CS473 – Algorithms I

All Pairs Shortest Paths

All Pairs Shortest Paths (APSP)

- given : directed graph $G = (V, E)$,
  weight function $\omega : E \rightarrow R$, $|V| = n$
- goal : create an $n \times n$ matrix $D = (d_{ij})$ of shortest path distances
  i.e., $d_{ij} = \delta (v_i , v_j)$
- 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
  - PQ = binary heap : O (V . (V+ E)lgV)

    $= O(V^3 \lg V)$ for dense graphs

    $= O(V^2 \lg V)$ for sparse graphs

- negative edge weights : use Bellman-Ford algorithm
  - $O(V.\ VE) = O(V^4)$ on dense graphs and

    $= O(V^3)$ on sparse graphs

Adjacency Matrix Representation of Graphs

- *n* x *n* matrix W = (ω_{ij}) of edge weights :

$$\omega_{ij} = \begin{cases} \omega(v_i, v_j) & \text{if } (v_i, v_j) \in E \\ \infty & \text{if } (v_i, v_j) \notin E \end{cases}$$

- assume $\omega_{ii} = 0$ for all $v_i \in V$, because
  - no neg-weight cycle

    $\Rightarrow$ shortest path to itself has no edge,

    i.e., $\delta(v_i, v_i) = 0$

Dynamic Programming

(1) Characterize the structure of an optimal solution.

(2) Recursively define the value of an optimal solution.

(3) Compute the value of an optimal solution in a bottom-up manner.

(4) Construct an optimal solution from information constructed in (3).

Shortest Paths and Matrix Multiplication

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

(1) Structure of a Shortest Path :

- 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 $\Rightarrow m \le n - 1$
- $i = j \Rightarrow |p_{ii}| = 0$ & $\omega(p_{ii}) = 0$
- $i \ne j \Rightarrow$ 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, $\delta(v_i, v_j) = \delta(v_i, v_k) + \omega_{kj}$

Shortest Paths and Matrix Multiplication

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

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

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

$$\blacktriangleright\ d_{ij}^{0} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$$

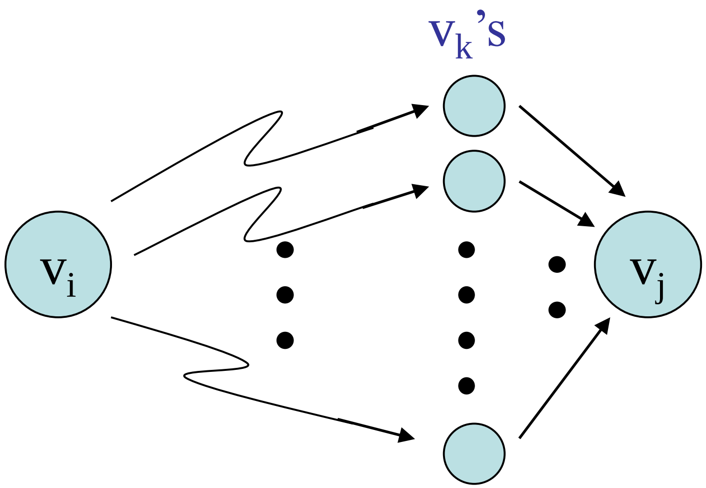
- $m \geq 1$: $d_{ij}^{m} = \min \{ d_{ij}^{m-1} , \min_{1 \leq k \leq n \wedge k \neq j} \{ d_{ik}^{m-1} + \omega_{kj} \}\}$

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

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 $v_k \in R_{v_i}$ and $(v_k, v_j) \in E$



- note : $\delta(v_i, v_j) = d_{ij}^{n-1} = d_{ij}^{n} = d_{ij}^{n+1}$, since $m \leq n - 1 = |V| - 1$

Shortest Paths and Matrix Multiplication

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

- given $W = D^1$, compute a series of matrices D^2, D^3, ..., D^{n-1} , where $D^m = (d_{ij}^{m})$ for $m = 1, 2,..., n-1$
  - ► final matrix D^{n-1} contains actual shortest path weights, i.e., $d_{ij}^{n-1} = \delta (v_i ,v_j)$

- SLOW-APSP(W)

  $D^1 \leftarrow W$

  for $m \leftarrow 2$ to $n-1$ do

  $\quad D^m \leftarrow$ EXTEND(D^{m-1} , W)

  return D^{n-1}

Shortest Paths and Matrix Multiplication

EXTEND (D , W)

