Shortest paths in a DAG

Let $G = (V, E)$ be a DAG with edge weights $\text{wt} : E \rightarrow \mathbb{R}$ (edge weights may be negative)

Linear time (i.e., $O(|V| + |E|)$) algorithm for Single Destination variant (reverse $G$ for Single Source variant)

Given $G$ as above and $t \in V$, find shortest paths from all nodes $v \in V$ to $t$

Assume $V = [0..n)$ and let $\text{TopSort}[0..n)$ be an array that lists vertices in a topological order

if $u \rightarrow v$ is an edge, then $u$ appears before $v$ in the $\text{TopSort}$ array
We will compute $d[v] =$ weight of shortest path to $t$ for all $v \in V$

$$
\begin{align*}
&d[v] \leftarrow \infty \text{ for } v \text{ in } [0..n) \\
&d[t] \leftarrow 0
\end{align*}
$$

for $i$ in reverse $[0..n)$

$$
\begin{align*}
u &\leftarrow TopSort[i] \\
d[u] &\leftarrow \min \left( \{d[u]\} \cup \\
&\{ wt(u, v) + d[v] : v \in \text{Successor}(u) \} \right)
\end{align*}
$$

A more concrete implementation of “$d[u] \leftarrow \min(\ldots)$”

for each $v \in \text{Successor}(u)$ do

$$
\begin{align*}
\text{if } wt(u, v) + d[v] < d[u] \text{ then } \\
&d[u] \leftarrow wt(u, v) + d[v]
\end{align*}
$$
Breadth first search (BFS)

Input: a graph $G = (V, E)$, and a node $s \in V$

- The graph is *unweighted*
- Equivalently, all edges have weight $1$

Outputs:

- the “shortest distance” array $d$, indexed by $V$, so that $d[\nu] =$ length of shortest path from $s$ to $\nu$
- a “breadth first search” tree $T$, represented as an array $\pi$ indexed by $V$

$\pi[\nu] = u$ means $u$ is $\nu$’s parent in $T$

the root $T$ is $s$, and paths in $T$ are shortest paths in $G$
Algorithm $BFS(G, s)$:

for each $v \in V$
   
   $Color[v] \leftarrow \text{white}$  // undiscovered
   
   $d[v] \leftarrow \infty$, $\pi[v] \leftarrow \text{Nil}$

$Color[s] \leftarrow \text{gray}$  // discovered
$d[s] \leftarrow 0$, $\pi[s] \leftarrow \text{Nil}$

$Q \leftarrow \text{NewQueue()}$  // a FIFO queue
$Q\text{.enqueue}(s)$

while not $Q\text{.empty()}$ do
   
   $u \leftarrow Q\text{.dequeue}()$

   for each $v \in \text{Successor}(u)$ do
      
      if $Color[v] = \text{white}$ then
         
         $Color[v] \leftarrow \text{gray}$  // discovered

         $d[v] \leftarrow d[u] + 1$, $\pi[v] \leftarrow u$

         $Q\text{.enqueue}(v)$

      $Color[u] \leftarrow \text{black}$  // finished
Example:
Running time:

- Each node enqueued at most once (by coloring)
- Each node dequeued at most
- Each adjacency list scanned at most once
- \( \therefore \) Running time \( = O(|V| + |E|) \)

Invariant:

- At the beginning of each loop iteration, \( Q \) contains all nodes that are colored \textit{gray}. 
Correctness

Notation: \( d[\nu] = \) computed distance
\( \delta(s, \nu) = \) length of shortest path from \( s \) to \( \nu \)

Shortest Path Lemma

If \( \delta(s, \nu) = m > 0 \), then \( \nu \) is the successor of some node \( u \) with \( \delta(s, u) = m - 1 \)

Proof:

- Consider a shortest path from \( s \) to \( \nu \):
  \[
  s \sim u \rightarrow \nu \\
  \begin{array}{c}
  m-1 \\
  m
  \end{array}
  \]

- The path \( s \sim u \) must be a shortest path from \( s \) to \( u \) (otherwise, we could find an even shorter path to \( \nu \)). QED
Theorem
Algorithm BFS eventually discovers every node reachable from \(s\)

Prove by induction on \(m\):

\[
\text{for all } v \in V, \text{ if } \delta(s, v) = m, \text{ then BFS discovers } v
\]

\(m = 0\): clear; \(m > 0\):

- Suppose \(v \in V\) with \(\delta(s, v) = m\)
- By Shortest Path Lemma, \(v\) has a predecessor \(u\) with \(\delta(s, u) = m - 1\)
- By induction, BFS discovered \(u\), and placed \(u\) in \(Q\)
- When BFS removes \(u\) from \(Q\), it discovers \(v\) (or finds that it was already discovered)
Theorem

BFS correctly computes $d[\nu] = \delta(s, \nu)$ for all $\nu \in V$

Proof:

• Let $\nu_0, \nu_1, \ldots$ be the nodes listed in the order they are removed from $Q$

• We can partition the execution of BFS into epochs $0, 1, 2, \ldots$

  $\nu_0, \ldots, \nu_j_0, \nu_{j_0+1}, \ldots, \nu_{j_1}, \ldots$

    epoch 0  epoch 1

• A new epoch starts at $\nu_j$ if $\delta(s, \nu_j) \neq \delta(s, \nu_{j-1})$
Prove by induction on $i$:

