CS2040S
Data Structures and Algorithms

Bellman-Ford

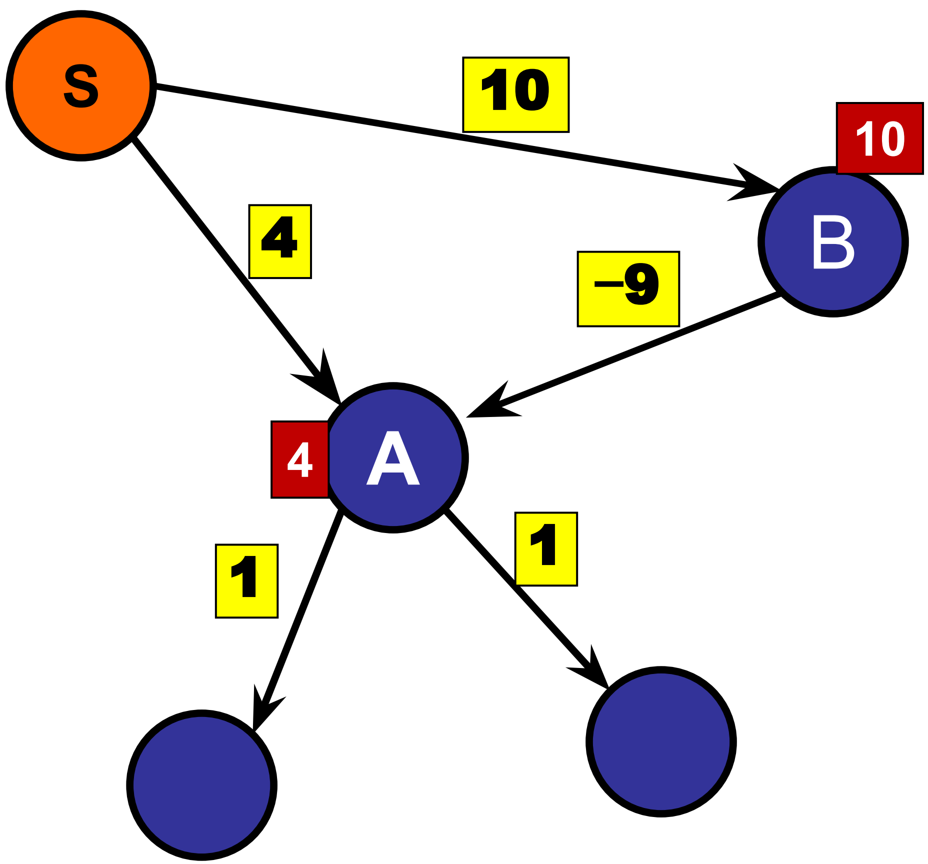
Last Time

Single Source Shortest Paths (SSSP):

- Dijkstra
  - SSSP on non-negatively weighted graphs

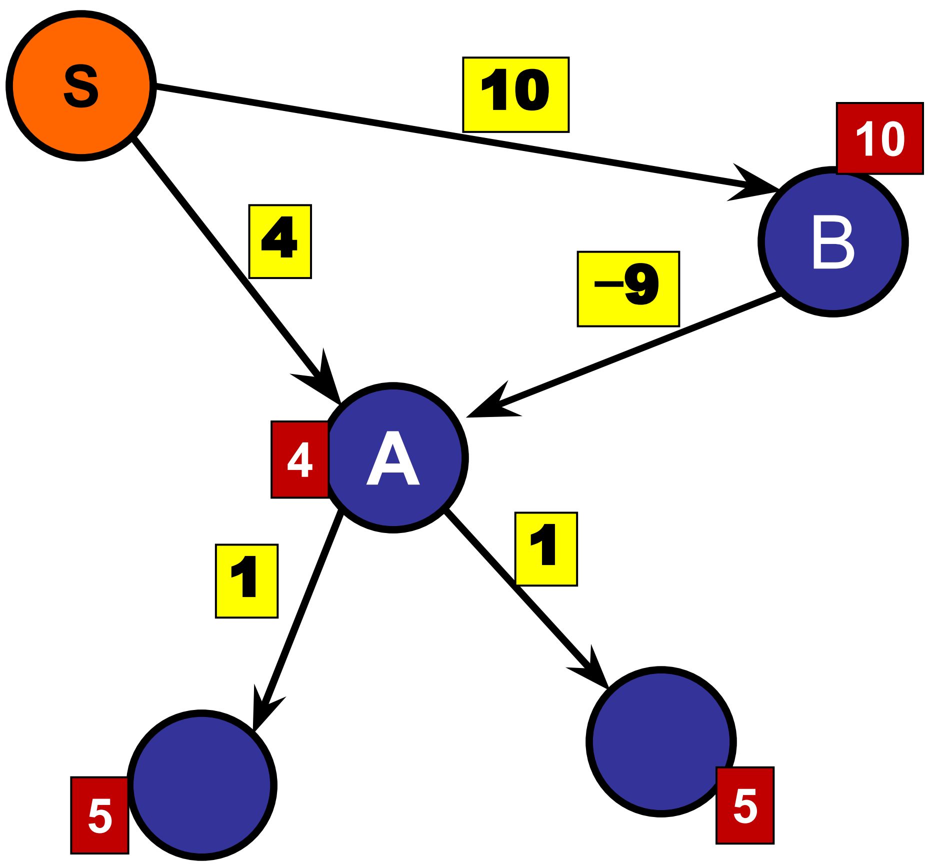
Dijkstra's Algorithm

Edges with negative weights?



Dijkstra's Algorithm

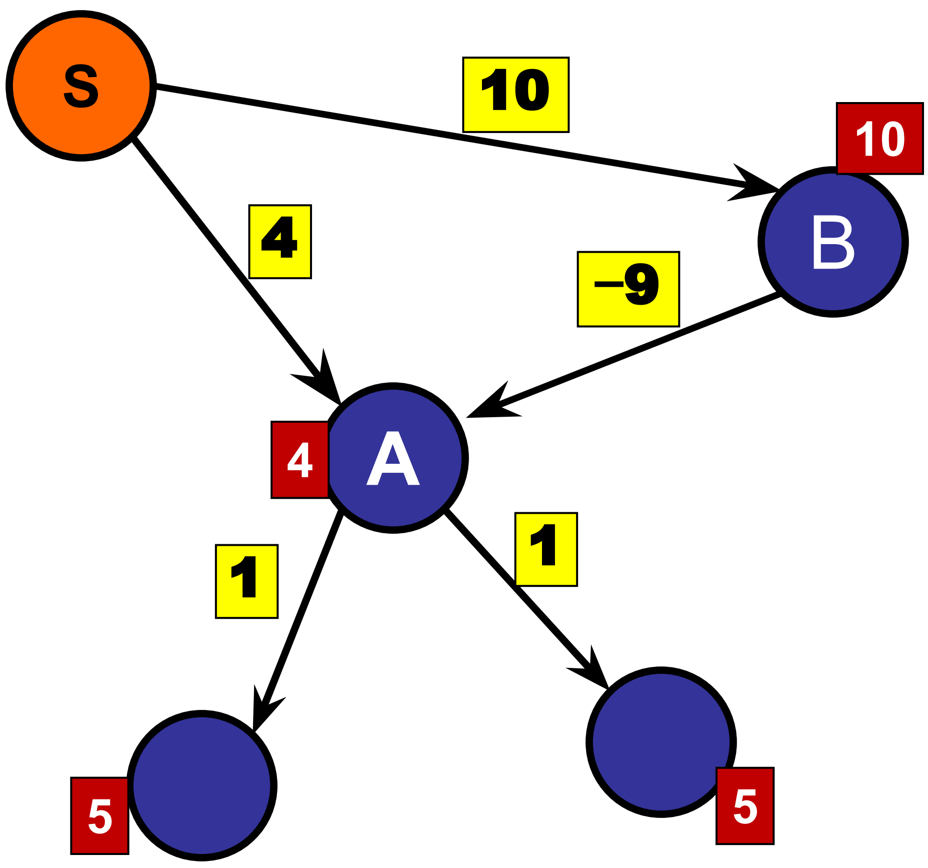
Edges with negative weights?



Step 1: Remove A.
Relax A.
Mark A done.

Dijkstra's Algorithm

Edges with negative weights?



Step 1: Remove A.
Relax A.
Mark A done.

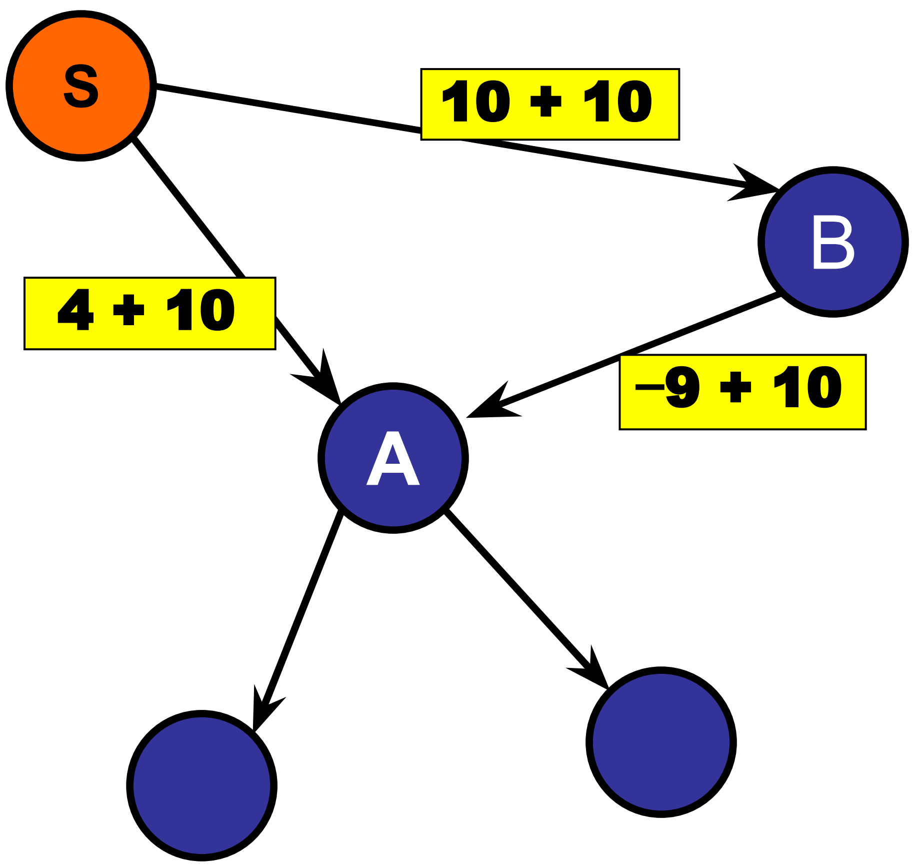
…

Step 4: Remove B.
Relax B.
Mark B done.

Oops: We need to update A.

Dijkstra's Algorithm

Can we reweight? e.g.: weight += 10

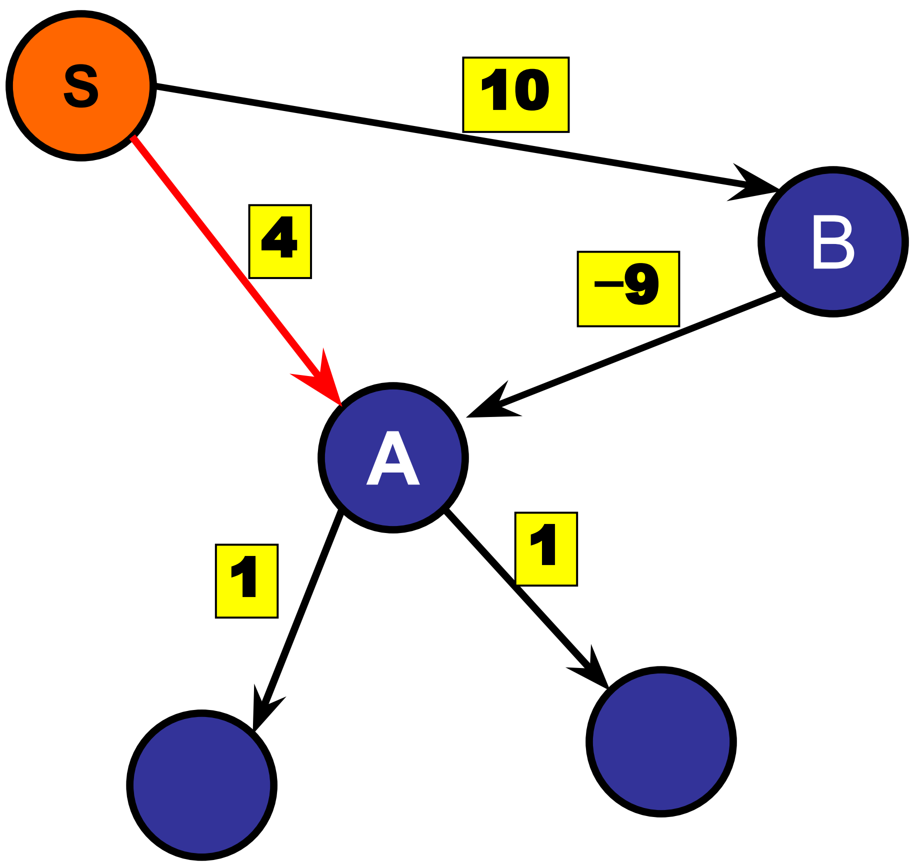


Can we reweight the graph?

1. Yes.
2. Only if there are no negative weight cycles.
3. No.

Dijkstra's Algorithm

Can we reweight?

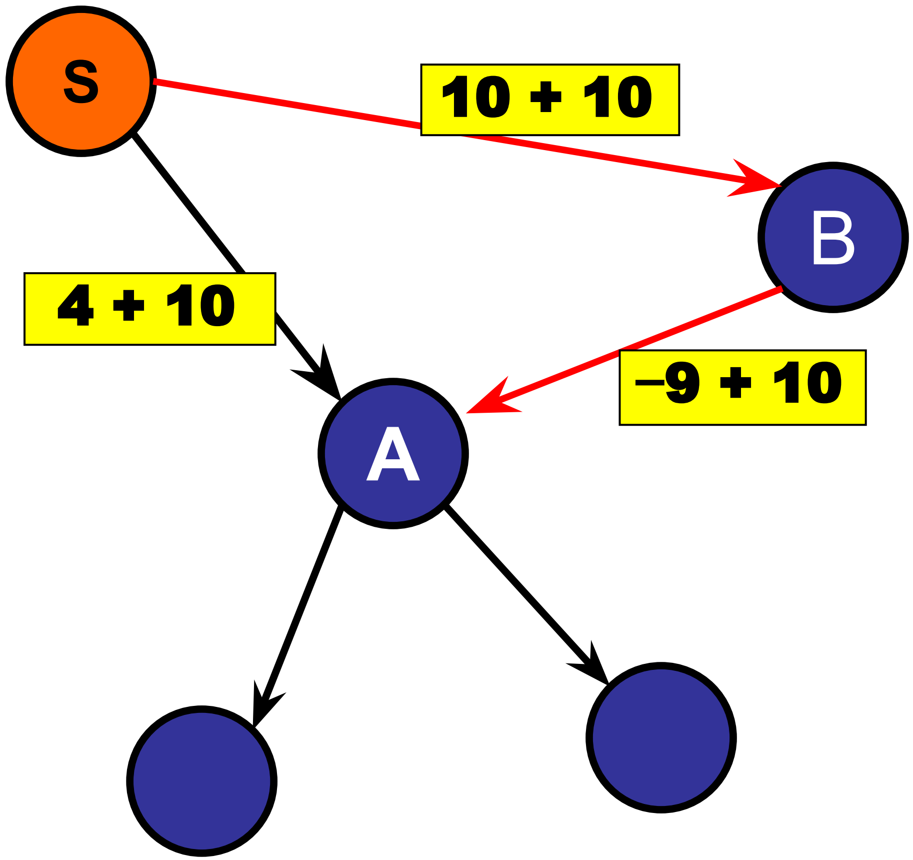


Path S-B-A: 1

Path S-A: 4

Dijkstra's Algorithm

Can we reweight?

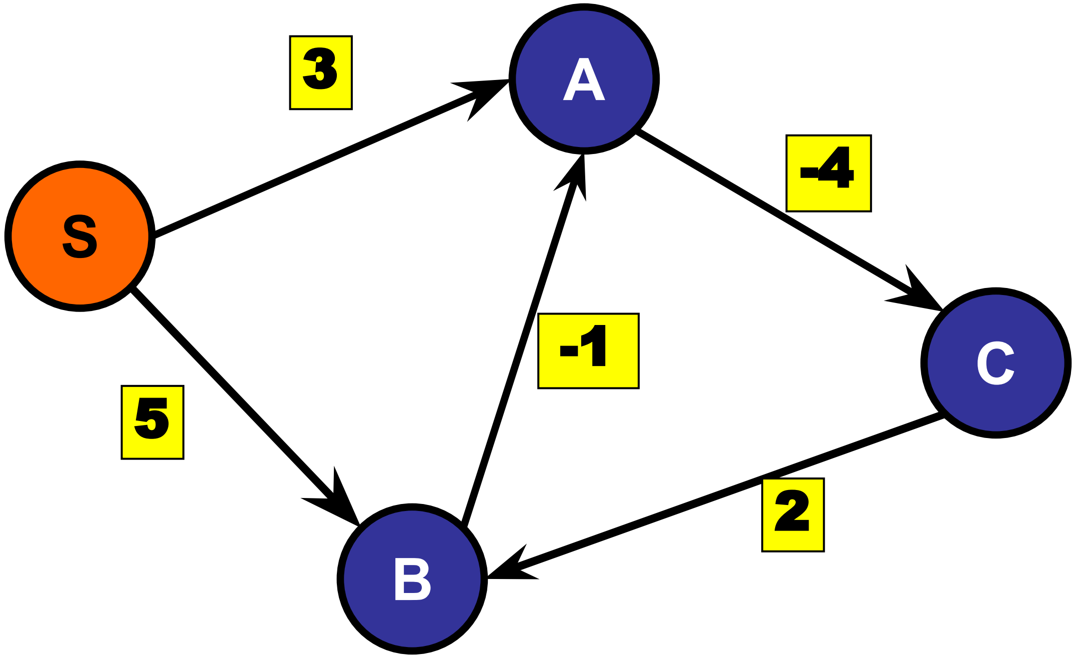


Path S-B-A: 21

Path S-A: 14

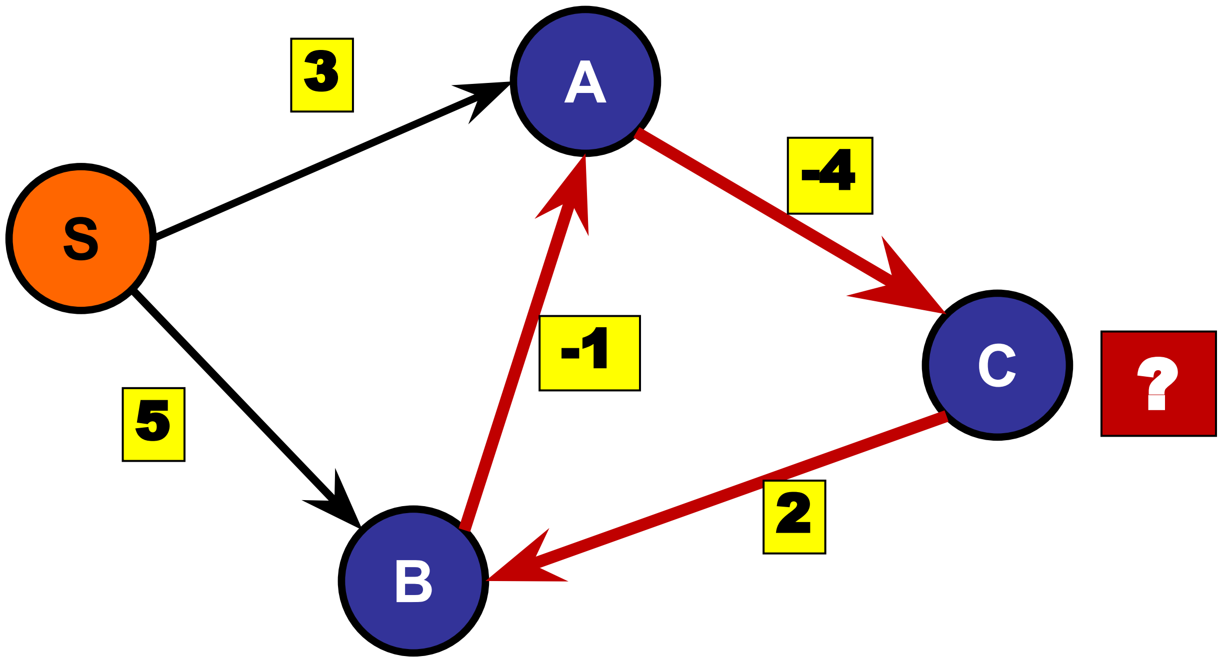
Negative Cycles:

What if edges have negative weight?



Negative Cycles:

What if edges have negative weight?



d(S,C) is infinitely negative!

Today

Single Source Shortest Paths (SSSP):

- Bellman Ford
  - SSSP on negative edge graphs
  - Negative Cycle Detection

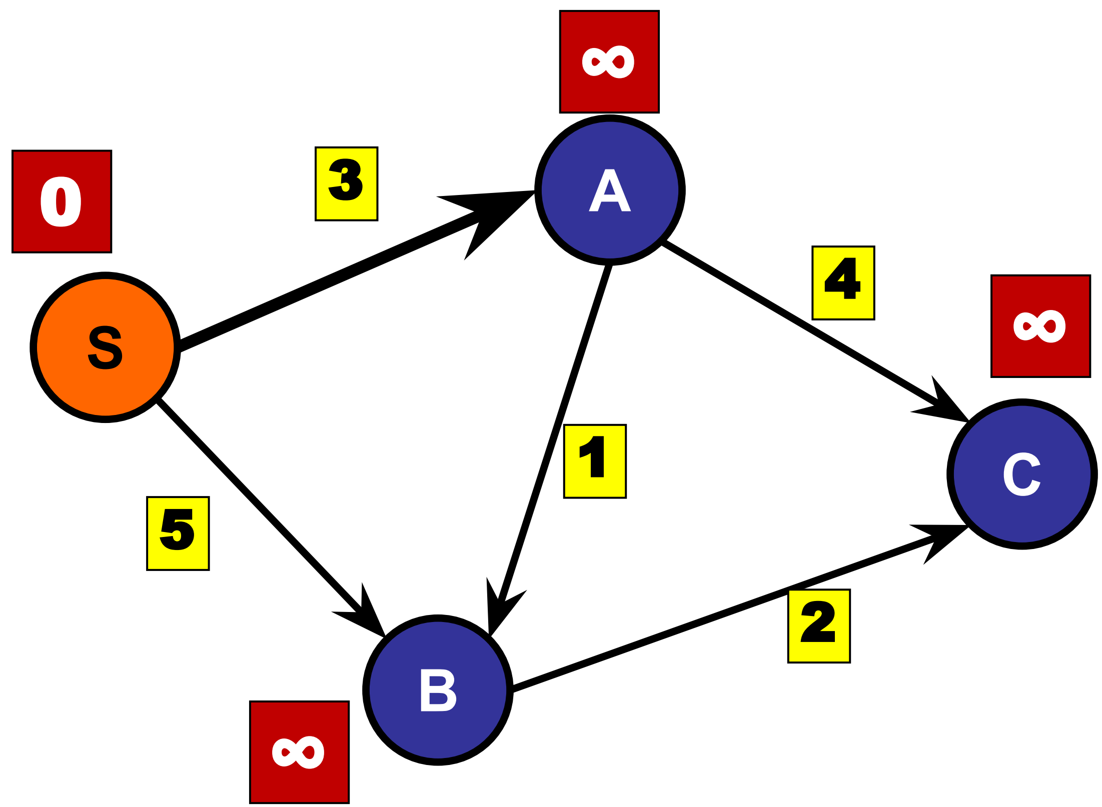
Some graph techniques

Recall: Path Relaxation

Let’s quickly refresh path relaxation, we’re going to be doing a lot of that today.

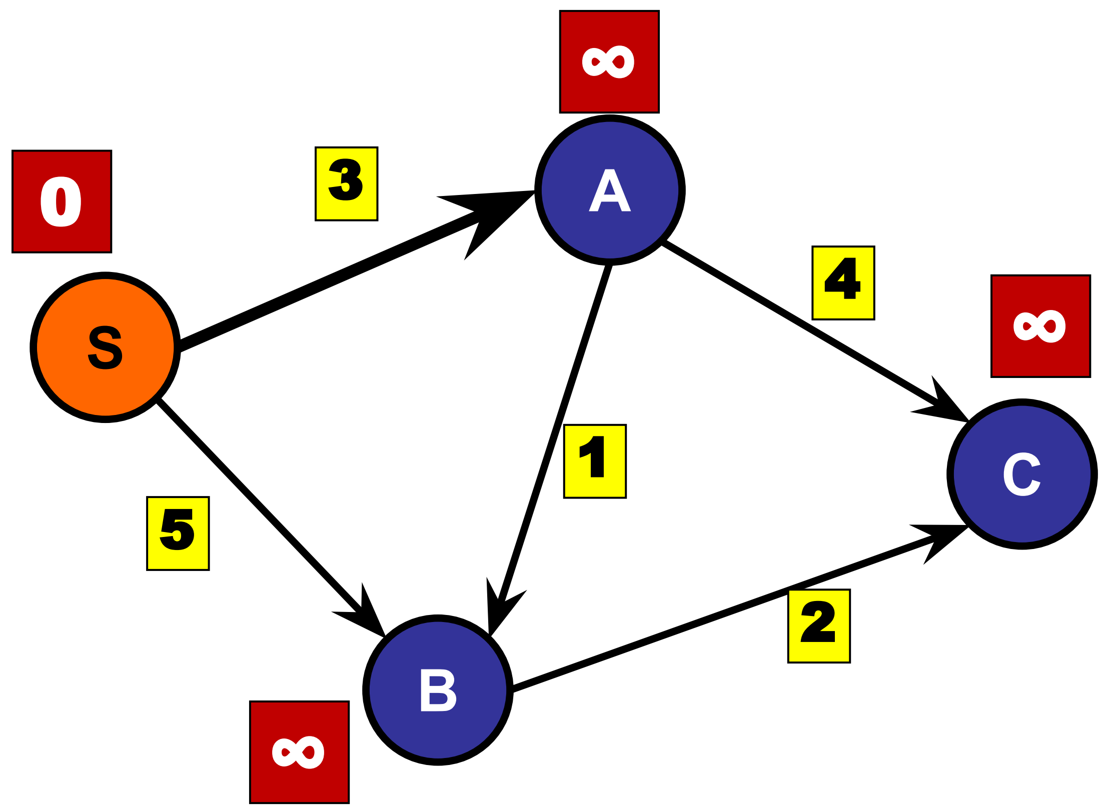
Recall: Path Relaxation

```
relax(int u, int v){
    if (dist[v] > dist[u] + weight(u,v))
        dist[v] = dist[u] + weight(u,v);
}
```



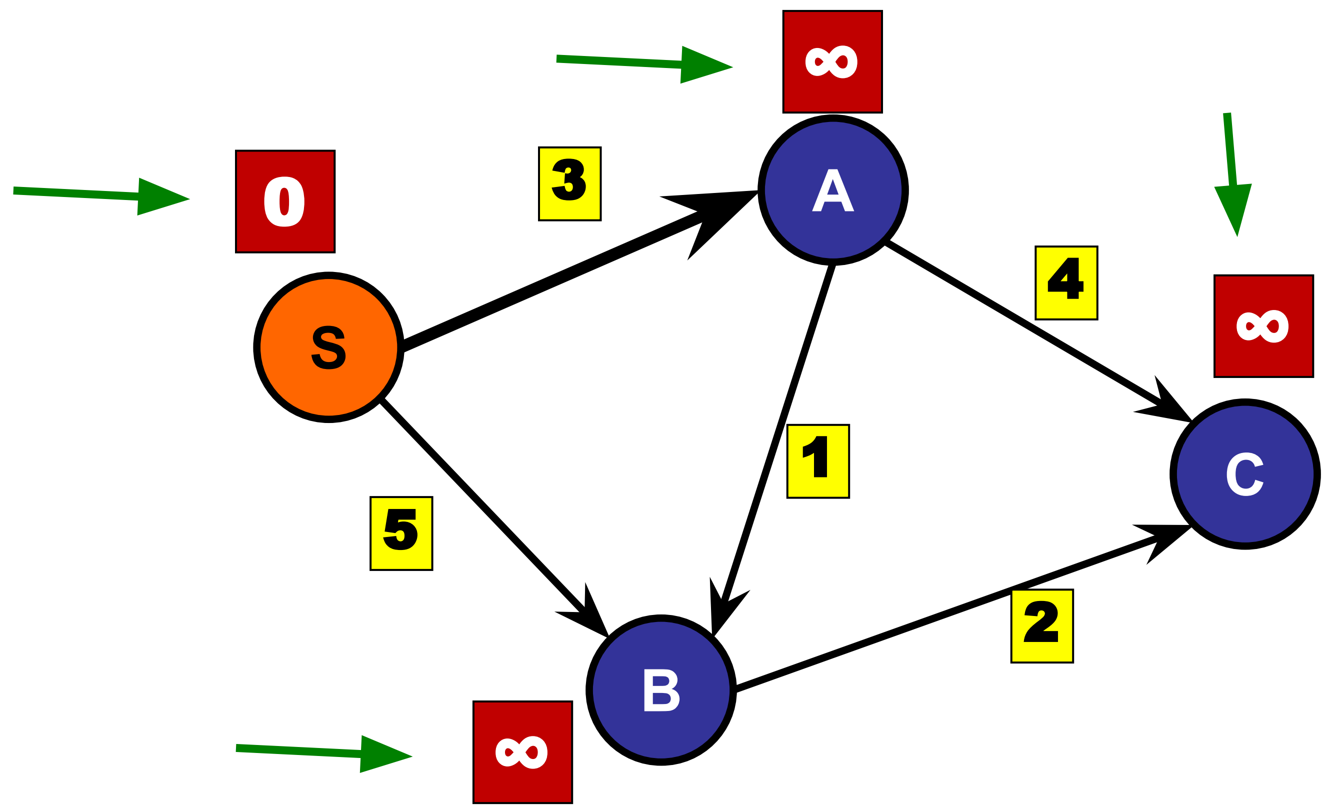
Recall: Path Relaxation

Let's try running path relaxation again.



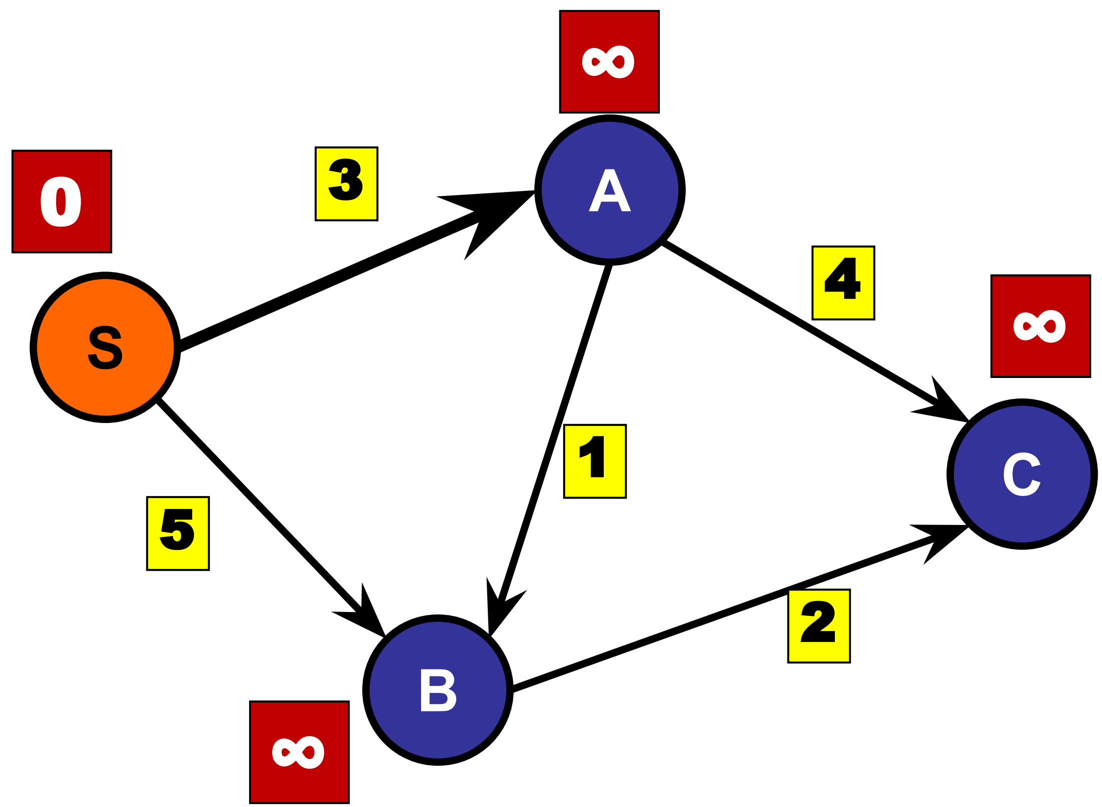
Recall: Path Relaxation

Our distance estimates.



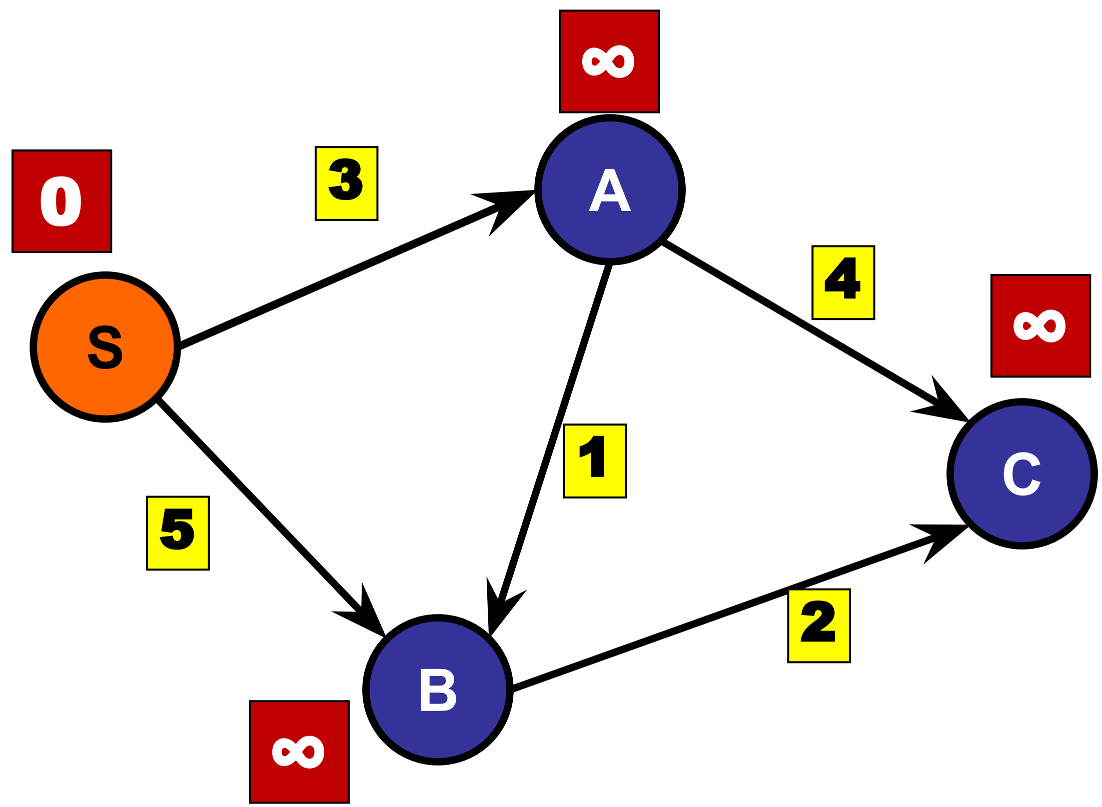
Recall: Path Relaxation

Let's say we ran relax() based on the edges we have, in some arbitrary ordering.



Recall: Path Relaxation

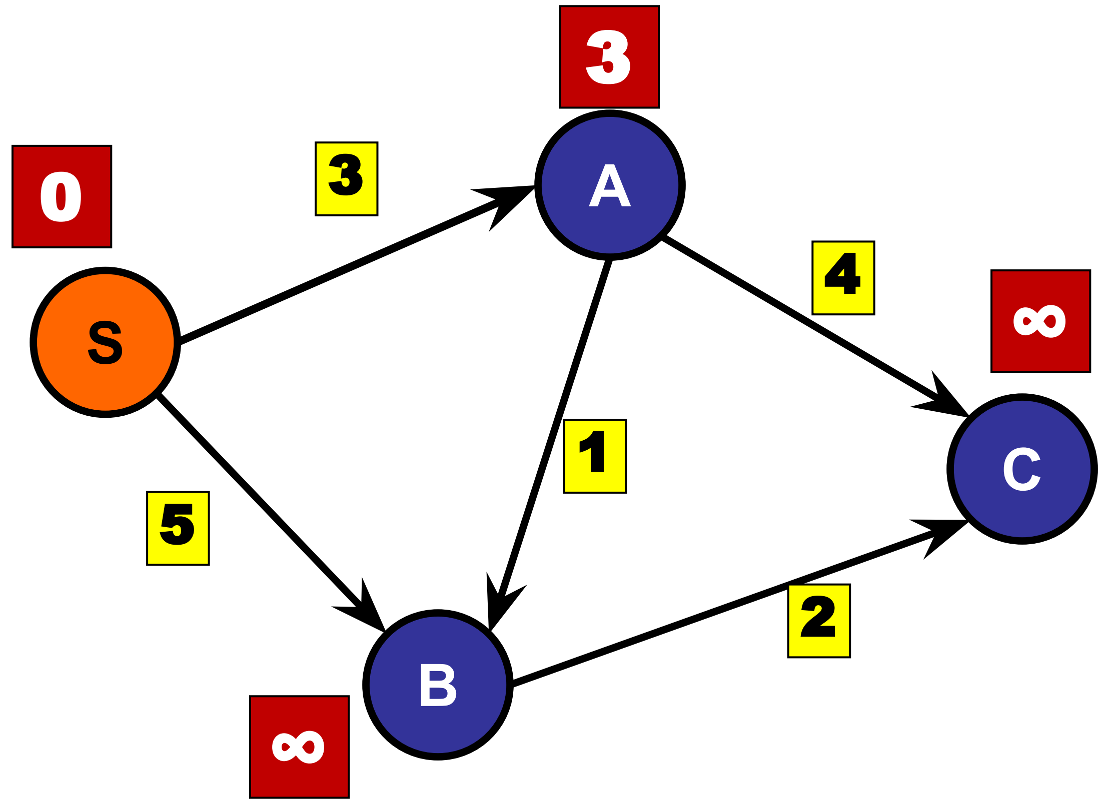
relax(S, A)



Recall: Path Relaxation

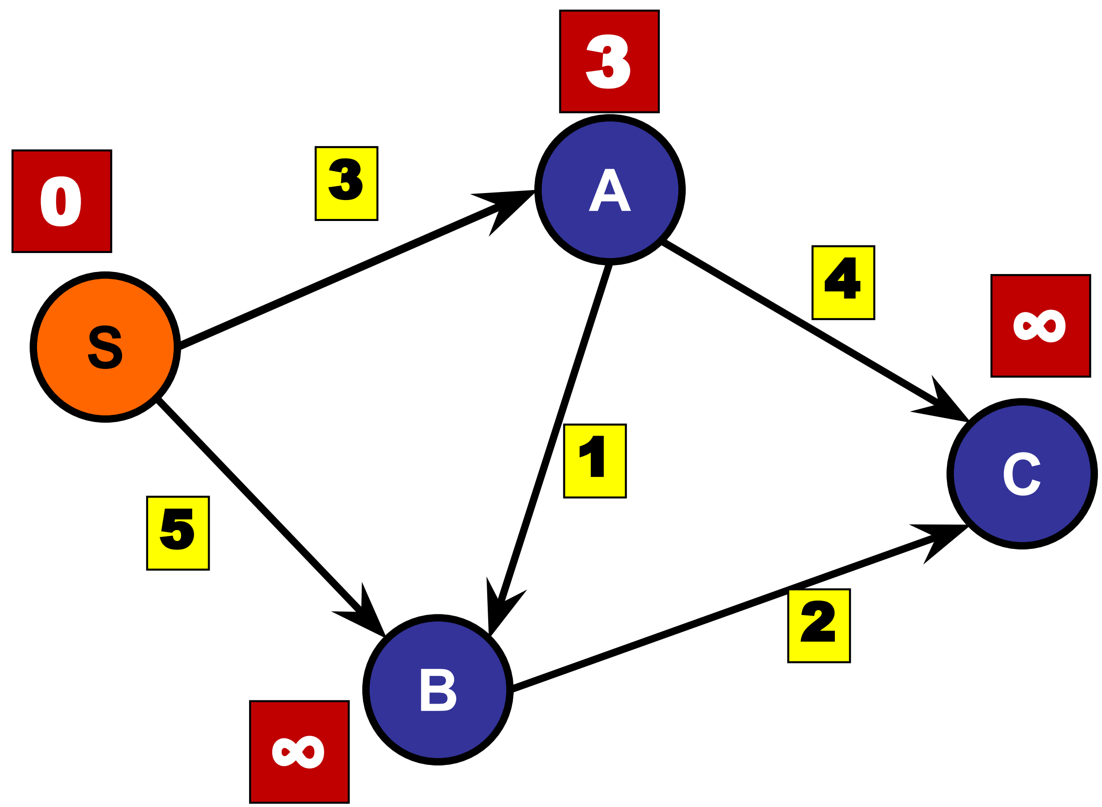
relax(S, A)

Reduced from ∞ to 3.



