Advanced Algorithms

Lecture 2
All-Pairs Shortest Paths

Chenjuan Guo
cguo@cs.aau.dk

ILO of Lecture 2

- All-pairs shortest paths using dynamic programming
  - Adjacency matrix and distance/predecessor matrix
  - Repeated squaring and Floyd-Warshall algorithm.
  - Definition of transitive closure of a directed graph.

All-pairs shortest paths

- Shortest paths between all pairs of vertices in a graph.

- Why the problem is useful?
  - E.g., PostNord, FlexDanmark.

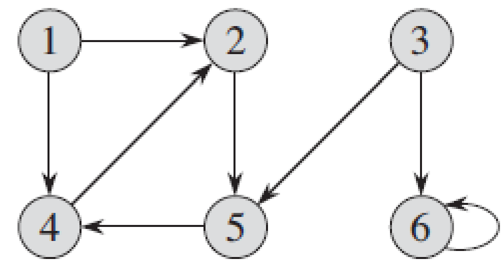
- How to solve the problem efficiently?
  - Repeatedly run one-to-all shortest paths |V| times.
  - Repeated squaring algorithm
  - Floyd-Warshall algorithm

Agenda

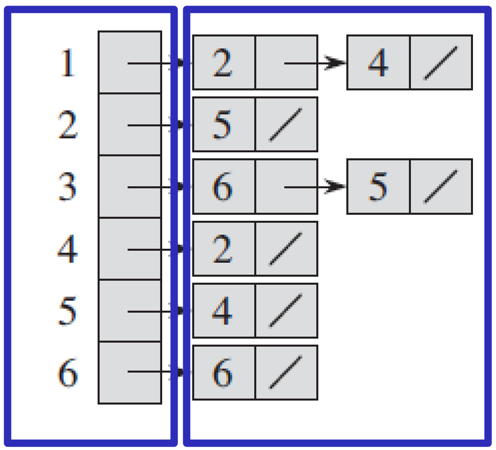
- Recall one-to-all shortest paths
- All-pairs shortest paths
- Repeated squaring algorithm
- Floyd-Warshall algorithm
- Transitive closure of a directed graph

Representing a graph

- Graph $G = (V, E, W)$
- Adjacency list vs. adjacency matrix



Space for the array: $\Theta(|V|)$



Space for the linked lists: $\Theta(|E|)$

Total space: $\Theta(|V|+|E|)$

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Space: $\Theta(|V|^2)$

- How to represent edge weights for a weighted graph?

One-to-All Shortest Path

- Input:
  - Directed, weighted graph `G = (V, E, W)`
  - Source vertex `s`.

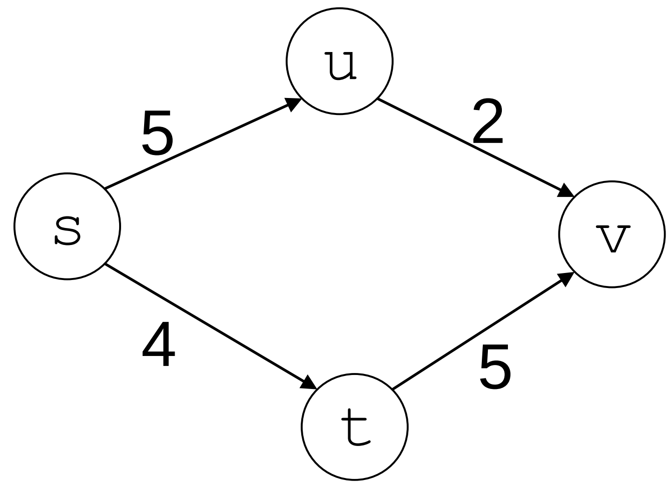
- The shortest-path weight

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

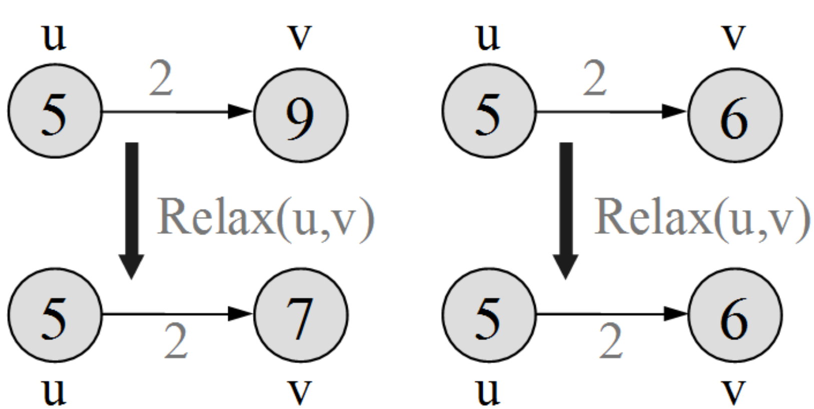
- Output:
  - A set of vertices `S`, `|S| = |V|`.
    - Each `u` ∈ `S`: `u.d()` and `u.parent()`.
    - Not reachable: `u.d()` = ∞.

