Shortest Paths in Weighted Graphs

Data Structures and Algorithms (094224)

Yuval Emek

Winter 2022/23

Distances revisited

- Digraph $G = (V, E)$
  - Some of the material in this lecture applies to undirected graphs as well
- Edge *weight* ⟨משקל⟩ function $w : E \to \mathbb{R}$
  - Associates a real weight $w(e)$ with each edge $e \in E$
  - G is called a *weighted* ⟨ממושקל⟩ digraph
- Generalize to edge subsets $F \subseteq E$: $w(F) = \sum_{e \in F} w(e)$
- Weight of path $P = \langle v_0, v_1, \ldots, v_k \rangle$ is $w(P) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$
  - A.k.a. (weighted) length of P
- Redefine the distance from $u \in V$ to $v \in V$ w.r.t. w:

$$\delta(u, v) = \begin{cases} \min\left\{ w(P) : u \overset{P}{\leadsto} v \right\}, & u \leadsto v \\ \infty, & u \not\leadsto v \end{cases}$$

  - $u \leadsto v$ denotes v reachable from u
  - $u \overset{P}{\leadsto} v$ denotes path P from u to v
- P is a shortest path from u to v if $w(P) = \delta(u, v)$
- The *unweighted* ⟨לא ממושקל⟩ case: $w(e) = 1$ for every $e \in E$

Negative weights

- Are the distances well defined in the presence of negative weights?
- What if G admits a negative weight cycle C?
  - $\delta(u, v)$ is unbounded (from below) for every $u \rightsquigarrow C \rightsquigarrow v$
    - Can make the path arbitrarily short by including more traversals of C
  - Define $\delta(u, v) = -\infty$
- Some shortest paths algorithms assume that the edge weights are non-negative
- Other shortest paths algorithms allow negative weight edges but forbid negative weight cycles
- If G doesn't admit negative weight cycles, then we can restrict our attention to simple shortest paths
  - Does it mean that every shortest path is necessarily simple?

yes, but there can be (kayyam) not simple path if there is a cycle with 0 weight(we can still find shortest simple path).

Subpaths of shortest paths are shortest

this lemma is true to (di/un)graphs

Lemma (subpaths of shortest paths)

Consider a shortest path $P = \langle v_0, v_1, \ldots, v_k \rangle$. Then, $P_{i,j} = \langle v_i, \ldots, v_j \rangle$ is a shortest path for every $0 \leq i \leq j \leq k$.

- Assume by contradiction: $w(Q) < w(P_{i,j})$ for some (v_i, v_j)-path Q
- Let $P' = v_0 \overset{P_{0,i}}{\rightsquigarrow} v_i \overset{Q}{\rightsquigarrow} v_j \overset{P_{j,k}}{\rightsquigarrow} v_k$
- $w(P') = w(P_{0,i}) + w(Q) + w(P_{j,k}) = w(P) - w(P_{i,j}) + w(Q) < w(P)$
- P is not a shortest (v_0, v_k)-path $_{(\rightarrow\leftarrow)}$ $\blacksquare$

Representing shortest paths

- Focus: single source shortest paths (SSSP) algorithm
  - Designated source vertex $s \in V$
  - Goal: compute the distances and shortest paths from s to all vertices reachable from s
    - BFS does so for unweighted (di)graphs
- A shortest paths tree
  - Tree rooted at s
  - Contains all vertices reachable from s
  - Unique simple (s, v)-path in the tree is a shortest (s, v)-path in G

The structure of the algorithms

- The SSSP algorithms we will see have many common features
- Additional attributes for each vertex $v \in V$:
  - $v.d$ = the distance from s to v in G
  - $v.\pi$ = the parent of v in the shortest paths tree
- The (directed) predecessor subgraph $G_\pi = (V_\pi, E_\pi)$
  - $V_\pi = \{v \in V \mid v.\pi \neq NIL\} \cup \{s\}$
  - $E_\pi = \{(v.\pi, v) \in E \mid v \in V_\pi - \{s\}\}$
  - At termination: (its undirected version is) a shortest paths tree for s
- Algorithm's structure:
  1. Initialize the d and π fields by calling procedure `Initialize_Single_Source`
  2. Update the d and π fields through calls to procedure `Relax`
     - d and π are not modified otherwise