Recall: Path Relaxation

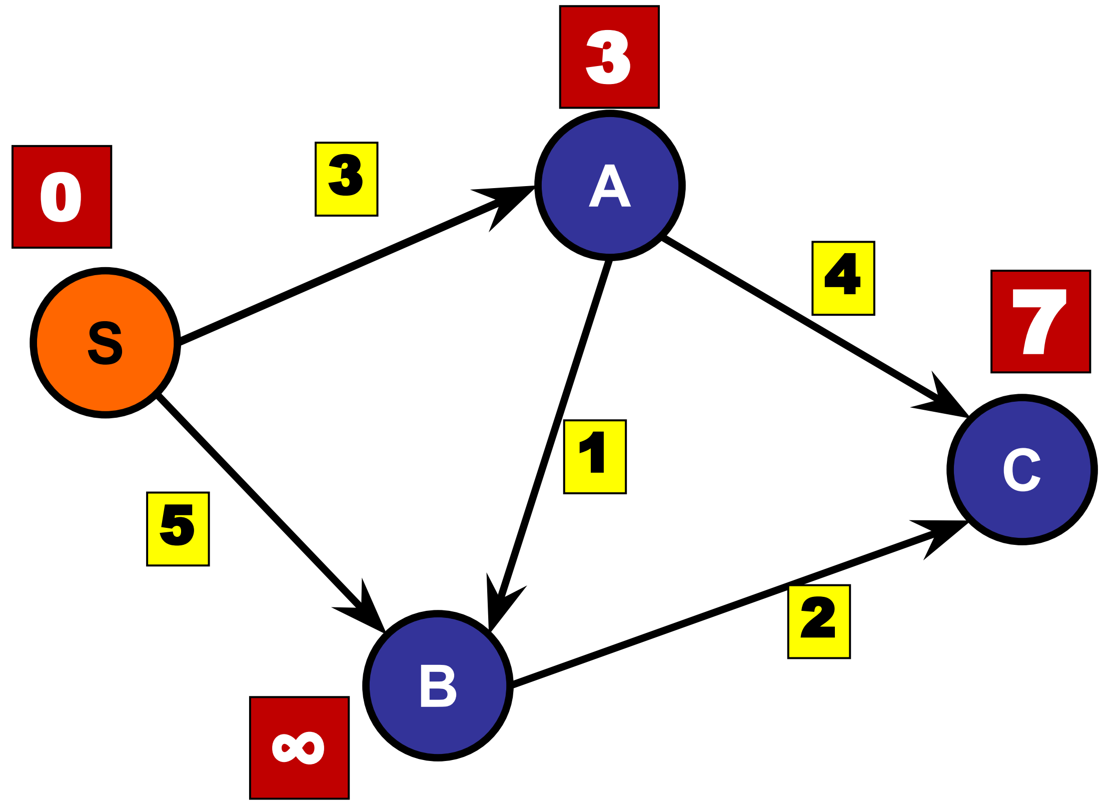
relax(A, C)



Recall: Path Relaxation

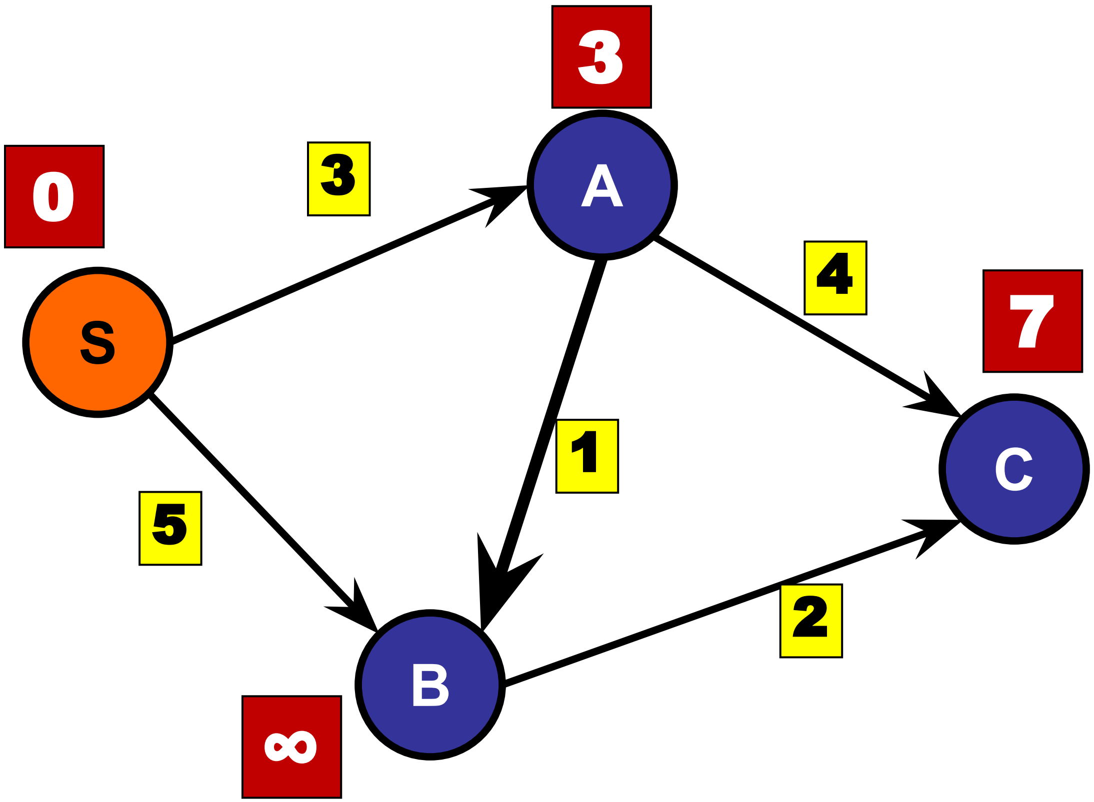
relax(A, C)

Reduced from ∞ to 3 + 4 = 7.



Recall: Path Relaxation

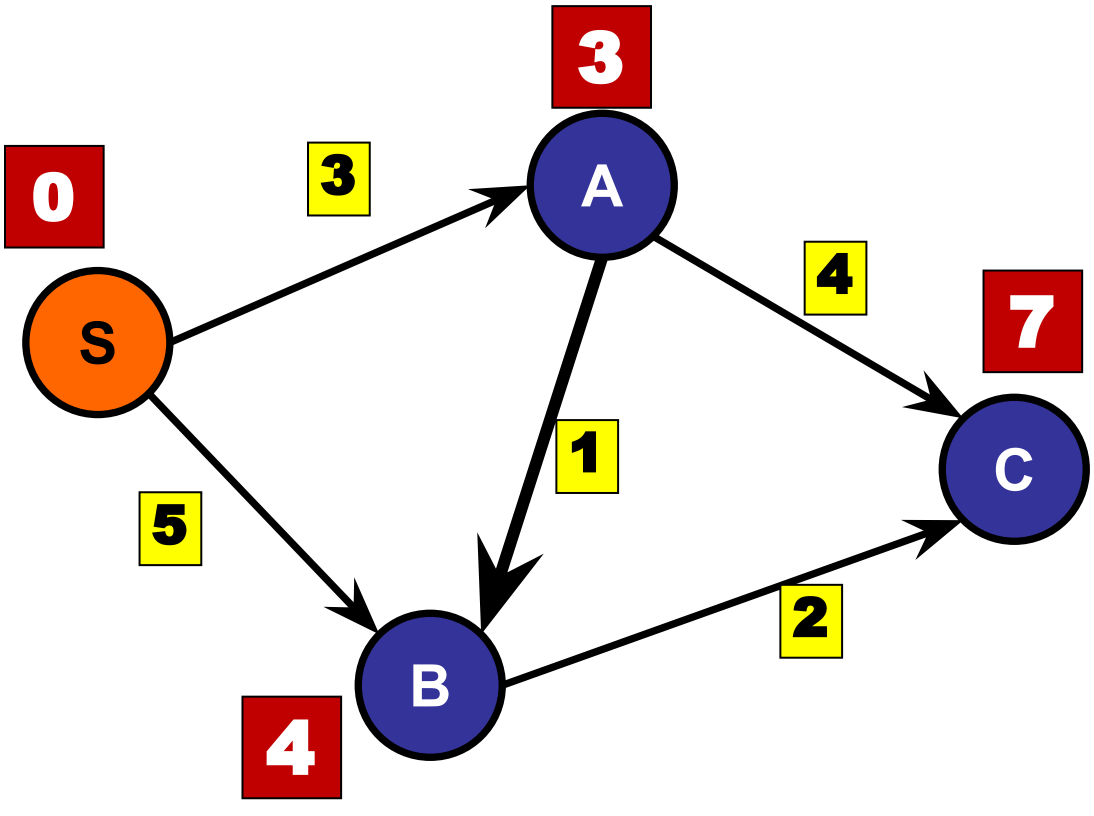
relax(A, B)



Recall: Path Relaxation

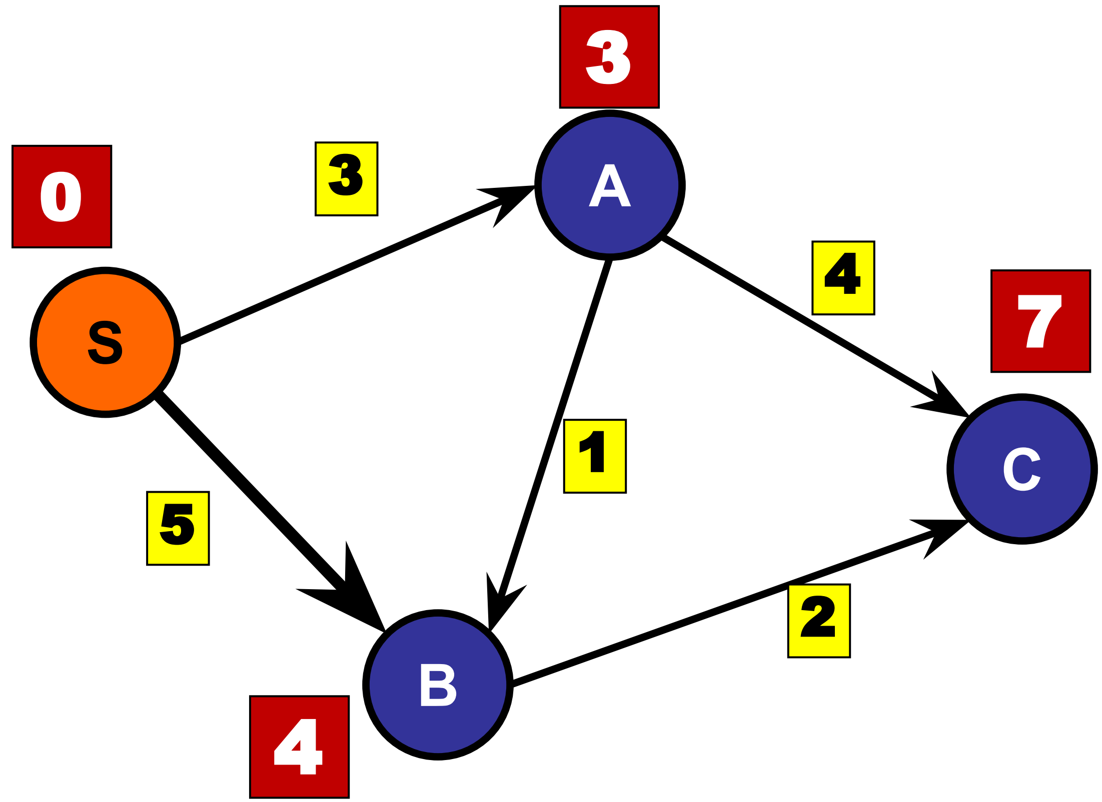
relax(A, B)

Reduced from ∞ to 3 + 1 = 4.



Recall: Path Relaxation

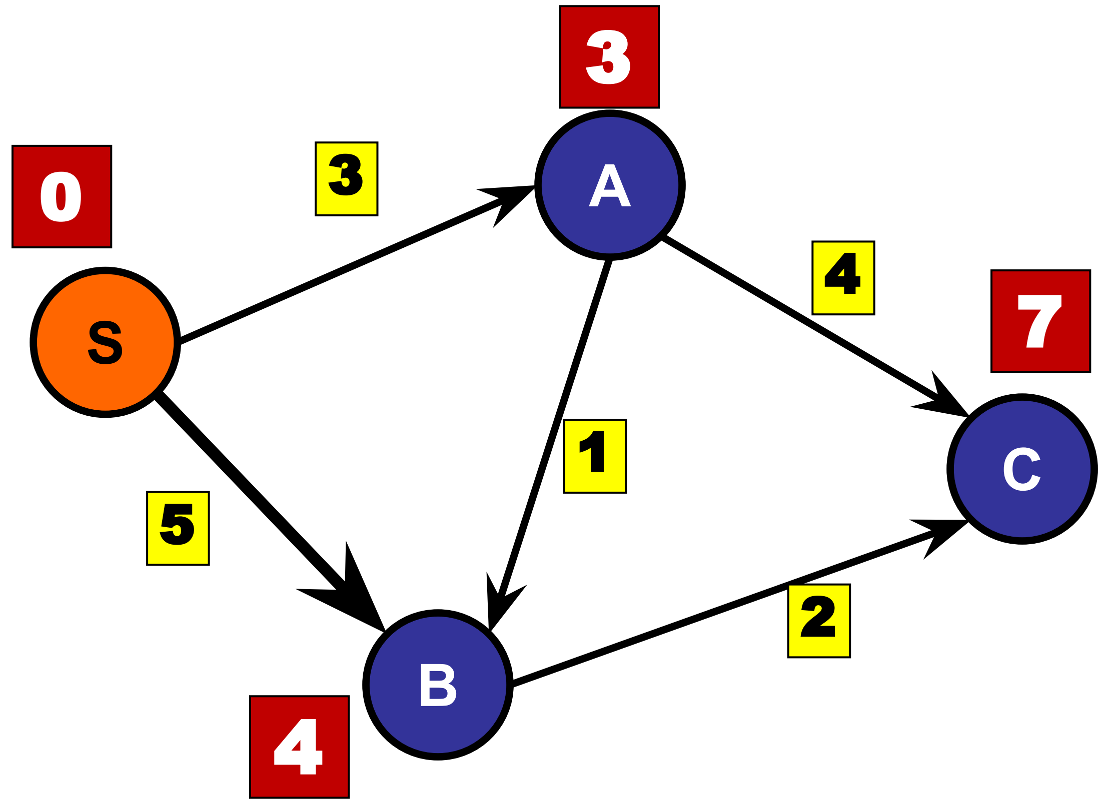
relax(S, B)



Recall: Path Relaxation

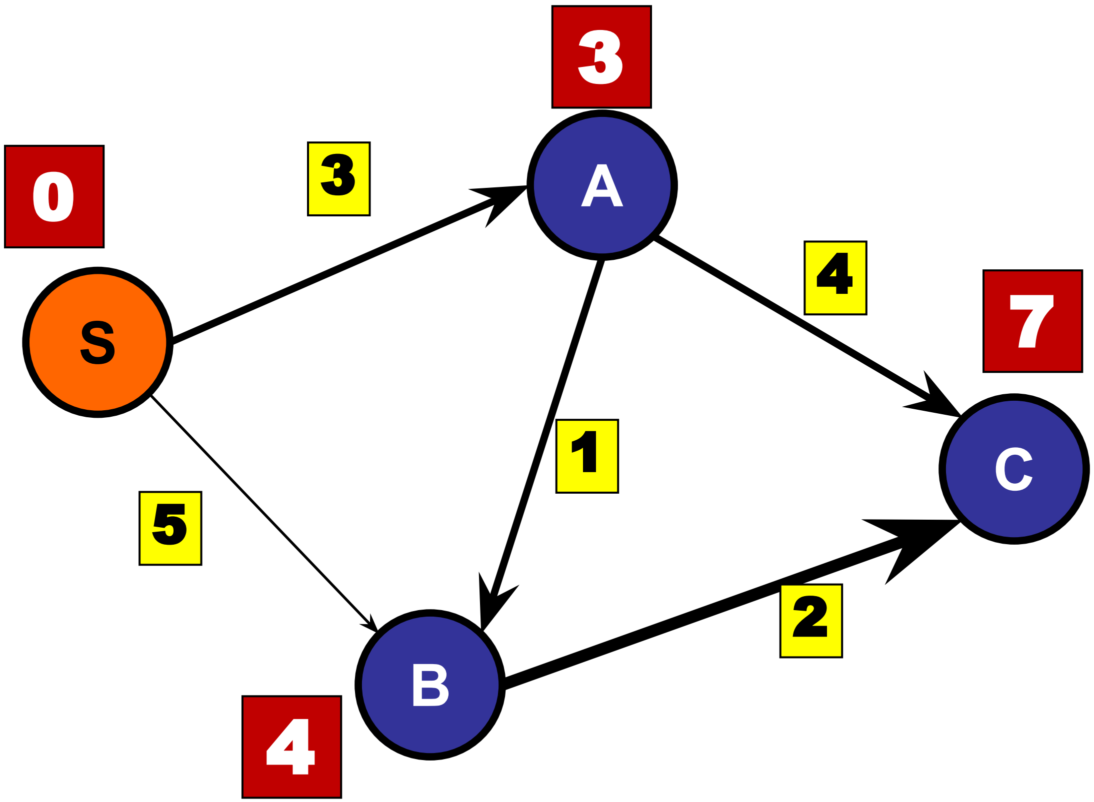
relax(S, B)

No change!



Recall: Path Relaxation

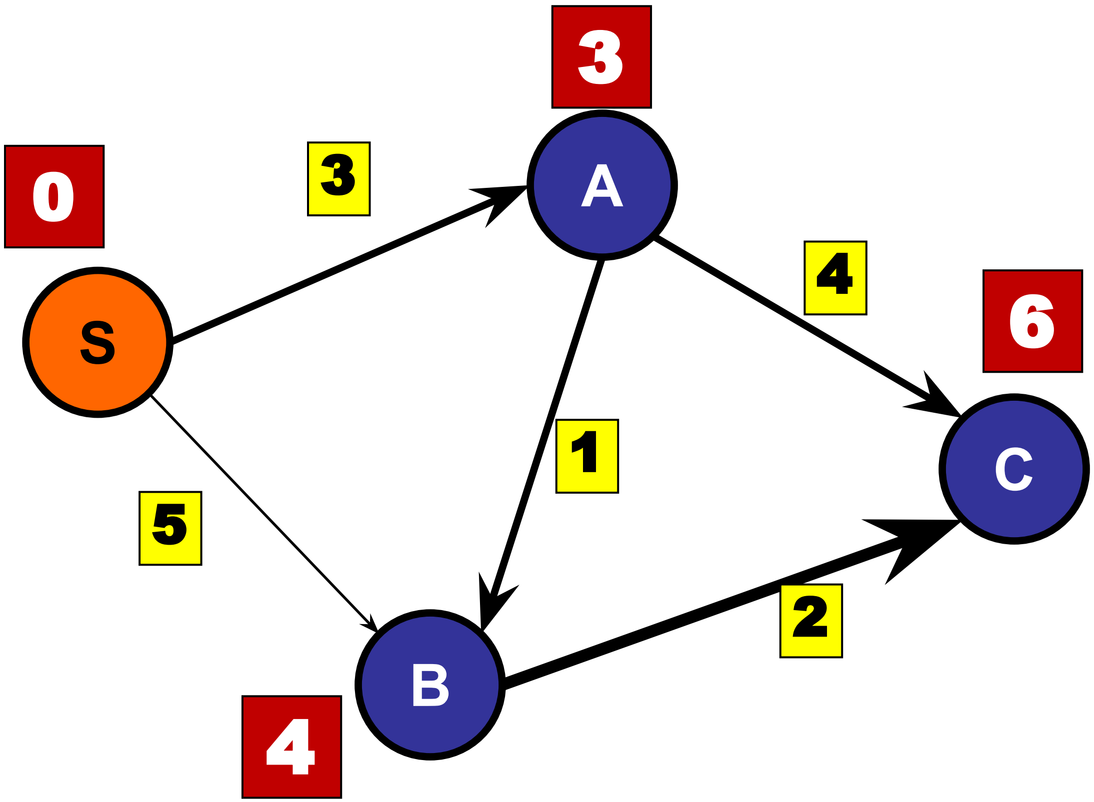
relax(B, C)



Recall: Path Relaxation

relax(B, C)

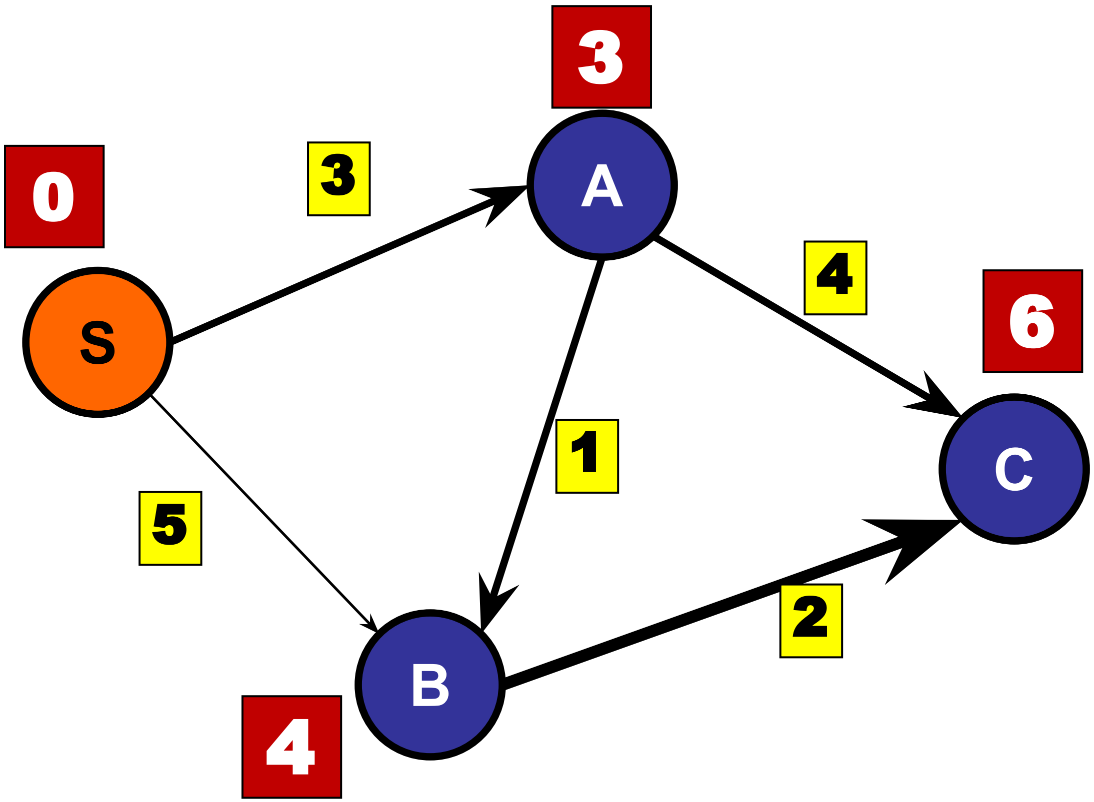
Reduced from 7 to 4 + 2 = 6.



Shortest Paths

for (edge **e** : graph)

relax(**e**)

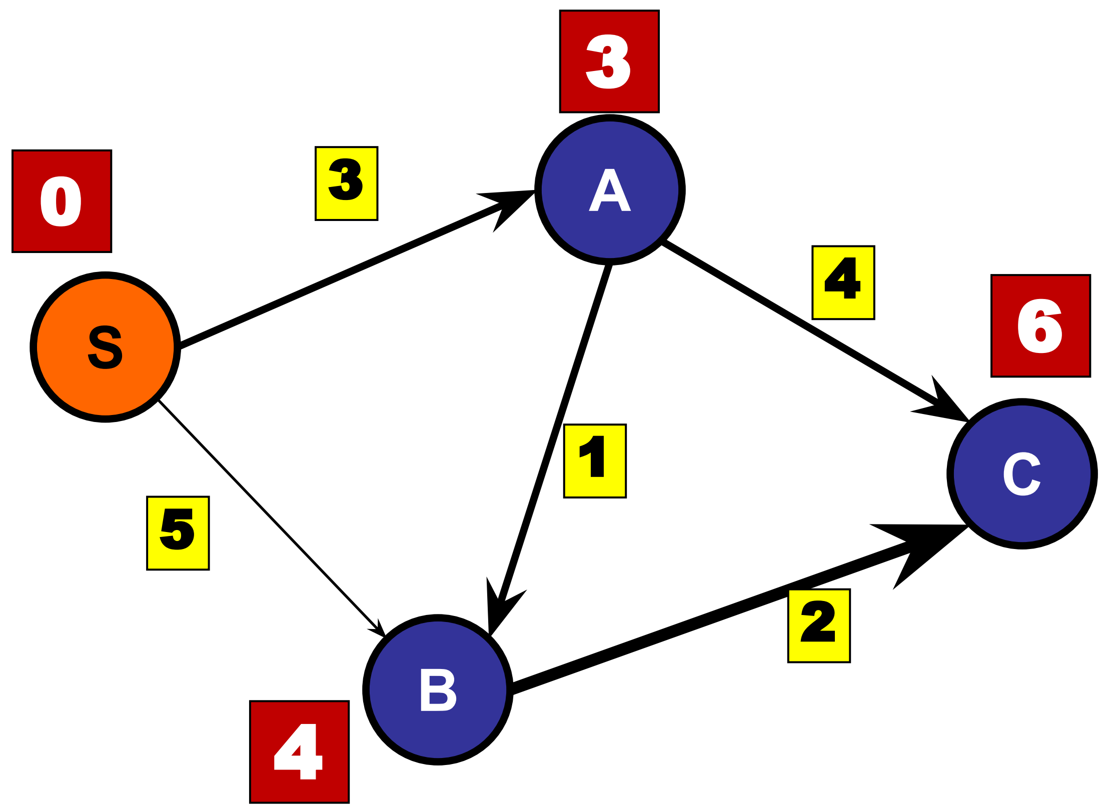


Shortest Paths

for (edge **e** : graph)

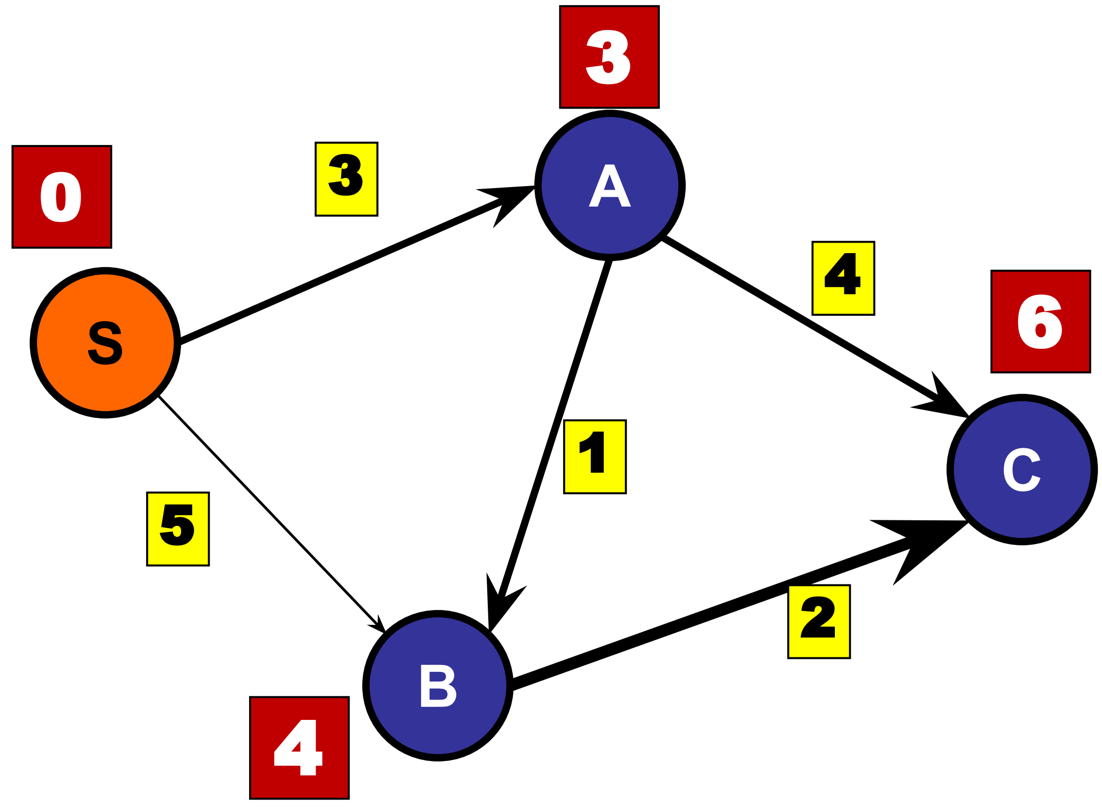
relax(**e**)

Let's say the order in which we iterate over the edges is not determined by us.



Does this algorithm work?
for every edge e: relax(e)

1. Yes
2. Sometimes
3. Never

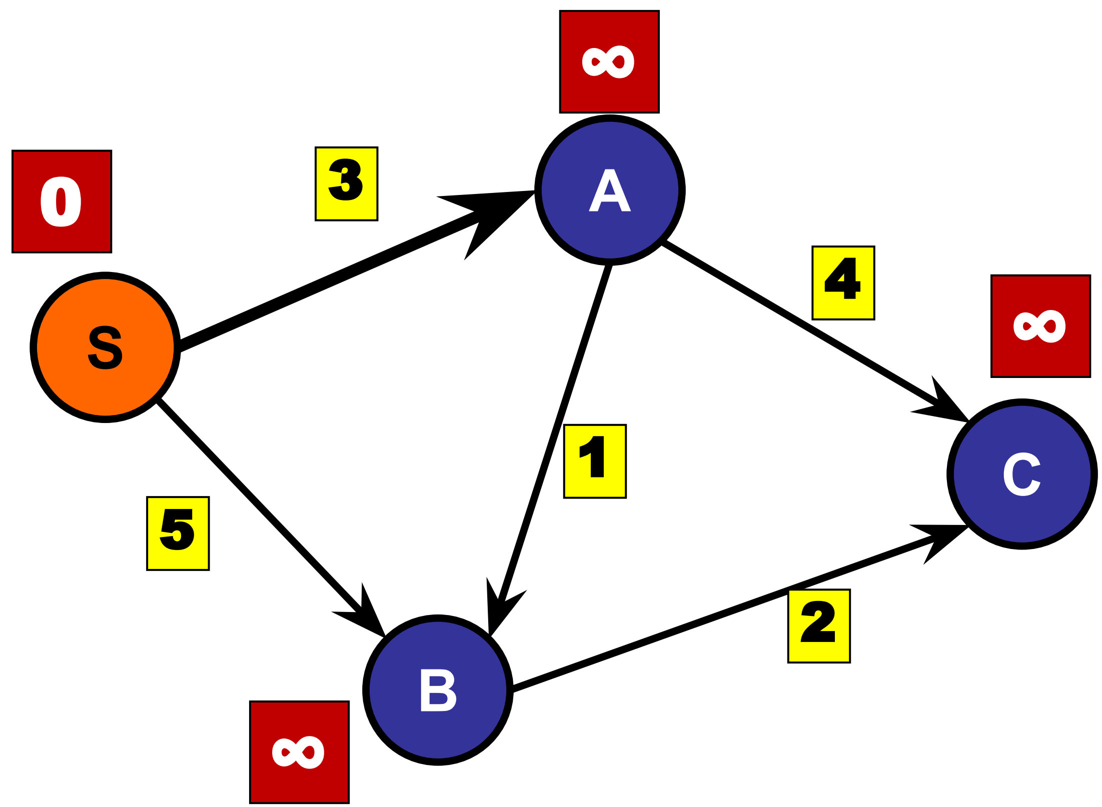


Shortest Paths

for (edge **e** : graph)

relax(**e**)

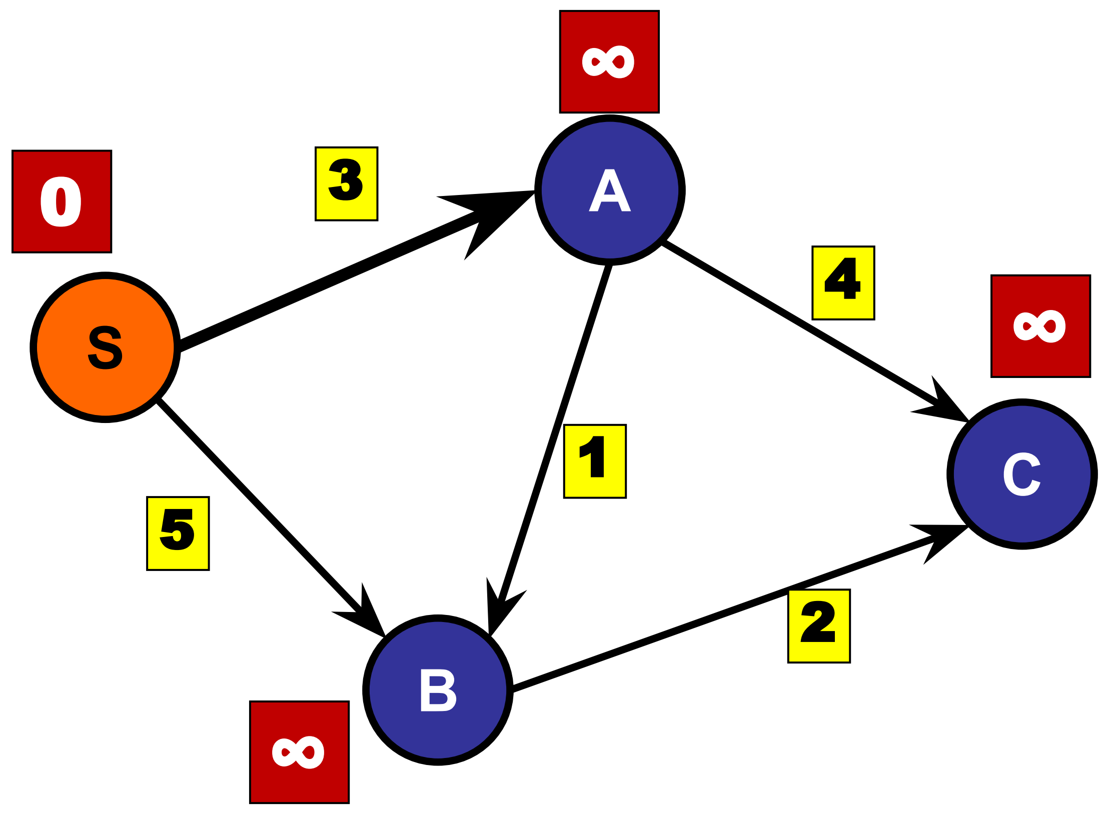
What if the ordering was:
(A, C)
(A, B)
(B, C)
(S, A)
(S, B)



What happens if we ran this for a single round? What is the distance estimate of A?

1. 3
2. ∞

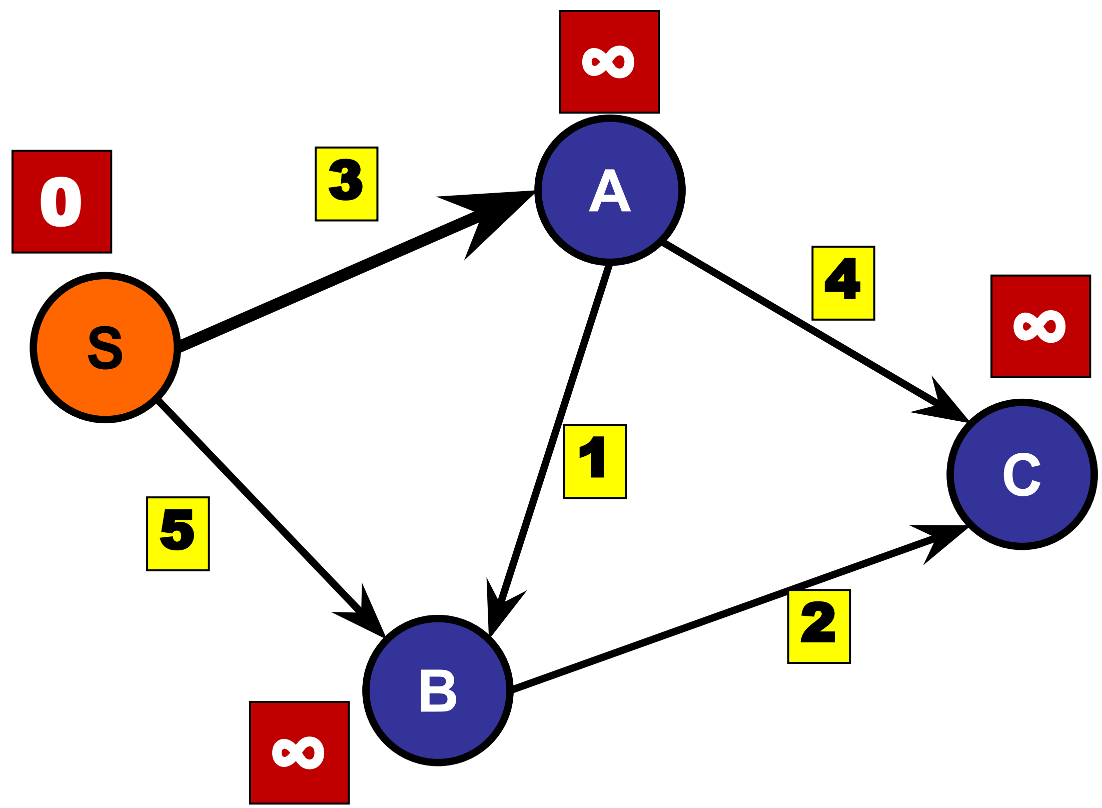
What if the ordering was:
(A, C)
(A, B)
(B, C)
(S, A)
(S, B)



What happens if we ran this for a single round? What is the distance estimate of B?