Relaxation Technique

- Relaxing an edge `(u, v)`
  - `u.d() + w(u, v)` vs. `v.d()`



- Intuition:
  - Improve the existing shortest path from `s` to `v`.




```
Relax (u,v,G)
if v.d() > u.d()+G.w(u,v) then
    v.setd(u.d()+G.w(u,v))
    v.setparent(u)
```

Dijkstra's algorithm

Complexity depends on how to implement the min-priority queue
- binary min-heap

```
Dijkstra(G,s)
for each vertex u ∈ G.V()
    u.setd(∞)
    u.setparent(NIL)
s.setd(0)
S ←∅                    // Set S is used to explain the algorithm
Q.init(G.V())           // Q is a priority queue ADT
while not Q.isEmpty()
    u ← Q.extractMin()
    S ←S ∪ {u}
    for each v ∈ u.adjacent() do
        Relax(u, v, G)
        Q.modifyKey(v)
```

Initialize all vertices: $\Theta(|V|)$

Initialize Q: $O(|V|)$

$|V|$ times of `Q.extractMin()`

$|E|$ times of edge relax

$|E|$ times of `Q.modifyKey()`

One-to-all shortest paths: $O(|E|*\lg|V|)$

All-pairs shortest paths: $O(|V|*|E|*\lg|V|)$

Bellman-Ford

```
Bellman-Ford(G,s)
for each vertex u ∈ G.V()
    u.setd(∞)
    u.setparent(NIL)
s.setd(0)
for i ← 1 to |G.V()|-1 do
    for each edge (u,v) ∈ G.E() do
        Relax (u,v,G)
for each edge (u,v) ∈ G.E() do
    if v.d() > u.d() + G.w(u,v) then
        return false
return true
```

Initialize all vertices: $\Theta(|V|)$

Keep relaxing edges: $\Theta(|V|*|E|)$

Check negative-weight cycles: $O(|E|)$

```
Relax (u,v,G)
if v.d() > u.d()+G.w(u,v) then
   v.setd(u.d()+G.w(u,v))
   v.setparent(u)
```

One-to all: $\Theta(|V|*|E|)$

All-pairs: $O(|V|^2|E|)$

Agenda

- Recall one-to-all shortest paths
- All-pairs shortest paths
- Repeated squaring algorithm
- Floyd-Warshall algorithm
- Transitive closure of a directed graph

All-pairs shortest path - Input

- Let $n = |V|$

- Input: adjacency matrix $\mathbf{W} \in R^{n \times n}$, where
  - $w_{ij}=0$, if $i=j$.
  - $w_{ij}>0$, if $i \neq j$ and edge $(i,j) \in E$.
  - $w_{ij}=\infty$, if $i \neq j$ and edge $(i,j) \notin E$.

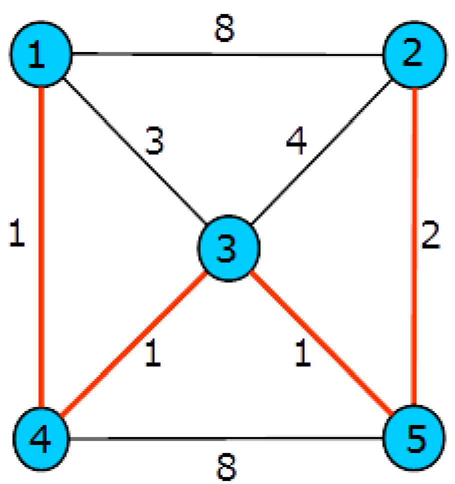
All-pairs shortest path - Output

- Distance matrix $\mathbf{D} \in \mathrm{R}^{n \times n}$, where
  - $d_{ij} = \delta(i, j)$: the shortest path weight from vertex i to vertex j.

- Predecessor matrix $\mathbf{P} \in \mathrm{R}^{n \times n}$, where
  - `i-th` row == shortest-path tree rooted at vertex i
    - One-to-all shortest paths
  - $p_{ij} = \mathrm{Nil}$, if
    - $i=j$ or
    - no path from vertex i to vertex j.
  - $p_{ij} = $ `j.parent()`
    - Vertex j's parent on the shortest path from vertex i to vertex j.

