Algorithms 11

CS201

Kaustuv Nag

Shortest-Paths Problems

In a *shortest-paths problem*, we are given a weighted, directed graph $G = (V, E)$, with weight function $w : E \to \mathbb{R}$ mapping edges to real-valued weights. The weight $w(p)$ of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituent edges:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

We define the *shortest-path weight* $\delta(u, v)$ from u to v by

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\} & \text{if there is a path from } u \text{ to } v; \\ \infty & \text{otherwise.} \end{cases}$$

A *shortest path* from vertex u to vertex v is then defined as any path p with weight $w(p) = \delta(u, v)$.

Variants of Shortest-Paths Problems

Single-source Shortest-paths Problem

Given a graph $G = (V, E)$, we want to find a shortest path from a given source vertex $s \in V$ to each vertex $v \in V$. The algorithm for the single-source problem can solve many other problems, including the following variants.

Single-destination shortest-paths problem Find a shortest path to a given destination vertex t from each vertex v. By reversing the direction of each edge in the graph, we can reduce this problem to a single-source problem.

Single-pair shortest-path problem Find a shortest path from u to v for given vertices u and v. If we solve the single-source problem with source vertex u, we solve this problem also.

All-pairs shortest-paths problem Find a shortest path from u to v for every pair of vertices u and v. Although we can solve this problem by running a single source algorithm once from each vertex, we usually can solve it faster.

Shortest-Paths Problems

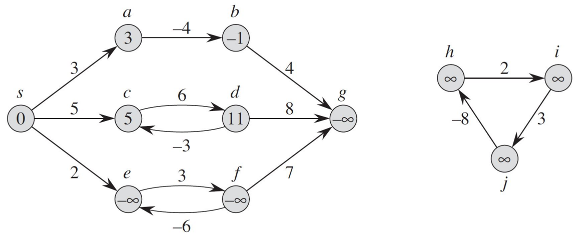
Lemma

Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $p = \langle v_0, v_1, \cdots, v_k \rangle$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \leq i \leq j \leq k$, let $p_{ij} = \langle v_i, v_{i+1}, \cdots, v_j \rangle$ be the subpath of p from vertex v_i to vertex v_j. Then, p_{ij} is a shortest path from v_i to v_j.

Proof

If we decompose path p into $v_0 \overset{p_{0i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$, then we have that $w(p) = w(p_{0i}) + w(p_{ij}) + w(p_{jk})$. Now, assume that there is a path p'_{ij} from v_i to v_j with weight $w(p_{ij}) < w(p'_{ij})$. Then, $v_0 \overset{p_{0i}}{\rightsquigarrow} v_i \overset{p'_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$ is a path from v_0 to v_k whose weight $w(p_{0i}) + w(p'_{ij}) + w(p_{jk})$ is less than $w(p)$, which contradicts the assumption that p is a shortest path from v_0 to v_k. ■

Shortest-Paths Problems: Negative Weights



- Infinitely many paths from s to c: $\langle s,c \rangle$, $\langle s,c,d,c \rangle$, $\langle s,c,d,c,d,c \rangle$, and so on. Because the cycle $\langle c,d,c \rangle$ has weight $6+(-3)=3>0$, the shortest path from s to c is $\langle s,c \rangle$, with weight $\delta(s,c)=w(s,c)=5$.
- By traversing the negative-weight cycle $\langle e,f,e \rangle$ arbitrarily many times, we can find paths from s to e with arbitrarily large negative weights, and so $\delta(s,e)=-\infty$.
- $\delta(s,h)=\delta(s,i)=\delta(s,j)=\infty$.
- A shortest path cannot have cycles. It contains at most $|V|$ distinct vertices, it also contains at most $|V|-1$ edges.