The initialization procedure

Initialize the d and π fields in G

`Initialize_Single_Source`(G, s)

for all $v \in G.V$ **do**
  $v.d = \infty$
  $v.\pi = NIL$
$s.d = 0$

- Run-time: $O(n)$

The triangle inequality

Lemma (triangle inequality)

For every edge $(u, v) \in E$, *we have* $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

- Case $\delta(s, v) = \infty$: $s \not\rightsquigarrow v$, thus $s \not\rightsquigarrow u$ and $\delta(s, u) = \infty$
- Case $\delta(s, u) = -\infty$: there exists a negative weight cycle C such that $s \rightsquigarrow C \rightsquigarrow u$, thus $s \rightsquigarrow C \rightsquigarrow v$ and $\delta(s, v) = -\infty$
- Assume that $-\infty < \delta(s, u), \delta(s, v) < \infty$
- Let Q be a shortest (s, u)-path
- Let $P = s \overset{Q}{\rightsquigarrow} u \rightarrow v$
- $w(P) = w(Q) + w(u, v) = \delta(s, u) + w(u, v)$
- P is a candidate for a shortest (s, v)-path, thus $\delta(s, v) \leq w(P)$ ■

Procedure `Relax`

Try to improve the shortest path to v through the edge (u, v)

`Relax`(u, v, w)

if $v.d > u.d + w(u, v)$ **then**
    $v.d = u.d + w(u, v)$ triangle inequality
    $v.\pi = u$

- Run-time: $O(1)$
- Improvements based on "realizing the triangle inequality"
- The basic operation of our algorithms
  - The only means by which d and π are modified after initialization

The monotonicity property

Lemma (monotonicity)

The value of $v.d$ is non-increasing over time.

- Set to $+\infty$ during initialization
- Can only decrease during calls to procedure `Relax` ■

The upper bound property

Lemma (upper bound)

$v.d \geq \delta(s, v)$ at all times.

- Proof by induction on the sequence of calls to procedure `Relax`
- Base: immediately after initialization, $v.d = \infty \geq \delta(s, v)$ for all $v \in V - \{s\}$ and $s.d = 0 \geq \delta(s, s)$
- Step: consider some call to $\texttt{Relax}(u, v, w)$
- The only d value that may change during the call is $v.d$
- If it changes, then after the call

$$
\begin{aligned}
v.d &= u.d + w(u, v) && \\
&\geq \delta(s, u) + w(u, v) && \text{ind. hyp.} \\
&\geq \delta(s, v) && \text{triangle inequality} \blacksquare
\end{aligned}
$$

The no path property

Corollary (no path)

If v is not reachable from s, then $v.d = \infty$ at all times.

- By the upper bound property, $v.d \geq \delta(s, v) = \infty$ at all times ■

The convergence property

Lemma (convergence)

If $s \rightsquigarrow u \rightarrow v$ is a shortest (s, v)-path and $u.d = \delta(s, u)$ prior to some call to $\texttt{Relax}(u, v, w)$, then $v.d = \delta(s, v)$ at all times after the call.

- Right after the call

$$
\begin{aligned}
v.d &\leq u.d + w(u, v) \\
&= \delta(s, u) + w(u, v) && \text{assumption on } u.d \\
&= \delta(s, v) && s \rightsquigarrow u \rightarrow v \text{ is shortest}
\end{aligned}
$$

  - By the upper bound property, inequality cannot be strict
- Subsequently, the value of $v.d$ doesn't change by the monotonicity and upper bound properties ■

The path relaxation property

Lemma (path relaxation)

Consider some path $P = \langle s = v_0, v_1, \dots, v_k \rangle$. If the sequence of calls to `Relax` *includes as a subsequence*[a] *the calls* `Relax`(v_0, v_1, w), `Relax`$(v_1, v_2, w), \dots,$ `Relax`(v_{k-1}, v_k, w), *then* $v_k.d \leq w(P)$ *at all times after this subsequence of calls.*

[a] Not necessarily contiguous.

