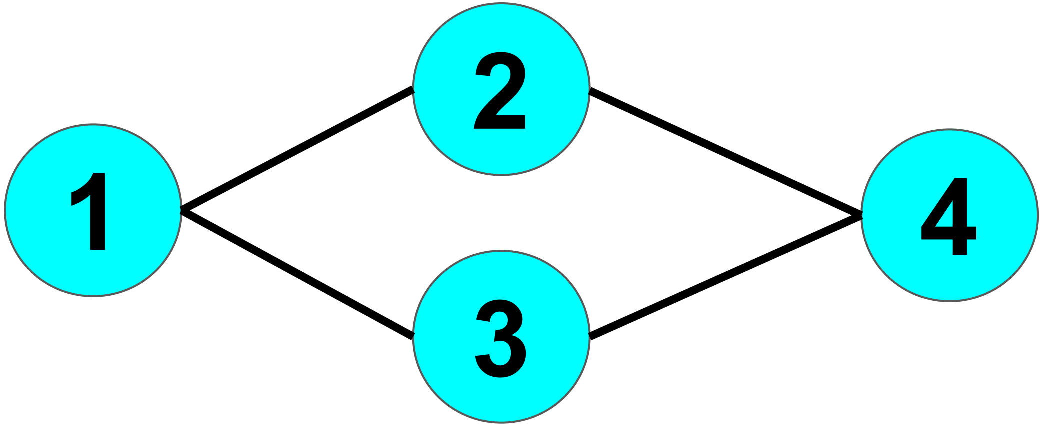
Exploring Betweenness Centrality

By: Shwetark Patel

Basic Terminology

- σ_{st} is the number of shortest paths from **s** to **t**
- $\sigma_{st}(v)$ is the number of shortest paths from **s** to **t** containing **v**

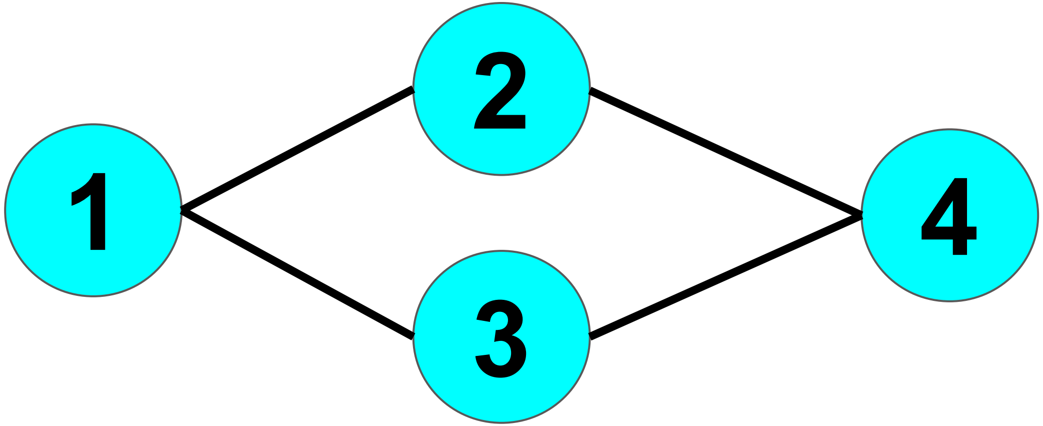


$\sigma_{14} = ?$

$\sigma_{14}(3) = ?$

Basic Terminology (Part 2)

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$$



$\delta_{14}(3) = ?$

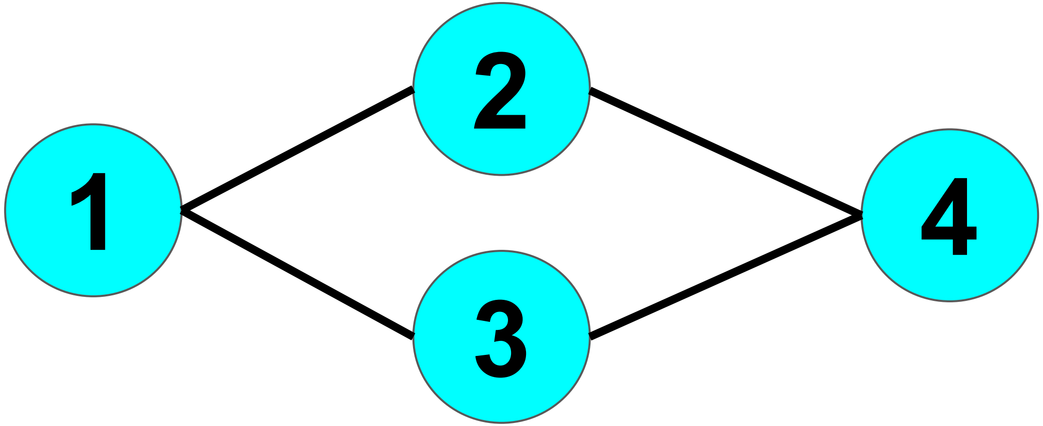
$\sigma_{14}(3) = 1$

$\sigma_{14} = 2$

$\delta_{14}(3) = 0.5$

Basic Terminology (Part 3)

