APSP

Computer Algorithm

TY-IT

Agenda

- Review of
  - Prims algorithm for MST
  - Dijkstra’s algorithm
  - Example
- APSP consideration
  - Matrix operation
  - Repeated squaring
- Floyd Warshall algorithm
- Transitive closure

APSP

- Weight matrix
  - X axis- source, Y axis- Destination
  - Matrix element W=$w_{i,j}$- weight of edge from I to j
- For each pair (i,j) see if there is shortest path through other vertex k
- $L^{(m)}_{i,j}$ be shortest path length from i,j with max m edges
- Compute $L^{(1)}_{i,j}$, $L^{(2)}_{i,j}$ ….$L^{(m)}_{i,j}$

APSP

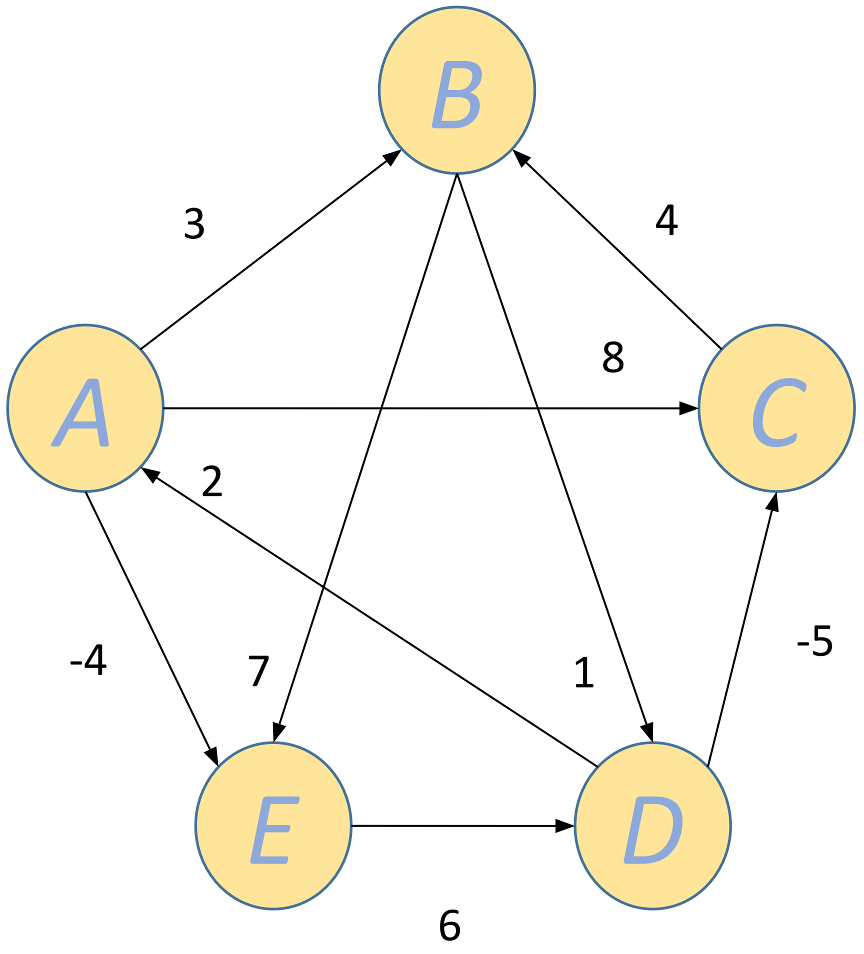
EXTEND-SHORTEST-PATHS(L, W)

$n \leftarrow rows[L]$
let $L' = (l'_{ij})$ be an $n \times n$ 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 $l'_{ij} \leftarrow \min(l'_{ij}, l_{ik} + w_{kj})$
return L'

- Repeat for n-1 times
- Complexity n^4
- Optimization- Repeated squaring –n^3 log n