1. 3
2. 4
3. 5
4. ∞

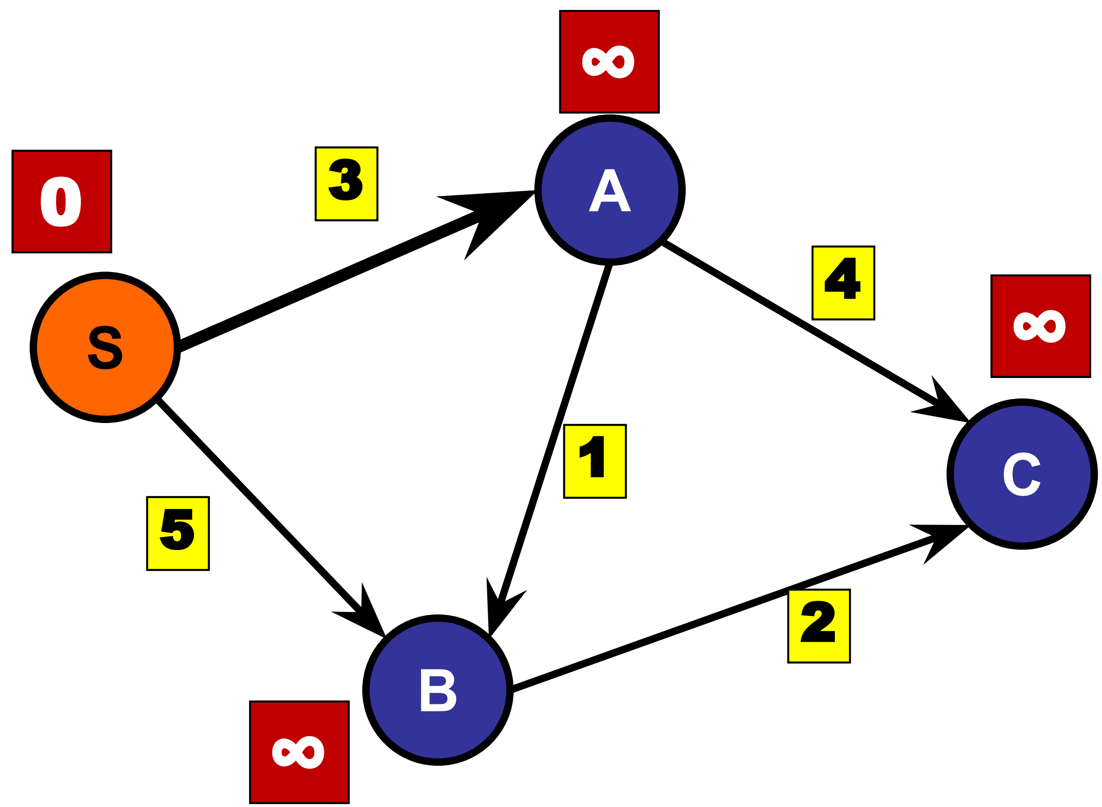
What if the ordering was:
(A, C)
(A, B)
(B, C)
(S, A)
(S, B)



What happens if we ran this for a single round? What is the distance estimate of C?

1. 7
2. 6
3. ∞

What if the ordering was:
(A, C)
(A, B)
(B, C)
(S, A)
(S, B)



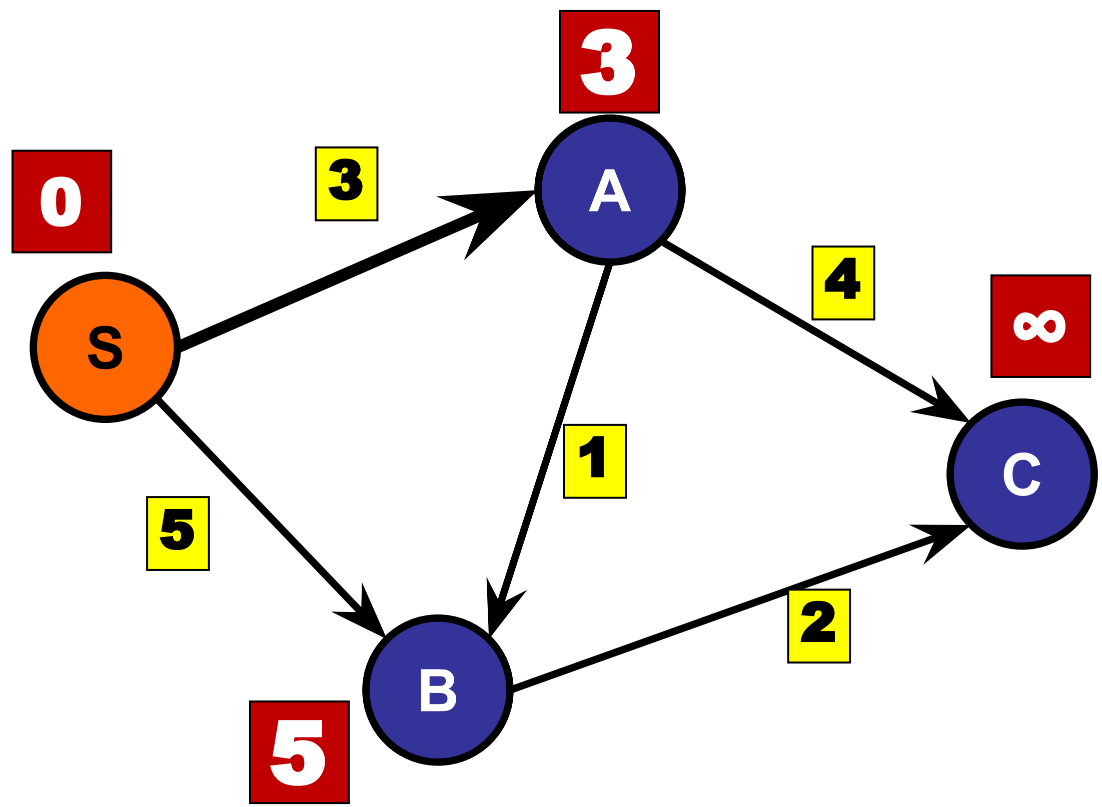
Shortest Paths

for (edge **e** : graph)

relax(**e**)

After 1 round:

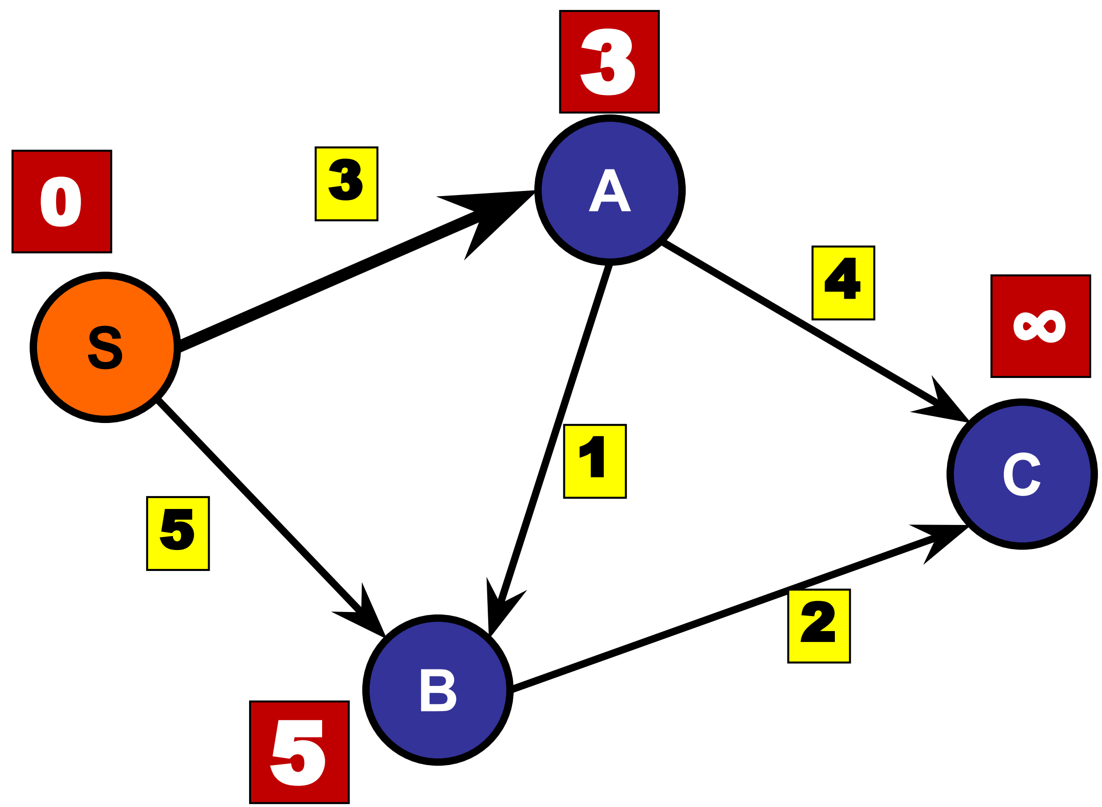
What if the ordering was:
(A, C)
(A, B)
(B, C)
(S, A)
(S, B)



Can we say that the nodes that are one hop away from source S have correct distance estimates?

1. Yes
2. No
3. Narp

What if the ordering was:
(A, C)
(A, B)
(B, C)
(S, A)
(S, B)



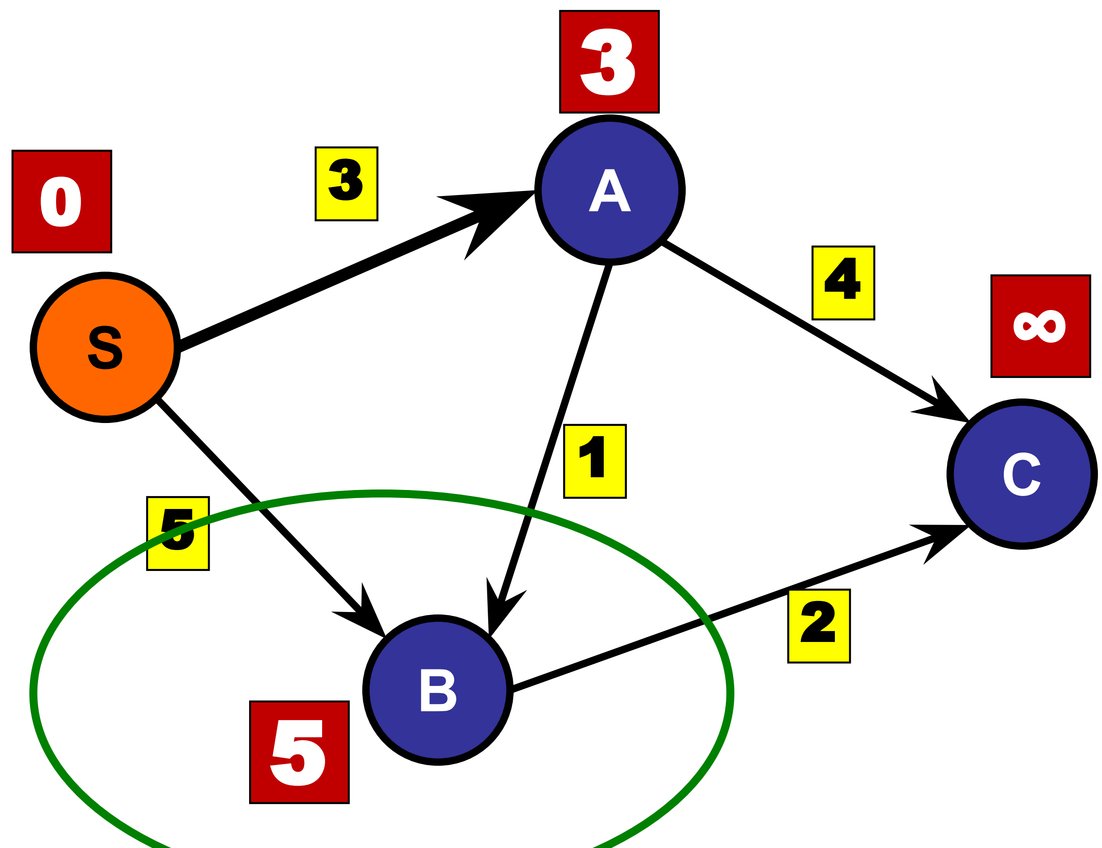
Shortest Paths

for (edge **e** : graph)

  relax(**e**)

After 1 round:
Node B's distance estimate is still not correct!

What if the ordering was:
(A, C)
(A, B)
(B, C)
(S, A)
(S, B)



Shortest Paths

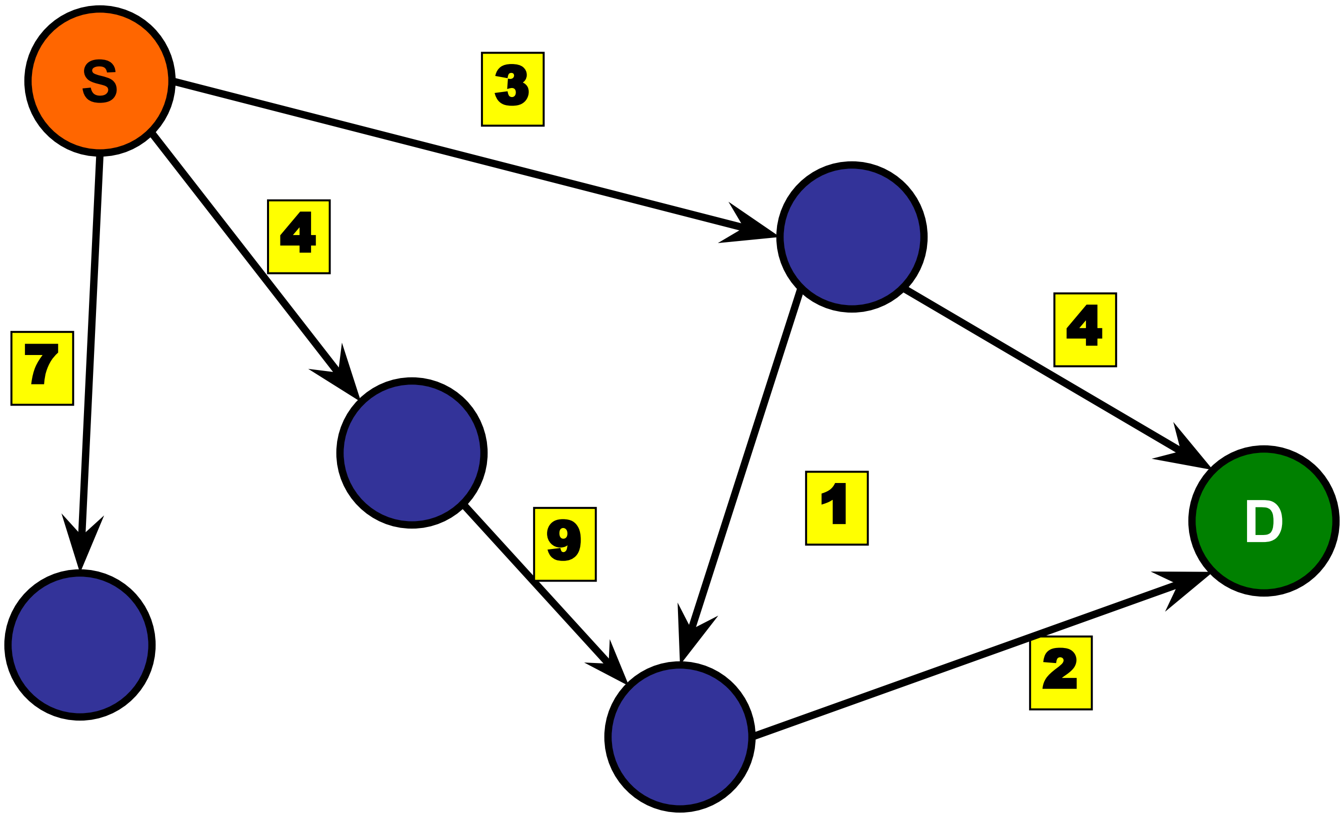
for (edge **e** : graph)

relax(**e**)

So this alone clearly doesn’t work. But can we at least say we’re making some progress?

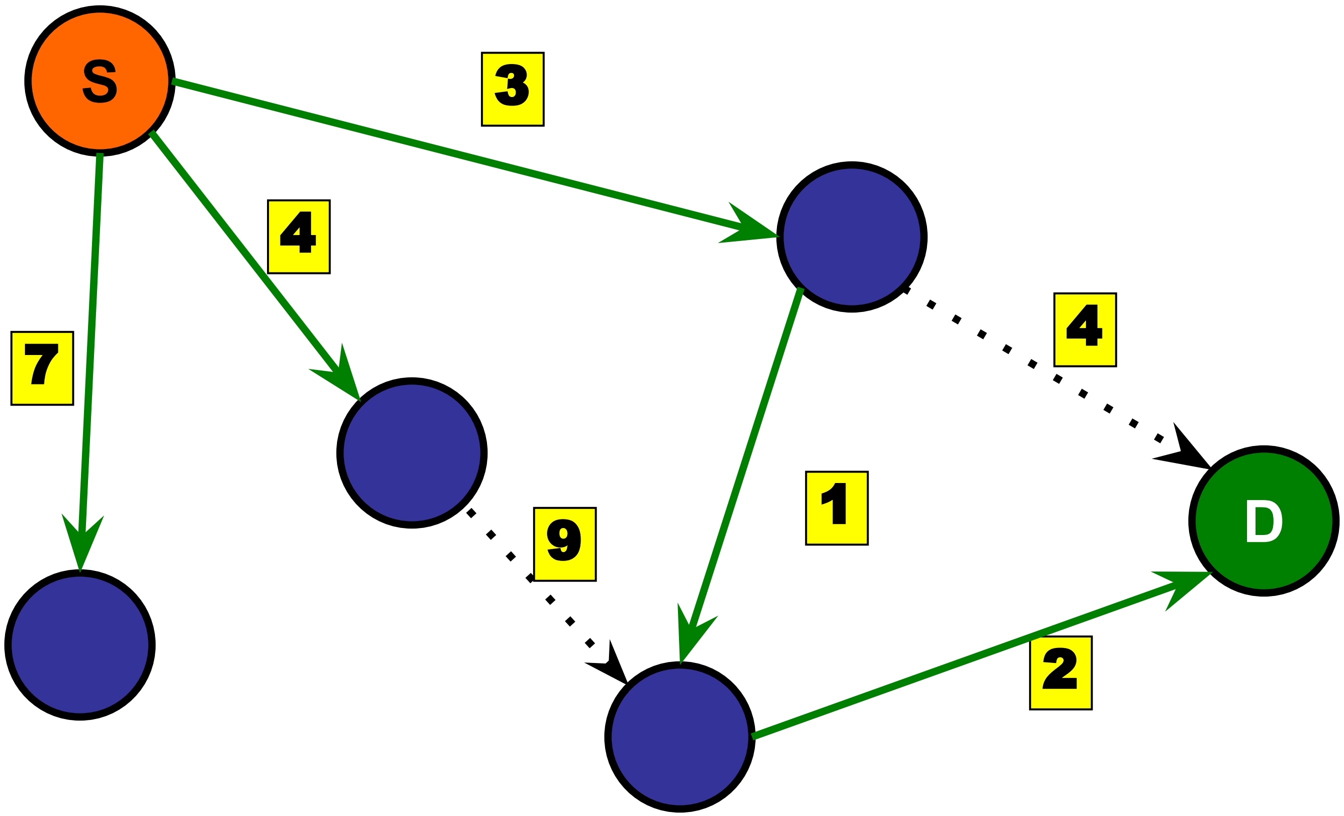
Idea:

Let’s consider some general directed, weighted graph.



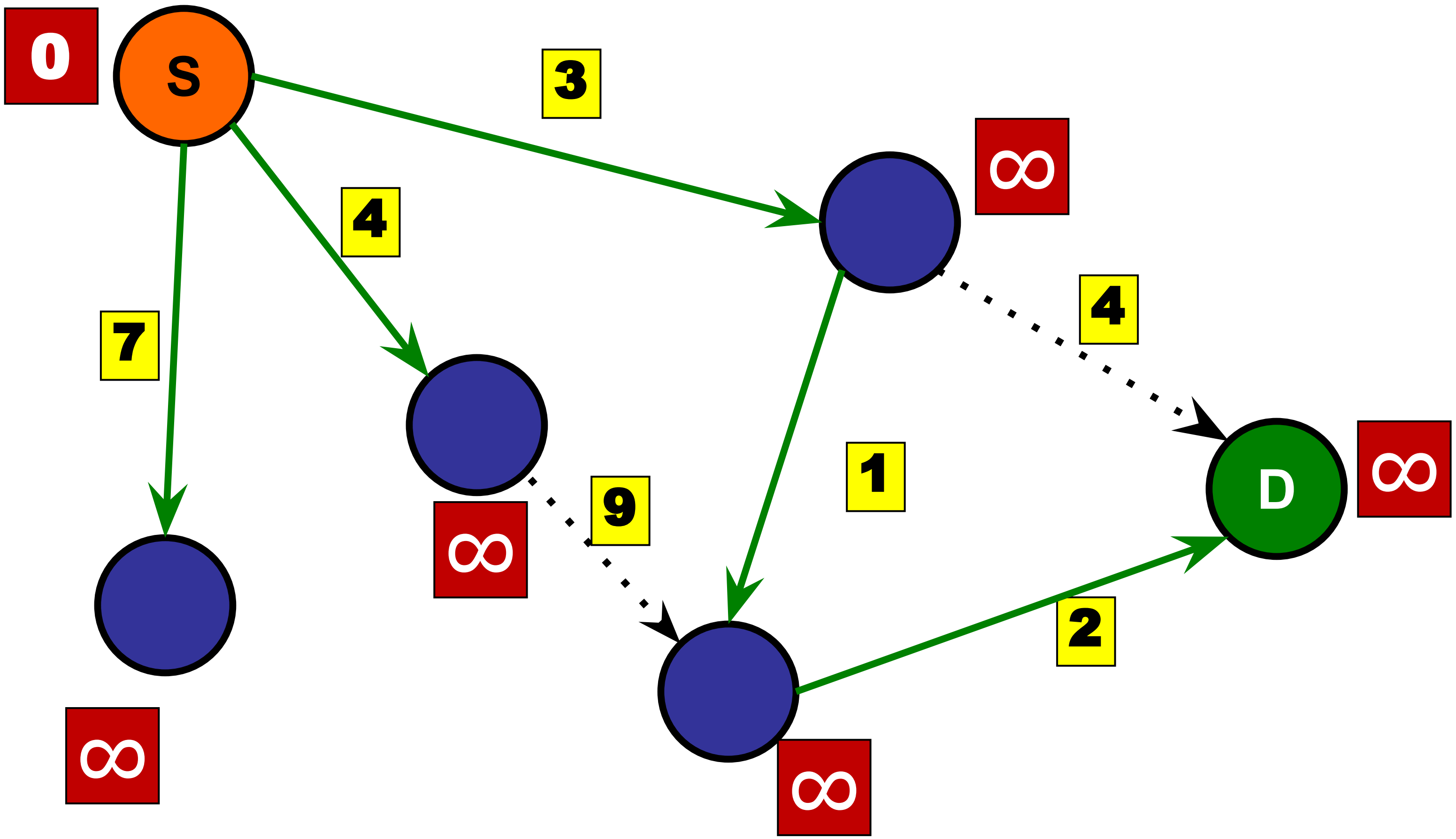
Idea:

Consider the shortest path tree of the graph:



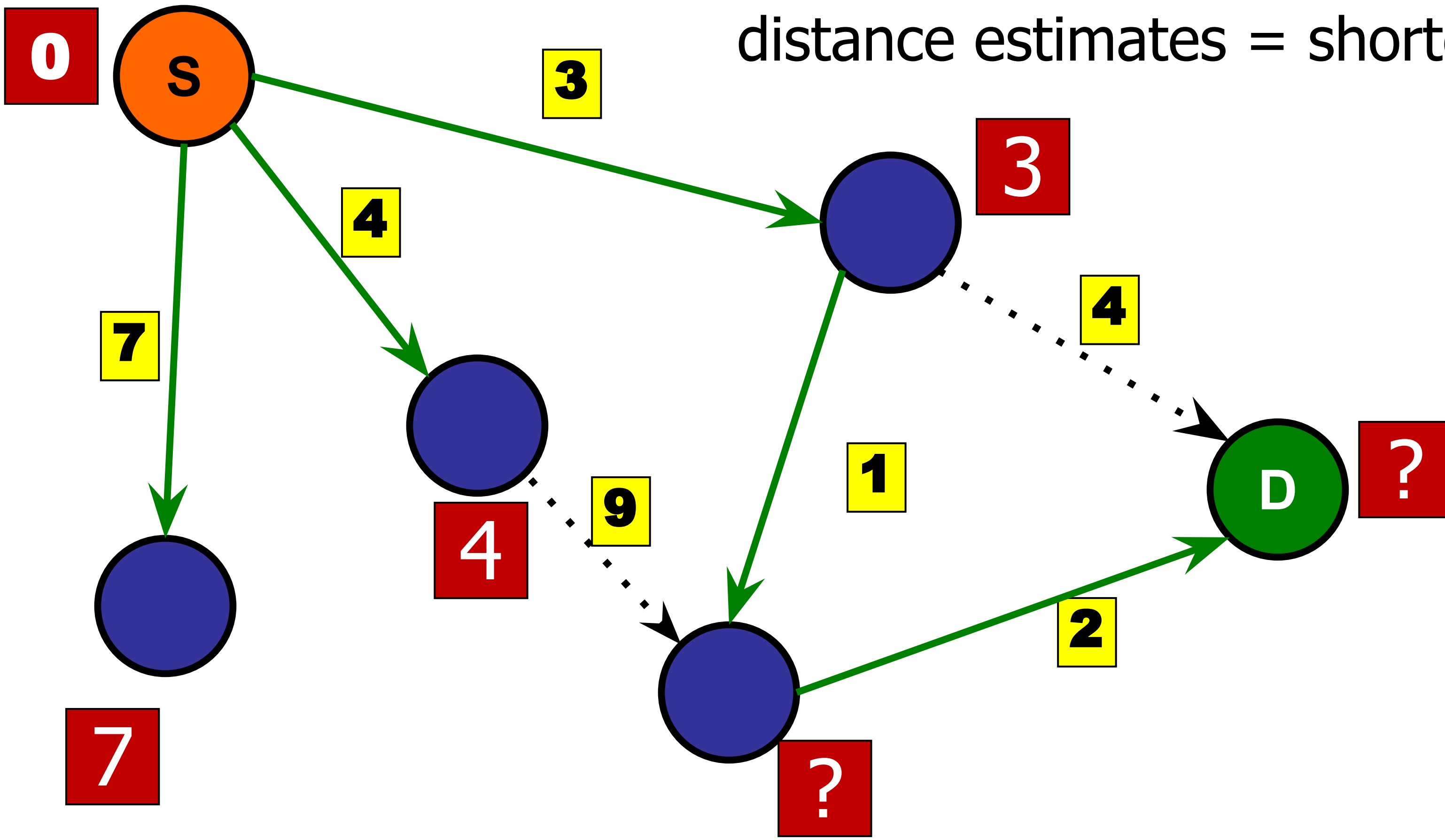
Idea:

What can we say about the distance estimates after one round of path relaxation over the edges?



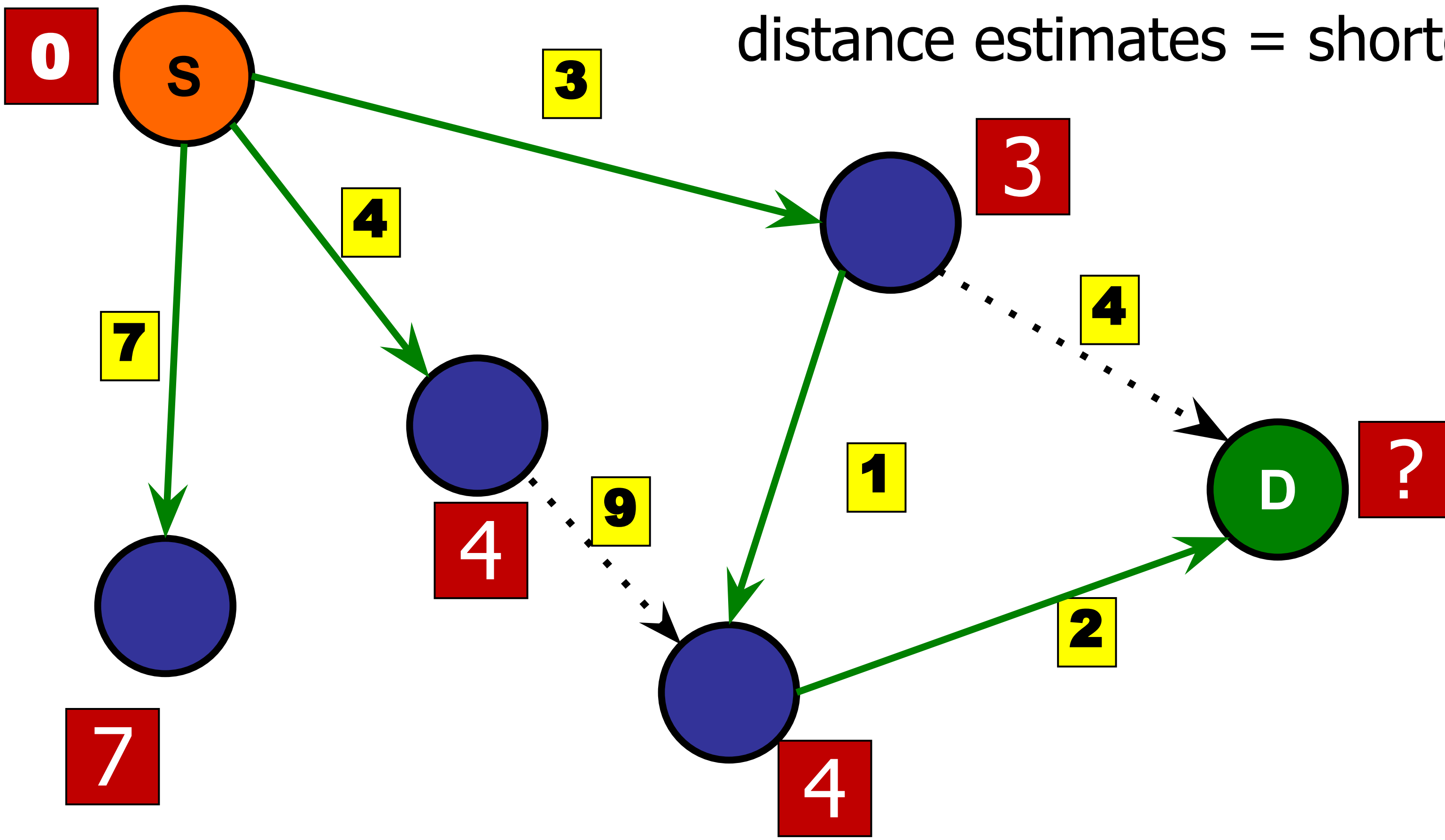
Idea:

After 1 round of relaxation, the nodes that are 1 hop away on the shortest path tree, have their distance estimates = shortest dist



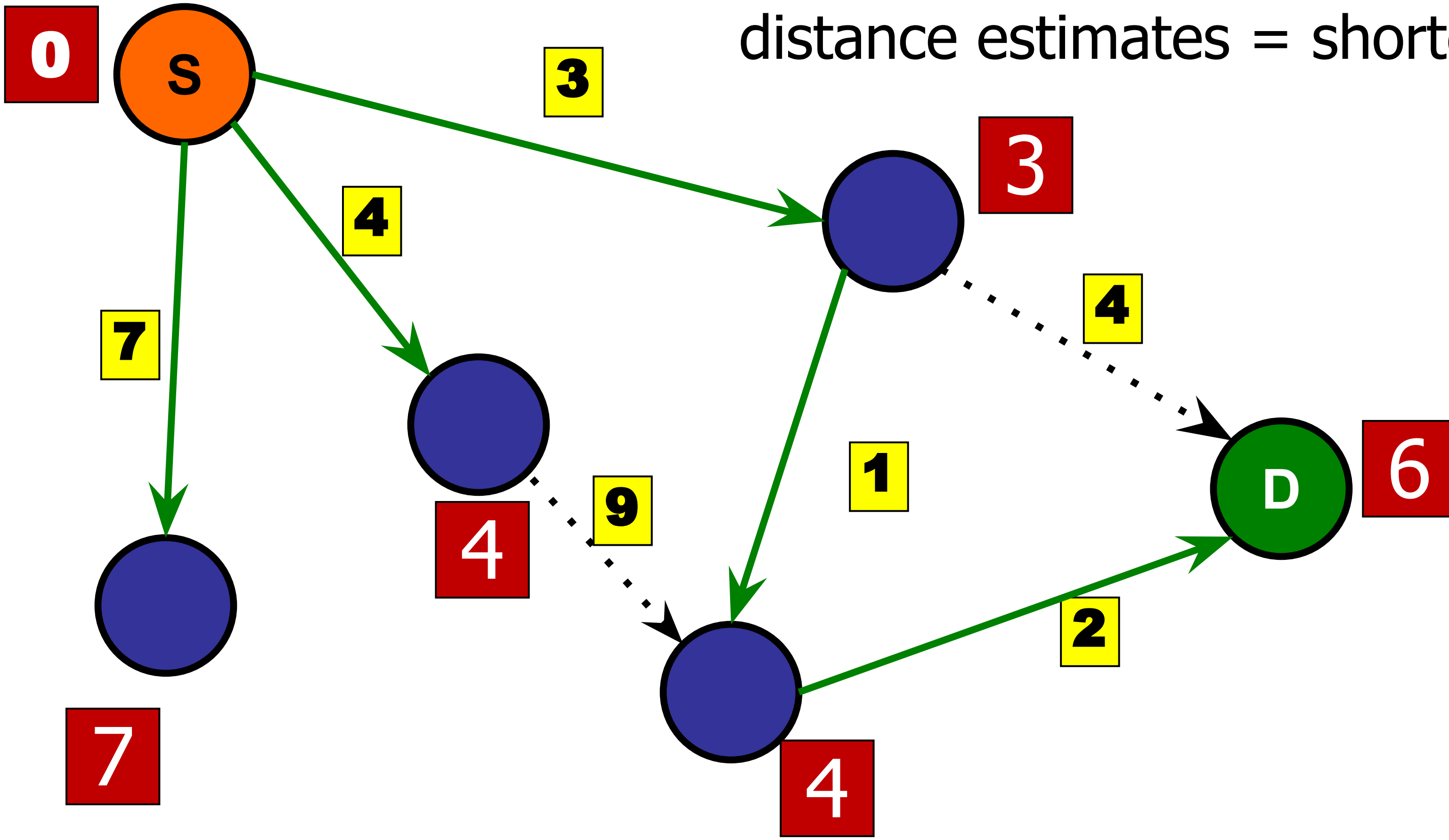
Idea:

After 2 rounds of relaxation, the nodes that are 2 hop away on the shortest path tree, have their distance estimates = shortest dist



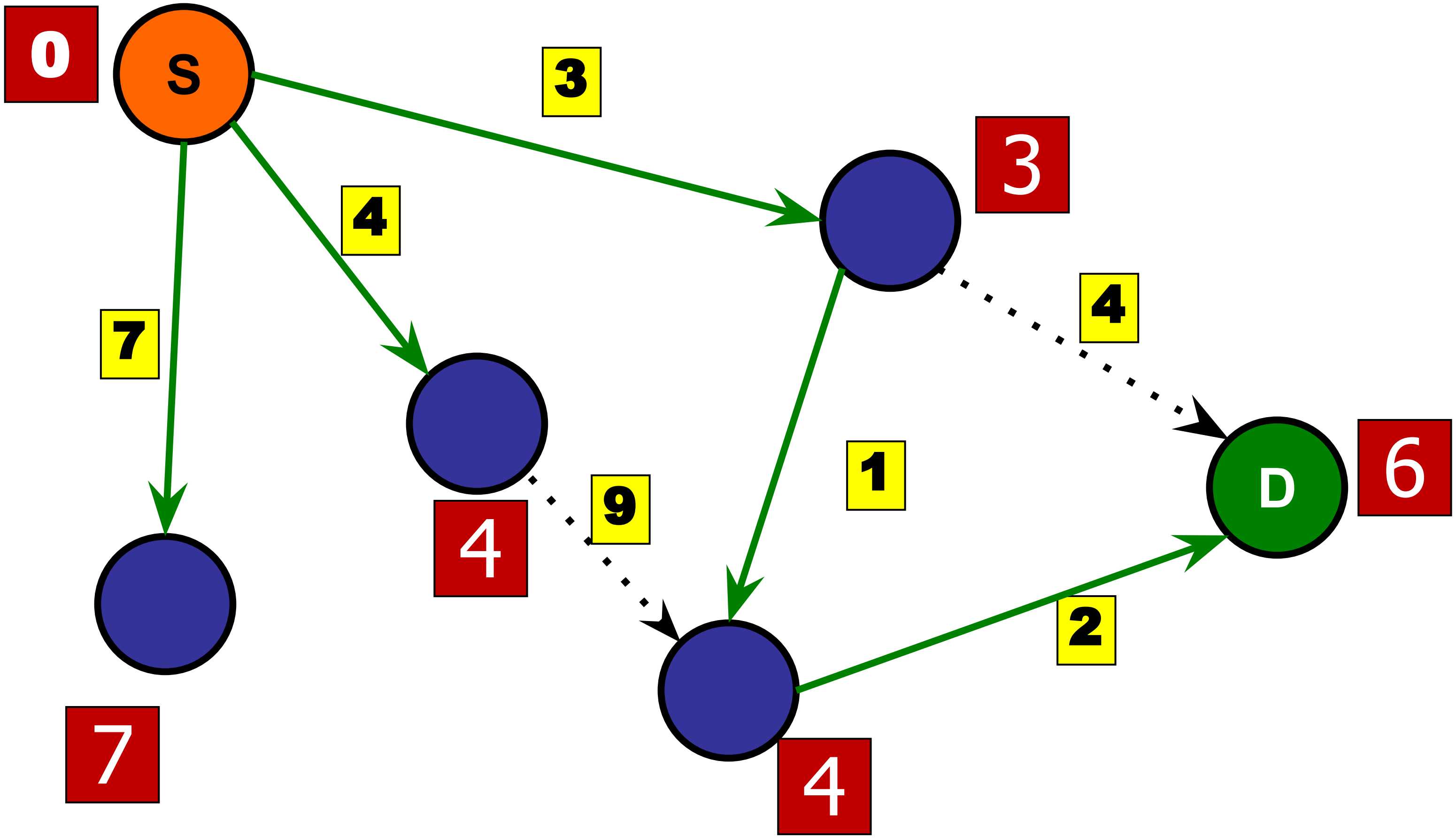
Idea:

After 3 rounds of relaxation, the nodes that are 3 hop away on the shortest path tree, have their distance estimates = shortest dist



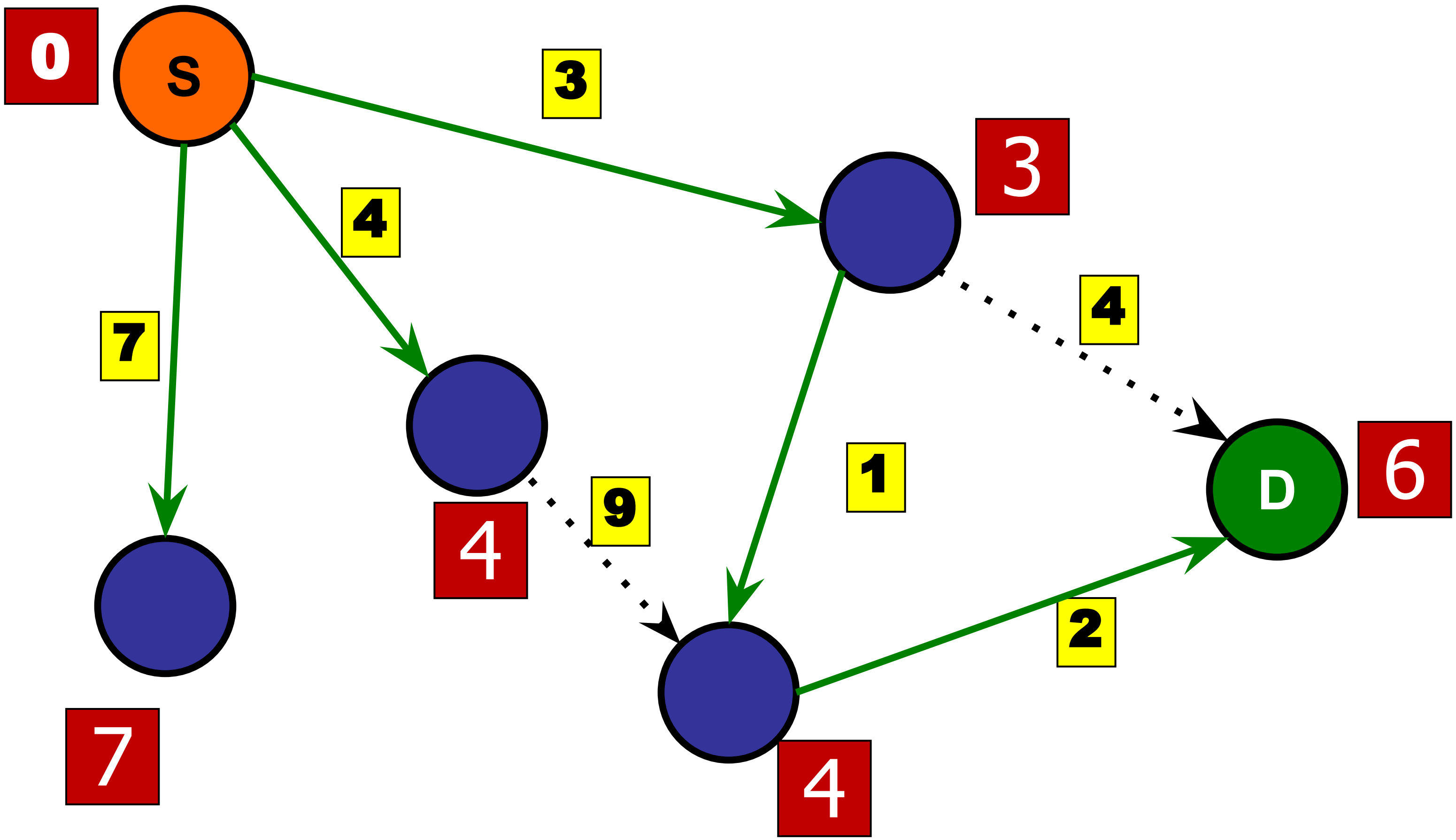
Idea:

To be clear: It takes **at most** *i* rounds to compute the correct distance that are *i* hops away on the shortest path tree



Idea:

Corollary: Since every node has to be at most |V| - 1 hops away from the source node s, we just need to run |V| - 1 rounds.



