Design and Analysis of Algorithms

CSE 5311

Lecture 22 All-Pairs Shortest Paths

Junzhou Huang, Ph.D.

Department of Computer Science and Engineering

All Pairs Shortest Paths (APSP)

- given : directed graph $G = (V, E)$,
  weight function $\omega : E \to R$, $|V| = n$

- 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 = linear array : $O(V^3 + VE) = O(V^3)$
  - PQ = binary heap : $O(V^2 \lg V + EV \lg V) = O(V^3 \lg V)$ for dense graphs
    - **better only for sparse graphs**
  - PQ = 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
  - $O(V^2 E) = O(V^4)$ on dense 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} & $v_k \to v_j$, 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 $\leftrightarrow$ $i = j$.

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

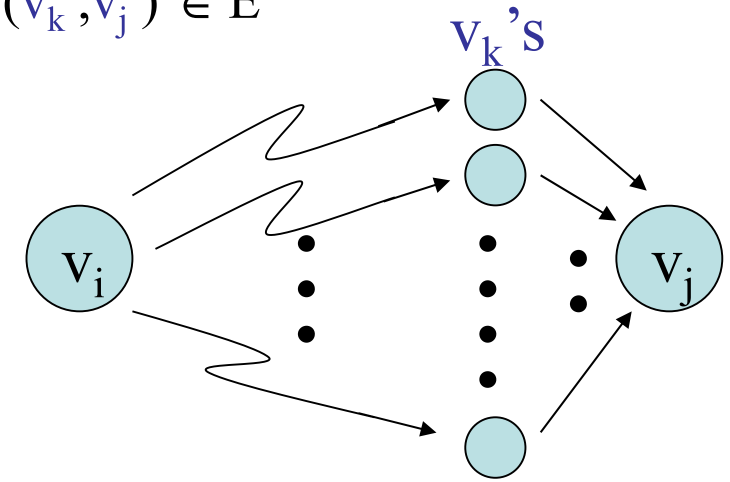
- *m ≥ 1* : $d_{ij}^{m} = \min \{ d_{ij}^{m-1} , \min_{1 \le k \le n \wedge k \ne j} \{ d_{ik}^{m-1} + \omega_{kj} \} \}$

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

since $\omega_{jj} = 0$ 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 $D^2, D^3, \ldots, D^{n-1}$, where $D^m = (d_{ij}^m)$ for $m = 1, 2, \ldots, n-1$
  - ► 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 ← 1  to n  do
    for j ← 1  to n  do
        d_ij ← ∞
        for k ← 1  to n  do
            d_ij ← min{d_ij , d_ik + ω_kj}
return D
```

MATRIX-MULT (A , B)

▶ **C** = (c_{ij}) is an n x n result matrix

```
for i ←1  to n  do
    for j ← 1  to n  do
        c_ij ← 0
        for k ← 1 to n  do
            c_ij ← c_ij + a_ik x 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