Shortest-Paths Problems: Representation of Paths

- Given a graph $G = (V, E)$, we maintain for each vertex $v \in V$ a predecessor $v.\pi$ that is either another vertex or NIL. The shortest-paths algorithms that we discuss set the π attributes so that the chain of predecessors originating at a vertex π runs backwards along a shortest path from s to π.

```
PRINT-PATH($G, s, v$)
1  if $v == s$
2      print $s$
3  elseif $v.\pi ==$ NIL
4      print "no path from" $s$ "to" $v$ "exists"
5  else PRINT-PATH($G, s, v.\pi$)
6      print $v$
```

Shortest-paths Trees

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$, and assume that G contains no negative-weight cycles reachable from the source vertex $s \in V$, so that shortest paths are well defined. A shortest-paths tree rooted at s is a directed subgraph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, such that

- V' is the set of vertices reachable from s in G,
- G' forms a rooted tree with root s, and
- for all $v \in V'$, the unique simple path from s to v in G' is a shortest path from s to v in G.

Shortest-path Estimate

For each vertex $v \in V$, we maintain an attribute $v.d$, which is an upper bound on the weight of a shortest path from source s to v. We call $v.d$ a shortest-path estimate.

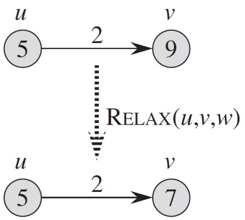
```
INITIALIZE-SINGLE-SOURCE$(G, s)$
1  for each vertex $v \in G.V$
2      $v.d = \infty$
3      $v.\pi = \text{NIL}$
4  $s.d = 0$
```

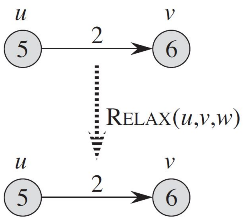
After initialization, we have $v.\pi = \text{NIL}$ for all $v \in V$, $s.d = 0$, and $v.d = \infty$ for $v \in V - \{s\}$.

Relaxation

The process of relaxing an edge (u, v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating (decreasing) $v.d$ and updating $v.\pi$.

```
RELAX(u, v, w)
1  if $v.d > u.d + w(u, v)$
2      $v.d = u.d + w(u, v)$
3      $v.\pi = u$
```





Properties of Shortest Paths and Relaxation

Triangle inequality: For any edge $(u,v) \in E$, we have $\delta(s,v) \leq \delta(s,u) + w(u,v)$.

Upper-bound property: We always have $v.d \geq \delta(s,v)$ for all vertices $v \in V$, and once $v.d$ achieves the value $\delta(s,v)$, it never changes.

No-path property: If there is no path from s to v, then we always have $v.d = \delta(s,v) = \infty$.

Convergence property: If $s \rightsquigarrow u \rightarrow v$ is a shortest path in G for some $u,v \in V$, and if $u.d = \delta(s,u)$ at any time prior to relaxing edge (u,v), then $v.d = \delta(s,v)$ at all times afterward.

Path-relaxation property: If $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0,v_1),(v_1,v_2),\ldots(v_{k-1},v_k)$, then $v_k.d = \delta(s,v_k)$. This property holds regardless of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of p.

Predecessor-subgraph property: Once $v.d = \delta(s,v)$ for all $v \in V$, the predecessor subgraph is a shortest-paths tree rooted at s.

Single Source Shortest Paths Problmes

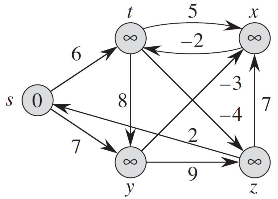
Bellman-Ford algorithm

- Given a weighted, directed graph $G = (V,E)$ with source s and weight function $w : E \rightarrow \mathbb{R}$, the Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source.
- If there is such a cycle, the algorithm indicates that no solution exists. If there is no such cycle, the algorithm produces the shortest paths and their weights.
- The algorithm relaxes edges, progressively decreasing an estimate $v.d$ on the weight of a shortest path from the source s to each vertex $v \in V$ until it achieves the actual shortest-path weight $\delta(s,v)$. The algorithm returns TRUE if and only if the graph contains no negative-weight cycles that are reachable from the source.
- The Bellman-Ford algorithm runs in time $O(VE)$.