$$C_B(v) = \sum_{s \neq v \neq t \in V} \delta_{st}(v).$$



$C_B(3) = ?$

$\delta_{14}(3) = 0.5$

$\delta_{12}(3) = 0$

$\delta_{24}(3) = 0$

$C_B(3) = 0.5$

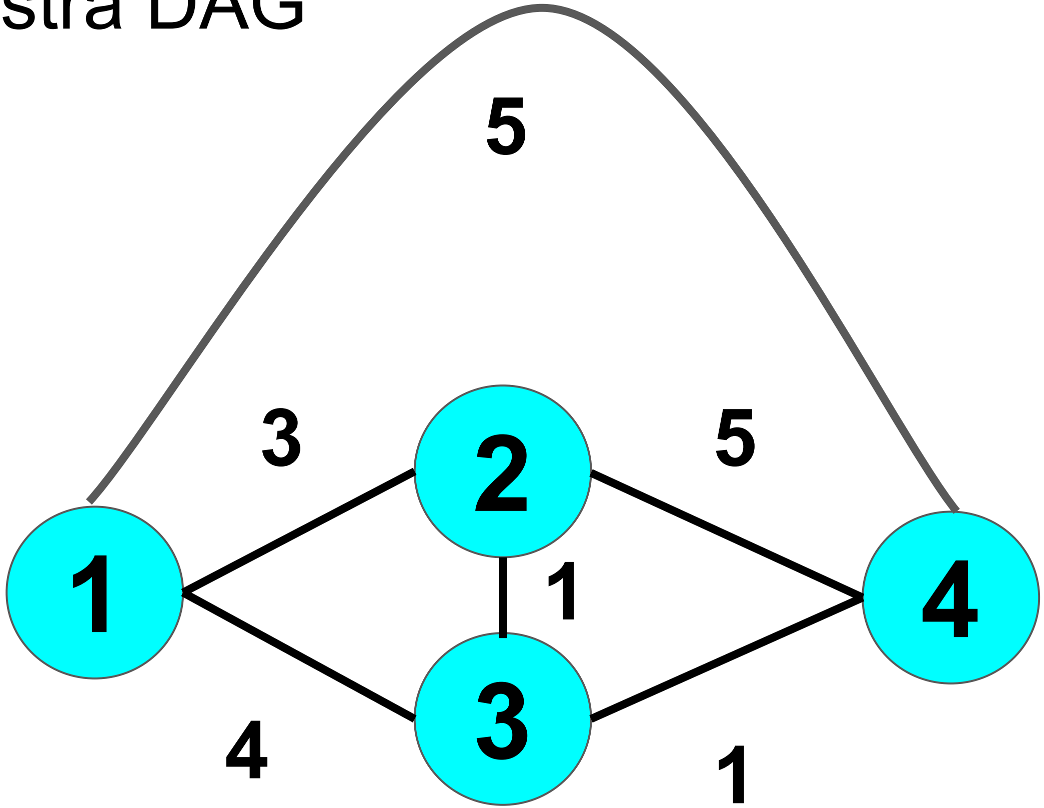
Problem

- Calculate betweenness centrality for each vertex
- Assumptions: Graph is undirected and connected
- Previous algorithm: $O(n^3)$ time and $O(n^2)$ space
- New algorithm: $O(nm)$ time for unweighted graphs and $O(nm + n^2\log(n))$ time on weighted graphs
- New algorithm: $O(n + m)$ space

Step 1

- Need to calculate σ_{st}
- Run Djikstra’s shortest path algorithm starting from **s.** Let **dist**[i] be the shortest path distance from **s** to i.
- Create the “Djikstra DAG”
  - Connect two vertices **u** and **v** with a directed edge if **dist**[u] + W[u, v] = **dist**[v]