APSP ANIMATED EXAMPLE

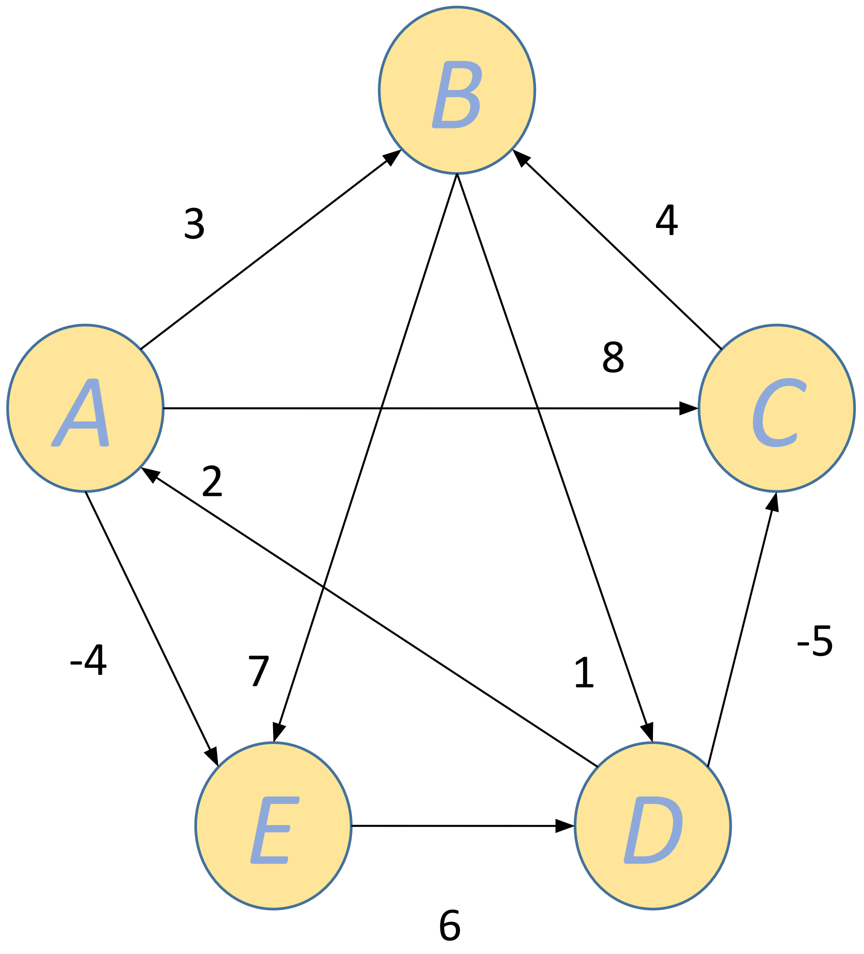
		A	B	C	D	E
	A	0	3	8	∞	-4
	B	∞	0	∞	1	7
W	C	∞	4	0	∞	∞
	D	2	∞	-5	0	∞
	E	∞	∞	∞	6	0



APSP ANIMATED EXAMPLE

		A	B	C	D	E
	A	0	3	8		-4
	B		0		1	7
$L^{(1)}$	C		4	0		
	D	2		-5	0	
	E				6	0

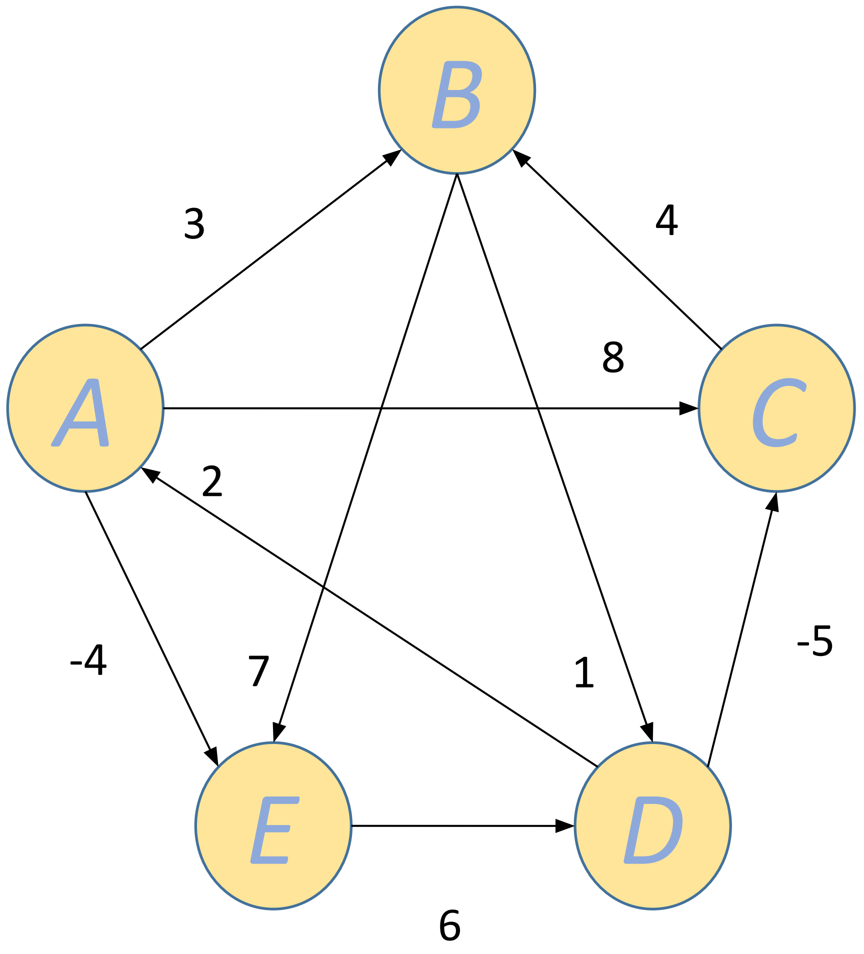
		A	B	C	D	E
	A	0	3	8	2	-4
	B	3	0	-4	1	7
$L^{(2)}$	C		4	0	5	11
	D	2	-1	-5	0	-2
	E	8		1	6	0



