Single-source shortest paths in DAGs. All Pairs Shortest Paths.

The Floyd-Warshall algorithm

November 2013

A DAG (directed acyclic graph) is a digraph without cycles.
ASSUMPTION: $G = (V, E)$ is a DAG with weight function $w : E \to \mathbb{R}$. We write $w(u, v)$ for the weight of arc $u \to v$. We know that

1. G has no cycles $\Rightarrow$ G has no negative cycles $\Rightarrow$ there is a minimal path between every pair of nodes.
2. The nodes of G can be sorted in topological order $v_1, \ldots, v_n$ such that $(v_i \to v_j) \in E$ implies $i < j$.

Single-source shortest paths in DAGs

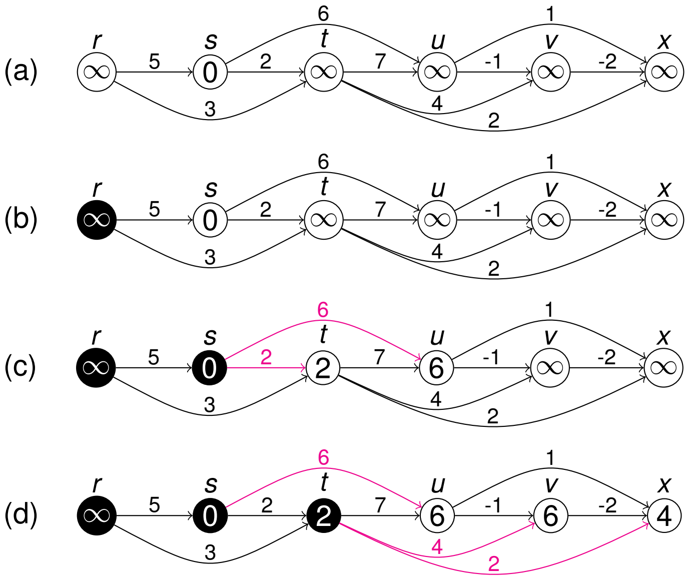
DAGSHORTESTPATHS(G, w, s)
1 topologically sort the nodes of G
2 INITIALIZESINGLESOURCE)(G, s)
3 **for** each node u taken in topologically sorted order
4   **for** each node $v \in Adj[u]$
5   RELAX(u, v, w)

Complexity analysis

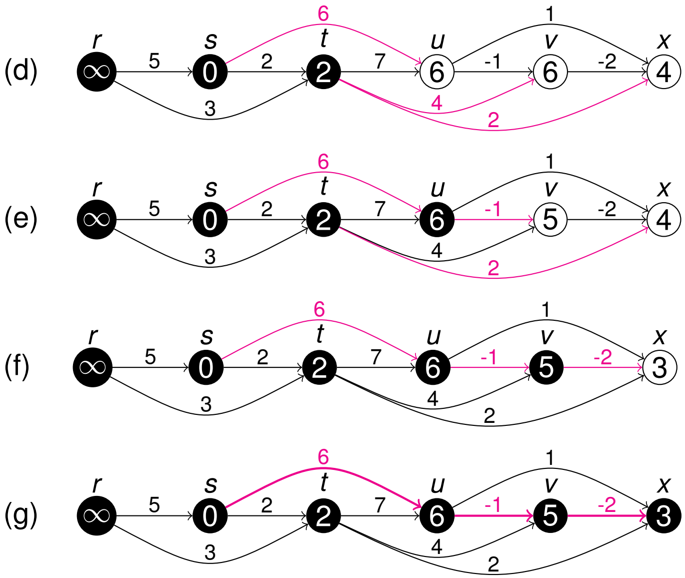
- Line 1 (topological sort) can be done in $\Theta(|V| + |E|)$ time.
- The outer **for** loop is done $|V|$ times.
- For each node, the arcs that leave it are examined exactly once.
- Line 5 takes $O(1)$ time.

