CSE 431/531: Algorithm Analysis and Design (Fall 2024)
Graph Algorithms

Lecturer: Kelin Luo

Department of Computer Science and Engineering
University at Buffalo

Outline

1. Minimum Spanning Tree
   - Kruskal's Algorithm
   - Reverse-Kruskal's Algorithm
   - Prim's Algorithm

2. Single Source Shortest Paths

3. All-Pair Shortest Paths and Floyd-Warshall

Outline

1. Minimum Spanning Tree
   - Kruskal's Algorithm
   - Reverse-Kruskal's Algorithm
   - Prim's Algorithm
2. Single Source Shortest Paths
3. All-Pair Shortest Paths and Floyd-Warshall

Outline

1. Minimum Spanning Tree
   - Kruskal's Algorithm
   - Reverse-Kruskal's Algorithm
   - Prim's Algorithm

2. Single Source Shortest Paths

3. All-Pair Shortest Paths and Floyd-Warshall

Outline

1. Minimum Spanning Tree
   - Kruskal's Algorithm
   - Reverse-Kruskal's Algorithm
   - Prim's Algorithm
2. Single Source Shortest Paths
3. All-Pair Shortest Paths and Floyd-Warshall

Outline

1. Minimum Spanning Tree
   - Kruskal's Algorithm
   - Reverse-Kruskal's Algorithm
   - Prim's Algorithm
2. Single Source Shortest Paths
3. All-Pair Shortest Paths and Floyd-Warshall

dynamic-programming(G, w, s)

$f^0[s] \leftarrow 0$ and $f^0[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to $n-1$ **do**
    copy $f^{\ell-1} \to f^{\ell}$
    for each $(u, v) \in E$ **do**
        if $f^{\ell-1}[u] + w(u, v) < f^{\ell}[v]$ **then**
            $f^{\ell}[v] \leftarrow f^{\ell-1}[u] + w(u, v)$
return $(f^{n-1}[v])_{v \in V}$

dynamic-programming(G, w, s)

$f^0[s] \leftarrow 0$ and $f^0[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to $n-1$ **do**
    copy $f^{\ell-1} \to f^{\ell}$
    for each $(u, v) \in E$ **do**
        if $f^{\ell-1}[u] + w(u, v) < f^{\ell}[v]$ **then**
            $f^{\ell}[v] \leftarrow f^{\ell-1}[u] + w(u, v)$
return $(f^{n-1}[v])_{v \in V}$

Obs. Assuming there are no negative cycles, then a shortest path contains at most $n-1$ edges

dynamic-programming(G, w, s)

```
f^0[s] ← 0 and f^0[v] ← ∞ for any v ∈ V \ {s}
for ℓ ← 1 to n − 1 do
    copy f^{ℓ−1} → f^ℓ
    for each (u, v) ∈ E do
        if f^{ℓ−1}[u] + w(u, v) < f^ℓ[v] then
            f^ℓ[v] ← f^{ℓ−1}[u] + w(u, v)
return (f^{n−1}[v])_{v∈V}
```

Obs. Assuming there are no negative cycles, then a shortest path contains at most $n-1$ edges

Proof.

If there is a path containing at least n edges, then it contains a cycle. Removing the cycle gives a path with the same or smaller length. $\square$

Dynamic Programming with Better Space Usage

dynamic-programming(G, w, s)

$f^{\text{old}}[s] \leftarrow 0$ and $f^{\text{old}}[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to $n-1$ **do**
    copy $f^{\text{old}} \rightarrow f^{\text{new}}$
    for each $(u, v) \in E$ **do**
        if $f^{\text{old}}[u] + w(u, v) < f^{\text{new}}[v]$ **then**
            $f^{\text{new}}[v] \leftarrow f^{\text{old}}[u] + w(u, v)$
    copy $f^{\text{new}} \rightarrow f^{\text{old}}$
return f^{old}

- f^{ℓ} only depends on $f^{\ell-1}$: only need 2 vectors

Dynamic Programming with Better Space Usage

dynamic-programming(G, w, s)

$f^{\text{old}}[s] \leftarrow 0$ and $f^{\text{old}}[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to $n-1$ **do**
  copy $f^{\text{old}} \to f^{\text{new}}$
  for each $(u, v) \in E$ **do**
    if $f^{\text{old}}[u] + w(u, v) < f^{\text{new}}[v]$ **then**
      $f^{\text{new}}[v] \leftarrow f^{\text{old}}[u] + w(u, v)$
  copy $f^{\text{new}} \to f^{\text{old}}$
return f^{old}

- f^{ℓ} only depends on $f^{\ell-1}$: only need 2 vectors
- only need 1 vector!