Djikstra DAG



Djikstra starting at node 1

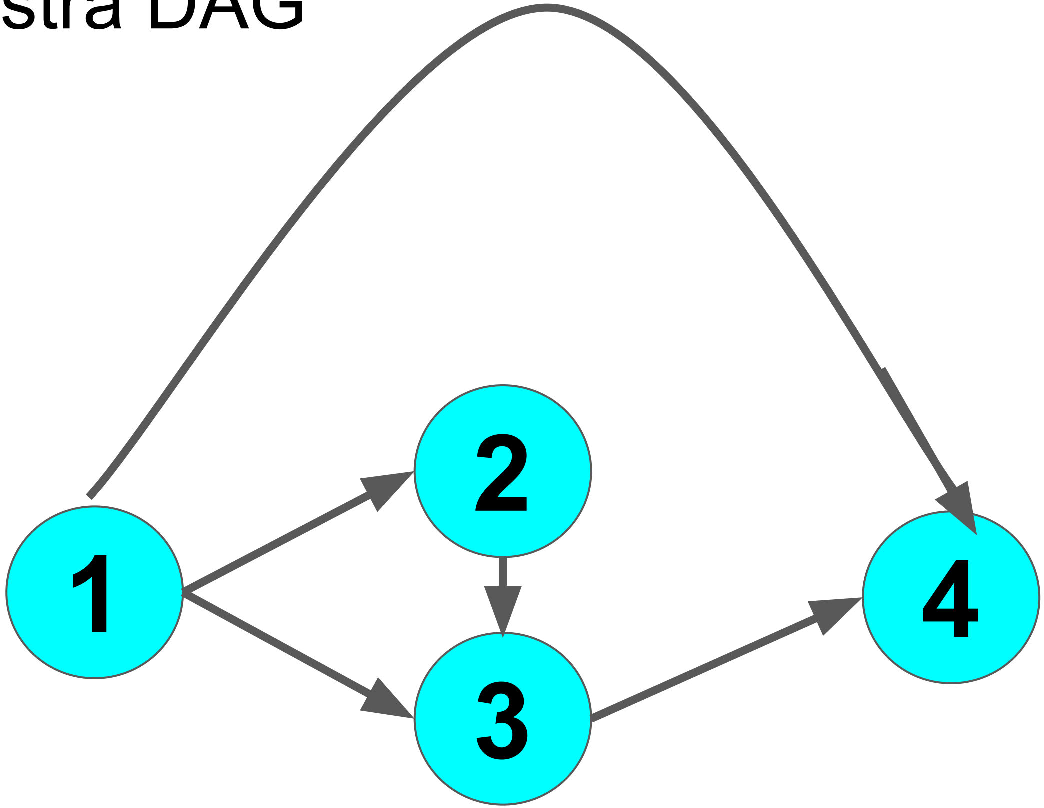
dist[1] = 0

dist[2] = 3

dist[3] = 4

dist[4] = 5

Djikstra DAG



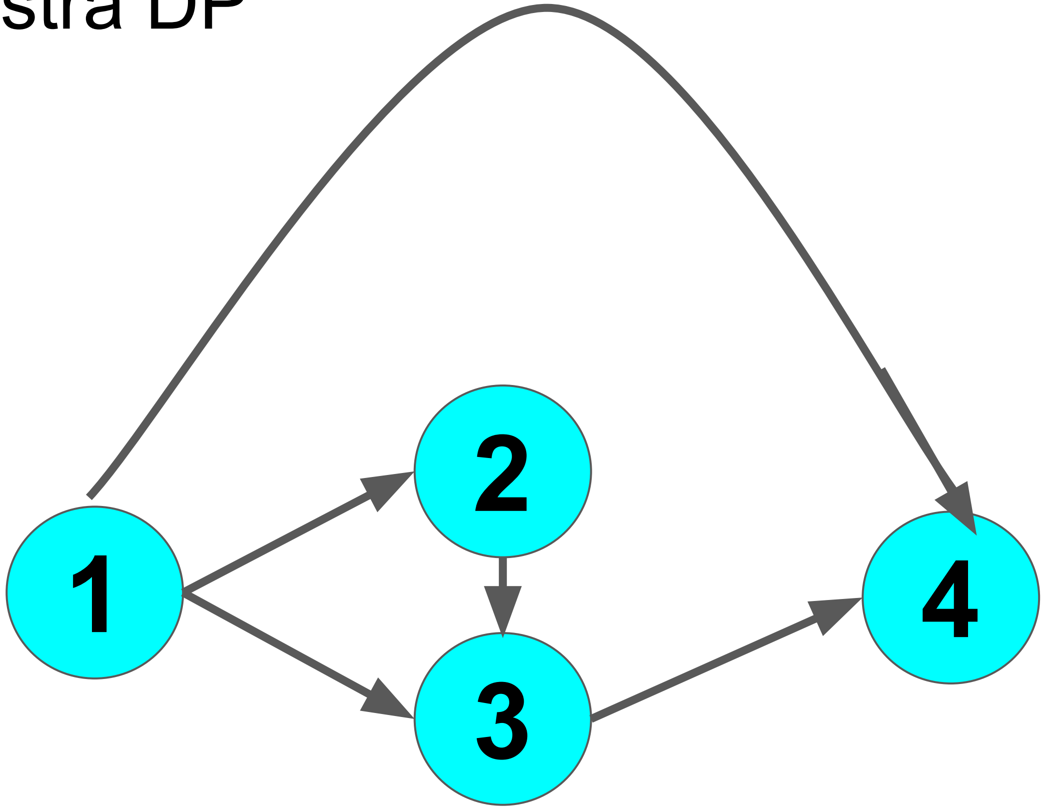
Djikstra DAG

- Any path along the Djikstra DAG is a shortest path
- All shortest paths are paths on the Djikstra DAG
- We can use dynamic programming along the DAG to determine number of shortest paths to any node from the source

Djikstra DAG Dynamic Programming

- Let dp[v] represent the number of shortest paths to v from the source
- Let P be the set of all nodes that have a directed edge in the Djikstra DAG to a node **v**
- Base case is **dp**[source] = 1
- **dp**[v] = $\sum_{u \in P} dp[u]$
- Calculate **dp** values in topological order

Djikstra DP



dp[1] = 1

dp[2] = 1

dp[3] = dp[1] + dp[2]
= 2

dp[4] = dp[1] + dp[4]
= 3

Step 1 Recap

- We can now compute σ_{st} for all pairs (s, t) through starting Djikstra's algorithm from all vertices
- Running Djikstra once takes $O(V\log(V) + E)$ time
- Running Djikstra V times takes $O(V^2\log(V) + VE)$ time
- Complexity of our algorithm so far is $O(V^2\log(V) + VE)$ for weighted graphs and $O(VE)$ for unweighted graphs (since we can just use BFS instead of Djikstra)

Next steps

- From here, if we store our σ_{st} values in an array and then naively compute all $\delta_{st}(v)$ values, our algorithm still takes $O(V^3)$ time and $O(V^2)$ space
- Smarter method: Another DP along the Djikstra DAG
- In order to compute $C_B(V)$ for each vertex, let's first solve the simple case where the Djikstra DAG is a tree

Step 2

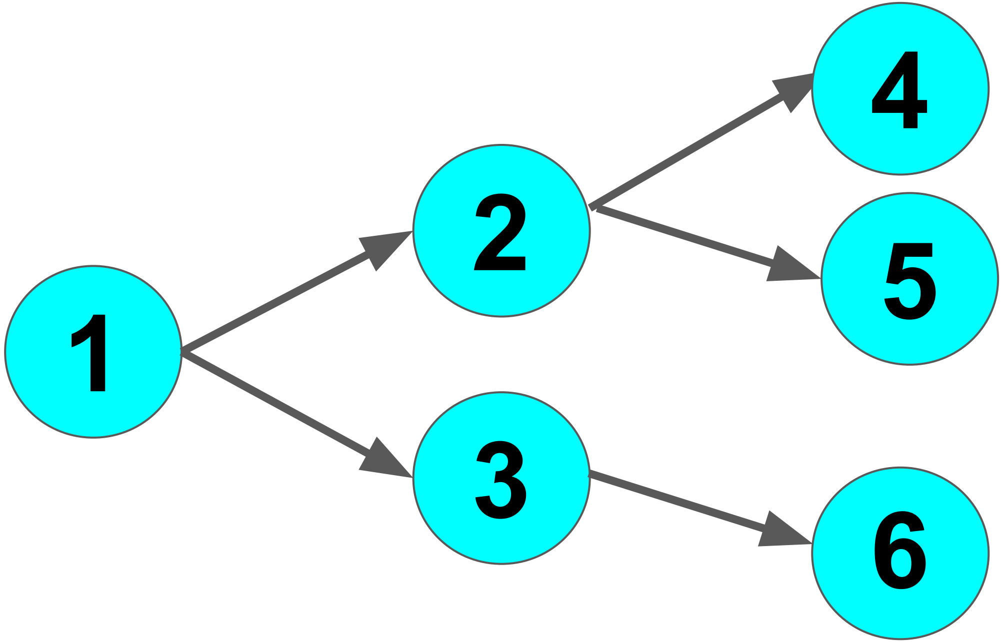
- $$\delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}(v).$$
- If we can compute these values quickly then we’d be set, since these lead directly to betweenness centrality
- Let’s see how to quickly compute these values if the Djikstra DAG is a tree

Case where Djikstra DAG is tree

- We can use DAG dynamic programming again, given a fixed source **s** as well as the Djikstra DAG
- Let **dp**[v] = $\delta_{s\bullet}(v)$
- If **R** is the set of vertices that **v** has an outgoing edge to, then:

$$dp[v] = \sum_{t \in R} 1 + dp[t]$$

- Calculate dp values in reverse topological order



dp[4] = 0

dp[5] = 0

dp[6] = 0

dp[2] = (1 + dp[4]) +
(1 + dp[5])
= 2

dp[3] = (1 + dp[6])
= 1

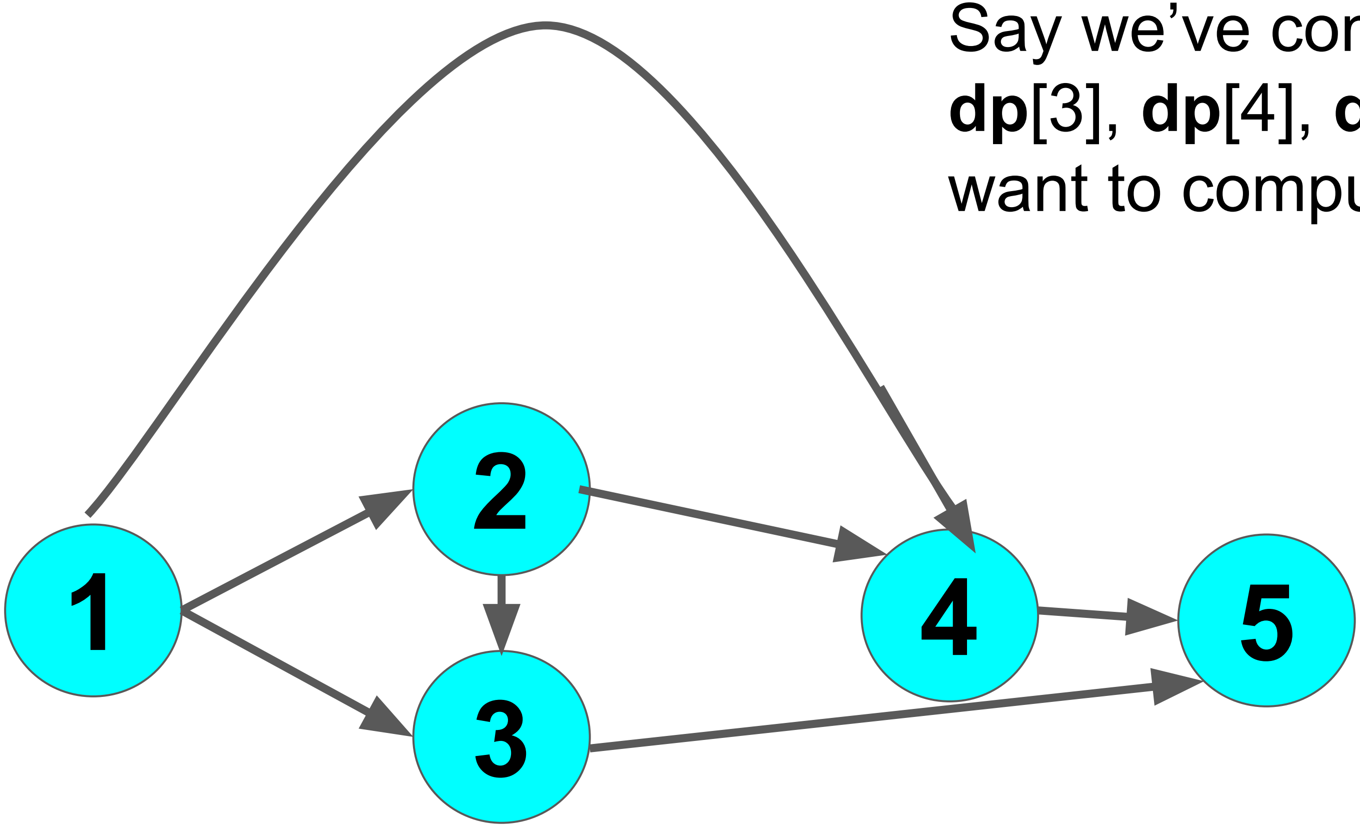
dp[1] = (1 + dp[2]) +
(1 + dp[3])
= 5

General case for Djikstra DAG

- Let **dp**[v] = $\delta_{s\cdot}(v)$
- If **R** is the set of vertices that **v** has an outgoing edge to, then:

$$dp[v] = \sum_{t \in R} (1 + dp[t]) \times \frac{\sigma_{sv}}{\sigma_{st}}$$

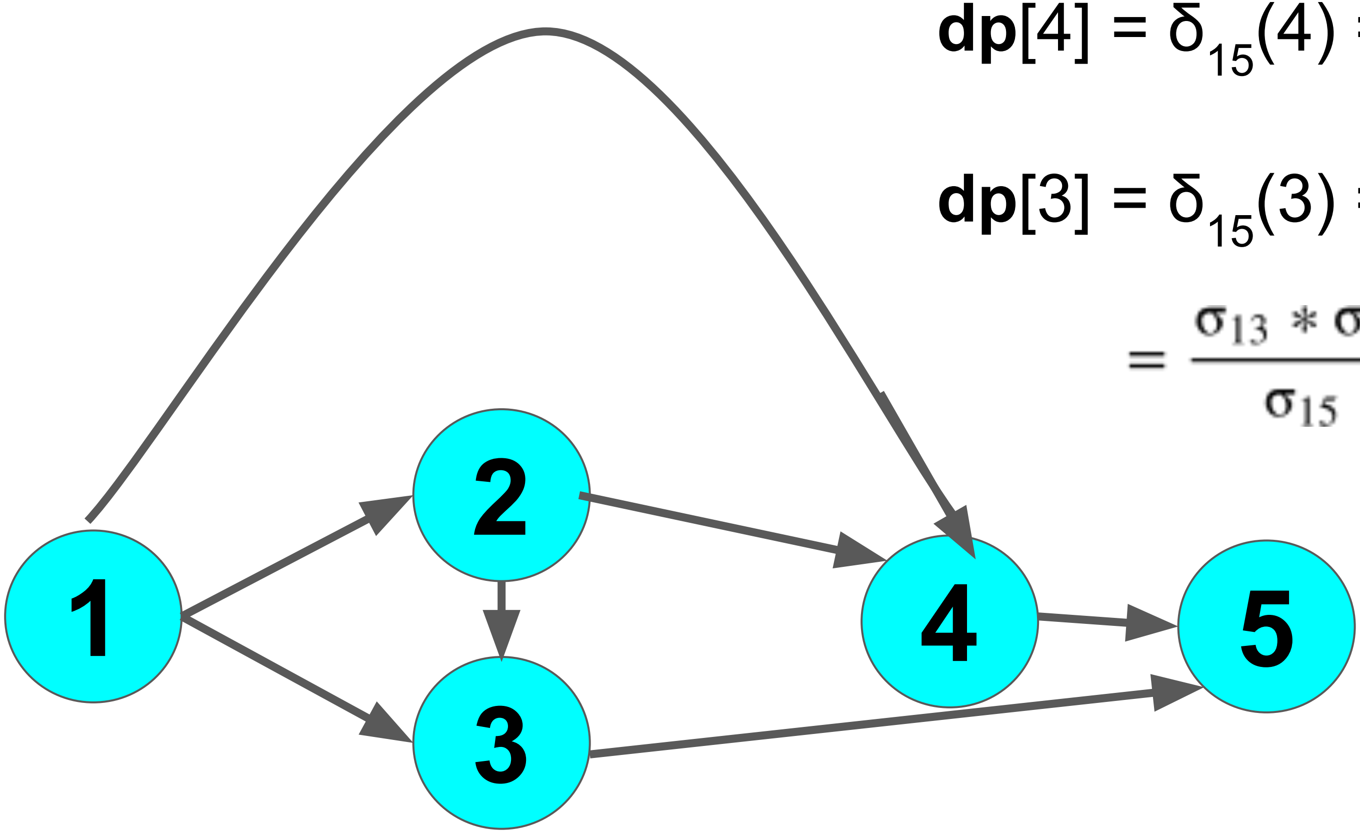
- Calculate dp values in reverse topological order