- Prove by induction on $i = 0, 1, \dots, k$ that after edge (v_{i-1}, v_i) is relaxed, $v_i.d \leq \sum_{j=1}^{i} w(v_{j-1}, v_j)$
- Base: after initialization, $v_0.d = s.d = 0$ and this value doesn't increase by the monotonicity property
- Step: consider the call to `Relax`(v_i, v_{i+1}, w)
  - By the ind. hyp., $v_i.d \leq \sum_{j=1}^{i} w(v_{j-1}, v_j)$ before the call
  - `Relax` ensures that $v_{i+1}.d \leq v_i.d + w(v_i, v_{i+1})$ right after the call
  - Doesn't increase by the monotonicity property ∎

The shortest path relaxation property

Corollary (shortest path relaxation)

Consider some shortest path $P = \langle s = v_0, v_1, \ldots, v_k \rangle$. If the sequence of calls to `Relax` *includes as a subsequence*[a] *the calls* $\texttt{Relax}(v_0, v_1, w), \texttt{Relax}(v_1, v_2, w), \ldots, \texttt{Relax}(v_{k-1}, v_k, w)$, *then* $v_k.d = \delta(s, v_k)$ *at all times after this subsequence of calls.*

[a] Not necessarily contiguous.

- By the path relaxation property, $v_k.d \leq w(P) = \delta(s, v_k)$ at all times after the subsequence
- The assertion follows by the upper bound property ■

The predecessor subgraph

Lemma

If G is free of negative weight cycles reachable from s, then the undirected version of G_π forms a tree at all times.

Prove:

1. G_π is cycle free
2. The undirected version of G_π is cycle free
3. If $v \in V_\pi$, then v is reachable from s in G_π

G_π is cycle free

- Assume by contradiction: G_π admits cycle $C = \langle v_0, v_1, \ldots, v_k = v_0 \rangle$
- W.l.o.g.: (v_{k-1}, v_k) is the last edge of the cycle to join E_π
  - Happened by relaxing the edge at some time t^* the time C was created in G_pi
- Claim 1: $s \rightsquigarrow C$
  - Since $v_i.\pi \neq NIL$, we know that $v_i.d < \infty$, hence $s \rightsquigarrow v_i$ by the no path property
- Claim 2: right before time t^*, $v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i)$ for every $1 \leq i \leq k-1$
  - $v_i.\pi$ was set to v_{i-1} during a call to $\texttt{Relax}(v_{i-1}, v_i, w)$ before time t^*
  - Right after this call, we had $v_i.d = v_{i-1}.d + w(v_{i-1}, v_i)$
  - $v_i.d$ didn't change since then because $v_i.\pi$ didn't
  - By the monotonicity property, since then, $v_{i-1}.d$ could only decrease
- Claim 3: right before time t^*, $v_k.d > v_{k-1}.d + w(v_{k-1}, v_k)$
  - The if condition (line 1) was satisfied during the call at time t^*

G_π is cycle free — cont.

- Combining Claims 2 and 3: right before time t^*,

$$\sum_{i=1}^{k} v_i.d > \sum_{i=1}^{k} (v_{i-1}.d + w(v_{i-1}, v_i)) = \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

v_k=v_0
- Since $\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$, we get

$$w(C) = \sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$$

  - $\Longrightarrow$ A negative weight cycle reachable from s (by Claim 1) $_{(\rightarrow\leftarrow)}$

The undirected version of G_π is cycle free

- Assume by contradiction:
  undirected version of G_π admits a simple cycle C
  - $k = \#$vertices in C $(= \#$edges in $C)$
- Consider the digraph obtained from C by directing the edges according to G_π
- For every vertex v, $\deg_{\text{in}}(v) + \deg_{\text{out}}(v) = 2$
- Sum of in-degrees = sum of out-degrees = k
  - $\Longrightarrow$ Average in-degree = average out-degree = 1
- C is not a (directed) cycle in G_π, hence not all vertices v satisfy $\deg_{\text{in}}(v) = \deg_{\text{out}}(v) = 1$
  - $\Longrightarrow$ There must exist a vertex v with $\deg_{\text{in}}(v) > 1$
- In contradiction to the construction of G_π $_{(\rightarrow\leftarrow)}$

