Walchand College of Engineering
(Government Aided Autonomous Institute)
Vishrambag, Sangli. 416415

Computer Algorithms

All Pair Shortest Path (APSP)

24-25
D B Kulkarni
Information Technology Department

Agenda

- APSP as an extension of SPSP/SSSP
- APSP consideration
  - Matrix operation: Complexity n^4
  - Repeated squaring: Complexity n^3 log n
- Floyd Warshall algorithm
- Transitive closure
- Max Flow
  - Introduction
  - Ford Fulkerson Algorithm
  - Min cut

APSP: Dynamic programming

Dynamic Programming Principles	Structure of shortest path Recursive solution to APSP Computing the shortest path weights from bottom-up

Consider the n × n weighted adjacency matrix

A = (a_{ij}), where a_{ij} = w(i, j) or ∞

$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.

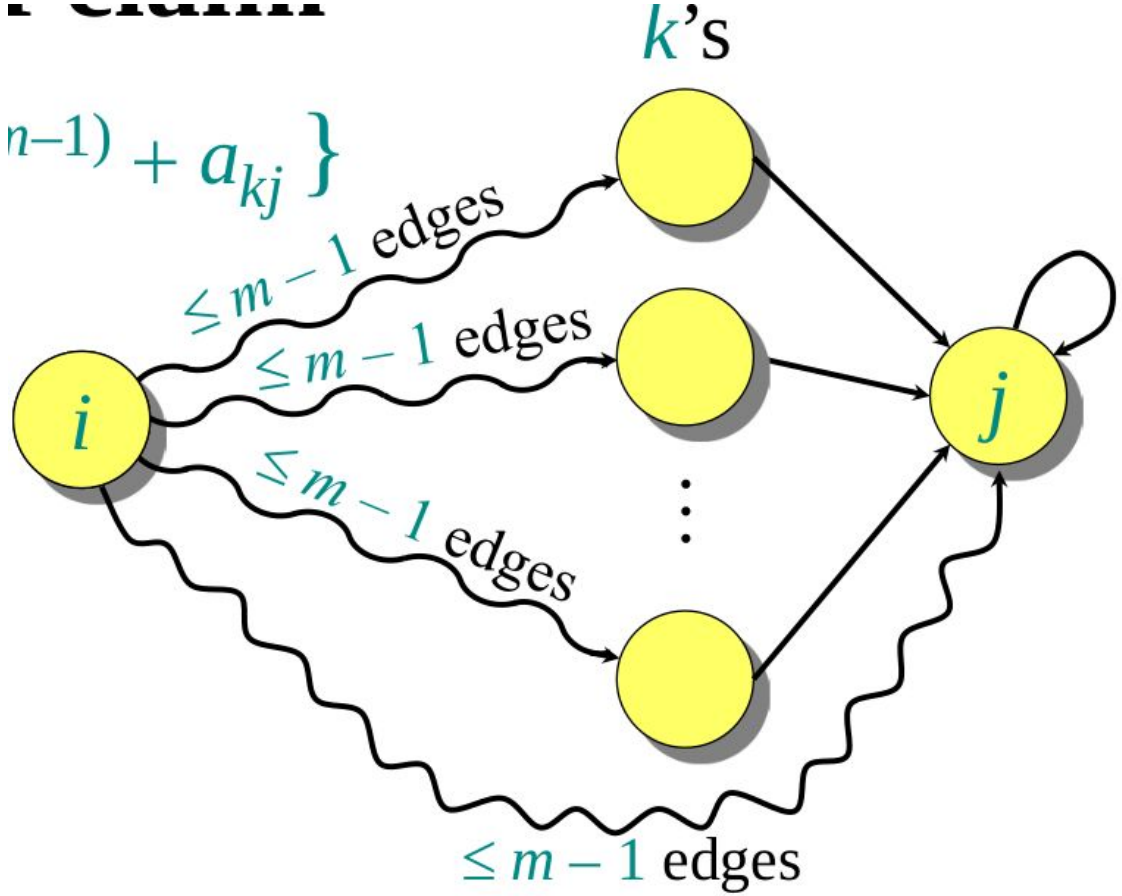
We have

d_{ij} =0 if i = j

=∞ if i ≠ j;

and for m = 1, 2, …, n – 1,

$d_{ij}^{(m)} = \min_k\{d_{ij}^{(m-1)}, d_{ik}^{(m-1)} + a_{kj}\}$



APSP

```
EXTEND-SHORTEST-PATHS(L, W)
1  n ← rows[L]
2  let L' = (l'_ij) be an n × n matrix
3  for i ← 1 to n
4       do for j ← 1 to n
5            do l'_ij ← ∞
6               for k ← 1 to n
7                    do l'_ij ← min(l'_ij, l_ik + w_kj)
8  return L'
```

- Extend- Add edge form destination in every iteration
- Repeat for n-1 times
- Complexity n^4
- Optimization- Repeated squaring –n^3 log n

Matrix multiplication

Compute $C = A \cdot B$, where C, A, and B are $n \times n$ matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Time = $\Theta(n^3)$ using the standard algorithm.

What if we map "+" $\rightarrow$ "min" and "$\cdot$" $\rightarrow$ "+"?

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\}$$

Thus, $D^{(m)} = D^{(m-1)}$ "$\times$" A.

Consequently, we can compute

$D^{(1)} = D^{(0)} \cdot A = A^1$

$D^{(2)} = D^{(1)} \cdot A = A^2$

$D^{(n-1)} = D^{(n-2)} \cdot A = A^{n-1}$,

yielding $D(n-1) = (\delta(i, j))$.

Matrix multiplication: Repeated squaring

Repeated squaring: $A^{2k} = A^k \times A^k$.

