# CSE408
# Floyd and warshal
# binomial coff

Lecture # 22

# 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).

# 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.

# 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.

# 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

# 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.

# Intermediate Vertices

Without loss of generality, we will assume that $V=\{1,2,\ldots,n\}$, 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 $\{2,\ldots,m-1\}$ the intermediate vertices of p.

# 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}.

# 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.

# 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}.

# Remark 2

Therefore, we can conclude that

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

# Recursive Formulation

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

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

If k>0, then

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

# 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
            dij(k) = min{dij(k-1) , dik(k-1) + dkj(k-1)};
        od;
    od;
od;
return D(n);
```

$D^{(0)} = W$; $d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$; return $D^{(n)}$

# 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.

# Thank You !!!