Dynamic Programming with Better Space Usage

dynamic-programming(G, w, s)

```
f[s] ← 0 and f[v] ← ∞ for any v ∈ V \ {s}
for ℓ ← 1 to n − 1 do
    copy f → f
    for each (u, v) ∈ E do
        if f[u] + w(u, v) < f[v] then
            f[v] ← f[u] + w(u, v)
    copy f → f
return f
```

- f^{ℓ} only depends on $f^{\ell-1}$: only need 2 vectors
- only need 1 vector!

Dynamic Programming with Better Space Usage

dynamic-programming(G, w, s)

$f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to $n-1$ **do**
  for each $(u, v) \in E$ **do**
    if $f[u] + w(u, v) < f[v]$ **then**
      $f[v] \leftarrow f[u] + w(u, v)$
return f

- f^{ℓ} only depends on $f^{\ell-1}$: only need 2 vectors
- only need 1 vector!

Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

```
f[s] ← 0 and f[v] ← ∞ for any v ∈ V \ {s}
for ℓ ← 1 to n − 1 do
    for each (u, v) ∈ E do
        if f[u] + w(u, v) < f[v] then
            f[v] ← f[u] + w(u, v)
return f
```

- f^{ℓ} only depends on $f^{\ell-1}$: only need 2 vectors
- only need 1 vector!

Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

$f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to $n-1$ **do**
  for each $(u, v) \in E$ **do**
    if $f[u] + w(u, v) < f[v]$ **then**
      $f[v] \leftarrow f[u] + w(u, v)$
return f

- Issue: when we compute $f[u] + w(u, v)$, $f[u]$ may be changed since the end of last iteration

Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

```
f[s] ← 0 and f[v] ← ∞ for any v ∈ V \ {s}
for ℓ ← 1 to n − 1 do
    for each (u, v) ∈ E do
        if f[u] + w(u, v) < f[v] then
            f[v] ← f[u] + w(u, v)
return f
```

- Issue: when we compute $f[u] + w(u,v)$, $f[u]$ may be changed since the end of last iteration
- This is OK: it can only "accelerate" the process!

Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

```
1: f[s] ← 0 and f[v] ← ∞ for any v ∈ V \ {s}
2: for ℓ ← 1 to n − 1 do
3:     for each (u, v) ∈ E do
4:         if f[u] + w(u, v) < f[v] then
5:             f[v] ← f[u] + w(u, v)
6: return f
```

- Issue: when we compute $f[u] + w(u, v)$, $f[u]$ may be changed since the end of last iteration
- This is OK: it can only "accelerate" the process!
- After iteration ℓ, $f[v]$ is at most the length of the shortest path from s to v that uses at most ℓ edges

Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

$f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to $n-1$ **do**
  for each $(u, v) \in E$ **do**
    if $f[u] + w(u, v) < f[v]$ **then**
      $f[v] \leftarrow f[u] + w(u, v)$
return f

- Issue: when we compute $f[u] + w(u, v)$, $f[u]$ may be changed since the end of last iteration
- This is OK: it can only "accelerate" the process!
- After iteration ℓ, $f[v]$ is at most the length of the shortest path from s to v that uses at most ℓ edges
- $f[v]$ is always the length of some path from s to v

Bellman-Ford Algorithm

- After iteration ℓ:

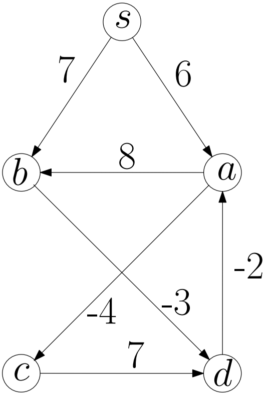
$$\begin{aligned} &\text{length of shortest } s\text{-}v \text{ path} \\ \le\ & f[v] \\ \le\ & \text{length of shortest } s\text{-}v \text{ path using at most } \ell \text{ edges} \end{aligned}$$

- Assuming there are no negative cycles:

$$\begin{aligned} &\text{length of shortest } s\text{-}v \text{ path} \\ =\ & \text{length of shortest } s\text{-}v \text{ path using at most } n-1 \text{ edges} \end{aligned}$$

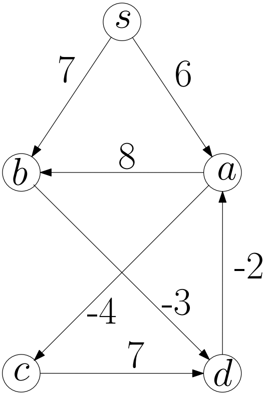
- So, assuming there are no negative cycles, after iteration $n-1$:

$$f[v] = \text{length of shortest } s\text{-}v \text{ path}$$



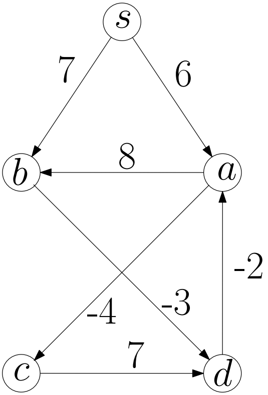
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	∞	∞	∞	∞