▶ D = (d_{ij}) is an n x n matrix

for $i \leftarrow 1$ **to** n **do**

  for $j \leftarrow 1$ **to** n **do**

    $d_{ij} \leftarrow \infty$

    for $k \leftarrow 1$ **to** n **do**

      $d_{ij} \leftarrow \mathbf{min}\{d_{ij}, d_{ik} + \omega_{kj}\}$

return D

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 : $\mathbf{c}_{ij} = \sum_{1 \le k \le n} \mathbf{a}_{ik} \text{ x } \mathbf{b}_{kj}$,

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

  “min” ↔ “t” & “t” ↔ “x” & “∞” ↔ “0”

- Thus, we compute the sequence of matrix products

$$D^1 = D^0 \text{ x } W = W \text{ ; note } D^0 = \text{identity matrix,}$$

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

$$D^2 = D^1 \text{ x } W = W^2$$

$$D^3 = D^2 \text{ x } W = W^3$$

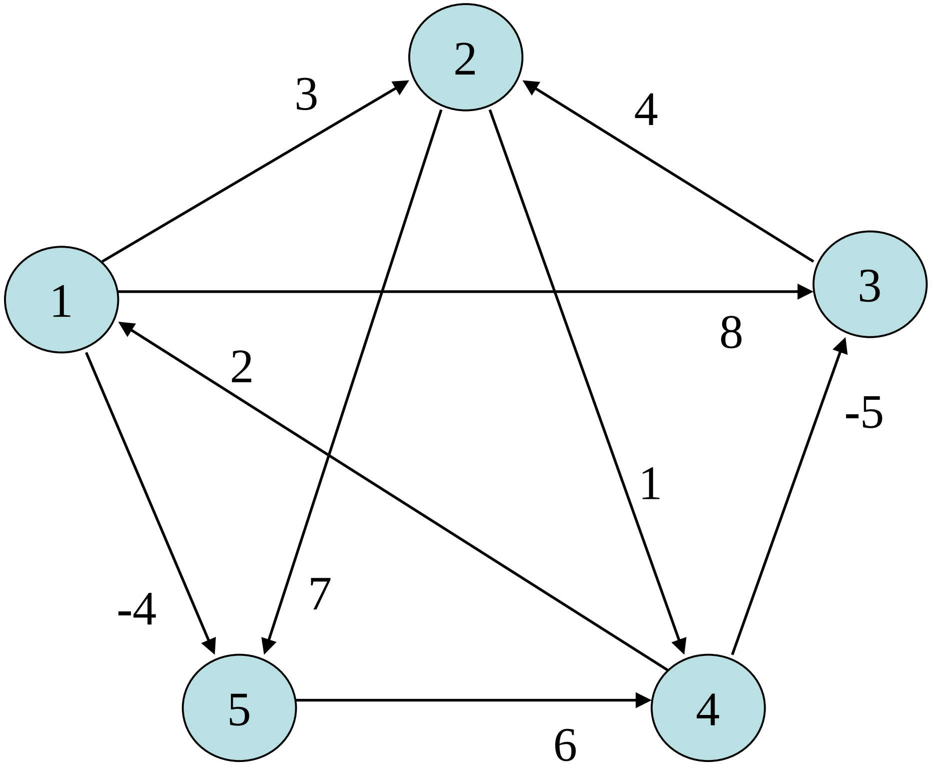
$$\vdots$$

$$D^{n-1} = D^{n-2} \text{ x } W = W^{n-1}$$

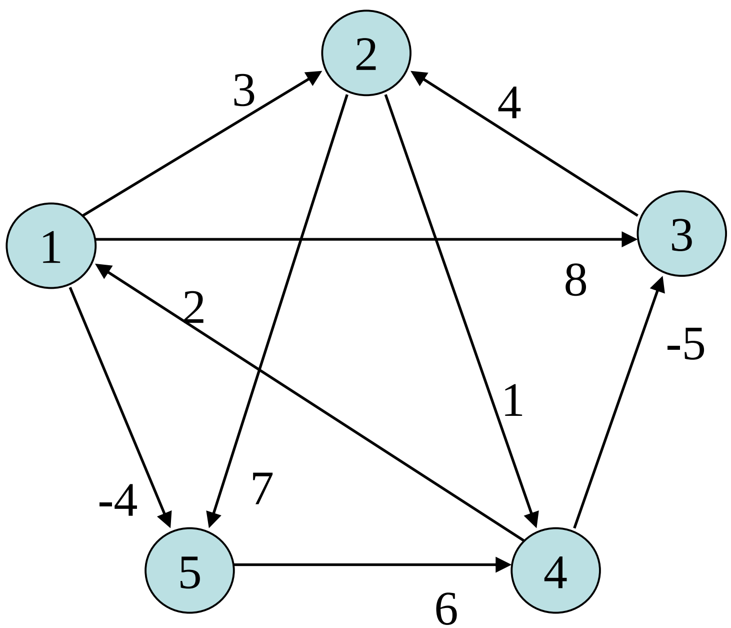
- running time : $\Theta(n^4) = \Theta(V^4)$
  - ► each matrix product : $\Theta(n^3)$
  - ► number of matrix products : n-1

