can be obtained by solving n single-source problems using the algorithm ShortestPaths of Section 4.8. Since each application of this procedure requires $O(n^2)$ time, the matrix A can be obtained in $O(n^3)$ time. We obtain an alternate $O(n^3)$ solution to this problem using the principle of optimality. Our alternate solution requires a weaker restriction on edge costs than required by ShortestPaths. Rather than require $cost(i, j) \geq 0$, for every edge $\langle i, j \rangle$, we only require that G have no cycles with negative length. Note that if we allow G to contain a cycle of negative length, then the shortest path between any two vertices on this cycle has length $-\infty$.

Let us examine a shortest i to j path in G, $i \neq j$. This path originates at vertex i and goes through some intermediate vertices (possibly none) and terminates at vertex j. We can assume that this path contains no cycles for if there is a cycle, then this can be deleted without increasing the path length (no cycle has negative length). If k is an intermediate vertex on this shortest path, then the subpaths from i to k and from k to j must be shortest paths from i to k and k to j, respectively. Otherwise, the i to j path is not of minimum length. So, the principle of optimality holds. This alerts us to the prospect of using dynamic programming. If k is the intermediate vertex with highest index, then the i to k path is a shortest i to k path in G going through no vertex with index greater than $k - 1$. Similarly the k to j path is a shortest k to j path in G going through no vertex of index greater than

$k-1$. We can regard the construction of a shortest i to j path as first requiring a decision as to which is the highest indexed intermediate vertex k. Once this decision has been made, we need to find two shortest paths, one from i to k and the other from k to j. Neither of these may go through a vertex with index greater than $k-1$. Using $A^k(i,j)$ to represent the length of a shortest path from i to j going through no vertex of index greater than k, we obtain

$$A(i,j) = \min \{ \min_{1 \le k \le n} \{A^{k-1}(i,k) + A^{k-1}(k,j)\}, cost(i,j)\} \tag{5.7}$$

Clearly, $A^0(i,j) = cost(i,j)$, $1 \le i \le n$, $1 \le j \le n$. We can obtain a recurrence for $A^k(i,j)$ using an argument similar to that used before. A shortest path from i to j going through no vertex higher than k either goes through vertex k or it does not. If it does, $A^k(i,j) = A^{k-1}(i,k) + A^{k-1}(k,j)$. If it does not, then no intermediate vertex has index greater than $k-1$. Hence $A^k(i,j) = A^{k-1}(i,j)$. Combining, we get

$$A^k(i,j) = \min \{A^{k-1}(i,j),\ A^{k-1}(i,k) + A^{k-1}(k,j)\},\quad k \ge 1 \tag{5.8}$$

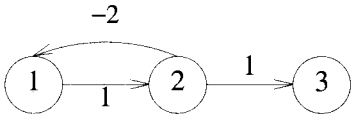
The following example shows that (5.8) is not true for graphs with cycles of negative length.

Example 5.14 Figure 5.5 shows a digraph together with its matrix A^0. For this graph $A^2(1,3) \neq \min\{A^1(1,3),\ A^1(1,2) + A^1(2,3)\} = 2$. Instead we see that $A^2(1,3) = -\infty$. The length of the path

$$1, 2, 1, 2, 1, 2, \ldots, 1, 2, 3$$

can be made arbitrarily small. This is so because of the presence of the cycle 1 2 1 which has a length of -1. □

Recurrence (5.8) can be solved for A^n by first computing A^1, then A^2, then A^3, and so on. Since there is no vertex in G with index greater than n, $A(i,j) = A^n(i,j)$. Function AllPaths computes $A^n(i,j)$. The computation is done inplace so the superscript on A is not needed. The reason this computation can be carried out in-place is that $A^k(i,k) = A^{k-1}(i,k)$ and $A^k(k,j) = A^{k-1}(k,j)$. Hence, when A^k is formed, the kth column and row do not change. Consequently, when $A^k(i,j)$ is computed in line 11 of Algorithm 5.3, $A(i,k) = A^{k-1}(i,k) = A^k(i,k)$ and $A(k,j) = A^{k-1}(k,j) = A^k(k,j)$. So, the old values on which the new values are based do not change on this iteration.



