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:

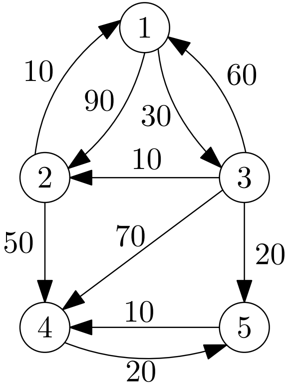
$$w(i,j) = \begin{cases} 0 & i = j \\ \text{weight of edge } (i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

- 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



$$f^0[1,4] = \infty$$

$$f^1[1,4] = \infty$$

$$f^2[1,4] = 140 \qquad (1 \to 2 \to 4)$$

$$f^3[1,4] = 90 \qquad (1 \to 3 \to 2 \to 4)$$

$$f^4[1,4] = 90 \qquad (1 \to 3 \to 2 \to 4)$$

$$f^5[1,4] = 60 \qquad (1 \to 3 \to 5 \to 4)$$

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

- $f^k[i,j]$: 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)

$f^{\text{old}} \leftarrow w$
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)

$f^{\text{old}} \leftarrow w$
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)

$f \leftarrow w$
for $k \leftarrow 1$ to n **do**
  copy $f \rightarrow f$
  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)

$f \leftarrow w$
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]
```

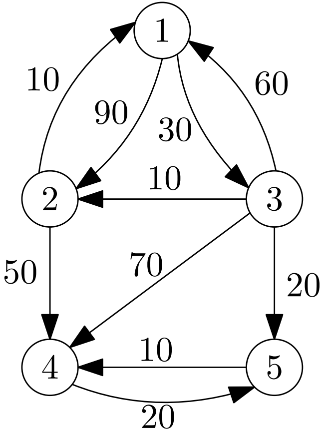
Lemma Assume there are no negative cycles in G. After iteration k, for $i, j \in V$, $f[i, j]$ is exactly the length of shortest path from i to j that only uses vertices in $\{1, 2, 3, \cdots, k\}$ as intermediate vertices.

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]
```

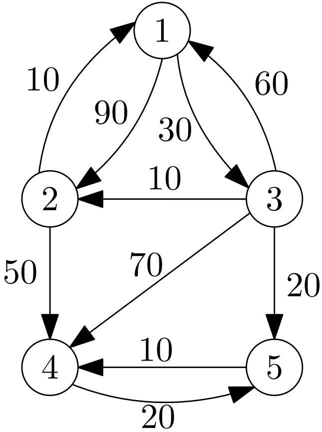
Lemma Assume there are no negative cycles in G. After iteration k, for $i, j \in V$, $f[i, j]$ is exactly the length of shortest path from i to j that only uses vertices in $\{1, 2, 3, \cdots, k\}$ as intermediate vertices.

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



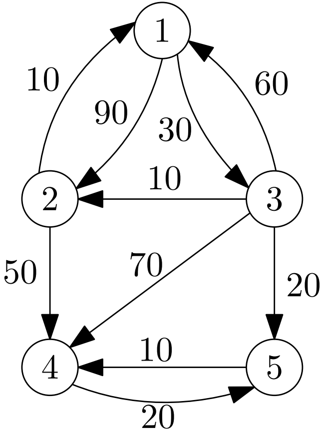
	1	2	3	4	5
1	0	90	30	∞	∞
2	10	0	∞	50	∞
3	60	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

-



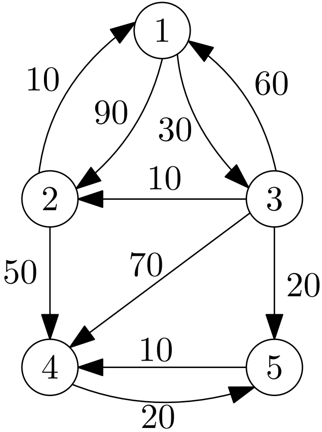
	1	2	3	4	5
1	0	90	30	∞	∞
2	10	0	∞	50	∞
3	60	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

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



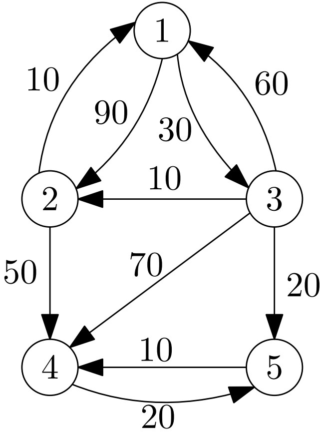
	1	2	3	4	5
1	0	90	30	∞	∞
2	10	0	40	50	∞
3	60	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

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



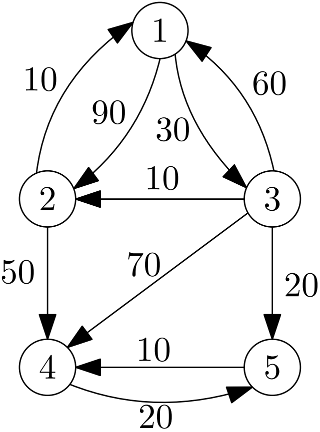
	1	2	3	4	5
1	0	90	30	∞	∞
2	10	0	40	50	∞
3	60	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