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

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

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

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

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

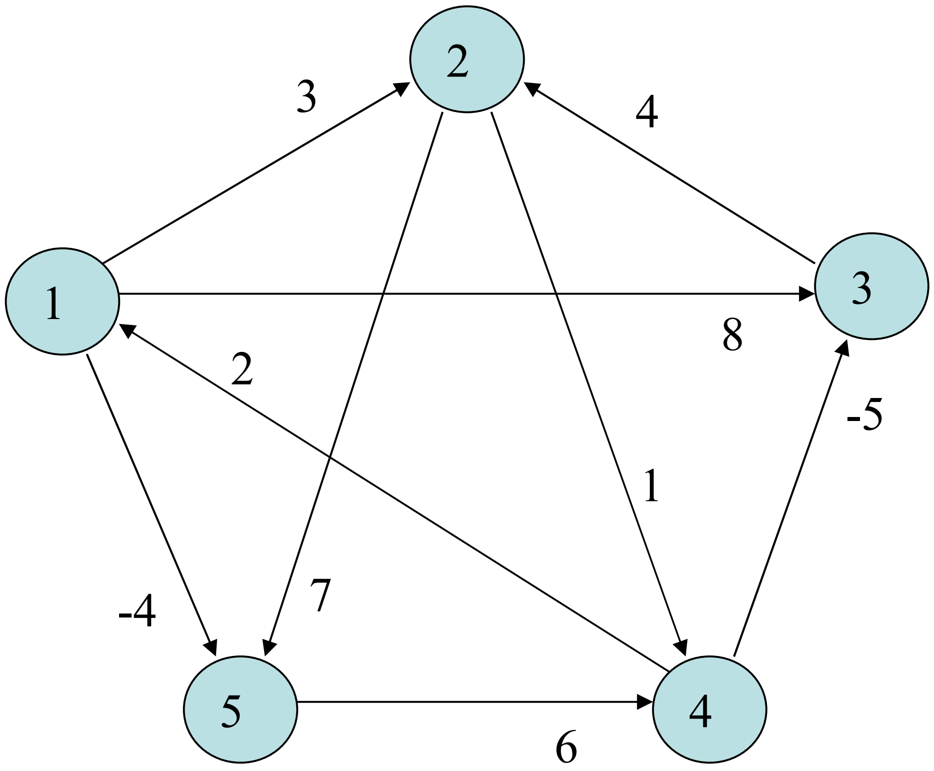
- running time : $\Theta(n^4) = \Theta(V^4)$

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

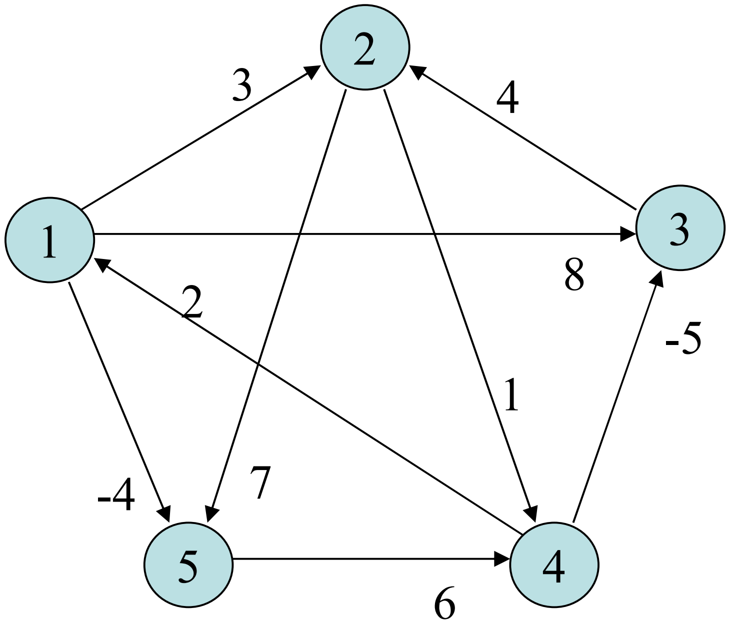
  ▶ number of matrix products : n-1

Shortest Paths and Matrix Multiplication

- Example



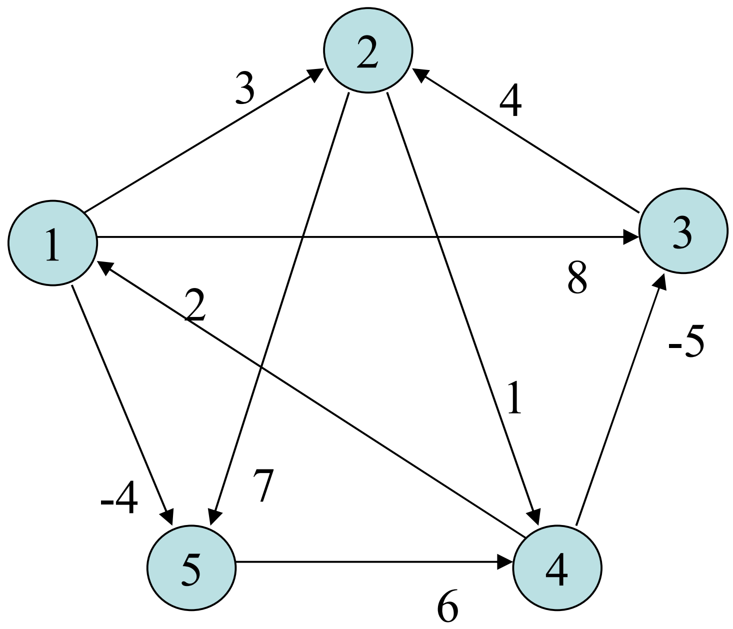
Shortest Paths and Matrix Multiplication