Compute A^2 , A^4, …, $A^{2\wedge(\lg(n-1))}$

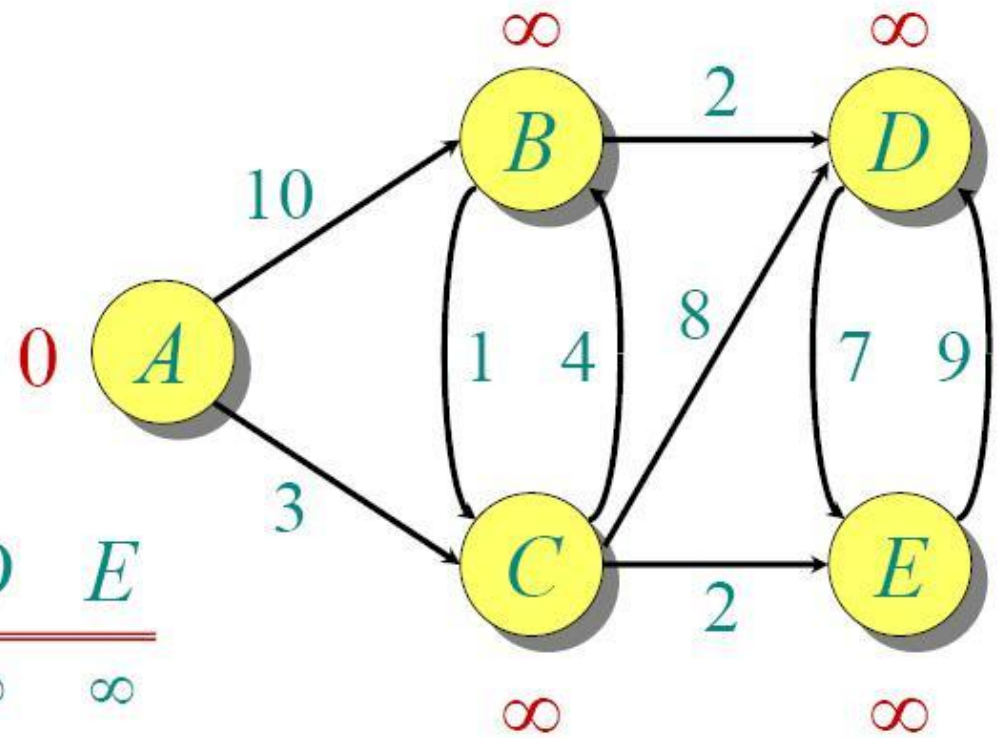
O(lg n) squarings

Note: $A^{n-1} = A^n = A^{n+1} = \ldots..$

Time = $\Theta(n^3 \lg n)$.

APSP Animated Example

Initialize:

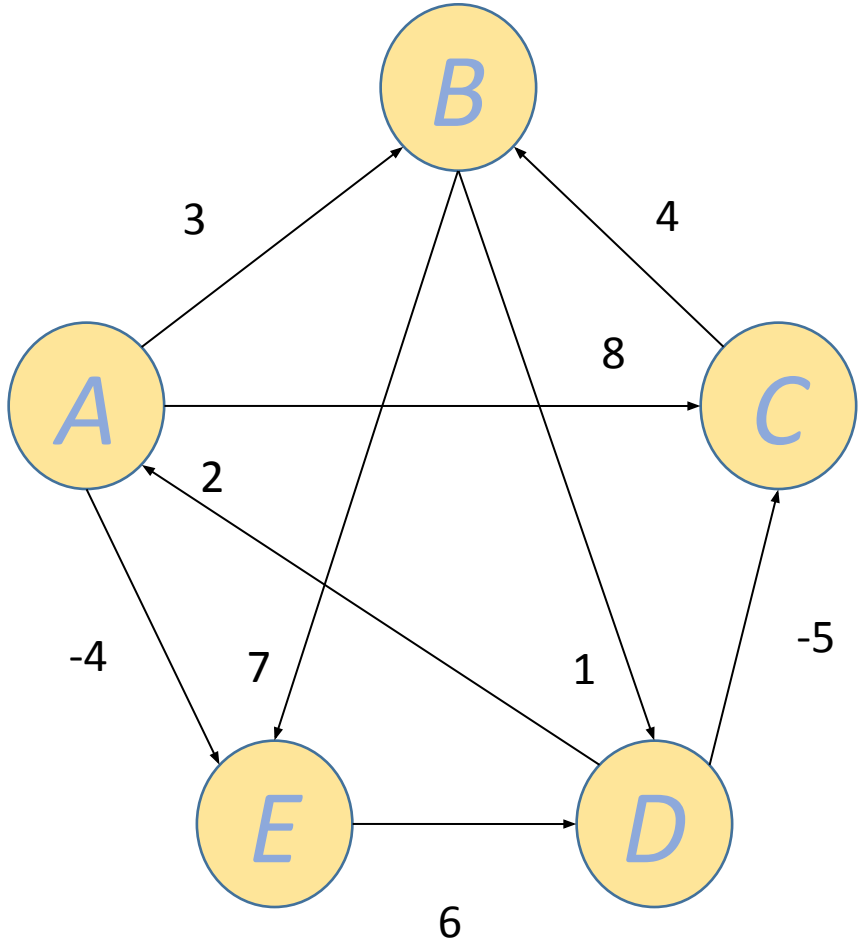


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

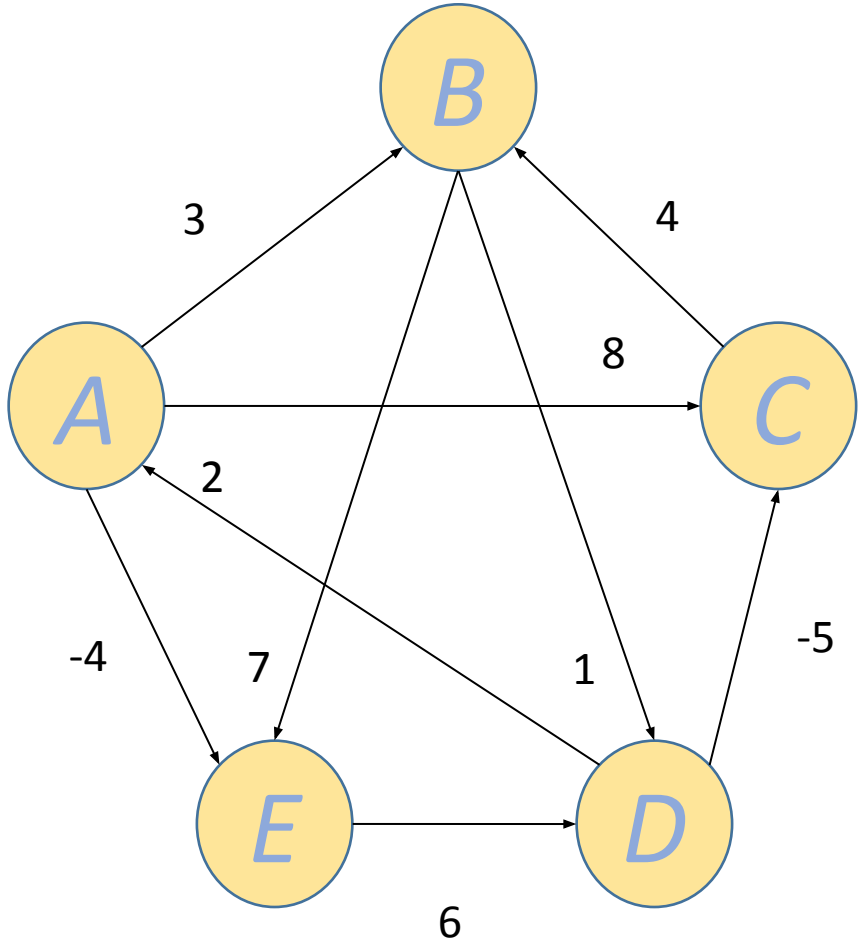
APSP Animated Example

		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



APSP Animated Example

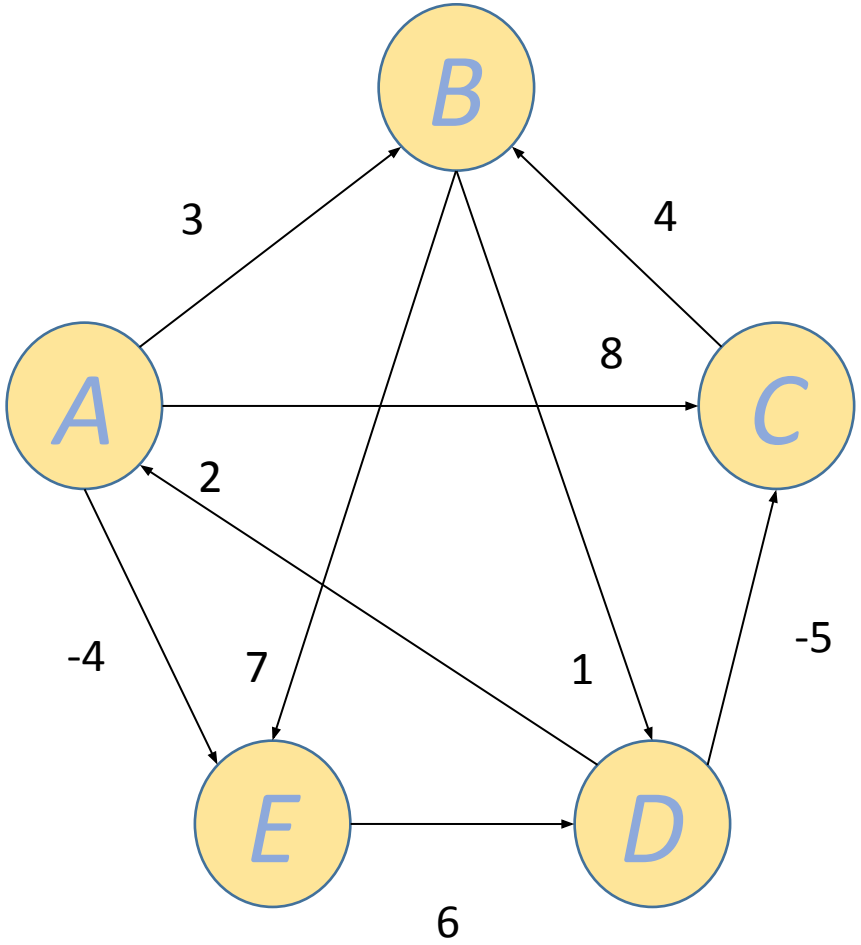
		A	B	C	D	E
	A	0	3	8	∞	-4
	B	∞	0	∞	1	7
$L^{(1)}$	C	∞	4	0	∞	∞
	D	2	∞	-5	0	∞
	E	∞	∞	∞	6	0