If $v \in V_\pi$, then v is reachable from s in G_π

- Starting with $v = v_0$, we tour the graph by moving from v_i to $v_{i+1} = v_i.\pi$
- This tour must stop at some vertex v_k with $v_k.\pi = NIL$
  - Otherwise, we reached a (directed) cycle in G_π — proved already that this can't be
- Assume by contradiction: $v_k \neq s$
- $v_{k-1}.\pi$ was set to v_k by calling $\texttt{Relax}(v_k, v_{k-1}, w)$
- At the time of this call $v_k.d$ must have been $< \infty$
- $\implies$ At that time, $v_k.\pi$ must have been $\neq NIL$ and it remained $\neq NIL$ ever since $_{(\rightarrow\leftarrow)}$
- Therefore $v_k = s$ and v is reachable from s in G_π $\blacksquare$

The shortest paths property

Lemma (shortest paths)

Suppose that G is free of negative weight cycles reachable from s. At all times, if $v.d = \delta(s, v) < \infty$, then the unique (s, v)-path in G_π is a shortest (s, v)-path in G.

- Let t^v be the time at which $v.d$ is set to $v.d = \delta(s, v)$
- By the monotonicity and upper bound properties, $v.d$ is not updated after time t^v, hence $v.\pi$ is not updated after time t^v
- Claim: if $v.\pi$ is set to u at time t^v, then $t^u < t^v$ and $\delta(s, v) = \delta(s, u) + w(u, v)$
  - `Relax`(u, v, w) is called at time t^v and at that time $v.d$ is set to

$$\begin{aligned} \delta(s, v) = v.d &= u.d + w(u, v) & \\ &\geq \delta(s, u) + w(u, v) & \text{upper bound property} \\ &\geq \delta(s, v) & \text{triangle inequality} \end{aligned}$$

  - $\Longrightarrow$ the inequalities are not strict, establishing the claim

The shortest paths property — cont.

- Let $P_v(t)$ be the unique (s, v)-path in G_π at time t
  - Assuming that $v \in V_\pi$ at time t
- Prove by induction on the times t^v: $\mathrm{len}(P_v(t^v)) = \delta(s, v)$ and $P_v(t) = P_v(t^v)$ for all $t \geq t^v$
- Base: the empty path is the unique (s, s)-path in G_π at all times and its length is indeed 0
- Step: consider some v whose $v.\pi$ field is set to $v.\pi = u$ at time t^v
- By the claim, $t^u < t^v$ and $\delta(s, v) = \delta(s, u) + w(u, v)$
- By the ind. hyp., $\mathrm{len}(P_u(t^u)) = \delta(s, u)$ and $P_u(t) = P_u(t^u)$ for all $t \geq t^u$

The shortest paths property — cont.

- Hence,

$$\begin{aligned}
\mathrm{len}(P_v(t^v)) &= \mathrm{len}(P_u(t^v)) + w(u,v) \\
&= \mathrm{len}(P_u(t^u)) + w(u,v) && t^v > t^u \\
&= \delta(s,u) + w(u,v) \\
&= \delta(s,v)
\end{aligned}$$

- $v.\pi$ is not updated after time t^v, therefore $P_v(t) = P_v(t^v)$ for all $t \geq t^v$ ■