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

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

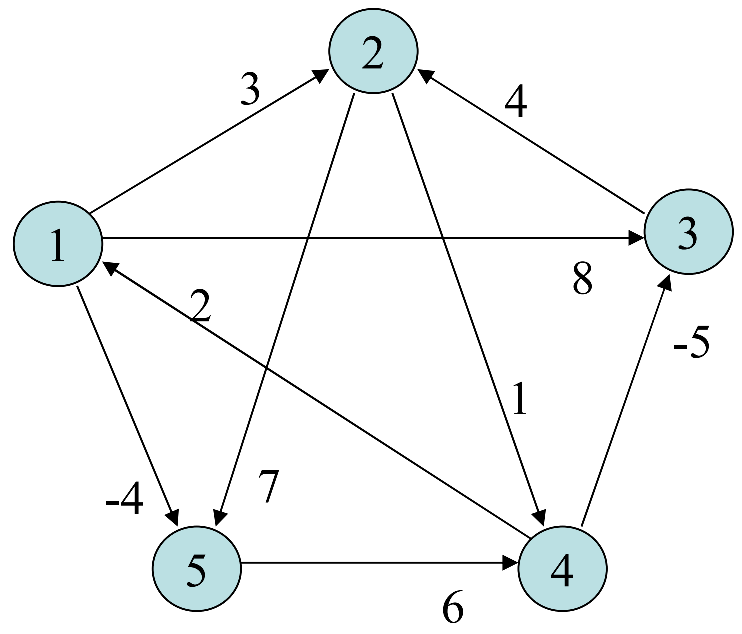
Shortest Paths and Matrix Multiplication



	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

$D^2 = D^1 W$

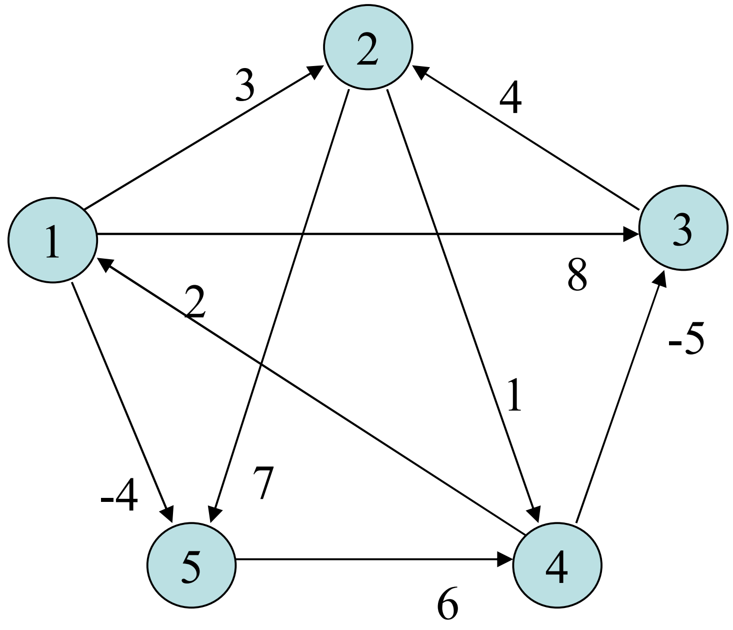
Shortest Paths and Matrix Multiplication



