Shortest Paths Revisited 4/4
All-Pairs Shortest Paths

Lecture 07.09 by *Marina Barsky*

Johnson’s algorithm

Results: All-Pairs Shortest Paths

For sparse graphs **with non-negative edges**: use n*Dijkstra

The best!

1. **Graphs with non-negative edge costs**:

n*Dijkstra (m log n) = O(nm log n) = $\begin{cases} \mathbf{O(n^2 \log n)} & \text{if } m=O(n) \text{ [sparse]} \\ O(n^3 \log n) & \text{if } m=O(n^2) \text{ [dense]} \end{cases}$

2. **General graphs:**

n*Bellman-Ford (nm) = $O(n^2m)$ = $\begin{cases} O(n^3) & \text{if } m=O(n) \text{ [sparse]} \\ O(n^4) & \text{if } m=O(n^2) \text{ [dense]} \end{cases}$

1*Floyd-Warshall: $\mathbf{O(n^3)}$

Can we do better for generic (sparse) graphs?

Motivation

- APSP = n*SSSP
- n*Dijkstra's algorithm = O(nm log n)
  for sparse graphs: **$O(n^2 \log n)$** **We want this complexity! But for general sparse graphs**

- **Idea:** use n*Dijkstra for general graphs
- **Obstacle:** we need to get rid of negative edge costs

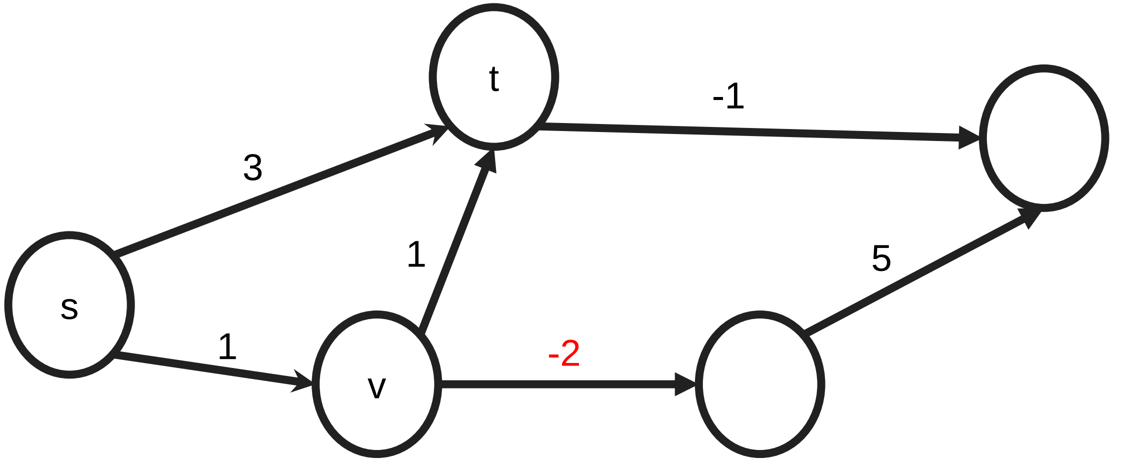
Johnson's algorithm

- Invoke Bellman-Ford SSSP once: O(nm)
- Use n times Dijkstra: O(nm log n)
- Total running time: O(nm log n) **For general graphs!**