Say we've computed **dp**[3], **dp**[4], **dp**[5] and want to compute **dp**[2]

$$\mathbf{dp}[4] = \delta_{15}(4) = \frac{\sigma_{15}(4)}{\sigma_{15}}$$

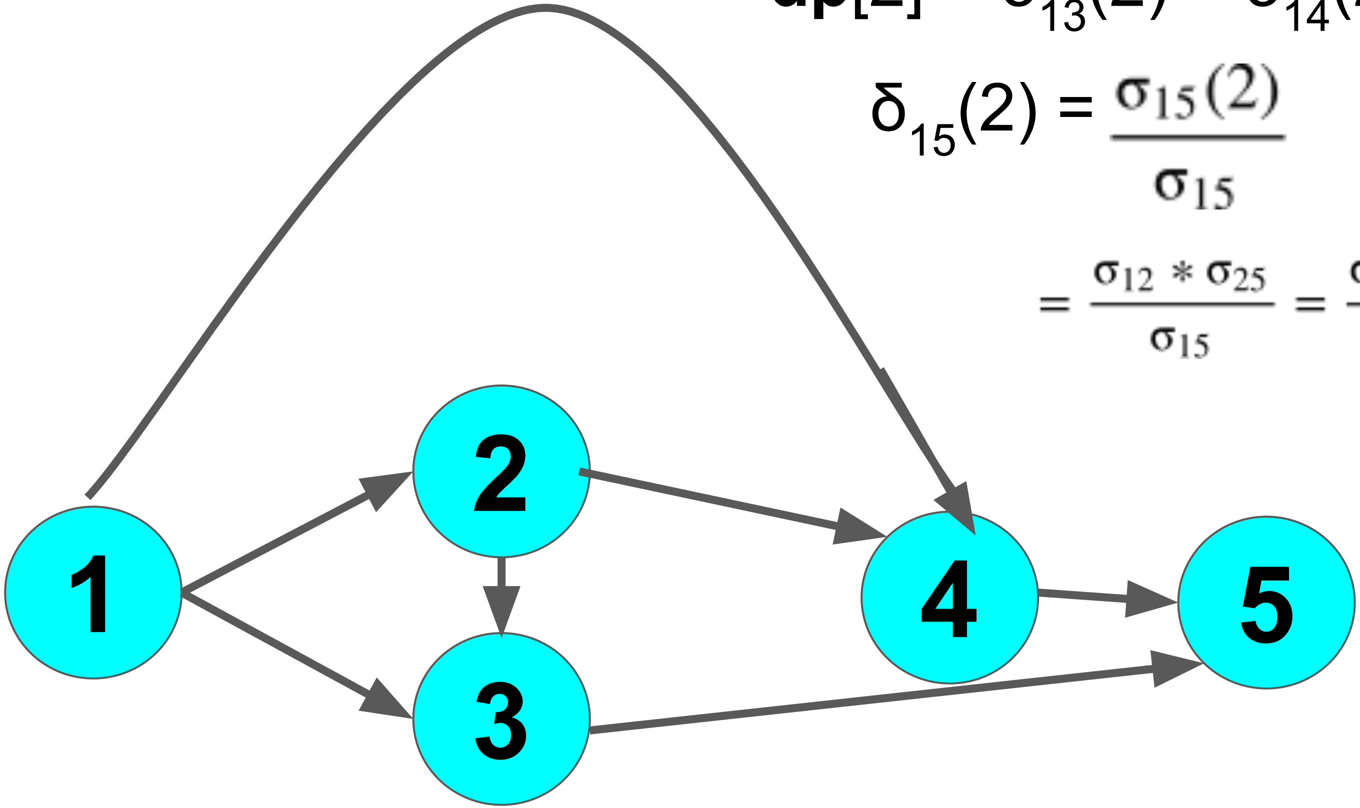
$$\mathbf{dp}[3] = \delta_{15}(3) = \frac{\sigma_{15}(3)}{\sigma_{15}} = \frac{\sigma_{13} * \sigma_{35}}{\sigma_{15}}$$



$$\mathbf{dp}[2] = \delta_{13}(2) + \delta_{14}(2) + \delta_{15}(2)$$

$$\delta_{15}(2) = \frac{\sigma_{15}(2)}{\sigma_{15}}$$

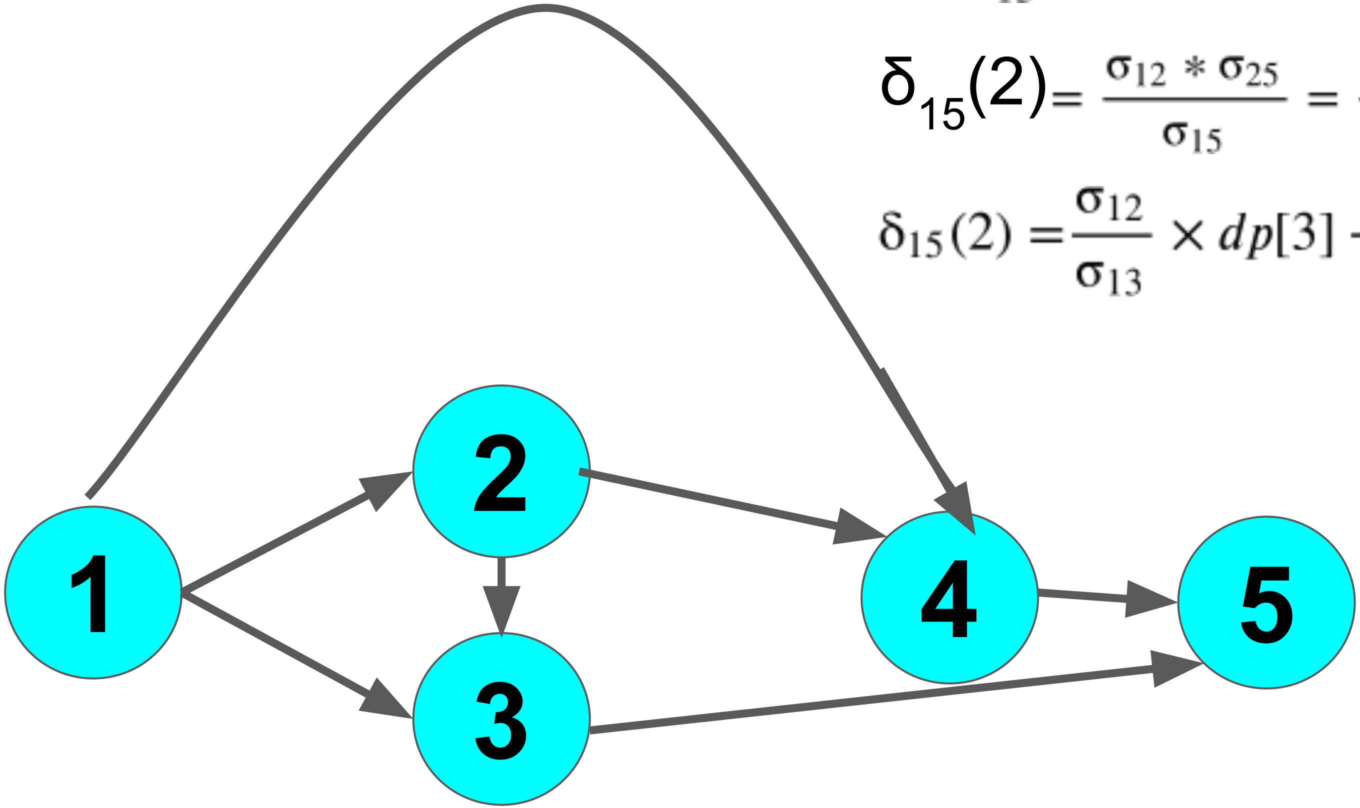
$$= \frac{\sigma_{12} * \sigma_{25}}{\sigma_{15}} = \frac{\sigma_{12} * (\sigma_{35} + \sigma_{45})}{\sigma_{15}}$$