Mini quiz (also on Moodle)

- Write the adjacency matrix for this graph.
- Give the `1-st` row of the predecessor matrix (i.e., vertex 1)
- Give the `1-st` row of the distance matrix (i.e., vertex 1)



0	8	3	1	∞
8	0	4	∞	2
3	4	0	1	1
1	∞	1	0	8
∞	2	1	8	0

NIL	5	4	1	3

0	5	2	1	3

Agenda

- Recall one-to-all shortest paths
- All-pairs shortest paths
- Repeated squaring algorithm
- Floyd-Warshall algorithm
- Transitive closure of a directed graph

Base Case

- $l^{(m)}_{ij}$:
  - the minimum weight of any path from vertex `i` to vertex `j` that contains at most ***m*** edges.
- Matrices $L^{(m)} = (l^{(m)}_{ij}) \in R^{n\times n}$, $m \in [0, n-1]$, where $n = |V|$
- $m=0$

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j\ , \\ \infty & \text{if } i \neq j\ . \end{cases} \qquad L^{(0)} = \begin{pmatrix} 0 & \infty & \infty & \cdots & \infty \\ \infty & 0 & \infty & \cdots & \infty \\ \infty & \infty & 0 & \cdots & \infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \infty & \infty & \infty & \cdots & 0 \end{pmatrix}$$

- $m=1$
  - $l^{(1)}_{ij} = \begin{cases} 0 & if\ i = j \\ w_{ij} & if\ i \neq j, (i,j) \in E \\ \infty & if\ i \neq j, (i,j) \notin E \end{cases}$

$L^{(1)} = W$

Adjacency matrix

Recursive Case

- $m \geq 1$

$$l_{ij}^{(m)} = \min_{1 \leq k \leq n} \{ l_{ik}^{(m-1)} + w_{kj} \} .$$

- $m=2$

- Intuition of obtaining $l^{(m)}_{ij}$
  - $l^{(m-1)}_{ik}$: shortest path weight from i to k using at most $m-1$ edges
  - Extend the shortest path $i \leadsto k$ with one more edge (k,j)
    - $l^{(m-1)}_{ik} + w_{kj}$
    - $1 \leq k \leq n$
  - $l^{(m)}_{ij}$ = get minimum = shortest path weight from i to j.

Distance Matrix

- Matrices $L^{(m)}=(l^{(m)}{}_{ij}) \in R^{n\times n}$, $m\in[0,n-1]$, where $n=|V|$

- Final distance matrix: $L^{(n-1)}{}_{ij}$
  - Path $p=<v_i, v_{i+1}, \ldots, v_j>$
  - Simple: distinct vertices on the path.
  - At most $n-1$ edges.

- Shortest path weights

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

Intuition: Divide and Conquer

- Naïve divide and conquer?

- Overlapping sub-problems?

- Dynamic programming for computing $L^{(m)}$
  - Increasing `m` from `0` to `n-1`.
  - Bottom up.

Bottom Up Algorithm

SLOW-ALL-PAIRS-SHORTEST-PATHS(W)

$n = W.rows$
$L^{(1)} = W$
for $m = 2$ **to** $n - 1$
    let $L^{(m)}$ be a new $n \times n$ matrix
    $L^{(m)} = \boxed{\text{EXTEND-SHORTEST-PATHS}(L^{(m-1)}, W)}$ Extend one edge
return $L^{(n-1)}$

EXTEND-SHORTEST-PATHS(L, W)

$n = L.rows$
let $L' = (l'_{ij})$ be a new $n \times n$ matrix
for $i = 1$ **to** n
    for $j = 1$ **to** n
        $l'_{ij} = \infty$
        for $k = 1$ **to** n
            $l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj})$
return L'