This will transform G into the graph with non-negative edge weights

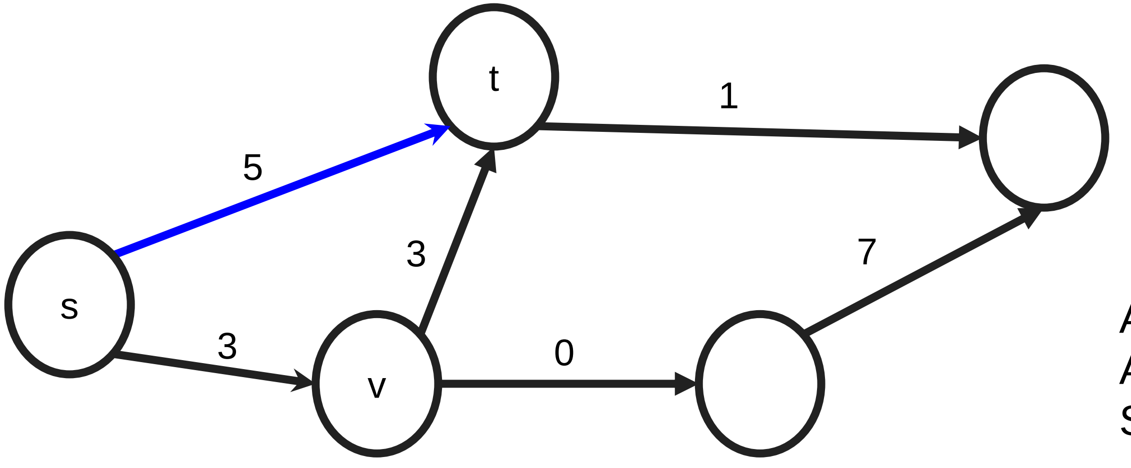
Reweighting technique which does not work

- Natural instinct: add max negative cost to the weight of each edge, making all edges non-negative