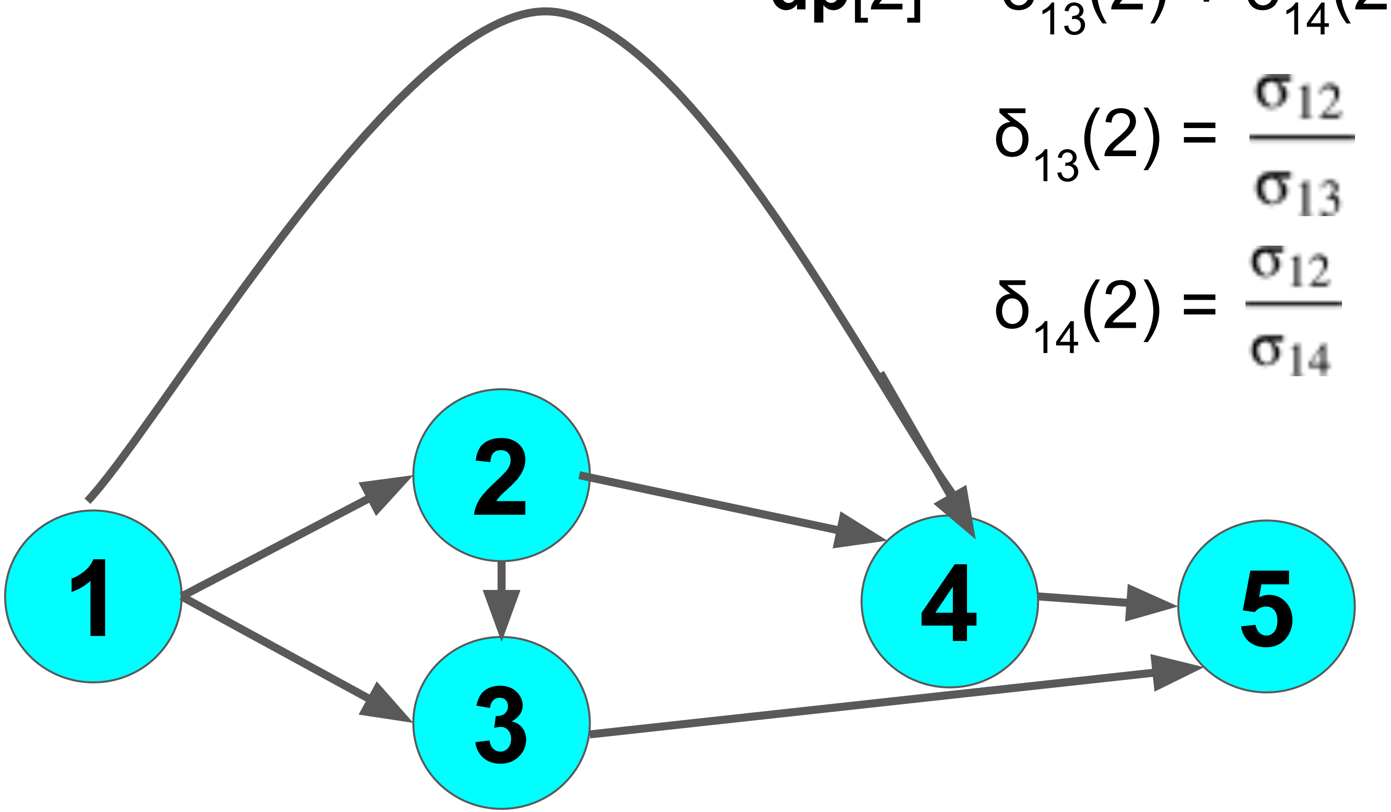
$$\mathbf{dp}[3] = \frac{\sigma_{13} * \sigma_{35}}{\sigma_{15}} \qquad \mathbf{dp}[4] = \frac{\sigma_{14} * \sigma_{45}}{\sigma_{15}}$$

$$\delta_{15}(2) = \frac{\sigma_{12} * \sigma_{25}}{\sigma_{15}} = \frac{\sigma_{12} * (\sigma_{35} + \sigma_{45})}{\sigma_{15}}$$

$$\delta_{15}(2) = \frac{\sigma_{12}}{\sigma_{13}} \times dp[3] + \frac{\sigma_{12}}{\sigma_{14}} \times dp[4]$$



