Today:

- All Pairs Shortest Paths

COSC 581, Algorithms

February 6, 2014

Many of these slides are adapted from several online sources

Reading Assignments

- Today's class:
  - Chapter 25.1-25.2

- Reading assignment for next class:
  - Chapter 16.1-16.2

- Announcement: Exam 1 is on Tues, Feb. 18
  - Will cover everything up through dynamic programming

All Pairs Shortest Paths (APSP)

- given : directed graph $G = (V, E)$,
  weight function ω : E → R, $|V| = n$

- goal : create an $n \times n$ matrix $L = (l_{ij})$ of shortest path distances
  i.e., $l_{ij} = \delta (i, j)$

- trivial solution : run a SSSP algorithm n times, one for
  each vertex as the source.

All Pairs Shortest Paths (APSP)

- ► all edge weights are nonnegative : use Dijkstra's algorithm
  - – Priority Queue = linear array : O (V^3 + VE) = O (V^3)
  - – Priority Queue = binary heap : O ($V^2 \lg V$ + $EV \lg V$) = O ($V^3 \lg V$) for dense graphs
    - **better only for sparse graphs**
  - – Priority Queue = Fibonacci heap : O ($V^2 \lg V$ + EV) = O (V^3) for dense graphs
    - **better only for sparse graphs**
- ► negative edge weights : use Bellman-Ford algorithm
  - – O (V^2E) = O (V^4) on dense graphs

Shortest Paths and Matrix Multiplication

Assumption : negative edge weights may be present, but no negative weight cycles.

(Step 1) Structure of a Shortest Path (new Optimal Substructure argument):

- Consider a shortest path p_{ij}^{m} from v_i to v_j such that $|p_{ij}^{m}| \le m$
  - ► i.e., path p_{ij}^{m} has at most m edges.

- no negative-weight cycle $\Rightarrow$ all shortest paths are simple

  $\Rightarrow$ m is finite $\Rightarrow m \le |V| - 1$

- $i = j \Rightarrow |p_{ii}| = 0$ & $\omega(p_{ii}) = 0$

- $i \ne j \Rightarrow$ decompose path p_{ij}^{m} into p_{ik}^{m-1} & $v_k \rightarrow v_j$, where $|p_{ik}^{m-1}| \le m - 1$
  - ► p_{ik}^{m-1} should be a shortest path from v_i to v_k by optimal substructure property.
  - ► Therefore, $\delta(i, j) = \delta(i, k) + \omega_{k\,j}$

Shortest Paths and Matrix Multiplication

(Step 2): A Recursive Solution to All Pairs Shortest Paths Problem :

- l_{ij}^{m} = minimum weight of any path from v_i to v_j that contains at most "*m*" edges.

- *m = 0* : There exists a shortest path from v_i to v_j with no edges ↔ i = j .

$$▶ \; l_{ij}^{0} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$$

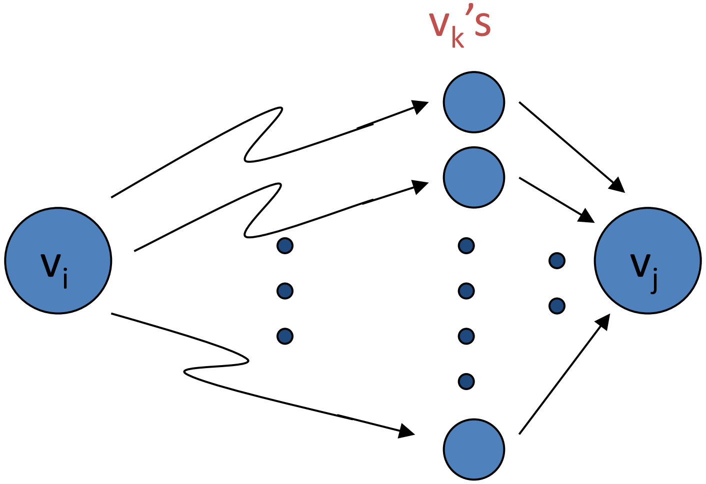
- *m ≥ 1* : $l_{ij}^{m} = \min \{l_{ij}^{m-1}, \min_{1 \leq k \leq n \wedge k \neq j} \{l_{ik}^{m-1} + \omega_{kj}\}\}$

$$= \min_{1 \leq k \leq n} \{l_{ik}^{m-1} + \omega_{kj}\} \text{ for all } v_k \in V,$$

since $\omega_{j\,j} = 0$ for all $v_j \in V$.