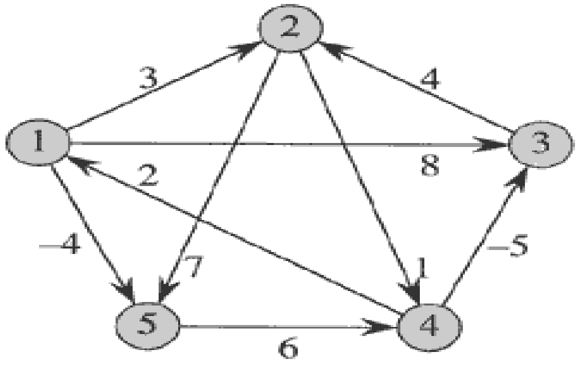
APSP ANIMATED EXAMPLE

		A	B	C	D	E
	A	0	3	-3	2	-4
	B	3	0	-4	1	-1
$L^{(3)}$	C	7	4	0	5	11
	D	2	-1	-5	0	-2
	E	8	5	1	6	0

		A	B	C	D	E
	A	0	1	-3	2	-4
	B	3	0	-4	1	-1
$L^{(4)}$	C	7	4	0	5	3
	D	2	-1	-5	0	-2
	E	8	5	1	6	0



APSP: Example



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

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

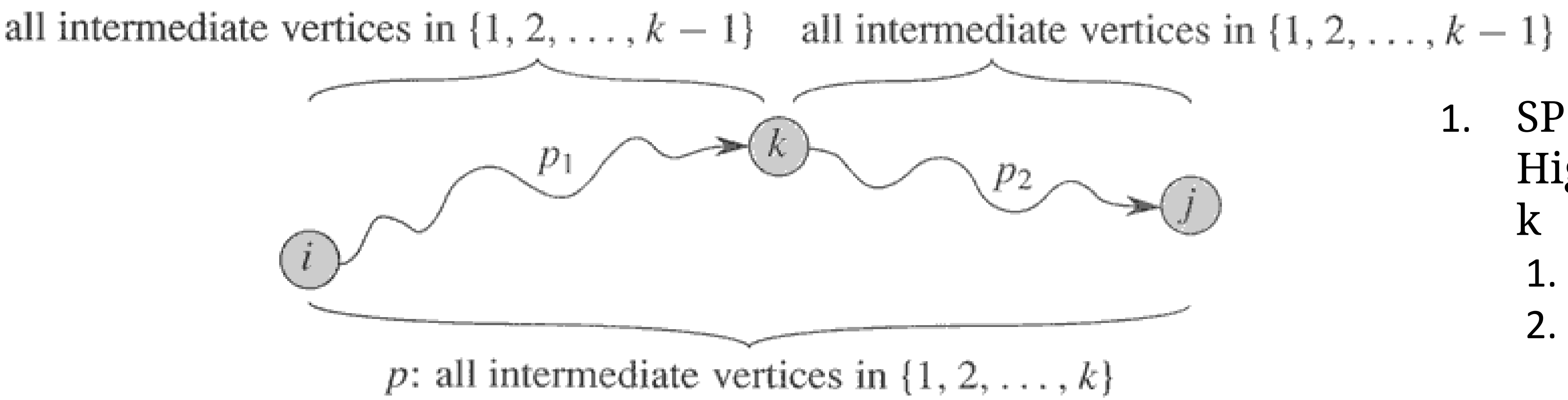
Figure 25.1 A directed graph and the sequence of matrices $L^{(m)}$ computed by SLOW-ALL-PAIRS-SHORTEST-PATHS. The reader may verify that $L^{(5)} = L^{(4)} \cdot W$ is equal to $L^{(4)}$, and thus $L^{(m)} = L^{(4)}$ for all $m \geq 4$.

