All Pairs Shortest Algorithm

All-Pairs Shortest Path Problem

Suppose we are given a directed graph G=(V,E) and a weight function w: E->R.

We assume that G does not contain cycles of weight 0 or less.

The All-Pairs Shortest Path Problem asks to find the length of the shortest path between any pair of vertices in G.

Quick Solutions

If the weight function is nonnegative for all edges, then we can use Dijkstra's single source shortest path algorithm for all vertices to solve the problem.

This yields an $O(n^3)$ algorithm on graphs with n vertices (on dense graphs and with a simple implementation).

3

Quick Solution

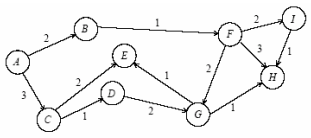
For arbitrary weight functions, we can use the Bellman-Ford algorithm applied to all vertices. This yields an $O(n^4)$ algorithm for graphs with n vertices.

4

Matrix-Multiplication Based Algorithm

- Consider the multiplication of the weighted adjacency matrix with itself - except, in this case, we replace the multiplication operation in matrix multiplication by addition, and the addition operation by minimization.
- Notice that the product of weighted adjacency matrix with itself returns a matrix that contains shortest paths of length 2 between any pair of nodes.
- It follows from this argument that A^n contains all shortest paths.

Matrix-Multiplication Based Algorithm



$$A^1 = \begin{pmatrix} 0 & 2 & 3 & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty & 1 & \infty & \infty & \infty \\ \infty & \infty & 0 & 1 & 2 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 2 & 3 & 2 \\ \infty & \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & 0 \end{pmatrix} \quad A^2 = \begin{pmatrix} 0 & 2 & 3 & 4 & 5 & 3 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty & 1 & 3 & 4 & 3 \\ \infty & \infty & 0 & 1 & 2 & \infty & 3 & \infty & \infty \\ \infty & \infty & \infty & 0 & 3 & \infty & 2 & 3 & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 3 & 0 & 2 & 3 & 2 \\ \infty & \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & 0 \end{pmatrix}$$

$$A^4 = \begin{pmatrix} 0 & 2 & 3 & 4 & 5 & 3 & 5 & 6 & 5 \\ \infty & 0 & \infty & \infty & 4 & 1 & 3 & 4 & 3 \\ \infty & \infty & 0 & 1 & 2 & \infty & 3 & 4 & \infty \\ \infty & \infty & \infty & 0 & 3 & \infty & 2 & 3 & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 3 & 0 & 2 & 3 & 2 \\ \infty & \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & 0 \end{pmatrix} \quad A^8 = \begin{pmatrix} 0 & 2 & 3 & 4 & 5 & 3 & 5 & 6 & 5 \\ \infty & 0 & \infty & \infty & 4 & 1 & 3 & 4 & 3 \\ \infty & \infty & 0 & 1 & 2 & \infty & 3 & 4 & \infty \\ \infty & \infty & \infty & 0 & 3 & \infty & 2 & 3 & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 3 & 0 & 2 & 3 & 2 \\ \infty & \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & 0 \end{pmatrix}$$

Matrix-Multiplication Based Algorithm

- A^n is computed by doubling powers - i.e., as A, A^2, A^4, A^8, and so on.
- We need *log n* matrix multiplications, each taking time $O(n^3)$.
- The serial complexity of this procedure is $O(n^3 \log n)$.
- This algorithm is not optimal, since the best known algorithms have complexity $O(n^3)$.

Floyd-Warshall

We will now investigate a dynamic programming solution that solved the problem in $O(n^3)$ time for a graph with n vertices.

This algorithm is known as the Floyd-Warshall algorithm, but it was apparently described earlier by Roy.

8

Representation of the Input

We assume that the input is represented by a weight matrix $W = (w_{ij})_{i,j \text{ in } E}$ that is defined by

$w_{ij} = 0$ if $i=j$

$w_{ij} = w(i,j)$ if $i \neq j$ and (i,j) in E

$w_{ij} = \infty$ if $i \neq j$ and (i,j) not in E

9

Format of the Output

If the graph has n vertices, we return a distance matrix (d_{ij}), where d_{ij} the length of the path from i to j.

10

Intermediate Vertices

Without loss of generality, we will assume that V={1,2,…,n}, i.e., that the vertices of the graph are numbered from 1 to n.

Given a path $p=(v_1, v_2,\ldots, v_m)$ in the graph, we will call the vertices v_k with index k in {2,…,m-1} the intermediate vertices of p.

11

Key Definition

The key to the Floyd-Warshall algorithm is the following definition:

Let $d_{ij}^{(k)}$ denote the length of the shortest path from i to j such that all intermediate vertices are contained in the set {1,…,k}.

12

Remark 1

A shortest path does not contain any vertex twice, as this would imply that the path contains a cycle. By assumption, cycles in the graph have a positive weight, so removing the cycle would result in a shorter path, which is impossible.

13

Remark 2

Consider a shortest path p from i to j such that the intermediate vertices are from the set {1,…,k}.

- If the vertex k is not an intermediate vertex on p, then $d_{ij}^{(k)} = d_{ij}^{(k-1)}$

If the vertex k is an intermediate vertex on p, then $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

Interestingly, in either case, the subpaths contain merely nodes from {1,…,k-1}.

14

Remark 2

Therefore, we can conclude that

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$$

15

Recursive Formulation

If we do not use intermediate nodes, i.e., when k=0, then

$$d_{ij}^{(0)} = w_{ij}$$

If k>0, then

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$$

16

The Floyd-Warshall Algorithm

```
Floyd-Warshall(W)
n = # of rows of W;
D(0) = W;
for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            d_ij^(k) = min{d_ij^(k-1) , d_ik^(k-1) + d_kj^(k-1)};
        od;
    od;
od;
 return D(n);
```

17

Time and Space Requirements

The running time is obviously $O(n^3)$.

However, in this version, the space requirements are high. One can reduce the space from $O(n^3)$ to $O(n^2)$ by using a single array d.

18

Running Times

- Matrix-multiplication based algorithm
  - $O(V^3 \log V)$
    - log V executions of "matrix-matrix" multiplication
      - Not quite matrix-matrix multiplication but same running time
- Floyd-Warshall algorithm
  - $O(V^3)$
    - V iterations of an $O(V^2)$ update loop
    - The constant is very small, so this is a "fast" $O(V^3)$

The End