Floyd's Algorithm (Dynamic Programming)

The All-Pairs Shortest Path Problem

We have seen Dijkstra's algorithm for solving the single-source shortest path problem, which computes all the shortest paths in a graph from a single starting vertex s. This algorithm, on dense graphs, is essentially O(|V|^2). If we needed the shortest path between every distinct pair of vertices in a graph, then we could run Dijkstra's algorithm |V| times (once for each vertex as the source) to obtain the solution in O(|V|^3) time.

There is a dynamic programming solution that is also O(|V|^3). However, because of its innate efficiency, this algorithm tends to outperform multiple iterations of Dijkstra's algorithm. This algorithm is known as Floyd's algorithm, is named for its designer and was developed in 1962.

Data Structure: Again, we'll use matrices, in particular a series of matrices $F^{(0)}, F^{(1)}, ..., F^{(n)}$ where $F^{(0)}$ is the "weight matrix of the graph G (with n vertices). So $f_{ij}^{(0)}$ contains a 0 if $i = j$, has the value ∞ if i and j are not connected by an edge, and contains the value cost(i , j) if i and j are adjacent.

Notation: For an entry of $F^{(k)}$, $f_{ij}^{(k)}$ denotes the shortest path from i to j using intermediate vertices only from the set $\{v_1, v_2, ... v_k\}$

What we need to compute: $F^{(n)}$, because each entry $f_{ij}^{(n)}$ will hold the length of the shortest path from i to j with no restriction on the intermediate vertices. That is, $f_{ij}^{(n)}$ will hold the length of the shortest path in G from i to j.

The key observation: In computing paths from i to j using vertices numbered k or less, all such paths can be divided into two classes:

1. Paths that do not use vertex k. In this case the shortest path in this group will be given by $f_{ij}^{(k-1)}$
2. Paths that do use vertex k. Note that such paths will not use vertex k more than once as an intermediate vertex, because otherwise the path would contain a cycle, which could be removed to shorten the path. So the path must look like this:

   $v_i ... v_k ... v_j$ where every vertex in the portions represented by … are labeled with subscripts less than k. Now, since the shortest path problem has the optimal

substructure property, the shortest path in this second class of paths will be equal to
the length of the shortest path from v_i to v_k (that uses vertices labeled less than k)
+
the length of the shortest path from v_k to v_j (that uses vertices labeled less than k)

But these values are $f_{ik}^{(k-1)}$ and $f_{kj}^{(k-1)}$ respectively.

So the recurrence relation that we seek is:

$$f_{ij}^{(k)} = \min(f_{ij}^{(k-1)}, f_{ik}^{(k-1)} + f_{kj}^{(k-1)}), k \geq 1 \text{ with } f_{ij}^{(0)} = weight(i,j)$$

Pseudo Code:

```
void Floyd(int[ ] [ ] F) // F is initialized as the "weight" matrix
 {
   for (k=1; k<=n; ++k)
     for (i=1; i<=n; ++i)
       for (j=1; j<=n; ++j)
         F[i,j] = min(F[i,j], F[i,k]+F[k,j])  //note:  we're overwriting the matrix
                                                   //this is ok
}
```

Complexity: The above algorithm is easily seen to be $\Theta(n^3)$.

The obvious modification: In addition to knowing the length of all the paths, it would be nice to be able to recover the paths themselves, if necessary.

Why can we overwrite the matrix? (That is, why do we only need one matrix?) To compute the elements of $F^{(i)}$, we use only the current values and the i^{th} row and column of the "previous" matrix. But these values don't change.

An example: (the values of the various matrices are given to demonstrate the execution of the algorithm. Dry run the code and look in detail at the computation of some of the matrix entries.

$$F^{(0)} = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix}$$

$$F^{(1)} = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix}$$

$$F^{(2)} = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ 9 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix}$$

$$F^{(3)} = \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 9 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix}$$

$$F^{(4)} = \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix}$$