APSP

- Repeat for n-1 times
- Complexity n^4
- Optimization- Repeated squaring –n^3 log n

Floyd Warshall Algorithm

- Index of vertex
- Intermediate vertex



1. SP from i to j may go thr Highest Index Vertex (HIV) k
   1. No – HIV k-1
   2. Yes, consider two sub paths { i to k, k to j}
2. Compute $d_{ij}^{(k)}$, start from $d_{ij}^{(0)}$

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

If ($d^{(k-1)} > d^{(k-1)} + d^{(k-1)}$) then $\square^{(k)} := \square^{(k-1)}$)

Flyod Warshall (FW) Algorithm

FLOYD-WARSHALL(W)

$n \leftarrow rows[W]$

$D^{(0)} \leftarrow W$

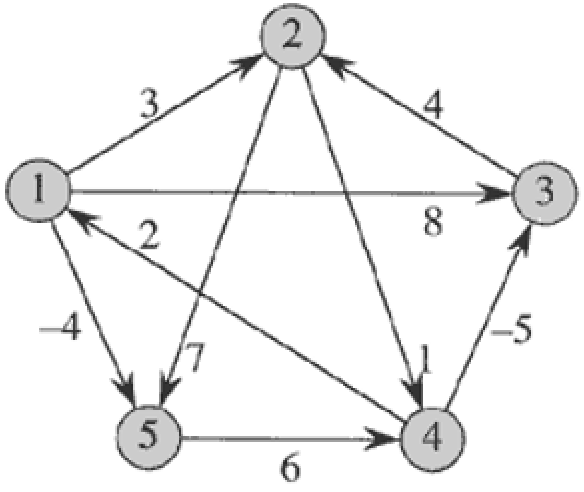
for $k \leftarrow 1$ **to** n

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

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

            do $d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$

return $D^{(n)}$



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

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

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

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

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

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \end{pmatrix}$$

Transitive closure

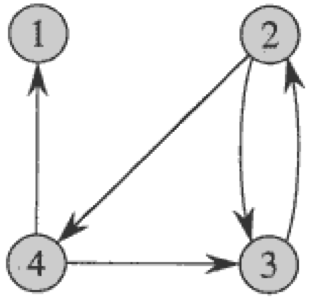
- Adjacency matrix to denote connectivity between nodes
  - 1- Path exixts
  - 0- No path
- Floyd Warshall algorithm can be used
- Start from $t_{ij}^{(0)}$
- $t_{ij}^{(k)} := t_{ij}^{(k-1)} \lor (t_{ik}^{(k-1)} \land t_{kj}^{(k-1)})$

Transitive closure (TC)

TRANSITIVE-CLOSURE(G)

$n \leftarrow |V[G]|$
for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
        do if $i = j$ or $(i, j) \in E[G]$
            then $t_{ij}^{(0)} \leftarrow 1$
            else $t_{ij}^{(0)} \leftarrow 0$
for $k \leftarrow 1$ **to** n
    do for $i \leftarrow 1$ **to** n
        do for $j \leftarrow 1$ **to** n
            do $t_{ij}^{(k)} \leftarrow t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right)$
return $T^{(n)}$