Shortest Paths and Matrix Multiplication

- To consider all possible shortest paths with $\leq m$ edges from v_i to v_j
  - ► consider shortest path with $\leq m - 1$ edges, from v_i to v_k, where $(v_k, v_j) \in E$



Shortest Paths and Matrix Multiplication

(Step 3) Computing the shortest-path weights bottom-up :

- Given $W = L^1$, compute a series of matrices $L^2, L^3, \ldots, L^{n-1}$, where $L^m = (l_{ij}^{m})$ for $m = 1, 2, \ldots, |V| -1$

  ▶ final matrix L^{n-1} contains actual shortest path weights, i.e., $l_{ij}^{n-1} = \delta(i, j)$

- SLOW-APSP(W)

```
SLOW-APSP( W )
    $L^1 \leftarrow W$
    for $m \leftarrow 2$ to $n-1$ do
        $L^m \leftarrow$ EXTEND( $L^{m-1}$, W )
    return $L^{n-1}$
```

Shortest Paths and Matrix Multiplication

```
EXTEND ( L , W )
▶ L = ( $l_{ij}$ ) is an n x n matrix
    for $i \leftarrow 1$ to $n$ do
        for $j \leftarrow 1$ to $n$ do
            $l_{ij} \leftarrow \infty$
            for $k \leftarrow 1$ to $n$ do
                $l_{ij} \leftarrow \min\{l_{ij}, l_{ik} + \omega_{kj}\}$
    return L
```

```
MATRIX-MULT ( A , B )
▶ C = ( $c_{ij}$ ) is an n x n result matrix
    for $i \leftarrow 1$ to $n$ do
        for $j \leftarrow 1$ to $n$ do
            $c_{ij} \leftarrow 0$
            for $k \leftarrow 1$ to $n$ do
                $c_{ij} \leftarrow c_{ij} + a_{ik} \times b_{kj}$
    return C
```

Shortest Paths and Matrix Multiplication

- Relation to matrix multiplication C = A × B : $\mathbf{c}_{ij} = \sum_{1 \le k \le n} \mathbf{a}_{ik} \times \mathbf{b}_{kj}$,

  ► $L^{m-1} \leftrightarrow A$ & $\mathbf{W} \leftrightarrow B$ & $L^{m} \leftrightarrow C$

  "min" ↔ "+" & "+" ↔ "x" & "∞" ↔ "0"

- Thus, we compute the sequence of matrix products

  $L^1 = L^0 \times W = W$; note L^0 = identity matrix,

  $L^2 = L^1 \times W = W^2$

  $L^3 = L^2 \times W = W^3$

  ⋮

  $L^{n-1} = L^{n-2} \times W = W^{n-1}$

  $$\text{i.e., } l_{ij}^{0} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$$