		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

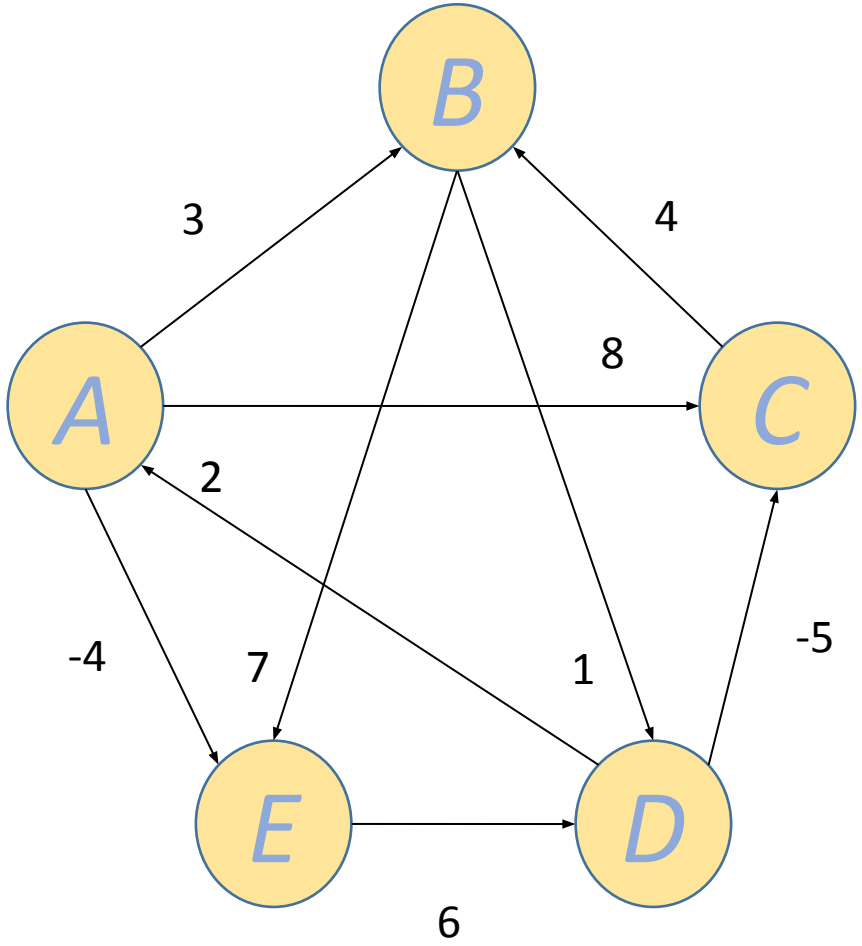
		A	B	C	D	E
	A	0	3	8	2	-4
	B	3	0	-4	1	7
$L^{(2)}$	C	∞	4	0	5	11
	D	2	-1	-5	0	-2
	E	8	∞	1	6	0



		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

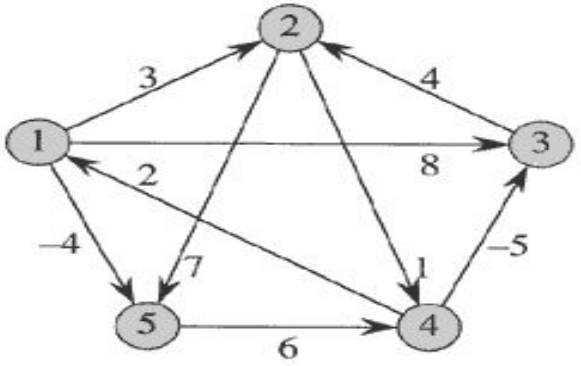
APSP Animated Example

		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 Analysis

- with Matrix operation: Complexity n^4
- Instead of increasing path length by one edge use “Repeated squaring”
  - Complexity $n^3 \log n$

Floyd Warshall Algorithm

- Based on Index of vertex

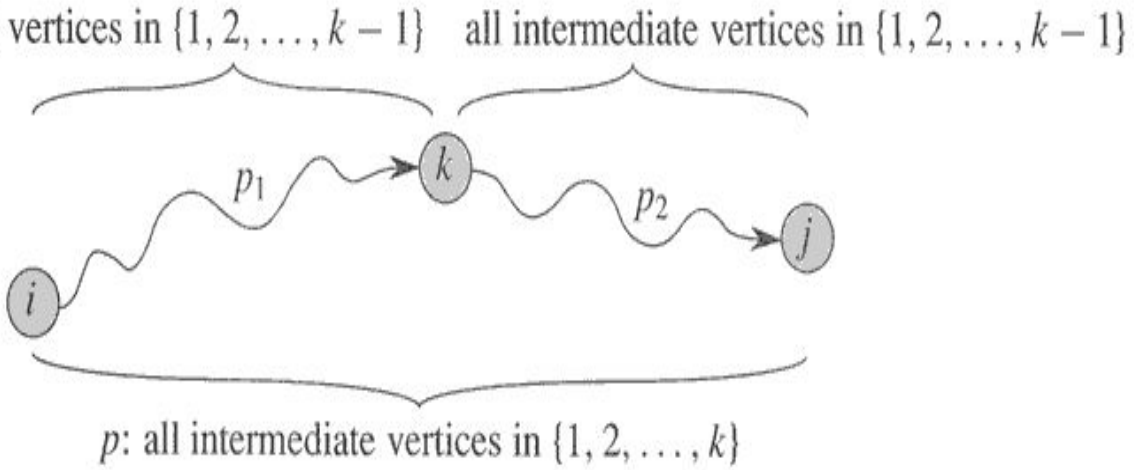


Figure 25.3 Path p is a shortest path from vertex i to vertex j, and k is the highest-numbered intermediate vertex of p. Path p_1, the portion of path p from vertex i to vertex k, has all intermediate vertices in the set $\{1, 2, \ldots, k-1\}$. The same holds for path p_2 from vertex k to vertex j.