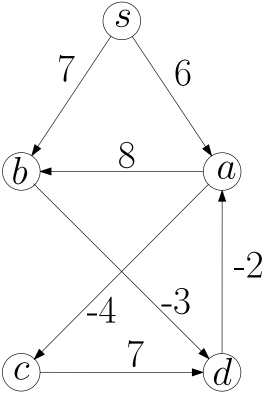
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	∞	∞	∞	∞



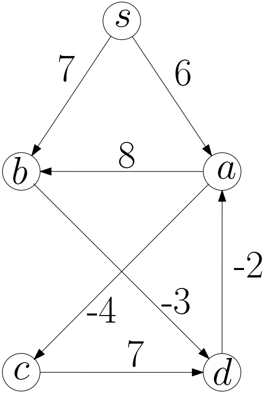
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	∞	∞	∞



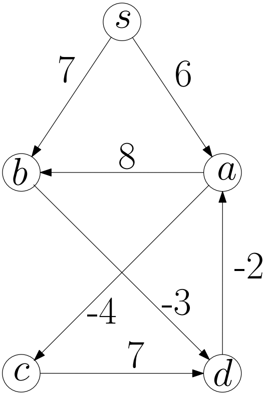
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	∞	∞	∞



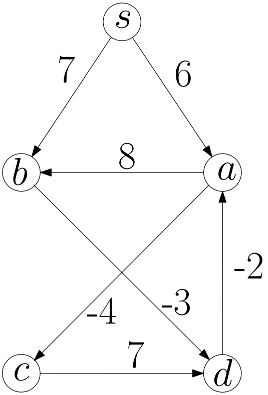
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	∞	∞



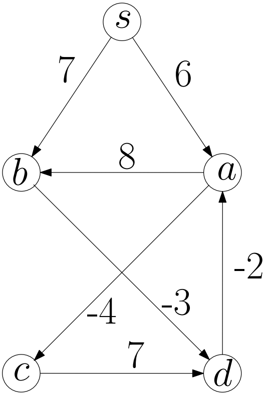
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	∞	∞



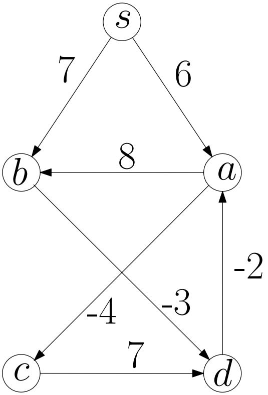
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	∞	∞



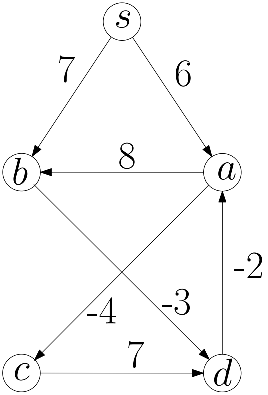
- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	6	7	2	∞



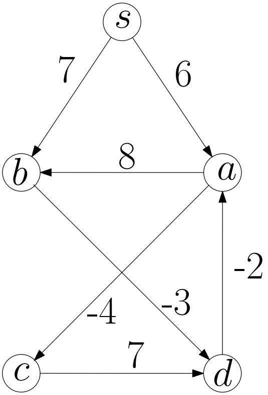
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	2	∞



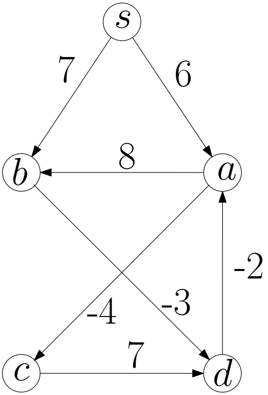
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	2	4



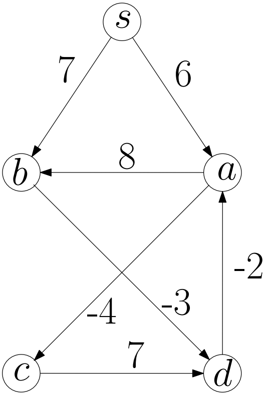
- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	6	7	2	4



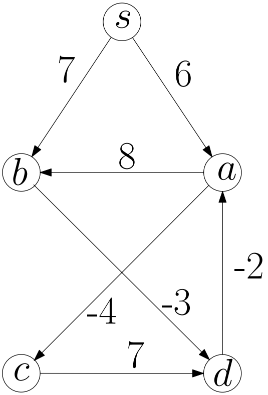
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	2	4



- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

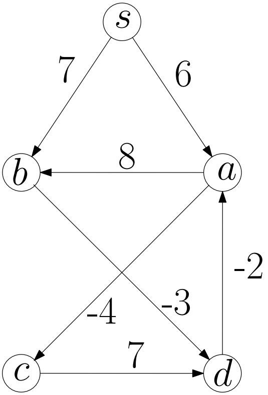
vertices	s	a	b	c	d
f	0	2	7	2	4



- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	2	7	2	4

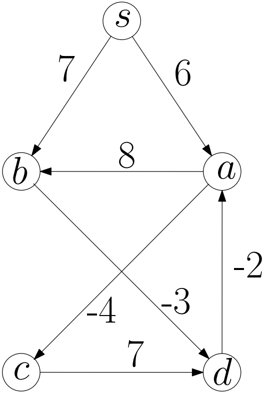
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	2	4

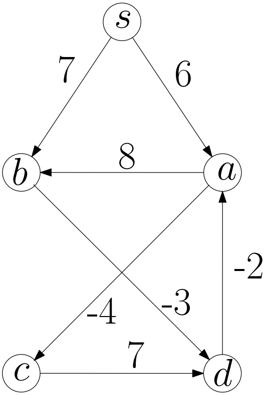
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	2	4

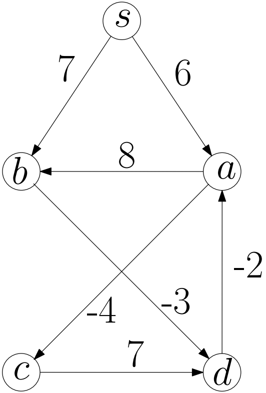
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	2	7	2	4

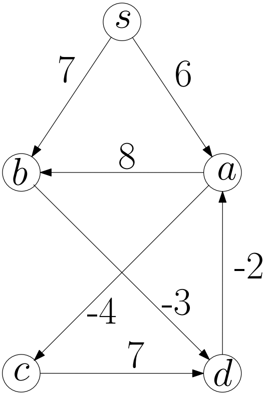
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	2	7	2	4

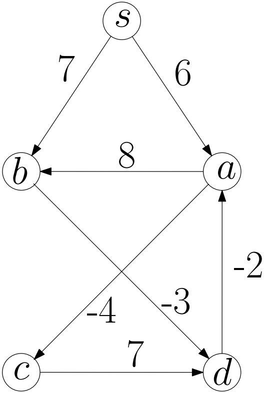
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

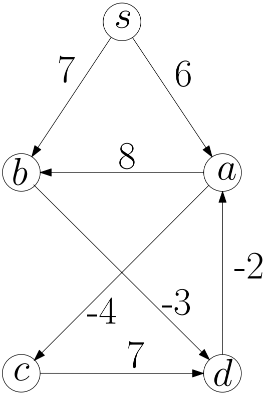
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

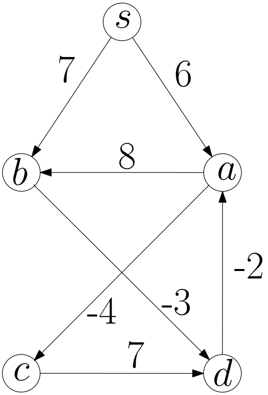
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

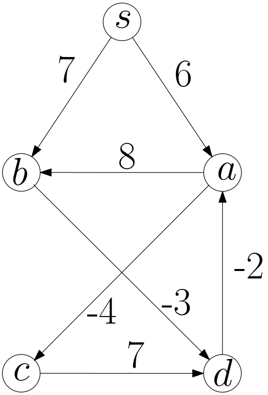
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

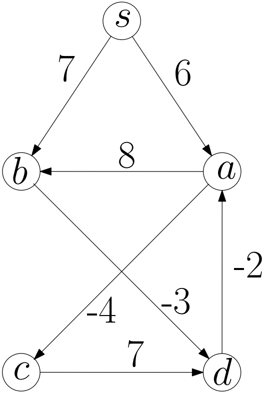
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

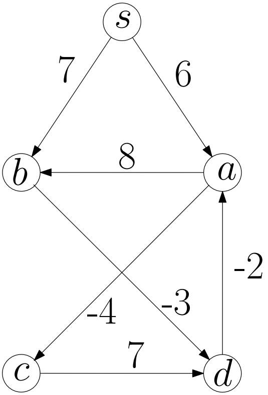
- end of iteration 1: 0, 2, 7, 2, 4
- end of iteration 2: 0, 2, 7, -2, 4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

- end of iteration 1: 0, 2, 7, 2, 4
- end of iteration 2: 0, 2, 7, -2, 4
- end of iteration 3: 0, 2, 7, -2, 4



- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

- end of iteration 1: 0, 2, 7, 2, 4
- end of iteration 2: 0, 2, 7, -2, 4
- end of iteration 3: 0, 2, 7, -2, 4
- Algorithm terminates in 3 iterations, instead of 4.