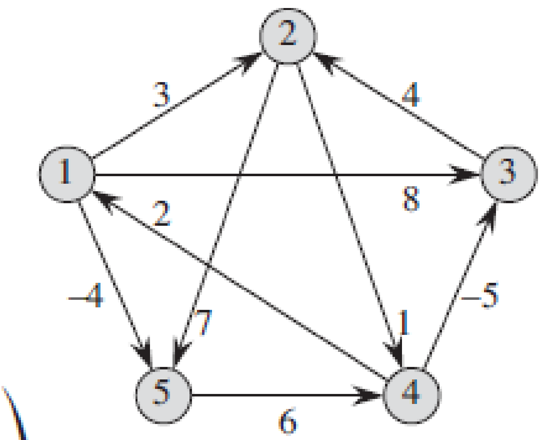
L’   L

$$l_{ij}^{(m)} = \min_{1 \le k \le n} \{l_{ik}^{(m-1)} + w_{kj}\} .$$

Example

$$L^{(0)} = \begin{pmatrix} 0 & \infty & \infty & \cdots & \infty \\ \infty & 0 & \infty & \cdots & \infty \\ \infty & \infty & 0 & \cdots & \infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \infty & \infty & \infty & \cdots & 0 \end{pmatrix}$$

$$W = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$



$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

`i=1` to `k=2` and `k=2` to `j=4`, so `3+1=4`

`i=1` to `k=5` and `k=5` to `j=4`, so `-4+6=2`

$l'_{ij} = \infty$
for $k = 1$ **to** n
  $l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj})$

$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & \boxed{2} & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Extends `n-1=5-1=4` times in total.

Run-time

EXTEND-SHORTEST-PATHS(L, W)

```
n = L.rows
let L' = (l'_ij) be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        l'_ij = ∞
        for k = 1 to n
            l'_ij = min(l'_ij, l_ik + w_kj)
return L'
```

Is this an efficient algorithm?

Three level of loops.
Each takes `n` iterations.
Thus, $\Theta(n^3)$

SLOW-ALL-PAIRS-SHORTEST-PATHS(W)

```
n = W.rows
L^(1) = W
for m = 2 to n - 1
    let L^(m) be a new n × n matrix
    L^(m) = EXTEND-SHORTEST-PATHS(L^(m-1), W)
return L^(n-1)
```

`n-1` times of $\Theta(n^3)$
Thus, $\Theta(n^4)$

Improvement – Repeated Squaring

SLOW-ALL-PAIRS-SHORTEST-PATHS(W)

```
n = W.rows
L^(1) = W
for m = 2 to n − 1
    let L^(m) be a new n × n matrix
    L^(m) = EXTEND-SHORTEST-PATHS(L^(m−1), W)
return L^(n−1)
```

Extend one more edge

$L^{(1)}$, $L^{(2)}$, $L^{(3)}$, $L^{(4)}$, $L^{(5)}$, …, $L^{(n-1)}$

if $n = 36$, $n-1 = 35$,

$\lceil \lg(n-1) \rceil = \lceil 5.2 \rceil = 6$, $2^6=64$

35 vs 6 times

FASTER-ALL-PAIRS-SHORTEST-PATHS(W)

```
n = W.rows
L^(1) = W
m = 1
while m < n − 1
    let L^(2m) be a new n × n matrix
    L^(2m) = EXTEND-SHORTEST-PATHS(L^(m), L^(m))
    m = 2m
return L^(m)
```

Extend m more edges

$L^{(1)}$, $L^{(2)}$, $L^{(4)}$, $L^{(8)}$, $L^{(16)}$, …, $L^{(x)}$,

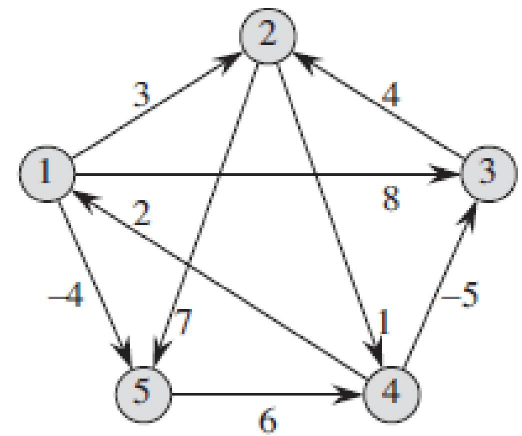
where $x = 2^{\lceil \lg(n-1) \rceil}$

$\lg n$ times

Total: $\Theta(n^3 \lg n)$

Example

$$L^{(0)} = \begin{pmatrix} 0 & \infty & \infty & \cdots & \infty \\ \infty & 0 & \infty & \cdots & \infty \\ \infty & \infty & 0 & \cdots & \infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \infty & \infty & \infty & \cdots & 0 \end{pmatrix} \quad W= \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$



$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

No need to compute $L^{(3)}$

$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix} \quad L^{(4)} = \begin{pmatrix} \boxed{0} & \boxed{1} & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Repeated Squaring vs. n×One-to-All