At the beginning of epoch $i$, $Q$ contains precisely all nodes $v$ such that $\delta(s, v) = i$, and $d[v] = i$ for all these nodes

$i = 0$: clear

Assume for $0, \ldots, i$ and prove for $i + 1$:

- By induction hypothesis, at beginning of epoch $i$, $Q$ contains precisely all nodes $v$ such that $\delta(s, v) = i$, and $d[v] = i$ for all these nodes
- Moreover, all nodes $v$ with $\delta(s, v) = i + 1$ are as yet undiscovered
- By Shortest Path Lemma, all nodes $v$ with $\delta(s, v) = i + 1$ will be discovered and placed at end of $Q$ during epoch $i$
- Epoch $i$ ends when all nodes $v$ with $\delta(s, v) = i$ have been removed from $Q$

QED. One can also easily show that $T$ is correct
Recap: Single source / destinations shortest paths

Assume $G = (V, E)$, with $n := |V|$ and $m := |E|$

- No negative edges: $O((n + m) \log n)$ — Dijkstra
- Bounded, non-negative, integer edge weights: $O(n + m)$ — Dijkstra variant (or BFS)
- DAG with arbitrary edge weights: $O(n + m)$
All pairs shortest paths

One approach:

- Run a single-source shortest path algorithm from each vertex
  - Dijkstra (no negative edges): $O(n \times (n + m) \log n)$, or $O(n^3)$

Floyd-Warshall Algorithm:

- no negative cycles
- running time $O(n^3)$
• Number the vertices \([1 \ldots n]\)

• For a path \(p = \langle v_0, v_1, \ldots, v_{\ell - 1}, v_{\ell} \rangle\), we say that \(v_1, \ldots, v_{\ell - 1}\) are *intermediate* vertices

• For \(k\) in \([0 \ldots n]\), let \(\delta^{(k)}(i, j) := \text{length of the shortest path from } i \text{ to } j \text{ whose intermediate vertices belong to } [1 \ldots k] \)

\[
\delta^{(0)}(i, j) = \begin{cases} 
0 & \text{if } i = j; \\
\text{wt}(i, j) & \text{if } i \neq j \text{ and } (i, j) \in E \\
\infty & \text{otherwise}
\end{cases}
\]

• For \(k > 0\)

\[
\delta^{(k)}(i, j) = \min \left( \delta^{(k-1)}(i, j), \delta^{(k-1)}(i, k) + \delta^{(k-1)}(k, j) \right)
\]
Straightforward implementation:

- Use a 3D array $D[i, j, k]$

\[
D[i, j, 0] \leftarrow \delta^{(0)}(i, j) \text{ for } i, j \text{ in } [0..n]
\]

for $k$ in $[1..n]$ do

  for $i$ in $[1..n]$ do

    for $j$ in $[1..n]$ do

      \[
d' \leftarrow D[i, k, k - 1] + D[k, j, k - 1]
      \]

      if $d' < D[i, j, k - 1]$ then

        \[
        D[i, j, k] \leftarrow d'
        \]

      else

        \[
        D[i, j, k] \leftarrow D[i, j, k - 1]
        \]

- Running time: $O(n^3)$

- Space: $O(n^3)$
Improving the space requirement:

- Since $D[\cdot, \cdot, k]$ depends only on $D[\cdot, \cdot, k-1]$, we can obviously get by with just two 2D arrays.

- In fact, we can get by with just a single array, with updates "in place".

Justification:

- $\delta^{(k)}(i, k) = \delta^{(k-1)}(i, k)$
- $\delta^{(k)}(k, j) = \delta^{(k-1)}(k, j)$

- Why? No negative cycles.

- So in the formula:

$$\delta^{(k)}(i, j) = \min(\delta^{(k-1)}(i, j), \delta^{(k-1)}(i, k) + \delta^{(k-1)}(k, j))$$

these don’t change in loop iteration $k$
Improved implementation:

- Use a 2D array $D[i, j]$

\[
D[i, j] \leftarrow \delta^{(0)}(i, j) \text{ for } i, j \text{ in } [0..n)
\]

for $k$ in $[1..n]$ do

for $i$ in $[1..n]$ do

for $j$ in $[1..n]$ do

\[d' \leftarrow D[i, k] + D[k, j]\]

if $d' < D[i, j]$

then $D[i, j] \leftarrow d'$
Adding path recovery:

- Two arrays: $D[i, j]$, $N[i, j]$
- $N[i, j] = \text{next vertex in the shortest path from } i \text{ to } j$

$D[i, j] \leftarrow \delta^{(0)}(i, j) \text{ for } i, j \in [0..n)$

$N[i, j] \leftarrow j \text{ for } i, j \in [0..n)$

for $k$ in $[1..n]$ do
  for $i$ in $[1..n]$ do
    for $j$ in $[1..n]$ do
      $d' \leftarrow D[i, k] + D[k, j]$
      if $d' < D[i, j]$
        then $D[i, j] \leftarrow d'$

$N[i, j] \leftarrow N[i, k]$

Printing a shortest path from $u$ to $v$:

$x \leftarrow u$, print $x$

while $x \neq v$ do: $x \leftarrow N[x, v]$, print $x$