Single Source Shortest Paths Problmes

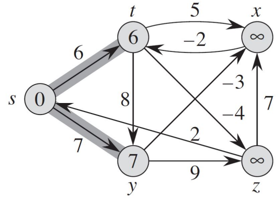
Bellman-Ford algorithm

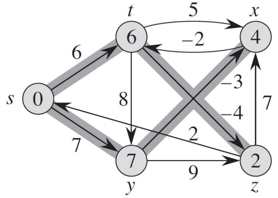
```
1 INITIALIZE-SINGLE-SOURCE$(G, s)$
2 for $i = 1$ to $|G.V| - 1$
3     for each edge $(u, v) \in G.E$
4         RELAX$(u, v, w)$
5 for each edge $(u, v) \in G.E$
6     if $v.d > u.d + w(u, v)$
7         return FALSE
8 return TRUE
```

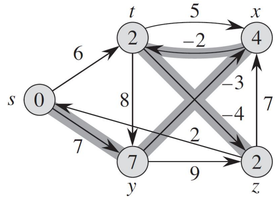


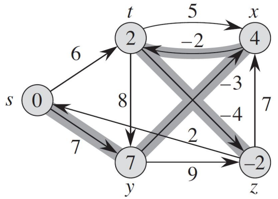
Single Source Shortest Paths Problmes

Bellman-Ford algorithm



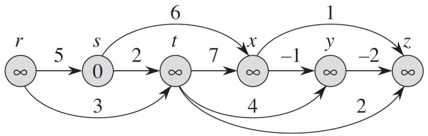


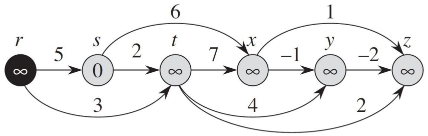


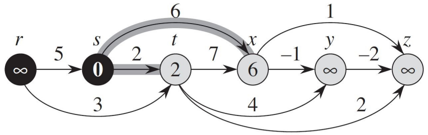


Single-source Shortest Paths in Directed Acyclic Graphs

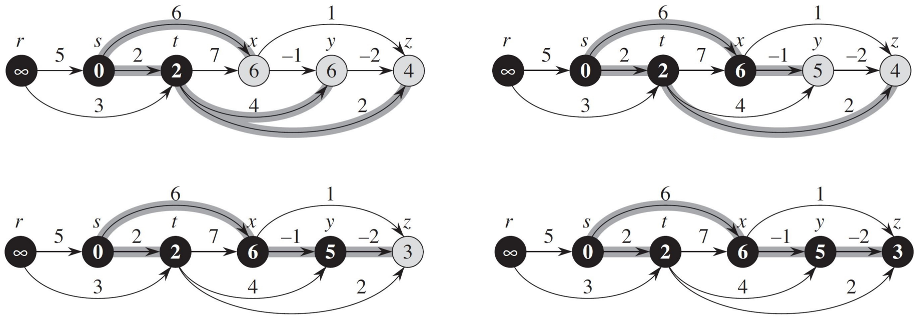
```
DAG-SHORTEST-PATHS$(G, w, s)$
1  topologically sort the vertices of $G$
2  INITIALIZE-SINGLE-SOURCE$(G, s)$
3  for each vertex $u$, taken in topologically sorted order
4      for each vertex $v \in G.Adj[u]$
5          RELAX$(u, v, w)$
```







