14.5 All-Pairs Shortest Paths

Suppose we wish to compute the shortest-path distance between every pair of vertices in a directed graph $\vec{G}$ with n vertices and m edges. Of course, if $\vec{G}$ has no negative-weight edges, then we could run Dijkstra's algorithm from each vertex in $\vec{G}$ in turn. This approach would take $O(n(n+m)\log n)$ time, assuming $\vec{G}$ is represented using an adjacency list structure. In the worst case, this bound could be as large as $O(n^3 \log n)$. Likewise, if $\vec{G}$ contains no negative-weight cycles, then we could run the Bellman-Ford algorithm starting from each vertex in $\vec{G}$ in turn. This approach would run in $O(n^2 m)$ time, which, in the worst case, could be as large as $O(n^4)$. In this section, we consider algorithms for solving the all-pairs shortest path problem in $O(n^3)$ time, even if the digraph contains negative-weight edges (but not negative-weight cycles).

14.5.1 A Dynamic Programming Shortest-Path Algorithm

The first all-pairs shortest-path algorithm we discuss is a variation on an algorithm we have given earlier in this book, namely, the Floyd-Warshall algorithm for computing the transitive closure of a directed graph (Algorithm 13.13).

Let $\vec{G}$ be a given weighted directed graph. We number the vertices of $\vec{G}$ arbitrarily as $(v_1, v_2, \ldots, v_n)$. As in any dynamic programming algorithm (Chapter 12), the key construct in the algorithm is to define a parametrized cost function that is easy to compute and also allows us to ultimately compute a final solution. In this case, we use the cost function, $D^k_{i,j}$, which is defined as the distance from v_i to v_j using only intermediate vertices in the set $\{v_1, v_2, \ldots, v_k\}$. Initially,

$$D^0_{i,j} = \begin{cases} 0 & \text{if } i = j \\ w((v_i, v_j)) & \text{if } (v_i, v_j) \text{ is an edge in } \vec{G} \\ +\infty & \text{otherwise.} \end{cases}$$

Given this parametrized cost function $D^k_{i,j}$, and its initial value $D^0_{i,j}$, we can then easily define the value for an arbitrary $k > 0$ as

$$D^k_{i,j} = \min\{D^{k-1}_{i,j}, D^{k-1}_{i,k} + D^{k-1}_{k,j}\}.$$

In other words, the cost for going from v_i to v_j using vertices numbered 1 through k is equal to the shorter of two possible paths. The first path is simply the shortest path from v_i to v_j using vertices numbered 1 through $k-1$. The second path is the sum of the costs of the shortest path from v_i to v_k using vertices numbered 1 through $k-1$ and the shortest path from v_k to v_j using vertices numbered 1 through $k-1$. Moreover, there is no other shorter path from v_i to v_j using vertices of $\{v_1, v_2, \ldots, v_k\}$ than these two. If there was such a shorter path and it excluded v_k, then it would violate the definition of $D^{k-1}_{i,j}$, and if there was such a shorter

Algorithm AllPairsShortestPaths($\vec{G}$):

Input: A simple weighted directed graph $\vec{G}$ without negative-weight cycles

Output: A numbering $v_1, v_2, \ldots, v_n$ of the vertices of $\vec{G}$ and a matrix D, such that $D[i, j]$ is the distance from v_i to v_j in $\vec{G}$

let $v_1, v_2, \ldots, v_n$ be an arbitrary numbering of the vertices of $\vec{G}$
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** n **do**
    if $i = j$ **then**
      $D^0[i, i] \leftarrow 0$
    if (v_i, v_j) is an edge in $\vec{G}$ **then**
      $D^0[i, j] \leftarrow w((v_i, v_j))$
    else
      $D^0[i, j] \leftarrow +\infty$
for $k \leftarrow 1$ to n **do**
  for $i \leftarrow 1$ to n **do**
    for $j \leftarrow 1$ to n **do**
      $D^k[i, j] \leftarrow \min\{D^{k-1}[i, j], D^{k-1}[i, k] + D^{k-1}[k, j]\}$
return matrix D^n

Algorithm 14.11: A dynamic programming algorithm to compute all-pairs shortest-path distances in a digraph without negative cycles.

path and it included v_k, then it would violate the definition of $D^{k-1}_{i,k}$ or $D^{k-1}_{k,j}$. In fact, note that this argument still holds even if there are negative cost edges in $\vec{G}$, just so long as there are no negative cost cycles. In Algorithm 14.11, we show how this cost-function definition allows us to build an efficient solution to the all-pairs shortest path problem. The running time for this dynamic programming algorithm is clearly $O(n^3)$, which implies the following.

Theorem 14.6: *Given a simple weighted directed graph $\vec{G}$ with n vertices and no negative-weight cycles, Algorithm 14.11 (*AllPairsShortestPaths*) computes the shortest-path distances between each pair of vertices of $\vec{G}$ in $O(n^3)$ time.*

14.5.2 Computing Shortest Paths via Matrix Multiplication

We can view the problem of computing the shortest-path distances for all pairs of vertices in a directed graph $\vec{G}$ as a matrix problem. In this subsection, we describe how to solve the all-pairs shortest-path problem in $O(n^3)$ time using this approach. We first describe how to use this approach to solve the all-pairs problem in $O(n^4)$ time, and then we show how this can be improved to $O(n^3)$ time by studying the problem in more depth. This matrix-multiplication approach to shortest paths is especially useful in contexts where we represent graphs using the adjacency matrix data structure.