Most negative m=-2

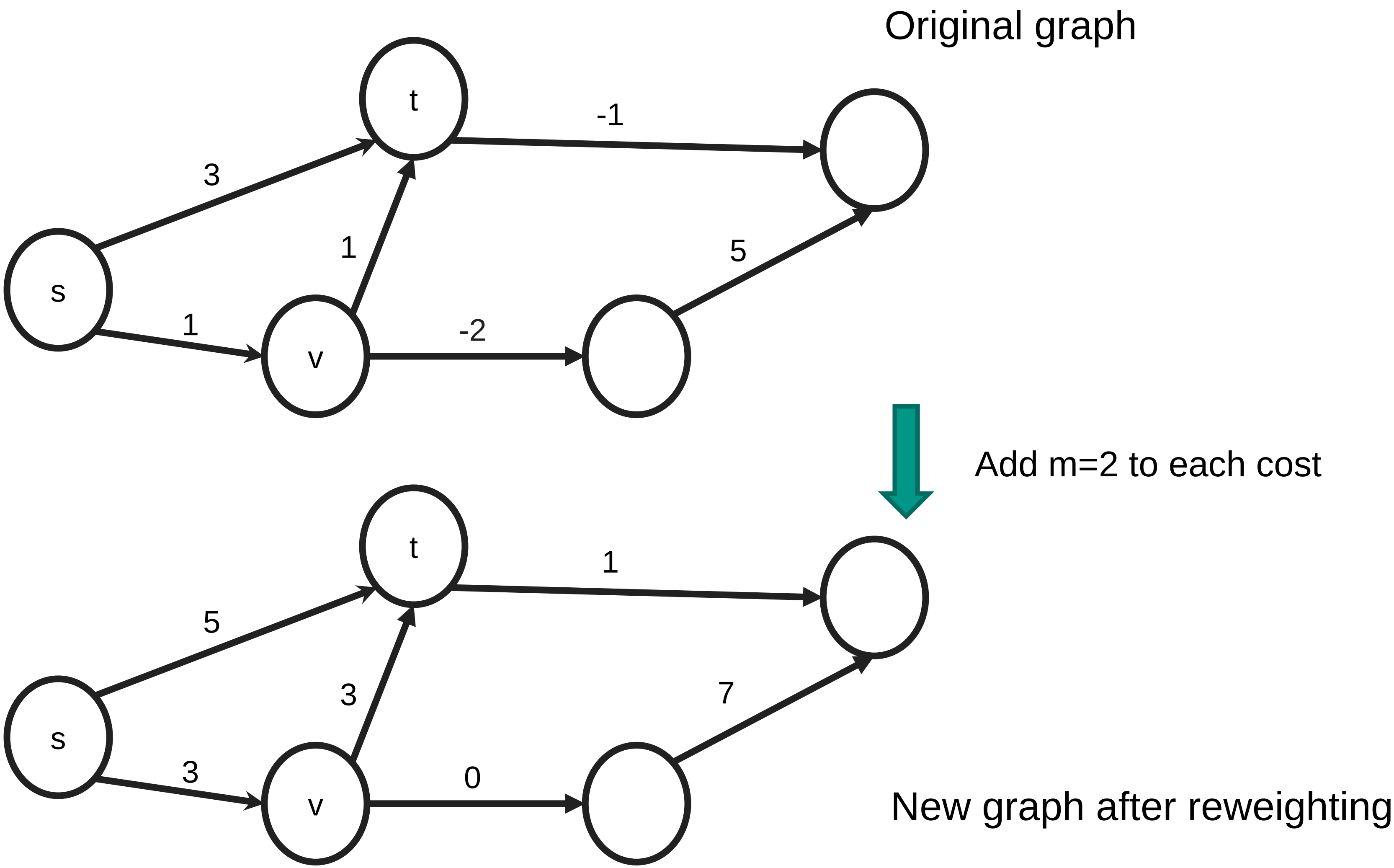
Add m=2 to each cost



Add -m to each edge weight.
After reweighting:
Shortest path s~>t is s-t!

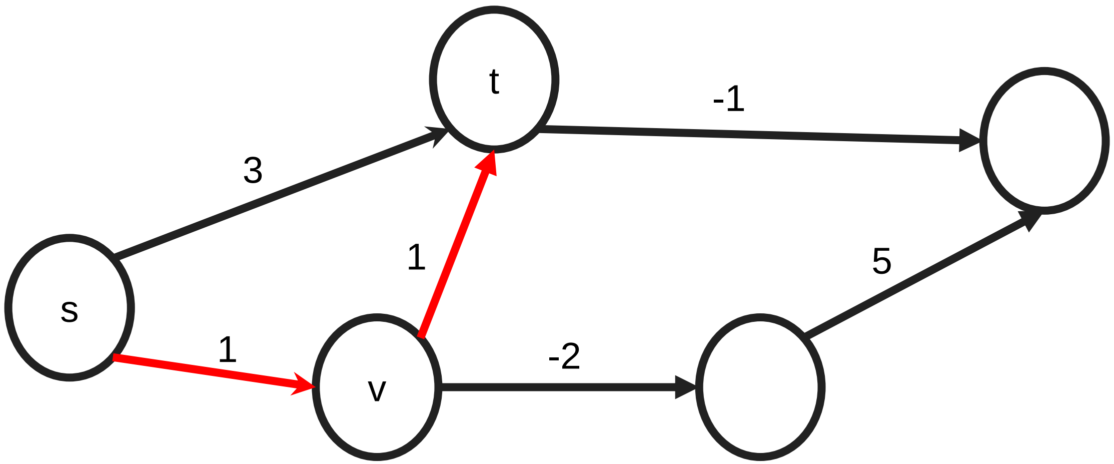
Reweighting technique which does not work

- Natural instinct: add max negative cost to the weight of each edge, making all edges non-negative

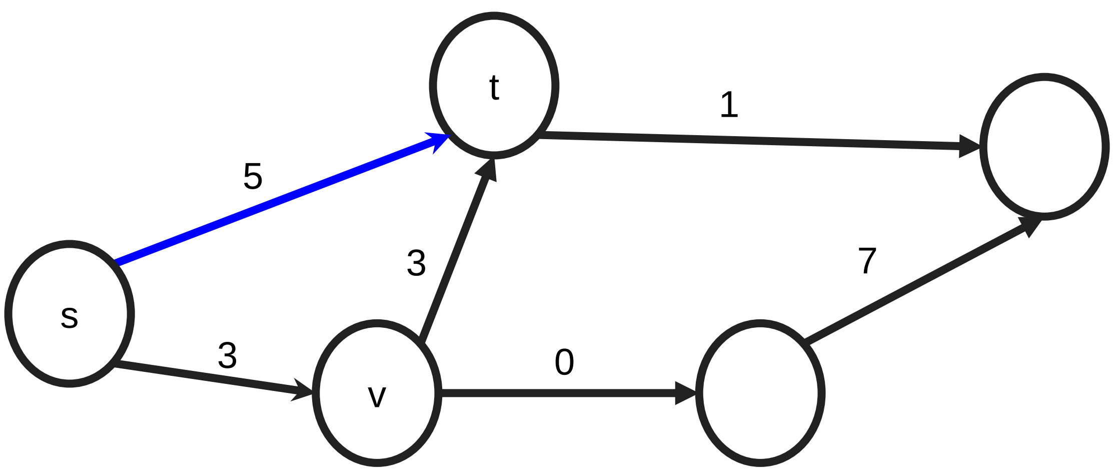


Reweighting technique which does not work

- However this does not preserve the original shortest paths!