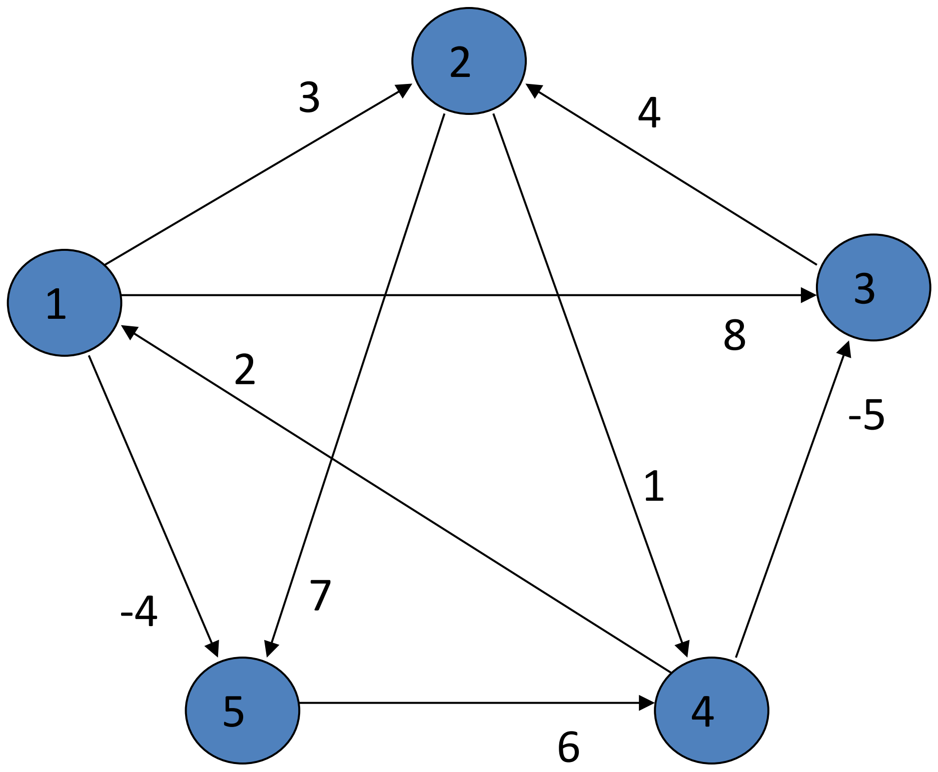
- Running time : $\Theta(V^4)$

  ► each matrix product : $\Theta(|V|^3)$

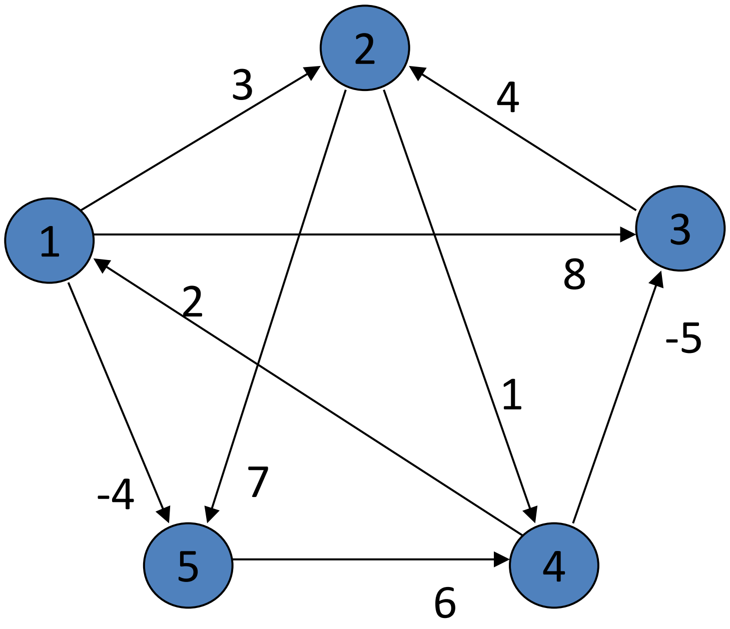
  ► number of matrix products : $|V| - 1$

Shortest Paths and Matrix Multiplication

Example:



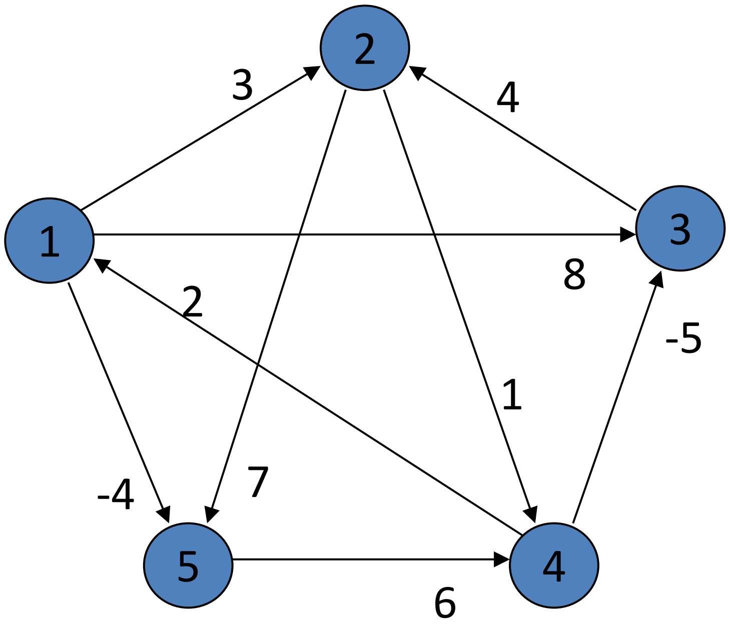
Shortest Paths and Matrix Multiplication



	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

$L^1 = L^0W$

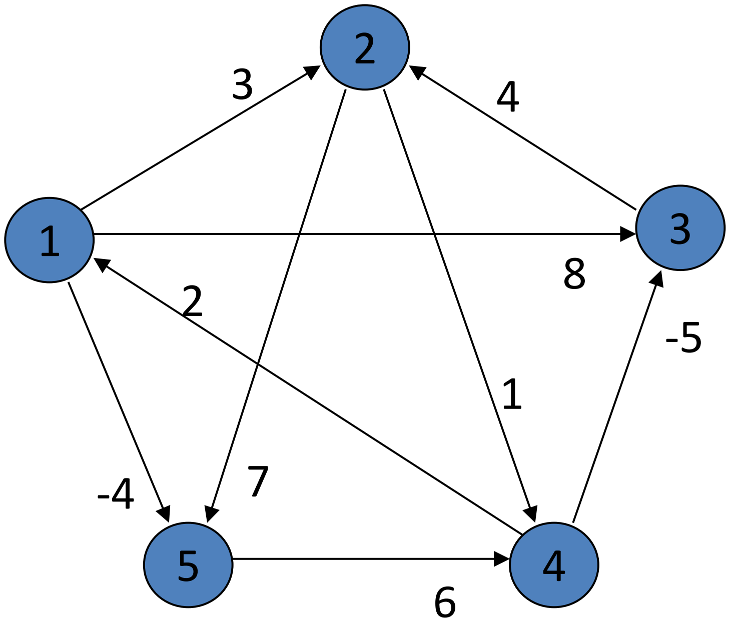
Shortest Paths and Matrix Multiplication



	1	2	3	4	5
1	0	3	8	2	-4
2	3	0	-4	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	8	∞	1	6	0

$L^2 = L^1 W$

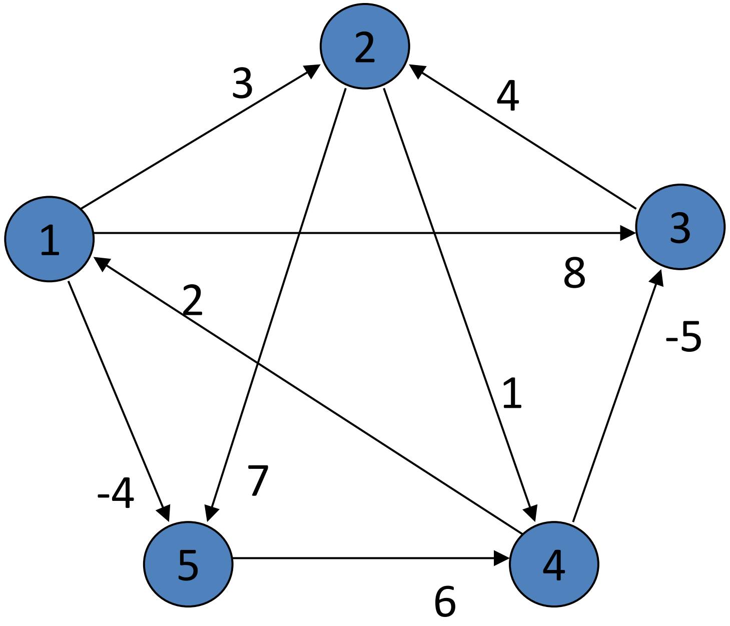
Shortest Paths and Matrix Multiplication