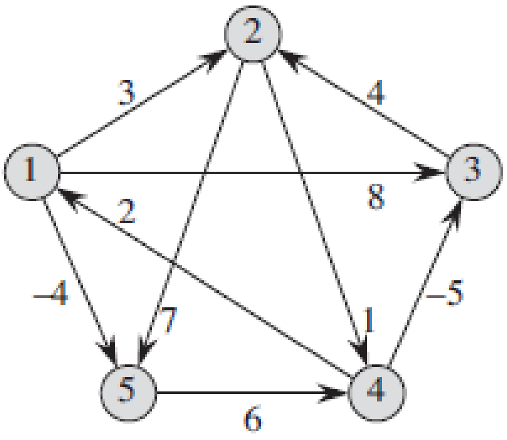
- Repeated Squaring
  - $\Theta(|V|^3 \lg |V|)$

- Dijkstra's algorithm – non-negative weights
  - $O(|E| \lg |V|)$
  - Worst case: $|E| = |V|^2$
  - n times: $O(|V||E| \lg |V|)$
  - Unless worst case $O(|V|^3 \lg |V|)$, Dijkstra's faster

- Bellman-Ford algorithm – negative weights
  - $O(|V||E|)$
  - Worst case: $|E| = |V|^2$
  - n times: $O(|V|^2 |E|)$
  - Repeated squaring faster

Mini-quiz

- How to reconstruct the shortest paths?



$$L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$l_{ij}^{(m)} = \min_{1 \le k \le n} \left\{ l_{ik}^{(m-1)} + w_{kj} \right\} .$$

Sub-problems

- Output:
  - Distance matrix $\mathbf{D} \in \mathbb{R}^{n\times n}$
  - Predecessor matrix $\mathbf{P} \in \mathbb{R}^{n\times n}$.

- A sub-problem: $L^{(m)} = (l^{(m)}{}_{ij})$
- Repeated squaring: $\lg(n)$ sub-problems
- Non repeated squaring: n sub-problems

- $L^{(m)} = (l^{(m)}{}_{ij}) \in \mathbb{R}^{n\times n}$
  - n^2 elements per matrix
  - *n* choices for computing $l^{(m)}{}_{ij}$

$$l_{ij}^{(m)} = \min_{1\le k\le n} \left\{ l_{ik}^{(m-1)} + w_{kj} \right\} .$$

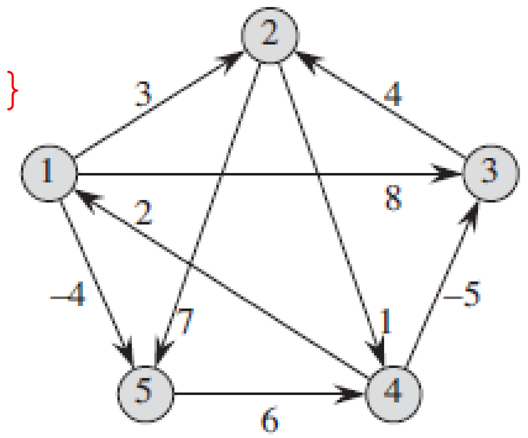
Agenda

- Recall one-to-all shortest paths
- All-pairs shortest paths
- Repeated squaring algorithm
- Floyd-Warshall algorithm
- Transitive closure of a directed graph

Concepts

- Simple path `p=<i, i+1, …, j-1, j>`
  - all vertices are distinct
- Intermediate vertices of `p`
  - `i+1, …, j-1`
  - Excl. `i`, `j`
- $d^{(k)}(i,j)$:
  - weight of shortest path `p` from vertex `i` to vertex `j`
  - intermediate vertices of `p` only allowed from vertex set `{1, …, k}`
- $d^{(k-1)}(i,j)$:
  - weight of shortest path `p` from vertex `i` to vertex `j`
  - intermediate vertices of `p` only allowed from vertex set `{1, …, k-1}`

Example of $d^{(k)}(i,j)$



- $d^{(1)}(1, 3) = 8$
  - $p = <1,3>$, allowed intermediate vertex set $\{1\}$
- $d^{(1)}(1, 4) = \infty$
  - no path, allowed intermediate vertex set $\{1\}$
- $d^{(2)}(1, 3) = 8$
  - $p=<1,3>$, allowed intermediate vertex set $\{1, 2\}$
- $d^{(2)}(1, 4) = 4$
  - $p=<1, 2, 4>$, allowed intermediate vertex set $\{1, 2\}$
- $d^{(4)}(1, 3) = -1$
  - $p=<1, 2, 4, 3>$, allowed intermediate vertex set $\{1, 2, 3, 4\}$
- $d^{(4)}(1, 4) = 4$
  - $p=<1, 2, 4>$, allowed intermediate vertex set $\{1, 2, 3, 4\}$

