CS 2009 – Design and Analysis of Algorithms

Shortest Path Problem – II

Rizwan Ul Haq

All-Pairs Shortest Paths Problem

- Given a weighted digraph G=(V,E),determine the length of the shortest path (i.e., distance) between all pairs of vertices in G.

- It aims to compute shortest path from each vertex v to every other vertex u.

Floyd-Warshall algorithm

- Given a weighted graph, we want to know the shortest path from one vertex in the graph to another.
  - The Floyd-Warshall algorithm determines the shortest path between all pairs of vertices in a graph, if there is no negative cycle.

Main idea

- a path exists between two vertices i, j, if
  - there is an edge from i to j; or
  - there is a path from i to j going through intermediate vertices $\{1, \ldots\ldots k\}$;

- These are two situations:
  1) k is an intermediate vertex on the shortest path.
  2) k is not an intermediate vertex on the shortest path.

Matrix Representation

- The graph is represented by an n x n matrix with the weights of the edges
- **Output Format:** an n x n distance $D=[\ d_{i,j}\ ]$ where $d_{i,j}$ is the distance from vertex i to j.

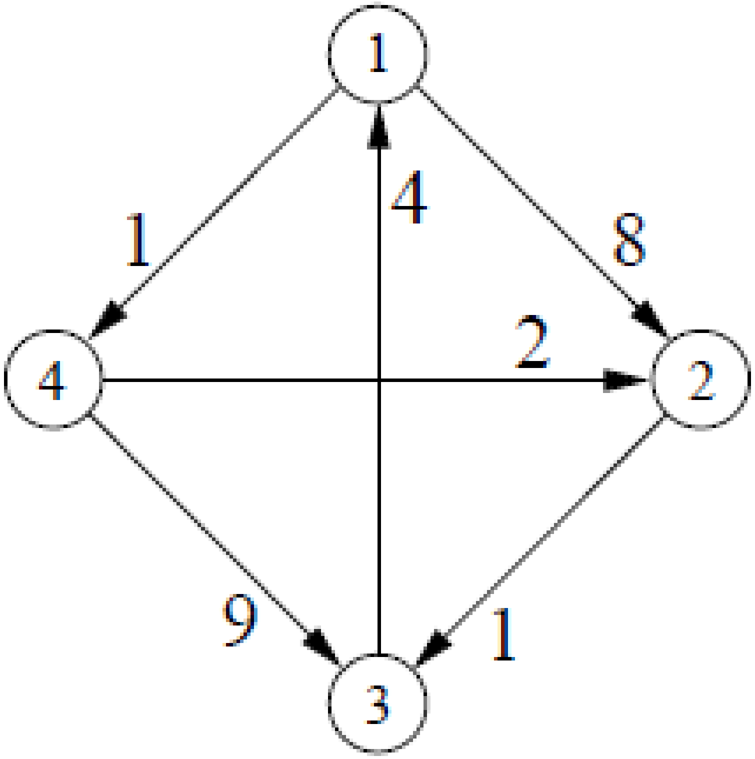
Floyd Warshall Algorithm

```
Floyd_Warshall (W) {
for i = 1 to n do { // initialize
    for j = 1 to n do {
        d[i,j] = W[i,j]
        pred[i,j] = null
                }
                    }
for k = 1 to n do                                   // use intermediates {1..k}
    for i = 1 to n do                   // ...from i
        for j = 1 to n do                           // ...to j
            if (d[i,k] + d[k,j]) < d[i,j]) {
                d[i,j] = d[i,k] + d[k,j]    // new shorter path length
                pred[i,j] = k   }           // new path is through k
                return d                    // matrix of final distances
                    }
```