Bellman-Ford:

Pseudocode:

```
Set up distance estimate array dist
 for |V| - 1 iterations:
     for edge (u, v) in the graph G:
         relax(dist, u, v)
```

What is the time complexity of the given algorithm?

1. $O(V + E)$
2. $O(VE)$
3. $O(V^2)$
4. $O(E^2)$

Pseudocode:

```
Set up distance estimate array dist
 for |V| - 1 iterations:
     for edge (u, v) in the graph G:
         relaxed = relax(dist, u, v)
```

Bellman-Ford

Claim:

If after a round, the distance estimates don’t change, we have found the shortest distances for all nodes.

Bellman-Ford

If after a round, the distance estimates don’t change, we have found the shortest distances for all nodes.

Intuition:

Let’s say **before** we ran a round of relaxations, and we started with distance array D1.

It didn’t change **after** the round of relaxations. So even if we ran even more iterations (up until all |V| - 1 of them), nothing will change.

Bellman-Ford

If after a round, the distance estimates don't change, we have found the shortest distances for all nodes.

Intuition:

Let's say **before** we ran a round of relaxations, and we started with distance array D1.

It didn't change **after** the round of relaxations. So even if we ran even more iterations (up until all |V| - 1 of them), nothing will change.

<u>Early termination!</u>

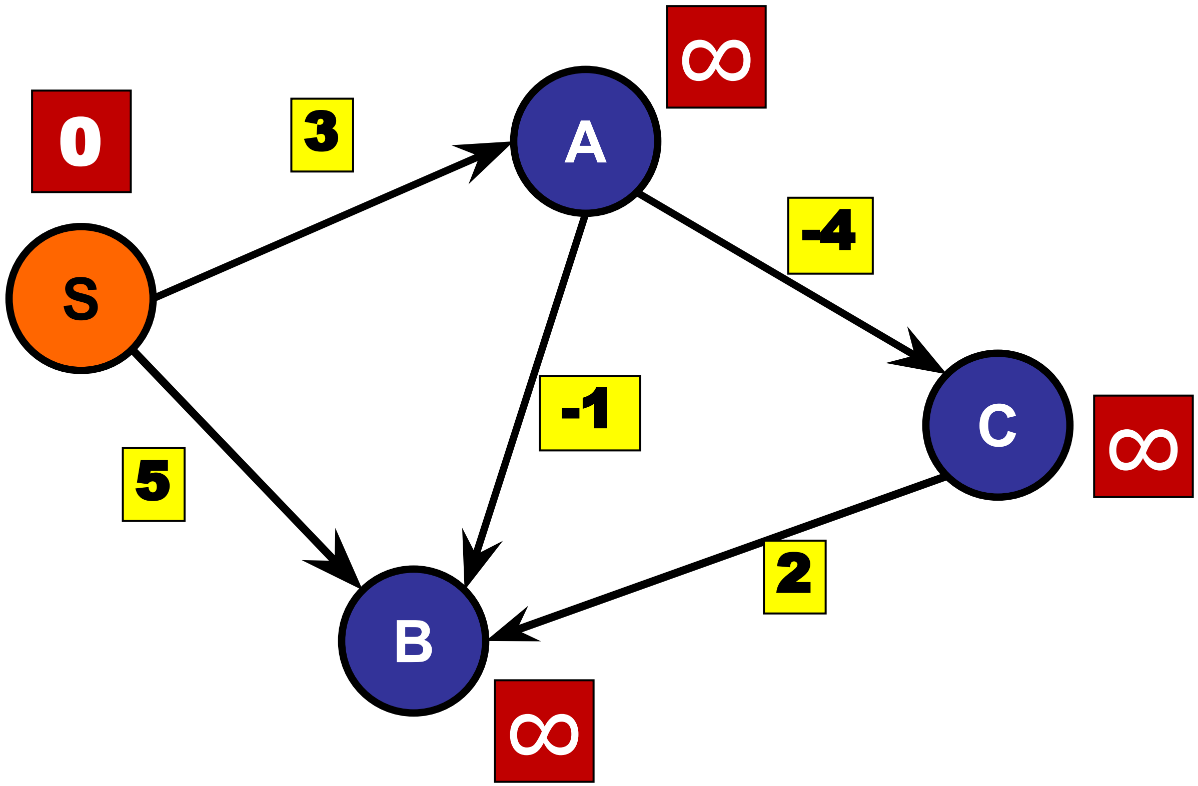
Bellman-Ford:

Pseudocode:

```
Set up distance estimate array dist
 for |V| - 1 iterations:
     for edge (u, v) in the graph G:
         relaxed |= relax(dist, u, v)
     if not relaxed: // no estimates have changed
         break
```

Bellman-Ford

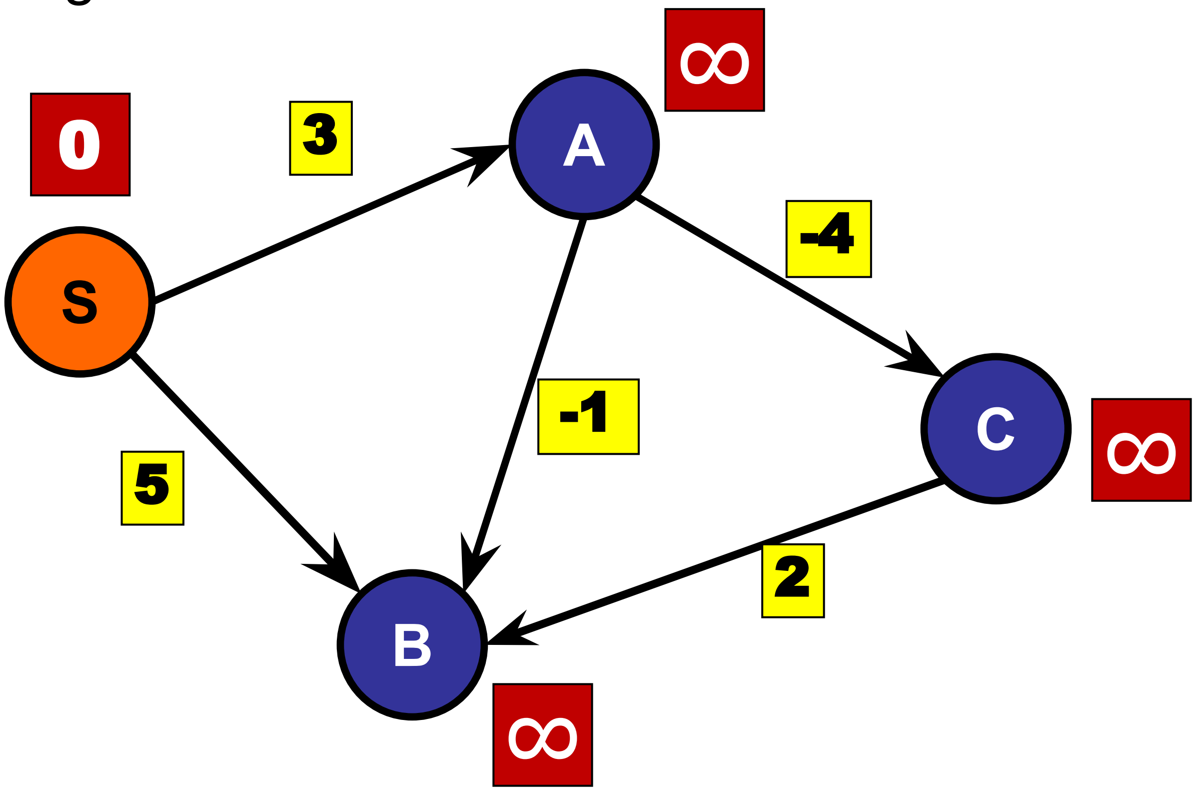
What if edges have negative weight?



Bellman-Ford

What if edges have negative weight?

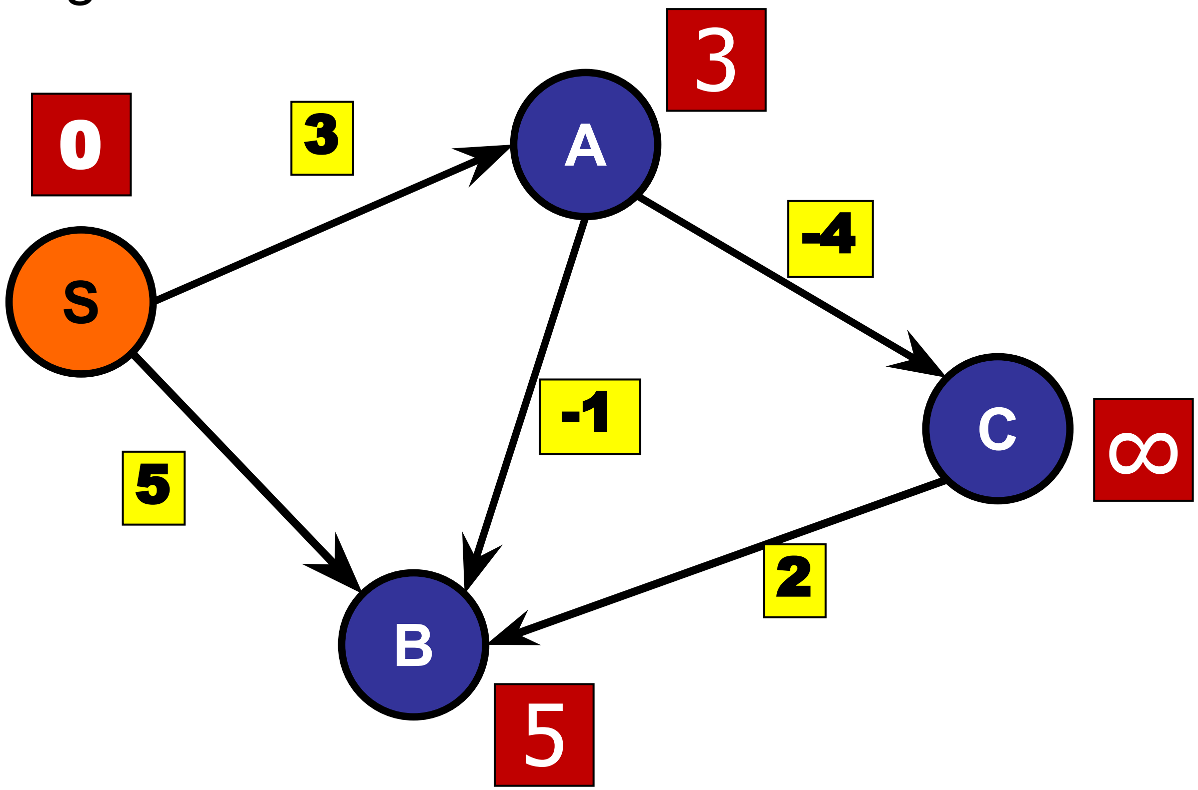
Assume ordering was:
(A, C)
(C, B)
(A, B)
(S, A)
(S, B)



Bellman-Ford

What if edges have negative weight?

Assume ordering was:
(A, C)
(C, B)
(A, B)
(S, A)
(S, B)

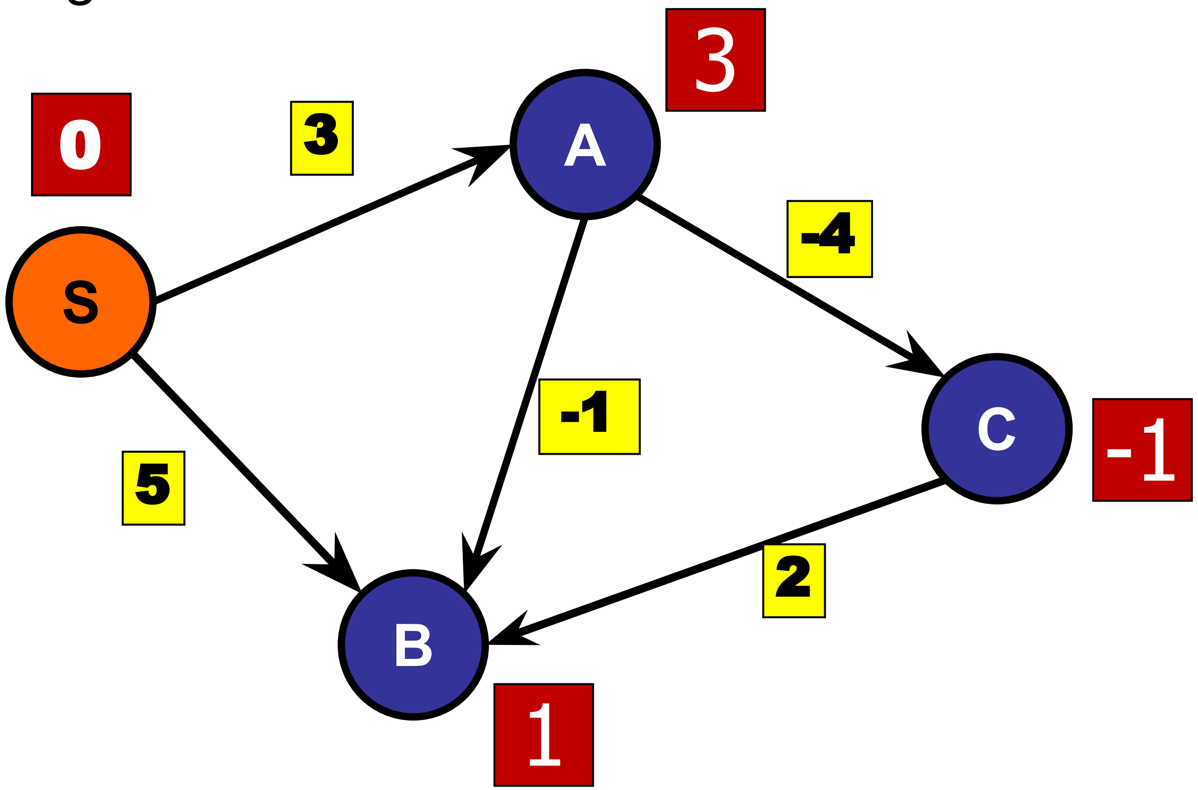


After 1 round.

Bellman-Ford

What if edges have negative weight?

Assume ordering was:
(A, C)
(C, B)
(A, B)
(S, A)
(S, B)

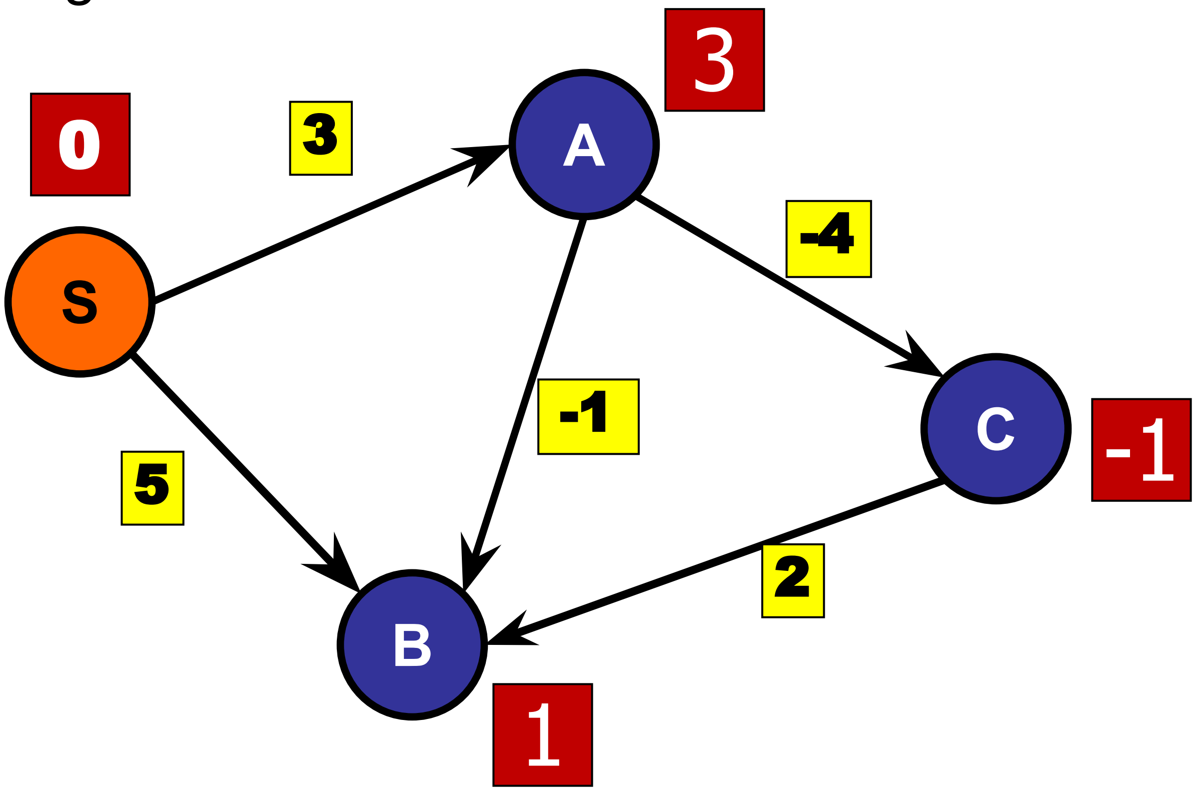


After 2 rounds.

Bellman-Ford

What if edges have negative weight?

Assume ordering was:
(A, C)
(C, B)
(A, B)
(S, A)
(S, B)

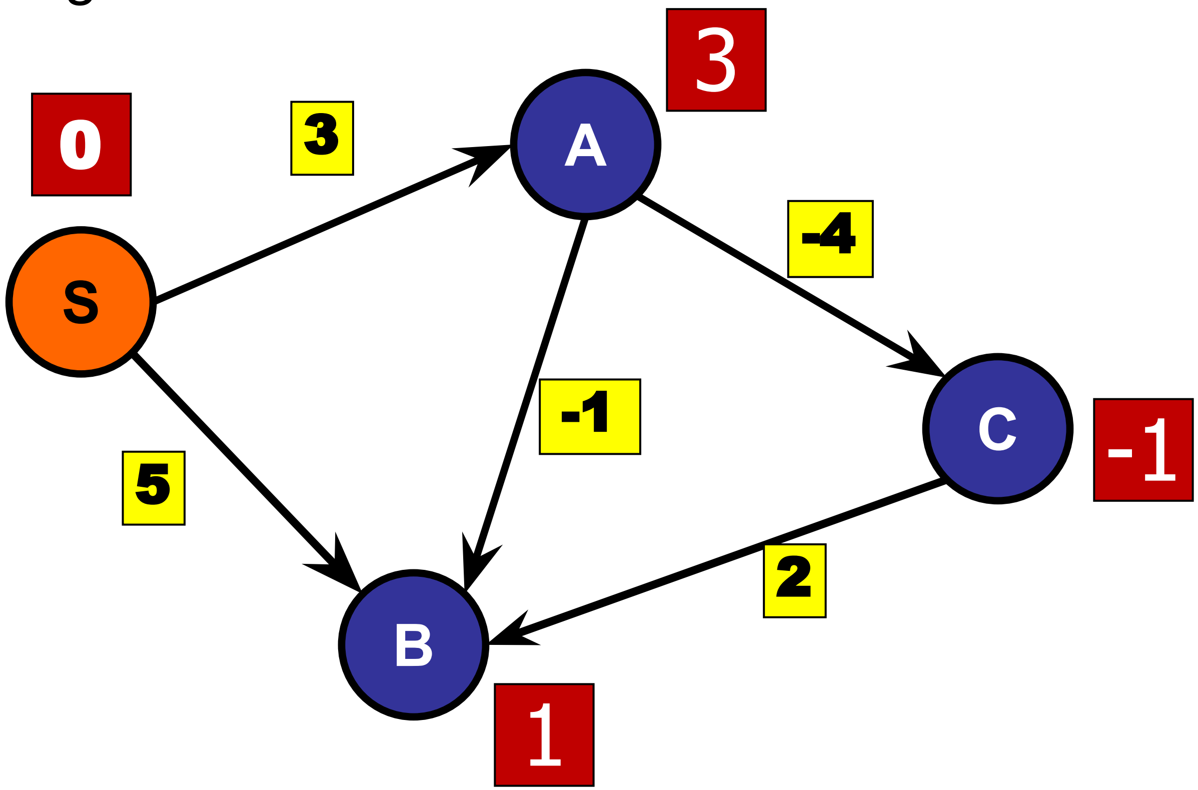


After 3 rounds. No changes already.

Bellman-Ford

What if edges have negative weight?

Assume ordering was:
(A, C)
(C, B)
(A, B)
(S, A)
(S, B)

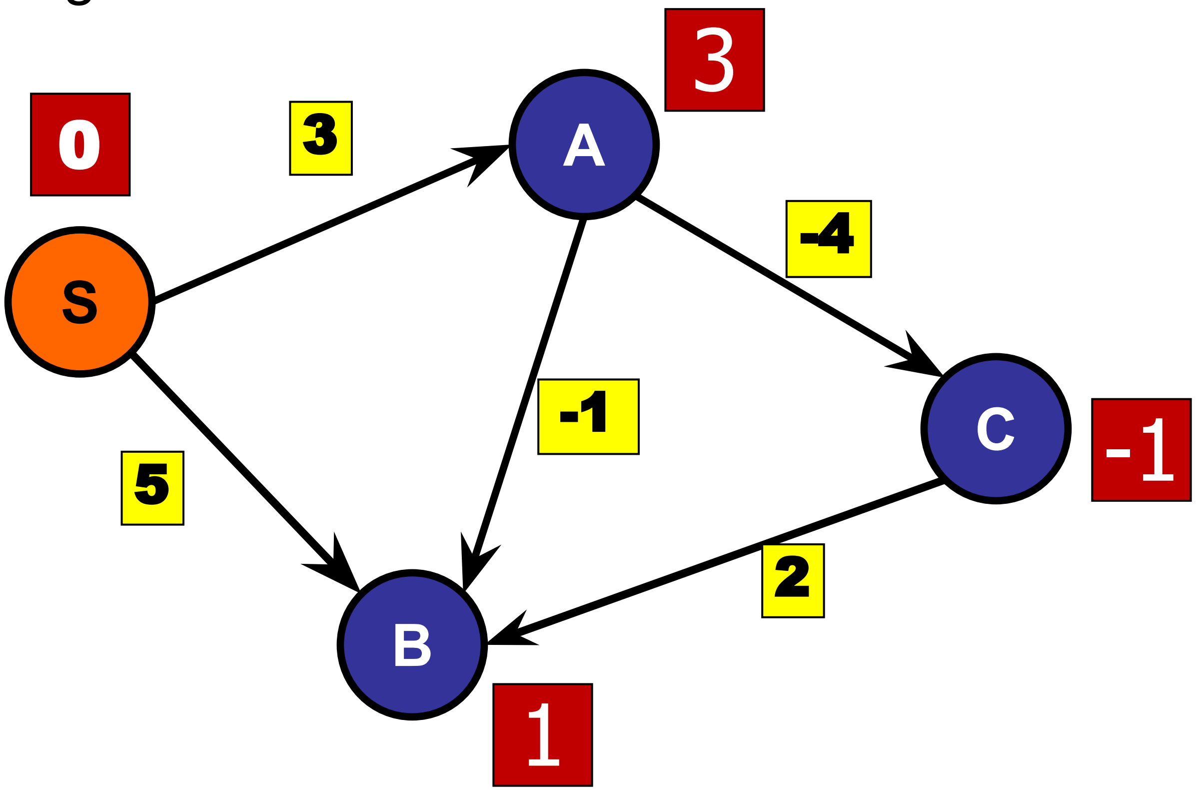


After 3 rounds. Shortest distances found!

Bellman-Ford

What if edges have negative weight?

Assume ordering was:
(A, C)
(C, B)
(A, B)
(S, A)
(S, B)

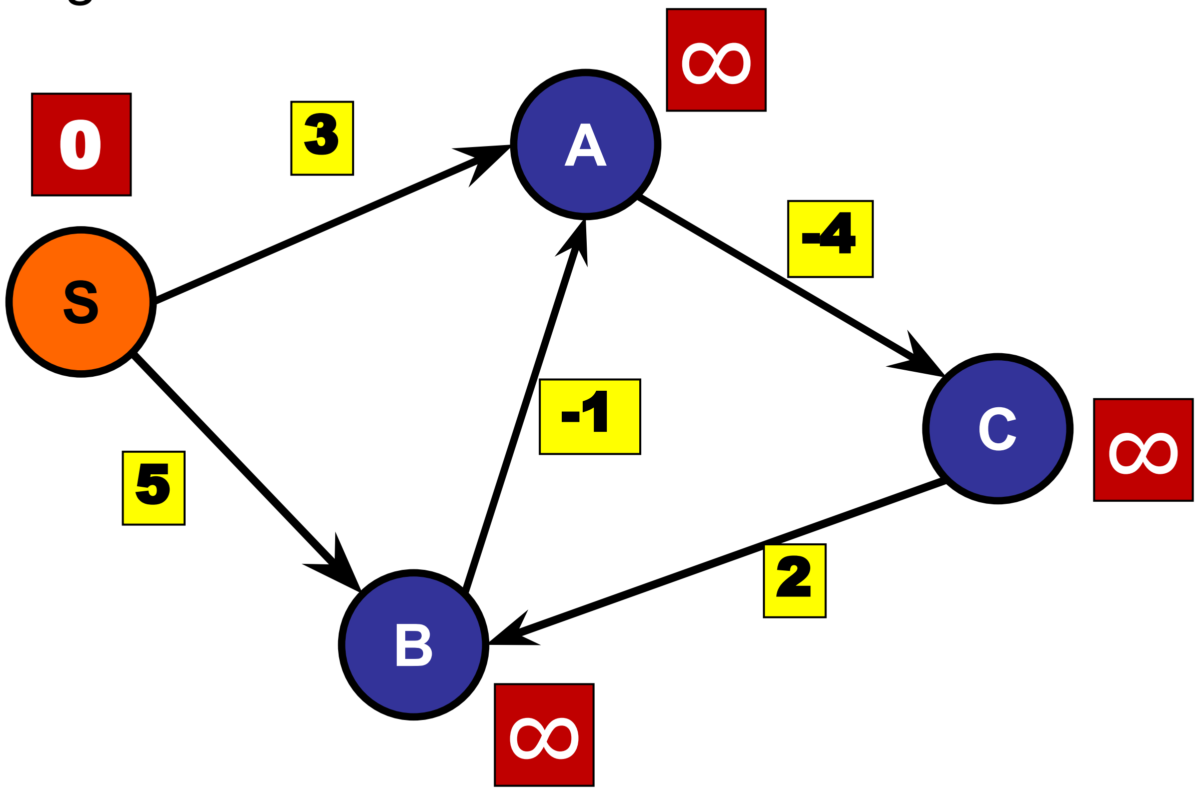


No problem!

Bellman-Ford

What if the graph has a negative cycle?

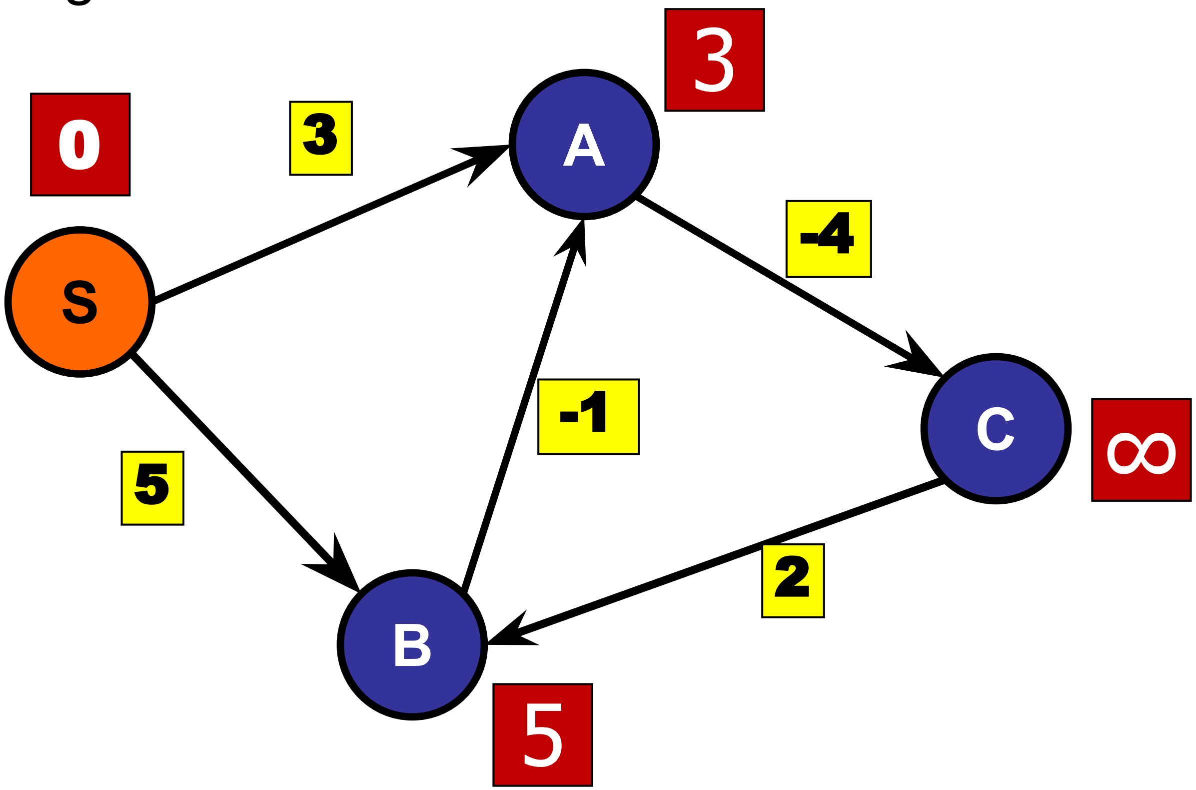
Assume ordering was:
(A, C)
(C, B)
(B, A)
(S, A)
(S, B)



Bellman-Ford

What if the graph has a negative cycle?

Assume ordering was:
(A, C)
(C, B)
(B, A)
(S, A)
(S, B)

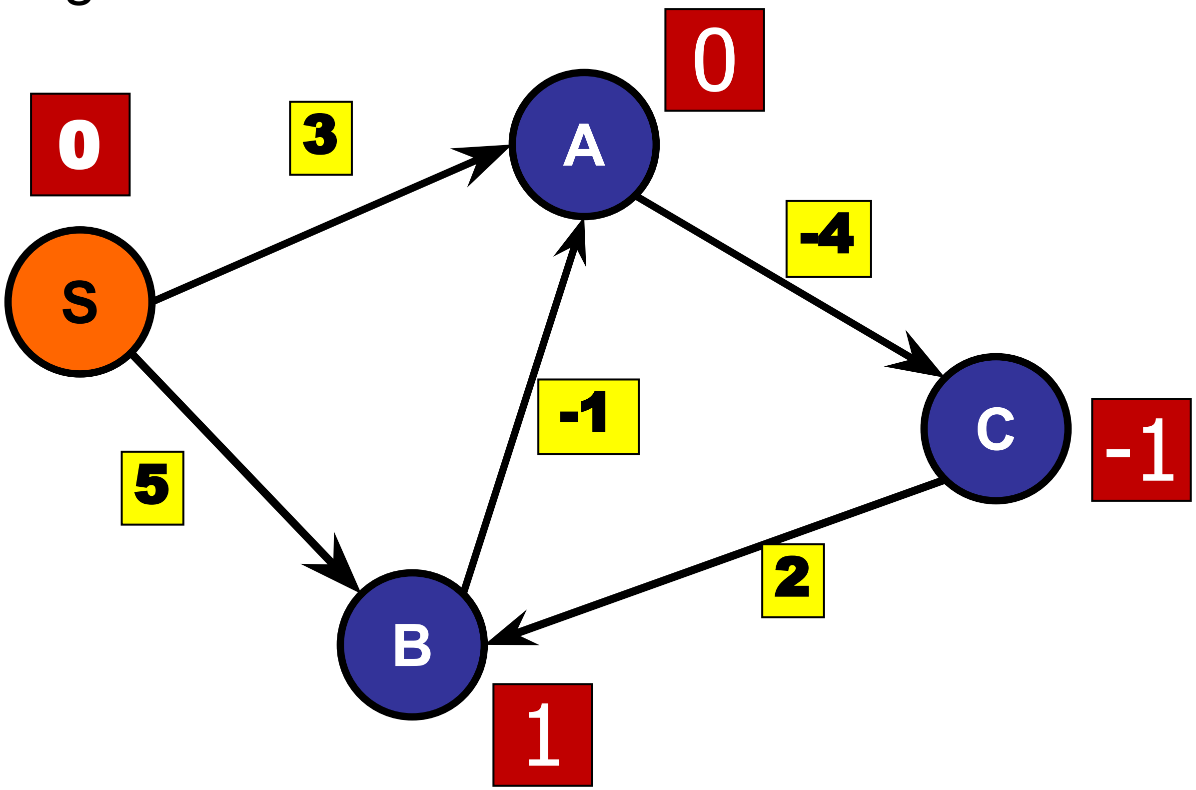


After 1 rounds.

Bellman-Ford

What if the graph has a negative cycle?

Assume ordering was:
(A, C)
(C, B)
(B, A)
(S, A)
(S, B)

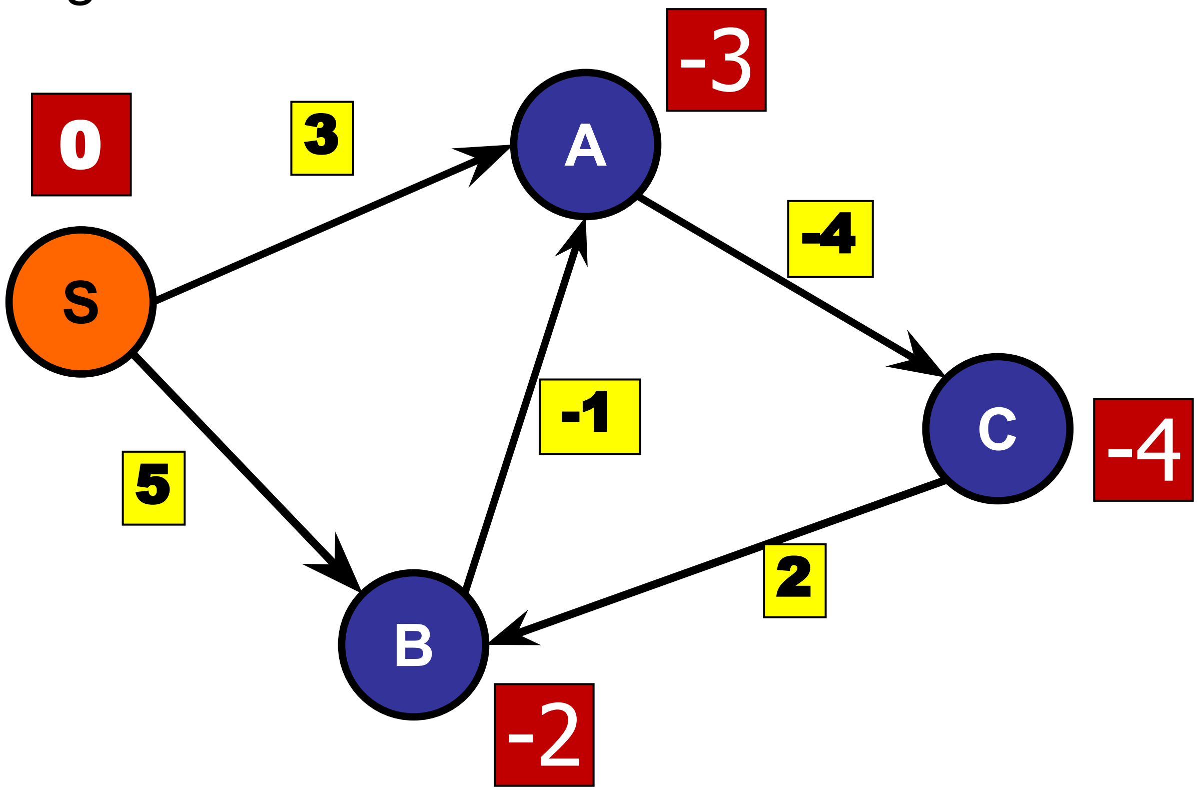


After 2 rounds.