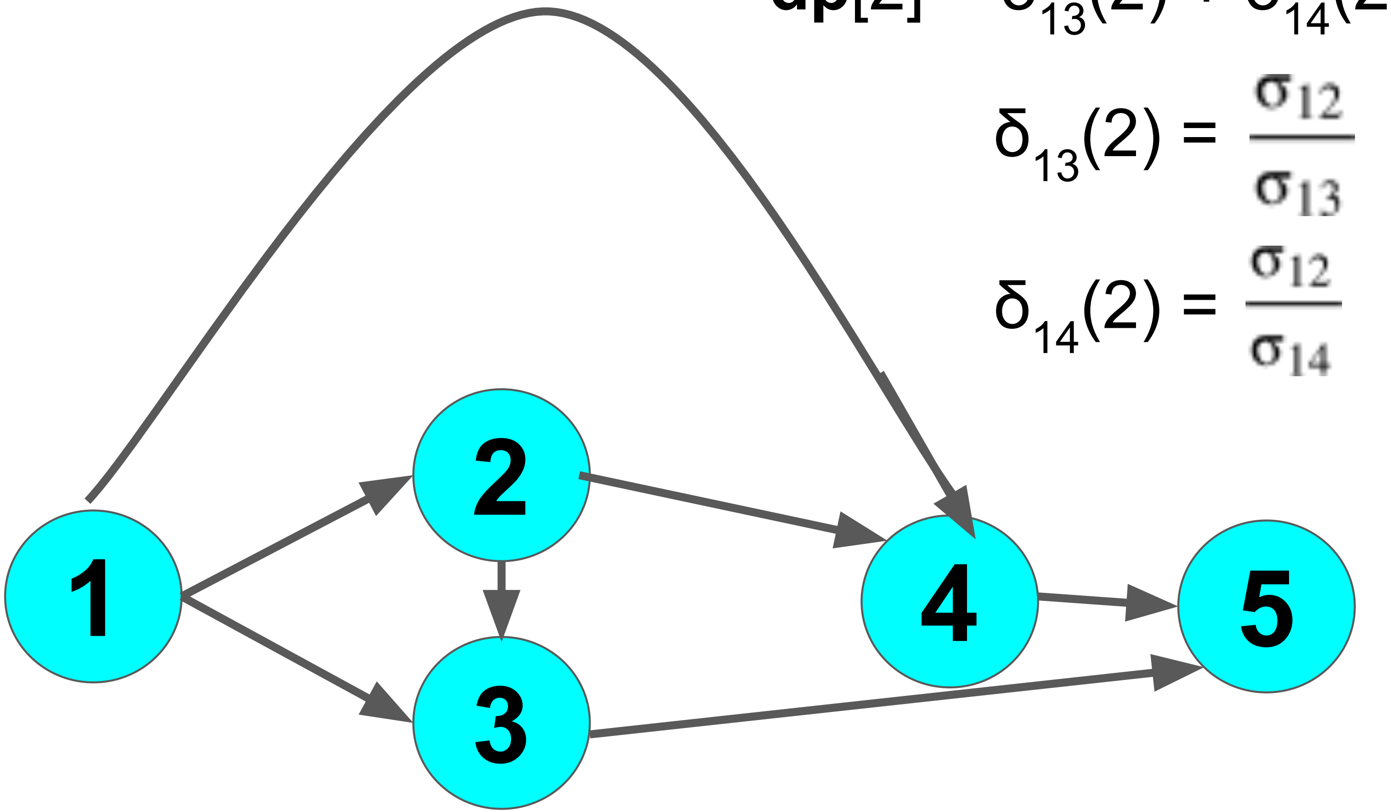
$$dp[v] = \sum_{t \in R} (1 + dp[t]) \times \frac{\sigma_{sv}}{\sigma_{st}}$$



$$\mathbf{dp}[2] = \delta_{13}(2) + \delta_{14}(2) + \delta_{15}(2)$$

$$\delta_{13}(2) = \frac{\sigma_{12}}{\sigma_{13}}$$

$$\delta_{14}(2) = \frac{\sigma_{12}}{\sigma_{14}}$$



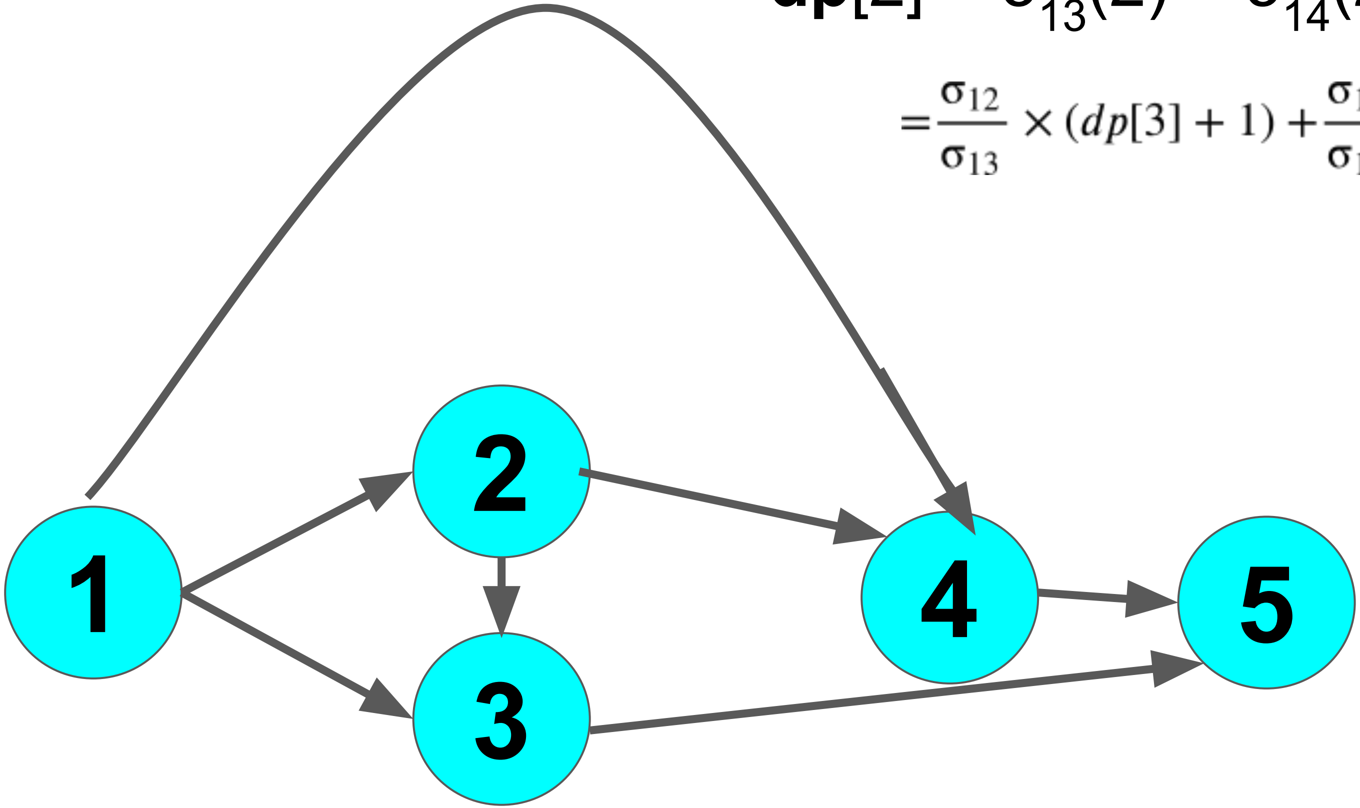
$$\textbf{dp}[2] = \delta_{13}(2) + \delta_{14}(2) + \delta_{15}(2)$$

