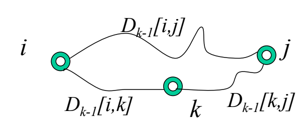
Floyd-Warshall's algorithm

- All pair shortest path problem
  - Given a directed graph G=<V, E>, find the shortest path between any pair of nodes.
- Data structure
  - The nodes of G are numbered from 1 to N.
  - A matrix W given the length/weight of each edge
    - W[i,i] = 0
    - W[i,j] = ∞ if the edge (i,j) does not exist.

The key idea

- The principle of optimality
  - If *k* is a node on the shortest path from *i* to *j*, then the part of the path from *i* to *k* and the part from *k* to *j* must also be optimal
- Construct a series of matrices of D_k, $k=1,2,..,n$, where $D_k[i,j]$ is the distance of the shortest path from *i* to *j* that only use node $\{1, 2,..., k\}$ as intermediate nodes.
  - $D_k[i,j] = \min(D_{k-1}[i,j],\ D_{k-1}[i,k] + D_{k-1}[k,j])$



Floyd's algorithm

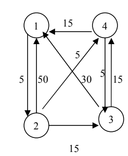
```
Floyd(int W[1..n][1..n])
{
   D = W;
   π = 0;
   for (k=1; k<=n; k++)
     for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
           if (D[i][k]+D[k][j] < D[i][j]) {
              D[i][j] = D[i][k]+D[k][j];
              π[i][j] = k;
           }
}
```

Track the shortest path

Cost: $\Theta(V^3)$

Why can we use just one array D?

Example



$$D_0 = W = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix}$$

$$D_1 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \underline{35} & 0 & 15 \\ 15 & \underline{20} & 5 & 0 \end{pmatrix} \quad D_2 = \begin{pmatrix} 0 & 5 & \underline{20} & \underline{10} \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \quad D_3 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ \underline{45} & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

$$D_4 = \begin{pmatrix} 0 & 5 & \underline{15} & 10 \\ \underline{20} & 0 & \underline{10} & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \quad \pi = \begin{pmatrix} 0 & 0 & 4 & 2 \\ 4 & 0 & 4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Find the shortest path

```
// Print shortest path between
  nodes i and j
printShortestPath(int i, int j)
{
   print i;
   printIntermediateNodes(i, j);
   print j;
}
```

```
// Print intermediate nodes in the shortest
   path between nodes i and j
printIntermediateNodes(int i, int j)
{
   int k = p[i][j];
   if (k == 0)
      return;
   else {
      printIntermediateNodes(i, k);
      print k;
      printIntermediateNodes(k, j);
   }
}
```

Find the shortest path between nodes 1 and 3 for the example.

Comparison

- Applying Dijkstra's algorithm on every node
  - If use a matrix of distances, $V*\Theta(V^2)= \Theta(V^3)$. Same asymptotic cost, but higher hidden constant
  - If use a heap, $V*\Theta(E\log V)= \Theta(VE\log V)$.
    - If E is close to V^2, Floyd's algorithm is better
    - If $E << V^2$, it is preferable to use Dijkstra's algorithm n times