Before reweighting:
Shortest path s~>t is s-v-t

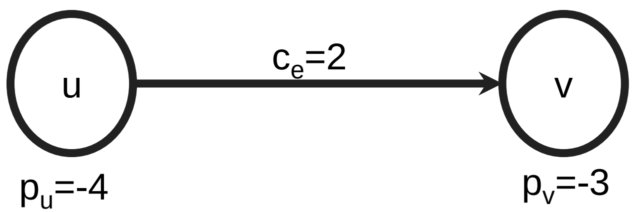


After reweighting:
Shortest path s~>t is s-t!

Reweighting idea: vertex tokens

- Let G=(V,E) be a directed graph with general edge lengths (including negative)
- Fix a token p_v for each vertex v ∈ V (any real number)

- Transform the cost c_e of every edge e=(u,v) to $c_e' = c_e + p_u - p_v$



$c_e' = 2 + (-4) - (-3) = 1$

- Then the cost of any path P with original length L between two vertices s,t in G will be modified by exactly the same amount:

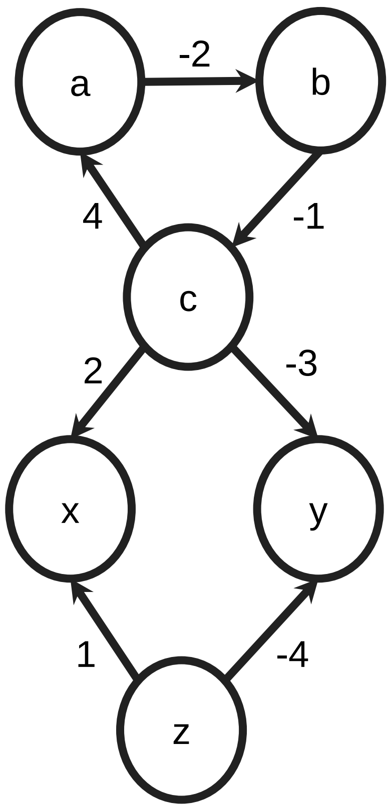
$$L' = L + p_u - p_v$$

$$L' = \sum_{all\ (u,v) \in P} [c_e + p_u - p_v]$$

The tokens of all intermediate nodes cancel themselves and leave only the tokens of the source and the destination vertices

