Shortest Paths

Dijkstra Bellman-Ford Floyd All-pairs paths

Lecturer: Georgy Gimel'farb

COMPSCI 220 Algorithms and Data Structures

Paths and Distances Revisited

Cost of a walk / path $v_0, v_1, \ldots, v_k$ in a digraph $G = (V, E)$ with edge weights $\{c(u, v) \mid (u, v) \in E\}$:

$$\mathrm{cost}(v_0, v_1, \ldots, v_k) = \sum_{i=0}^{k-1} c(v_i, v_{i+1})$$

Distance $d(u, v)$ between two vertices u and v of $V(G)$: the minimum cost of a path between u and v.

Eccentricity of a node $u \in V$: $\mathrm{ec}[u] = \max_{v \in V} d(u, v)$.

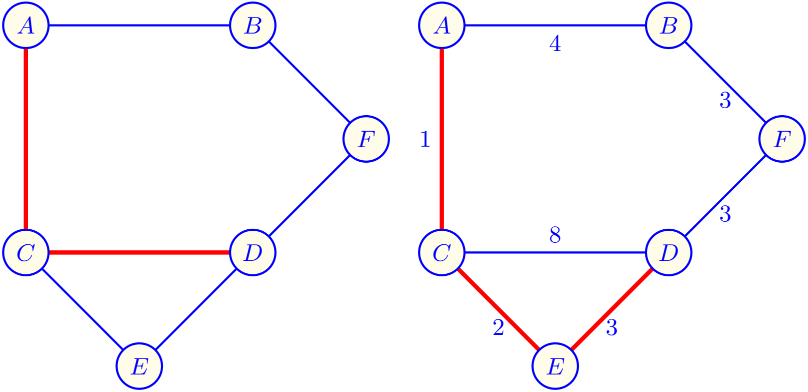
Radius of G: the minimum eccentricity of $u \in V$: $\min_{u \in V} \mathrm{ec}[u]$.

Diameter of G: the maximum eccentricity of $u \in V$: $\max_{u \in V} \mathrm{ec}[u]$.

Note: there are analogous definitions for graphs.

Unweighted / Weighted Graphs: Shortest Paths

The shortest path from the vertex A to the vertex D:



$\min\{2_{A,C,D}, 3_{A,C,E,D}, 3_{A,B,F,D}\}$ $\min\{9_{A,C,D}, 6_{A,C,E,D}, 10_{A,B,F,D}\}$

Single-source Shortest Path (SSSP) in $G = (V, E, c)$

Given a source node v, find the shortest (minimum weight) path to each other node.

- Weight of a path: the sum of weights (costs) on the arcs.
- BFS works only if all weights $c(u, v)$; $(u, v) \in E$, are equal.
- **Dijkstra's algorithm** – one of the known solutions.
  - A **greedy** algorithm: each locally best choice is globally best.
  - Works only if all weights are non-negative.
  - Initial paths: one-arc paths from s to v of weight $\mathrm{cost}(s, v)$.
  - Each step compares the shortest paths with and without each new node.

Single-source Shortest Path (SSSP) in $G = (V, E, c)$

1. Build a list S of visited nodes (say, using a priority queue).
2. Iterative propagation of the shortest paths:
    1. Choose the closest unvisited node u being on a path with internal nodes in S.
    2. If adding the node u has established shorter paths, update distances of remaining unvisited nodes v from the source s.

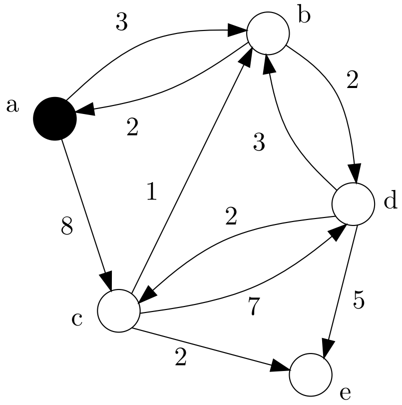
Complexity depends on data structures used.

- For a priority queue, such as a binary heap, running time $\mathrm{O}((m+n)\log n)$ is possible.
    - If every node is reachable from the source: $\mathrm{O}(m \log n)$.
- More sophisticated Fibonacci heaps lead to the best complexity of $\mathrm{O}(m + n \log n)$.

Dijkstra's Algorithm

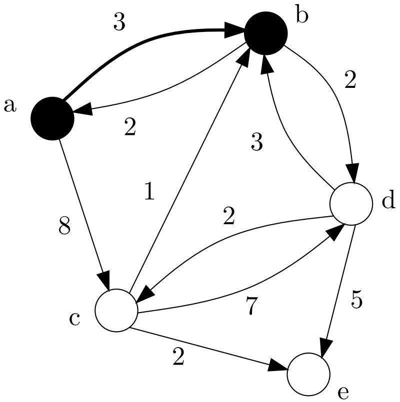
algorithm `Dijkstra`(weighted digraph (G, c), node $s \in V(G)$)
  array $colour[n] = \{\text{WHITE}, \dots, \text{WHITE}\}$
  array $dist[n] = \{c[s, 0], \dots, c[s, n-1]\}$
  $colour[s] \leftarrow \text{BLACK}$
  while there is a WHITE node **do**
    pick a WHITE node u, such that $dist[u]$ is minimum
    $colour[u] \leftarrow \text{BLACK}$
    for each x adjacent to u **do**
      if $colour[x] = \text{WHITE}$ **then**
        $dist[x] \leftarrow \min\left\{dist[x], dist[u] + c[u, x]\right\}$
      end if
    end for
  end while
  return $dist$
end

Dijkstra's Algorithm: Example 1



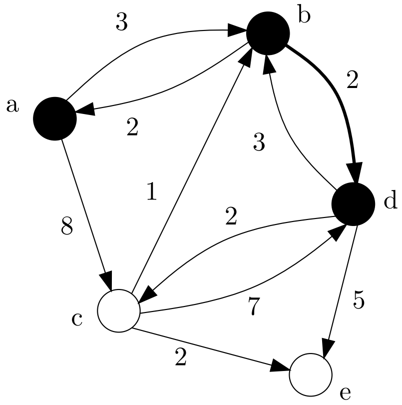
BLACK List S	$dist[x]$				
	a	b	c	d	e
a	0	3	8	∞	∞
$a\ b$	0	3	8	5	∞
$a\ b\ d$	0	3	7	5	10
$a\ b\ c\ d$	0	3	7	5	9
$a\ b\ c\ d\ e$	0	3	7	5	9

Dijkstra's Algorithm: Example 1



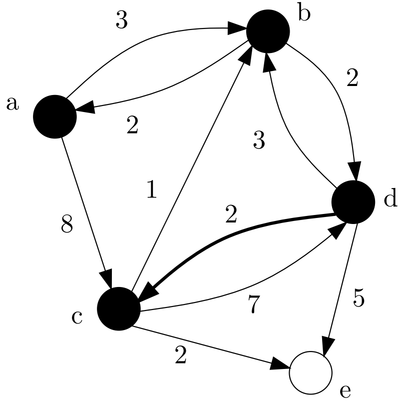
BLACK List S	$dist[x]$				
	a	b	c	d	e
a	0	3	8	∞	∞
$a\ b$	0	3	8	5	∞
$a\ b\ d$	0	3	7	5	10
$a\ b\ c\ d$	0	3	7	5	9
$a\ b\ c\ d\ e$	0	3	7	5	9

Dijkstra's Algorithm: Example 1



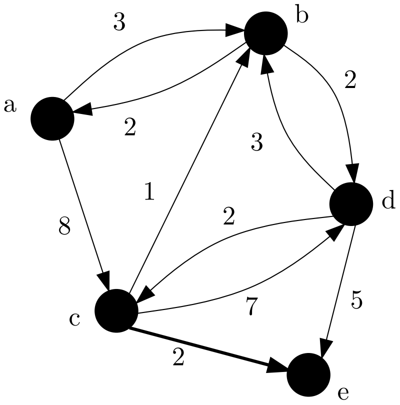
BLACK List S	$dist[x]$				
	a	b	c	d	e
a	0	3	8	∞	∞
$a\ b$	0	3	8	5	∞
$a\ b\ d$	0	3	7	5	10
$a\ b\ c\ d$	0	3	7	5	9
$a\ b\ c\ d\ e$	0	3	7	5	9

Dijkstra's Algorithm: Example 1



BLACK List S	$dist[x]$				
	a	b	c	d	e
a	0	3	8	∞	∞
$a\ b$	0	3	8	5	∞
$a\ b\ d$	0	3	7	5	10
$a\ b\ c\ d$	0	3	7	5	9
$a\ b\ c\ d\ e$	0	3	7	5	9

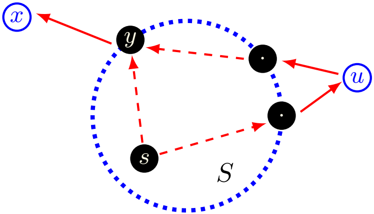
Dijkstra's Algorithm: Example 1



BLACK List S	$dist[x]$				
	a	b	c	d	e
a	0	3	8	∞	∞
$a\ b$	0	3	8	5	∞
$a\ b\ d$	0	3	7	5	10
$a\ b\ c\ d$	0	3	7	5	9
$a\ b\ c\ d\ e$	0	3	7	5	9

Why Does Dijkstra's Algorithm Work?

Let an S-**path** be a path starting at node s and ending at node x with all the intermediate nodes coloured BLACK, i.e., from the list S, except possibly x.



Theorem 6.8: Suppose that all arc weights are nonnegative.

Then these two properties hold at the top of **while**-loop:

P1: If $x \in V(G)$, then $dist[x]$ is the minimum cost of an S-path from s to x.

P2: If $colour[w] = $ BLACK (i.e., $w \in S$), then $dist[w]$ is the minimum cost of a path from s to w.

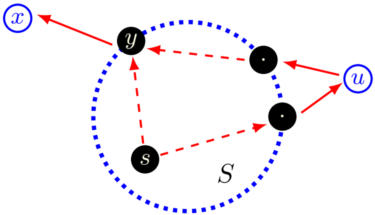
Once a node u is added to S and $dist[u]$ is updated, $dist[u]$ never changes in subsequent steps. After $S = V$, $dist$ holds the goal shortest distances.

Proving Why Dijkstra's Algorithm Works

$\boxed{\text{The update rule: } dist[x] \leftarrow \min\left\{dist[x],\ dist[u] + c[u,x]\right\}}$.

$dist[x]$ is the length of some path from s to x at every step.

- If $x \in S$, then it is an S-path.
- Updated $dist[v]$ never increases.



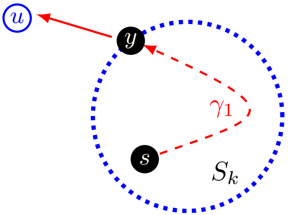
To prove P1 and P2: induction on the number of times k of going through the while-loop (S_k; $S_0 = \{s\}$; $dist[s] = 0$).

- $k = 0$: P1 and P2 hold as $dist[s] = 0$.
- Inductive hypothesis: P1 and P2 hold for $k \geq 0$; $S_{k+1} = S_k \bigcup \{u\}$.
- Inductive steps for P2 and P1:
  - Consider any s-to-w S_{k+1}-path $\gamma = (s, \ldots, y, u)$ of the weight $|\gamma|$.
  - If $w \in S_k$, consider the hypothesis.
  - If $w \notin S_k$, γ extends some s-to-y S_k-path $\gamma_1 = (s, \ldots, y)$.