Shortest Paths and Matrix Multiplication

- Example



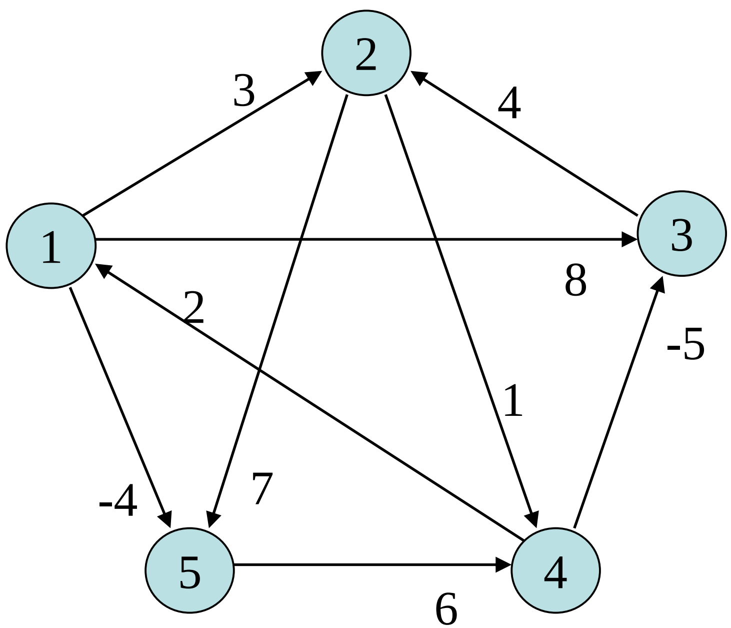
Shortest Paths and Matrix Multiplication



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

$$D^1 = D^0 W$$

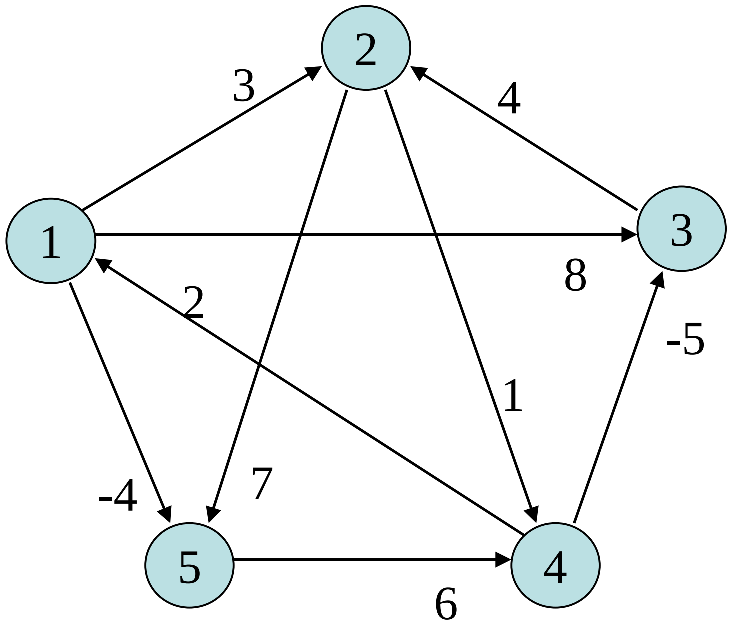
Shortest Paths and Matrix Multiplication