Bellman-Ford

What if the graph has a negative cycle?

Assume ordering was:
(A, C)
(C, B)
(B, A)
(S, A)
(S, B)

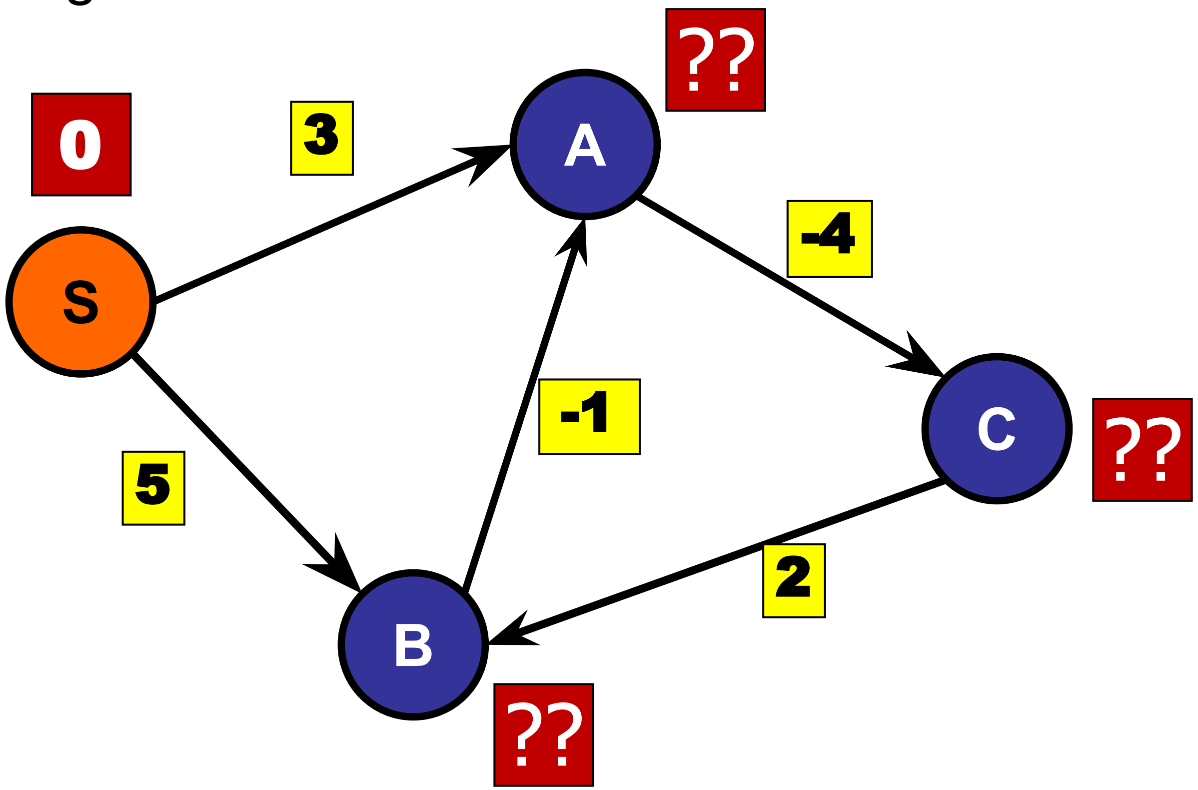


After 3 rounds.

Bellman-Ford

What if the graph has a negative cycle?

Assume ordering was:
(A, C)
(C, B)
(B, A)
(S, A)
(S, B)

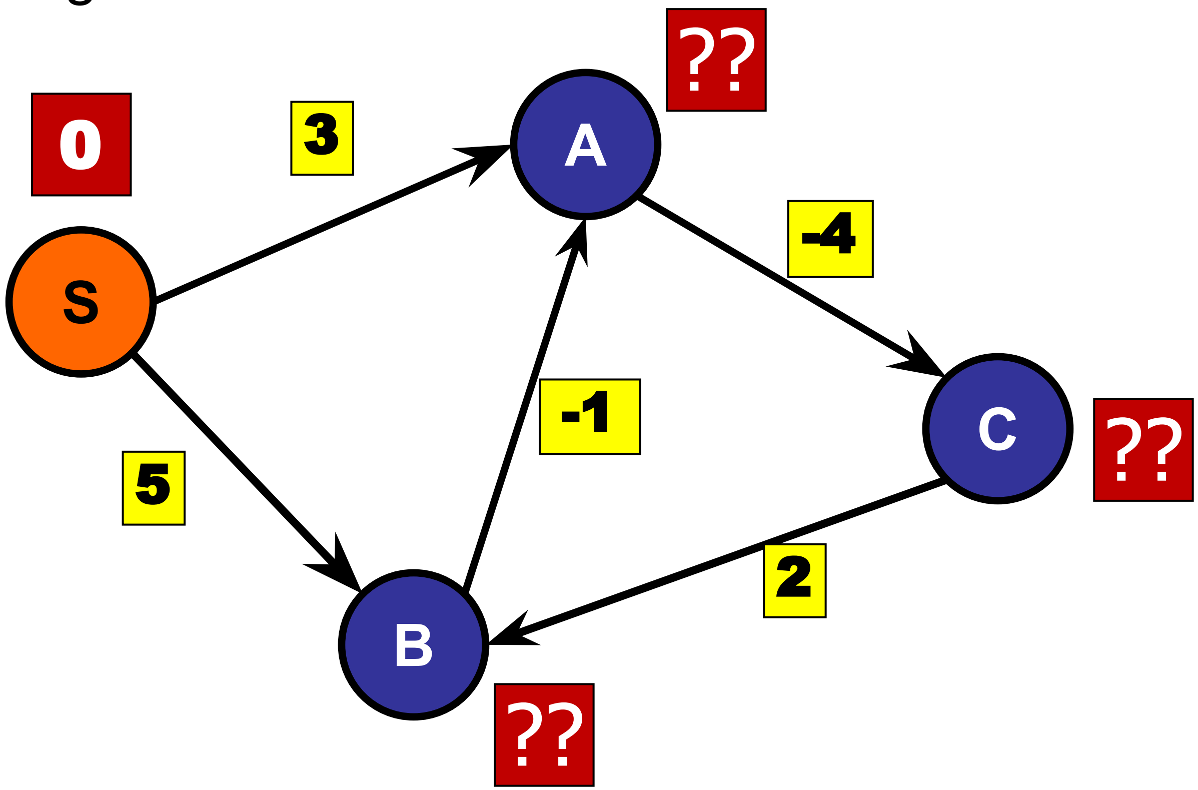


After ???? rounds.

Bellman-Ford

What if the graph has a negative cycle?

Assume ordering was:
(A, C)
(C, B)
(B, A)
(S, A)
(S, B)

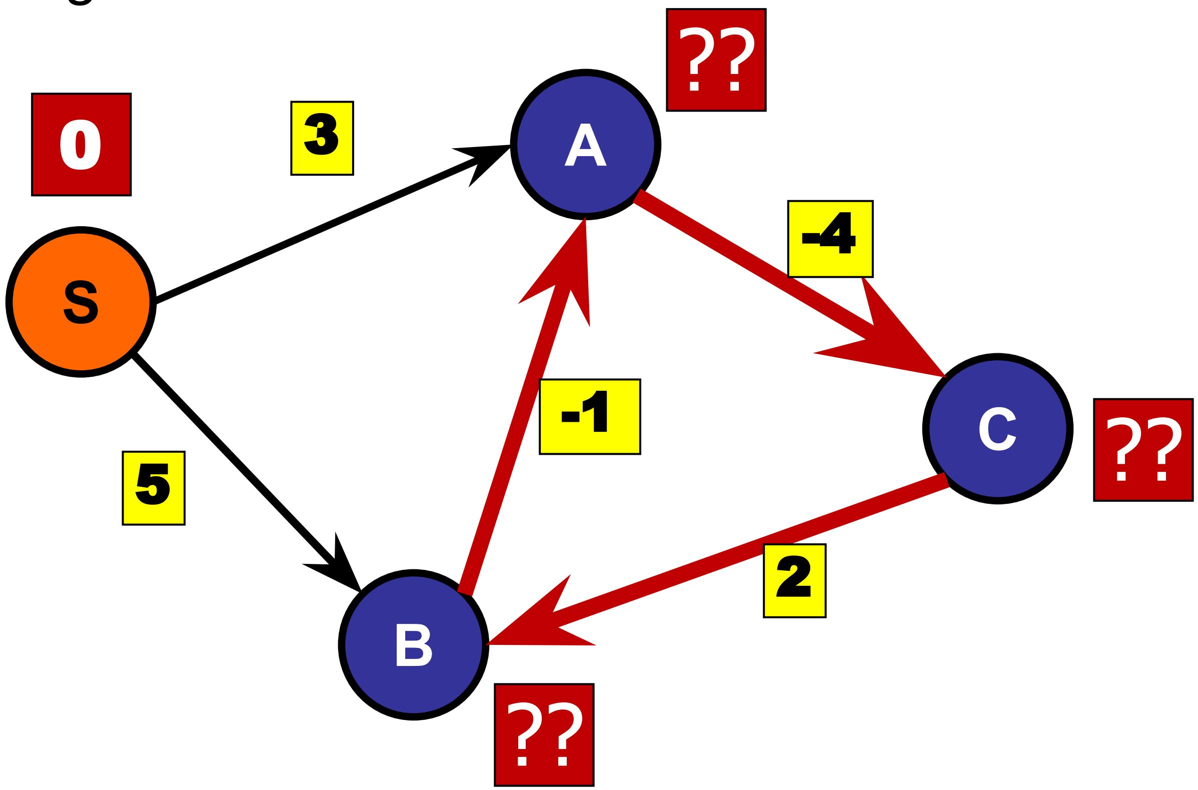


d(S,C) is infinitely negative!

Bellman-Ford

What if the graph has a negative cycle?

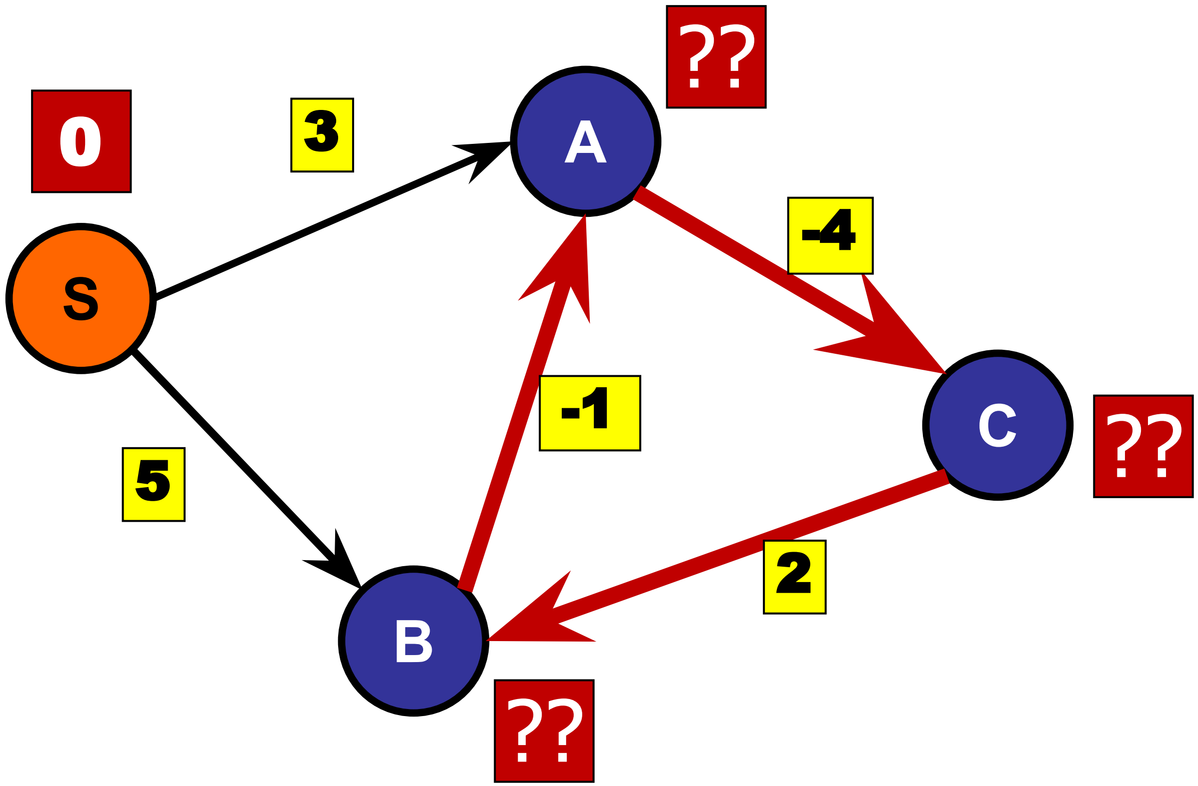
Assume ordering was:
(A, C)
(C, B)
(B, A)
(S, A)
(S, B)



Notice here that we don't even have a cycle made of all negative edges.

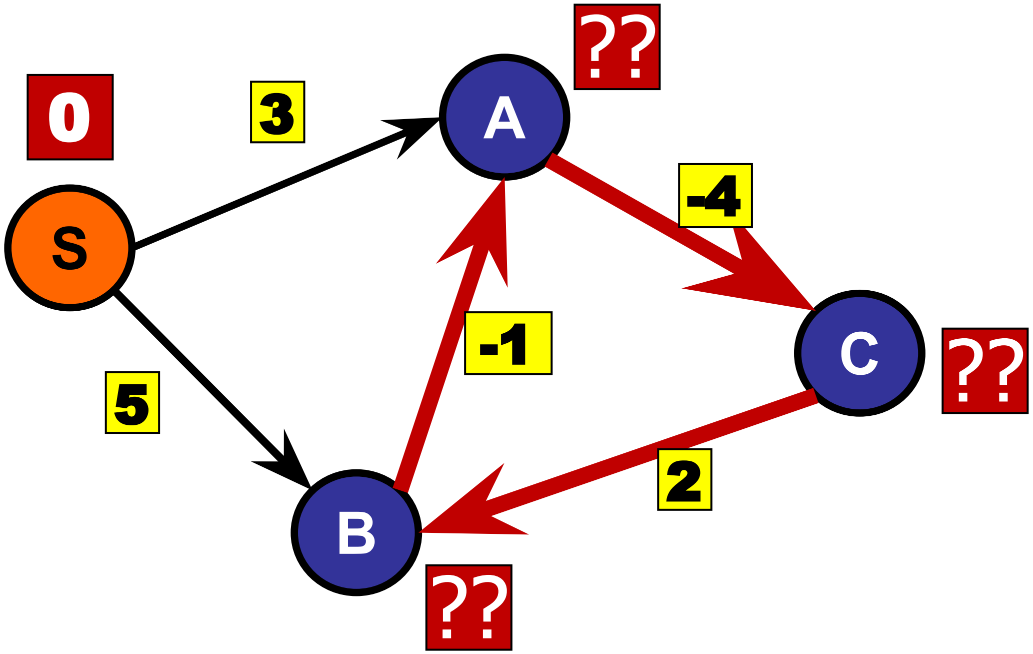
Detecting negative cycles

We know that any shortest paths should be found after |V| - 1 iterations.



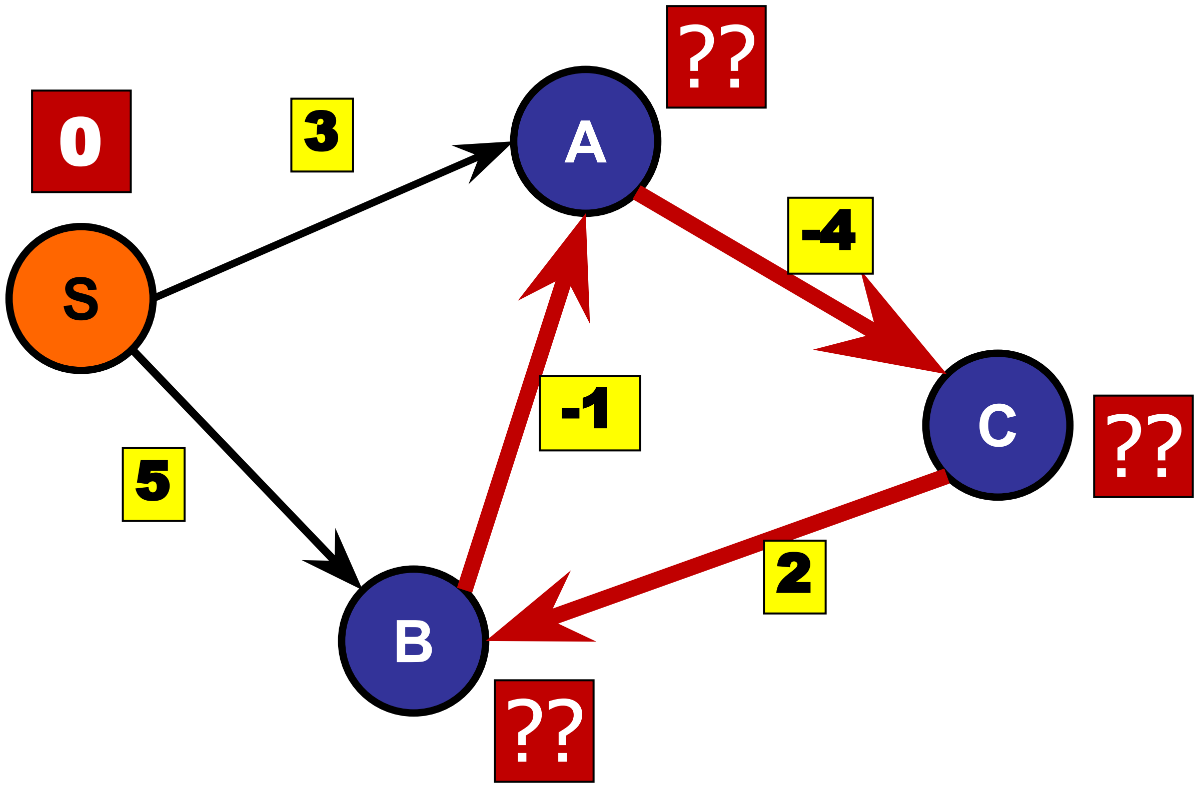
If there is a negative cycle, what happens to the distance estimates on the $|V|^{th}$ iteration?

1. Estimates remain unchanged
2. Some estimates will go down
3. Some estimates will go up



Detecting negative cycles

We know that any shortest paths should be found after |V| - 1 iterations.



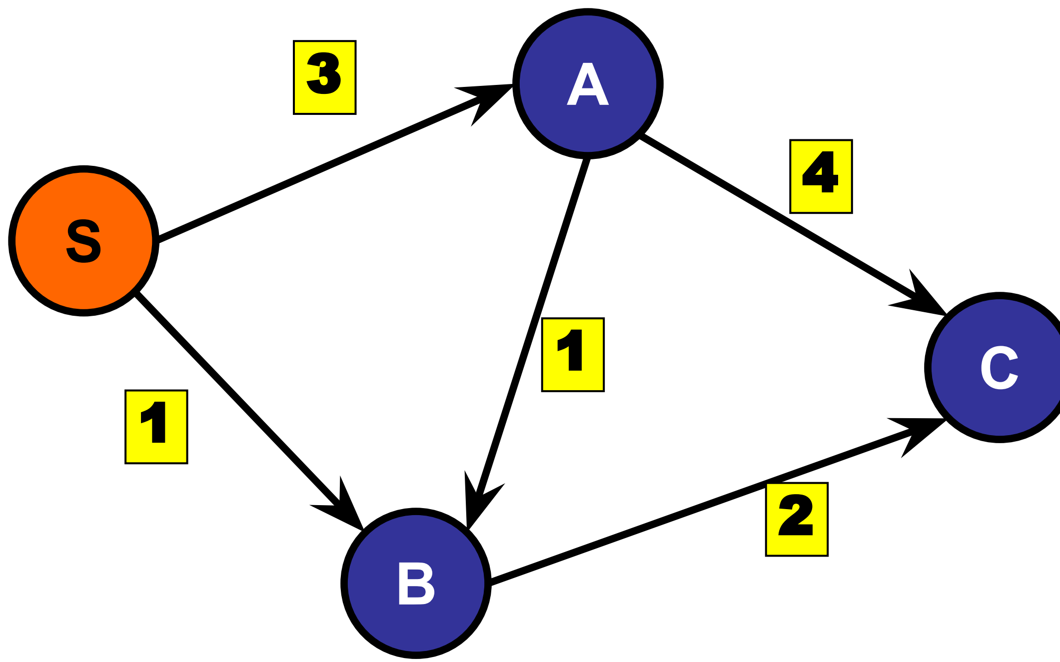
Run one more iteration, if any distance estimates go down, we know there is a negative cycle

Shortest Paths (Recall)

Key idea: triangle inequality

$\delta(S, C) \leq \delta(S, A) + \delta(A, C)$

(Side Quiz: Does this also hold if our edge weights are negative?)



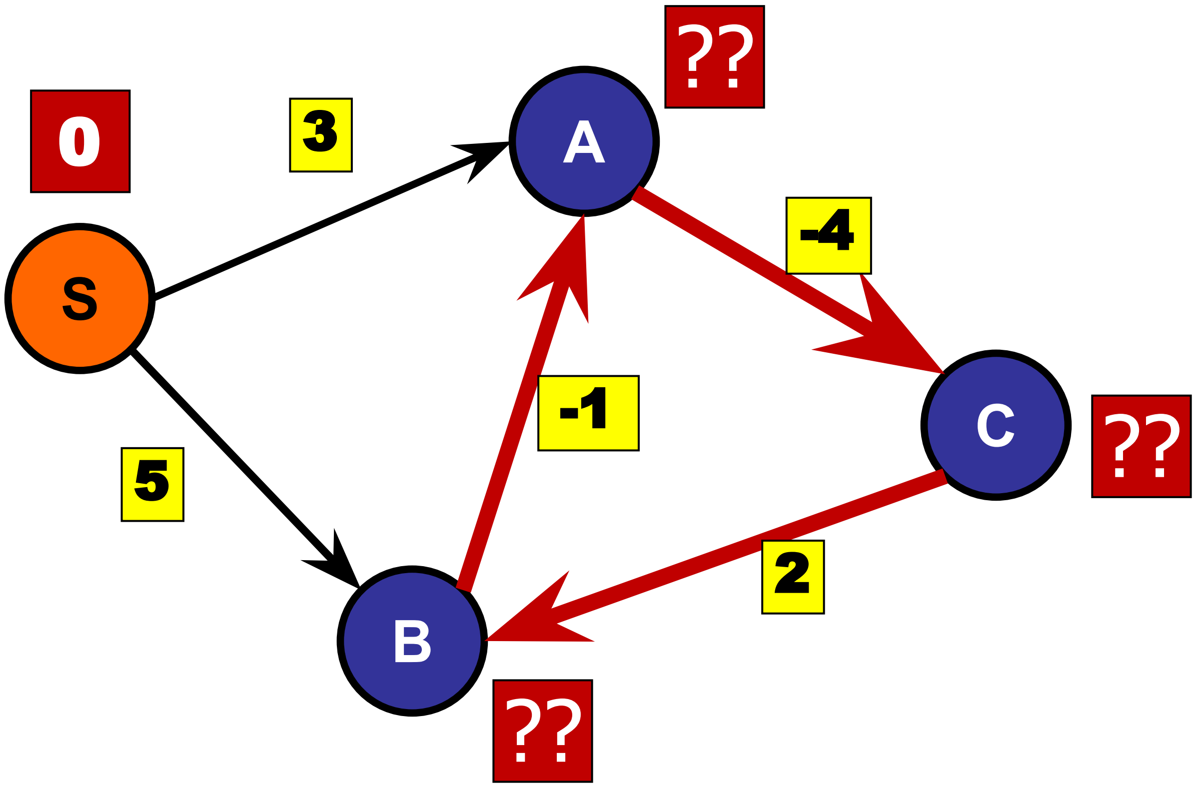
So Far:

- **Unweighted** graph
  - BFS
  - O(V + E)

- **Weighted** graphs with <u>non-negative edges</u>.
  - Dijkstra
  - O(E log V)

- **Weighted** graphs with <u>no negative cycles</u>.
  - Bellman-Ford
  - O(VE)

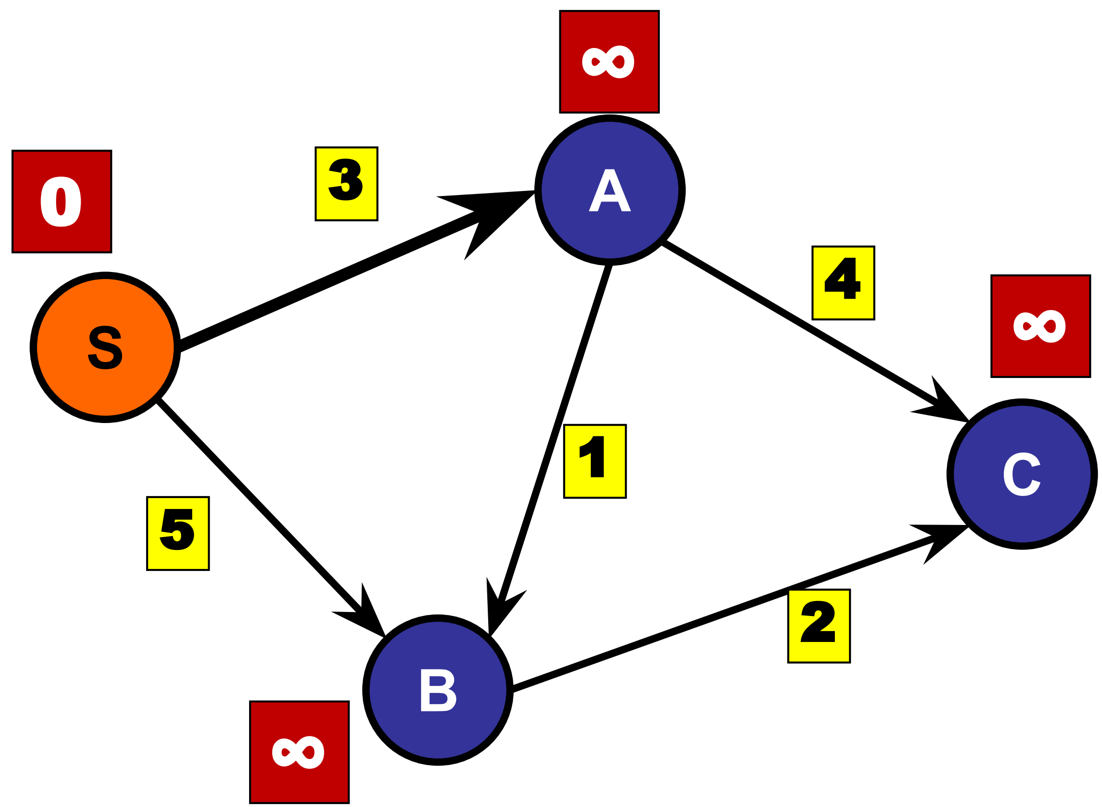
Not the end of the story:



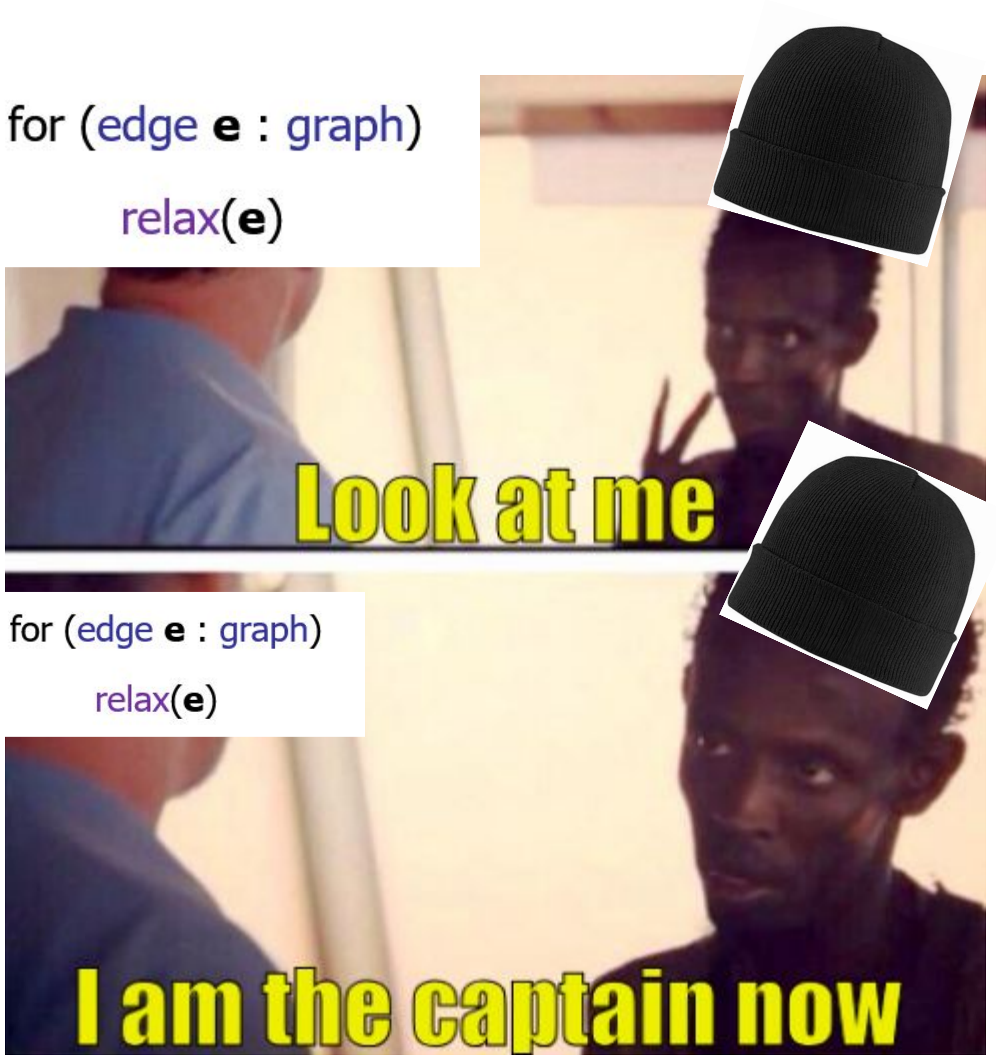
Previously:

Let's say we ran relax() based on the edges we have, in some arbitrary ordering.

What if we had control over this?

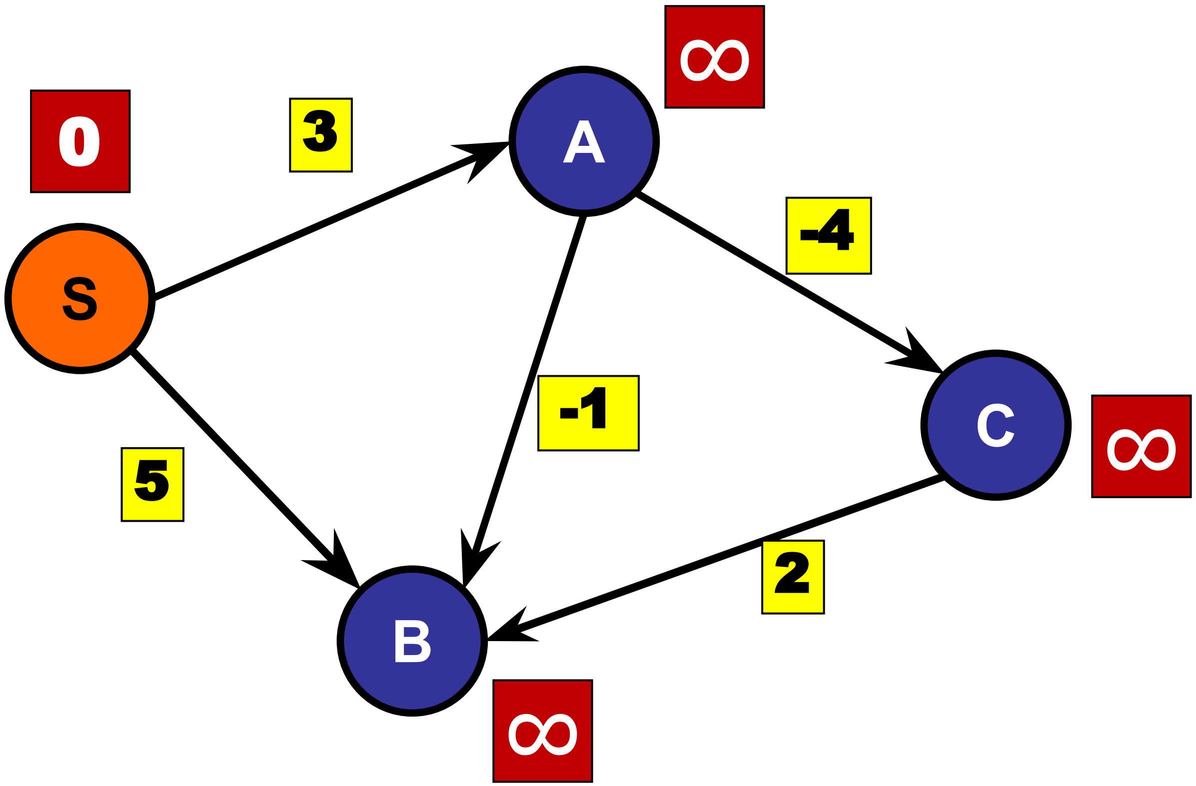


Changing the Ordering:



Bellman-Ford on DAG

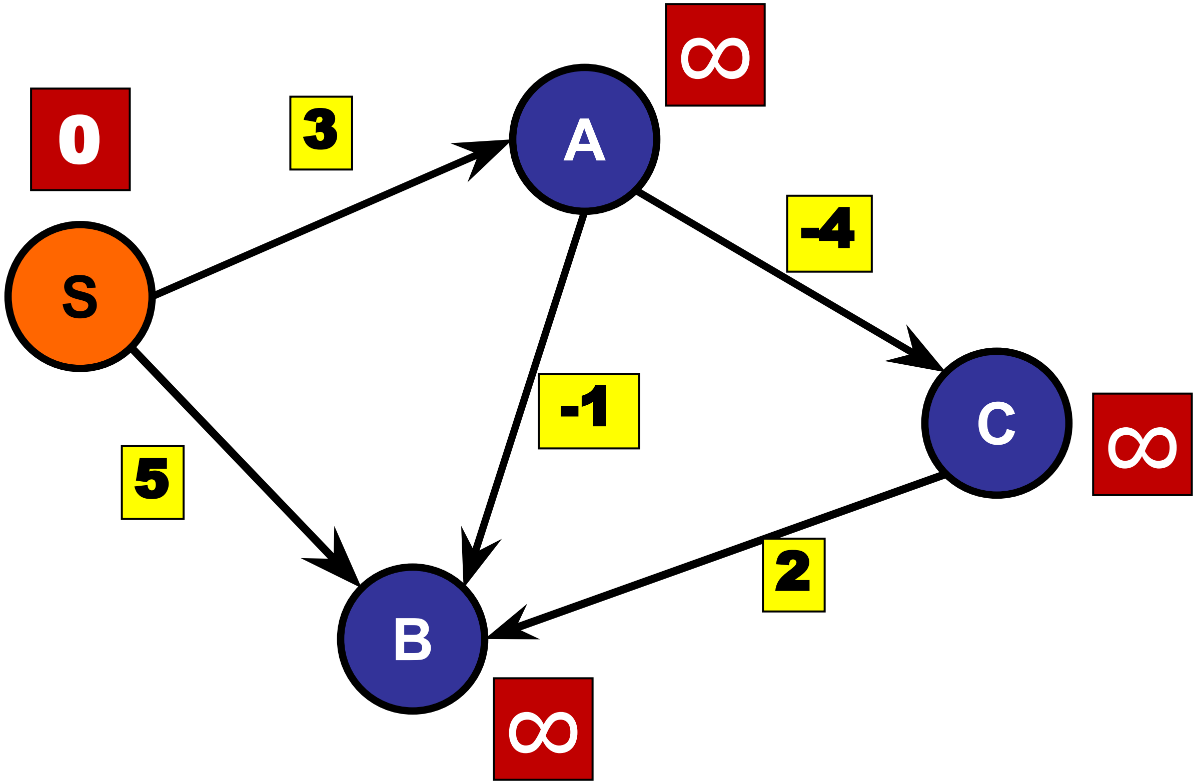
What happens if the graph is a DAG?



Bellman-Ford on DAG

What happens if the graph is a DAG?

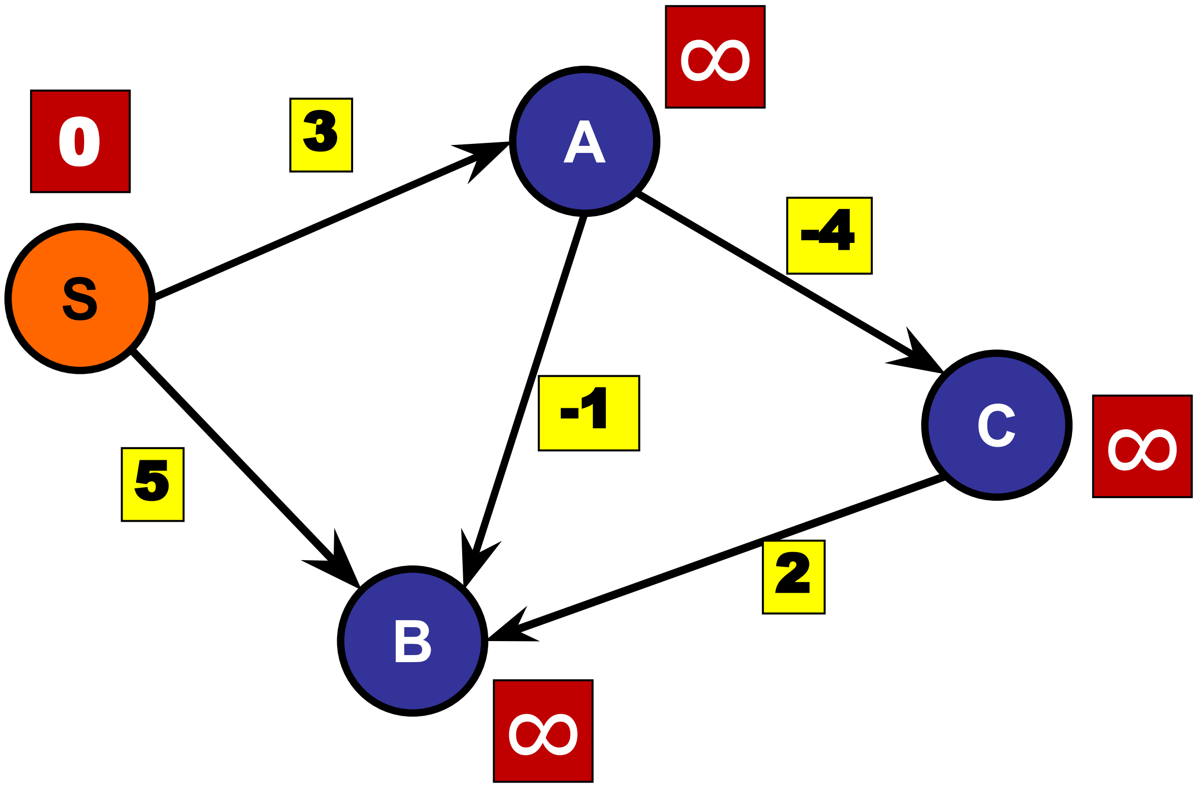
Toposort it first!



Bellman-Ford on DAG

What happens if the graph is a DAG?