	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

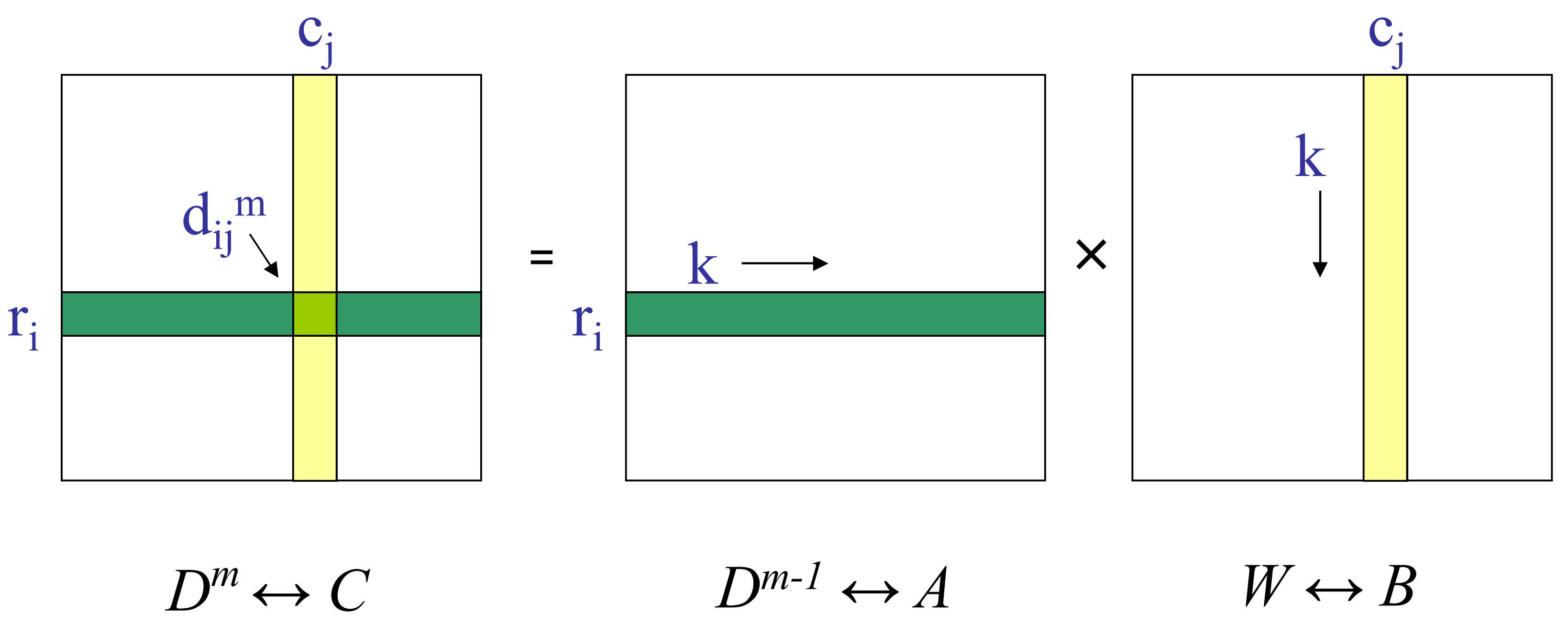


	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

$D^4 = D^3 W$

SSSP and Matrix-Vector Multiplication

- relation of APSP to one step of matrix multiplication



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) ∈ 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

$$
\begin{aligned}
D^1 &= W \\
D^2 &= W^2 = W \times W \\
D^4 &= W^4 = W^2 \times W^2 \\
D^8 &= W^8 = W^4 \times W^4 \\
&\vdots
\end{aligned}
$$

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

- 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 : $\lceil \lg(n-1) \rceil$
  - ► simple code, no complex data structures, small hidden constants in Θ-notation.

