Shortest paths in a DAG

Let \( G = (V, E) \) be a DAG with edge weights \( wt : E \to \mathbb{R} \) (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 \( TopSort[0..n] \) be an array that lists vertices in a topological order

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

**Idea:**

Evaluate $d$ values right to left

$$d[u] = \min \{ \text{wt}(u, v) + d[v], \text{wt}(u, v') + d[v'] \}$$

**Algorithm:**

for $v$ in $[0..n)$ do: $d[v] \leftarrow \infty$

$d[t] \leftarrow 0$

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

$u \leftarrow \text{TopSort}[i]$

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

if $\text{wt}(u, v) + d[v] < d[u]$ then

$d[u] \leftarrow \text{wt}(u, v) + d[v]$
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 $d[v] =$ length of shortest path from $s$ to $v$
- a “breadth first search” tree $T$, represented as an array $\pi$ indexed by $V$

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

$s$ is the root $T$, and paths in $T$ are shortest paths in $G$
Basic Idea:

place $s$ in bucket 0
for $i$ in $[0..n)$ do
  for each $u$ in bucket $i$ do
    for each $v \in \text{Successor}(u)$ do
      if $v$ is not already in some bucket then
        place $v$ in bucket $i + 1$

Claim: a node is placed in bucket $i \iff$ it is at distance $i$ from $s$

- If $\delta(s, v) = i + 1 > 0$, then $v$ is the successor of some node $u$ with $\delta(s, u) = i$
  - Why? Consider a shortest path from $s$ to $v$:
    \[
    s \rightsquigarrow u \rightarrow v
    \]
      \[
      \begin{array}{c}
        \text{i} \\
        \text{i+1}
      \end{array}
    \]
  - The path $s \rightsquigarrow u$ must be a shortest path from $s$ to $u$
    (otherwise, we could find an even shorter path to $v$)
Observations:

- Instead of \( n \) buckets, we can just use a single FIFO queue
- The nodes in the front of the queue are all the unexamined nodes in bucket \( i \)
- The nodes in the rear of the queue are all the nodes in bucket \( i + 1 \)
Algorithm \(BFS(G, s)\):

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

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

\(Q \leftarrow \text{CreateQueue()} \quad // \text{a FIFO queue}\)
\(Q.\text{enqueue}(s)\)

while not \(Q.\text{empty}()\) do
  \(u \leftarrow Q.\text{dequeue}()\)
  for each \(v \in \text{Successor}(u)\) do
    if \(\text{Color}[v] = \text{white}\) then
      \(\text{Color}[v] \leftarrow \text{gray} \quad // \text{discovered}\)
      \(d[v] \leftarrow d[u] + 1, \pi[v] \leftarrow u\)
      \(Q.\text{enqueue}(v)\)

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

- Each node enqueued at most once (by coloring)
- Each node dequeued at most
- Each adjacency list scanned at most once
- \[ \therefore \text{Running time} = O(|V| + |E|) \]
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 \textit{intermediate} vertices

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

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

• 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 $D[i, j, k]$

\[
D[i, j, 0] \leftarrow \delta^{(0)}(i, j) \text{ for } i, j \text{ in } [1..n]
\]

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 $D[i, j]$

\[
D[i, j] \leftarrow \delta^{(0)}(i, j) \text{ for } i, j \text{ in } [1..n]
\]

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

\[
D[i, j] \leftarrow \delta^{(0)}(i, j) \text{ for } i, j \in [1..n] \\
N[i, j] \leftarrow j \text{ for } i, j \in [1..n]
\]

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 $x$