Bellman-Ford(G, w, s)

$f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to n **do**
    $updated \leftarrow$ false
    for each $(u, v) \in E$ **do**
        if $f[u] + w(u, v) < f[v]$ **then**
            $f[v] \leftarrow f[u] + w(u, v)$
            $updated \leftarrow$ true
    if not $updated$, then return f
output "negative cycle exists"

Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

$f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to n **do**
    $updated \leftarrow$ false
    for each $(u, v) \in E$ **do**
        if $f[u] + w(u, v) < f[v]$ **then**
            $f[v] \leftarrow f[u] + w(u, v)$, $\pi[v] \leftarrow u$
            $updated \leftarrow$ true
    if not $updated$, then return f
output "negative cycle exists"

- $\pi[v]$: the parent of v in the shortest path tree

Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

$f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to n **do**
  $\textit{updated} \leftarrow$ false
  for each $(u, v) \in E$ **do**
    if $f[u] + w(u, v) < f[v]$ **then**
      $f[v] \leftarrow f[u] + w(u, v)$, $\pi[v] \leftarrow u$
      $\textit{updated} \leftarrow$ true
  if not $\textit{updated}$, then return f
output "negative cycle exists"

- $\pi[v]$: the parent of v in the shortest path tree
- Running time $= O(nm)$

Outline

1. Minimum Spanning Tree
   - Kruskal's Algorithm
   - Reverse-Kruskal's Algorithm
   - Prim's Algorithm
2. Single Source Shortest Paths
3. All-Pair Shortest Paths and Floyd-Warshall

All-Pair Shortest Paths

All Pair Shortest Paths

Input: directed graph $G = (V, E)$,

$w : E \to \mathbb{R}$ (can be negative)

Output: shortest path from u to v for every $u, v \in V$

All-Pair Shortest Paths

All Pair Shortest Paths

Input: directed graph $G = (V, E)$,

$w : E \to \mathbb{R}$ (can be negative)

Output: shortest path from u to v for every $u, v \in V$

```
1: for every starting point s ∈ V do
2:     run Bellman-Ford(G, w, s)
```

All-Pair Shortest Paths

All Pair Shortest Paths

Input: directed graph $G = (V, E)$,

$w : E \to \mathbb{R}$ (can be negative)

Output: shortest path from u to v for every $u, v \in V$

1: **for** every starting point $s \in V$ **do**
2: run Bellman-Ford(G, w, s)

- Running time $= O(n^2 m)$

Summary of Shortest Path Algorithms we learned

algorithm	graph	weights	SS?	running time
Simple DP	DAG	$\mathbb{R}$	SS	$O(n+m)$
Dijkstra	U/D	$\mathbb{R}_{\geq 0}$	SS	$O(n \log n + m)$
Bellman-Ford	U/D	$\mathbb{R}$	SS	$O(nm)$
Floyd-Warshall	U/D	$\mathbb{R}$	AP	$O(n^3)$

- DAG = directed acyclic graph U = undirected D = directed
- SS = single source AP = all pairs

Design a Dynamic Programming Algorithm

- It is convenient to assume $V = \{1, 2, 3, \cdots, n\}$

Design a Dynamic Programming Algorithm

- It is convenient to assume $V = \{1, 2, 3, \cdots, n\}$
- For simplicity, extend the w values to non-edges:

$$w(i,j) = \begin{cases} 0 & i = j \\ \text{weight of edge } (i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

Design a Dynamic Programming Algorithm

- It is convenient to assume $V = \{1, 2, 3, \cdots, n\}$
- For simplicity, extend the w values to non-edges:

$$w(i,j) = \begin{cases} 0 & i = j \\ \text{weight of edge } (i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

- For now assume there are no negative cycles

Design a Dynamic Programming Algorithm

- It is convenient to assume $V = \{1, 2, 3, \cdots, n\}$
- For simplicity, extend the w values to non-edges:

$$w(i,j) = \begin{cases} 0 & i = j \\ \text{weight of edge } (i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

- For now assume there are no negative cycles

Cells for Floyd-Warshall Algorithm

Design a Dynamic Programming Algorithm

- It is convenient to assume $V = \{1, 2, 3, \cdots, n\}$
- For simplicity, extend the w values to non-edges:

$$w(i,j) = \begin{cases} 0 & i = j \\ \text{weight of edge } (i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

- For now assume there are no negative cycles

Cells for Floyd-Warshall Algorithm

- First try: $f[i,j]$ is length of shortest path from i to j

Design a Dynamic Programming Algorithm

- It is convenient to assume $V = \{1, 2, 3, \cdots, n\}$
- For simplicity, extend the w values to non-edges:

$$w(i,j) = \begin{cases} 0 & i = j \\ \text{weight of edge } (i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