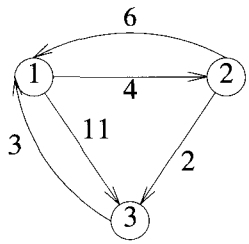
$$\begin{bmatrix} 0 & 1 & \infty \\ -2 & 0 & 1 \\ \infty & \infty & 0 \end{bmatrix}$$

Figure 5.5 Graph with negative cycle

```
0   Algorithm AllPaths(cost, A, n)
1   // cost[1 : n, 1 : n] is the cost adjacency matrix of a graph with
2   // n vertices; A[i, j] is the cost of a shortest path from vertex
3   // i to vertex j. cost[i, i] = 0.0, for 1 ≤ i ≤ n.
4   {
5       for i := 1 to n do
6           for j := 1 to n do
7               A[i, j] := cost[i, j]; // Copy cost into A.
8       for k := 1 to n do
9           for i := 1 to n do
10              for j := 1 to n do
11                  A[i, j] := min(A[i, j], A[i, k] + A[k, j]);
12  }
```

Algorithm 5.3 Function to compute lengths of shortest paths

Example 5.15 The graph of Figure 5.6(a) has the cost matrix of Figure 5.6(b). The initial A matrix, $A^{(0)}$, plus its values after 3 iterations $A^{(1)}$, $A^{(2)}$, and $A^{(3)}$ are given in Figure 5.6. □



(a) Example digraph

A^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

(b) A^0

A^1	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

(c) A^1

A^2	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

(d) A^2

A^3	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

(e) A^3

Figure 5.6 Directed graph and associated matrices

Let $M = \max\{cost(i,j) | \langle i,j \rangle \in E(G)\}$. It is easy to see that $A^n(ij) \leq (n-1)M$. From the working of AllPaths, it is clear that if $\langle i,j \rangle \notin E(G)$ and $i \neq j$, then we can initialize $cost(i,j)$ to any number greater than $(n-1)M$ (rather than the maximum allowable floating point number). If, at termination, $A(i,j) > (n-1)M$, then there is no directed path from i to j in G. Even for this choice of ∞, care should be taken to avoid any floating point overflows.

The time needed by AllPaths (Algorithm 5.3) is especially easy to determine because the looping is independent of the data in the matrix A. Line 11 is iterated n^3 times, and so the time for AllPaths is $\Theta(n^3)$. An exercise examines the extensions needed to obtain the i to j paths with these lengths. Some speedup can be obtained by noticing that the innermost **for** loop need be executed only when $A(i,k)$ and $A(k,j)$ are not equal to ∞.

EXERCISES

1. (a) Does the recurrence (5.8) hold for the graph of Figure 5.7? Why?

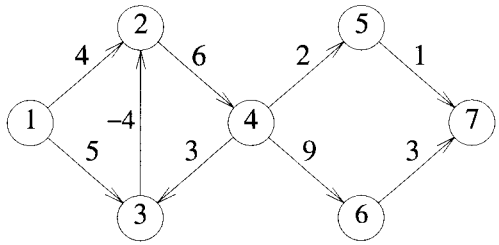


Figure 5.7 Graph for Exercise 1

   (b) Why does Equation 5.8 not hold for graphs with cycles of negative length?

2. Modify the function AllPaths so that a shortest path is output for each pair of vertices (i, j). What are the time and space complexities of the new algorithm?

3. Let A be the adjacency matrix of a directed graph G. Define the transitive closure A^+ of A to be a matrix with the property $A^+(i,j) = 1$ iff G has a directed path, containing at least one edge, from vertex i to vertex j. $A^+(i,j) = 0$ otherwise. The reflexive transitive closure A^* is a matrix with the property $A^*(i,j) = 1$ iff G has a path, containing zero or more edges, from i to j. $A^*(i,j) = 0$ otherwise.

   (a) Obtain A^+ and A^* for the directed graph of Figure 5.8.

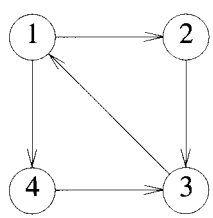


Figure 5.8 Graph for Exercise 3

   (b) Let $A^k(i,j) = 1$ iff there is a path with zero or more edges from i to j going through no vertex of index greater than k. Define A^0 in terms of the adjacency matrix A.