Proving Why Dijkstra's Algorithm Works

Inductive step for P2:

- For $w \in S_{k+1}$ and $w \neq u$, P2 holds by inductive hypothesis.
- For $w = u$, P2 holds, too, because any S_{k+1}-path $\gamma = (s, \ldots, y, u)$ of weight $|\gamma|$ extends some S_k-path $\gamma_1 = (s, \ldots, y)$ of weight $|\gamma_1|$:
    - By the inductive hypothesis, $dist[y] \leq |\gamma_1|$.
    - By the update rule, $dist[u] \leq dist[y] + c(y, u)$.
    - Therefore, $dist[u] \leq |\gamma| = |\gamma_1| + c(y, u)$.



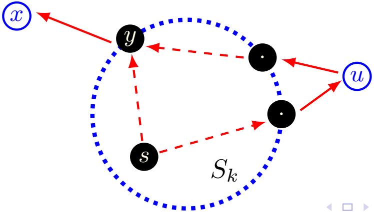
Proving Why Dijkstra's Algorithm Works

Inductive step for P1: $x \in V(G)$; γ – any s-to-x S_{k+1}-path;
$S_{k+1} = S_k \bigcup \{u\}$:

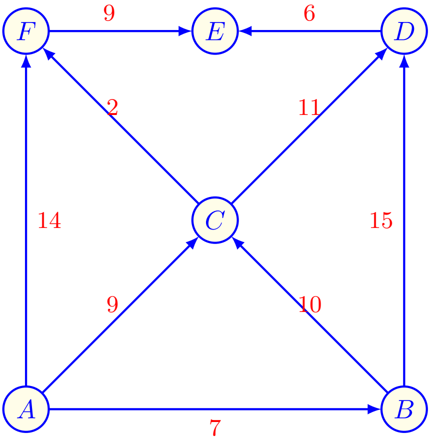
- $u \notin \gamma$: γ is an S_k-path and $|\gamma| \leq dist[x]$ by the inductive hypothesis.
- $u \in \gamma = (\overbrace{s, \ldots, u}^{\gamma_1}, x)$: by the update rule, $|\gamma| = |\gamma_1| + c(u,x) \geq dist[x]$.
- $u \in \gamma = (\overbrace{s, \ldots, u, \ldots, y}^{\gamma_1}, x)$, returning to S_k after u: by the update rule,

$$|\gamma| = |\gamma_1| + c(y,x) \geq |\beta| + c(y,x) \geq dist[y] + c(y,x) \geq dist[x]$$

where $|\beta|$ is the min weight of an s-to-y S_k-path.



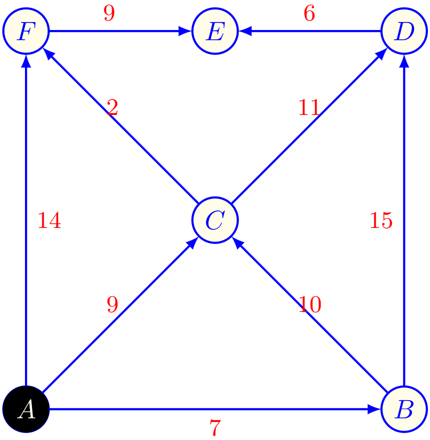
Dijksra's Algorithm: Example 2



Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

for $u \in V(G)$ $dist[u] \leftarrow c[A, u]$

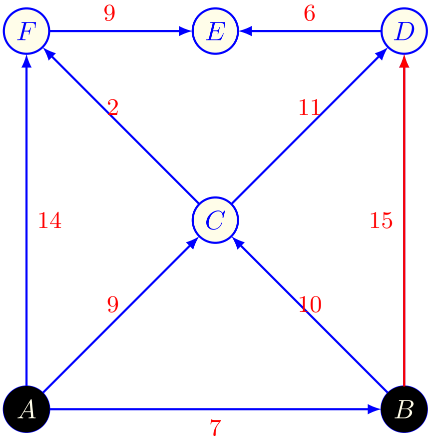
Dijksra's Algorithm: Example 2



Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

$colour[A] \leftarrow \text{BLACK}; \; dist[A] \leftarrow 0$

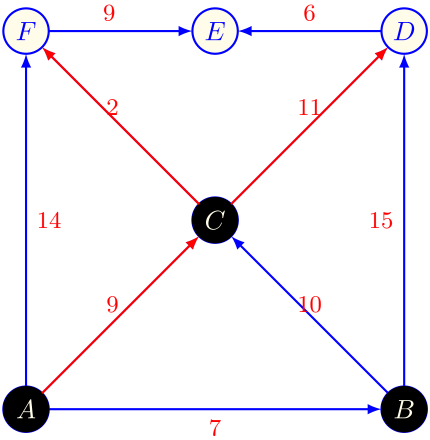
Dijksra's Algorithm: Example 2



Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

while-loop:
WHITE B, C, D, E, F: min $dist[B]$
$colour[B] \leftarrow$ BLACK
for $x \in V(G)$
$dist[x] \leftarrow$
$\min\{dist[x], dist[B] + c[B, x]\}$

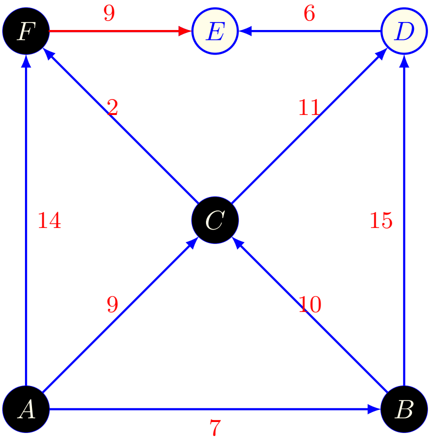
Dijksra's Algorithm: Example 2



Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

while-loop:
WHITE C, D, E, F: min $dist[C]$
$colour[C] \leftarrow \text{BLACK};$
for $x \in V(G)$
$dist[x] \leftarrow$
$\min\left\{dist[x], dist[C] + c[C, x]\right\}$

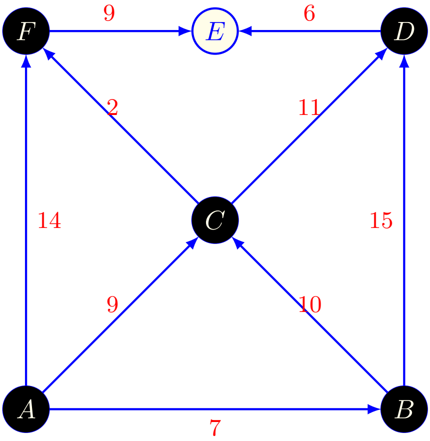
Dijksra's Algorithm: Example 2



Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

while-loop:
WHITE D, E, F: min $dist[F]$
$colour[F] \leftarrow$ BLACK;
for $x \in V(G)$
$dist[x] \leftarrow$
$\min\left\{dist[x], dist[F] + c[F, x]\right\}$

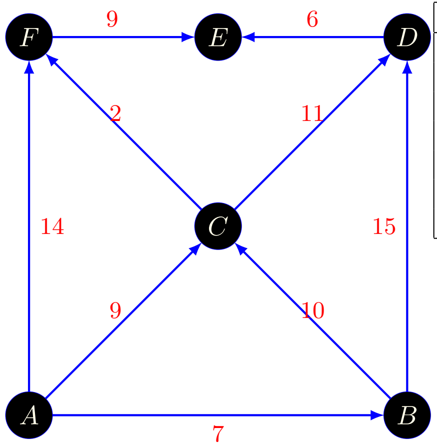
Dijksra's Algorithm: Example 2



Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

while-loop:
WHITE D, E: min $dist[D]$
$colour[D] \leftarrow$ BLACK;
for $x \in V(G)$
$dist[x] \leftarrow$
$\min\left\{dist[x], dist[D] + c[D, x]\right\}$

Dijksra's Algorithm: Example 2

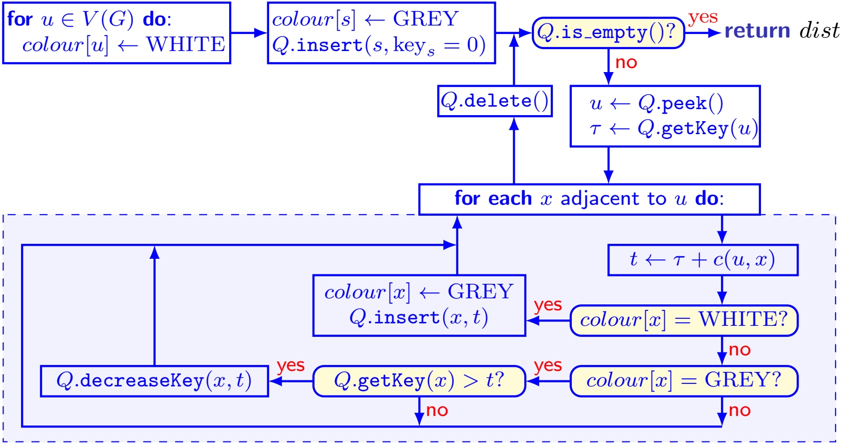


Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

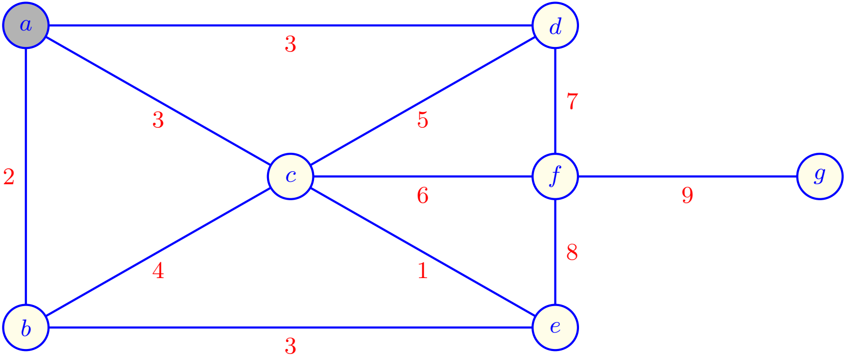
while-loop:
    WHITE E: min $dist[E]$
    $colour[E] \leftarrow \mathrm{BLACK};$
    for $x \in V(G)$
        $dist[x] \leftarrow$
$\min\{dist[x], dist[E] + c[E,x]\}$

Dijksta's Algorithm: PFS Version

Input: weighted digraph (G, c); source node $s \in V(G)$;
priority queue Q; arrays $dist[0..n-1]$; $colour[0..n-1]$



Dijkstra's Algorithm: PFS Version: Start at a

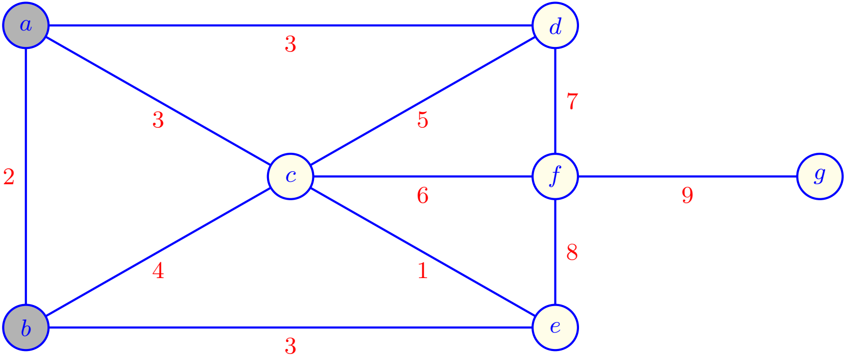


Initialisation:

Priority queue $Q = \{a_{\text{key}=0}\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0						
$dist[v]$	–	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Steps 1 – 2