- For now assume there are no negative cycles

Cells for Floyd-Warshall Algorithm

- First try: $f[i,j]$ is length of shortest path from i to j
- Issue: do not know in which order we compute $f[i,j]$'s

Design a Dynamic Programming Algorithm

- It is convenient to assume $V = \{1, 2, 3, \cdots, n\}$
- For simplicity, extend the w values to non-edges:

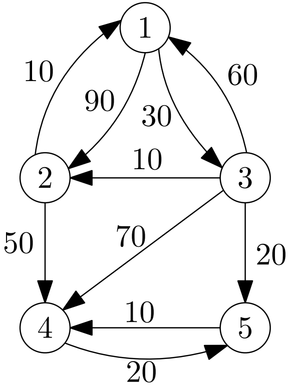
$$w(i,j) = \begin{cases} 0 & i = j \\ \text{weight of edge } (i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

- For now assume there are no negative cycles

Cells for Floyd-Warshall Algorithm

- First try: $f[i,j]$ is length of shortest path from i to j
- Issue: do not know in which order we compute $f[i,j]$'s
- $f^k[i,j]$: length of shortest path from i to j that only uses vertices $\{1, 2, 3, \cdots, k\}$ as intermediate vertices

Example for Definition of $f^k[i,j]$'s



$$
\begin{aligned}
f^0[1,4] &= \infty \\
f^1[1,4] &= \infty \\
f^2[1,4] &= 140 && (1 \to 2 \to 4) \\
f^3[1,4] &= 90 && (1 \to 3 \to 2 \to 4) \\
f^4[1,4] &= 90 && (1 \to 3 \to 2 \to 4) \\
f^5[1,4] &= 60 && (1 \to 3 \to 5 \to 4)
\end{aligned}
$$

$$w(i,j) = \begin{cases} 0 & i = j \\ \text{weight of edge } (i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

- $f^k[i,j]$: length of shortest path from i to j that only uses vertices $\{1, 2, 3, \cdots, k\}$ as intermediate vertices

$$w(i,j) = \begin{cases} 0 & i = j \\ \text{weight of edge } (i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

- $f^k[i,j]$: length of shortest path from i to j that only uses vertices $\{1, 2, 3, \cdots, k\}$ as intermediate vertices

$$f^k[i,j] = \begin{cases} & k = 0 \\ & k = 1, 2, \cdots, n \end{cases}$$

$$w(i,j) = \begin{cases} 0 & i = j \\ \text{weight of edge } (i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

- $f^k[i,j]$: length of shortest path from i to j that only uses vertices $\{1, 2, 3, \cdots, k\}$ as intermediate vertices

$$f^k[i,j] = \begin{cases} w(i,j) & k = 0 \\ & k = 1, 2, \cdots, n \end{cases}$$

$$w(i,j) = \begin{cases} 0 & i = j \\ \text{weight of edge } (i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

- $f^k[i,j]$: length of shortest path from i to j that only uses vertices $\{1,2,3,\cdots,k\}$ as intermediate vertices

$$f^k[i,j] = \begin{cases} w(i,j) & k = 0 \\ \min \left\{ \right. & k = 1,2,\cdots,n \end{cases}$$

$$w(i,j) = \begin{cases} 0 & i = j \\ \text{weight of edge } (i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

- $f^k[i,j]$: length of shortest path from i to j that only uses vertices $\{1, 2, 3, \cdots, k\}$ as intermediate vertices

$$f^k[i,j] = \begin{cases} w(i,j) & k = 0 \\ \min \begin{cases} f^{k-1}[i,j] \\ \\ \end{cases} & k = 1, 2, \cdots, n \end{cases}$$

$$w(i,j) = \begin{cases} 0 & i = j \\ \text{weight of edge } (i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

- $f^k[i,j]$: length of shortest path from i to j that only uses vertices $\{1, 2, 3, \cdots, k\}$ as intermediate vertices

$$f^k[i,j] = \begin{cases} w(i,j) & k = 0 \\ \min \begin{cases} f^{k-1}[i,j] \\ f^{k-1}[i,k] + f^{k-1}[k,j] \end{cases} & k = 1, 2, \cdots, n \end{cases}$$

Floyd-Warshall(G, w)

$f^0 \leftarrow w$
for $k \leftarrow 1$ to n **do**
    copy $f^{k-1} \rightarrow f^k$
    for $i \leftarrow 1$ to n **do**
        for $j \leftarrow 1$ to n **do**
            if $f^{k-1}[i,k] + f^{k-1}[k,j] < f^k[i,j]$ **then**
                $f^k[i,j] \leftarrow f^{k-1}[i,k] + f^{k-1}[k,j]$