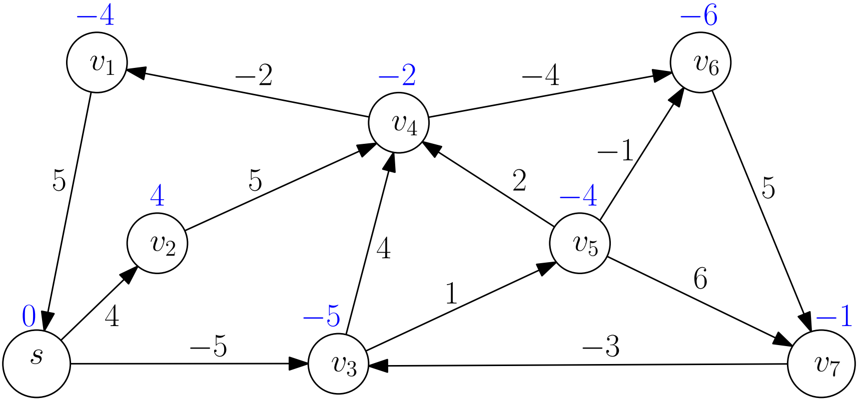
The Bellman-Ford algorithm

- Input:
  - Digraph $G = (V, E)$
  - Weight function $w : E \to \mathbb{R}$
  - Source vertex $s \in V$
- Outputs an error message if G admits a negative weight cycle reachable from s
- Computes (if no error):
  - $\delta(s, v)$ for each vertex $v \in V$
  - Shortest paths tree rooted at s

Pseudocode

```
Bellman_Ford(G, w, s)
Initialize_Single_Source(G, s)
for i = 1, ..., |G.V| − 1 do
    for all (u, v) ∈ G.E do
        Relax(u, v, w)
for all (u, v) ∈ G.E do
    if v.d > u.d + w(u, v) then
        error: "negative weight cycle"
```

Example



Run-time analysis

- The initialization takes $O(n)$ time
- The for loop of lines 2–4 makes $O(n)$ iterations
- The for loops of lines 3–4 and lines 5–7 make $O(m)$ iterations
- Each call to `Relax` takes $O(1)$ time
- $O(mn)$ time in total
- Exercise: show that this is asymptotically tight

Correctness of the distances

Lemma

If G is free of negative weight cycles reachable from s, then upon termination, $v.d = \delta(s, v)$ for every vertex $v \in V$.

- Holds for vertices v not reachable from s by the no path property
- Consider a simple shortest (s, v)-path $\langle s = v_0, v_1, \ldots, v_k = v \rangle$
  - Simple path: $k \leq n - 1$
- Edge (v_{i-1}, v_i) is relaxed in iteration i of the for loop of lines 2–4
- The assertion follows by the shortest path relaxation property ■

Correctness of the shortest paths tree

Corollary

If G is free of negative weight cycles reachable from s, then upon termination, (the undirected version of) G_π is a shortest paths tree rooted at s.

- Combine the previous lemma with the shortest paths property ■

Detecting negative weight cycles

Lemma

`Bellman_Ford`(G, s, w) *outputs an error message iff* G *admits a negative weight cycle reachable from* s.

- Suppose that G is free of negative weight cycles reachable from s
- Already proved: upon termination of the for loop of lines 2–4, $v.d = \delta(s, v)$ for every $v \in V$
- Therefore, for every edge $(u, v) \in E$,

$$\begin{aligned} v.d &= \delta(s, v) \\ &\leq \delta(s, u) + w(u, v) && \text{triangle inequality} \\ &= u.d + w(u, v) \end{aligned}$$

- $\Longrightarrow$ The if condition in line 6 fails for all edges

Detecting negative weight cycles — cont.

- Suppose that G admits a negative weight cycle $C = \langle v_0, v_1, \ldots, v_k = v_0 \rangle$ reachable from s
- $s \rightsquigarrow v_i$ implies the existence of a simple path $s \overset{P}{\rightsquigarrow} v_i$ for every $1 \leq i \leq k$
  - P contains $\leq n - 1$ edges
- By the path relaxation property, upon termination of the for loop of lines 2–4, $v_i.d < \infty$ for every $1 \leq i \leq k$
- Assume by contradiction: if condition in line 6 fails for all edges in C
  - $\Longrightarrow$ upon termination of the for loop of lines 2–4, $v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$ for every $1 \leq i \leq k$
- Summing over all *i*s: $\sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$
- Since $\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$, we conclude that $\sum_{i=1}^{k} w(v_{i-1}, v_i) \geq 0$ $_{(\rightarrow\leftarrow)}$ $\blacksquare$