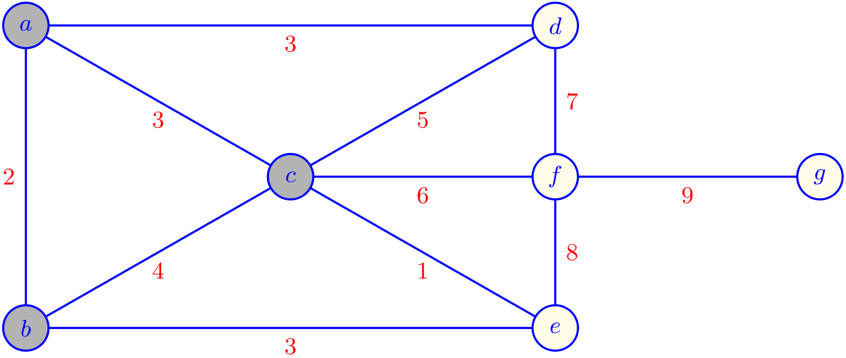


$u \leftarrow a$; $t_1 \leftarrow \text{key}_a = 0$; $x \in \{b, c, d\}$

$x \leftarrow b$: $t_2 = t_1 + \text{cost}(a, b) = 2$; $Q = \{a_0, b_2\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2					
$dist[v]$	–	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 3

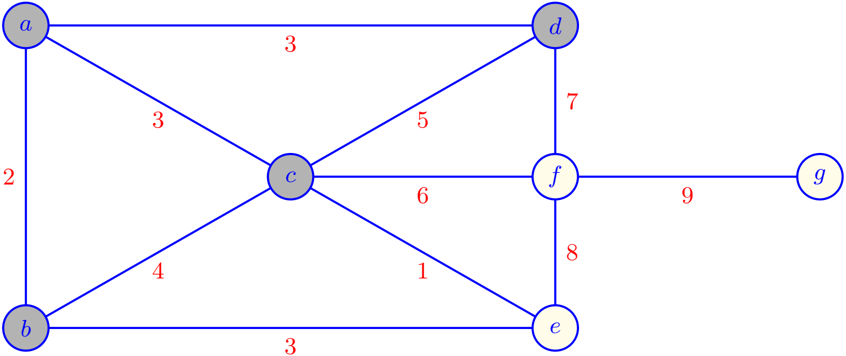


$u = a$; $t_1 = \text{key}_a = 0$; $x \in \{b, c, d\}$

$x \leftarrow c$: $t_2 = t_1 + \text{cost}(a, c) = 3$; $Q = \{a_0, b_2, c_3\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3				
$dist[v]$	–	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 4

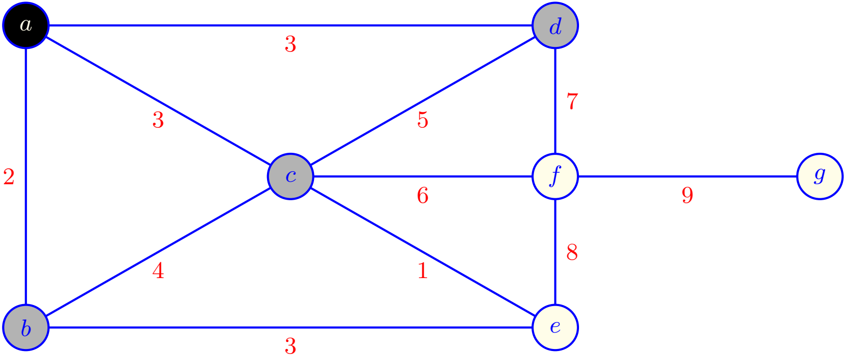


$u = a$; $t_1 = \text{key}_a = 0$; $x \in \{b, c, d\}$

$x \leftarrow d$: $t_2 = t_1 + \text{cost}(a, d) = 3$; $Q = \{a_0, b_2, c_3, d_3\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3			
$dist[v]$	–	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 5

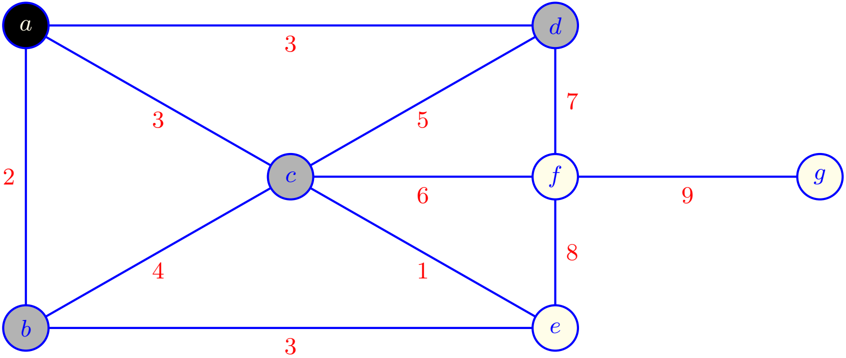


Completing the **while**-loop for $u = a$

$dist[a] \leftarrow t_1 = 0$; $Q = \{b_2, c_3, d_3\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3			
$dist[v]$	0	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Steps 6 – 7

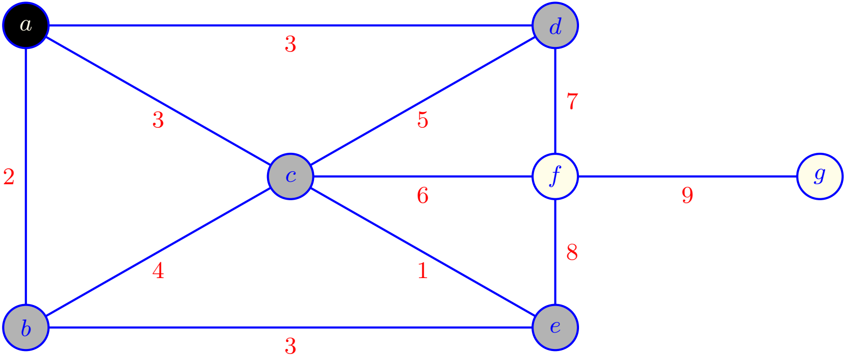


$u \leftarrow b$; $t_1 \leftarrow \text{key}_b = 2$; $x \in \{c, e\}$

$x \leftarrow c$: $t_2 = t_1 + \text{cost}(b, c) = 2 + 4 = 6$; $\text{key}_c = 3 < t_2 = 6$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3			
$dist[v]$	0	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 8

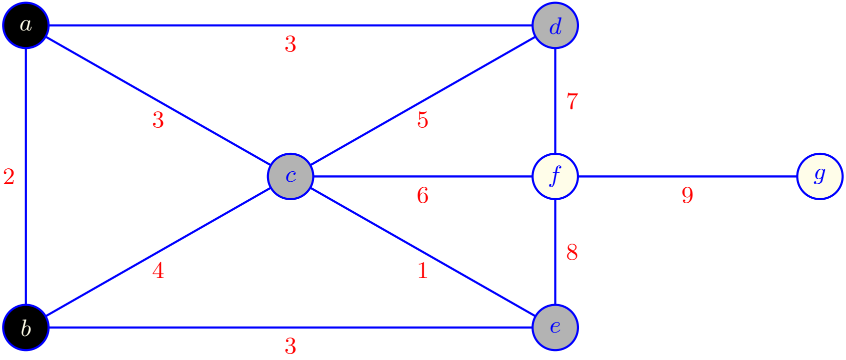


$u = b$; $t_1 = \text{key}_b = 2$; $x \in \{c, e\}$

$x \leftarrow e$: $t_2 = t_1 + \text{cost}(b, e) = 2 + 3 = 5$; $Q = \{b_2, c_3, d_3, e_5\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	5		
$dist[v]$	0	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 9

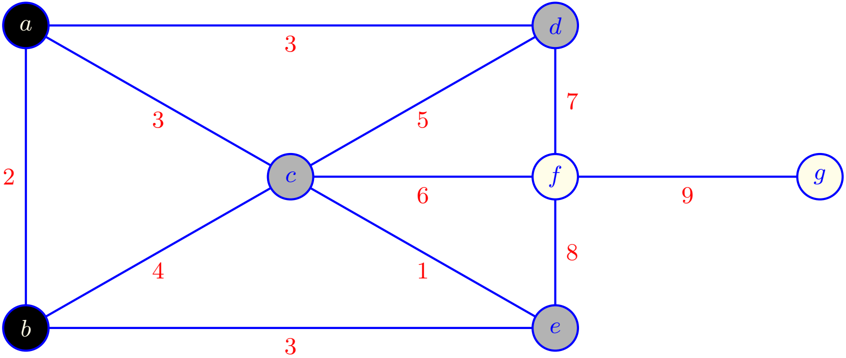


Completing the **while**-loop for $u = b$

$dist[b] \leftarrow t_1 = 2$; $Q = \{c_3, d_3, e_5\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	5		
$dist[v]$	0	2	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Steps 10 – 11

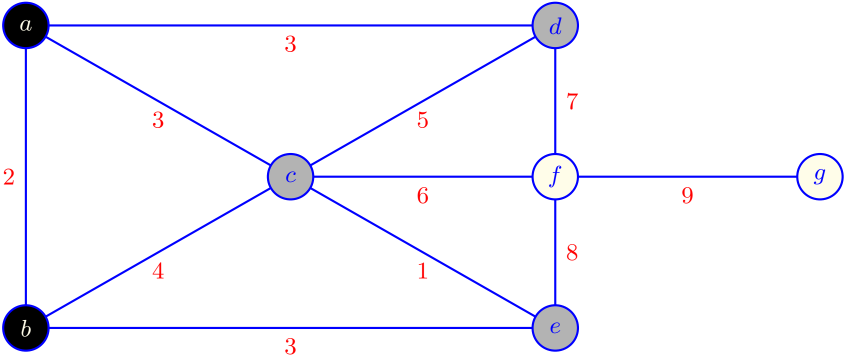


$u \leftarrow c$; $t_1 \leftarrow \mathrm{key}_c = 3$; $x \in \{d, e, f\}$

$x \leftarrow d$: $t_2 = t_1 + \mathrm{cost}(c, d) = 3 + 5 = 8$; $\mathrm{key}_d = 3 < t_2 = 8$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	5		
$dist[v]$	0	2	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 12

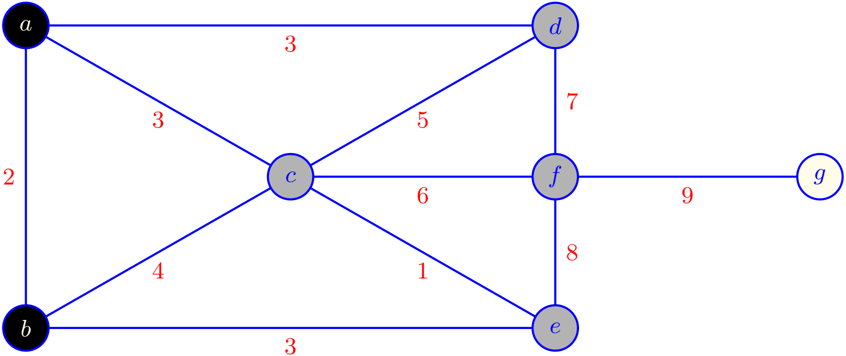


$u = c$; $t_1 = \text{key}_c = 3$; $x \in \{d, e, f\}$

$x \leftarrow e$: $t_2 = t_1 + \text{cost}(c, d) = 3 + 1 = 4$; $\text{key}_e = 5 < t_2 = 4$; $\text{key}_e \leftarrow 4$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4		
$dist[v]$	0	2	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 13