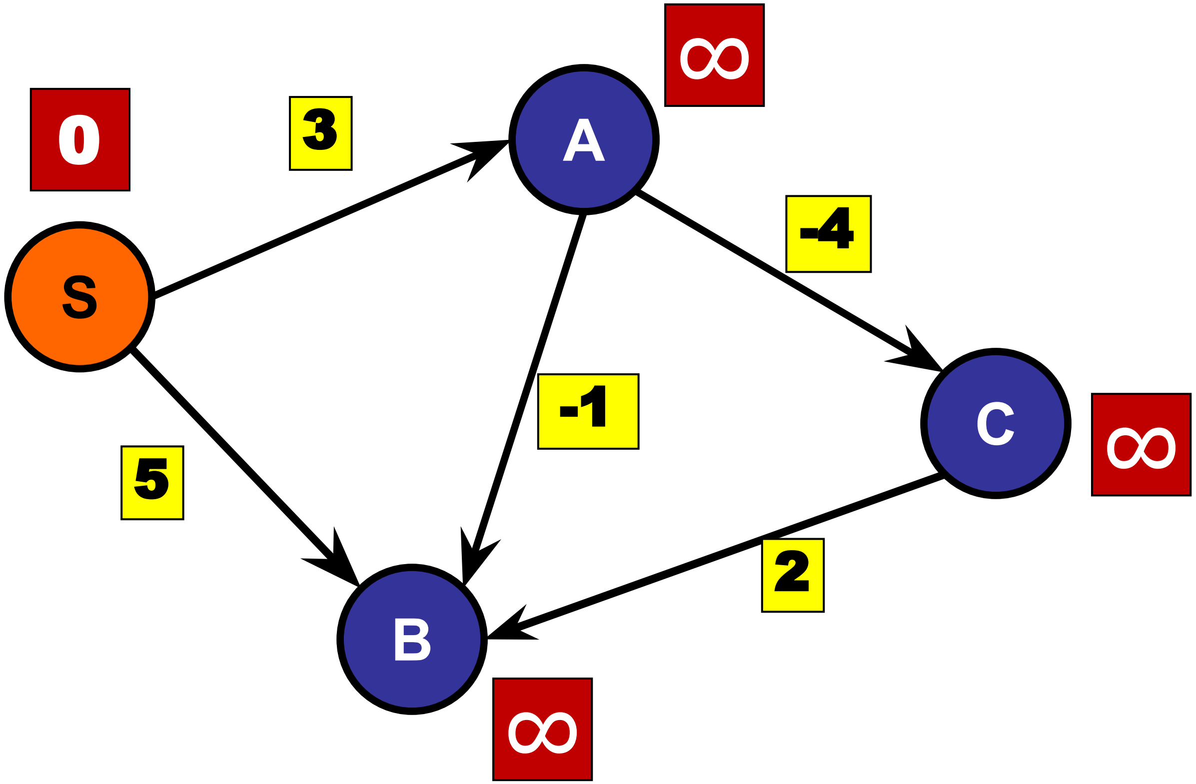
Toposort it first!



Toposorted order: S, A, C, B

Bellman-Ford on DAG

In topo-sort order:



Toposorted order: S, A, C, B

Bellman-Ford On DAG:

Pseudocode:

Set up distance estimate array dist
Get toposorted list of nodes topo_list

```
for u in topo_list: (from first to last)
    for neighbour v in u.neighbour_list:
        relax(dist, u, v)
```

What is the time complexity of this algorithm?

1. $O(V + E)$
2. $O(V^2)$
3. $O(VE)$
4. $O(E^2)$

Set up distance estimate array dist
Get toposorted list of nodes topo_list

for u in topo_list: (from first to last)

    for neighbour v in u.neighbour_list:

        relax(dist, u, v)

Bellman-Ford On DAG:

O(V + E)

Pseudocode:

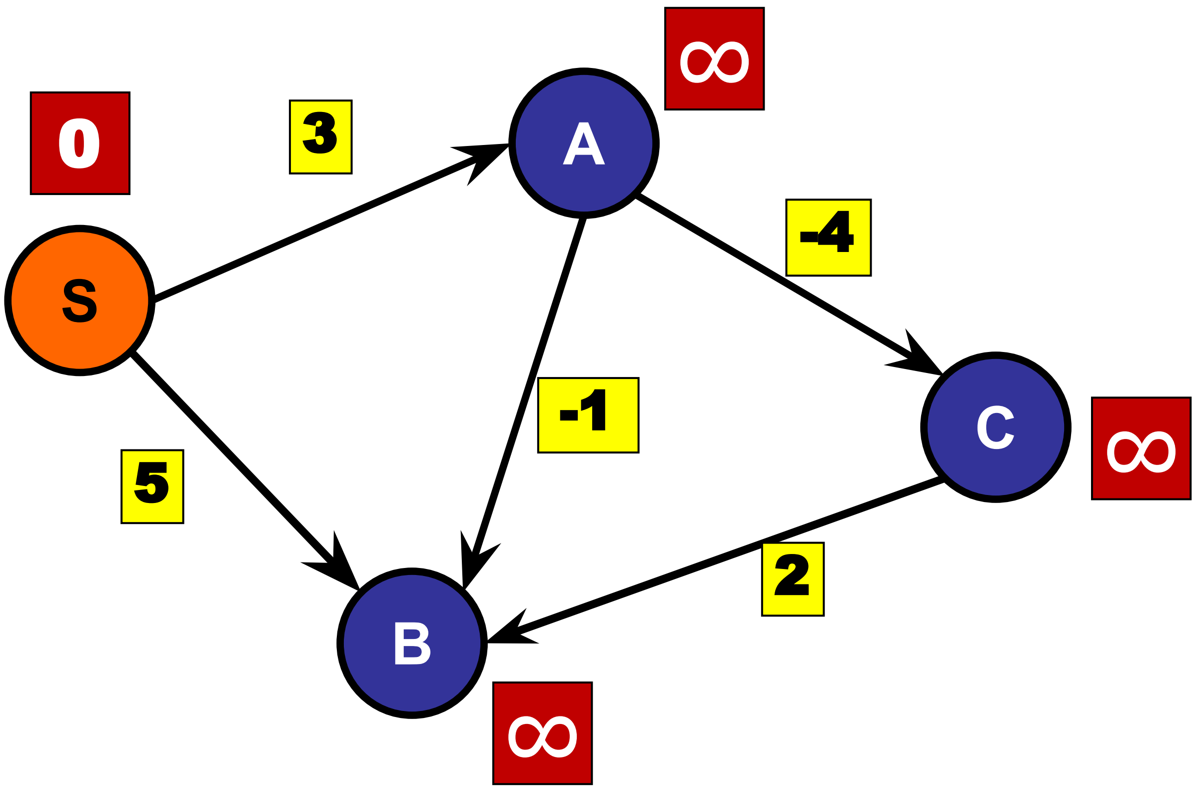
Set up distance estimate array dist
Get toposorted list of nodes topo_list

```
for u in topo_list: (from first to last)
    for neighbour v in u.neighbour_list:
        relax(dist, u, v)
```

O(V + E)

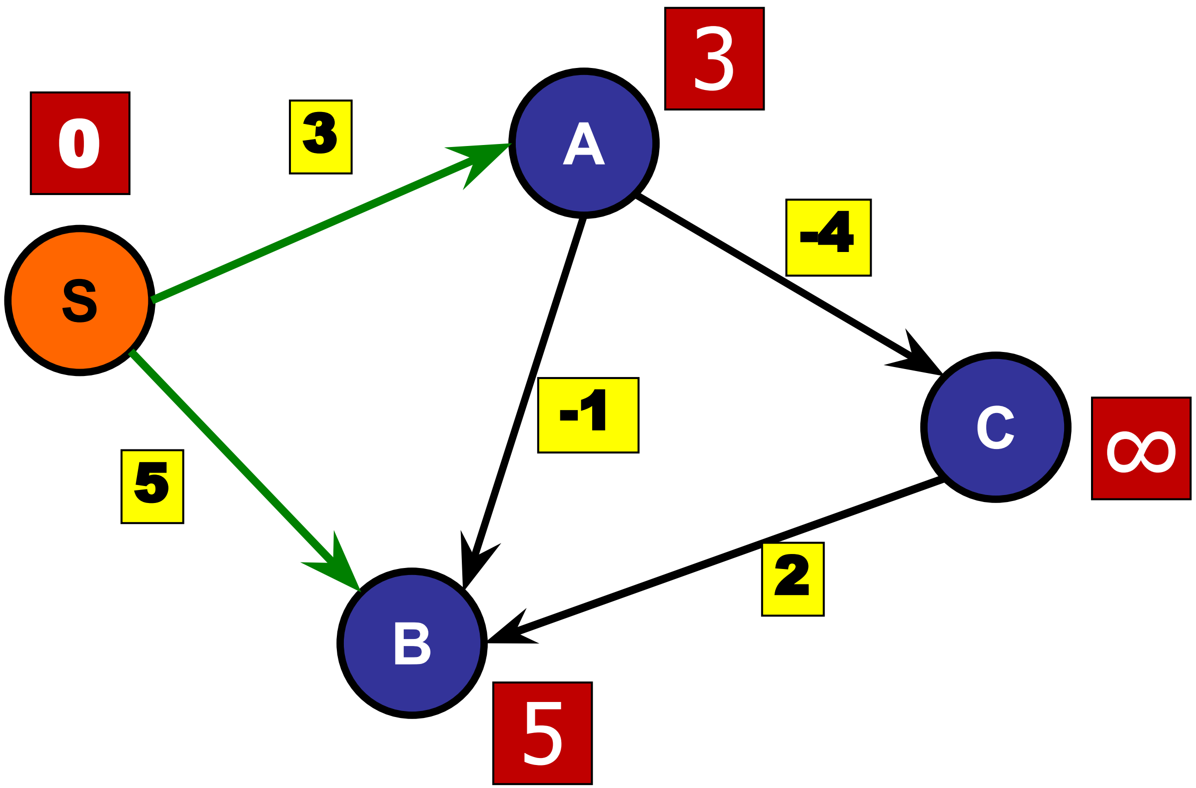
Bellman-Ford on DAG

In topo-sort order: S, A, C, B



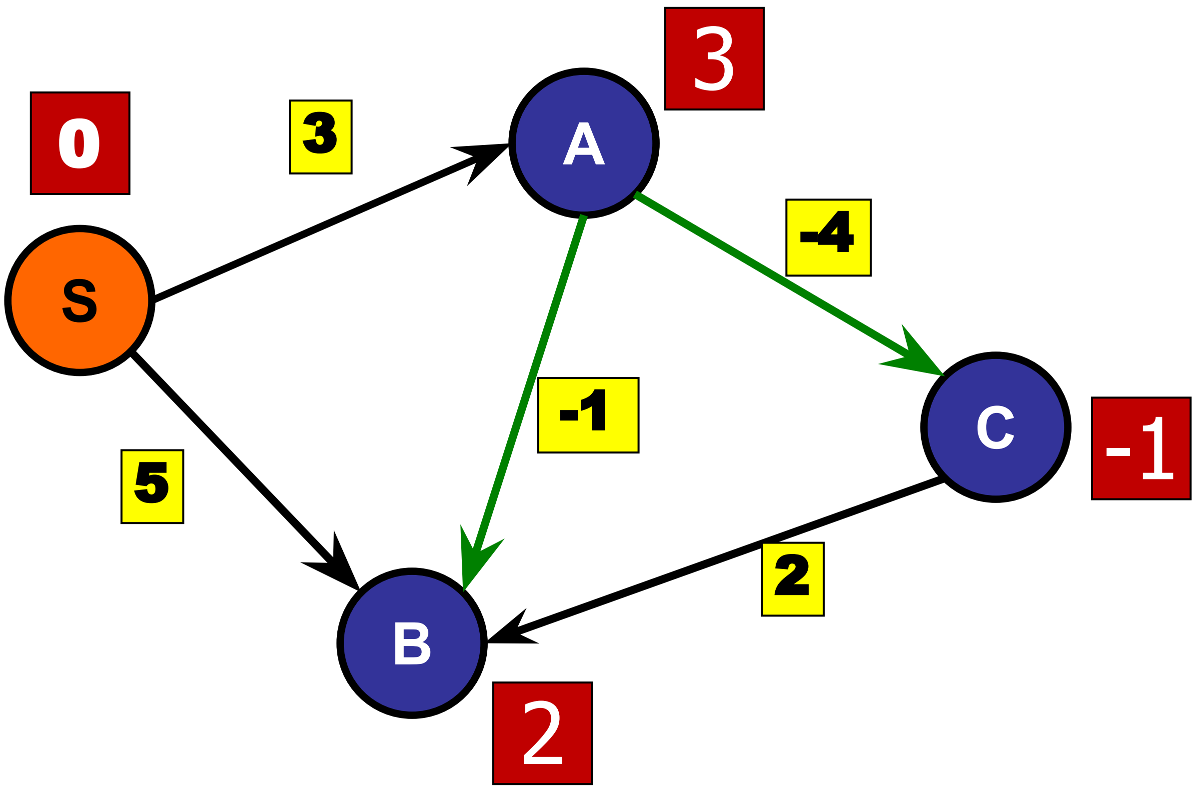
Bellman-Ford on DAG

In topo-sort order: S, A, C, B



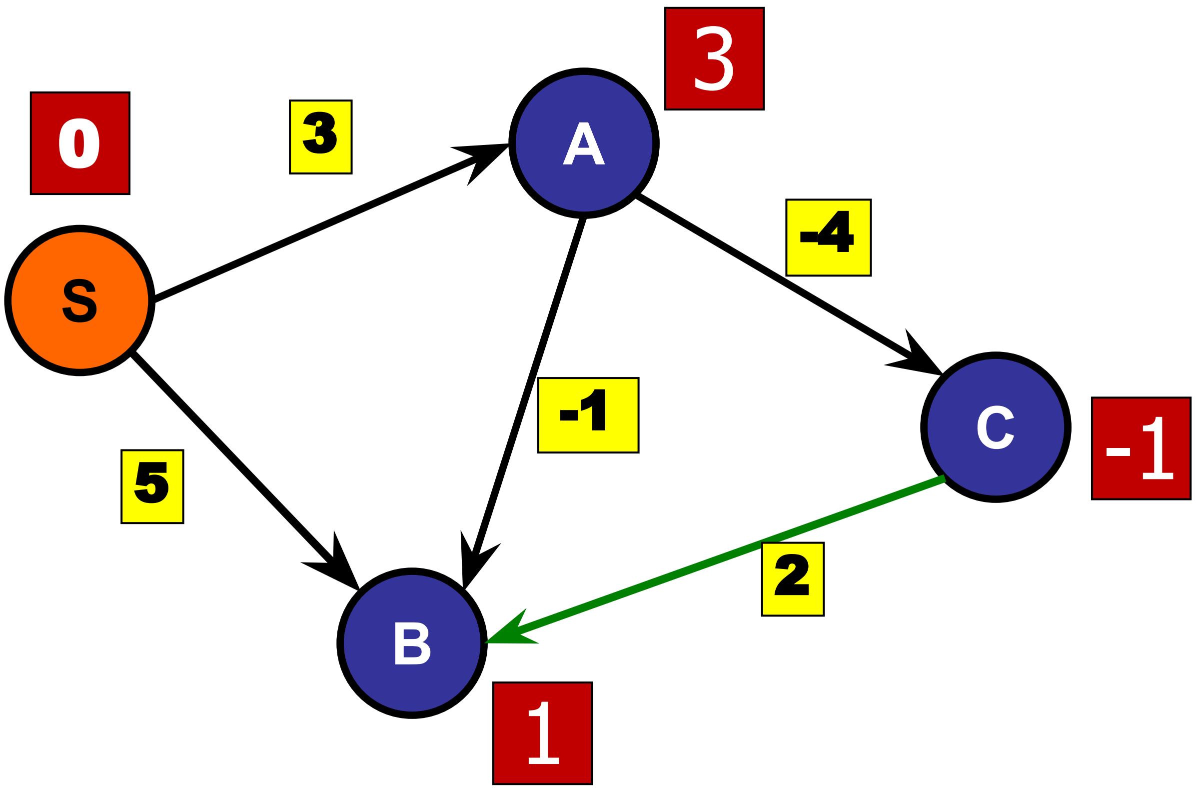
Bellman-Ford on DAG

In topo-sort order: S, A, C, B



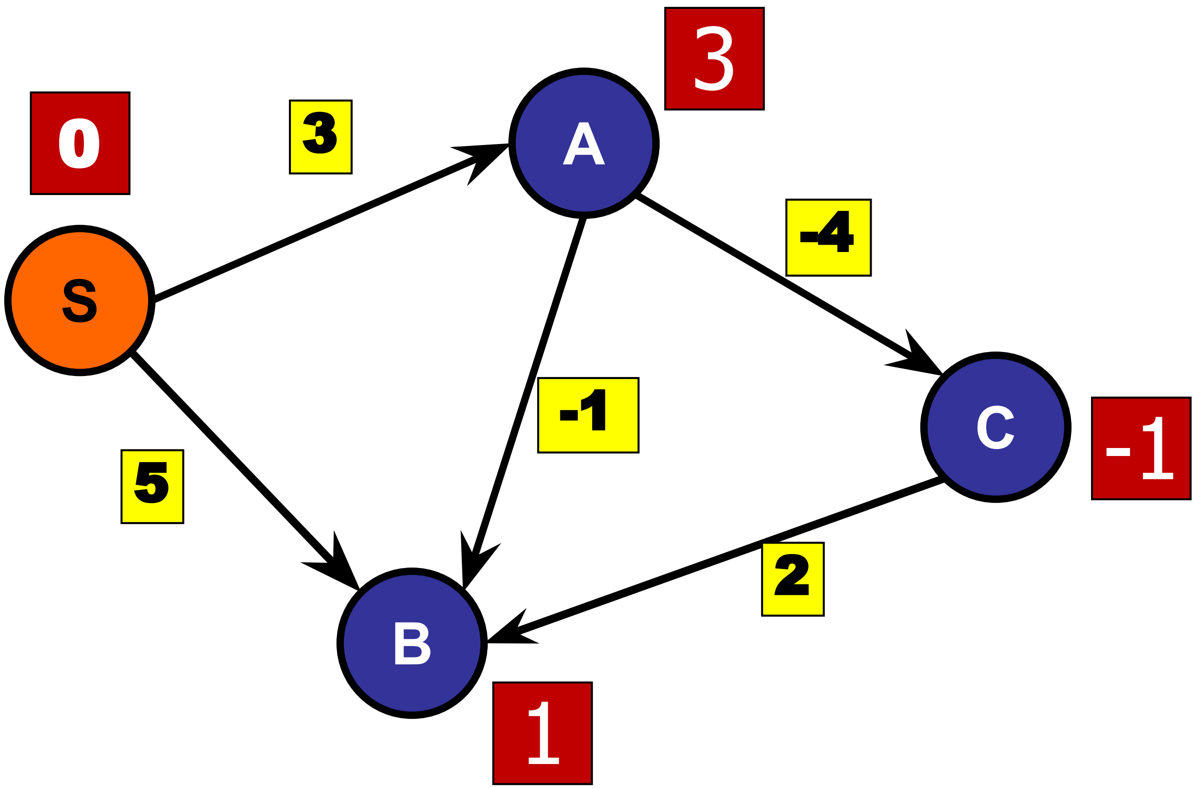
Bellman-Ford on DAG

In topo-sort order: S, A, C, B



Bellman-Ford on DAG

In topo-sort order: S, A, C, $\boxed{\text{B}}$



So Far:

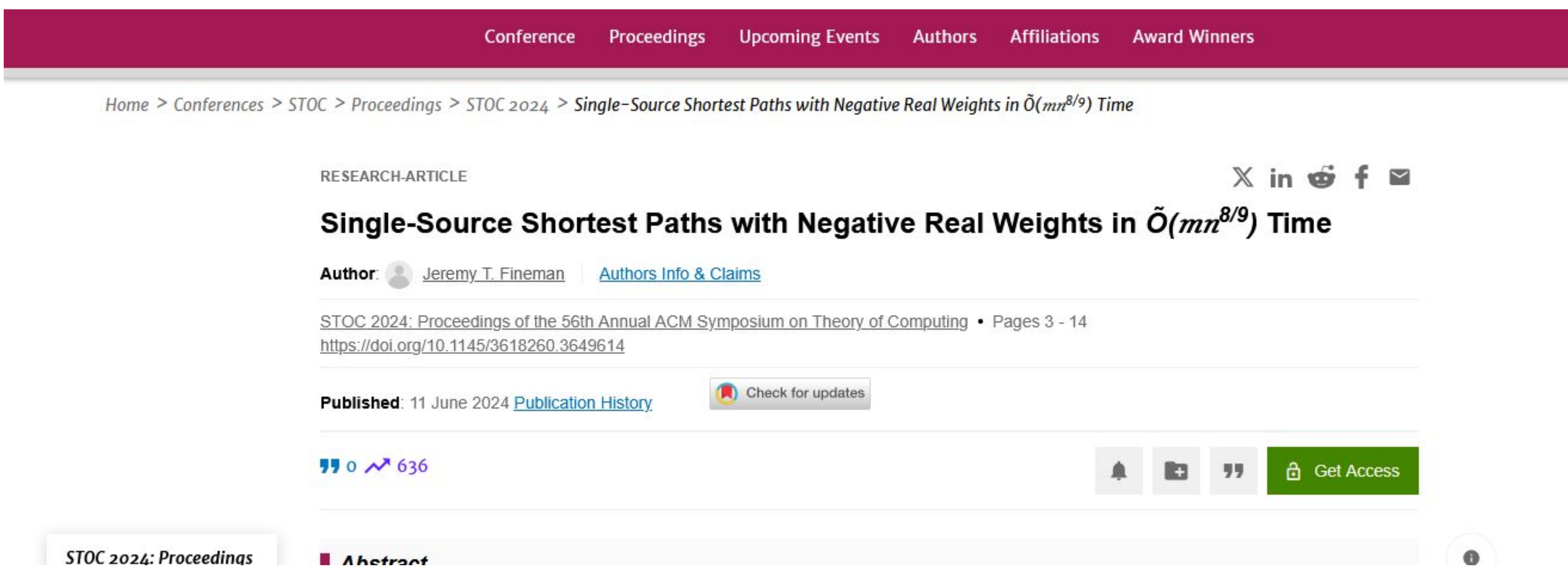
- **Unweighted** graph
  - BFS
  - O(V + E)
  -
- **Weighted** graphs with <u>non-negative edges</u>.
  - Dijkstra
  - O(E log V)

- **Weighted** graphs with <u>no negative cycles</u>.
  - Bellman-Ford
  - O(VE) on general
  - O(V + E) with toposort on DAG

Bonus: Active Research

What about if we could randomise?

Jeremy Fineman in STOC' 2024 showed an algorithm that runs in $O(EV^{8/9})$ time with high probability.


Conference Proceedings Upcoming Events Authors Affiliations Award Winners

Home > Conferences > STOC > Proceedings > STOC 2024 > Single-Source Shortest Paths with Negative Real Weights in $\tilde{O}(mn^{8/9})$ Time

RESEARCH-ARTICLE

Single-Source Shortest Paths with Negative Real Weights in $\tilde{O}(mn^{8/9})$ Time

Author: Jeremy T. Fineman Authors Info & Claims

STOC 2024: Proceedings of the 56th Annual ACM Symposium on Theory of Computing • Pages 3 - 14
https://doi.org/10.1145/3618260.3649614

Published: 11 June 2024 Publication History

Check for updates

0 636

Get Access

STOC 2024: Proceedings

Abstract

Today

Single Source Shortest Paths (SSSP):

- Bellman Ford
  - SSSP on negative edge graphs
  - Negative Cycle Detection

Some graph techniques

Technique: Graph Modifications

Now that we've talked about single-source shortest paths. Let's think about:

Technique: Graph Modifications

Now that we've talked about single-source shortest paths. Let's think about:

What if we had multiple sources, and we just want the shortest path to/from one of these sources?

Technique: Graph Modifications

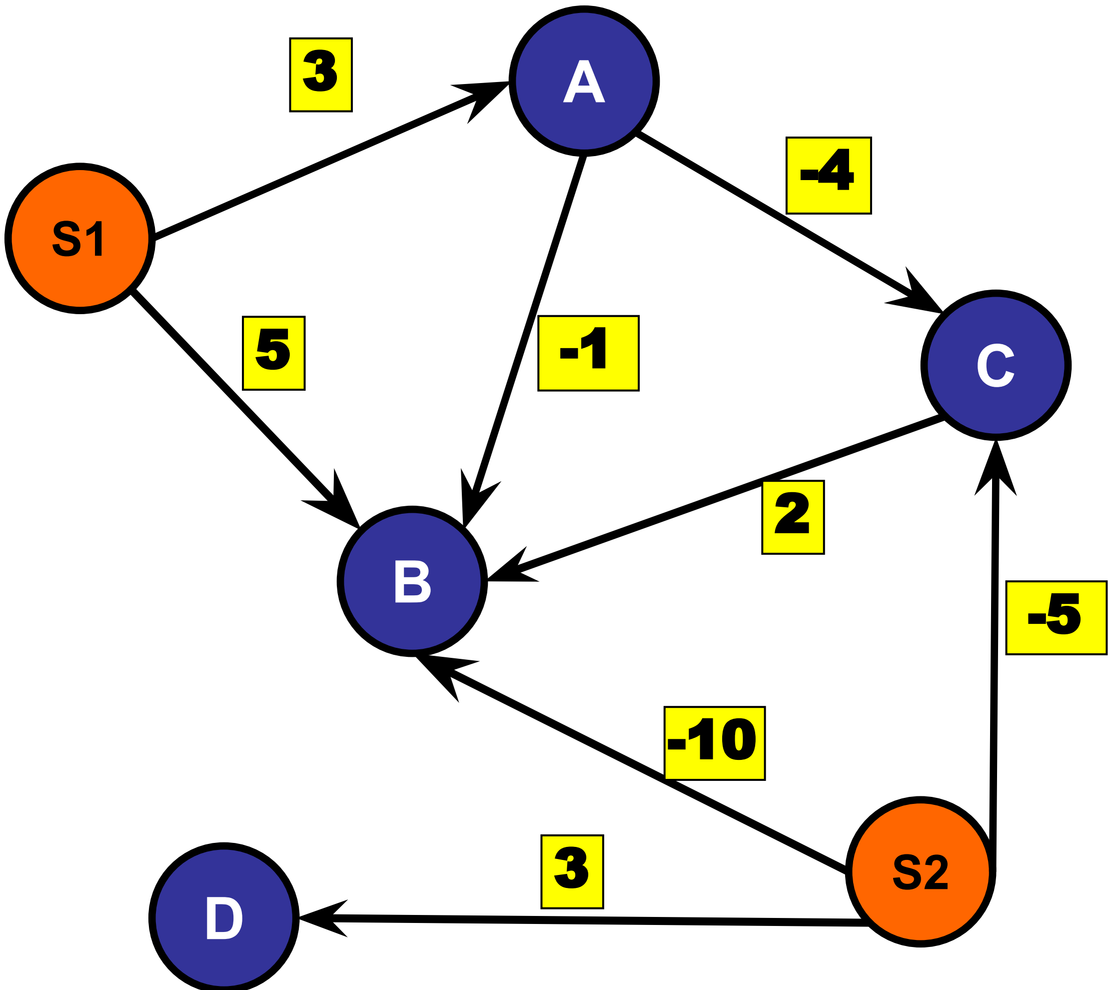
Now that we've talked about single-source shortest paths. Let's think about:

What if we had multiple sources, and we just want the shortest path to/from **any** of these sources?

E.g. we have many fire stations. We just want the closest one to reach the target as soon as possible.

Technique: Graph Modifications

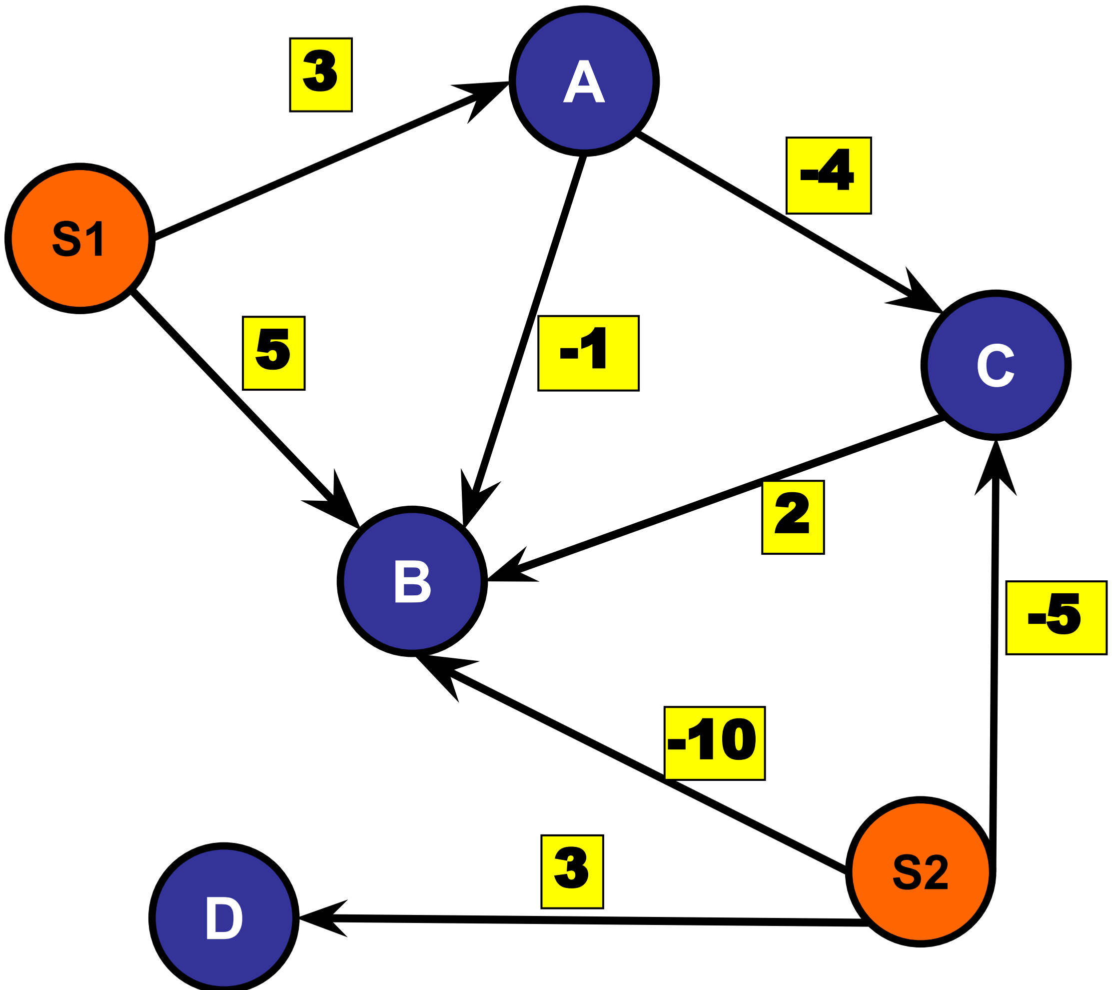
E.g. we have two sources s1 and s2.



Technique: Graph Modifications

Obvious solution: Run SSSP once from S1.

A	B	C	D
3	1	-1	∞

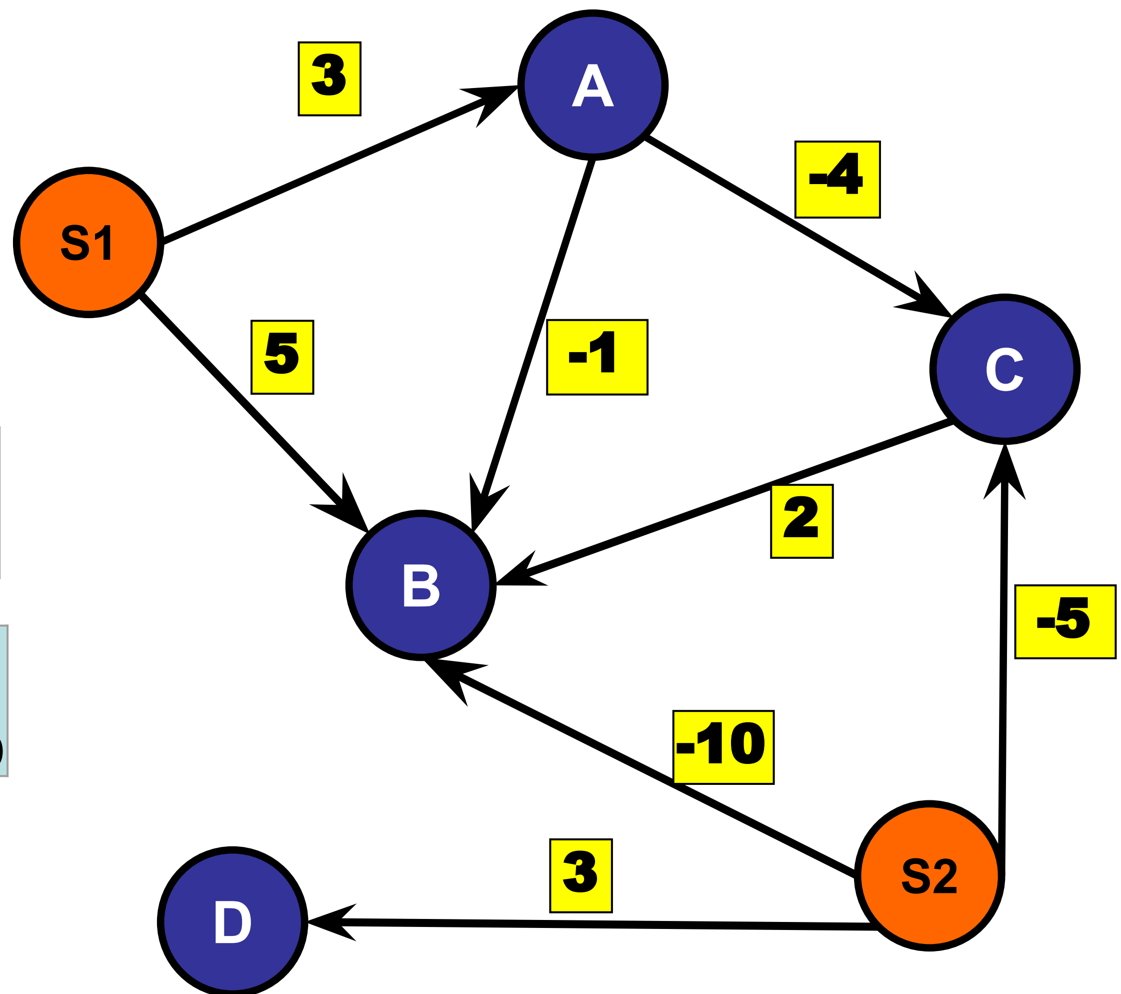


Technique: Graph Modifications

Obvious solution: Run SSSP once from S1.
Run SSSP again from S2.

3 A	1 B	-1 C	∞ D

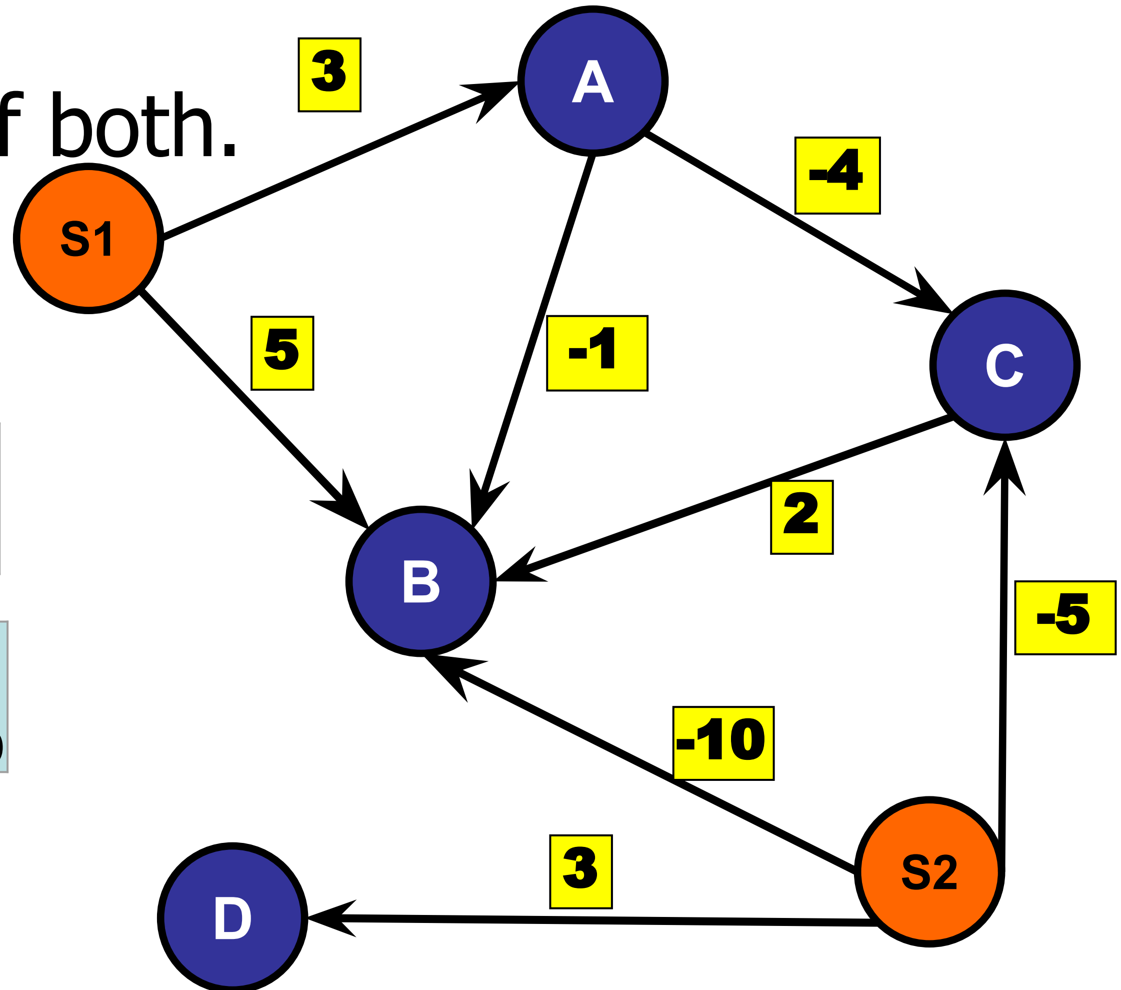
∞ A	-10 B	-5 C	3 D



Technique: Graph Modifications

Obvious solution: Run SSSP once from S1.
Run SSSP again from S2.

Output minimum of both.