The Dijkstra algorithm

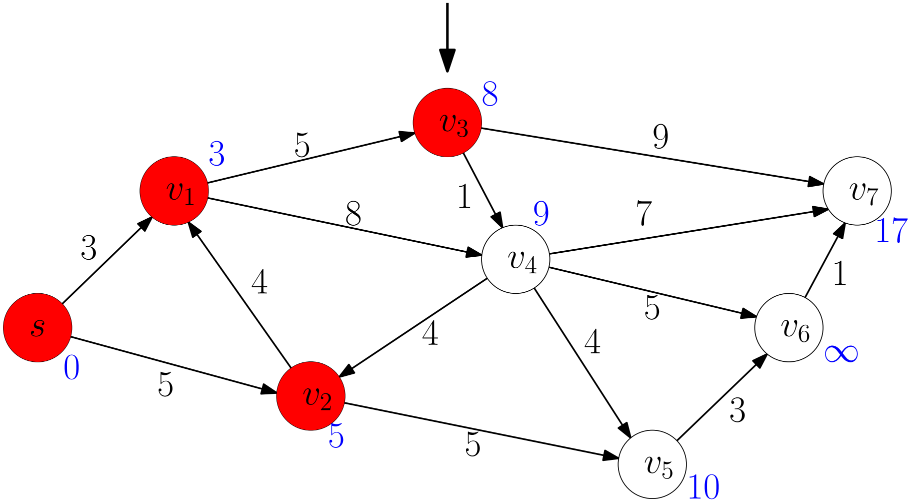
- Input:
  - Digraph $G = (V, E)$
  - Weight function $w : E \to \mathbb{R}_{\geq 0}$
    - Note the non-negative edge weights
  - Source vertex $s \in V$
- Computes:
  - $\delta(s, v)$ for each vertex $v \in V$
  - Shortest paths tree rooted at s
- Works also for undirected graphs

Pseudocode

```
Dijkstra(G, w, s)
1: Initialize_Single_Source(G, s)
2: Q = G.V
3: while Q ≠ ∅ do
4:     u = Extract_Min(Q)                    ▷ minimum w.r.t. d
5:     for all v ∈ G.Adj[u] do
6:         Relax(u, v, w)
```

- Resemblance to BFS

Example



Correctness of the distances

Lemma

When vertex u is extracted from Q (line 4), it holds that $u.d = \delta(s, u)$.

- Let t^v be the time just before $v \in V$ is extracted from Q (line 4)
- Assume by contradiction: the lemma is false for some vertex $u \in V$
- Let u be the vertex that minimizes t^u among all vertices with $u.d \neq \delta(s, u)$ at time t^u
  - $u.d > \delta(s, u)$ at time t^u by the upper bound property
- $u \neq s$ because s is the first to be extracted from Q with $s.d = 0 = \delta(s, s)$
- $s \rightsquigarrow u$ because otherwise $\delta(s, u) = \infty \not< u.d$
- Let P be a shortest (s, u)-path

Correctness of the distances — cont.

- At time t^u, P leads from a vertex in $V - Q$ to a vertex in Q
  - Decompose $P = s \rightsquigarrow x \to y \rightsquigarrow u$ so that $x \notin Q$ and $y \in Q$ at time t^u
- Since $x \notin Q$ at time t^u, it follows that $t^x < t^u$
- By the choice of u, we know that $x.d = \delta(s, x)$ at time t^x
- Edge (x, y) is relaxed soon after time t^x, so $y.d = \delta(s, y)$ at all times afterwards by the convergence property
- At time t^u, we have

$$\begin{aligned} y.d &= \delta(s, y) && t^u > t^x \\ &\leq \delta(s, u) && y \text{ precedes } u \text{ along } P \\ &< u.d && \text{assumption on } u \end{aligned}$$

- $\Longrightarrow$ y should have been selected in line 4 at time t^u $_{(\to\leftarrow)}$ ■

Correctness of the shortest paths tree

Corollary

Upon termination, (the undirected version of) G_π is a shortest paths tree rooted at s.