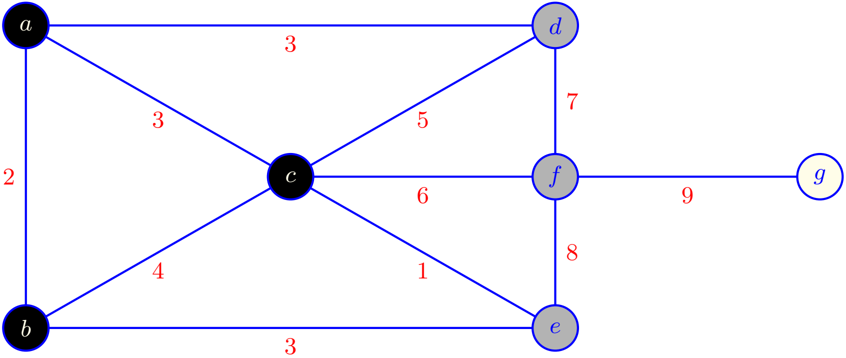


$u = c$; $t_1 = \text{key}_c = 3$; $x \in \{d, e, f\}$

$x \leftarrow f$: $t_2 = t_1 + \text{cost}(c, f) = 3 + 6 = 9$; $Q = \{c_3, d_3, e_4, f_9\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	
$dist[v]$	0	2	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 14

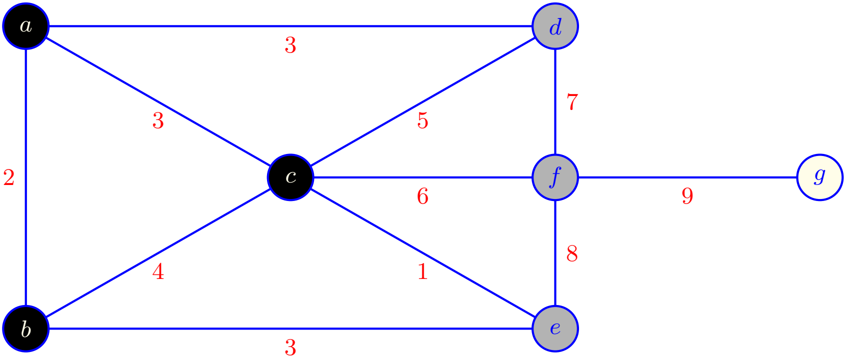


Completing the **while**-loop for $u = c$

$dist[c] \leftarrow t_1 = 3$; $Q = \{d_3, e_4, f_9\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	
$dist[v]$	0	2	3	–	–	–	–

Dijkstra's Algorithm: PFS Version: Steps 15 – 16

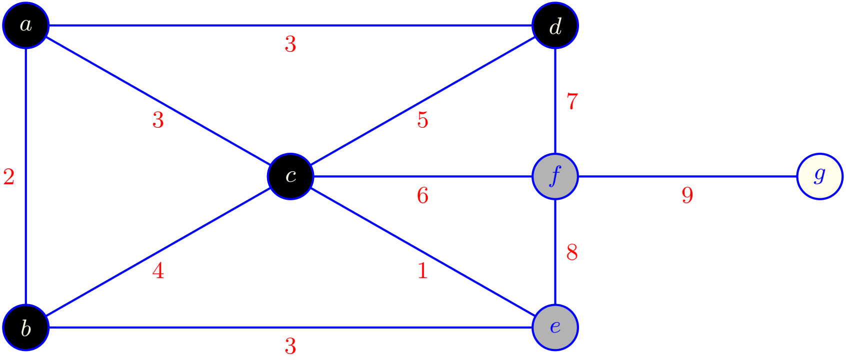


$u \leftarrow d$; $t_1 \leftarrow \text{key}_d = 3$; $x \in \{f\}$

$x \leftarrow f$: $t_2 = t_1 + \text{cost}(d, f) = 3 + 7 = 10$; $\text{key}_f = 9 < t_2 = 10$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	
$dist[v]$	0	2	3	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 17

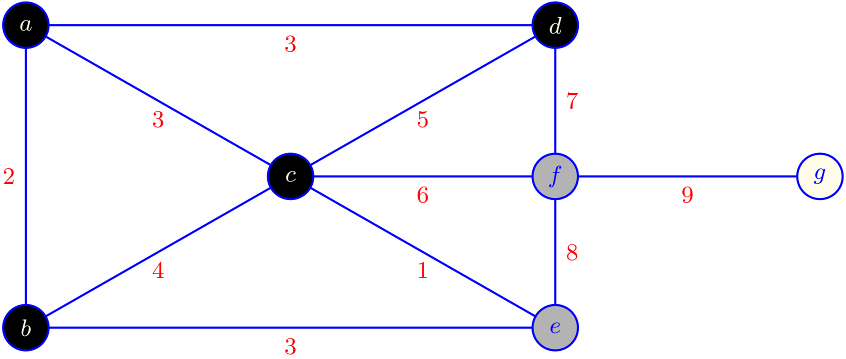


Completing the **while**-loop for $u = d$

$dist[d] \leftarrow t_1 = 3$; $Q = \{e_4, f_9\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	
$dist[v]$	0	2	3	3	–	–	–

Dijkstra's Algorithm: PFS Version: Steps 18 – 19

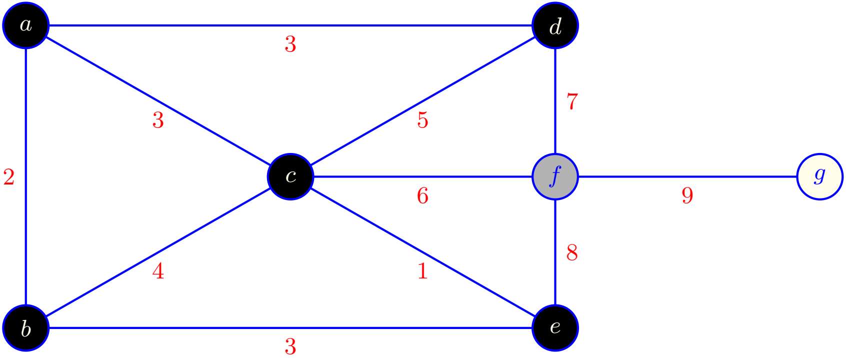


$u \leftarrow e$; $t_1 \leftarrow \text{key}_e = 4$; $x \in \{f\}$

$x \leftarrow f$: $t_2 = t_1 + \text{cost}(e, f) = 4 + 8 = 12$; $\text{key}_f = 9 < t_2 = 12$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	
$dist[v]$	0	2	3	3	–	–	–

Dijkstra's Algorithm: PFS Version: Step 20

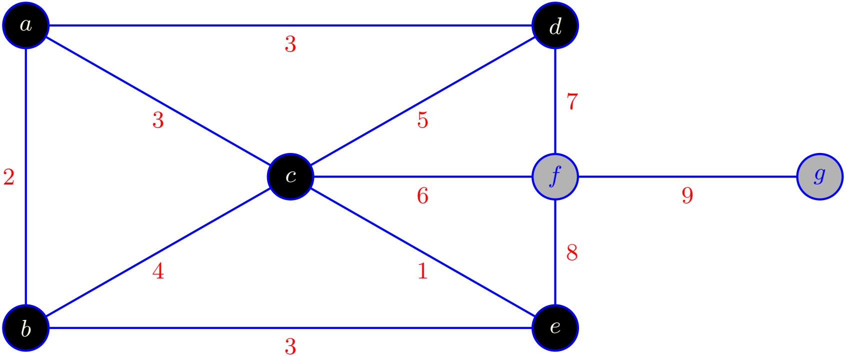


Completing the **while**-loop for $u = e$

$dist[e] \leftarrow t_1 = 4$; $Q = \{f_9\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	
$dist[v]$	0	2	3	3	4	–	–

Dijkstra's Algorithm: PFS Version: Steps 21 – 22

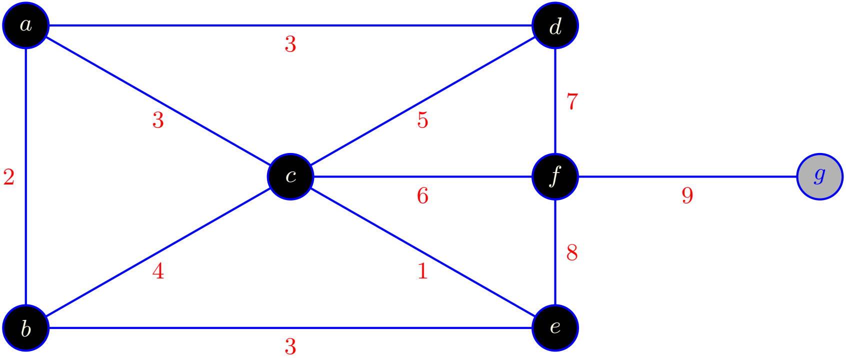


$u \leftarrow f$; $t_1 \leftarrow \text{key}_f = 9$; $x \in \{g\}$

$x \leftarrow g$: $t_2 = t_1 + \text{cost}(f, g) = 9 + 9 = 18$; $Q = \{f_9, g_{18}\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	18
$dist[v]$	0	2	3	3	4	–	–

Dijkstra's Algorithm: PFS Version: Step 23

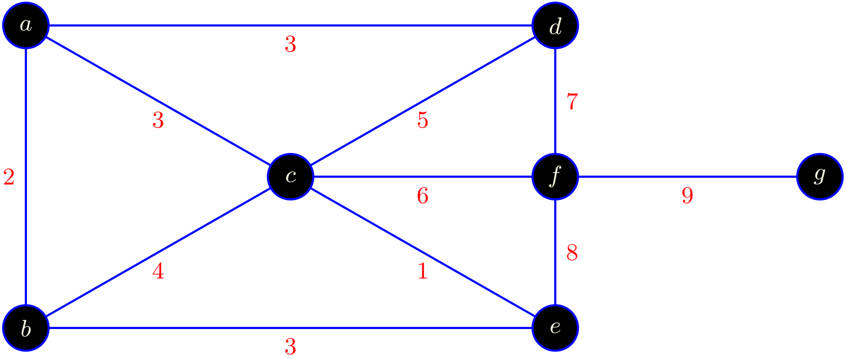


Completing the **while**-loop for $u = f$

$dist[f] \leftarrow t_1 = 9$; $Q = \{g_{18}\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	18
$dist[v]$	0	2	3	3	4	9	–

Dijkstra's Algorithm: PFS Version: Steps 24 – 25



Completing the **while**-loop for $u = g$

$dist[g] \leftarrow t_1 = 18$; no adjacent verices for g; empty $Q = \{\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	18
$dist[v]$	0	2	3	3	4	9	18

SSSP: Bellman-Ford Algorithm

algorithm `Bellman-Ford`(weighted digraph (G, c); node s)
  array $dist[n] = \{\infty, \infty, \ldots\}$
  $dist[s] \leftarrow 0$
  for i **from** 0 **to** $n-1$ **do**
    for $x \in V(G)$ **do**
      for $v \in V(G)$ **do**
        $dist[v] \leftarrow \min(dist[v], dist[x] + c(x, v))$
      end for
    end for
  end for
  return $dist$
end

Time complexity – $\Theta(n^3)$; unlike the Dijkstra's algorithm, it handles negative weight arcs (but no negative weight cycles making the SSSP senseless).