	1	2	3	4	5
1	0	3	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	11
4	2	-1	-5	0	-2
5	8	5	1	6	0

$L^3 = L^2 W$

Shortest Paths and Matrix Multiplication



	1	2	3	4	5
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

$L^4 = L^3 W$

Improving Running Time Through Repeated Squaring

- Idea : goal is not to compute all L^m matrices
  - ▶ we are interested only in matrix L^{n-1}
- Recall : no negative-weight cycles ⇒ $L^m = L^{n-1}$ for all $m \geq |V| -1$
- We can compute L^{n-1} with only $\lceil \lg(n-1) \rceil$ matrix products as

$$L^1 = W$$
$$L^2 = W^2 = W \times W$$
$$L^4 = W^4 = W^2 \times W^2$$
$$L^8 = W^8 = W^4 \times W^4$$
$$\vdots$$
$$L^{2^{\lceil \lg(n-1) \rceil}} = L^{2^{\lceil \lg(n-1) \rceil}} = L^{2^{\lceil \lg(n-1) \rceil -1}} \times L^{2^{\lceil \lg(n-1) \rceil -1}}$$

- This technique is called repeated squaring.

Improving Running Time Through Repeated Squaring

- FASTER-APSP (W)

```
L^1 ← W
m ← 1
while m < n-1 do
    L^2m ← EXTEND ( L^m , L^m )
    m ← 2m
return L^m
```

- Final iteration computes L^{2m} for some $n-1 \leq 2m \leq 2n-2 \Rightarrow L^{2m} = L^{n-1}$
- Running time : $\Theta(n^3 \lg n) = \Theta(V^3 \lg V)$
  - ► each matrix product : $\Theta(n^3)$
  - ► # of matrix products : $\lceil \lg(n-1) \rceil$
  - ► simple code, no complex data structures, small hidden constants in Θ-notation.

Exercise

Give an efficient algorithm to find the length (number of edges) of a minimum-length negative-weight cycle in a graph.

Floyd-Warshall Algorithm

Assumption : negative-weight edges, but no negative-weight cycles

(Step 1) The Structure of a Shortest Path (yet another optimal substructure argument):

- Definition : intermediate vertex of a path $p = < v_1 , v_2 , v_3 , \dots , v_k >$

  ► any vertex of p other than v_1 or v_k .

- p_{ij}^{m} : a shortest path from v_i to v_j with all intermediate vertices from $V_m = \{ v_1 , v_2 , \dots , v_m \}$