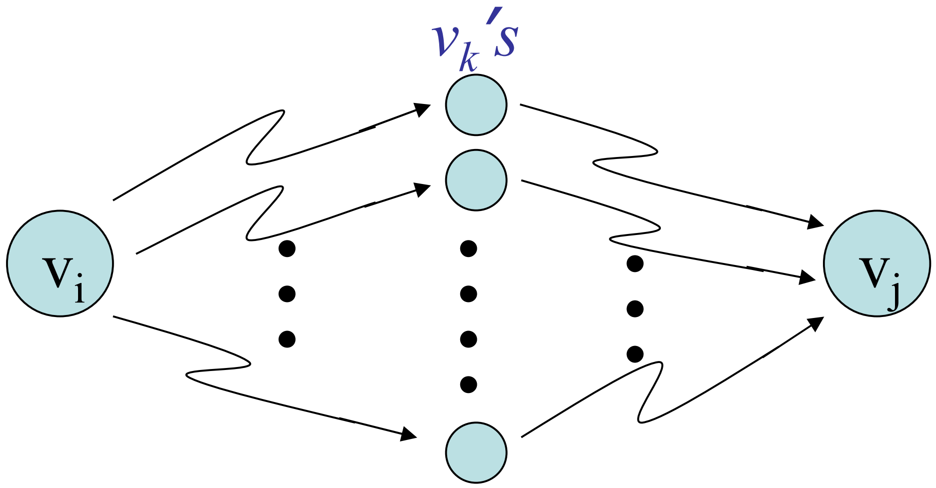
Idea Behind Repeated Squaring

- 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 $p = \langle v_1, v_2, v_3, \ldots, v_k \rangle$
  - ► 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}