Single-source Shortest Paths in Directed Acyclic Graphs



Single-source Shortest Paths in Directed Acyclic Graphs

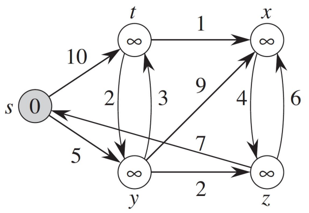
Running time

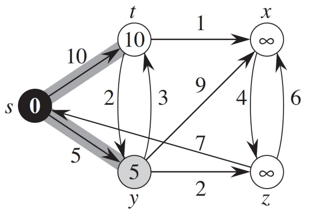
- The topological sort of line 1 takes $\Theta(V+E)$ time. The call of INITIALIZE-SINGLE-SOURCE in line 2 takes $\Theta(V)$ time.
- The for loop of lines 3–5 makes one iteration per vertex. Altogether, the for loop of lines 4–5 relaxes each edge exactly once. Because each iteration of the inner for loop takes $\Theta(1)$ time, the total running time is $\Theta(V+E)$, which is linear in the size of an adjacency-list representation of the graph.

Single-source Shortest Paths

Dijkstra's algorithm

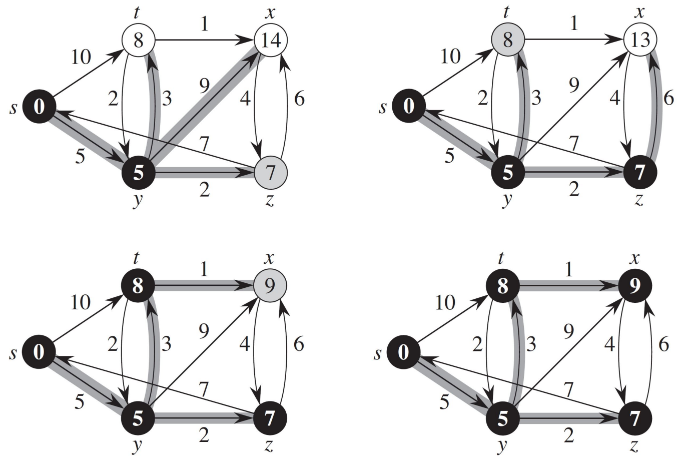
```
DIJKSTRA$(G, w, s)$
1  INITIALIZE-SINGLE-SOURCE$(G, s)$
2  $S = \emptyset$
3  $Q = G.V$
4  while $Q \neq \emptyset$
5      $u$ = EXTRACT-MIN$(Q)$
6      $S = S \cup \{u\}$
7      for each vertex $v \in G.Adj[u]$
8          RELAX$(u, v, w)$
```





Single-source Shortest Paths

Dijkstra’s algorithm



Single-source Shortest Paths

Dijkstra's algorithm: running time

- The running time depends on how we implement the priority queue. If we use a min-heap:
  - Setp 1 takes $O(V)$ time.
  - Step 2 takes $O(1)$ time.
  - Step 3 takes $O(V)$ time.
  - Step 4-8 is executed $|V|$ times.
  - Step 7-8 is executed $|E|$ times.
  - Step 5 takes $O(\lg V)$ time. Thus in total takes $O(V \lg V)$ time.
  - Step 6 takes $O(1)$ time.
  - Step 8 requires $O(\lg V)$ times. This in total takes $O(E \lg V)$ times.
  - The total runtime is $O((E+V)\lg V)$.