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



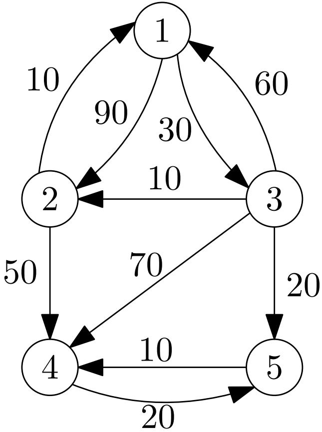
	1	2	3	4	5
1	0	90	30	140	∞
2	10	0	40	50	∞
3	60	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

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



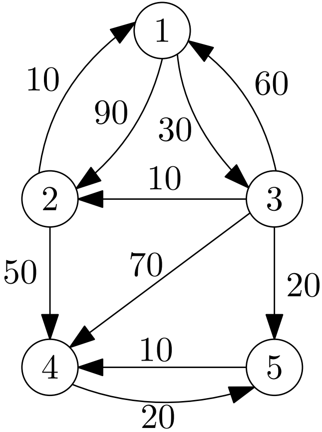
	1	2	3	4	5
1	0	90	30	140	∞
2	10	0	40	50	∞
3	60	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

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



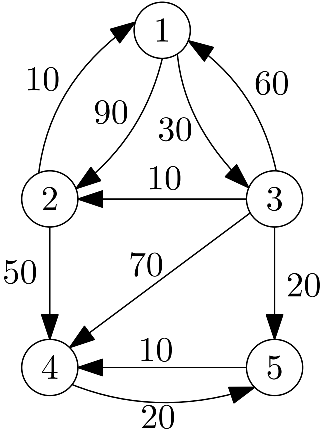
	1	2	3	4	5
1	0	90	30	140	∞
2	10	0	40	50	∞
3	20	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

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



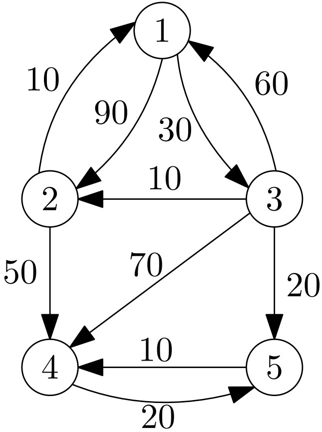
	1	2	3	4	5
1	0	90	30	140	∞
2	10	0	40	50	∞
3	20	10	0	70	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

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



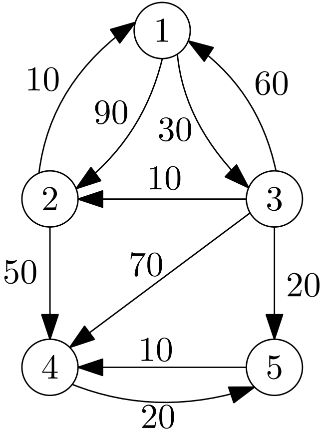
	1	2	3	4	5
1	0	90	30	140	∞
2	10	0	40	50	∞
3	20	10	0	60	20
4	∞	∞	∞	0	20
5	∞	∞	∞	10	0

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



	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

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

Recovering Shortest Paths

Floyd-Warshall(G, w)

$f \leftarrow w$, $\pi[i,j] \leftarrow \perp$ for every $i, j \in V$
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**
                $f[i,j] \leftarrow f[i,k] + f[k,j]$, $\pi[i,j] \leftarrow k$

Recovering Shortest Paths

Floyd-Warshall(G, w)

$f \leftarrow w$, $\pi[i,j] \leftarrow \perp$ for every $i, j \in V$
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**
        $f[i,j] \leftarrow f[i,k] + f[k,j]$, $\pi[i,j] \leftarrow k$

print-path(i, j)

if $\pi[i,j] = \perp$ **then** then
  if $i \neq j$ **then** print$(i, \text{``,''})$
else
  print-path$(i, \pi[i,j])$, print-path$(\pi[i,j], j)$

Detecting Negative Cycles

Floyd-Warshall(G, w)

1: $f \leftarrow w$, $\pi[i, j] \leftarrow \bot$ for every $i, j \in V$
2: **for** $k \leftarrow 1$ to n **do**
3:     **for** $i \leftarrow 1$ to n **do**
4:         **for** $j \leftarrow 1$ to n **do**
5:             **if** $f[i, k] + f[k, j] < f[i, j]$ **then**
6:                 $f[i, j] \leftarrow f[i, k] + f[k, j]$, $\pi[i, j] \leftarrow k$

Detecting Negative Cycles

Floyd-Warshall(G, w)

$f \leftarrow w$, $\pi[i,j] \leftarrow \bot$ for every $i, j \in V$
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**
                $f[i,j] \leftarrow f[i,k] + f[k,j]$, $\pi[i,j] \leftarrow k$
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**
                report "negative cycle exists" and exit

Summary of Shortest Path Algorithms

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

CSE 431/531: Algorithm Analysis and Design (Fall 2023)

NP-Completeness

Lecturer: Kelin Luo

Department of Computer Science and Engineering
University at Buffalo

NP-Completeness Theory

- The topics we discussed so far are positive results: how to design efficient algorithms for solving a given problem.
- NP-Completeness provides negative results: some problems can not be solved efficiently.