- Thus the relative lengths of different paths between s and t remain the same

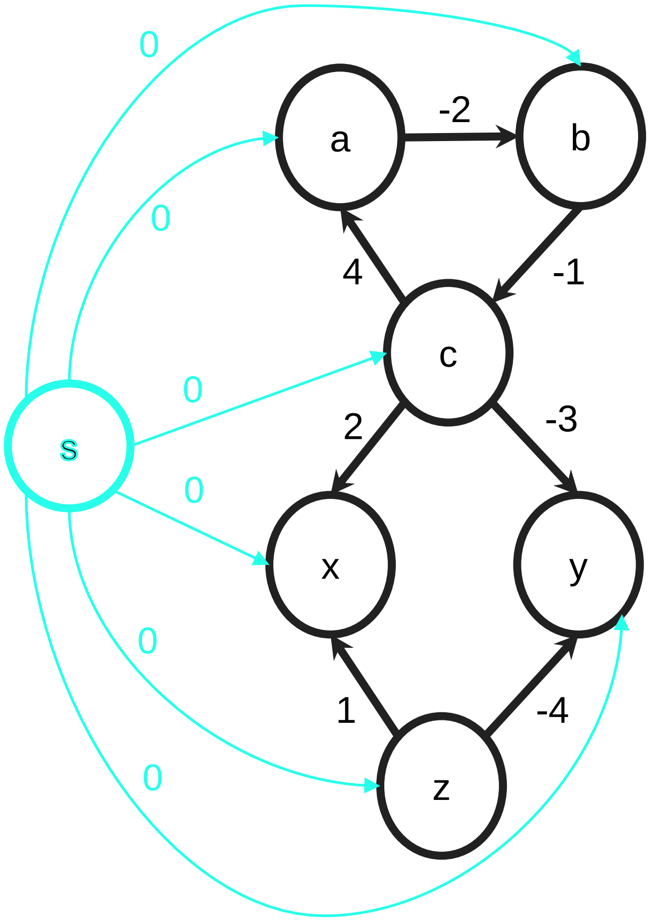
Computing magical vertex tokens



Sample graph with negative edge lengths but without negative cycles

- Compute magical vertex tokens running SSSP Bellman-Ford algorithm once

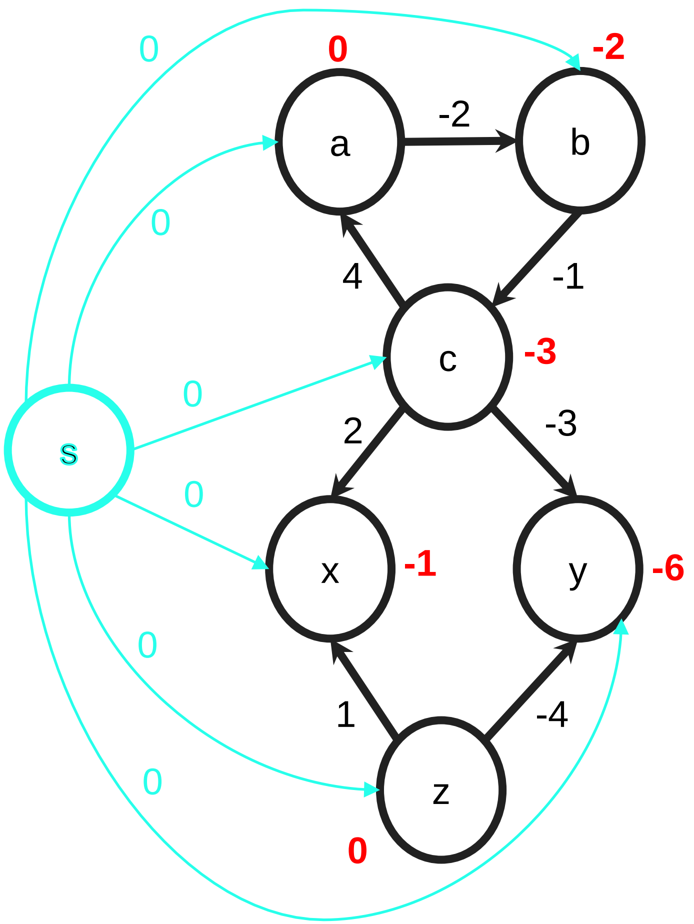
Computing magical vertex tokens



Adding artificial source vertex s with edges of cost 0 to every vertex in G

- Compute magical vertex tokens running SSSP Bellman-Ford algorithm once
- Add artificial source vertex s which has an outgoing edge of cost 0 to every vertex in G. Adding s will not change any shortest paths between original vertices of G, because s has no incoming edges (no path in the original graph can go through s)