3_A	1_B	-1_C	∞_D

∞_A	-10_B	-5_C	3_D

Technique: Graph Modifications

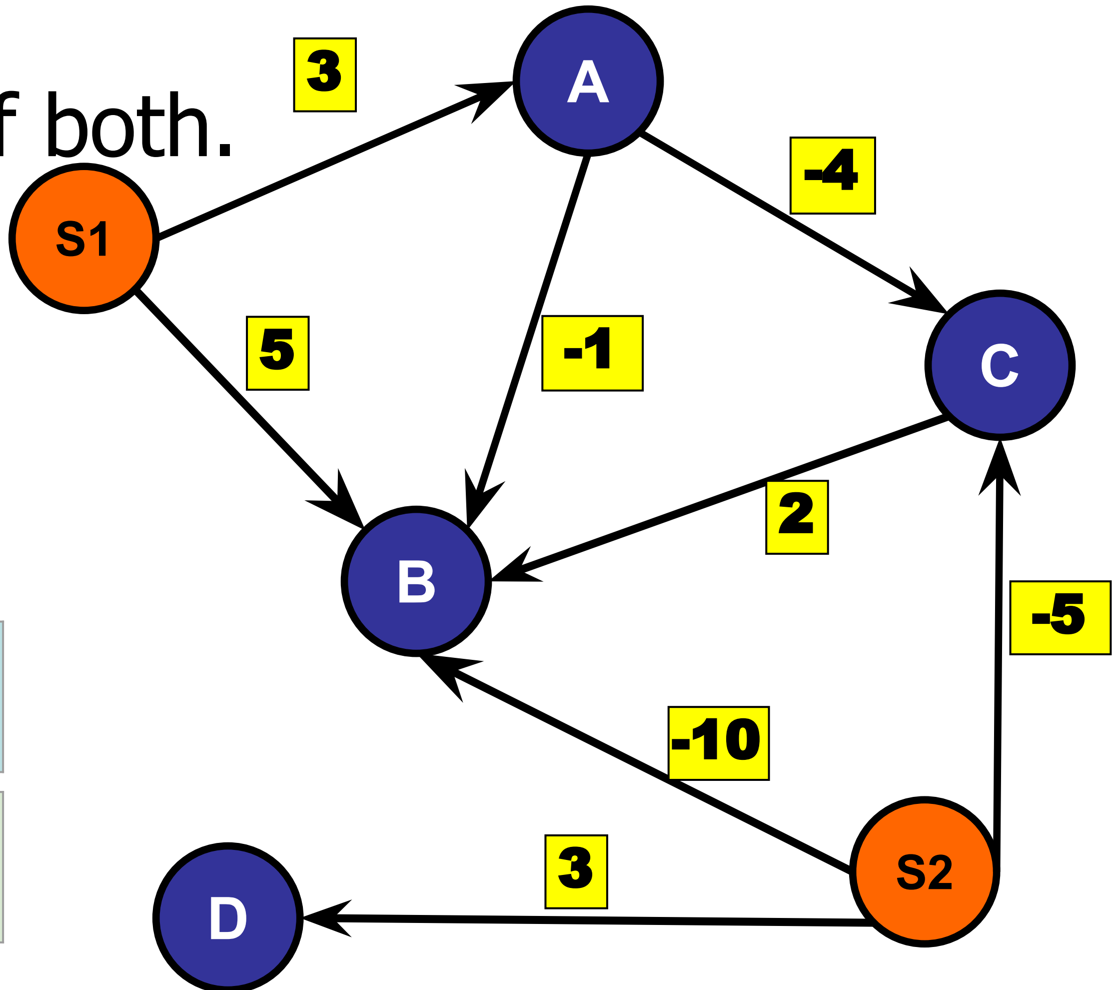
Obvious solution: Run SSSP once from S1.
Run SSSP again from S2.

Output minimum of both.

3 A	1 B	-1 C	∞ D

∞ A	-10 B	-5 C	3 D

3 A	-10 B	-5 C	3 D



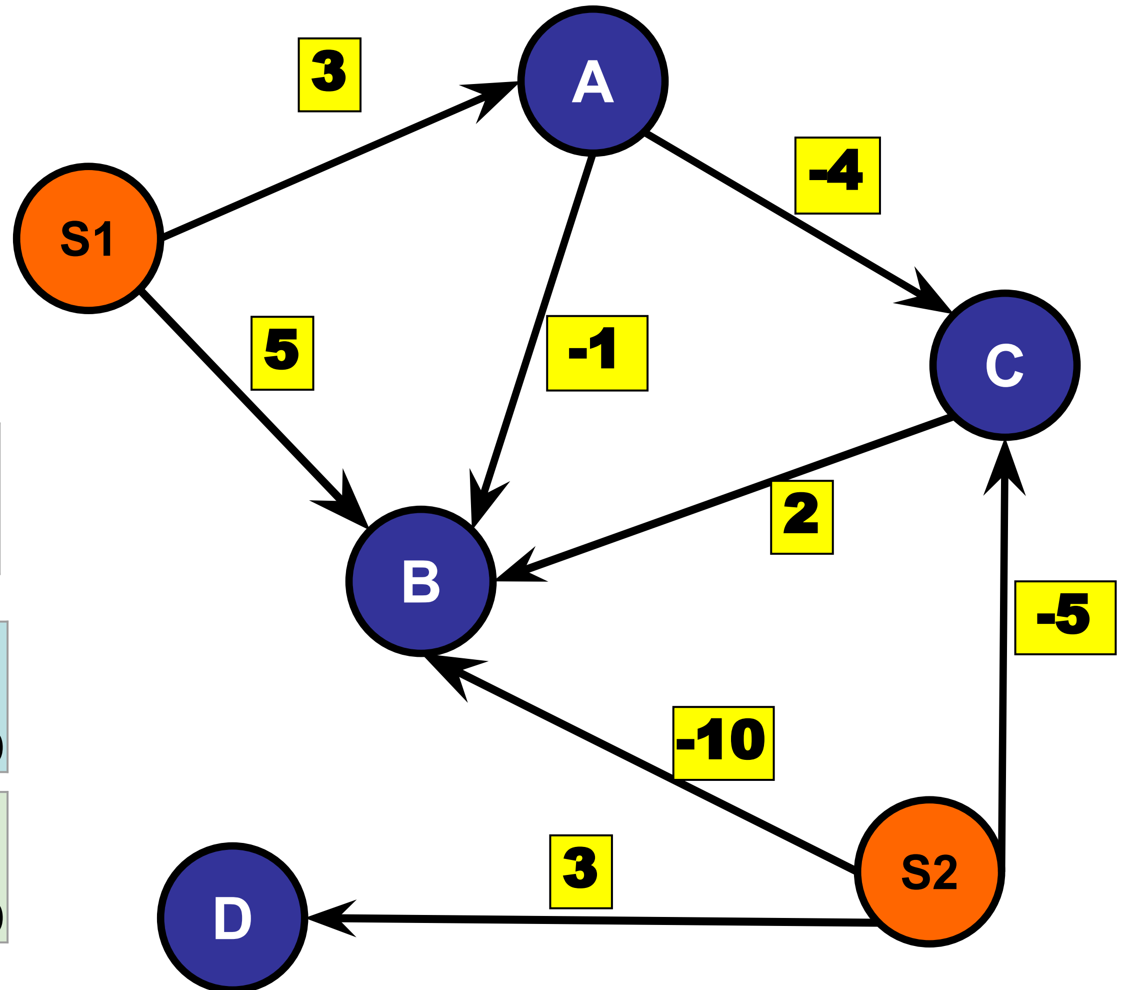
Technique: Graph Modifications

But if we had t sources, this means running t copies of SSSP.

3_A	1_B	-1_C	∞_D

∞_A	-10_B	-5_C	3_D

3_A	-10_B	-5_C	3_D



Technique: Graph Modifications

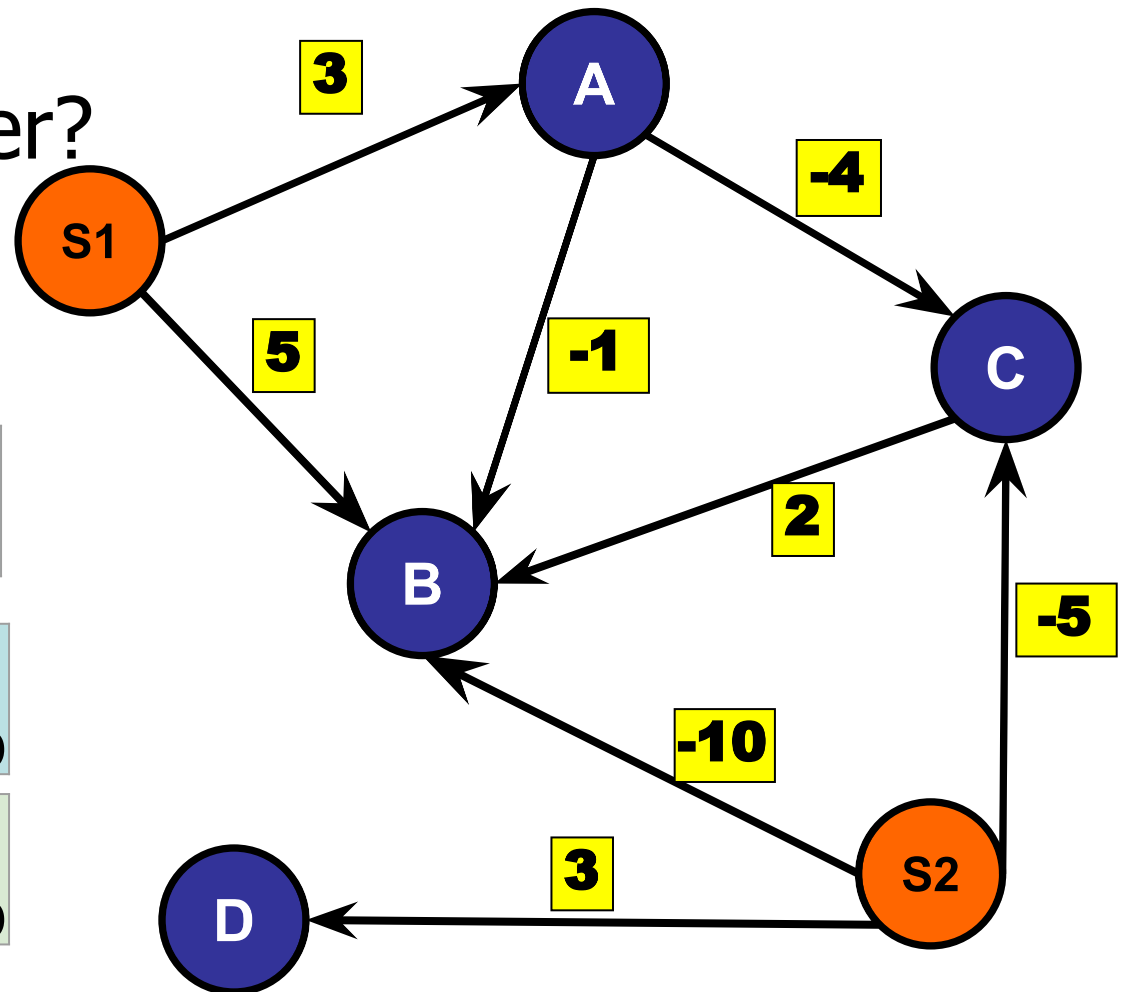
But if we had t sources, this means running t copies of SSSP.

How do we do better?

3 A	1 B	-1 C	∞ D

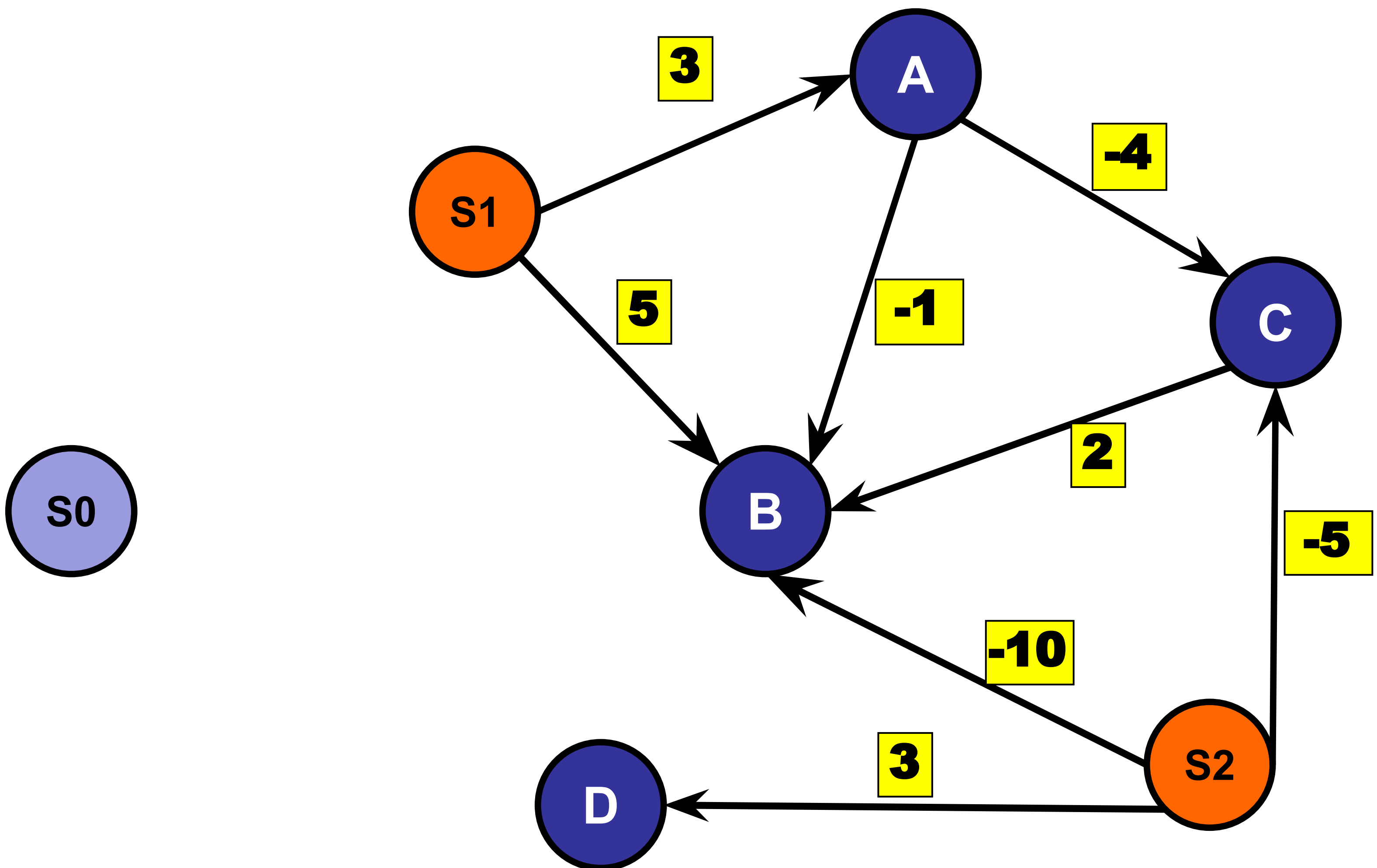
∞ A	-10 B	-5 C	3 D

3 A	-10 B	-5 C	3 D



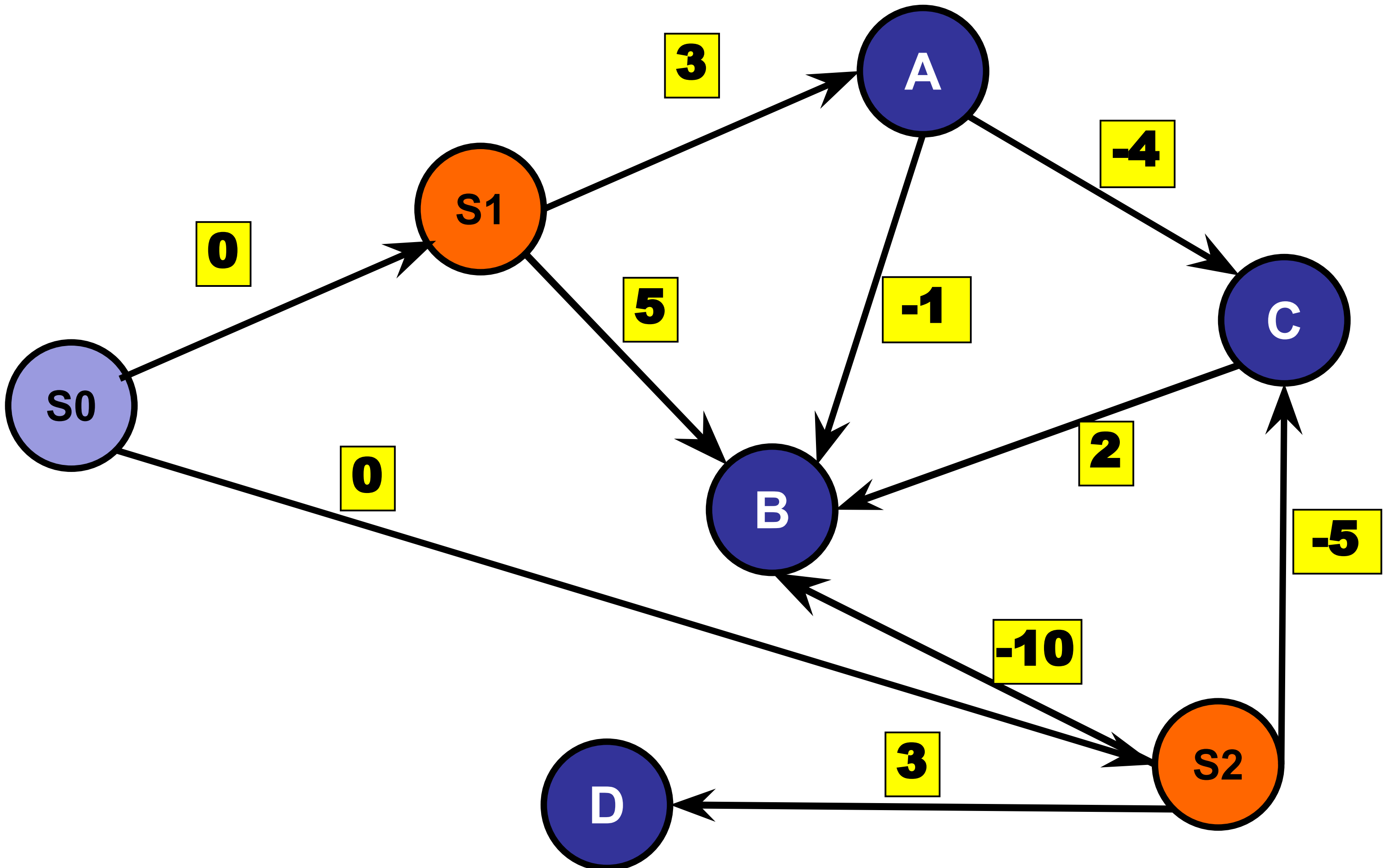
Technique: Graph Modifications

Idea: Make a false SINGLE source, s0.



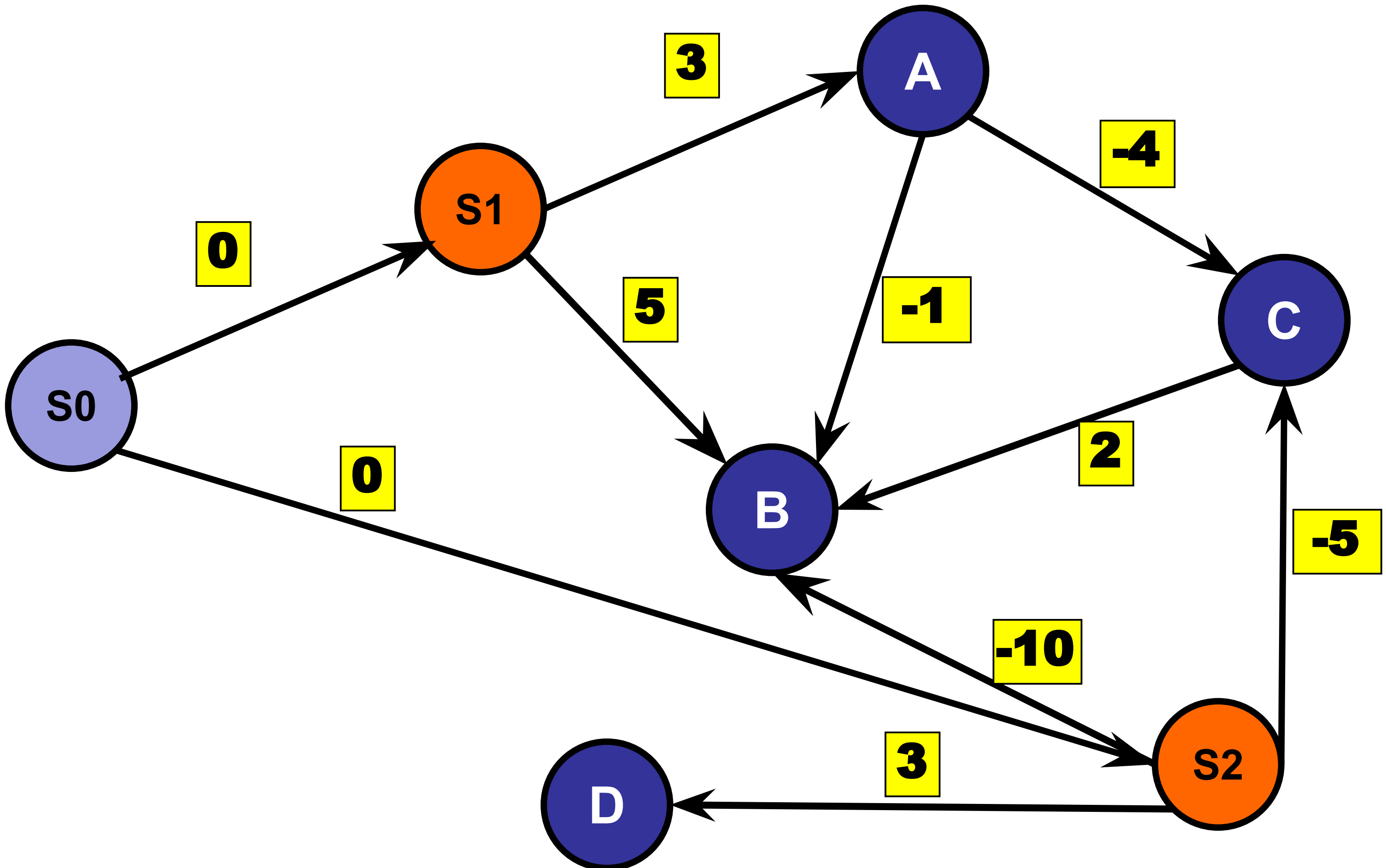
Technique: Graph Modifications

Idea: Point s0 to all sources, with edge costing 0



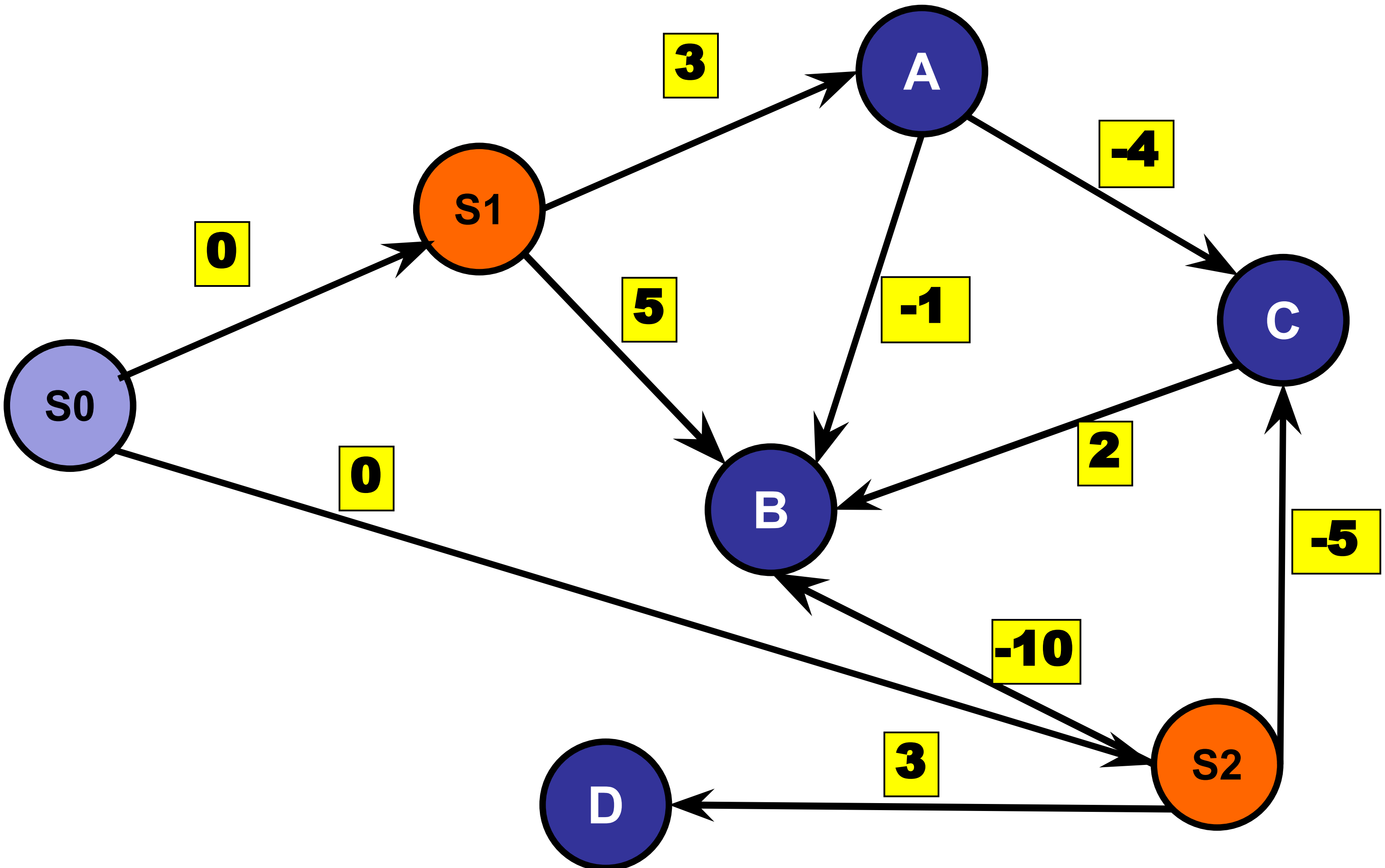
Technique: Graph Modifications

Idea: Run a single copy of SSSP from s0.



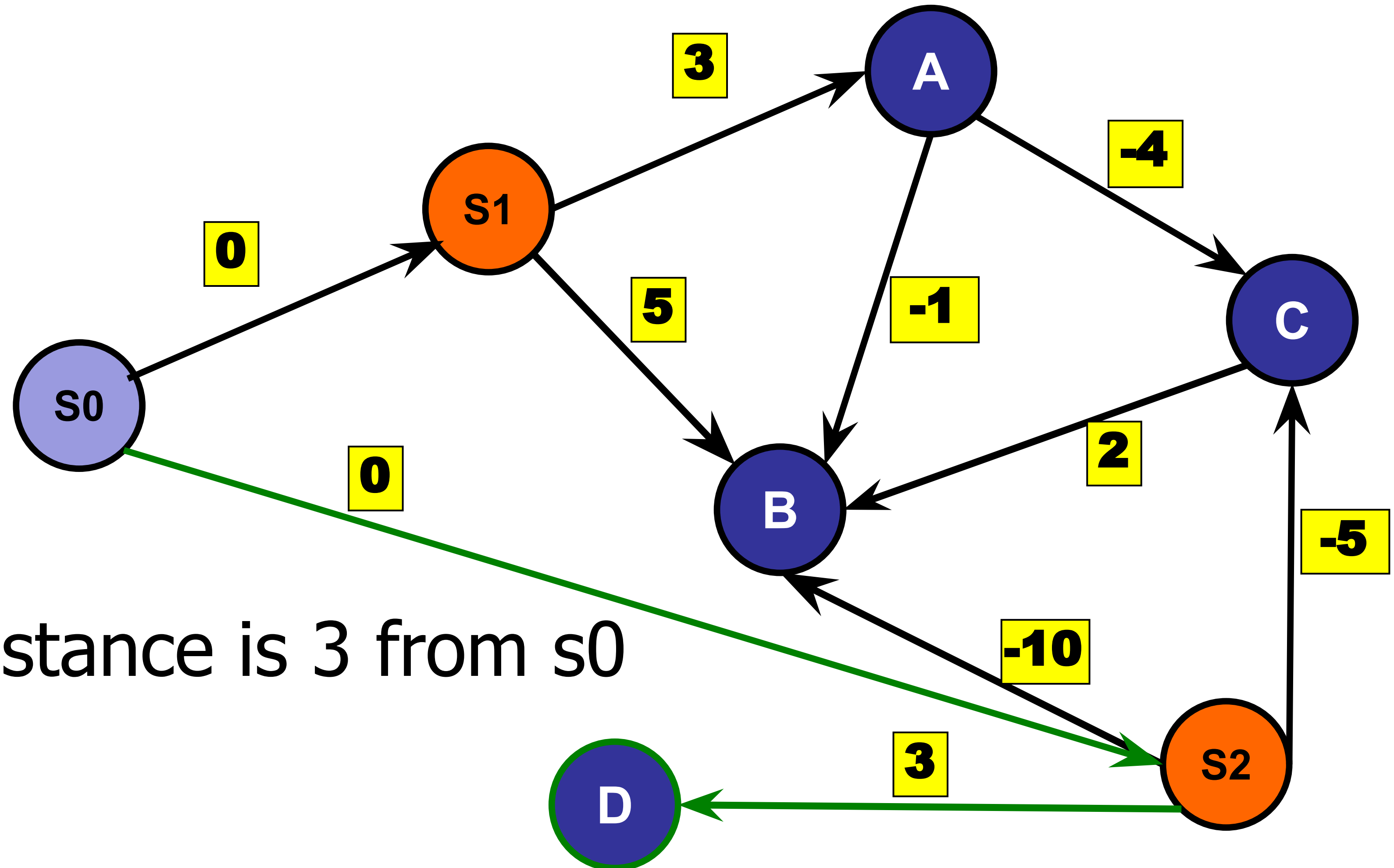
Technique: Graph Modifications

The shortest path from s0 must go through one of the original sources.



Technique: Graph Modifications

The shortest path from s0 must go through one of the original sources.

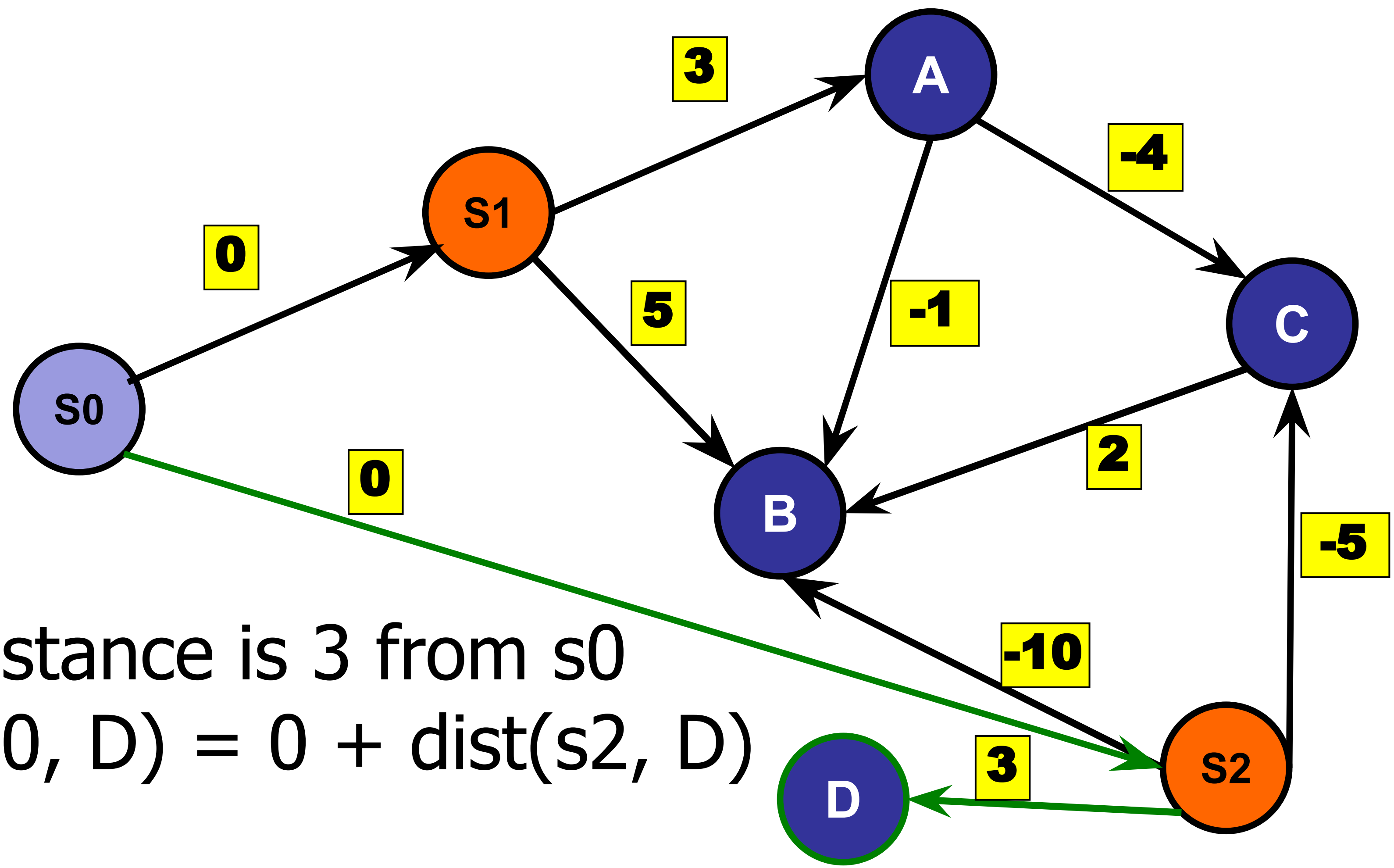


E.g.

D's distance is 3 from s0

Technique: Graph Modifications

The shortest path from s0 must go through one of the original sources.



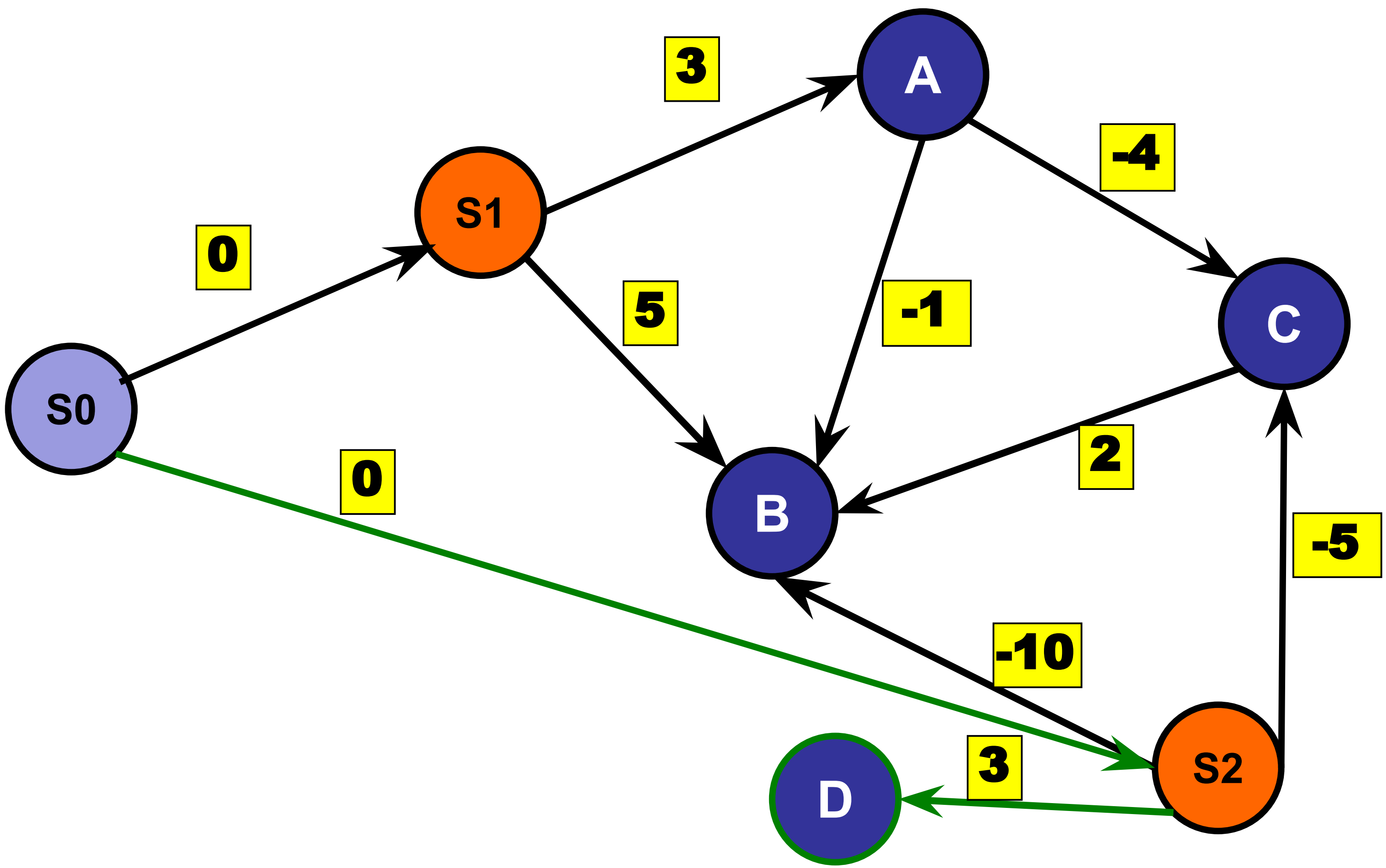
E.g.

D's distance is 3 from s0

dist(s0, D) = 0 + dist(s2, D)

Technique: Graph Modifications

This solves the problem!



Technique: Graph Modifications

What about if we want shortest path that takes at **exactly** k edges?

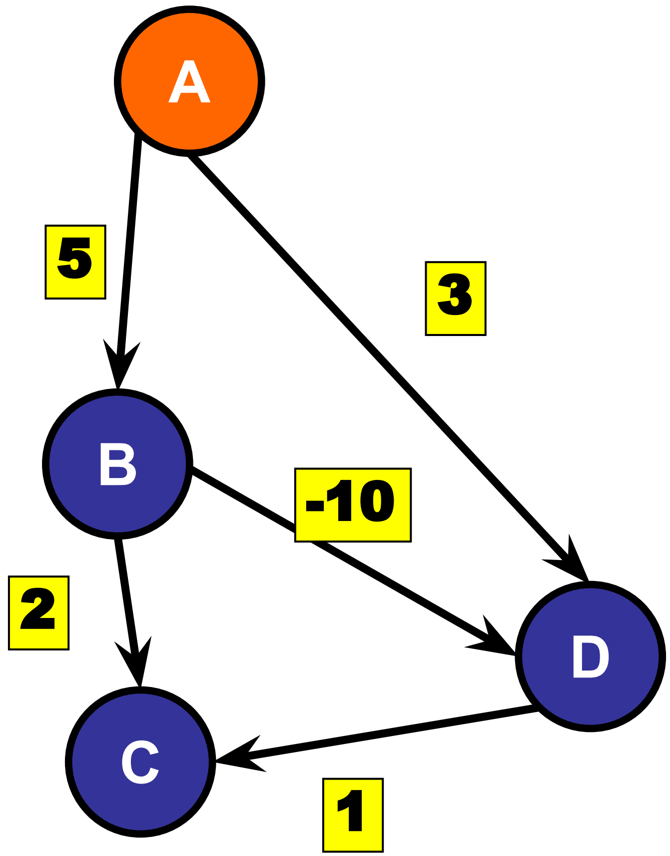
Technique: Graph Modifications

What about if we want shortest path that takes at **exactly** k edges?

Now we can't just run SSSP because we don't know how many edges the shortest path takes.