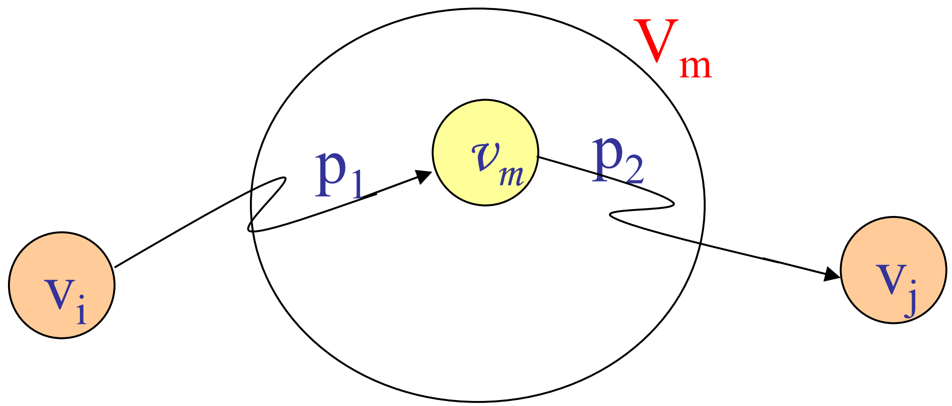
$\Rightarrow p_{ij}^m = p_{ij}^{m-1}$

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

$D \leftarrow W$

for $m \leftarrow 1$ to n do

  for $i \leftarrow 1$ to n do

    for $j \leftarrow 1$ 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

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 )
```

Johnson's Algorithm for Sparse Graphs

(1) Preserving shortest paths by edge reweighting :

- L1 : given $G = (V , E)$ with $\omega : E \rightarrow R$
  - ► let $h : V \rightarrow R$ be any weighting function on the vertex set
  - ► define $\hat{\omega}(\omega , h) : E \rightarrow R$ as $\hat{\omega}(u , v) = \omega(u , v) + h (u) - h (v)$
  - ► let $p_{0k} = < v_0 , v_1 , \ldots , v_k >$ be a path from v_0 to v_k

(a) $\hat{\omega}(p_{0k}) = \omega(p_{0k}) + h (v_0) - h (v_k)$

(b) $\omega(p_{0k}) = \delta(v_0, v_k)$ in (G, ω) $\Leftrightarrow$ $\hat{\omega}(p_{0k}) = \hat{\delta}(v_0, v_k)$ in $(G, \hat{\omega})$

(c) (G , ω) has a neg-wgt cycle $\Leftrightarrow$ $(G , \hat{\omega})$ has a neg-wgt cycle

Johnson's Algorithm for Sparse Graphs

(2) Producing nonnegative edge weights by reweighting :

- given (G, ω) with $G = (V, E)$ and $\omega : E \rightarrow R$
  construct a new graph (G', ω') with $G' = (V', E')$ and
  $\omega' = E' \rightarrow R$
  - ► $V' = V \cup \{ s \}$ for some new vertex $s \notin V$
  - ► $E' = E \cup \{ (s, v) : v \in V \}$
  - ► $\omega'(u,v) = \omega(u,v) \quad \forall (u,v) \in E$ and $\omega'(s,v) = 0, \quad \forall v \in V$

- vertex s has no incoming edges $\Rightarrow s \notin R_v$ for any v in V
  - ► no shortest paths from $u \neq s$ to v in G' contains vertex s
  - ► (G', ω') has no neg-wgt cycle $\Leftrightarrow$ (G, ω) has no neg-wgt cycle

Johnson’s Algorithm for Sparse Graphs

- suppose that G and G' have no neg-wgt cycle

- L2 : if we define $h(v) = \delta(s, v) \;\; \forall\, v \in V$ in G' and $\hat{\omega}$ according to L1.

  ► we will have $\hat{\omega}(u,v) = \omega(u,v) + h(u) - h(v) \geq 0 \;\; \forall\, v \in V$

proof : for every edge $(u, v) \in E$

$$\delta(s, v) \leq \delta(s, u) + \omega(u, v) \text{ in } G' \text{ due to triangle inequality}$$

$$h(v) \leq h(u) + \omega(u, v) \Rightarrow 0 \leq \omega(u, v) + h(u) - h(v) = \hat{\omega}(u, v)$$

Johnson's Algorithm for Sparse Graphs

Computing All-Pairs Shortest Paths

- adjacency list representation of G.
- returns n x n matrix $D = (d_{ij})$ where

  $d_{ij} = \delta_{ij}$,

  or reports the existence of a neg-wgt cycle.

Johnson's Algorithm for Sparse Graphs

- JOHNSON(G,ω)

  ▶ D=(d_{ij}) is an nxn matrix

  ▶ construct ($G' = (V', E')$, ω') s.t. $V' = V \cup \{s\}$; $E' = E \cup \{ (s,v) : \forall v \in V \}$

  ▶ $\omega'(u,v) = \omega(u,v)$, $\forall (u,v) \in E$ & $\omega'(s,v) = 0$ $\forall$ $v \in V$

  if BELLMAN-FORD(G', ω', s) = FALSE then

      return "negative-weight cycle"

  else

      for each vertex $v \in V' - \{s\} = V$ do

          $h[v] \leftarrow d'[v]$ ▶ $d'[v] = \delta'(s,v)$ computed by BELLMAN-FORD(G', ω', s)

      for each edge $(u,v) \in E$ do

          $\hat{\omega}(u,v) \leftarrow \omega(u,v) + h[u] - h[v]$ ▶ edge reweighting

      for each vertex $u \in V$ do

          run DIJKSTRA(G, $\hat{\omega}$, u) to compute $\hat{d}[v] = \hat{\delta}(u,v)$ for all v in $V \in (G,\hat{\omega})$

          for each vertex $v \in V$ do

              $d_{uv} = \hat{d}[v] - (h[u] - h[v])$

  return D

Johnson's Algorithm for Sparse Graphs

- running time : $O(V^2 \lg V + EV)$
  - ► edge reweighting

    BELLMAN-FORD(G', ω', s) : $O(EV)$

    computing ω values : $O(E)$
  - ► |V| runs of DIJKSTRA : $|V| \times O(V \lg V + EV)$

    $= O(V^2 \lg V + EV)$;

    PQ = fibonacci heap