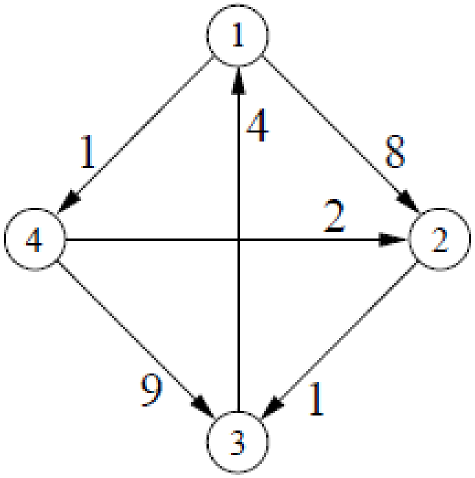
Compute shortest path using predecessor information

- The pred[i, j] can be used to extract the final path.
- Here is the idea, whenever we discover that the shortest path from i to j passes through an intermediate vertex k, we set pred[i,j] = k. If the shortest path does not pass through any intermediate vertex, then pred[i, j] = null.
- To find the shortest path from i to j, we consult pred[i,j]. If it is null, then the shortest path is just the edge (i, j). Otherwise, we recursively compute the shortest path from i to pred[i, j] and the shortest path from pred[i, j] to j.

Example



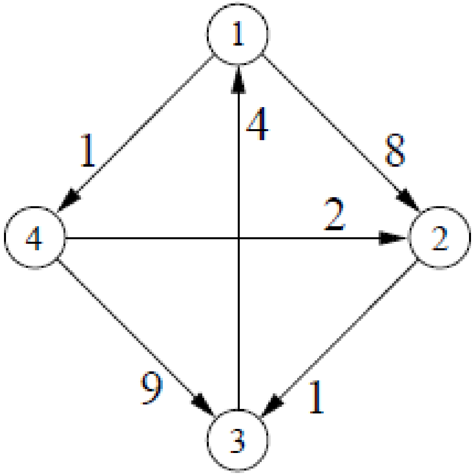
Initially



$$D^{(0)} = \begin{bmatrix} 0 & 8 & ? & 1 \\ ? & 0 & 1 & ? \\ 4 & ? & 0 & ? \\ ? & 2 & 9 & 0 \end{bmatrix} \quad P^{(0)} = \begin{pmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{pmatrix}$$

? = infinity

Ist Iteration: k=1



$$D^{(0)} = \begin{bmatrix} 0 & 8 & ? & 1 \\ ? & 0 & 1 & ? \\ 4 & ? & 0 & ? \\ ? & 2 & 9 & 0 \end{bmatrix} \quad \mathbf{P^{(0)}} = \begin{pmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{pmatrix}$$

? = infinity

```
if (d[i,k] + d[k,j]) < d[i,j]) {
    d[i,j] = d[i,k] + d[k,j]
    pred[i,j] = k   }
```

$$D^{(1)} = \begin{bmatrix} 0 & 8 & ? & 1 \\ ? & 0 & 1 & ? \\ 4 & 12 & 0 & 5 \\ ? & 2 & 9 & 0 \end{bmatrix} \quad \mathbf{P^{(1)}} = \begin{pmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 0 \end{pmatrix}$$

2nd Iteration: k=2

```
if (d[i,k] + d[k,j]) < d[i,j]) {
    d[i,j] = d[i,k] + d[k,j]
    pred[i,j] = k   }
```

$$D^{(1)} = \begin{bmatrix} 0 & 8 & ? & 1 \\ ? & 0 & 1 & ? \\ 4 & 12 & 0 & 5 \\ ? & 2 & 9 & 0 \end{bmatrix} \qquad \mathbf{P}^{(1)} = \begin{pmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 0 \end{pmatrix}$$

$$D^{(2)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ ? & 0 & 1 & ? \\ 4 & 12 & 0 & 5 \\ ? & 2 & 3 & 0 \end{bmatrix} \qquad \mathbf{P}^{(2)} = \begin{pmatrix} 0 & -1 & 2 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & 2 & 0 \end{pmatrix}$$