TC: Example



$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

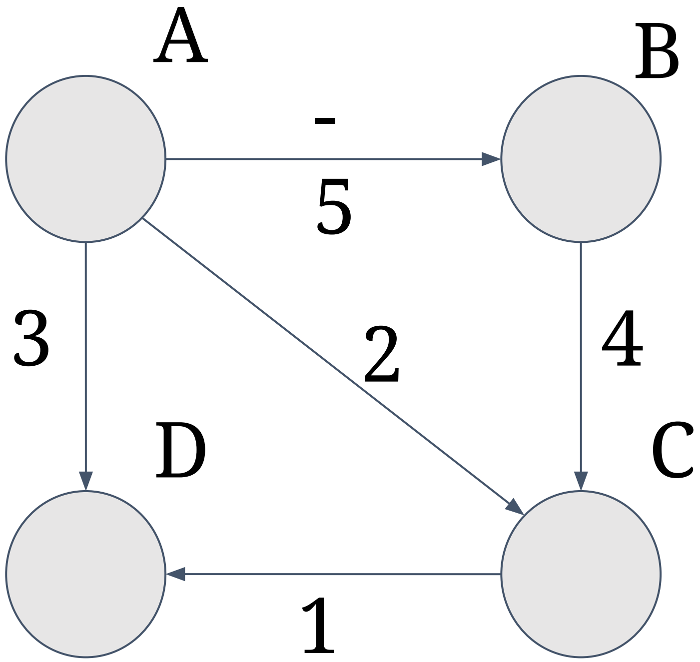
$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Figure 25.5 A directed graph and the matrices $T^{(k)}$ computed by the transitive-closure algorithm.

Johnson algorithm

- Used for sparse graph
- Let the given graph be G. Add a new vertex s to the graph, add edges from s to all vertices of G. Let the modified graph be G’.
- Run Bellman-Ford algorithm on G’ with s as source. Let the distances calculated by Bellman-Ford be h[0], h[1], .. h[V-1].
- Reweight : For each edge (u, v), assign the new weight as “original weight + h[u] – h[v]”.
- Remove the added vertex s and run Dijkstra’s algorithm for every vertex.