	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

$D^2 = D^1 W$

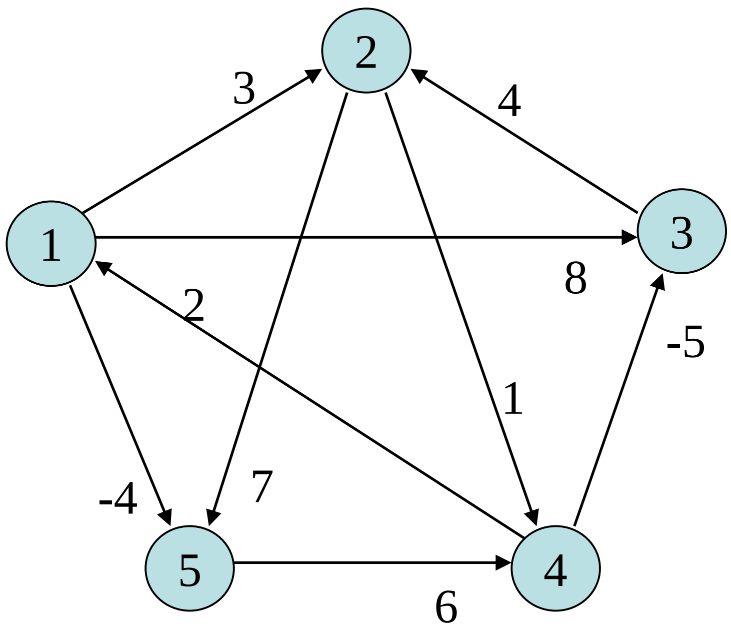
Shortest Paths and Matrix Multiplication



	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

$D^3 = D^2 W$

Shortest Paths and Matrix Multiplication

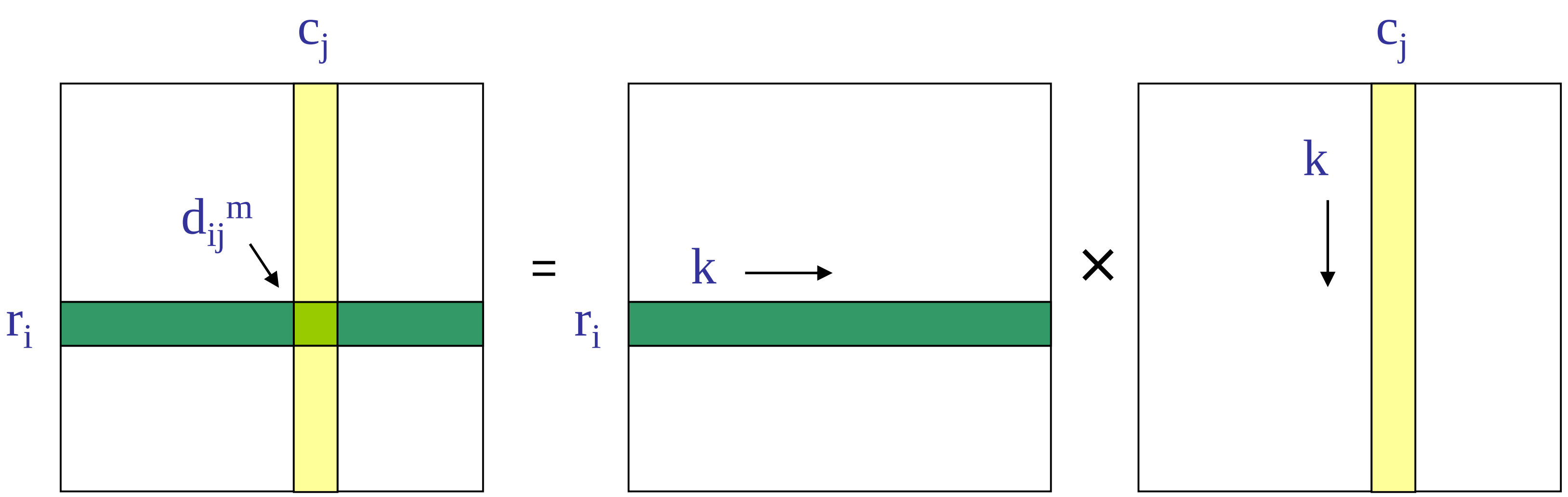


	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

$D^4 = D^3 W$

SSSP and Matrix-Vector Multiplication

- relation of APSP to one step of matrix multiplication



$D^m \leftrightarrow C$ $\qquad$ $D^{m-1} \leftrightarrow A$ $\qquad$ $W \leftrightarrow B$

SSSP and Matrix-Vector Multiplication

- d_{ij}^{n-1} at row r_i and column c_j of product matrix $= \delta(v_i{=}s, v_j)$ for $j = 1, 2, 3, \ldots, n$

- row r_i of the product matrix = solution to single-source shortest path problem for $s = v_i$.