$$\delta_{13}(2) = \frac{\sigma_{12}}{\sigma_{13}}$$

$$\delta_{14}(2) = \frac{\sigma_{12}}{\sigma_{14}}$$

$$\mathbf{dp}[2] = \delta_{13}(2) + \delta_{14}(2) + \delta_{15}(2)$$

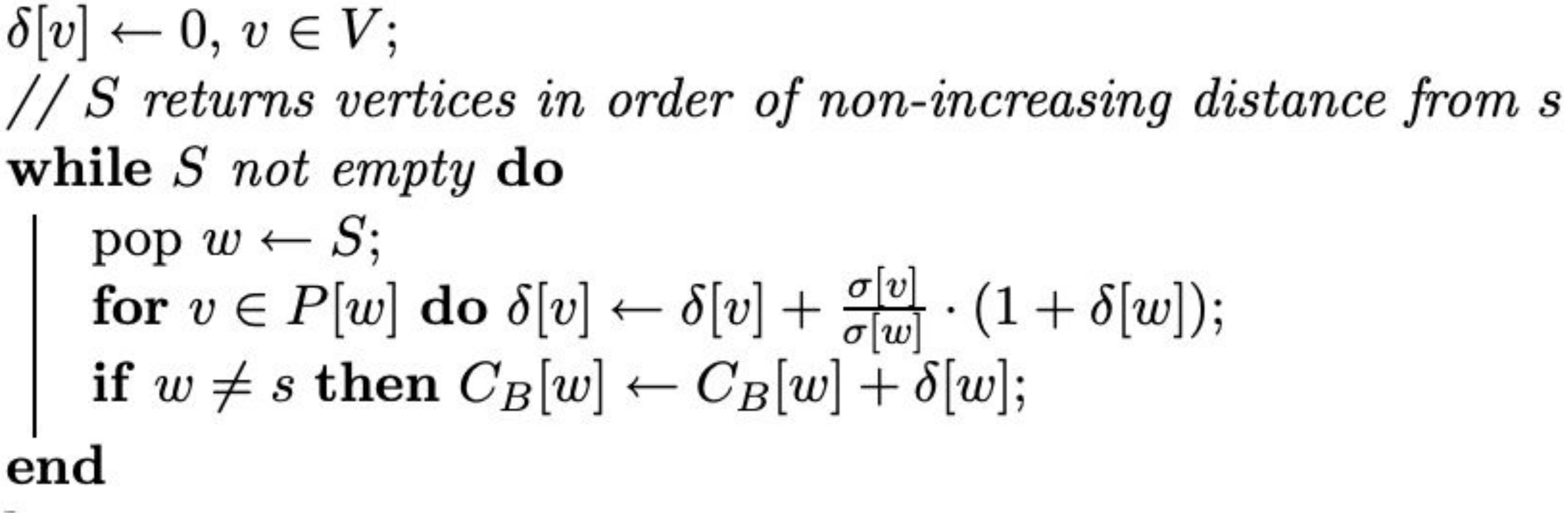
$$= \frac{\sigma_{12}}{\sigma_{13}} \times (dp[3] + 1) + \frac{\sigma_{12}}{\sigma_{14}} \times (dp[4] + 1)$$



Wrapping up

- Can easily compute betweenness centrality for each vertex **v** now by adding up **dp**[v] over all sources (except when **v** is the source).
- Should divide all betweenness centrality values by 2 at the end
- Space required is $O(V + E)$ and time required is $O(V^2\log(V) + VE)$ for weighted graphs and $O(VE)$ for unweighted graphs.

Wrapping up

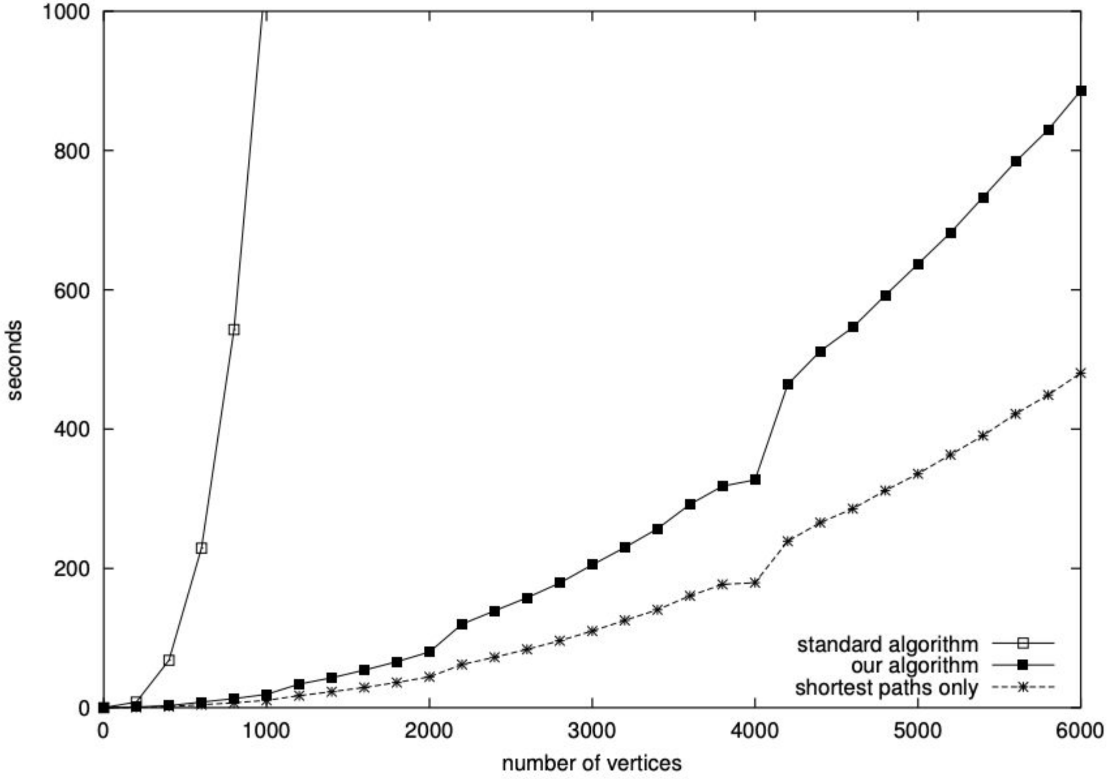


```
$\delta[v] \leftarrow 0,\ v \in V$;
// S returns vertices in order of non-increasing distance from s
while $S$ not empty do
    pop $w \leftarrow S$;
    for $v \in P[w]$ do $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]} \cdot (1 + \delta[w])$;
    if $w \neq s$ then $C_B[w] \leftarrow C_B[w] + \delta[w]$;
end
```

Experimental Results

- Groundbreaking algorithm -- Led to significant speedups

Experimental Results



Importance

- Betweenness centrality is a very important metric for a network
- This algorithm significantly improved existing methods, which had to go through all triples (s, t, v) to compute $\delta_{st}(v)$

Discussion

- Combining the results of this paper and "Direction-Optimizing Breadth First Search" on social networks
- Thinking about time complexities more
  - Previous: $O(n^2)$ space and $O(n^3)$ time
  - Now: $O(n + m)$ space and $O(nm)$ time