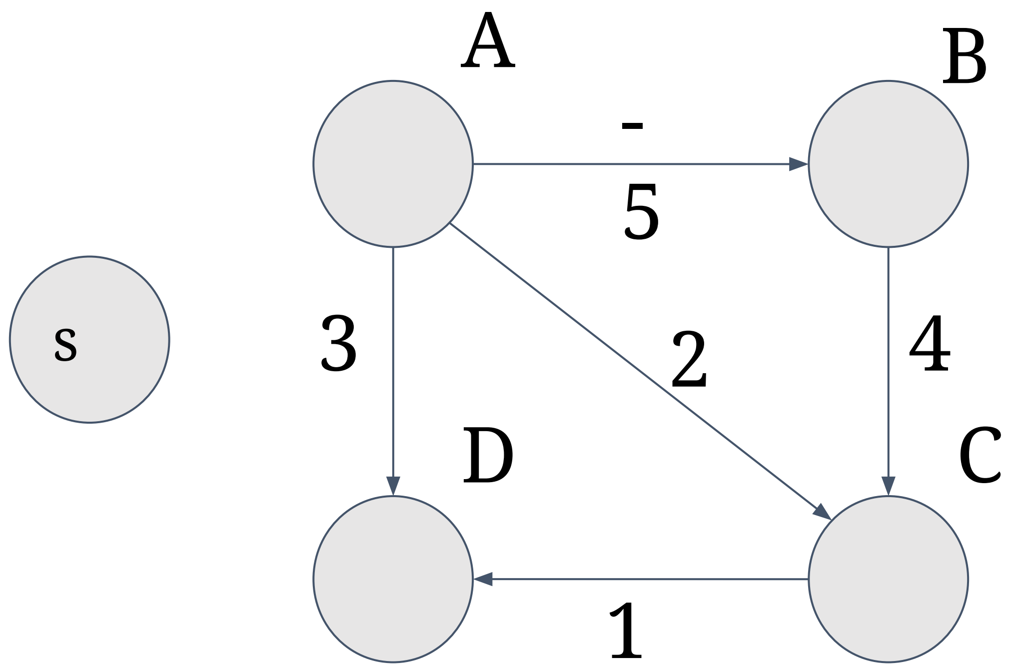
Johnson Algorithm: Example



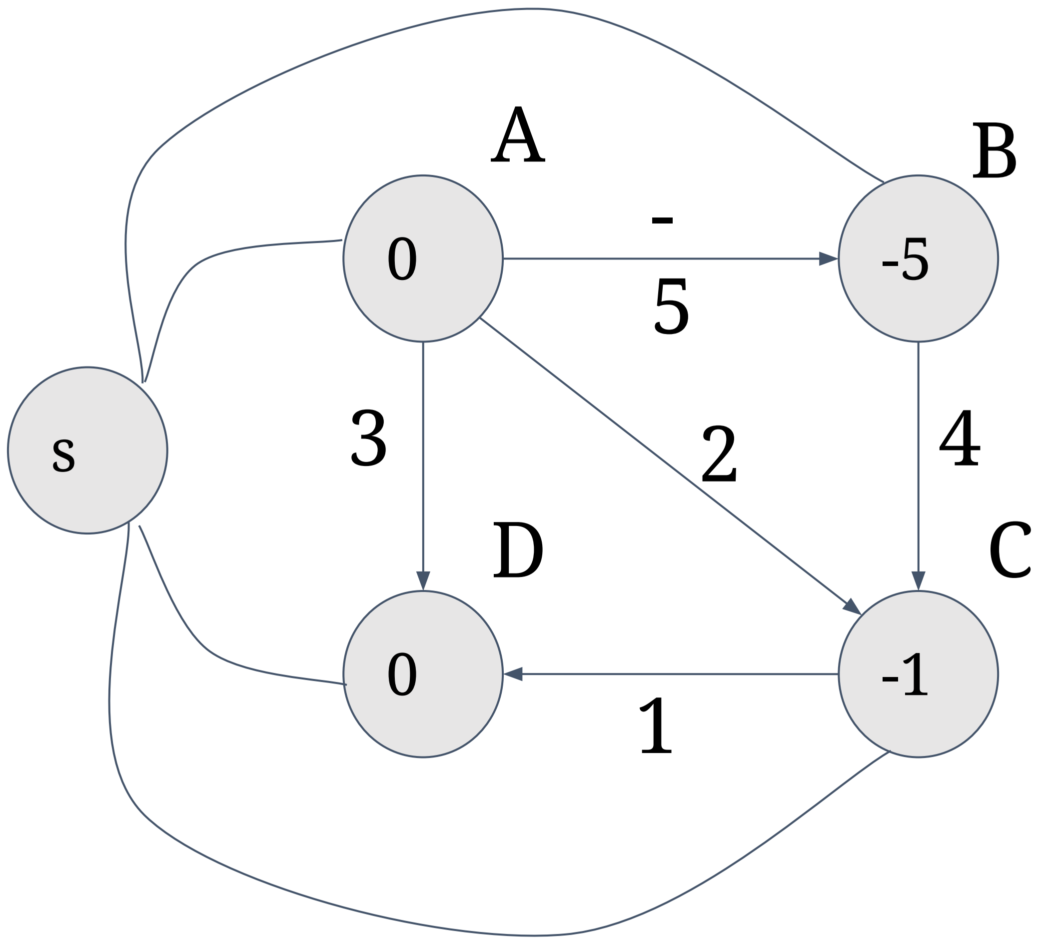
The reason that Johnson's algorithm is better for sparse graphs is that its time complexity depends on the number of edges in the graph, while Floyd-Warshall's does not. Johnson's algorithm runs in O(V^2 log(V) + |V| |E|) time.

So, if the number of edges is small (i.e. the graph is sparse), it will run faster than the O(V^3) runtime of Floyd-Warshall.