- Does SP from vertex i to j go thr Highest Index Vertex (HIV) of k?
  - No – HIV could be k-1
  - Yes, consider two sub paths {i to k} and {k to j}
  - Recurse
- Compute $d_{ij}^{(k)}$, start from $d_{ij}^{(0)}$

- SP length estimate for i to j with highest intermediate vertex k is $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
- $\prod_{ij}^{(k)}$ is highest index of intermediate vertex between shortest path from i to j ,

  if $(d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$ then $\prod_{ij}^{(k)} = \prod_{kj}^{(k-1)}$

Floyd Warshall Algorithm

```
FLOYD-WARSHALL(W)
1   n ← rows[W]
2   D^(0) ← W
3   for k ← 1 to n
4       do for i ← 1 to n
5           do for j ← 1 to n
6               do d_ij^(k) ← min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1))
7   return D^(n)
```

Floyd Warshall Algorithm

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

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

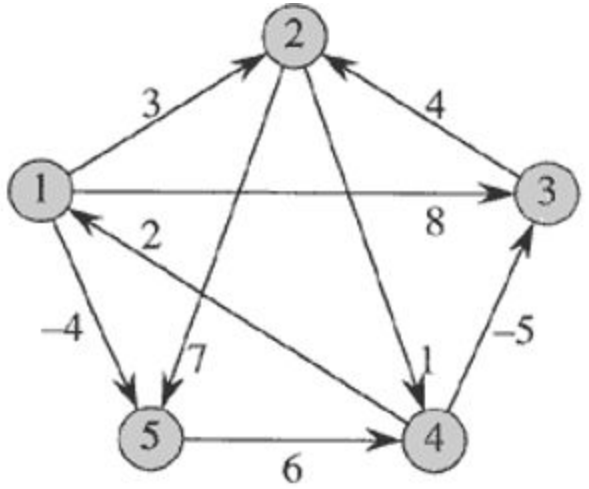
$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL}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

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

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

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

Figure 25.4 The sequence of matrices $D^{(k)}$ and $\Pi^{(k)}$ computed by the Floyd-Warshall algorithm for the graph in Figure 25.1.



- Only final matrix D and ∏ are enough to compute SP
- Find shortest path from vertex 1 to 3
- Find $D_{1,3}^{(5)}$=-3 (shortest path length)
- Find intermediate highest index $\prod_{1,3}^{(5)}$=4 (vertex)
  - shortest path exists from (1…4…3)
  - (1.. $\prod_{1,4}^{(5)}$=5,4, $\prod_{4,3}^{(5)}$=4,3)=(1..5..4..3)
  - (1,$\prod_{1,5}^{(5)}$=1, 5, $\prod_{5,4}^{(5)}$=5, $\prod_{5,4}^{(5)}$=5,4, $\prod_{4,3}^{(5)}$=4,3) =(1,5,4,3)

Transitive closure

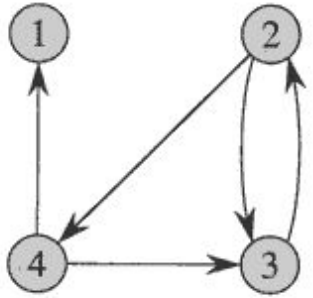
- Adjacency matrix to denote connectivity between nodes
  - 1- Path exists
  - 0- No path
- Floyd Warshall algorithm can be used
- Start from $t_{ij}^{(0)}$
- $t_{ij}^{(k)} := t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$

Transitive closure

```
TRANSITIVE-CLOSURE(G)
 1  n ← |V[G]|
 2  for i ← 1 to n
 3       do for j ← 1 to n
 4            do if i = j or (i, j) ∈ E[G]
 5                 then t_ij^(0) ← 1
 6                 else t_ij^(0) ← 0
 7  for k ← 1 to n
 8       do for i ← 1 to n
 9            do for j ← 1 to n
10                 do t_ij^(k) ← t_ij^(k−1) ∨ (t_ik^(k−1) ∧ t_kj^(k−1))
11  return T^(n)
```

Transitive closure (TC): Example

TC: Example



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

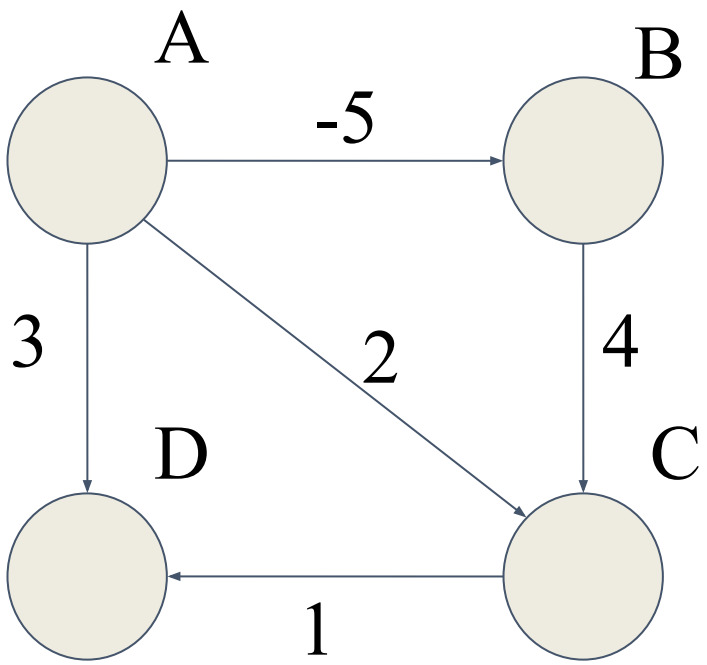
$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

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.