► r_i of C = matrix B multiplied by r_i of A

$$\Rightarrow D_i^{m} = D_i^{m-1} \, x \, W$$

SSSP and Matrix-Vector Multiplication

- let $D_i^0 = d^0$, where $d_j^0 = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{otherwise} \end{cases}$
- we compute a sequence of n-1 "matrix-vector" products

$$d_i^1 = d_i^0 \text{ x } W$$

$$d_i^2 = d_i^1 \text{ x } W$$

$$d_i^3 = d_i^2 \text{ x } W$$

$$\vdots$$

$$d_i^{n-1} = d_i^{n-2} \text{ x } W$$

SSSP and Matrix-Vector Multiplication

- this sequence of matrix-vector products
  - ► same as Bellman-Ford algorithm.
  - ► vector $d_i^{m} \Rightarrow$ d values of Bellman-Ford algorithm after m-th relaxation pass.
  - ► $d_i^{m} \leftarrow d_i^{m-1} x\ W$

    $\Rightarrow$ *m-th* relaxation pass over all edges.

SSSP and Matrix-Vector Multiplication

BELLMAN-FORD (G , v_i)

▶ perform RELAX (u , v) for

▶ every edge (u , v) $\in$ E

for $j \leftarrow 1$ to n do

for $k \leftarrow 1$ to n do

RELAX (v_k , v_j)

RELAX (u , v)

$d_v = \min \{ d_v , d_u + \omega_{uv} \}$

EXTEND (d_i , W)

▶ d_i is an n-vector

for $j \leftarrow 1$ to n do

$d_j \leftarrow \infty$

for $k \leftarrow 1$ to n do

$d_j \leftarrow \min \{ d_j , d_k + \omega_{kj} \}$

Improving Running Time Through Repeated Squaring

- idea : goal is not to compute all D^m matrices
  - ► we are interested only in matrix D^{n-1}
- recall : no negative-weight cycles $\Rightarrow D^m = D^{n-1}$ for all $m \geq n-1$
- we can compute D^{n-1} with only $\lceil \lg(n-1) \rceil$ matrix products as

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

- This technique is called repeated squaring.

Improving Running Time Through Repeated Squaring

- FASTER-APSP (W)

  $D^1 \leftarrow W$

  $m \leftarrow 1$

  while $m < n\text{-}1$ do

  $\quad D^{2m} \leftarrow$ EXTEND (D^m , D^m)

  $\quad m \leftarrow 2m$

  return D^m

- final iteration computes D^{2m} for some $n\text{-}1 \leq 2m \leq 2n\text{-}2 \Rightarrow D^{2m} = D^{n-1}$

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

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

  ► # of matrix products : $\lceil \lg(n\text{-}1) \rceil$

  ► simple code, no complex data structures, small hidden constants in Θ-notation.

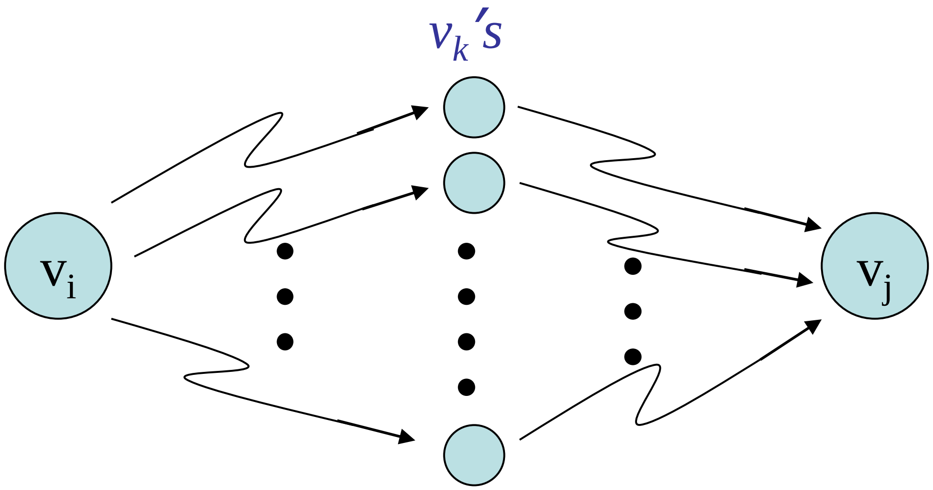
Idea Behind Repeated Squaring

- decompose p_{ij}^{2m} as p_{ik}^{m} & p_{kj}^{m}, where

p_{ij}^{2m} : $v_i \rightsquigarrow v_j$

p_{ik}^{m} : $v_i \rightsquigarrow v_k$

p_{kj}^{m} : $v_k \rightsquigarrow v_j$



Floyd-Warshall Algorithm

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

(1) The Structure of a Shortest Path :