Floyd-Warshall Algorithm

- Definition $d^{(k)}(i,j)$:
  - weight of shortest path p from vertex i to vertex j
  - intermediate vertices of p only allowed from vertex set $\{1, \ldots, k\}$

- For any pair of vertices i and j,
- $d^{(n)}(i,j)$ == solution to all-pairs shortest paths.
  - weight of shortest path p from vertex i to vertex j
  - intermediate vertices of p only allowed from vertex set $\{1, \ldots, n\}$

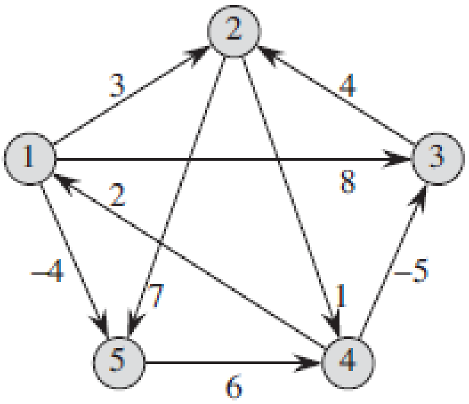
- Solving $d^{(k)}(i,j)$ as a sub-problem.

- Base case: $d^{(0)}(i,j) = w_{ij}$

Intuition-1

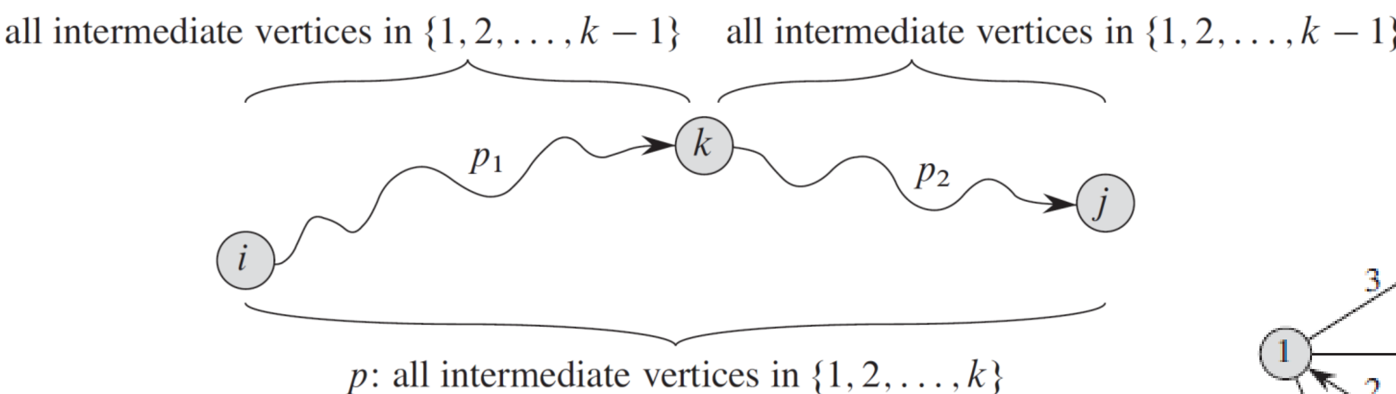
- Case 1: vertex $k \notin$ path p
  - Allowed intermediate vertex set $\{1, \ldots, k-1\}$
  - $d^{(k)}(i, j) = d^{(k-1)}(i, j)$

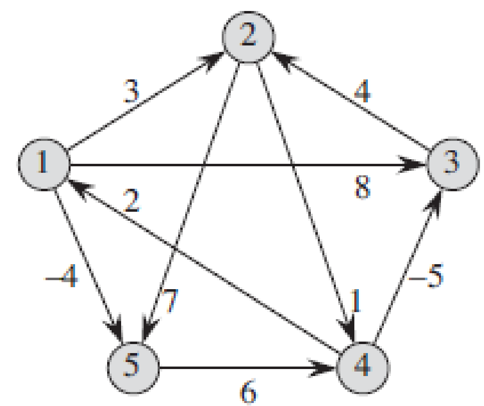
- $d^{(2)}(1, 3) = 8$, $p=\langle 1,3 \rangle$, $\{1, 2\}$
- $d^{(2)}(1, 3) = d^{(1)}(1, 3) = 8$
  - Intermediate vertex $2 \notin$ shortest path $p=\langle 1, 3 \rangle$