Johnson Algorithm: Example

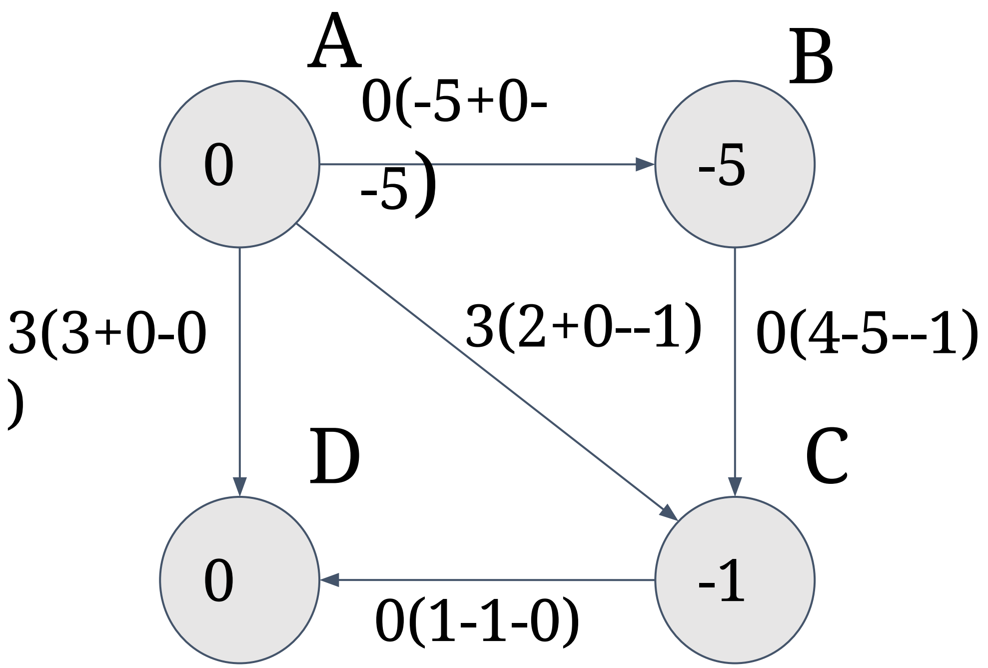


Johnson Algorithm: Example



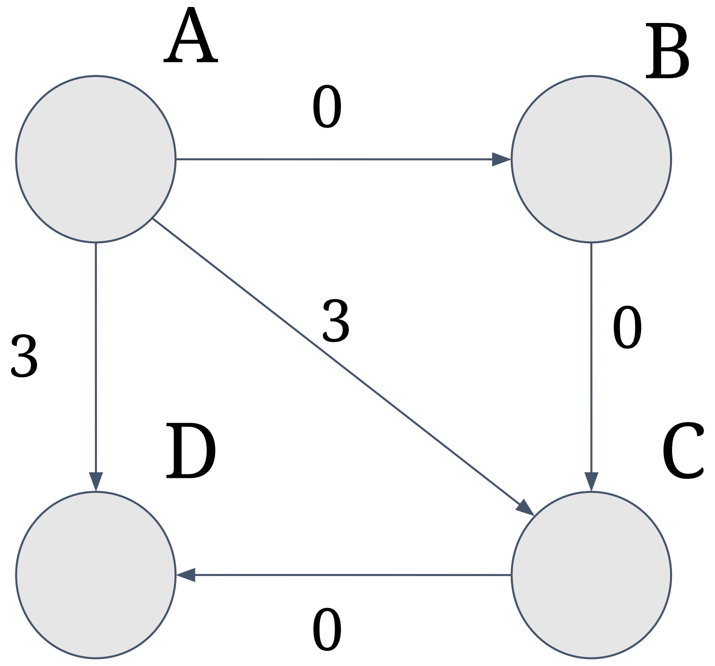
- Apply Bellman Ford Algorithm from s

Johnson Algorithm: Example



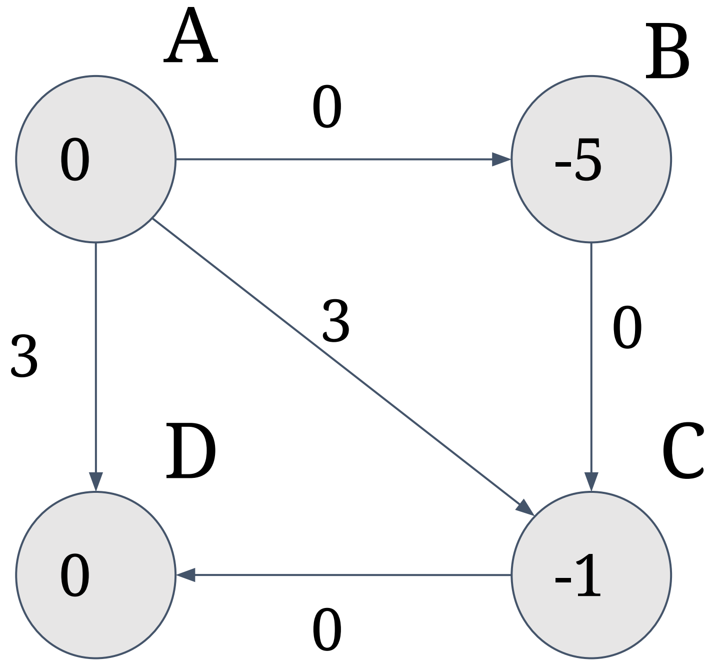
- remove the source vertex s
- reweight the edges using following formula. w(u, v) = w(u, v) + h[u] – h[v]