- Definition : intermediate vertex of a path $p = < v_1 , v_2 , v_3 , \ldots , v_k >$
  - ► 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 $V_m = \{ v_1 , v_2 , \ldots , v_m \}$
- 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}

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

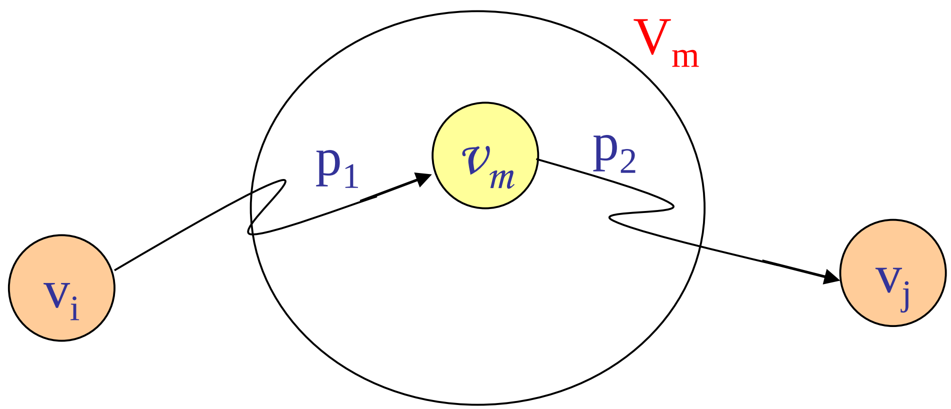
Floyd-Warshall Algorithm

- 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

(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 : $d_{ij}^{n} = \delta(v_i, v_j)$ 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

(3) Computing Shortest Path Weights Bottom Up :

FLOYD-WARSHALL(W)

▶ $D^0, D^1, \ldots, D^n$ 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 ← 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

Transitive Closure of a Directed Graph

- $G' = (V, E')$: transitive closure of $G = (V, E)$, where
  ▶ $E' = \{ (v_i, v_j)$: there exists a path from v_i to v_j in $G \}$
- trivial solution : assign W such that

$$\omega_{ij} = \begin{cases} 1 \text{ if } (v_i, v_j) \in E \\ \infty \quad \text{otherwise} \end{cases}$$

  ▶ run Floyd-Warshall algorithm on W
  ▶ $d_{ij}^{n} < n \Rightarrow$ there exists a path from v_i to v_j , i.e., $(v_i, v_j) \in E'$
  ▶ $d_{ij}^{n} = \infty \Rightarrow$ no path from v_i to v_i , i.e., $(v_i, v_j) \notin E'$
  ▶ running time : $\Theta(n^3) = \Theta(V^3)$

Transitive Closure of a Directed Graph

- Better $\Theta(V^3)$ algorithm : saves time and space.

  ► W = adjacency matrix : $\omega_{ij} = \begin{cases} 1 & \text{if } i = j \text{ or } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$

  ► run Floyd-Warshall algorithm by replacing "min" → "$\vee$" & "+" → "$\wedge$"

- define $t_{ij}^{m} = \begin{cases} 1 & \text{if } \exists \text{ a path from } v_i \text{ to } v_j \text{ with all intermediate vertices from } V_m \\ 0 & \text{otherwise} \end{cases}$

  ► $t_{ij}^{n} = 1 \Rightarrow (v_i, v_j) \in E'$ & $t_{ij}^{n} = 0 \Rightarrow (v_i, v_j) \notin E'$

- recursive definition for $t_{ij}^{m} = t_{ij}^{m-1} \vee (t_{im}^{m-1} \wedge t_{mj}^{m-1})$ with $t_{ij}^{0} = \omega_{ij}$

Transitive Closure of a Directed Graph

```
T-CLOSURE (G )
    ▶ T = ( t_ij ) is an n x n boolean matrix
      for i ← 1 to n do
         for j ← 1 to n do
            if  i = j  or ( v_i , v_j ) ∈ E  then
               t_ij ← 1
            else
               t_ij ← 0
      for  m ← 1 to n do
         for i ← 1 to n do
            for j ← 1 to n do
               t_ij ← t_ij ∨ ( t_im ∧ t_mj )
```