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  $S = \emptyset$
3  $Q = G.V$
4  while $Q \neq \emptyset$
5      $u$ = EXTRACT-MIN($Q$)
6      $S = S \cup \{u\}$
7      for each vertex $v \in G.Adj[u]$
8          RELAX($u, v, w$)
```

All-Pairs Shortest Paths

- Given a weighted, directed graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}$ that maps edges to real-valued weights. We wish to find, for every pair of vertices $u, v \in V$, a shortest (least-weight) path from u to v, where the weight of a path is the sum of the weights of its constituent edges.
- We typically want the output in tabular form: the entry in u's row and v's column should be the weight of a shortest path from u to v.
- We can solve an all-pairs shortest-paths problem by running a single-source shortest-paths algorithm $|V|$ times, once for each vertex as the source.
- If all edge weights are nonnegative, we can use Dijkstra's algorithm.
- If the graph has negative-weight edges, we can use Bellman-Ford algorithm once from each vertex.

All-Pairs Shortest Paths

- For convenience, we assume that the vertices are numbered $1, 2, \ldots, |V|$, so that the input is an $n \times n$ matrix W representing the edge weights of an n-vertex directed graph $G = (V, E)$. That is, $W = (w_{ij})$, where

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

- We allow negative-weight edges, but we assume that *the input graph contains no negative-weight cycles*.
- The tabular output of the all-pairs shortest-paths algorithms is an $n \times n$ matrix $D = (d_{ij})$, where entry d_{ij} contains the weight of a shortest path from vertex i to vertex j. Therefore, $d_{ij} = \delta(i,j))$ at termination.

All-Pairs Shortest Paths

- We use a predecessor matrix $\Pi = (\pi_{ij})$, where $\pi_{ij} = \text{NIL}$ if either $i = j$ or there is no path from i to j, and otherwise π_{ij} is the predecessor of j on some shortest path from i.
- For each vertex $i \in V$, we define the predecessor subgraph of G for i as $G_{\pi,i} = (V_{\pi,i}, E_{\pi,i})$, where

$$V_{\pi,i} = \{j \in V : \pi_{ij} \neq \text{NIL}\} \cup \{i\}$$

and

$$E_{\pi,i} = \{(\pi_{ij}, j) : j \in V_{\pi,i} - \{i\}\}$$

All-Pairs Shortest Paths

```
PRINT-ALL-PAIRS-SHORTEST-PATH($\Pi, i, j$)
1  if $i == j$
2      print $i$
3  elseif $\pi_{ij} ==$ NIL
4      print "no path from" $i$ "to" $j$ "exists"
5  else PRINT-ALL-PAIRS-SHORTEST-PATH($\Pi, i, \pi_{ij}$)
6      print $j$
```

All-Pairs Shortest Paths

The structure of a shortest path

- Suppose that we represent the graph by an adjacency matrix $W = (w_{ij})$.
- Consider a shortest path p from vertex i to vertex j, and suppose that p contains at most m edges. Assuming that there are no negative-weight cycles, m is finite.
- If $i = j$, then p has weight 0 and no edges.
- If vertices i and j are distinct, then we decompose path p into $i \overset{p'}{\rightsquigarrow} k \to j$, where path p' now contains at most $m - 1$ edges.
- Since p' is a shortest path from i to k, and so $\delta(i,j) = \delta(i,k) + w_{kj}$.

All-Pairs Shortest Paths

A recursive solution to the all-pairs shortest-paths problem

- Let $l_{ij}^{(m)}$ be the minimum weight of any path from vertex i to vertex j that contains at most m edges.
- When $m = 0$, there is a shortest path from i to j with no edges if and only if $i = j$. Thus,

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

All-Pairs Shortest Paths

A recursive solution to the all-pairs shortest-paths problem

- For $m \geq 1$, we compute $l_{ij}^{(m)}$ as the minimum of $l_{ij}^{(m-1)}$ (the weight of a shortest path from i to j consisting of at most $m+1$ edges) and the minimum weight of any path from i to j consisting of at most m edges, obtained by looking at all possible predecessors k of j. Thus,

$$\begin{aligned} l_{ij}^{(m)} &= \min\left(l_{ij}^{(m-1)}, \min_{1\leq k\leq n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}\right) \\ &= \min_{1\leq k\leq n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\} \end{aligned}$$

The latter equality follows since $w_{jj} = 0$ for all j.

All-Pairs Shortest Paths

A recursive solution to the all-pairs shortest-paths problem

- If the graph contains no negative-weight cycles, then for every pair of vertices i and j for which $\delta(i,j) \leq \infty$, there is a shortest path from i to j that is simple and thus contains at most $n-1$ edges.
- A path from vertex i to vertex j with more than $n-1$ edges cannot have lower weight than a shortest path from i to j.
- The actual shortest-path weights are therefore given by

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \dots .$$

All-Pairs Shortest Paths

Computing the shortest-path weights bottom up

- Taking as our input the matrix $W = (w_{ij})$, we now compute a series of matrices $L^{(1)}, L^{(2)}, \dots, L^{(n-1)}$, where for $m = 1, 2, \dots, n-1$, we have $L^{(m)} = \left(l_{ij}^{(m)}\right)$. The final matrix $L^{(n-1)}$ contains the actual shortest-path weights.
- Note that, $L^{(1)} = W$.