$\Rightarrow$ $\Theta(|V| + |E|)$ time.

Single-source shortest paths in DAGs
Running example



Single-source shortest paths in DAGs
Running example



- The magenta arrows are from the predecessor of a node to the node.
- The values inside circles represent the shortest path estimates after each execution of the **for** loop of the algorithm (lines 3-5).

Theorem

If a weighted directed graph $G = (V, E)$ has source node s and no cycles, then at the termination of the DAGSHORTESTPATHS algorithm, we have

1. $d[v] = \delta(s, v)$ for all nodes $v \in V$, and
2. the predecessor subgraph G_π is a shortest paths tree.

PROOF. Cormen *et* al, *Introduction to Algorithms*, Section 25.4.

All pairs shortest path: The problem

ASSUMPTION: Given a directed, weighted graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$.

- Find shortest paths between all pairs of vertices in a graph.
- Example:
  - Make a table of distances between all pairs of cities for a road atlas.

What did we learn so far?

- Single-source shortest-path algorithms:
  - Dijkstra: $O(|V|^2)$
  - Bellman-Ford: $O(|V|\,|E|)$.

- If $w : E \to \mathbb{R}_+$, we can run Dijkstra's single-path algorithm for every node $v \in |V|$

  $\Rightarrow$ overall complexity: $O(|V|) \cdot O(|V|^2) = O(|V|^3)$.
- $w : E \to \mathbb{R}$, we can run Bellman-Ford single-path algorithm for every node $v \in |V|$

  $\Rightarrow$ overall complexity: $O(|V|) \cdot O(|V|\,|E|) = O(|V|^4)$.

Q: Can we do better?

All pairs shortest path algorithms

- We assume given the adjacency matrix of G with n nodes
- For all $i, j \in V$, we know

$$w_{i,j} = \begin{cases} 0 & \text{if } i = j \\ \text{weight of directed edge } (i,j) & \text{if } i \neq j \text{ and } (i,j) \in E \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E. \end{cases}$$

- We want to compute
  - The $n \times n$ matrix $D = (d_{i,j})$ where every $d_{i,j}$ is the length of a shortest path from node i to node j
  - The $n \times n$ predecessor matrix $\Pi = (\pi_{i,j})$ where

$$\pi_{i,j} = \begin{cases} nil & \text{if } i = j \text{ or there is no path from } i \text{ to } j \\ k & \text{if there exists a shortest path } i \rightsquigarrow k \to j \text{ from } i \text{ to } j. \end{cases}$$

All pairs shortest path algorithms

- Suppose we know the predecessor matrix $\Pi = (\pi_{i,j})$.
- For every $i \in V$, define the predecessor graph of G for i as follows: $G_{\pi,i} = (V_{\pi,i}, E_{\pi,i})$, where
  - $V_{\pi,i} := \{j \in V \mid \pi_{i,j} \neq nil\} \cup \{i\}$
  - $E_{\pi,i} := \{(\pi_{i,j}, j) \mid j \in V_{\pi,i}$ and $\pi_{i,j} \neq nil\}$
- For all $i \in V$, $G_{\pi,i}$ is a shortest-path tree with root i.

The structure of a shortest path

- All subpaths of a shortest path are shortest paths.
- Suppose $i \overset{p}{\leadsto} j$ is a shortest path.
  - if $i = j$ then p has weight 0 and no edges.
  - If $i \neq j$ then p contains a finite number of edges, say $\leq m$ edges, and we can write $p = i \overset{p'}{\leadsto} k \to j$, where p' is a path with $\leq m - 1$ edges. Note that
    - p' is a shortest path from i to $k \Rightarrow \delta(i, j) = \delta(i, k) + w_{k,j}$