Floyd-Warshall(G, w)

```
f^old ← w
for k ← 1 to n do
    copy f^old → f^new
    for i ← 1 to n do
        for j ← 1 to n do
            if f^old[i,k] + f^old[k,j] < f^new[i,j] then
                f^new[i,j] ← f^old[i,k] + f^old[k,j]
```

Floyd-Warshall(G, w)

$f^{\text{old}} \leftarrow w$
for $k \leftarrow 1$ to n **do**
    copy $f^{\text{old}} \rightarrow f^{\text{new}}$
    for $i \leftarrow 1$ to n **do**
        for $j \leftarrow 1$ to n **do**
            if $f^{\text{old}}[i,k] + f^{\text{old}}[k,j] < f^{\text{new}}[i,j]$ **then**
                $f^{\text{new}}[i,j] \leftarrow f^{\text{old}}[i,k] + f^{\text{old}}[k,j]$

Floyd-Warshall(G, w)

```
f ← w
for k ← 1 to n do
    copy f → f
    for i ← 1 to n do
        for j ← 1 to n do
            if f[i, k] + f[k, j] < f[i, j] then
                f[i, j] ← f[i, k] + f[k, j]
```

Floyd-Warshall(G, w)

$f \leftarrow w$
for $k \leftarrow 1$ to n **do**
    for $i \leftarrow 1$ to n **do**
        for $j \leftarrow 1$ to n **do**
            if $f[i,k] + f[k,j] < f[i,j]$ **then**
                $f[i,j] \leftarrow f[i,k] + f[k,j]$

Floyd-Warshall(G, w)

```
f ← w
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            if f[i,k] + f[k,j] < f[i,j] then
                f[i,j] ← f[i,k] + f[k,j]
```

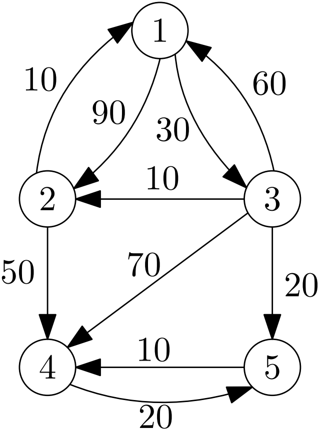
Lemma Assume there are no negative cycles in G. After iteration k, for $i, j \in V$, $f[i, j]$ is exactly the length of shortest path from i to j that only uses vertices in $\{1, 2, 3, \cdots, k\}$ as intermediate vertices.

Floyd-Warshall(G, w)

```
f ← w
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            if f[i,k] + f[k,j] < f[i,j] then
                f[i,j] ← f[i,k] + f[k,j]
```

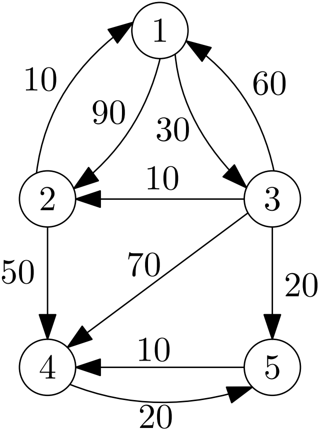
Lemma Assume there are no negative cycles in G. After iteration k, for $i, j \in V$, $f[i,j]$ is exactly the length of shortest path from i to j that only uses vertices in $\{1, 2, 3, \cdots, k\}$ as intermediate vertices.

- Running time $= O(n^3)$.



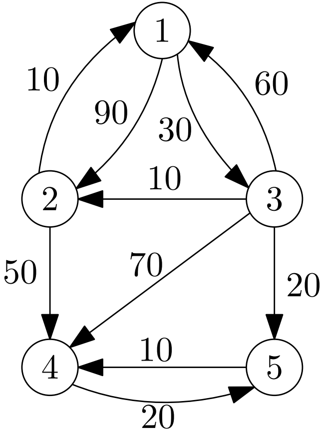
	1	2	3	4	5
1	0	90	30	∞	∞
2	10	0	∞	50	∞
3	60	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

-



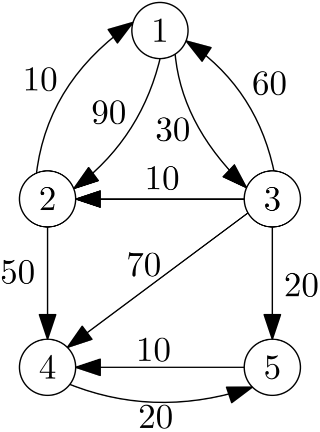
	1	2	3	4	5
1	0	90	30	∞	∞
2	10	0	∞	50	∞
3	60	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