```
EXTEND-SHORTEST-PATHS$(L, W)$
1  $n = L.rows$
2  let $L' = (l'_{ij})$ be a new $n \times n$ matrix
3  for $i = 1$ to $n$
4      for $j = 1$ to $n$
5          $l'_{ij} = \infty$
6          for $k = 1$ to $n$
7              $l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj})$
8  return $L'$
```

All-Pairs Shortest Paths

Relation to matrix multiplication

- Consider the substitutions: (i) $l_{ij}^{(m-1)} \longrightarrow a$, (ii) $w \longrightarrow b$, (iii) $l_{ij}^{(m-1)} \longrightarrow c$, (iv) $\min \longrightarrow +$, and (iv) $+ \longrightarrow \cdot$.

```
EXTEND-SHORTEST-PATHS$(L, W)$
1  $n = L.rows$
2  let $L' = (l'_{ij})$ be a new $n \times n$ matrix
3  for $i = 1$ to $n$
4      for $j = 1$ to $n$
5          $l'_{ij} = \infty$
6          for $k = 1$ to $n$
7              $l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj})$
8  return $L'$
```

```
SQUARE-MATRIX-MULTIPLY$(A, B)$
1  $n = A.rows$
2  let $C$ be a new $n \times n$ matrix
3  for $i = 1$ to $n$
4      for $j = 1$ to $n$
5          $c_{ij} = 0$
6          for $k = 1$ to $n$
7              $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
8  return $C$
```

All-Pairs Shortest Paths

Relation to matrix multiplication

- Letting $A \cdot B$ denote the matrix "product" returned by $\textsc{Extend-Shortest-Paths}(A, B)$, we compute the sequence of $n-1$ matrices

$$\begin{aligned}
L^{(1)} &= L^{(0)} \cdot W = W \\
L^{(2)} &= L^{(1)} \cdot W = W^2 \\
L^{(3)} &= L^{(2)} \cdot W = W^3 \\
&\vdots \\
L^{(n-1)} &= L^{(n-2)} \cdot W = W^{n-1}
\end{aligned}$$

All-Pairs Shortest Paths

- The following procedure computes this sequence in $\Theta(n^4)$ time.

```
SLOW-ALL-PAIRS-SHORTEST-PATHS$(W)$
1  $n = W.rows$
2  $L^{(1)} = W$
3  for $m = 2$ to $n - 1$
4      let $L^{(m)}$ be a new $n \times n$ matrix
5      $L^{(m)} =$ EXTEND-SHORTEST-PATHS$(L^{(m-1)}, W)$
6  return $L^{(n-1)}$
```

All-Pairs Shortest Paths

Improving the running time

- Letting $A \cdot B$ denote the matrix "product" returned by EXTEND-SHORTEST-PATHS(A, B), we compute the sequence of $n - 1$ matrices (because the "product" is associative)

$$
\begin{aligned}
L^{(1)} &= W \\
L^{(2)} &= W^2 = W \cdot W \\
L^{(4)} &= W^4 = W^2 \cdot W^2 \\
&\vdots \\
L^{2^{\lceil \lg(n-1) \rceil}} &= W^{2^{\lceil \lg(n-1) \rceil}} = W^{2^{\lceil \lg(n-2) \rceil}} \cdot W^{2^{\lceil \lg(n-2) \rceil}}
\end{aligned}
$$

All-Pairs Shortest Paths

- The following procedure computes this sequence in $\Theta(n^3 \lg n)$ time.