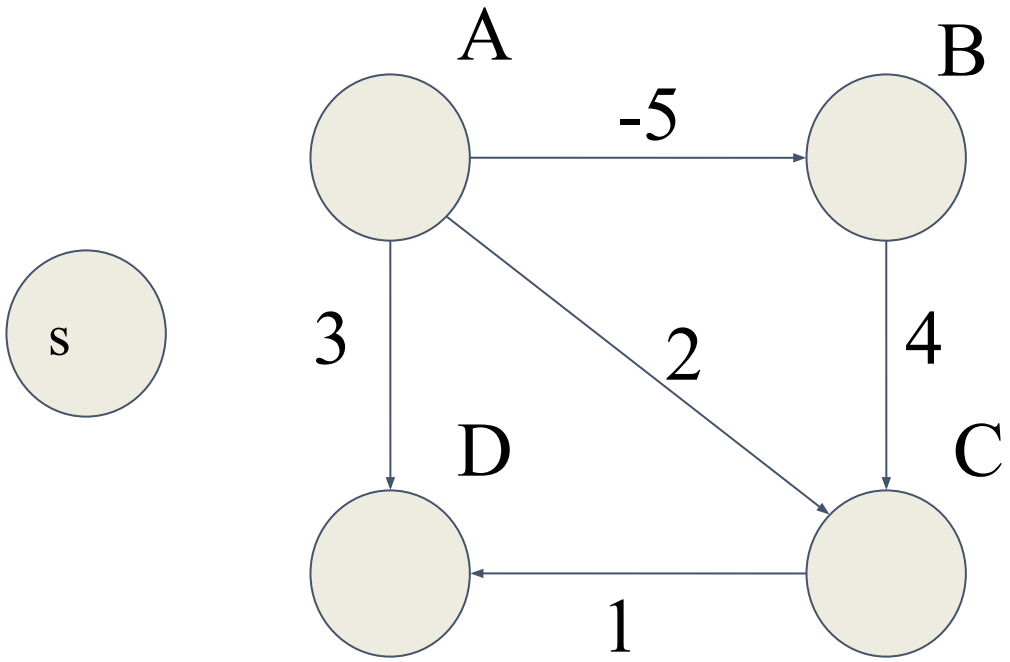
Johnson algorithm: Example



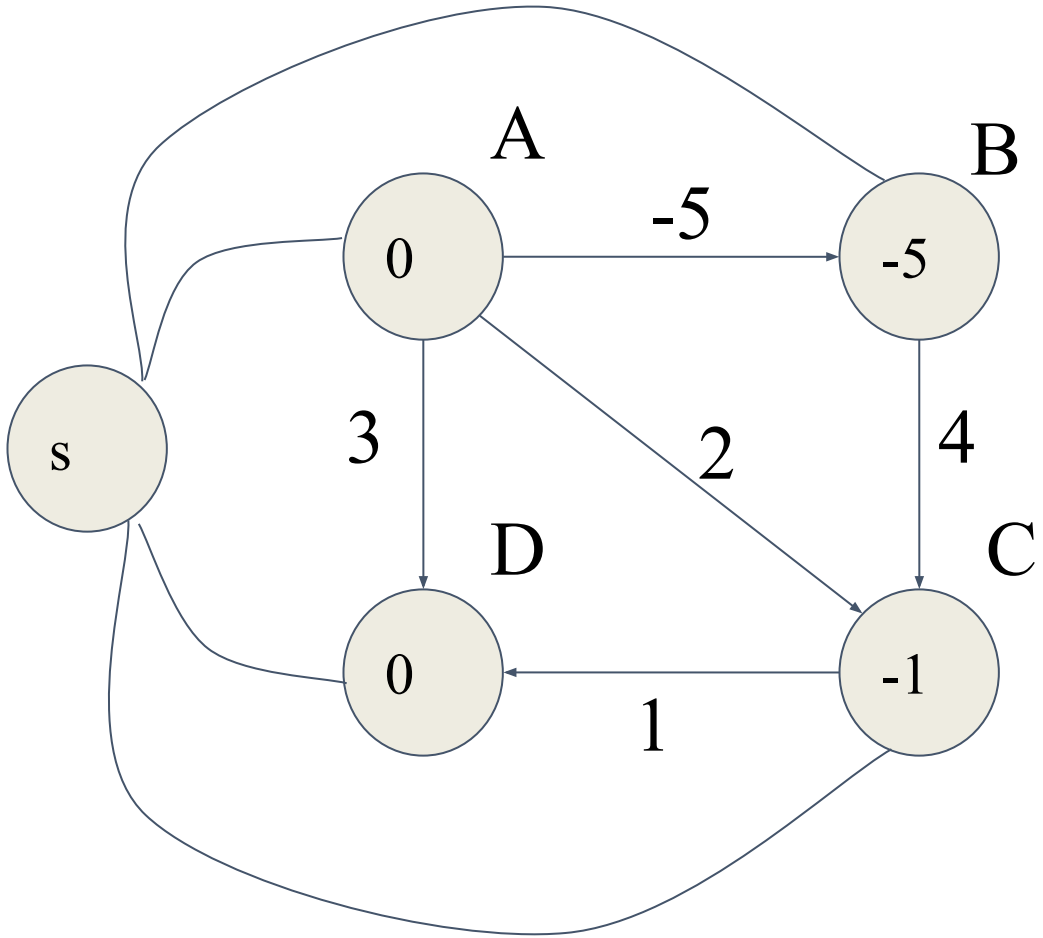
The reason that Johnson's algorithm is better for sparse graphs is that its time complexity depends on the number of edges in the graph, while Floyd-Warshall's does not. Johnson's algorithm runs in $O(V^2 \log(V) + |V|\,|E|)$ time.