Intuition-2

- Case 2: intermediate vertex `k` ∈ path `p`
  - $d^{(k)}(i, j) = d^{(k-1)}(i, k) + d^{(k-1)}(k, j)$





- $d^{(4)}(1, 3) = -1$, p=<1, 2, **4**, 3>, {1, 2, 3, 4}.
- $d^{(\mathbf{3})}(1, \mathbf{4}) + d^{(\mathbf{3})}(\mathbf{4}, 3) = 4 + (-5) = -1$

Recurrence

$$d^{(k)}(i,j)=\begin{cases} w_{ij} & \text{if } k=0 \\ \min\Big(\overbrace{d^{(k-1)}(i,j)}^{\texttt{k}\text{ **is not** an intermediate vertex in }\texttt{p}},\ \underbrace{d^{(k-1)}(i,k)+d^{(k-1)}(k,j)}_{\texttt{k}\text{ **is** an intermediate vertex in }\texttt{p}}\Big) & \text{if } k\geqslant 1 \end{cases}$$

- Sub-problem: `d(k)(i, j)`
- Order of `k`
  - Increasing `k` from `0` to `n`
  - bottom up

Floyd-Warshall Algorithm

```
Floyd-Warshall(W[1..n][1..n])
D ← W      // D(0)
for k ←1 to n do // compute D(k)
    for i ←1 to n do
        for j ←1 to n do
            if D[i][k] + D[k][j] < D[i][j] then
                D[i][j] ←D[i][k] + D[k][j]
return D
```

$$d^{(k)}(i,j)=\begin{cases} w_{ij} & \text{if } k=0 \\ \min\left(\underbrace{d^{(k-1)}(i,j)}, \underbrace{d^{(k-1)}(i,k)+d^{(k-1)}(k,j)}\right) & \text{if } k\geqslant 1 \end{cases}$$

Predecessor Matrix

- Initialization:

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

- Updating:

```
Floyd-Warshall(W[1..n][1..n])
D ← W      // D^(0)
for k ← 1 to n do // compute D^(k)
    for i ←1 to n do
        for j ←1 to n do
            if D[i][k] + D[k][j] < D[i][j] then
                D[i][j] ←D[i][k] + D[k][j]
                P[i][j] ←P[k][j]
return D
```

D[i][1]

D[1][j]

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

$$\begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & \boxed{1} \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

k=1 D[4][1]+D[1][5]<D[4][5]

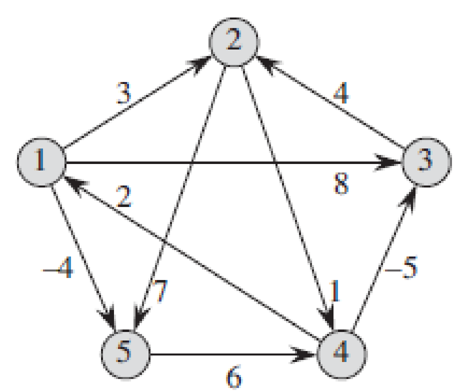
2+(-4) < ∞

P[4][5] ← P[1][5]

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

$$\begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & \boxed{1} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

```
Floyd-Warshall(W[1..n][1..n])
D ← W      // D(0)
for k ←1 to n do // compute D(k)
    for i ←1 to n do
        for j ←1 to n do
            if D[i][k] + D[k][j] < D[i][j] then
                D[i][j] ←D[i][k] + D[k][j]
                P[i][j] ←P[k][j]
return D
```



D[i][2]

D[2][j]

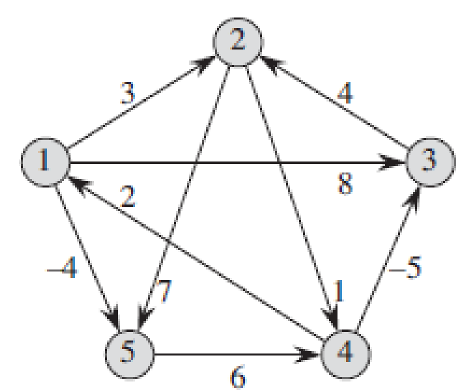
$$D^{(1)} = \begin{pmatrix} 0 & \boxed{3} & 8 & \boxed{\infty} & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL}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

k=2 D[1][2]+D[2][4]<D[1][4] P[1][4] ← P[2][4]

3+1 < ∞

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

```
Floyd-Warshall(W[1..n][1..n])
D ← W      // D(0)
for k ←1 to n do // compute D(k)
    for i ←1 to n do
        for j ←1 to n do
            if D[i][k] + D[k][j] < D[i][j] then
                D[i][j] ← D[i][k] + D[k][j]
                P[i][j] ← P[k][j]
return D
```