3rd Iteration: k=3

$$D^{(2)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ ? & 0 & 1 & ? \\ 4 & 12 & 0 & 5 \\ ? & 2 & 3 & 0 \end{bmatrix} \qquad \mathbf{P}^{(2)} = \begin{pmatrix} 0 & -1 & 2 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & 2 & 0 \end{pmatrix}$$

```
if (d[i,k] + d[k,j]) < d[i,j]) {
    d[i,j] = d[i,k] + d[k,j]
    pred[i,j] = k   }
```

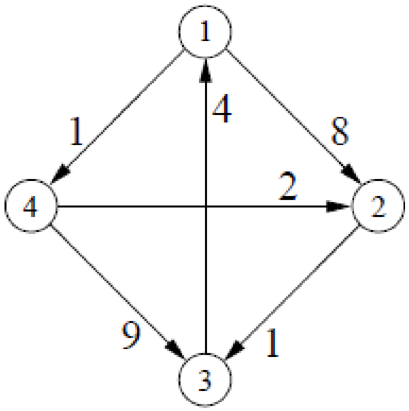
$$D^{(3)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \qquad \mathbf{P}^{(3)} = \begin{pmatrix} 0 & -1 & 2 & -1 \\ 3 & 0 & -1 & 3 \\ -1 & 1 & 0 & 1 \\ 3 & -1 & 2 & 0 \end{pmatrix}$$

4^{th} Iteration: k=4

if (d[i,k] + d[k,j]) < d[i,j]) {
d[i,j] = d[i,k] + d[k,j]
pred[i,j] = k }

$$D^{(3)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \quad \mathbf{P}^{(3)} = \begin{pmatrix} 0 & -1 & 2 & -1 \\ 3 & 0 & -1 & 3 \\ -1 & 1 & 0 & 1 \\ 3 & -1 & 2 & 0 \end{pmatrix}$$

$$D^{(4)} = \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \quad \mathbf{P}^{(4)} = \begin{pmatrix} 0 & 4 & 4 & -1 \\ 3 & 0 & -1 & 3 \\ -1 & 4 & 0 & 1 \\ 3 & -1 & 2 & 0 \end{pmatrix}$$



$$\begin{pmatrix} 0 & 4 & 4 & -1 \\ 3 & 0 & -1 & 3 \\ -1 & 4 & 0 & 1 \\ 3 & -1 & 2 & 0 \end{pmatrix}$$