- Relationship between p_{ij}^{m} and p_{ij}^{m-1}

  ► depends on whether v_m is an intermediate vertex of p_{ij}^{m}

- Case 1: v_m is not an intermediate vertex of p_{ij}^{m}

$\Rightarrow$ all intermediate vertices of p_{ij}^{m} are in V_{m-1}

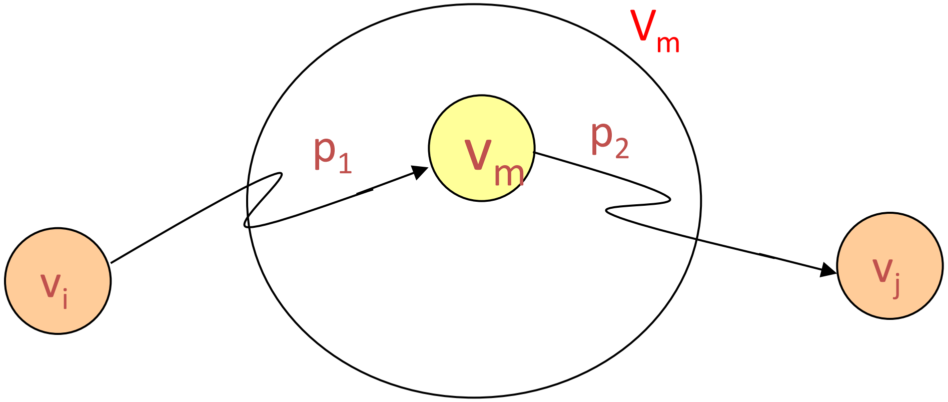
$\Rightarrow p_{ij}^{m} = p_{ij}^{m-1}$

Floyd-Warshall Algorithm

- Case 2 : v_m is an intermediate vertex of p_{ij}^{m}
  - decompose path as $v_i \rightsquigarrow v_m \rightsquigarrow v_j$

    $\Rightarrow p_1 : v_i \rightsquigarrow v_m$ & $p_2 : v_m \rightsquigarrow v_j$
  - by opt. structure property both p_1 & p_2 are shortest paths.
  - v_m is not an intermediate vertex of p_1 & p_2

    $\Rightarrow p_1 = p_{im}^{m-1}$ & $p_2 = p_{mj}^{m-1}$



Floyd-Warshall Algorithm

(Step 2) A Recursive Solution to APSP Problem :

- $d_{ij}^{m} = \omega(p_{ij})$: weight of a shortest path from v_i to v_j with all intermediate vertices from

  $V_m = \{ v_1 , v_2 , \ldots , v_m \}$.

- Note : $d_{ij}^{n} = \delta(i, j)$ since $V_n = V$

  ► i.e., all vertices are considered for being intermediate vertices of p_{ij}^{n} .

Floyd-Warshall Algorithm

- Compute d_{ij}^{m} in terms of d_{ij}^{k} with smaller $k < m$
- $m = 0$: V_0 = empty set

  $\Rightarrow$ path from v_i to v_j with no intermediate vertex.

  i.e., v_i to v_j paths with at most one edge

  $\Rightarrow d_{ij}^{0} = \omega_{i\,j}$

- $m \geq 1$: $d_{ij}^{m} = \min \{d_{ij}^{m-1}, d_{im}^{m-1} + d_{mj}^{m-1}\}$

Floyd-Warshall Algorithm

(Step 3) Computing Shortest Path Weights Bottom Up :

```
FLOYD-WARSHALL( W )
    ► $D^0$, $D^1$, … , $D^n$ are $n$ x $n$ matrices
    for $m \leftarrow 1$ to $n$ do
        for $i \leftarrow 1$ to $n$ do
            for $j \leftarrow 1$ to $n$ do
                $d_{ij}^{m} \leftarrow \min \{d_{ij}^{m-1}, d_{im}^{m-1} + d_{mj}^{m-1}\}$
    return $D^n$
```

Floyd-Warshall Algorithm

```
FLOYD-WARSHALL ( W )
        ▶ D  is an $n$ x $n$ matrix
        D ← W
        for $m$ ← $1$ to $n$ do
            for $i$ ← $1$ to $n$ do
                for $j$ ← $1$ to $n$ do
                    if  $d_{ij} > d_{im} + d_{mj}$  then
                        $d_{ij} \leftarrow d_{im} + d_{mj}$
        return D
```

Floyd-Warshall Algorithm

- Maintaining *n* D matrices can be avoided by dropping all superscripts.
  - *m-th* iteration of outermost for-loop

    begins with $D = D^{m-1}$

    ends with $D = D^{m}$
  - computation of d_{ij}^{m} depends on d_{im}^{m-1} and d_{mj}^{m-1} .

    no problem if d_{im} & d_{mj} are already updated to d_{im}^{m} & d_{mj}^{m} since $d_{im}^{m} = d_{im}^{m-1}$ & $d_{mj}^{m} = d_{mj}^{m-1}$.

- Running time : $\Theta(n^3) = \Theta(V^3)$

  simple code, no complex data structures, small hidden constants

Reading Assignments

- Reading assignment for next class:
  - Chapter 16.1-16.2

- Announcement: Exam 1 is on Tues, Feb. 18
  - Will cover everything up through dynamic programming