- $i = 2,\ k = 1,\ j = 3$



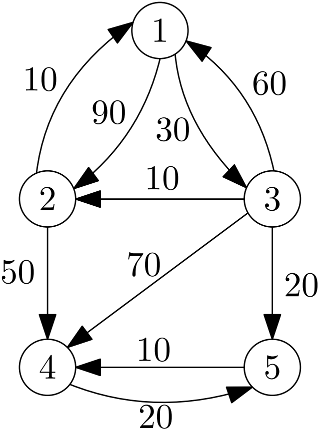
	1	2	3	4	5
1	0	90	30	∞	∞
2	10	0	40	50	∞
3	60	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

- $i = 2,\ k = 1,\ j = 3$



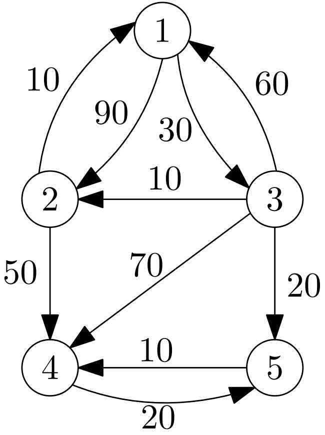
	1	2	3	4	5
1	0	90	30	∞	∞
2	10	0	40	50	∞
3	60	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

- $i = 1,\ k = 2,\ j = 4$



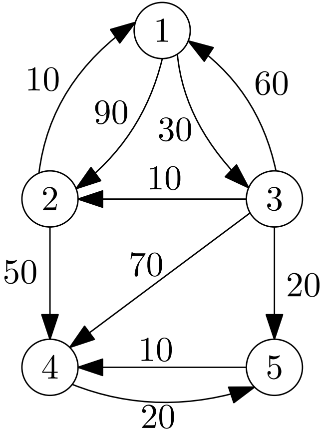
	1	2	3	4	5
1	0	90	30	140	∞
2	10	0	40	50	∞
3	60	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

- $i = 1,\ k = 2,\ j = 4$



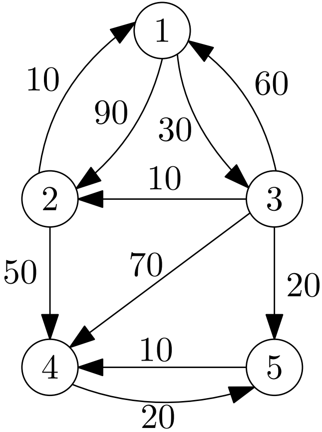
	1	2	3	4	5
1	0	90	30	140	∞
2	10	0	40	50	∞
3	60	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

- $i = 3,\ k = 2,\ j = 1,$



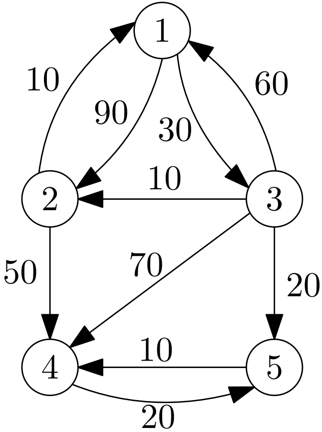
	1	2	3	4	5
1	0	90	30	140	∞
2	10	0	40	50	∞
3	20	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

- $i = 3$, $k = 2$, $j = 1$,



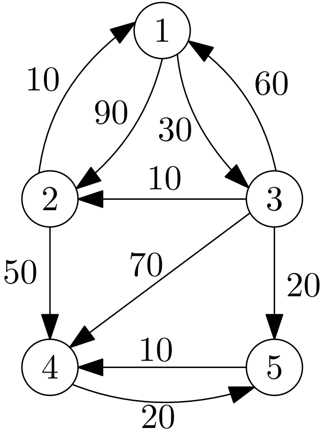
	1	2	3	4	5
1	0	90	30	140	∞
2	10	0	40	50	∞
3	20	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

- $i = 3,\ k = 2,\ j = 4$



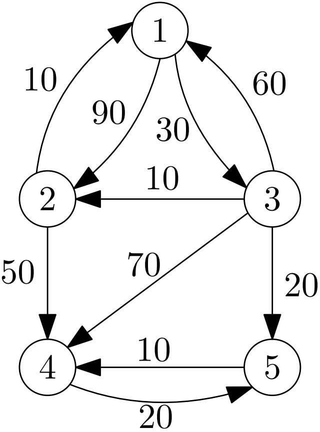
	1	2	3	4	5
1	0	90	30	140	∞
2	10	0	40	50	∞
3	20	10	0	60	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

- $i = 3,\ k = 2,\ j = 4$



	1	2	3	4	5
1	0	90	30	140	∞
2	10	0	40	50	∞
3	20	10	0	60	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

- $i = 1,\ k = 3,\ j = 2$



	1	2	3	4	5
1	0	40	30	140	∞
2	10	0	40	50	∞
3	20	10	0	60	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

- $i = 1,\ k = 3,\ j = 2$