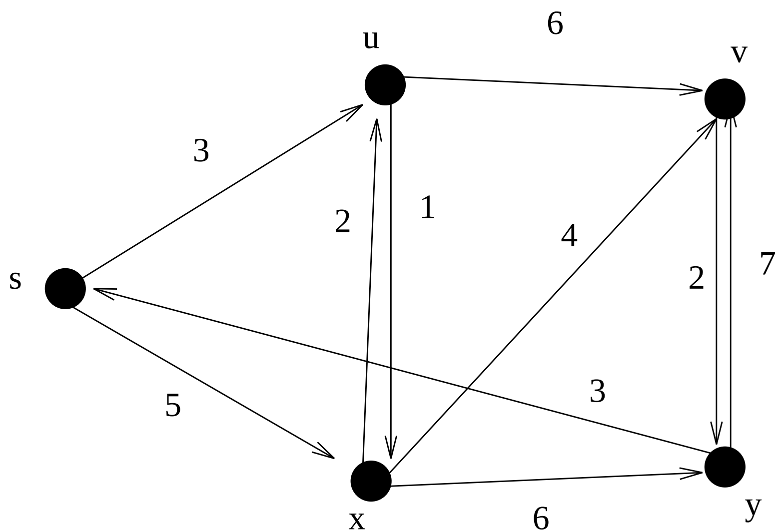
Lecture 19:
All-Pairs Shortest Paths (1997)

Steven Skiena

Department of Computer Science
State University of New York
Stony Brook, NY 11794–4400

http://www.cs.sunysb.edu/∼skiena

Give two more shortest path trees for the following graph:



Run through Dijkstra's algorithm, and see where there are ties which can be arbitrarily selected.

There are two choices for how to get to the third vertex x, both of which cost 5.

There are two choices for how to get to vertex v, both of which cost 9.

All-Pairs Shortest Path

Notice that finding the shortest path between a pair of vertices (s, t) in worst case requires first finding the shortest path from s to all other vertices in the graph.

Many applications, such as finding the center or diameter of a graph, require finding the shortest path between all pairs of vertices.

We can run Dijkstra's algorithm n times (once from each possible start vertex) to solve all-pairs shortest path problem in $O(n^3)$. Can we do better?

Improving the complexity is an open question but there is a *super-slick* dynamic programming algorithm which also runs in $O(n^3)$.

Dynamic Programming and Shortest Paths

The four-step approach to dynamic programming is:

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute this recurrence in a bottom-up fashion.
4. Extract the optimal solution from computed information.

From the adjacency matrix, we can construct the following matrix:

$$
\begin{array}{ll}
D[i,j] = \infty, & \text{if } i \neq j \text{ and } (v_i, v_j) \text{ is not in } E \\
D[i,j] = w(i,j), & \text{if } (v_i, v_j) \in E \\
D[i,j] = 0, & \text{if } i = j
\end{array}
$$

This tells us the shortest path going through no intermediate nodes.

There are several ways to characterize the shortest path between two nodes in a graph. Note that the shortest path from i to j, $i \neq j$, using at most M edges consists of the shortest path from i to k using at most $M-1$ edges+$W(k,j)$ for some k.

This suggests that we can compute all-pair shortest path with an induction based on the number of edges in the optimal path.

Let $d[i,j]^m$ be the length of the shortest path from i to j using at most m edges.

What is $d[i,j]^0$?

$$d[i,j]^0 = 0 \text{ if } i = j$$

$$= \infty \text{ if } i \neq j$$

What if we know $d[i,j]^{m-1}$ for all i, j?

$$\begin{aligned} d[i,j]^m &= \min(d[i,j]^{m-1}, \min(d[i,k]^{m-1} + w[k,j])) \\ &= \min(d[i,k]^{m-1} + w[k,j]), 1 \leq k \leq i \end{aligned}$$

since $w[k,k] = 0$

This gives us a recurrence, which we can evaluate in a bottom up fashion:

for $i = 1$ to n

for $j = 1$ to n

$d[i,j]^m = \infty$

for $k = 1$ to n

$d[i,j]^0$=Min($d[i,k]^m$, $d[i,k]^{m-1} + d[k,j]$)

This is an $O(n^3)$ algorithm just like matrix multiplication, but it only goes from m to $m + 1$ edges.

Since the shortest path between any two nodes must use at most n edges (unless we have negative cost cycles), we must repeat that procedure n times ($m = 1$ to n) for an $O(n^4)$ algorithm.

We can improve this to $O(n^3 \log n)$ with the observation that any path using at most $2m$ edges is the function of paths using at most m edges each. This is just like computing $a^n = a^{n/2} \times a^{n/2}$. So a logarithmic number of multiplications suffice for exponentiation.

Although this is slick, observe that even $O(n^3 \log n)$ is slower than running Dijkstra's algorithm starting from each vertex!

The Floyd-Warshall Algorithm

An alternate recurrence yields a more efficient dynamic programming formulation. Number the vertices from 1 to n.

Let $d[i,j]^k$ *be the shortest path from* i *to* j *using only vertices from* $1, 2, ..., k$ *as possible intermediate vertices.*

What is $d[j,j]^0$? With no intermediate vertices, any path consists of at most one edge, so $d[i,j]^0 = w[i,j]$.

In general, adding a new vertex $k+1$ helps iff a path goes through it, so

$$\begin{aligned} d[i,j]^k &= w[i,j] \text{ if } k = 0 \\ &= \min(d[i,j]^{k-1}, d[i,k]^{k-1} + d[k,j]^{k-1}) \text{ if } k \geq 1 \end{aligned}$$

Although this looks similar to the previous recurrence, it isn't. The following algorithm implements it:

$d^o = w$
for $k = 1$ to n
  for $i = 1$ to n
    for $j = 1$ to n
    $d[i, j]^k = \min(d[i, j]^{k-1}, d[i, k]^{k-1} + d[k, j]^{k-1})$

This obviously runs in $\Theta(n^3)$ time, which asymptotically is no better than a calls to Dijkstra's algorithm. However, the loops are so tight and it is so short and simple that it runs better in practice by a constant factor.