SSSP: Bellman-Ford Algorithm (Alternative Form)

algorithm `Bellman-Ford`(weighted digraph (G, c); node s)
  array $dist[n] = \{\infty, \infty, \ldots\}$
  $dist[s] \leftarrow 0$
  for i **from** 0 **to** $n-1$ **do**
    for $(x, v) \in E(G)$ **do**
      $dist[v] \leftarrow \min(dist[v], dist[x] + c(x, v))$
    end for
  end for
  return $dist$
end

Replacing the two nested **for**-loops by the nodes $x, v \in V(G)$ with a single **for**-loop by the arcs $(x, v) \in E(G)$.

Time complexity: $\Theta(mn)$ using adjacency lists vs. $\Theta(n^3)$ using an adjacency matrix.

Bellman-Ford Algorithm

Slower than Dijkstra's algorithm when all arcs are nonnegative.

Basic idea as in Dijkstra's: to find the single-source shortest paths (SSSP) under progressively relaxing restrictions.

- Dikstra's: one node a time based on their current distance estimate.
- Bellman-Ford: all nodes at "level" $0, 1, \dots, n-1$ in turn.
    - Level of a node v – the minimum possible number of arcs in a minimum weight path to that node from the source s.

Theorem 6.9

If a graph G contains no negative weight cycles, then after the i^{th} iteration of the outer **for**-loop, the element $dist[v]$ contains the minimum weight of a path to v for all nodes v with level at most i.

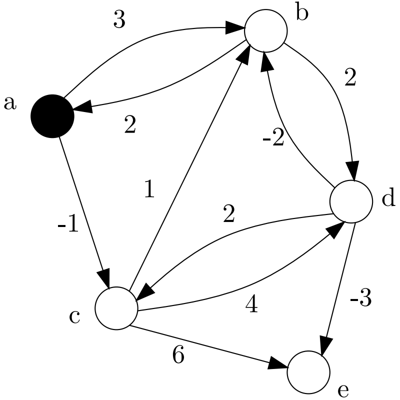
Proving Why Bellman-Ford Algorithm Works

Just as for Dijkstra's, the update ensures $dist[v]$ never increases.

Induction by the level i of the nodes:

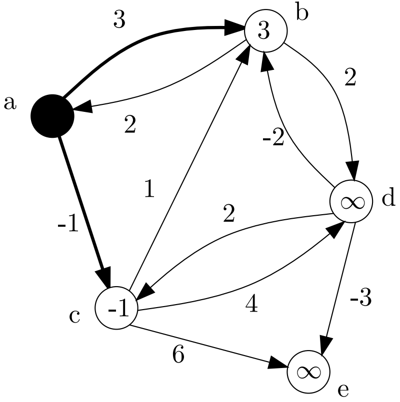
- **Base case**: $i = 0$; the result is true due to initialisation:
$$dist[s] = 0;\ dist[v] = \infty;\ v \in V \backslash s.$$
- **Induction hypothesis**: $dist[v]$; $v \in V$, are true for $i - 1$.
- **Induction step** for a node v at level i:
  - Due to no negative weight cycles, a min-weight s-to-v path, γ, has i arcs.
  - If y is the last node before v and γ_1 the subpath to y, then $dist[y] \leq |\gamma_1|$ by the induction hypothesis.
  - Thus by the update rule:
$$dist[v] \leq dist[y] + c(y, v) \leq |\gamma_1| + c(y, v) \leq |\gamma|$$
as required at level i.

Illustrating Bellman-Ford Algorithm



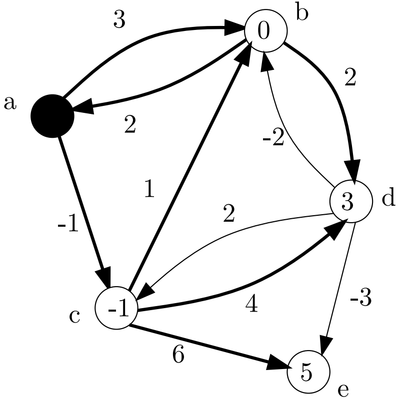
i	$dist[x]$				
	a	b	c	d	e
0	0	∞	∞	∞	∞
1	0	**3**	**-1**	∞	∞
2	0	**0**	-1	**3**	**5**
3	0	0	-1	**2**	**0**
4	0	0	-1	2	**-1**

Illustrating Bellman-Ford Algorithm



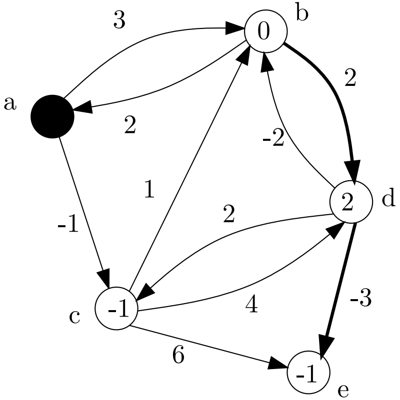
i	$dist[x]$				
	a	b	c	d	e
0	0	∞	∞	∞	∞
1	0	**3**	**-1**	∞	∞
2	0	**0**	-1	**3**	**5**
3	0	0	-1	**2**	**0**
4	0	0	-1	2	**-1**

Illustrating Bellman-Ford Algorithm



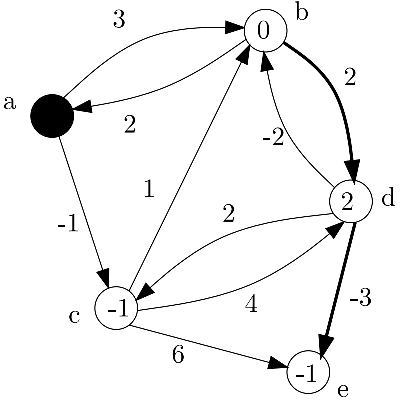
i	$dist[x]$				
	a	b	c	d	e
0	0	∞	∞	∞	∞
1	0	**3**	**-1**	∞	∞
2	0	**0**	-1	**3**	**5**
3	0	0	-1	2	0
4	0	0	-1	2	-1

Illustrating Bellman-Ford Algorithm



i	$dist[x]$				
	a	b	c	d	e
0	0	∞	∞	∞	∞
1	0	$\mathbf{3}$	$\mathbf{-1}$	∞	∞
2	0	$\mathbf{0}$	-1	$\mathbf{3}$	$\mathbf{5}$
3	0	0	-1	$\mathbf{2}$	$\mathbf{0}$
4	0	0	-1	2	-1

Illustrating Bellman-Ford Algorithm



i	$dist[x]$				
	a	b	c	d	e
0	0	∞	∞	∞	∞
1	0	**3**	**-1**	∞	∞
2	0	**0**	-1	**3**	**5**
3	0	0	-1	**2**	**0**
4	0	0	-1	2	**-1**

Illustrating Bellman-Ford Algorithm (Alternative Form)

Arc (x,v):	a,b	a,c	b,a	b,d	c,b	c,d	c,e	d,b	d,c	d,e
$c(x,v)$:	3	-1	2	2	1	4	6	-2	2	-3

Iteration $i = 0$

x, v	Distance $d[v] \leftarrow \min\{d[v], d[x] + c(x,v)\}$						a	b	c	d	e
							0	∞	∞	∞	∞
a, b	$d[\mathrm{b}]$	$\leftarrow$	$\min\{\infty,$	$0+3\}$	$=$	3	0	3	∞	∞	∞
a, c	$d[\mathrm{c}]$	$\leftarrow$	$\min\{\infty,$	$0-1\}$	$=$	-1	0	3	-1	∞	∞
b, a	$d[\mathrm{a}]$	$\leftarrow$	$\min\{0,$	$3+2\}$	$=$	0	0	3	-1	∞	∞
b, d	$d[\mathrm{d}]$	$\leftarrow$	$\min\{\infty,$	$3+2\}$	$=$	5	0	3	-1	5	∞
c, b	$d[\mathrm{b}]$	$\leftarrow$	$\min\{3,$	$-1+1\}$	$=$	0	0	0	-1	5	∞
c, d	$d[\mathrm{d}]$	$\leftarrow$	$\min\{5,$	$-1+4\}$	$=$	3	0	0	-1	3	∞
c, e	$d[\mathrm{e}]$	$\leftarrow$	$\min\{\infty,$	$-1+6\}$	$=$	5	0	0	-1	3	5
d, b	$d[\mathrm{b}]$	$\leftarrow$	$\min\{0,$	$3-2\}$	$=$	0	0	0	-1	3	5
d, c	$d[\mathrm{c}]$	$\leftarrow$	$\min\{-1,$	$3+2\}$	$=$	-1	0	0	-1	3	5
d, e	$d[\mathrm{e}]$	$\leftarrow$	$\min\{5,$	$3-3\}$	$=$	0	0	0	-1	3	0

Illustrating Bellman-Ford Algorithm (Alternative Form)

Arc (x,v):	a,b	a,c	b,a	b,d	c,b	c,d	c,e	d,b	d,c	d,e
$c(x,v)$:	3	-1	2	2	1	4	6	-2	2	-3

Iteration $i = 1$

x, v	Distance $d[v] \leftarrow \min\{d[v], d[x] + c(x,v)\}$						a	b	c	d	e
							0	0	-1	3	0
a, b	$d[\mathrm{b}]$	$\leftarrow$	$\min\{0,$	$0+3\}$	$=$	0	0	0	-1	3	0
a, c	$d[\mathrm{c}]$	$\leftarrow$	$\min\{-1,$	$0-1\}$	$=$	-1	0	0	-1	3	0
b, a	$d[\mathrm{a}]$	$\leftarrow$	$\min\{0,$	$0+2\}$	$=$	0	0	0	-1	3	0
b, d	$d[\mathrm{d}]$	$\leftarrow$	$\min\{3,$	$0+2\}$	$=$	2	0	0	-1	2	0
c, b	$d[\mathrm{b}]$	$\leftarrow$	$\min\{0,$	$-1+1\}$	$=$	0	0	0	-1	2	0
c, d	$d[\mathrm{d}]$	$\leftarrow$	$\min\{2,$	$-1+4\}$	$=$	2	0	0	-1	2	0
c, e	$d[\mathrm{e}]$	$\leftarrow$	$\min\{0,$	$-1+6\}$	$=$	0	0	0	-1	2	0
d, b	$d[\mathrm{b}]$	$\leftarrow$	$\min\{0,$	$2-2\}$	$=$	0	0	0	-1	2	0
d, c	$d[\mathrm{c}]$	$\leftarrow$	$\min\{-1,$	$2+2\}$	$=$	-1	0	0	-1	2	0
d, e	$d[\mathrm{e}]$	$\leftarrow$	$\min\{0,$	$2-3\}$	$=$	-1	0	0	-1	2	-1

Illustrating Bellman-Ford Algorithm (Alternative Form)

Arc (x,v):	a,b	a,c	b,a	b,d	c,b	c,d	c,e	d,b	d,c	d,e
$c(x,v)$:	3	-1	2	2	1	4	6	-2	2	-3

Iteration $i = 2..4$

x, v	Distance $d[v] \leftarrow \min\{d[v], d[x] + c(x,v)\}$						a	b	c	d	e
							0	0	-1	2	-1
a, b	$d[\mathrm{b}]$	$\leftarrow$	$\min\{0,$	$0+3\}$	$=$	0	0	0	-1	2	-1
a, c	$d[\mathrm{c}]$	$\leftarrow$	$\min\{-1,$	$0-1\}$	$=$	-1	0	0	-1	2	-1
b, a	$d[\mathrm{a}]$	$\leftarrow$	$\min\{0,$	$0+2\}$	$=$	0	0	0	-1	2	-1
b, d	$d[\mathrm{d}]$	$\leftarrow$	$\min\{2,$	$0+2\}$	$=$	2	0	0	-1	2	-1
c, b	$d[\mathrm{b}]$	$\leftarrow$	$\min\{0,$	$-1+1\}$	$=$	0	0	0	-1	2	-1
c, d	$d[\mathrm{d}]$	$\leftarrow$	$\min\{2,$	$-1+4\}$	$=$	2	0	0	-1	2	-1
c, e	$d[\mathrm{e}]$	$\leftarrow$	$\min\{-1,$	$-1+6\}$	$=$	-1	0	0	-1	2	-1
d, b	$d[\mathrm{b}]$	$\leftarrow$	$\min\{0,$	$3-2\}$	$=$	0	0	0	-1	2	-1
d, c	$d[\mathrm{c}]$	$\leftarrow$	$\min\{-1,$	$3+2\}$	$=$	-1	0	0	-1	2	-1
d, e	$d[\mathrm{e}]$	$\leftarrow$	$\min\{-1,$	$3-3\}$	$=$	-1	0	0	-1	2	-1

