All Pairs Shortest Path (APSP) Problem

Given a directed graph G = (V,E), where each edge (v,w) has a nonnegative cost C[v,w], for all pairs of vertices (v,w) find the cost of the lowest cost path from v to w.

- A generalization of the single-source-shortest-path problem.
- Use Dijkstra's algorithm, varying the source node among all the nodes in the graph.

We will consider a slight extension to this problem: find the **lowest cost path** between each pair of vertices.

- We must recover the path itself, and not just the cost of the path.

Floyd's Algorithm

Floyd's algorithm takes as input the cost matrix C[v,w]

- C[v,w] = oo if (v,w) is not in E

It returns as output

- a distance matrix D[v,w] containing the cost of the lowest cost path from v to w
    - initially D[v,w] = C[v,w]
- a path matrix P, where P[v,w] holds the intermediate vertex k on the least cost path between v and w that led to the cost stored in D[v,w].

We iterate N times over the matrix D, using k as an index. On the kth iteration, the D matrix contains the solution to the APSP problem, where the paths only use vertices numbered 1 to k.

On the next iteration, we compare the cost of going from i to j using only vertices numbered 1..k (stored in D[i,j] on the kth iteration) with the cost of using the k+1th vertex as an intermediate step, which is D[i,k+1] (to get from i to k+1) plus D[k+1,j] (to get from k+1 to j).

If this results in a lower cost path, we remember it.

After N iterations, all possible paths have been examined, so D[v,w] contains the cost of the lowest cost path from v to w using all vertices if necessary.

The Algorithm

```
FloydAPSP (int N, rmatrix &C, rmatrix &D, imatrix &P)
{
  int i,j,k;
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
       D[i][j] = C[i][j];
       P[i][j] = -1;
    }
    D[i][i] = 0.0;
  }
  for (k = 0; k < N; k++) {
    for (i = 0; i < N; i++) {
      for (j = 0; j < N; j++) {
        if (D[i][k] + D[k][j] < D[i][j]) {
          D[i][j] = D[i][k] + D[k][j];
          P[i][j] = k;
```

```
        } } } }
  } /* FloydAPSP */
```

Clearly the algorithm is O(N^3).

Finding a Least Cost Path

Floyd's algorithm (modified to find the least cost paths, and not just the cost of the paths) produces a matrix P, which, for each pair of nodes u and v, contains an intermediate node on the least cost path from u to v

So the least cost path from u to v is the least cost path from u to P[u,v], followed by the least cost path from P[u,v] to v.

The following procedure uses the P matrix produced earlier to print the intermediate vertices on the least cost path from node u to node v.

```
Path (int u, int v, imatrix &P)
{
  int k;

  k = P[u][v];
  if (k == -1) return;
  path(u,k);
  cout << k;
  path(k,v);

} /* Path */
```

Note that this procedure could loop forever on an arbitrary matrix, but Floyd's algorithm ensures that we cannot have k on the shortest path from u to v **and** v on the shortest path from u to k.

Proof that Floyd's Algorithm Works

We will prove that after k iterations over the matrix D, D[i,j] is the cost of the cheapest path from i to j that does not include a vertex numbered > k.

Proof by induction on k.

Basis: Let k = 0 (ie, no iterations yet performed).

Then no intermediate vertices on a path from i to j are allowed, so D[i,j] should be C[i,j] if (i,j) in E, and infinity otherwise. The initialization step does exactly this.

Induction step: Assume that after k iterations, D[i,j] is the cost of the lowest cost path from i to j excluding all vertices from k+1 to N.

On the next (k+1) iteration, we are allowed to include vertex k+1 in any path.

For all pairs (i,j), the lowest cost path from i to j excluding vertices k+2 thru N goes thru k+1 iff there is a low cost path from i to k+1 and from k+1 to j, excluding vertices k+2 thru N.

But the cheapest path from i to k+1 without using nodes k+2 thru N is simply D[i,k+1] (by the induction hypothesis).

Similarly, the lowest cost path from k+1 to j without using nodes k+2 thru N is D[k+1,j].

Thus, we should use node k+1 to get from i to j iff D[i,k+1] + D[k+1,j] < D[i,j], the cheapest path excluding k+1. Since this is exactly what is stored on the k+1th iteration, we have completed the proof.

Comparison with Dijkstra's Algorithm

The all-pairs-shortest-path problem is generalization of the single-source-shortest-path problem, so we can use Floyd's algorithm, or Dijkstra's algorithm (varying the source node over all nodes).

- Floyd's algorithm is O(N^3)
- Dijkstra's algorithm with an adjacency matrix is O(N^2), so varying over N source nodes is O(N^3)
- Dijkstra's algorithm with adjacency lists is O(E log N), so varying over N source nodes is O(N E log N)

For large sparse graphs, Dijkstra's algorithm is preferable.