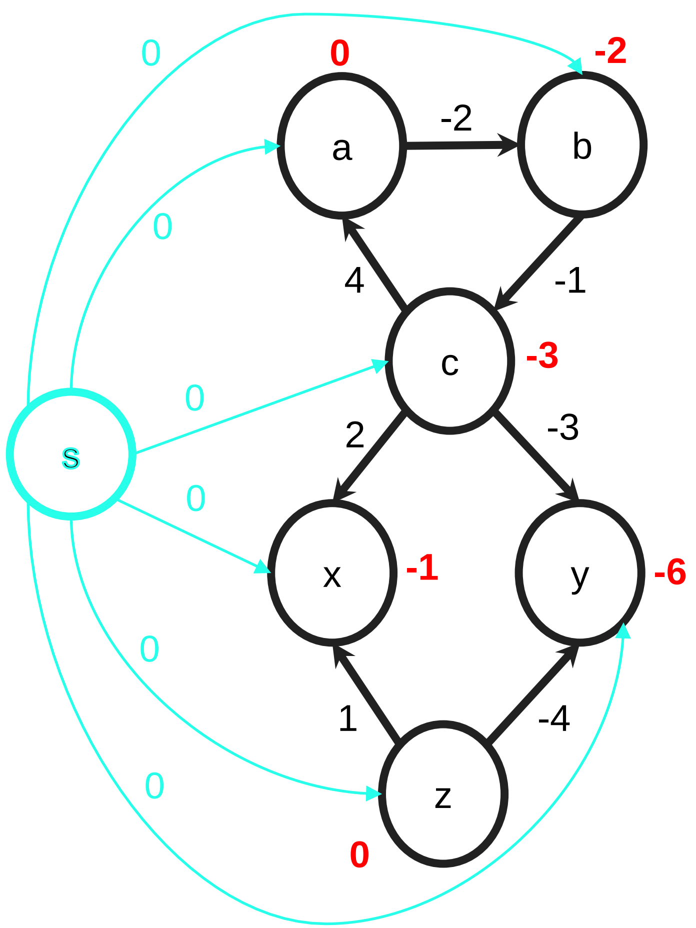
Computing magical vertex tokens



For each vertex: costs of single-source shortest paths from s

- Compute magical vertex tokens running SSSP Bellman-Ford algorithm once
- Add artificial source vertex s which has an outgoing edge of cost 0 to every vertex in G
- Run Bellman-Ford and compute the costs of shortest paths from s to every other vertex

Computing magical vertex tokens



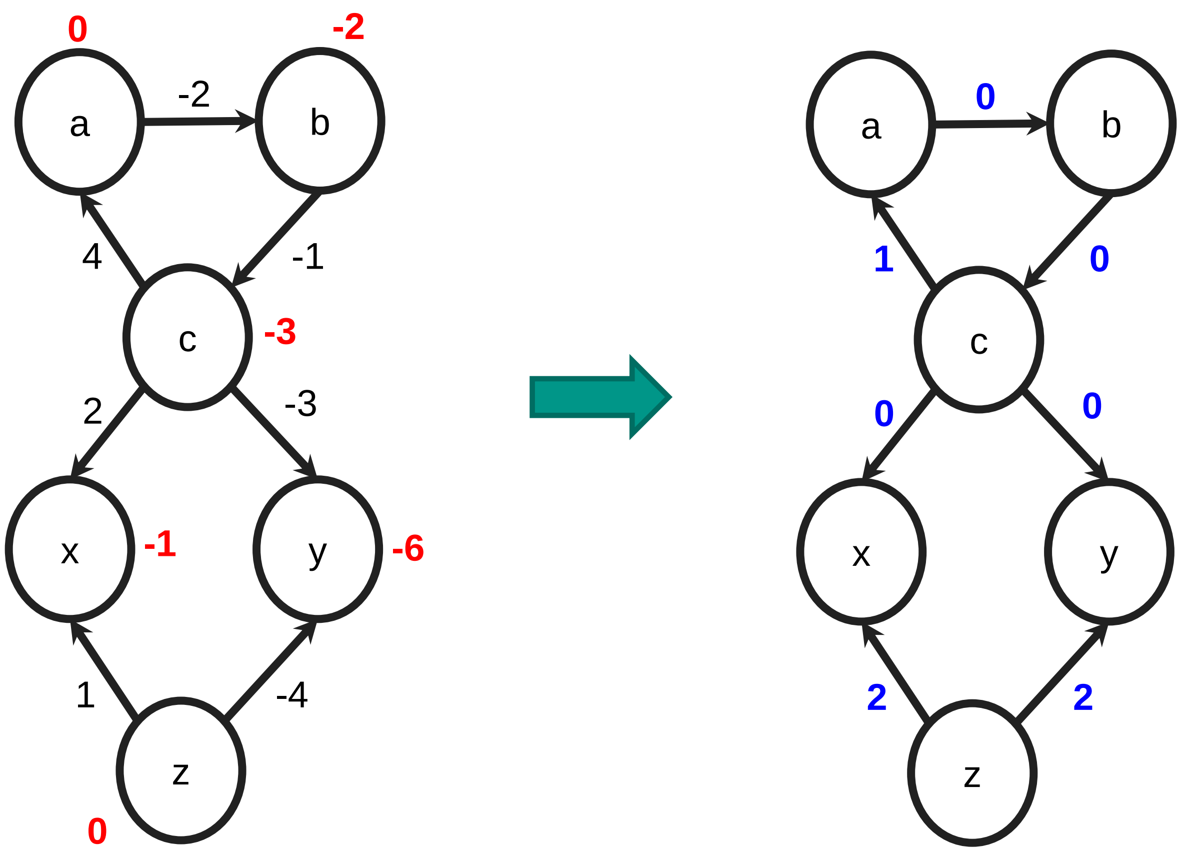
For each vertex: costs of single-source shortest paths from s