- Shortest Path from 1 to 4

  1->4

- Shortest Path from 4 to 3

  4->2->3

- Shortest Path from 3 to 2

  3->1->4->2

Analysis of Floyd Warshall Algorithm

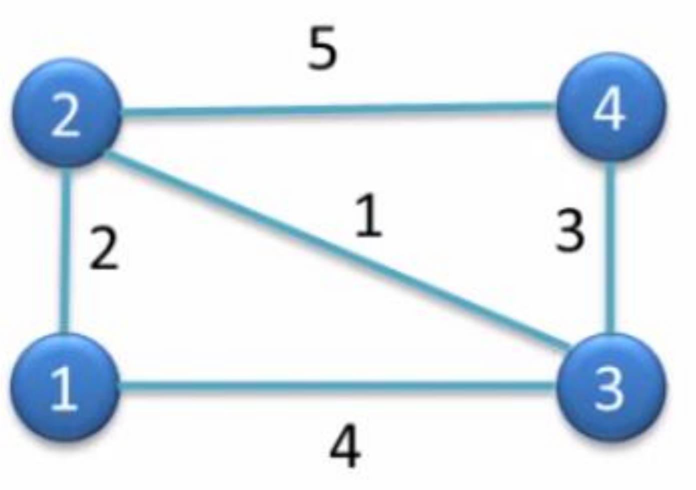
- $O(n^3)$
- $O(V^3)$

Floyd Warshall Algorithm

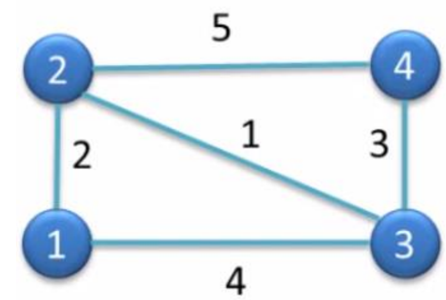
```
Floyd_Warshall (W) {
for i = 1 to n do { // initialize
    for j = 1 to n do {
        d[i,j] = W[i,j]
        pred[i,j] = null
            }
                }
for k = 1 to n do                          // use intermediates {1..k}
    for i = 1 to n do              // ...from i
        for j = 1 to n do                  // ...to j
            if (d[i,k] + d[k,j]) < d[i,j]) {
                d[i,j] = d[i,k] + d[k,j]   // new shorter path length
                pred[i,j] = k  }           // new path is through k
                return d              // matrix of final distances
                    }
```

6

Task



Initially



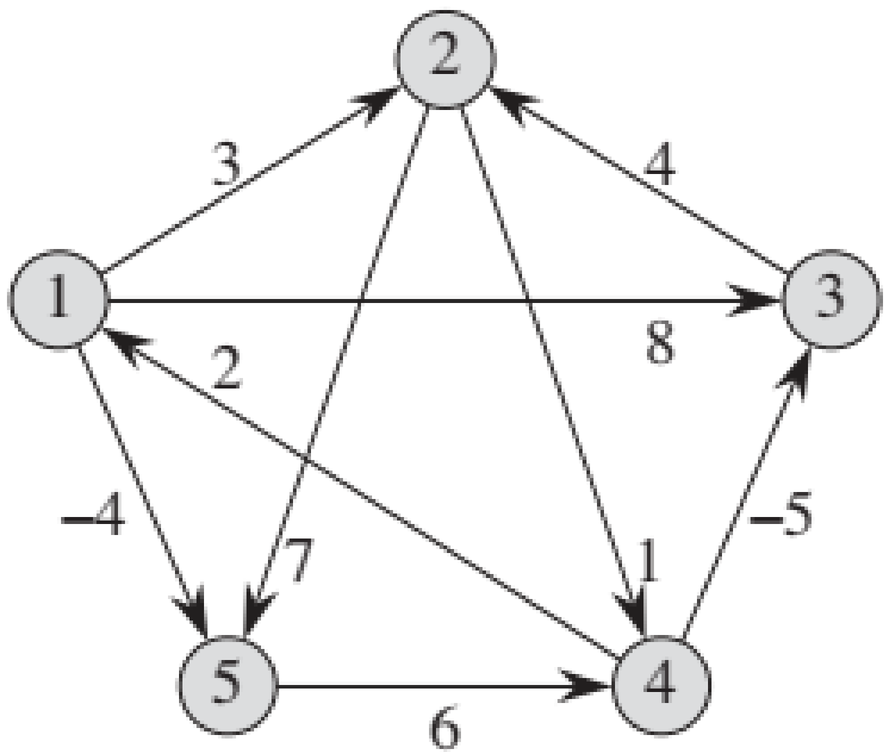
D_0	1	2	3	4
1	0	2	4	∞
2	2	0	1	5
3	4	1	0	3
4	∞	5	3	0

Distance Table

S_0	1	2	3	4
1	0	-1	-1	-1
2	-1	0	-1	-1
3	-1	-1	0	-1
4	-1	-1	-1	0

Sequence Table

Task 2



Compute shortest path using predecessor information

```
Path(i,j) {
    if pred[i,j] = null
        output(i,j)
    else {
        Path(i, pred[i,j]);
        Path(pred[i,j], j);
    }
}
```

Application areas

- Running single source algorithms for each vertex is equivalent to Floyd Warshal (FW).
- In case of only ***+ve*** weights, Dijkstra is obvious choice.
- If the graph is ***dense*** with ***-ve*** weights, then FW is better approach for all source shortest path.
- But to do if the graph is ***sparse***?