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

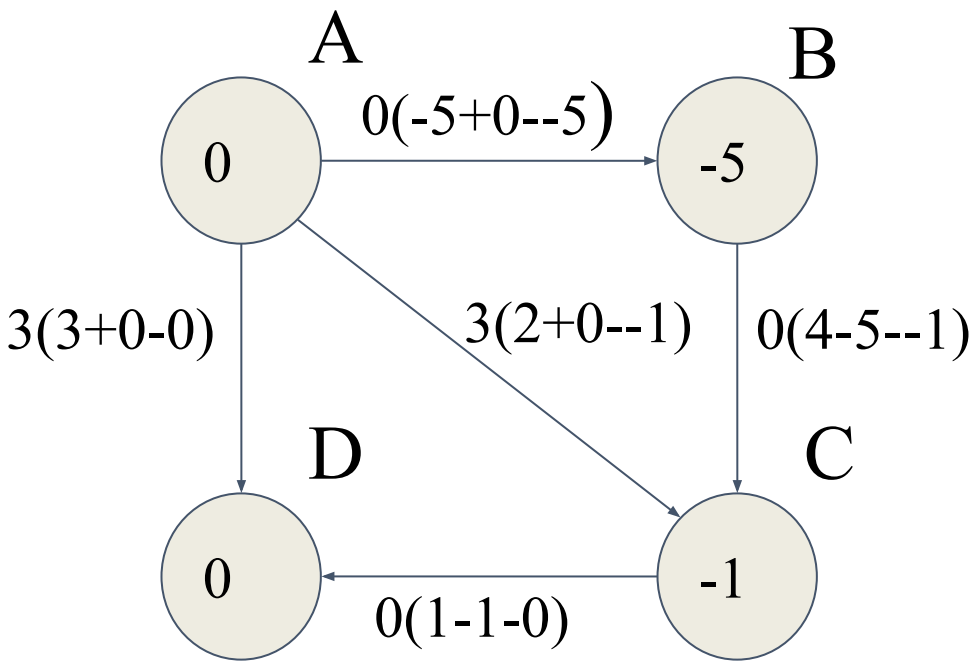


Johnson algorithm: Example



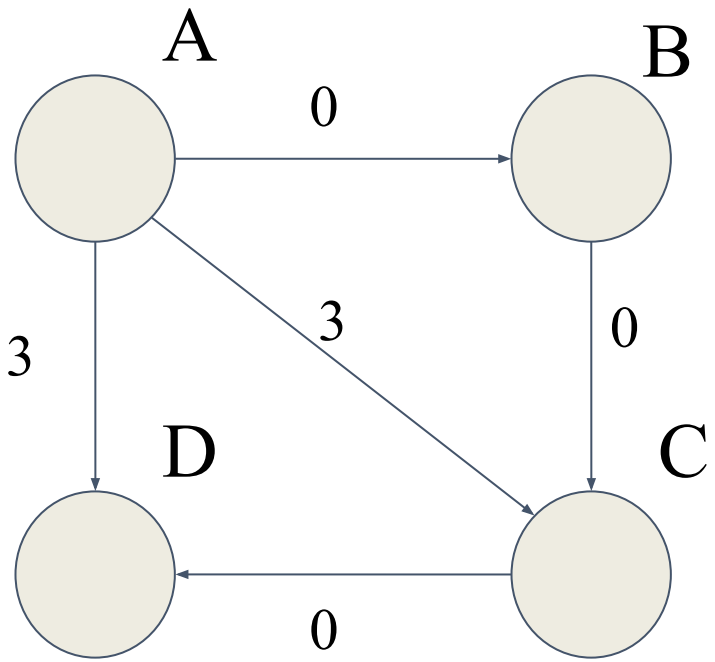
- Apply Bellman Ford Algorithm from s

Johnson Algorithm: Example



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

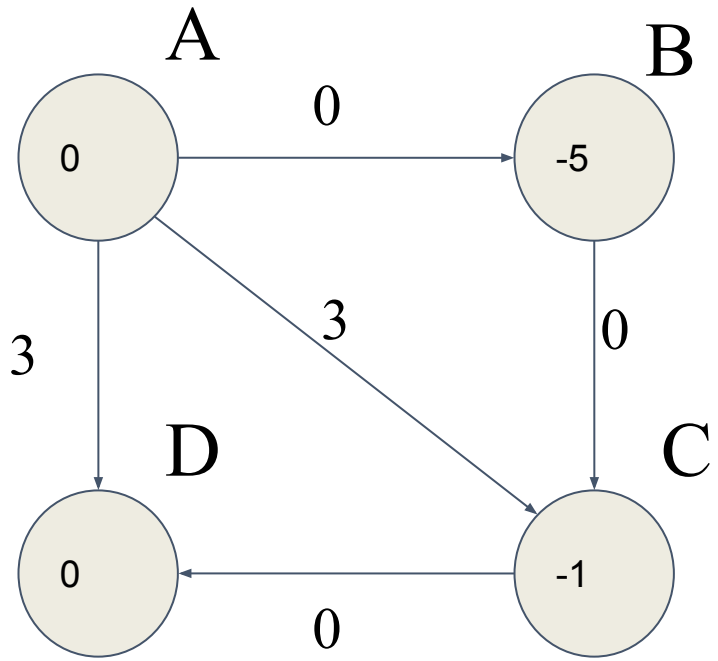
Johnson algorithm: Example



- Run Dijkstra's algorithm from every vertex

	A	B	C	D
A	0	0	0	0
B	inf	0	0	0
C	inf	inf	0	0
D	inf	inf	inf	0

Johnson algorithm: Example



	A	B	C	D
A	0	0	0	0
B	inf	0	0	0
C	inf	inf	0	0
D	inf	inf	inf	0

- Run Dijkstra's algorithm from every vertex

$\delta'(u, v) = \delta(u, v) + h[v] - h[u]$

$\delta'(A,B)=\delta(A,B)+h[B]-h[A]=0-5-0=-5$

$\delta'(A,C)=\delta(A,C)+h[C]-h[A]=0-1-0=-1$

$\delta'(A,D)=\delta(A,D)+h[D]-h[A]=0-0-0=0$

$\delta'(B,C)=\delta(B,C)+h[C]-h[B]=0-1- -5=4$

$\delta'(B,D)=\delta(B,D)+h[D]-h[B]=0-0- -5=5$

$\delta'(C,D)=\delta(C,D)+h[D]-h[C]=0-0- -1=1$

	A	B	C	D
A	0	-5	-1	0
B	inf	0	4	5
C	inf	inf	0	1
D	inf	inf	inf	0

Problems: Arbitrage

Arbitrage - Near simultaneous purchase and sale of securities or foreign exchange in different markets in order to profit from price discrepancies.