Comments on Bellman-Ford Algorithm

- This (non-greedy) algorithm handles negative weight arcs, but not negative weight cycles.
- Running time with the two innermost nested **for**-loops: $\mathrm{O}(n^3)$.
  - Runs slower than the Dijkstra's algorithm since considers all nodes at "level" $i = 0, 1, \ldots, n-1$, in turn.
- The alternative form where the two inner-most **for**-loops are replaced with: **for** $(u, v) \in E(V)$ runs in time $O(nm)$.
  - The outer **for**-loop (by i) in this case can be terminated after no distance changes during the iteration (e.g., after $i = 2$ in the example on Slide 39).
- Bellman-Ford algorithm can be modified to detect negative weight cycle (see Textbook, Exercise 6.3.4)

All Pairs Shortest Path (APSP) Problem

Given a weighted digraph (G, c), determine for each pair of nodes $u, v \in V(G)$ (the length of) a minimum weight path from u to v.

Convenient output: a distance matrix $D = \left[D[u, v]\right]_{u,v \in V(G)}$

- Time complexity $\Theta(nA_{n,m})$ of computing the matrix D by finding the single-source shortest paths (SSSP) from each node as the source in turn.
  - $A_{n=|V(G)|,m=|E(G)|}$ – the complexity of the SSSP algorithm.
  - The APSP complexity $\Theta(n^3)$ for the adjacency matrix version of the Dijkstra's SSSP algorithm: $A_{n,m} = n^2$.
  - The APSP complexity $\Theta(n^2 m)$ for the Bellman-Ford SSSP algorithm: $A_{n,m} = mn$.

All Pairs Shortest Path (APSP) Problem

Floyd's algorithm – one of the known simpler algorithms for computing the distance matrix (three nested **for**-loops; $\Theta(n^3)$ time complexity):

1. Number all nodes (say, from 0 to $n-1$).
2. At each step k, maintain the matrix of shortest distances from node i to node j, not passing through nodes higher than k.
3. Update the matrix at each step to see whether the node k shortens the current best distance.

An alternative to running the SSSP algorithm from each node.

- Better than the Dijkstra's algorithm for dense graphs, probably not for sparse ones.
- Unlike the Dijkstra's algorithm, can handle negative costs.
- Based on Warshall's algorithm (just tells whether there is a path from node i to node j, not concerned with length).

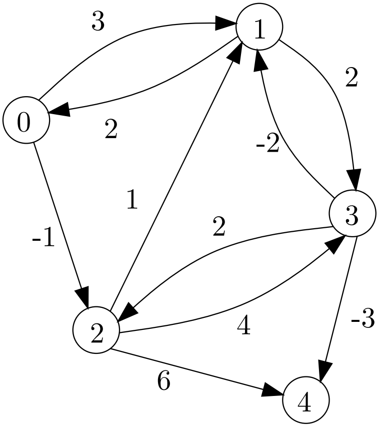
Floyd's Algorithm

algorithm `Floyd`(weighted digraph (G, c))
Initialisation: **for** $u, v \in V(G)$ **do** $D[u, v] \leftarrow c(u, v)$ **end for**
for $x \in V(G)$ **do**
    for $u \in V(G)$ **do**
        for $v \in V(G)$ **do**
            $D[u, v] \leftarrow \min\{D[u, v], D[u, x] + D[x, v]\}$
        end for
    end for
end for

This algorithm is based on **dynamic programming** principles.

At the bottom of the outer **for**-x-loop, $D[u, v]$ for each $u, v \in V(G)$ is the length of the shortest path from u to v passing through intermediate nodes x having been seen in that loop.

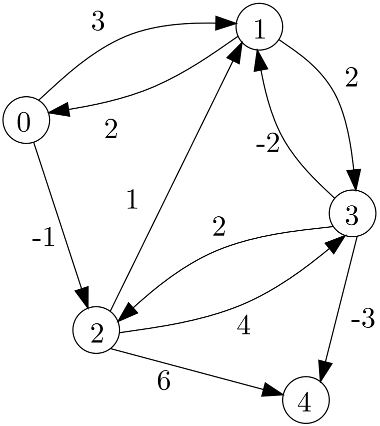
Illustrating Floyd's Algorithm



$$
\begin{array}{c c}
 & \begin{array}{ccccc} \color{red}{0} & \color{red}{1} & \color{red}{2} & \color{red}{3} & \color{red}{4} \end{array} \\
\begin{array}{c} \color{red}{0} \\ \color{red}{1} \\ \color{red}{2} \\ \color{red}{3} \\ \color{red}{4} \end{array} &
\begin{bmatrix}
0 & 3 & -1 & \infty & \infty \\
2 & 0 & \infty & 2 & \infty \\
\infty & 1 & 0 & 4 & 6 \\
\infty & -2 & 2 & 0 & -3 \\
\infty & \infty & \infty & \infty & 0
\end{bmatrix}
\end{array}
$$

Adjacency/cost matrix $c[u, v]$

Illustrating Floyd's Algorithm: $x = 0$

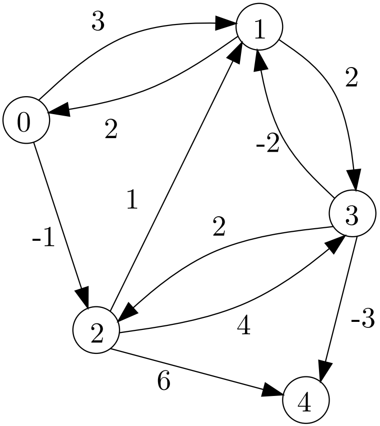


$$
\begin{array}{c|c}
 & \begin{matrix} 0 & 1 & 2 & 3 & 4 \end{matrix} \\
\begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} &
\begin{bmatrix}
0 & 3 & -1 & \infty & \infty \\
2 & 0 & \mathbf{1} & 2 & \infty \\
\infty & 1 & 0 & 4 & 6 \\
\infty & -2 & 2 & 0 & -3 \\
\infty & \infty & \infty & \infty & 0
\end{bmatrix}
\end{array}
$$

Distance matrix $D_0[u, v]$

$$D_0[1,2] = \min\{\infty, 2_{c[1,0]} - 1_{c[0,1]}\} = 1$$

Illustrating Floyd's Algorithm: $x = 1$

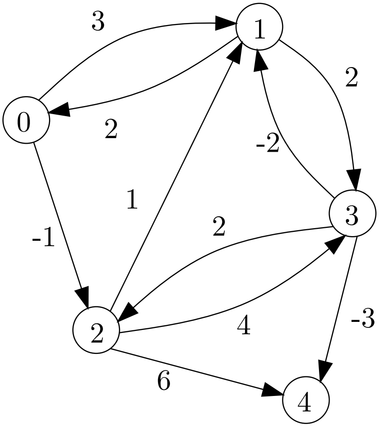


$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \left[\begin{array}{c|c|ccc} 0 & 3 & -1 & \mathbf{5} & \infty \\ \hline 2 & 0 & 1 & 2 & \infty \\ \hline \mathbf{3} & 1 & 0 & \mathbf{3} & 6 \\ \mathbf{0} & -2 & \mathbf{-1} & 0 & -3 \\ \infty & \infty & \infty & \infty & 0 \end{array}\right] \end{array}$$

Distance matrix $D_1[u, v]$

$D_1[0,3] = \min\{\infty, 3_{D_0[0,1]} + 2_{D_0[1,3]}\} = 5$
$D_1[2,3] = \min\{4, 1_{D_0[2,1]} + 2_{D_0[1,3]}\} = 3$
$D_1[3,2] = \min\{2, -2_{D_0[3,1]} + 1_{D_0[1,2]}\} = -1$

Illustrating Floyd's Algorithm: $x = 2$

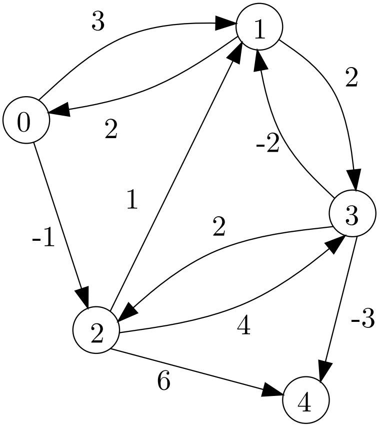


$$
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0 & \mathbf{0} & -1 & \mathbf{2} & \mathbf{5} \\
1 & 2 & 0 & 1 & 2 & \mathbf{7} \\
2 & 3 & 1 & 0 & 3 & 6 \\
3 & 0 & -2 & -1 & 0 & -3 \\
4 & \infty & \infty & \infty & \infty & 0
\end{array}
$$

Distance matrix $D_2[u, v]$

$D_2[0,1] = \min\{3, -1_{D_1[0,2]} + 1_{D_1[2,1]}\} = 0$
$D_2[0,3] = \min\{5, -1_{D_1[0,2]} + 3_{D_1[2,3]}\} = 2$
$D_2[0,4] = \min\{\infty, -1_{D_1[0,2]} + 6_{D_1[2,4]}\} = 5$
$D_2[1,4] = \min\{\infty, 1_{D_1[1,2]} + 6_{D_1[2,4]}\} = 7$

Illustrating Floyd's Algorithm: $x = 3$



$$
\begin{array}{c c}
 & \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} &
\left[\begin{array}{ccc|c|c}
0 & 0 & -1 & 2 & \mathbf{-1} \\
2 & 0 & 1 & 2 & \mathbf{-1} \\
3 & 1 & 0 & 3 & \mathbf{0} \\
\hline
0 & -2 & -1 & 0 & -3 \\
\hline
\infty & \infty & \infty & \infty & 0
\end{array}\right]
\end{array}
$$

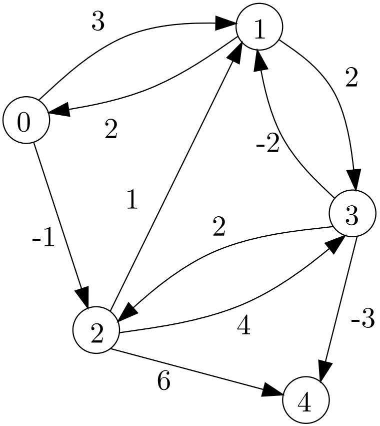
Distance matrix $D_3[u, v]$

$D_3[0,4] = \min\{5, 2_{D_2[0,3]} - 3_{D_2[3,4]}\} = -1$

$D_3[1,4] = \min\{7, 2_{D_1[1,3]} - 3_{D_1[3,4]}\} = -1$

$D_3[2,4] = \min\{6, 3_{D_1[2,3]} - 3_{D_1[3,4]}\} = 0$

Illustrating Floyd's Algorithm: $x = 4$



$$
\begin{array}{c c}
 & \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} &
\begin{bmatrix}
0 & 0 & -1 & 2 & -1 \\
2 & 0 & 1 & 2 & -1 \\
3 & 1 & 0 & 3 & 0 \\
0 & -2 & -1 & 0 & -3 \\
\infty & \infty & \infty & \infty & 0
\end{bmatrix}
\end{array}
$$

Final distance matrix $D \equiv D_4[u, v]$

Proving Why Floyd's Algorithm Works

Theorem 6.12: At the bottom of the outer **for**-loop, for all nodes u and v, $D[u,v]$ contains the minimum length of all paths from u to v that are restricted to using only intermediate nodes that have been seen in the outer **for**-loop.