Johnson Algorithm: Example



- Run Dijkstra's algorithm from every vertex

Johnson Algorithm: Example



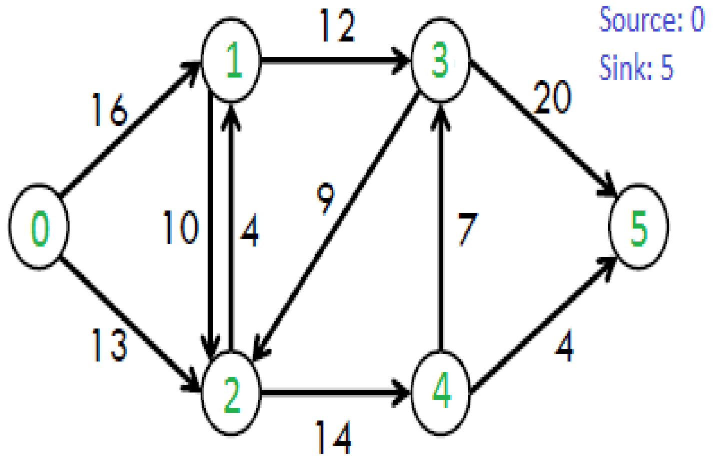
- Run Dijkstra's algorithm from every vertex

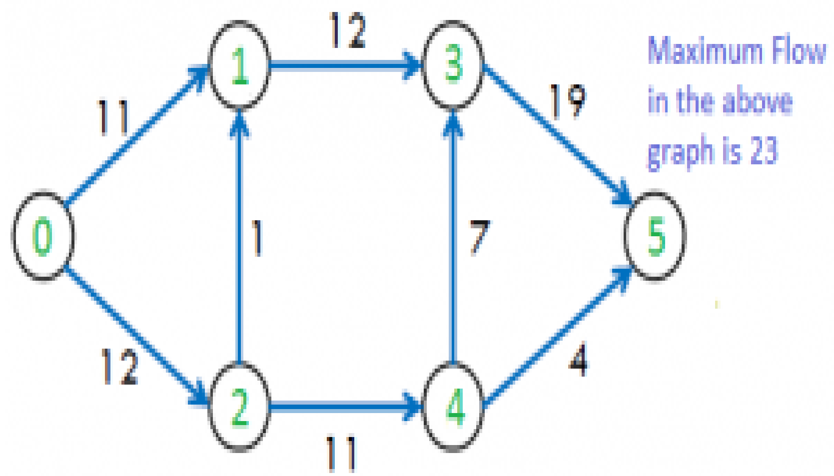
Maximum flow: Example

- Maximum flow problems involve finding a feasible flow through a single-source, single-sink flow network that is maximum.

- Each edge is labeled with capacity, the maximum amount of stuff that it can carry. The goal is to figure out how much stuff can be pushed from the vertex s(source) to the vertex t(sink).

- maximum flow possible is : **23**





- Flow network
  - Directed graph
  - One source, sink
  - Non negative weights edge
- Residual graph Gf
  - Edges from G that have some non zero flow
    - $C_f(u,v) = c(u,v) - f(u,v)$
  - Add the edges: Reverse edges to decrease the flow
    - $C_f(v,u) = c(v,u) + f(u,v)$
- Augmenting path (p)
  - Path from s to t
  - Residual capacity: Smallest capacity of path p

Ford Fulkerson (FF) Algorithm

- $f_m = 0$
- While there exists augmenting path
  - Identify augmenting path
  - $C_f(p)$ = smallest capacity of p
  - $f_m = f_m + C_f(p)$
  - For each edge in p
    - $C_f(u,v) = c(u,v) - f(u,v)$
    - $C_f(v,u) = c(v,u) + f(u,v)$

FF Algorithm: Example

- https://www.youtube.com/watch?v=Tl90tNtKvxs

FF Algorithm: Complexity

- Flow increased by at least one unit in every iteration
  - max value of f_m -max flow times
  - Finding augmenting path- O(E)
- Total complexity: $O(E^* f_m)$

Min cut

- **Minimal Cut :** It is one which replacement of any of its member reconnects the Graph.
- **Minimum Cut :** In a weighted graph each cut has capacity. A cut with minimum capacity is minimum cut.
- **Max Flow Min Cut Theorem :** The maximum flow between any two arbitrary nodes in any graph cannot exceed the capacity of the minimum cut separating those two nodes.

- Thank you