```
FASTER-ALL-PAIRS-SHORTEST-PATHS($W$)
$n = W.rows$
$L^{(1)} = W$
$m = 1$
while $m < n - 1$
    let $L^{(2m)}$ be a new $n \times n$ matrix
    $L^{(2m)} = $ EXTEND-SHORTEST-PATHS($L^{(m)}, L^{(m)}$)
    $m = 2m$
return $L^{(m)}$
```

All-Pairs Shortest Paths

Floyd-Warshall algorithm

- Uses a dynamic-programming formulation to solve the all-pairs shortest-paths problem on a directed graph $G = (V, E)$ in $O(n^3)$ time.
- We assume negative-weight edges may be present, but there are no negative-weight cycles.
- Let $d_{ij}^{(k)}$ be the shortest path from vertex i to vertex j for which all intermediate vertices are in the set $\{1, 2, \ldots, k\}$. Then,

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$

Because for any path, all intermediate vertices are in the set $\{1, 2, \ldots, n\}$, the matrix $D^{(n)} = \left(d_{ij}^{(n)}\right)$ gives the final answer: $d_{ij}^{(n)} = \delta(i, j)$ for all $i, j \in V$.

All-Pairs Shortest Paths

Floyd-Warshall algorithm

- Uses a dynamic-programming formulation to solve the all-pairs shortest-paths problem on a directed graph $G = (V, E)$ in $O(n^3)$ time.
- We assume negative-weight edges may be present, but there are no negative-weight cycles.
- Let $d_{ij}^{(k)}$ be the shortest path from vertex i to vertex j for which all intermediate vertices are in the set $\{1, 2, \dots, k\}$. Then,

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$

  Because for any path, all intermediate vertices are in the set $\{1, 2, \dots, n\}$, the matrix $D^{(n)} = \left(d_{ij}^{(n)}\right)$ gives the final answer: $d_{ij}^{(n)} = \delta(i, j)$ for all $i, j \in V$.

All-Pairs Shortest Paths

Floyd-Warshall algorithm

```
FLOYD-WARSHALL($W$)
1  $n = W.rows$
2  $D^{(0)} = W$
3  for $k = 1$ to $n$
4      let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
5      for $i = 1$ to $n$
6          for $j = 1$ to $n$
7              $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
8  return $D^{(n)}$
```

All-Pairs Shortest Paths

Floyd-Warshall algorithm

- We compute a sequence of predecessor matrices $\Pi^{(0)}, \Pi^{(1)}, \Pi^{(2)}, \ldots, \Pi^{(n)}$, where $\Pi = \Pi^{(0)}$ is the final predecessor matrix.
- We define $\pi_{ij}^{(k)}$ as the predecessor of vertex j on a shortest path from vertex i with all intermediate vertices in the set $\{1, 2, \ldots, k\}$.
- We define

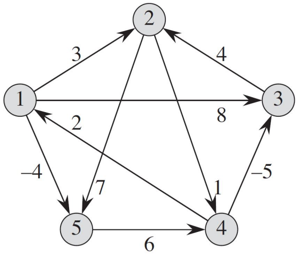
$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty. \end{cases}$$

and

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$

All-Pairs Shortest Paths

Floyd-Warshall algorithm



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

All-Pairs Shortest Paths

Floyd-Warshall algorithm

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

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

All-Pairs Shortest Paths

Floyd-Warshall algorithm

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

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

All-Pairs Shortest Paths

Floyd-Warshall algorithm

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

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

All-Pairs Shortest Paths

Floyd-Warshall algorithm

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

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

All-Pairs Shortest Paths

Floyd-Warshall algorithm

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

White Board

White Board