- Let $d_{i,j}^{(m)} :=$ minimum weight of any path with length $\leq m$ from i to j,
- When $m = 0$, there is a shortest path from i to j with no edges if and only if $i = j$. Thus

$$d_{i,j}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$$

- Recursive formula for $m > 0$:

$$\begin{aligned} d_{i,j}^{(m)} &= \min_{1\leq k\leq n}\left(d_{i,j}^{(m-1)}, \min_{1\leq k\leq n}\left\{d_{i,k}^{(m-1)} + w_{k,j}\right\}\right) \\ &= \min_{1\leq k\leq n}\left\{d_{i,k}^{(m-1)} + w_{k,j}\right\} \end{aligned}$$

NOTE. The last equality holds because $w_{j,j} = 0$ for all j.

Shortest path weights

Q: How to compute the shortest path weights $\delta_{i,j}$ for all nodes $i, j \in V$?

A: If the graph contains no negative-weight cycles then

- All shortest paths are simple and they contain at most $n - 1$ edges
- A path from i to j with more than $n - 1$ edges can not have less weight than a shortest path from i to j
- $\Rightarrow \delta_{i,j} = \delta_{i,j}^{(n-1)} = \delta_{i,j}^{(n)} = \cdots$

Given input matrix $W = (w_{i,j})$

Compute the series of matrices $D^{(1)}, \ldots, D^{(n-1)}$ where for $1 \leq m < n$, we have $D^{(m)} = (\delta_{i,j}^{(m)})$

- Since $d_{i,j}^{(1)} = w_{i,j}$ for all $i, j \in V$, we have $D^{(1)} = W$.

- EXTEND-SHORTEST-PATHS(D, W) is supposed to take the following arguments:
  - $D = D^{(m-1)}$
  - The weight matrix W of the graph

  and will return $D^{(m)}$

Shortest path weights
The bottom-up approach

```
EXTEND-SHORTEST-PATHS(D, W)
n ← rows[D]
let D′ = (d′_ij) be an n × n matrix
for i ← 1 to n
        do for j ← 1 to n
                do d′_ij ← ∞
                   for k ← 1 to n
                       do d′_ij ← min(d′_ij, d_ik + w_kj)
return D′
```

- Complexity = $\Theta(n^3)$ – due to the three nested **for** loops.

- If A, B are $n \times n$ matrices and $C = (c_{i,j}) = A \cdot B$ then

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} \cdot b_{k,j}$$

- The matrix multiplication formula is equivalent with the recursive formula for $d^{(m)}$, if we do the following changes:

$$\begin{array}{rcl} d^{(m-1)} & \leftrightarrow & a \\ w & \leftrightarrow & b \\ d^{(m)} & \leftrightarrow & c \\ \min & \leftrightarrow & + \\ + & \leftrightarrow & \cdot \end{array}$$

Shortest path weights

The relation between the bottom-up approach and matrix multiplication

- Based on the previous observation, we get the following matrix multiplication algorithm by changing EXTEND-SHORTEST-PATHS:

```
MATRIX-MULTIPLY(A, B)
n ← rows[A]
let C be an n × n matrix
for i ← 1 to n
      do for j ← 1 to n
              do c_ij ← 0
                 for k ← 1 to n
                     do c_ij ← c_ij + a_ik · b_kj
return C
```

- Time complexity of MATRIX-MULTIPLY: $\Theta(n^3)$.

If we write $A \cdot B$ instead of EXTEND-SHORTEST-PATHS(A, B), we compute

Computation	Time complexity
$D^{(1)} = D^{(0)} \cdot W = W$	$\Theta(n^3)$
$D^{(2)} = D^{(1)} \cdot W = W^2$	$\Theta(n^3)$
$\vdots$	
$D^{(n-1)} = D^{(n-2)} \cdot W = W^{n-1}$	$\Theta(n^3)$

- In the end, W^{n-1} contains the shortest-path weights of all pairs of nodes.
- Total time complexity: $(n-1) \cdot \Theta(n^3) = \Theta(n^4)$ time.