Rs -> US$ -> EURO ->AUS $ ->Rs >>>> Any profit?

- Use of discrepancies in currency exchange rate.
- For the sake of simplicity, let's assume
  - there are no transaction costs
  - you can trade any currency amount in fractional quantities.
- Example
  - 1 U.S. dollar bought 0.82 Euro, 1 Euro bought 129.7 Japanese Yen,
  - 1 Japanese Yen bought bought 0.70 Indian Rupee,1 Indian Rupee bought 0.0135489 USD U.S. dollars.
  - Then, by converting currencies, a trader can start with 1 U.S. dollar
    - 0.82*129.7*0.7* 0.0135489 USD =1.042 US dollars, thus making a 4.2% profit.
- Weighted directed graphs can be represented as an adjacency matrix.

Arbitrage: Analysis

Weighted directed graphs can be represented as an adjacency matrix.

- Arbitrage opportunities arise when a cycle is determined such that the edge weights satisfy the following expression $w_1 * w_2 * w_3 * \ldots * w_n > 1$
  - The above constraint of finding the cycles is harder in graphs.

Let's take the logarithm on both sides, such that

- $\log(w_1) + \log(w_2) + \log(w_3) + \ldots + \log(w_n) > 0$

  Taking the negative log, this becomes
  $(-\log(w_1)) + (-\log(w_2)) + (-\log(w_3)) + \ldots + (-\log(w_n)) < 0$

If we can find a cycle of vertices such that the sum of their weights is negative, then there exists an opportunity for currency arbitrage.

Luckily, Bellman-Ford algorithm is a standard graph algorithm that can be used to easily detect negative weight cycles in $O(|V*E|)$ time.

Max Flow

Maximum amount of flow that the network would allow to flow from source to sink

- source generates or produces whatever is flowing in the network
- sink consumes whatever is flowing in the network

Application- Movement of

- ➢ supply chain management SCM
- ➢ force/warfare

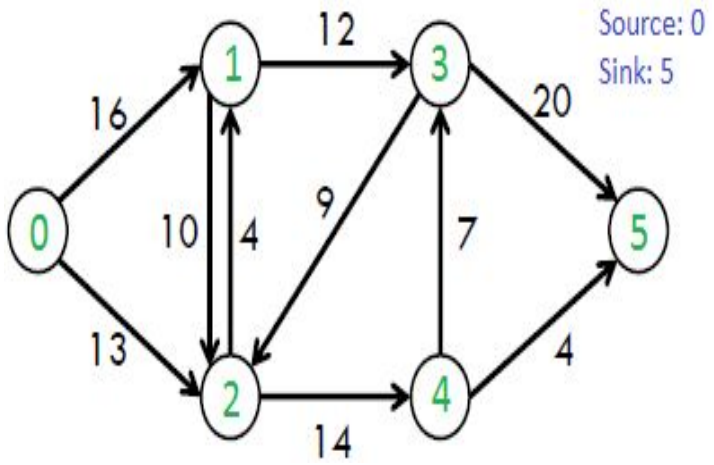
Ford–Fulkerson (FF) algorithm is a greedy algorithm that computes the maximum flow in a flow network.

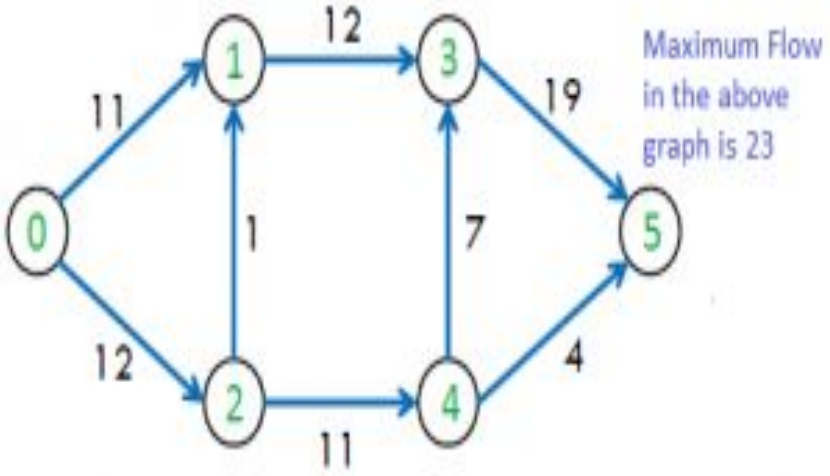
Max Flow: Example

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





Max Flow: Terminology

- Flow network
  - Directed graph
  - One source (s), sink(t)
  - Non negative weights edge, capacity c(u,v)
- Residual graph G_f
  - 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 the path p

Ford Fulkerson (FF) Algorithm

- $f_m = 0$
- While there exists augmenting path
  - Identify augmenting path
  - $C_f(p)$ = smallest capacity of p
  - new $f_m = f_m + C_f(p)$
  - 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: $O(E* f_m)$

Minimum Cut

- **Minimal Cut :** It is one in which replacement of any of its member/s 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