When algorithm terminates, all nodes have been seen and $D[u,v]$ is the length of the shortest u-to-v path.

Notation: S_k – the set of nodes seen after k passes through this loop; S_k-path – one with all intermediate nodes in S_k; D_k – the corresponding value of D.

Induction on the outer **for**-loop:

- **Base case**: $k = 0$; $S_0 = \emptyset$, and the result holds.
- **Induction hypothesis**: It holds after $k \geq 0$ times through the loop.
- **Inductive step**: To show that $D_{k+1}[u,v]$ after $k+1$ passes through this loop is the minimum length of an u-to-v S_{k+1}-path.

Proving Why Floyd's Algorithm Works

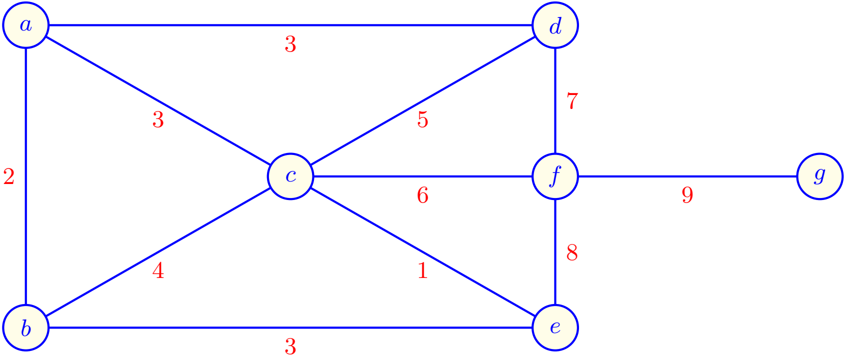
Inductive step:
Suppose that x is the last node seen in the loop, so $S_{k+1} = S_k \bigcup \{x\}$.

- Fix an arbitrary pair of nodes $u, v \in V(G)$ and let L be the min-length of an u-to-v S_{k+1}-path, so that obviously $L \leq D_{k+1}[u, v]$.
- To show that also $D_{k+1}[u, v] \leq L$, choose an u-to-v S_{k+1}-path γ of length L. If $x \notin \gamma$, the result follows from the induction hypothesis.
- If $x \in \gamma$, let γ_1 and γ_2 be, respectively, the u-to-x and x-to-v subpaths. Then γ_1 and γ_2 are S_k-paths and by the inductive hypothesis,

$$L \geq |\gamma_1| + |\gamma_2| \geq D_k[u, x] + D_k[x, v] \geq D_{k+1}[u, v]$$

Non-negativity of the weights is not used in the proof, and Floyd's algorithm works for negative weights (but negative weight cycles should not be present).

Floyd's Algorithm: Example 2



Computing all-pairs shortest paths

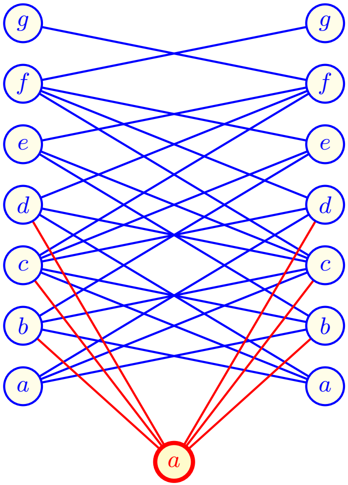
Floyd's Algorithm: Example 2

Initialisation

$$\left[D[u,v]\right]_{u,v\in V(G)} \leftarrow \overbrace{\begin{array}{c|ccccccc|}
a & 0 & 2 & 3 & 3 & \infty & \infty & \infty \\
b & 2 & 0 & 4 & \infty & 3 & \infty & \infty \\
c & 3 & 4 & 0 & 5 & 1 & 6 & \infty \\
d & 3 & \infty & 5 & 0 & \infty & 7 & \infty \\
e & \infty & 3 & 1 & \infty & 0 & 8 & \infty \\
f & \infty & \infty & 6 & 7 & 8 & 0 & 9 \\
g & \infty & \infty & \infty & \infty & \infty & 9 & 0 \\
\hline
 & a & b & c & d & e & f & g
\end{array}}^{\text{Initialisation: } c(u,v)]}$$

for $x \in V = \{a, b, c, d, e, f, g\}$ **do**
  for $u \in V = \{a, b, c, d, e, f, g\}$ **do**
    for $v \in V = \{a, b, c, d, e, f, g\}$ **do**
      $D[u,v] \leftarrow \min\{D[u,v],\ D[u,x] + D[x,v]\}$
    end for
  end for
end for

Floyd's Algorithm: Example 2 $x \leftarrow a$



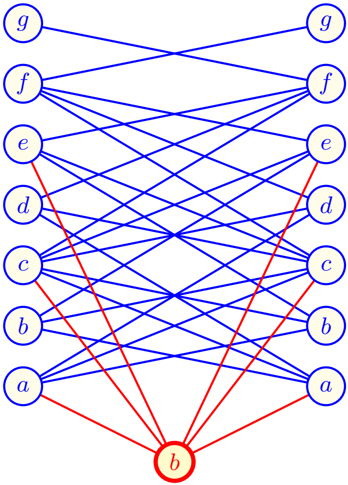
$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \underbrace{\left[\begin{array}{ccccccc} 0 & 2 & 3 & 3 & \infty & \infty & \infty \\ 2 & 0 & 4 & 5 & 3 & \infty & \infty \\ 3 & 4 & 0 & 5 & 1 & 6 & \infty \\ 3 & 5 & 5 & 0 & \infty & 7 & \infty \\ \infty & 3 & 1 & \infty & 0 & 8 & \infty \\ \infty & \infty & 6 & 7 & 8 & 0 & 9 \\ \infty & \infty & \infty & \infty & \infty & 9 & 0 \end{array}\right]} \end{array}$$

$$D[u,v] \leftarrow \min\{D[u,v],\ D[u,a] + D[a,v]\};\ (u,v) \in V^2$$

E.g.,

$$\begin{aligned} D[b,d] &\leftarrow \min\{D[b,d], D[b,a] + D[a,d]\} \\ &= \min\{\infty, 2+3\} = 5 \end{aligned}$$

Floyd's Algorithm: Example 2 $x \leftarrow b$



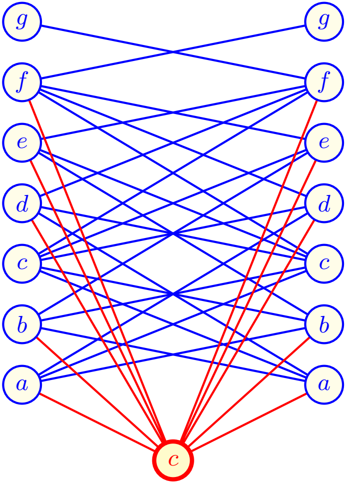
$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array}\begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \underbrace{\begin{bmatrix} 0 & 2 & 3 & 3 & 5 & \infty & \infty \\ 2 & 0 & 4 & 5 & 3 & \infty & \infty \\ 3 & 4 & 0 & 5 & 1 & 6 & \infty \\ 3 & 5 & 5 & 0 & 8 & 7 & \infty \\ 5 & 3 & 1 & 8 & 0 & 8 & \infty \\ \infty & \infty & 6 & 7 & 8 & 0 & 9 \\ \infty & \infty & \infty & \infty & \infty & 9 & 0 \end{bmatrix}}_{} \end{array}$$

$$D[u,v] \leftarrow \min\{D[u,v],\ D[u,b] + D[b,v]\}\,;\ (u,v) \in V^2$$

E.g.,

$$\begin{aligned} D[a,e] &\leftarrow \min\{D[a,e], D[a,b] + D[b,e]\} \\ &= \min\{\infty, 2+3\} = 5 \end{aligned}$$

Floyd's Algorithm: Example 2 $x \leftarrow c$

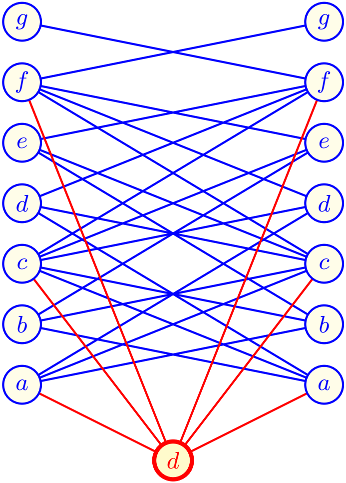


$$
\underbrace{\begin{array}{c|ccccccc|}
 & a & b & c & d & e & f & g \\
a & 0 & 2 & 3 & 3 & 4 & 9 & \infty \\
b & 2 & 0 & 4 & 5 & 3 & 10 & \infty \\
c & 3 & 4 & 0 & 5 & 1 & 6 & \infty \\
d & 3 & 5 & 5 & 0 & 6 & 7 & \infty \\
e & 4 & 3 & 1 & 6 & 0 & 7 & \infty \\
f & 9 & 10 & 6 & 7 & 7 & 0 & 9 \\
g & \infty & \infty & \infty & \infty & \infty & 9 & 0
\end{array}}_{D[u,v] \leftarrow \min\{D[u,v],\ D[u,c]+D[c,v]\};\ (u,v)\in V^2}
$$

E.g.,

$$
\begin{aligned}
D[a,f] &\leftarrow \min\{D[a,f], D[a,c]+D[c,f]\} \\
&= \min\{\infty, 3+6\} = 9
\end{aligned}
$$