Shortest path weights
The bottom-up approach

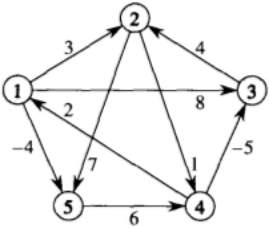
```
SLOW-ALL-PAIRS-SHORTEST-PATHS(W)
n ← rows[W]
D^(1) ← W
for m ← 2 to n − 1
      do D^(m) ← EXTEND-SHORTEST-PATHS(D^(m−1), W)
return D^(n−1)
```

The computation of $D^{(n-1)}$ can be improved if we use the method of repeated squaring:

```
FASTER-ALL-PAIRS-SHORTEST-PATHS(W)
n ← rows[W]
D^(1) ← W
m ← 1
while n − 1 > m
      do D^(2m) ← EXTEND-SHORTEST-PATHS(D^(m), D^(m))
            m ← 2m
return D^(m)
```

Total time complexity: $\lceil \log_2(n-1) \rceil \cdot \Theta(n^3) = \Theta(n^3 \cdot \lceil \log_2(n-1) \rceil)$.

The bottom-up approach without repeated squaring
Example



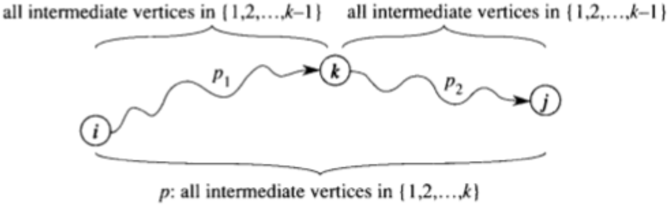
$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad D^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

ASSUMPTION: there are no cycles with negative weight

- An intermediate node of a simple path $p = (v_1, v_2, \ldots, v_{\ell-1}, v_\ell)$ is any node $v \in \{v_2, v_3, \ldots, v_{\ell-1}\}$.
- The Floyd-Warshall algorithm is based on the following observation:
  - Assume $V = \{1, 2, \ldots, n\}$. For all $i, j \in V$ consider all paths from i to j whose intermediate nodes are from $\{1, \ldots, k\}$. Let $i \overset{p}{\leadsto} j$ be such a minimum-weight path from i to j, and let $d_{i,j}^{[k]}$ be its weight.
  - $d_{i,j}^{[k]}$ can be computed by recursion on k.

1. If k is an intermediate node of path p, we have $i \overset{p_1}{\leadsto} k \overset{p_2}{\leadsto} j$:



   where p_1 is minimum-weight path from i to k through nodes $\{1, \ldots, k-1\}$, and p_2 is a minimum-weight path from k to j through nodes $\{1, \ldots, k-1\}$. Thus $d_{i,j}^{[k]} = d_{i,k}^{[k-1]} + d_{k,j}^{[k-1]}$

2. If j is not intermediate node of path p then all intermediate nodes of p are from $\{1, \ldots, k-1\}$, thus $d_{i,j}^{[k]} = d_{i,j}^{[k-1]}$.

A recursive definition of $d_{i,j}^{[k]}$:

$$d_{i,j}^{[k]} := \begin{cases} w_{i,j} & \text{if } k = 0, \\ \min\left(d_{i,j}^{[k-1]}, d_{i,k}^{[k-1]} + d_{k,j}^{[k-1]}\right) & \text{if } k \geq 1. \end{cases}$$

- $d_{i,j}^{[n]}$ is the weight of a shortest path from i to j for all $i, j \in V$.

$\Rightarrow$ we wish to compute the $n \times n$ matrix $D^{[n]} = \left(d_{i,j}^{[n]}\right)$.

The Floyd-Warshall algorithm

- Computes $d_{i,j}^{[k]}$ in order of increasing values of k.