- Combine the previous lemma with the shortest paths property ■

Run-time analysis

- The initialization takes $O(n)$ time
- Each vertex is extracted from Q (line 4) exactly once
- $\Longrightarrow$ Each edge is relaxed (line 6) exactly once
- A naive implementation:
  - Each call to `Extract_Min` takes $O(n)$ time
  - Run-time: $O(n^2 + m) = O(n^2)$
- Implementing Q using a binary heap:
  - Each call to `Extract_Min` takes $O(\log n)$ time
  - Each call to `Decrease_Key` (hidden in line 6) takes $O(\log n)$ time
  - Run-time: $O((n + m) \log n)$
- Implementing Q using a Fibonacci heap:
  - Run-time: $O(n \log n + m)$
  - Beyond the scope of this course

Computing all pairs shortest paths

- Goal: given a digraph $G = (V, E)$ with edge weight function $w : E \to \mathbb{R}$, compute the distance (and a shortest path) from u to v for all $(u, v) \in V \times V$
- No negative weight edges: invoke Dijkstra for every source vertex
  - Run-time: $O(n^2 \log n + nm)$
- Negative weight edges: invoke Bellman-Ford for every source vertex
  - Run-time: $O(n^2 m)$
  - Can do better with dynamic programming...

The Floyd-Warshall algorithm

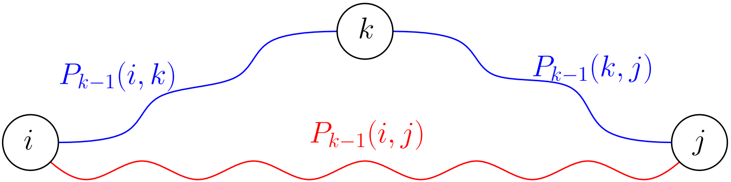
- Input:
  - Digraph $G = (V, E)$
    - $V = [n] = \{1, \ldots, n\}$
  - Weight function $w : E \to \mathbb{R}$
    - No negative weight cycles
    - $w(i,j) = \infty$ if $(i,j) \notin E$
- Computes:
  - The distance $\delta(i,j)$ for every $i, j \in [n]$
  - The successor $s(i,j)$ of i along a shortest (i,j)-path for every $i, j \in [n]$

Intermediate vertices

- v_i is an *intermediate vertex* of path $\langle v_1, \ldots, v_k \rangle$ if $1 < i < k$
- Let $P_k(i,j)$ be a shortest (i,j)-path that uses only vertices in $[k]$ as intermediate vertices
  - Not necessarily all vertices in $[k]$
- Define $\delta_k(i,j) = w(P_k(i,j))$
- Recursive formula:

$$\delta_k(i,j) = \begin{cases} w(i,j), & k = 0 \\ \min\left\{\delta_{k-1}(i,j), \delta_{k-1}(i,k) + \delta_{k-1}(k,j)\right\}, & k > 0 \end{cases}$$

  - Construct $P_k(i,j)$ by taking $P_{k-1}(i,j)$ or $P_{k-1}(i,k) \circ P_{k-1}(k,j)$



Pseudocode

$\texttt{Floyd} - \texttt{Warshall}(G, w)$

```
n = |G.V|
new n × n table D_0[1...n, 1...n]
initialize D_0 so that D_0[i, j] = w(i, j)
for k = 1, ..., n do
    new n × n table D_k[1...n, 1...n]
    for i = 1, ..., n do
        for j = 1, ..., n do
            D_k[i, j] = min {D_{k-1}[i, j], D_{k-1}[i, k] + D_{k-1}[k, j]}
return D_n
```

Retrieving the actual paths

- The aforementioned pseudocode computes the distances (optimal values), but not the shortest paths (optimal solutions)
- Interested in the *successor matrix* $S \in V^{n \times n}$ so that $S[i, j] = s(i, j)$
- Can be constructed by modifying the algorithm to keep track of table updates
  - Often the case in dynamic programming algorithms
- S can also be computed from D_n in time $O(n^3)$ in a black-box fashion
  - Exercise

Run-time analysis

- Initialization takes $O(n^2)$ time
- Line 8 takes $O(1)$ time
- Three nested for loops, each with n iterations
- Total run-time: $O(n^3)$

Space analysis

- The algorithm allocates n tables, each of size $n \times n$
  - $O(n^3)$ space
- Notice: during iteration k of the outer-most for loop, we only need table D_{k-1}
  - Tables $D_1, \ldots, D_{k-2}$ can be deleted
- Space decreases to $O(n^2)$
  - Is it asymptotically optimal?