Floyd's Algorithm: Example 2 $x \leftarrow d$



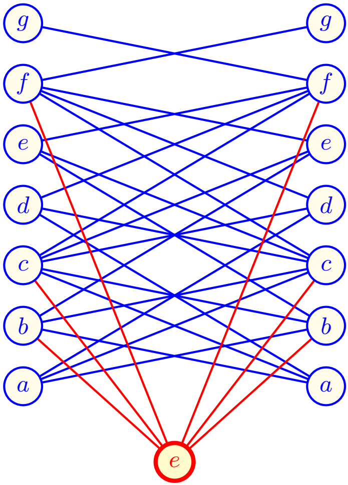
$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \left[\begin{array}{ccccccc} 0 & 2 & 3 & 3 & 4 & 9 & \infty \\ 2 & 0 & 4 & 5 & 3 & 10 & \infty \\ 3 & 4 & 0 & 5 & 1 & 6 & \infty \\ 3 & 5 & 5 & 0 & 8 & 7 & \infty \\ 4 & 3 & 1 & 8 & 0 & 7 & \infty \\ 9 & 10 & 6 & 7 & 7 & 0 & 9 \\ \infty & \infty & \infty & \infty & \infty & 9 & 0 \end{array}\right] \end{array}$$

$$D[u,v] \leftarrow \min\{D[u,v],\ D[u,d] + D[d,v]\};\ (u,v) \in V^2$$

E.g.,

$$D[a,f] \leftarrow \min\{D[a,f], D[a,d] + D[d,f]\} = \min\{9, 3+7\} = 9$$

Floyd's Algorithm: Example 2 $x \leftarrow e$



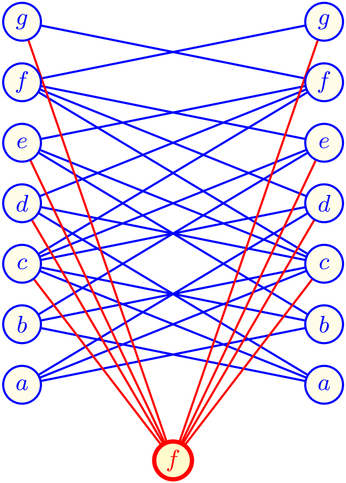
$$
\begin{array}{c|ccccccc}
 & a & b & c & d & e & f & g \\
\hline
a & 0 & 2 & 3 & 3 & 4 & 9 & \infty \\
b & 2 & 0 & 4 & 5 & 3 & 10 & \infty \\
c & 3 & 4 & 0 & 5 & 1 & 6 & \infty \\
d & 3 & 5 & 5 & 0 & 8 & 7 & \infty \\
e & 4 & 3 & 1 & 8 & 0 & 7 & \infty \\
f & 9 & 10 & 6 & 7 & 7 & 0 & 9 \\
g & \infty & \infty & \infty & \infty & \infty & 9 & 0
\end{array}
$$

$$D[u,v] \leftarrow \min\{D[u,v],\ D[u,e] + D[e,v]\};\ (u,v) \in V^2$$

E.g.,

$$D[b,f] \leftarrow \min\{D[b,f], D[b,e] + D[e,f]\} = \min\{9, 3+7\} = 9$$

Floyd's Algorithm: Example 2 $x \leftarrow f$



$$
\begin{array}{c|ccccccc}
 & a & b & c & d & e & f & g \\
\hline
a & 0 & 2 & 3 & 3 & 4 & 9 & 18 \\
b & 2 & 0 & 4 & 5 & 3 & 10 & 19 \\
c & 3 & 4 & 0 & 5 & 1 & 6 & 15 \\
d & 3 & 5 & 5 & 0 & 8 & 7 & 16 \\
e & 4 & 3 & 1 & 8 & 0 & 7 & 16 \\
f & 9 & 10 & 6 & 7 & 7 & 0 & 9 \\
g & 18 & 19 & 15 & 16 & 16 & 9 & 0
\end{array}
$$

$$D[u,v] \leftarrow \min\{D[u,v],\ D[u,f] + D[f,v]\}\,; \quad (u,v) \in V^2$$

E.g.,

$$D[a,g] \leftarrow \min\{D[a,g], D[a,f] + D[f,g]\} = \min\{\infty, 9+9\} = 18$$

Computing Actual Shortest Paths

- In addition to knowing the shortest distances, the shortest paths are often to be reconstructed.
- The Floyd's algorithm can be enhanced to compute also the **predecessor matrix** $\Pi = [\pi_{ij}]_{i,j=1,1}^{n,n}$ where vertex $\pi_{i,j}$ precedes vertex j on a shortest path from vertex i; $1 \leq i, j \leq n$.

Compute a sequence $\Pi^{(0)}, \Pi^{(1)}, \ldots \Pi^{(n)}$,

where vertex $\pi_{i,j}^{(k)}$ precedes the vertex j on a shortest path from vertex i with all intermediate vertices in $V_{(k)} = \{1, 2, \ldots, k\}$.

For case of no intermediate vertices:

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

Computing Actual Shortest Paths

- In addition to knowing the shortest distances, the shortest paths are often to be reconstructed.
- The Floyd's algorithm can be enhanced to compute also the **predecessor matrix** $\Pi = [\pi_{ij}]_{i,j=1,1}^{n,n}$ where vertex $\pi_{i,j}$ precedes vertex j on a shortest path from vertex i; $1 \leq i, j \leq n$.

Compute a sequence $\Pi^{(0)}, \Pi^{(1)}, \ldots \Pi^{(n)}$,

where vertex $\pi_{i,j}^{(k)}$ precedes the vertex j on a shortest path from vertex i with all intermediate vertices in $V_{(k)} = \{1, 2, \ldots, k\}$.

For case of no intermediate vertices:

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

Floyd's Algorithm with Predecessors

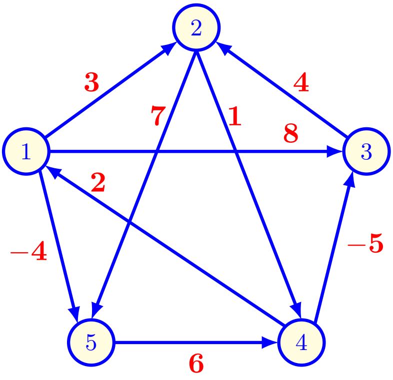
algorithm `FloydPred(` weighted digraph (G, c) `)`

$D \leftarrow c$ $\boxed{\text{Create initial distance matrix from weights.}}$

$\Pi \leftarrow \Pi^{(0)}$ $\boxed{\text{Initialize predecessors from } c \text{ as in Slide 60.}}$

for k **from** 1 **to** n **do**
  for i **from** 1 **to** n **do**
    for j **from** 1 **to** n **do**
      if $D[i,j] > D[i,k] + D[k,j]$ **then**
        $D[i,j] \leftarrow D[i,k] + D[k,j]; \quad \Pi[i,j] \leftarrow \Pi[k,j]$
      end if
    end for
  end for
end for

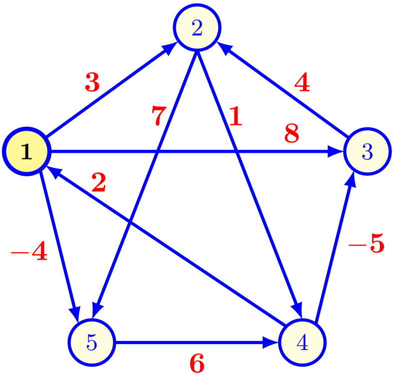
Illustrating Floyd's Algorithm with Predecessors



$$D^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{array}$$

$$\Pi^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \mathrm{NIL} & 1 & 1 & \mathrm{NIL} & 1 \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 2 & 2 \\ \mathrm{NIL} & 3 & \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} \\ 4 & \mathrm{NIL} & 4 & \mathrm{NIL} & \mathrm{NIL} \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 5 & \mathrm{NIL} \end{bmatrix} \end{array}$$

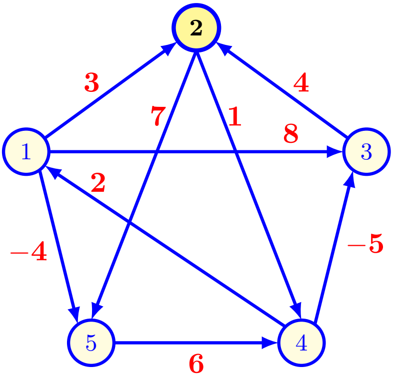
Illustrating Floyd's Algorithm with Predecessors: $k = 1$



$$D^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{array}$$

$$\Pi^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \mathrm{NIL} & 1 & 1 & \mathrm{NIL} & 1 \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 2 & 2 \\ \mathrm{NIL} & 3 & \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} \\ 4 & 1 & 4 & \mathrm{NIL} & 1 \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 5 & \mathrm{NIL} \end{bmatrix} \end{array}$$

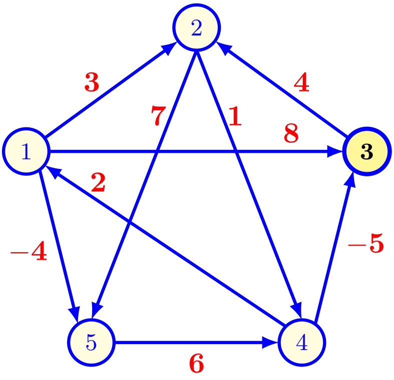
Illustrating Floyd's Algorithm with Predecessors: $k = 2$



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

$$\Pi^{(2)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{bmatrix} \end{array}$$

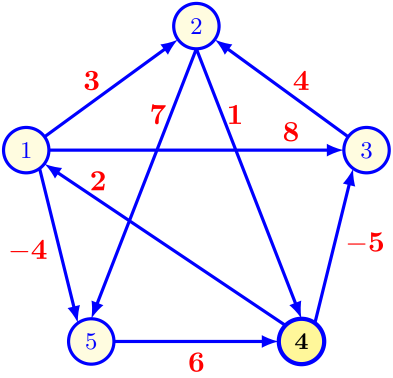
Illustrating Floyd's Algorithm with Predecessors: $k = 3$



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

$$\Pi^{(3)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{bmatrix} \end{array}$$

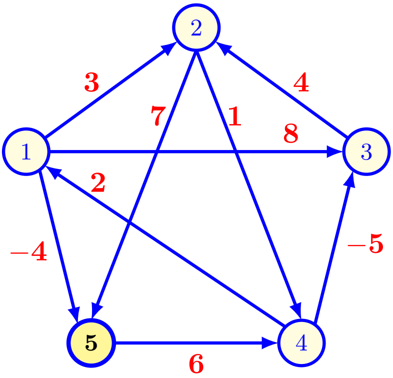
Illustrating Floyd's Algorithm with Predecessors: $k = 4$



$$D^{(4)} = \begin{array}{cc} & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} \end{array}$$

$$\Pi^{(4)} = \begin{array}{cc} & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{bmatrix} \end{array}$$

Illustrating Floyd's Algorithm with Predecessors: $k = 5$



$$D^{(5)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & -3 & 2 & -4 \\ 2 & 3 & 0 & -4 & 1 & -1 \\ 3 & 7 & 4 & 0 & 5 & 3 \\ 4 & 2 & -1 & -5 & 0 & -2 \\ 5 & 8 & 5 & 1 & 6 & 0 \end{array}$$

$$\Pi^{(5)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{NIL} & 3 & 4 & 5 & 1 \\ 2 & 4 & \text{NIL} & 4 & 2 & 1 \\ 3 & 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 4 & 3 & 4 & \text{NIL} & 1 \\ 5 & 4 & 3 & 4 & 5 & \text{NIL} \end{array}$$

Getting Shortest Paths from Π Matrix

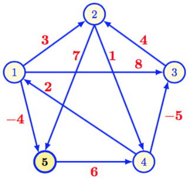
The recursive algorithm using the predecessor matrix $\Pi = \Pi^{(n)}$ to print **the shortest path** between vertices i and j:

algorithm `PrintPath`(Π, i, j)

if $i = j$ **then print** i
else
    if $\pi_{i,j} = \mathrm{NIL}$ **then print** "no path from i to j"
    else
        `PrintPath`(Π, i, $\pi_{i,j}$)
        print j
    end if
end if

Illustrating PrintPath Algorithm

$$\Pi^{(5)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{bmatrix} \end{array}$$



`PrintPath`($\Pi^{(5)}$, 5, 3)
→ `PrintPath`($\Pi^{(5)}$, 5, $\pi_{5,3} = 4$)
  → `PrintPath`($\Pi^{(5)}$, 5, $\pi_{5,4} = 5$)
    print 5
  print 4
print 3

`PrintPath`($\Pi^{(5)}$, 1, 2)
→ `PrintPath`($\Pi^{(5)}$, 1, $\pi_{1,2} = 3$)
  → `PrintPath`($\Pi^{(5)}$, 1, $\pi_{1,3} = 4$)
    → `PrintPath`($\Pi^{(5)}$, 1, $\pi_{1,4} = 5$)
      → `PrintPath`($\Pi^{(5)}$, 1, $\pi_{1,5} = 1$)
        print 1
      print 5
    print 4
  print 3
print 2