  FLOYD-WARSHALL(W)
  1 $n := rows[W]$
  2 $D^{[0]} := W$
  3 **for** $k := 1$ to n
  4   **for** $i := 1$ to n
  5   **for** $j := 1$ to n
  6             $d_{ij}^{[k]} := \min\left(d_{ij}^{[k-1]}, d_{ik}^{[k-1]} + d_{kj}^{[k-1]}\right)$
  7 **return** $D^{[n]}$

- Running time complexity:
  - There are 3 nested loops (lines 3-6)
  - The execution time of line 6 takes $O(1)$ time

  $\Rightarrow \Theta(n^3)$ time complexity.

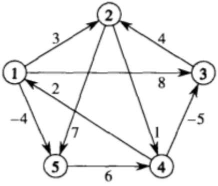
The predecessor matrix Π can be computed incrementally, via a sequence of matrices

$$\Pi^{[0]}, \Pi^{[1]}, \ldots, \Pi^{[n]}$$

where $\Pi = \Pi^{[n]}$ and $\pi_{i,j}^{[k]}$ is the predecessor of vertex j on a shortest path from vertex i with all intermediate nodes in the set $\{1, 2, \ldots, k\}$.
Formally, we have

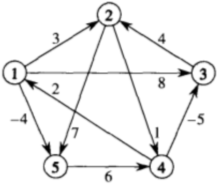
$$\pi_{i,j}^{[0]} = \begin{cases} nil & \text{if } i = j \text{ or } w_{i,j} = \infty, \\ i & \text{if } i \neq j \text{ and } w_{i,j} < \infty \end{cases}$$

$$\pi_{i,j}^{[k]} = \begin{cases} \pi_{i,j}^{[k-1]} & \text{if } d_{i,j}^{[k-1]} \leq d_{i,k}^{[k-1]} + d_{k,j}^{[k-1]} \\ \pi_{k,j}^{[k-1]} & \text{if } d_{i,j}^{[k-1]} > d_{i,k}^{[k-1]} + d_{k,j}^{[k-1]} \end{cases}$$

For the graph  we have

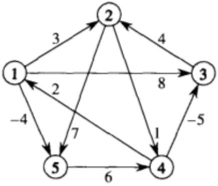
$$D^{[0]} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{[0]} = \begin{pmatrix} nil & 1 & 1 & nil & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & nil & nil \\ 4 & nil & 4 & nil & nil \\ nil & nil & nil & 5 & nil \end{pmatrix}$$

$$D^{[1]} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{[1]} = \begin{pmatrix} nil & 1 & 1 & nil & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & nil & nil \\ 4 & 1 & 4 & nil & 1 \\ nil & nil & nil & 5 & nil \end{pmatrix}$$

For the graph  we have

$$D^{[2]} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{[2]} = \begin{pmatrix} nil & 1 & 1 & 2 & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & 2 & 2 \\ 4 & 1 & 4 & nil & 1 \\ nil & nil & nil & 5 & nil \end{pmatrix}$$

$$D^{[3]} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{[3]} = \begin{pmatrix} nil & 1 & 1 & 2 & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & 2 & 2 \\ 4 & 3 & 4 & nil & 1 \\ nil & nil & nil & 5 & nil \end{pmatrix}$$

For the graph  we have

$$D^{[4]} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{[4]} = \begin{pmatrix} nil & 1 & 4 & 2 & 1 \\ 4 & nil & 4 & 2 & 1 \\ 4 & 3 & nil & 2 & 1 \\ 4 & 3 & 4 & nil & 1 \\ 4 & 3 & 4 & 5 & nil \end{pmatrix}$$

$$D^{[5]} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{[5]} = \begin{pmatrix} nil & 3 & 4 & 5 & 1 \\ 4 & nil & 4 & 2 & 1 \\ 4 & 3 & nil & 2 & 1 \\ 4 & 3 & 4 & nil & 1 \\ 4 & 3 & 4 & 5 & nil \end{pmatrix}$$