Q: Why do we study negative results?

NP-Completeness Theory

- The topics we discussed so far are positive results: how to design efficient algorithms for solving a given problem.
- NP-Completeness provides negative results: some problems can not be solved efficiently.

Q: Why do we study negative results?

- A given problem X cannot be solved in polynomial time.
- Without knowing it, you will have to keep trying to find polynomial time algorithm for solving X. All our efforts are doomed!

Efficient = Polynomial Time

- Polynomial time: $O(n^k)$ for any constant $k > 0$
- Example: $O(n), O(n^2), O(n^{2.5}\log n), O(n^{100})$
- Not polynomial time: $O(2^n), O(n^{\log n})$

Efficient = Polynomial Time

- Polynomial time: $O(n^k)$ for any constant $k > 0$
- Example: $O(n), O(n^2), O(n^{2.5}\log n), O(n^{100})$
- Not polynomial time: $O(2^n), O(n^{\log n})$
- Almost all algorithms we learnt so far run in polynomial time

Efficient = Polynomial Time

- Polynomial time: $O(n^k)$ for any constant $k > 0$
- Example: $O(n), O(n^2), O(n^{2.5}\log n), O(n^{100})$
- Not polynomial time: $O(2^n), O(n^{\log n})$
- Almost all algorithms we learnt so far run in polynomial time

Reason for Efficient = Polynomial Time

- For natural problems, if there is an $O(n^k)$-time algorithm, then k is small, say 4
- A good cut separating problems: for most natural problems, either we have a polynomial time algorithm, or the best algorithm runs in time $\Omega(2^{n^c})$ for some c
- Do not need to worry about the computational model

Outline

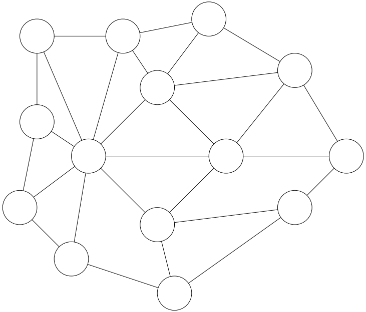
Example: Hamiltonian Cycle Problem

Def. Let G be an undirected graph. A Hamiltonian Cycle (HC) of G is a cycle C in G that passes each vertex of G exactly once.

Hamiltonian Cycle (HC) Problem

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

Output: whether G contains a Hamiltonian cycle



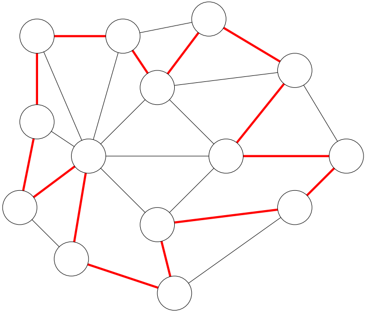
Example: Hamiltonian Cycle Problem

Def. Let G be an undirected graph. A Hamiltonian Cycle (HC) of G is a cycle C in G that passes each vertex of G exactly once.

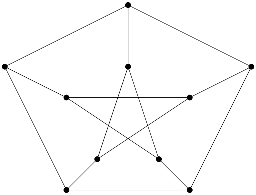
Hamiltonian Cycle (HC) Problem

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

Output: whether G contains a Hamiltonian cycle



Example: Hamiltonian Cycle Problem



- The graph is called the Petersen Graph. It has no HC.

Example: Hamiltonian Cycle Problem

Hamiltonian Cycle (HC) Problem

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

Output: whether G contains a Hamiltonian cycle

Example: Hamiltonian Cycle Problem

Hamiltonian Cycle (HC) Problem

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

Output: whether G contains a Hamiltonian cycle

Algorithm for Hamiltonian Cycle Problem:

- Enumerate all possible permutations, and check if it corresponds to a Hamiltonian Cycle

Example: Hamiltonian Cycle Problem

Hamiltonian Cycle (HC) Problem

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

Output: whether G contains a Hamiltonian cycle

Algorithm for Hamiltonian Cycle Problem:

- Enumerate all possible permutations, and check if it corresponds to a Hamiltonian Cycle
- Running time: $O(n!m) = 2^{O(n \lg n)}$
- Better algorithm: $2^{O(n)}$
- Far away from polynomial time

Example: Hamiltonian Cycle Problem

Hamiltonian Cycle (HC) Problem

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

Output: whether G contains a Hamiltonian cycle

Algorithm for Hamiltonian Cycle Problem:

- Enumerate all possible permutations, and check if it corresponds to a Hamiltonian Cycle
- Running time: $O(n!m) = 2^{O(n \lg n)}$
- Better algorithm: $2^{O(n)}$
- Far away from polynomial time
- HC is NP-hard: it is unlikely that it can be solved in polynomial time.