- Compute magical vertex tokens running SSSP Bellman-Ford algorithm once
- Add artificial source vertex s which has an outgoing edge of cost 0 to every vertex in G
- Run Bellman-Ford and compute the costs of shortest paths from s to every other vertex
- At the end - set p_v = cost of the shortest path s~>v

These are your magical vertex tokens, which will make the cost of each edge non-negative!

Transforming edges

- p_v = cost of a shortest path s~>v
- For every edge e=(u,v) new cost $c_e' = c_e + p_u - p_v$



Transformed graph with non-negative edge costs:
ready to run n*Dijkstra to compute all-pair shortest paths

Johnson’s algorithm

- Convert G(V,E) into G' by adding a new vertex s and n edges (s,v) of cost 0 to every vertex v ∈ V

- Run Bellman-Ford (G' with source s) [if it reports a negative-cost cycle – halt]

- For each v ∈ V define p_v = cost of the shortest path s~>v in G’
  For each edge e=(u,v) ∈ E, define new cost $c_e' = c_e + p_u - p_v$

- Run Dijkstra n times on G using new edge costs and starting from every vertex v ∈ V

- Extract the cost of the original path for each pair of vertices

easy?
Think how

Reduction of the APSS problem for general graph to:
1 SSSP for general graphs + n SSSP for graphs with non-negative edge costs

Johnson's algorithm: running time

$O(n)$

- Convert G(V,E) into G' by adding a new vertex s and n edges (s,v) of cost 0 to every vertex v ∈ V

$O(nm)$

- Run Bellman-Ford (G' with source s) [if it reports a negative-cost cycle – halt]

$O(m)$

- For each v ∈ V define pv = cost of the shortest path s~>v in G'
  For each edge e=(u,v) ∈ E, define new cost $c_e' = c_e + p_u – p_v$

$n*O(m \log n)$

- Run Dijkstra n times on G using new edge costs and starting from every vertex v ∈ V

$O(n^2)$

- Extract the cost of the original path for each pair of vertices

$O(mn \log n)$

Much better than $O(n^3)$ Floyd-Warshall **for sparse graphs**

Johnson's algorithm: correctness

- We have already proven that using tokens of each vertex to reweigh edges does not change the order of paths u~>v: the shortest path remains the shortest even after reweighting: see Reweighting technique slide

- What remains is to prove the following:

Lemma

For every edge e=(u,v) of G, the reweighted edge cost $c_e' = c_e + p_u - p_v$ is non-negative.

Lemma

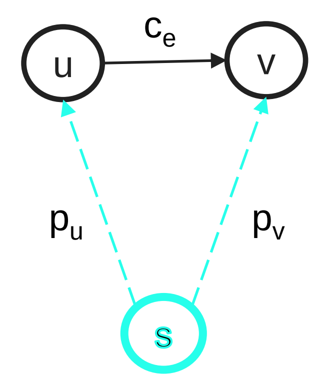
For every edge e=(u,v) of G, the reweighted edge cost $c_e' = c_e + p_u - p_v$ is non-negative.

Proof

- Let (u,v) be an arbitrary pair of vertices in G connected by an edge e $u \to v$ with cost c_e.
- By construction,

  p_u = cost of a shortest path from s to u

  p_v = cost of a shortest path from s to v



If p_u is the cost of a shortest path s~>u
Then $p_u + c_e$ is the length of some path from s to v. This may be a shortest path from s to v, but there could be an even shorter path from s to v which does not pass through vertex u.
Hence, $p_u + c_e \geq p_v$

- Therefore, $c_e' = c_e + p_u - p_v \geq 0$ ∎