Run-time

```
Floyd-Warshall(W[1..n][1..n])
D ← W      // D^(0)
for k ← 1 to n do // compute D^(k)
    for i ←1 to n do
        for j ←1 to n do
            if D[i][k] + D[k][j] < D[i][j] then
                D[i][j] ←D[i][k] + D[k][j]
                P[i][j] ←P[k][j]
return D
```

Three level of loops.
Each takes n iterations.
Thus, $\Theta(n^3)$

Sub-problems

- Floyd Warshall
  - A sub-problem: $D^{(k)}$, k = 1, …, n
  - 3 choices for $d^{(k)}(i, j)$.

$$d^{(k)}(i,j) = \min\left(d^{(k-1)}(i,j), d^{(k-1)}(i,k) + d^{(k-1)}(k,j)\right)$$

Sub-problem	# sub-problems	# cells	Choices per cell	Total
Non repeated squaring $L^{(m)}$	n	$n \times n$	n	n^4
Repeated squaring $L^{(m)}$	$lg\, n$	$n \times n$	n	$\lg n \times n^3$
Floyd warshall $D^{(k)}$	n	$n \times n$	3	n^3

- Which sub-problems are overlapping?
  - See Moodle.

Run Time Summary

- Non-negative weights a graph
  - Dijkstra’s algorithm: $O(|V|*|E|\lg|V|)$
  - Worst case ($|E|=|V|^2$): $O(|V|^3\lg|V|)$

- Negative weights graph
  - Bellman-Ford: $O(|V|^2|E|)$
  - Worst case ($|E|=|V|^2$): $O(|V|^4)$

- Repeated squaring: $\Theta(|V|^3\lg|V|)$
- Floyd-Warshall: $\boldsymbol{\Theta(|V|^3)}$

Agenda

- Recall one-to-all shortest paths
- All-pairs shortest paths
- Repeated squaring algorithm
- Floyd-Warshall algorithm
- Transitive closure of a directed graph

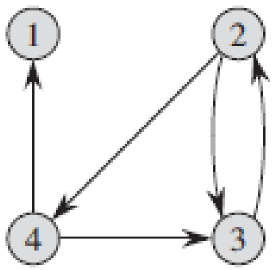
Transitive Closure of Directed Graph

- Purpose
  - Find out whether there is a path for two vertices `i` and `j`.
  - Indicate reachability of two vertices `i` and `j`.

- Examples
  - Whether I can go from `i` to `j`
  - Whether `i` is a friend of `j`.

- Transitive closure of direct graph `G=(V, E)`: `G*=(V, E*)`
  - `E*={(i,j)}`
  - Satisfying: there is a path from vertex `i` to vertex `j` in `G`

```
E*={(1,1),
    (2,1), (2,2), (2,3), (2,4),
    (3,1), (3,2), (3,3), (3,4),
    (4,1), (4,2), (4,3), (4,4)}
```



Transitive Closure By Floyd-Warshall

- Assign edge weight to 1 for each edge of E.
- Run the Floyd-Warshall algorithm.

- $d^{(n)}(i, j) < n$
  - there is a path from vertex i to vertex j
  - $(i, j) \in E^*$

- $d^{(n)}(i, j) = \infty$
  - no path from vertex i to vertex j
  - $(i, j) \notin E^*$

An alternative algorithm

- The same asymptotic run time, but can save time and space in practice.

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i, j) \notin E , \\ 1 & \text{if } i = j \text{ or } (i, j) \in E , \end{cases}$$

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right)$$

- Floyds-Warshell

$$d^{(k)}(i, j) = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min\left(d^{(k-1)}(i, j), d^{(k-1)}(i, k) + d^{(k-1)}(k, j)\right) & \text{if } k \geqslant 1 \end{cases}$$

ILO of Lecture 2

- All-pairs shortest paths using dynamic programming
  - To understand the adjacency matrix and the predecessor matrix, which are the representations of the input and output of most of the all-pairs shortest-path algorithms.
  - To understand how the dynamic programming principles play out in the repeated squaring and Floyd-Warshall algorithm.
  - Understand the definition of transitive closure of a directed graph.

Lecture 3

- Flow network
  - to understand the formalisms of flow networks and flows;
  - to understand the Ford-Fulkerson method and why it works;
  - to understand the Edmonds-Karp algorithm and to be able to analyze its worst-case running time;
  - to be able to apply the Ford and Fulkerson method to solve the maximum-bipartite-matching problem.