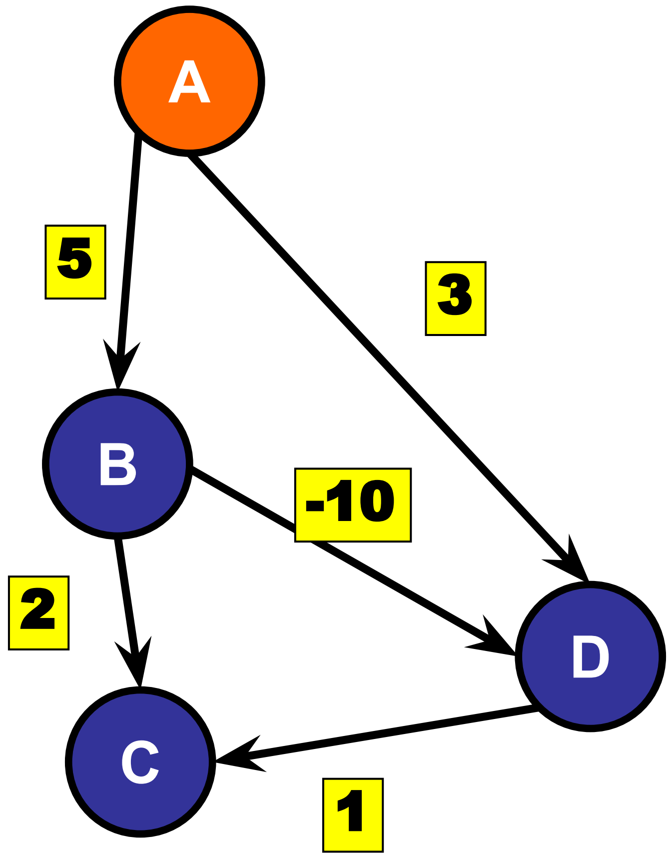
Technique: Graph Modifications

E.g. Shortest from A to C using exactly ? edge.



Technique: Graph Modifications

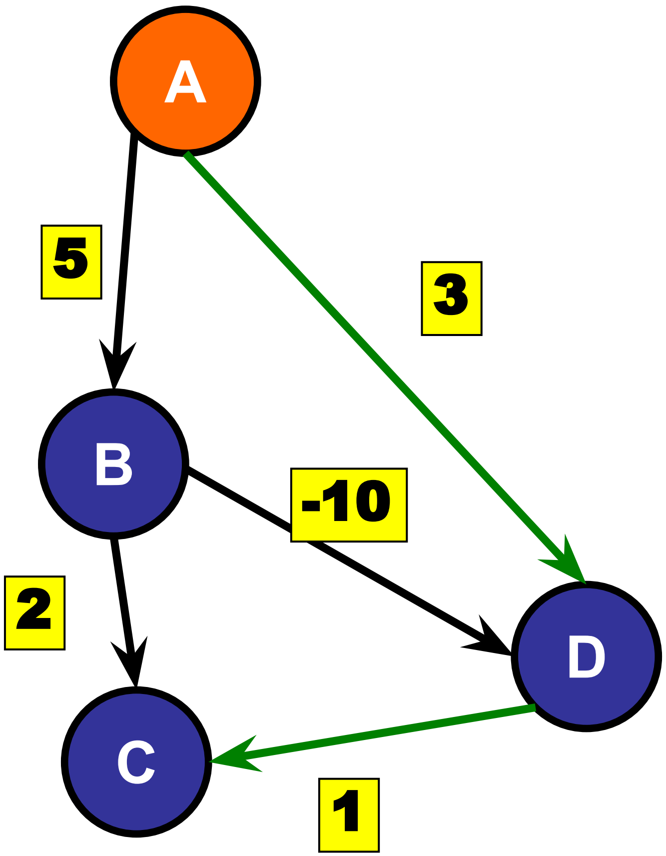
E.g. Shortest from A to C using exactly 1 edge.
= impossible



Technique: Graph Modifications

E.g. Shortest from A to C using exactly 2 edges.

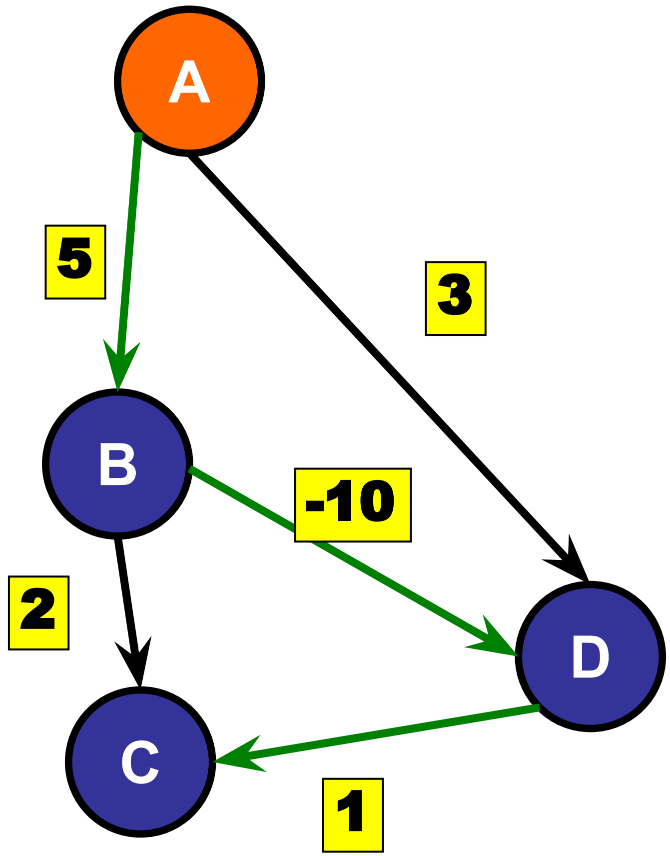
= 4



Technique: Graph Modifications

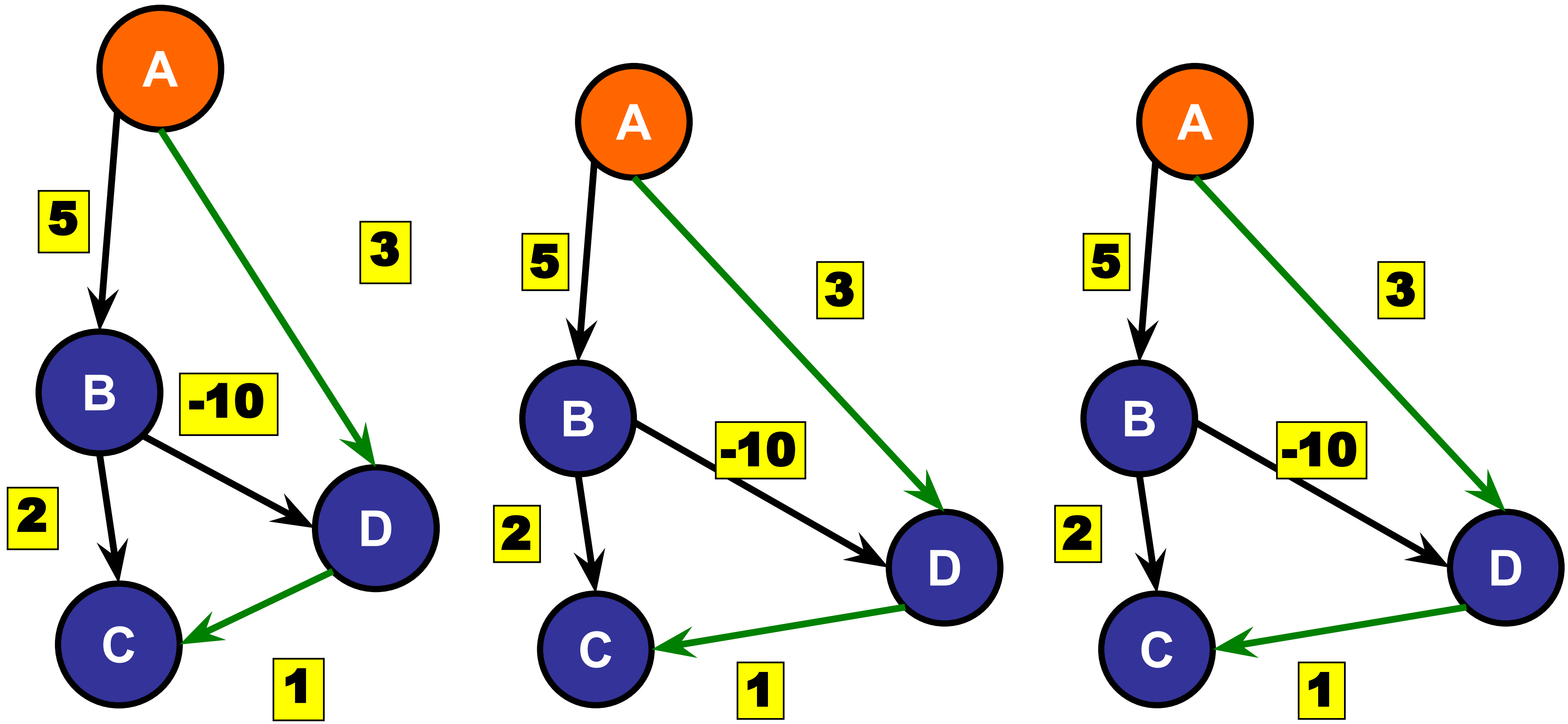
E.g. Shortest from A to C using exactly 3 edges.

= 5 -10 + 1 = -4



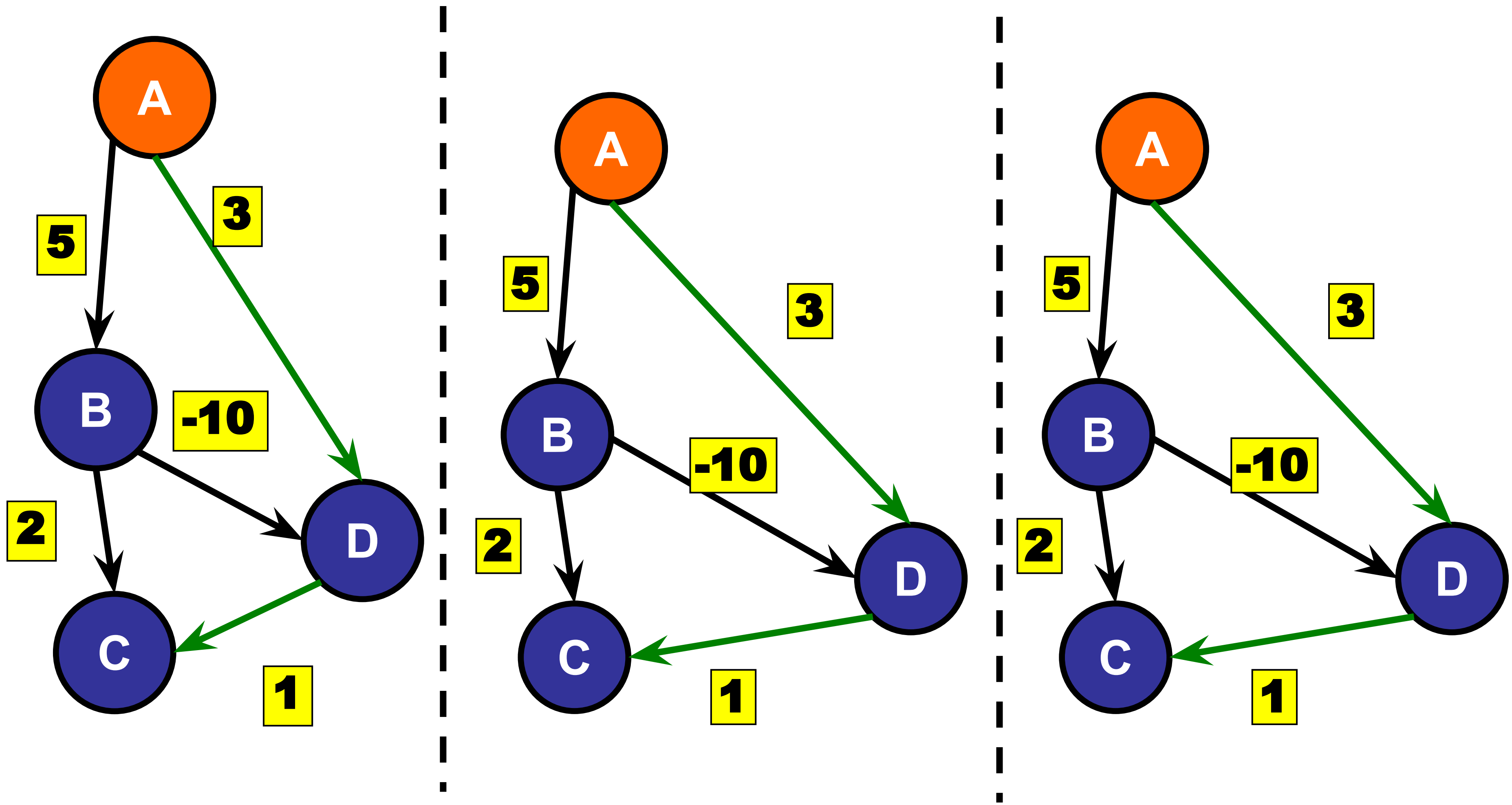
Technique: Graph Modifications

Idea, what happens if we copied the graph k + 1 times? E.g. k = 2



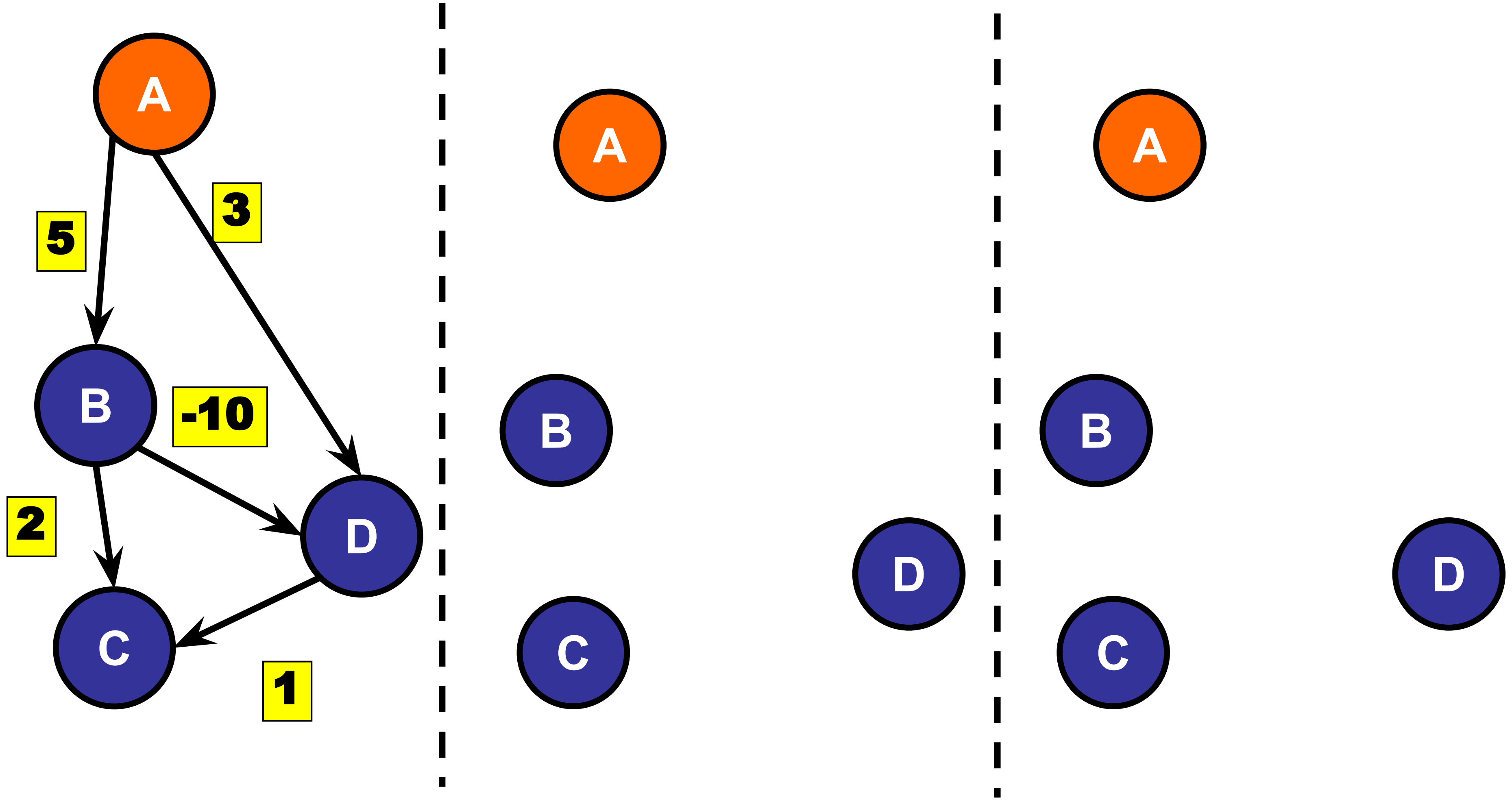
Technique: Graph Modifications

Call each copy a layer:



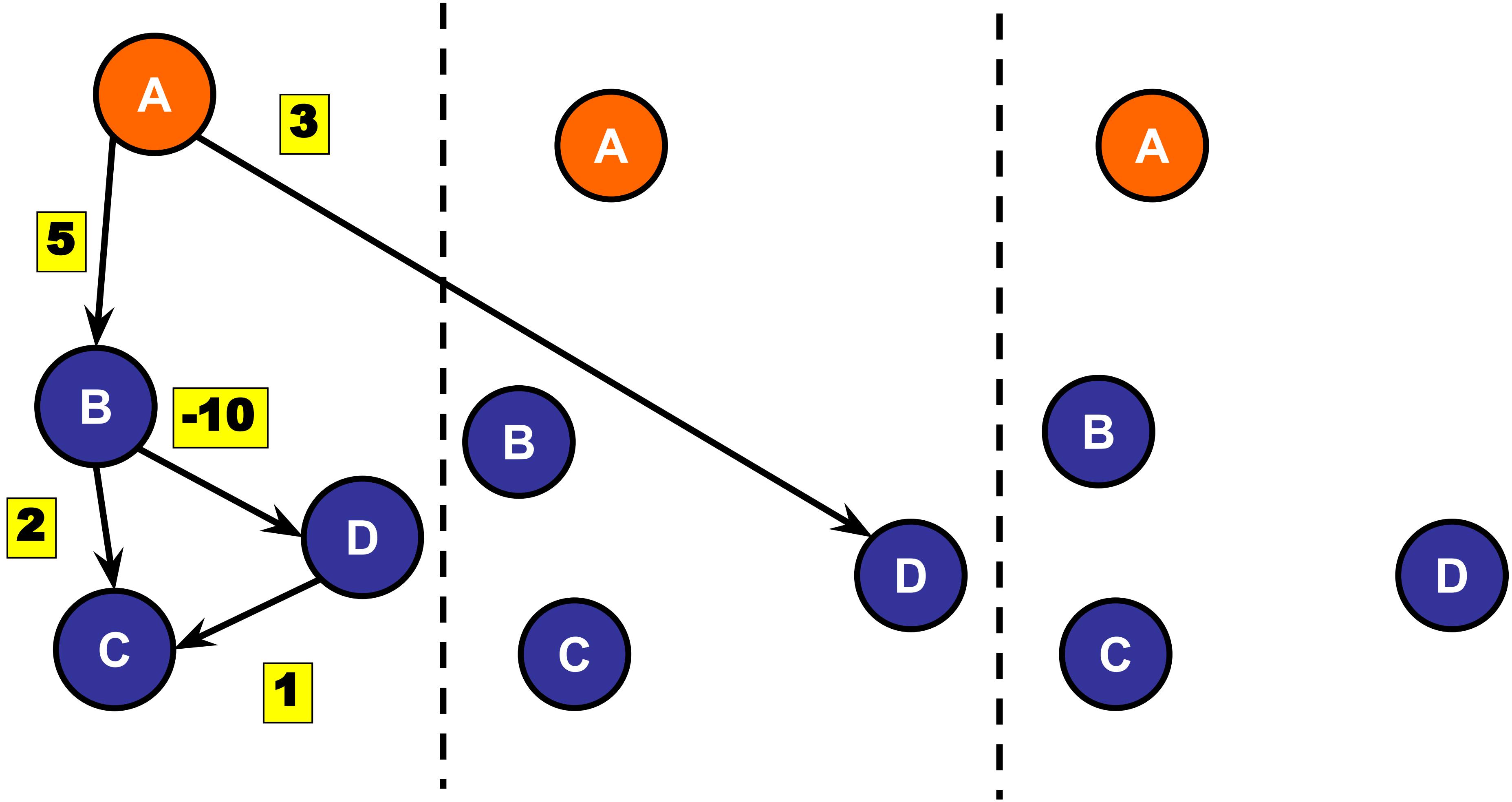
Technique: Graph Modifications

Now if originally, A goes to D,
then A in layer 0 goes to D in layer 1.



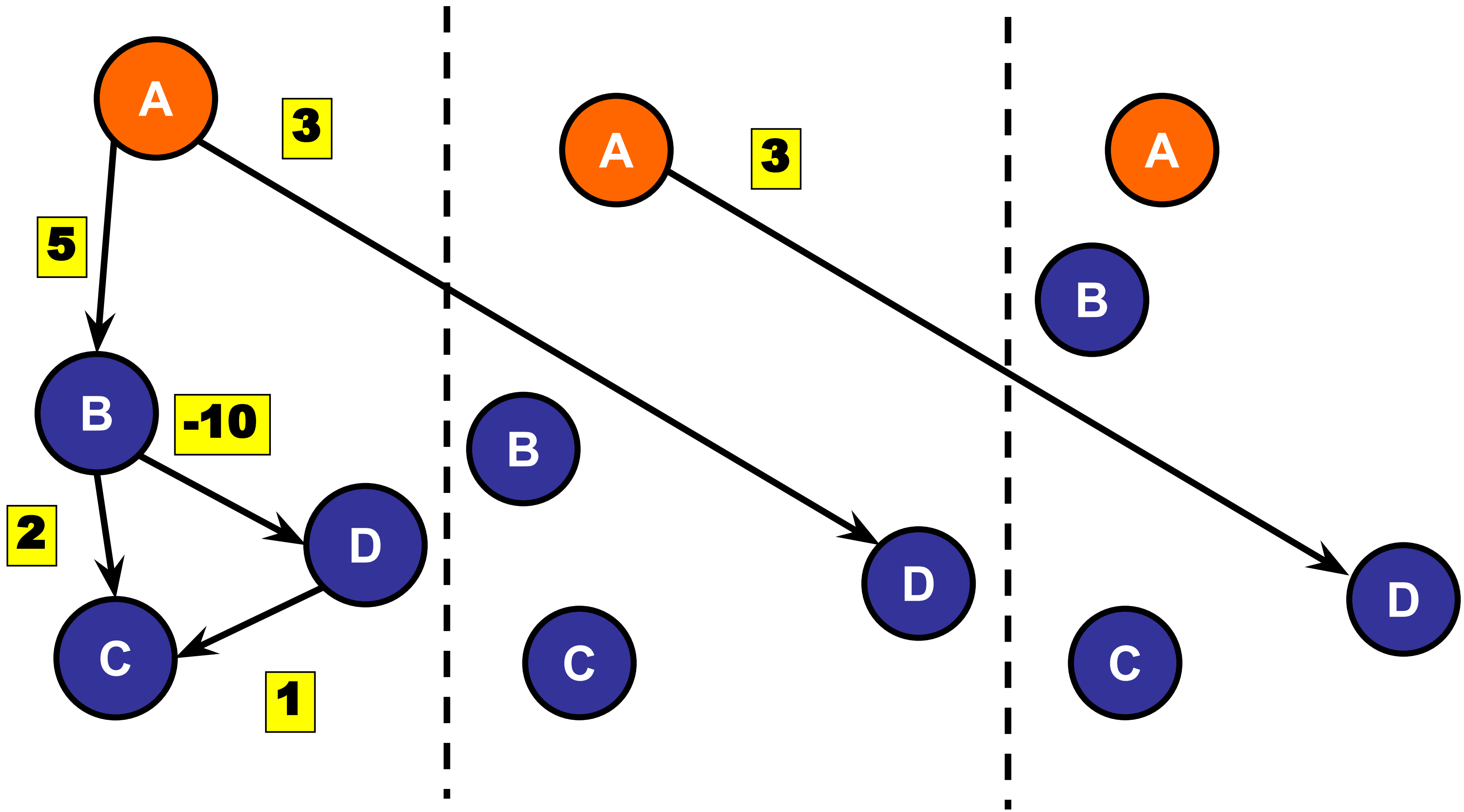
Technique: Graph Modifications

Now if originally, A goes to D,
then A in layer 0 goes to D in layer 1.



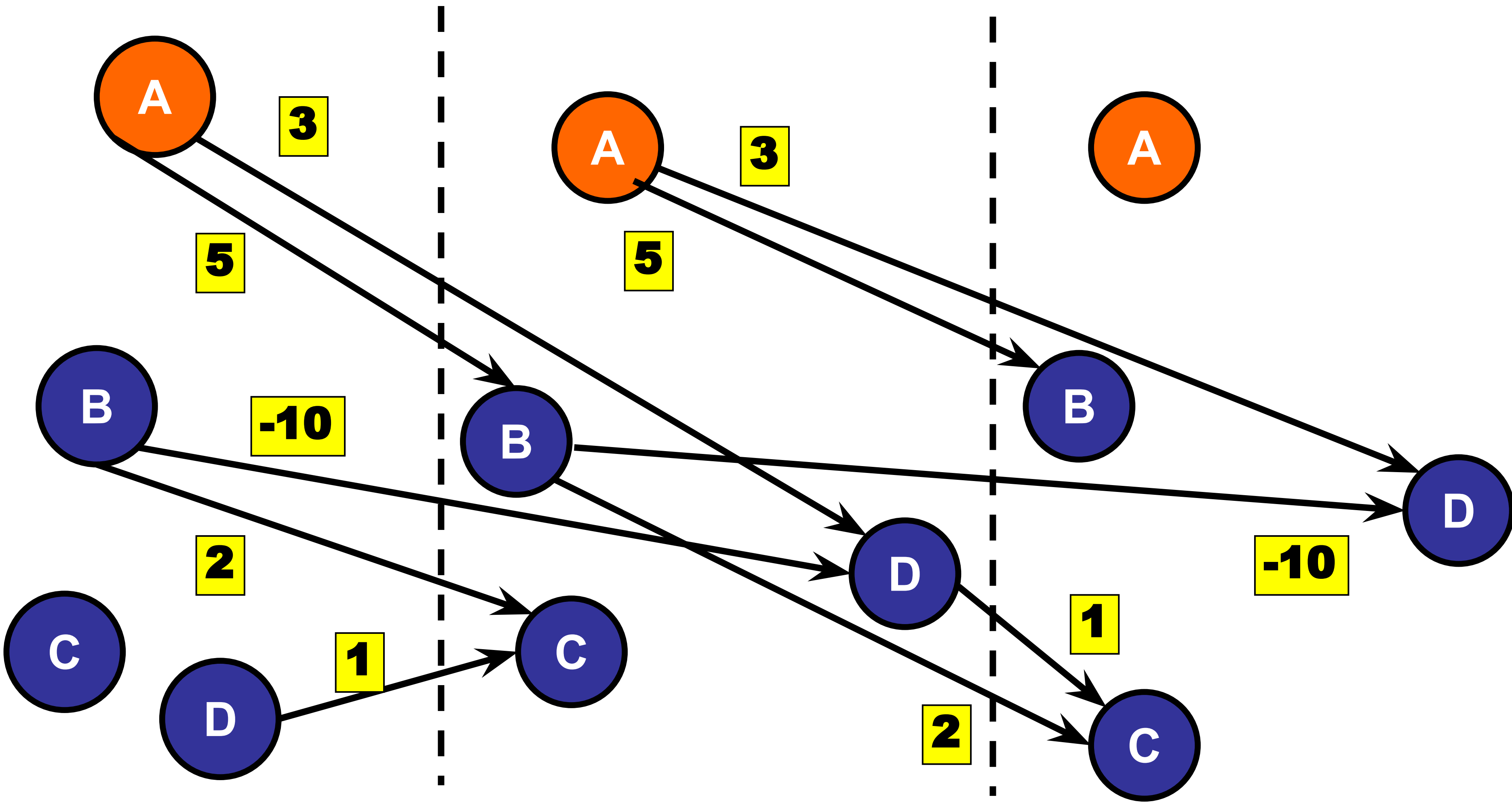
Technique: Graph Modifications

Now if originally, A goes to D, similarly then A in layer 1 goes to D in layer 2.



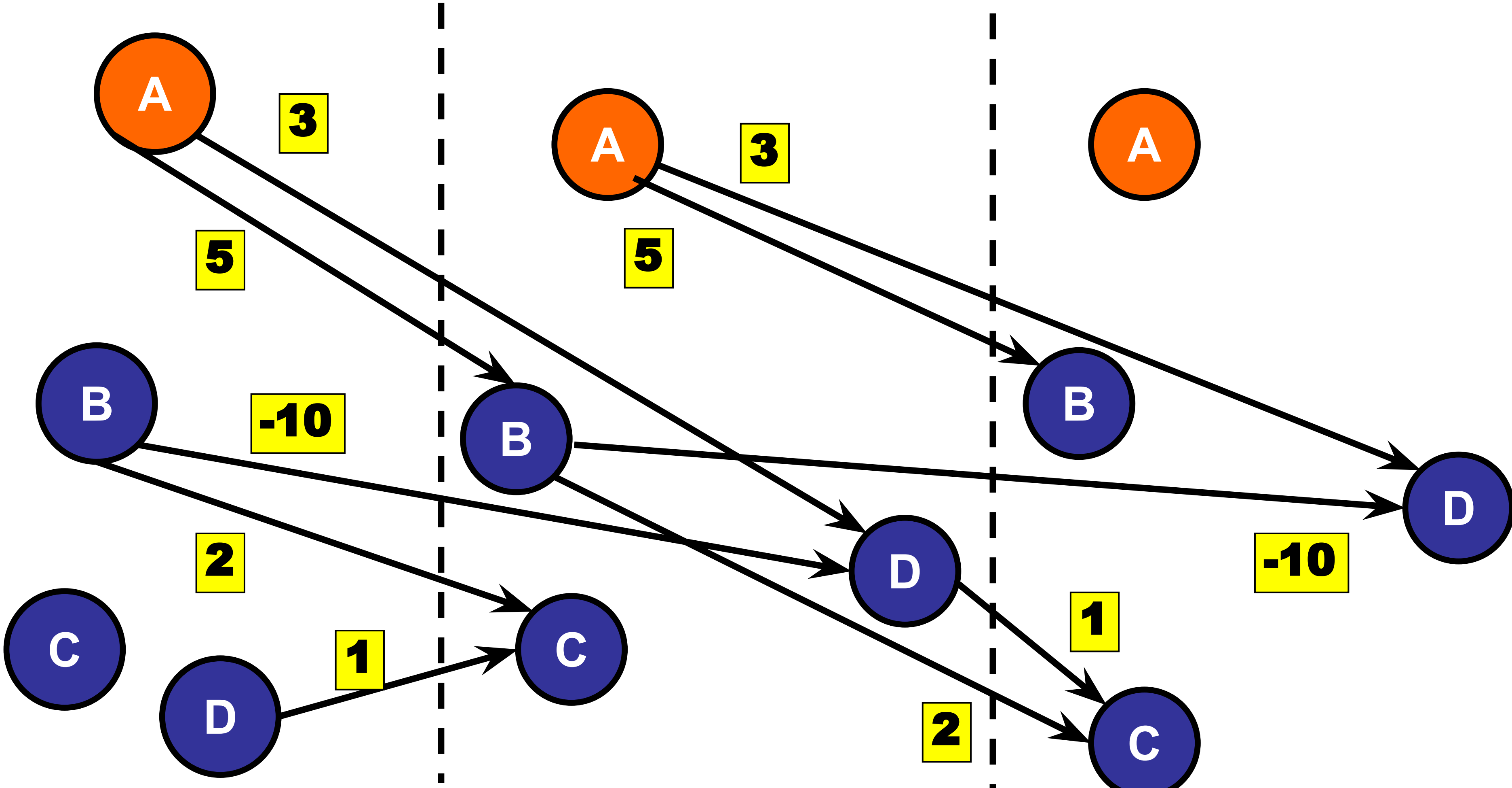
Technique: Graph Modifications

Do this for all edges: if (u, v) is an edge,
Then draw edge from u from layer i to v from layer i + 1.



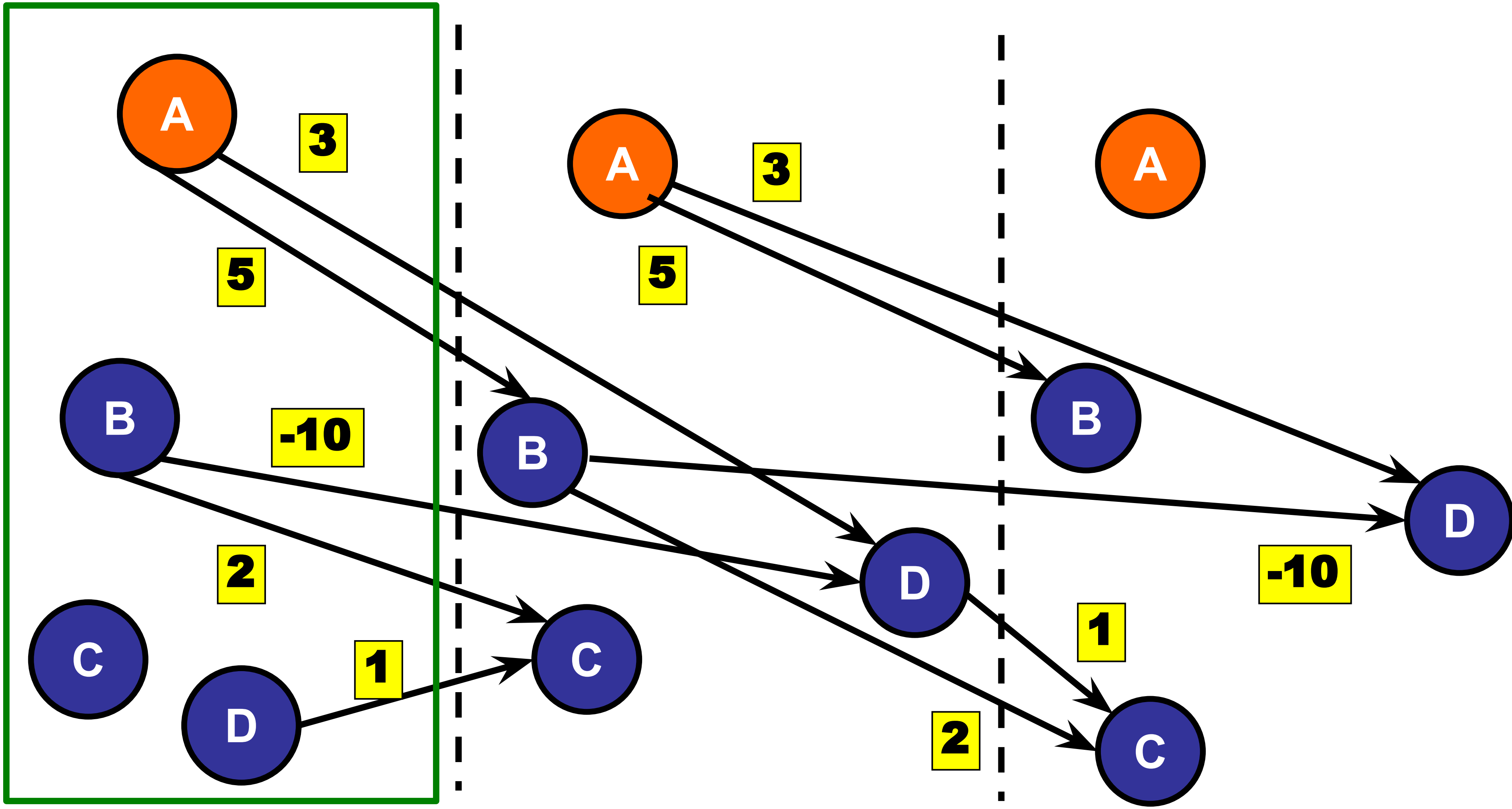
Technique: Graph Modifications

Intuition: If we are on the ith layer, we have taken exactly i steps in the original graph.



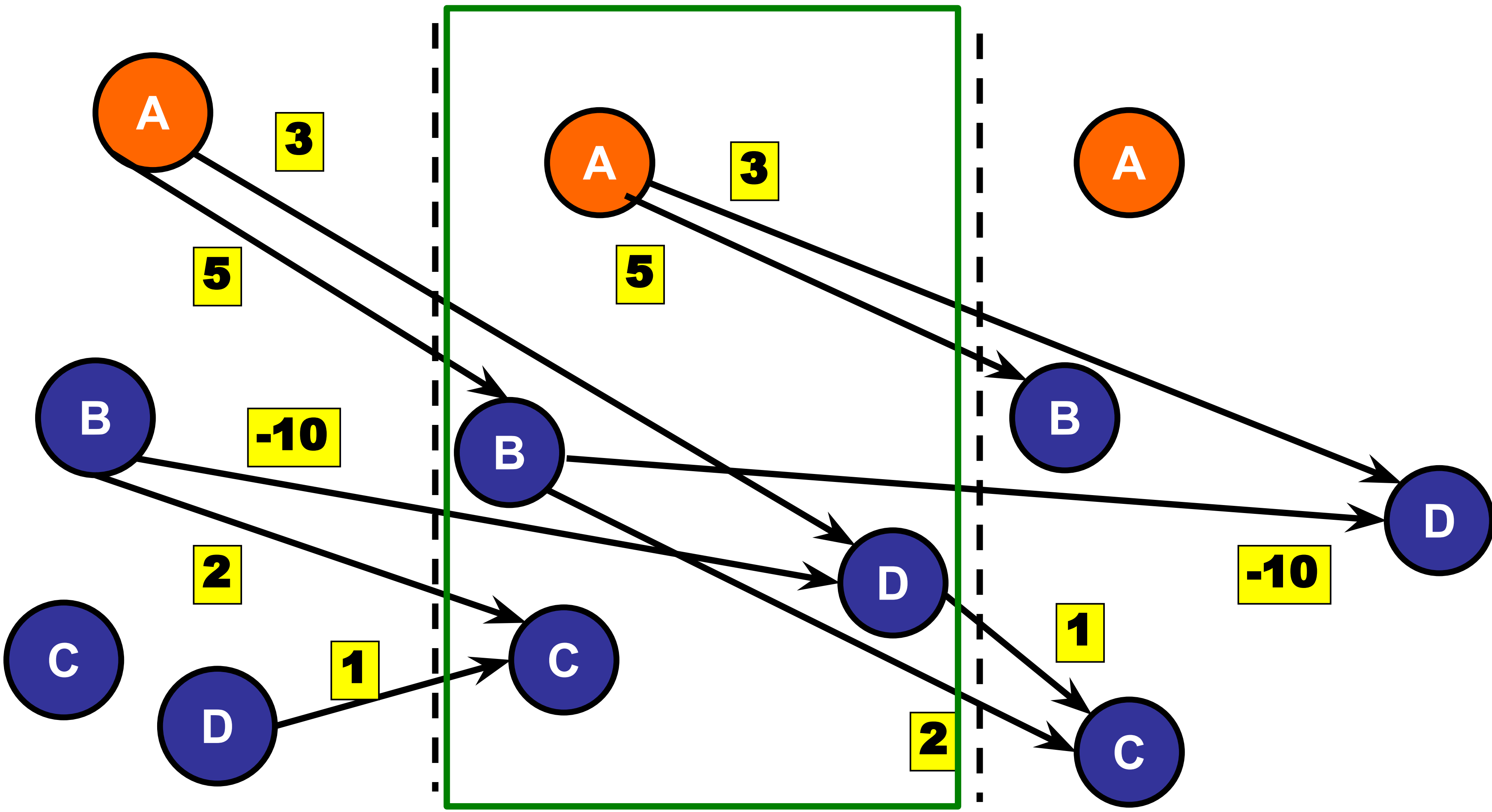
Technique: Graph Modifications

Intuition: If we are on the ith layer, we have taken exactly i steps in the original graph. 0th layer = 0 steps



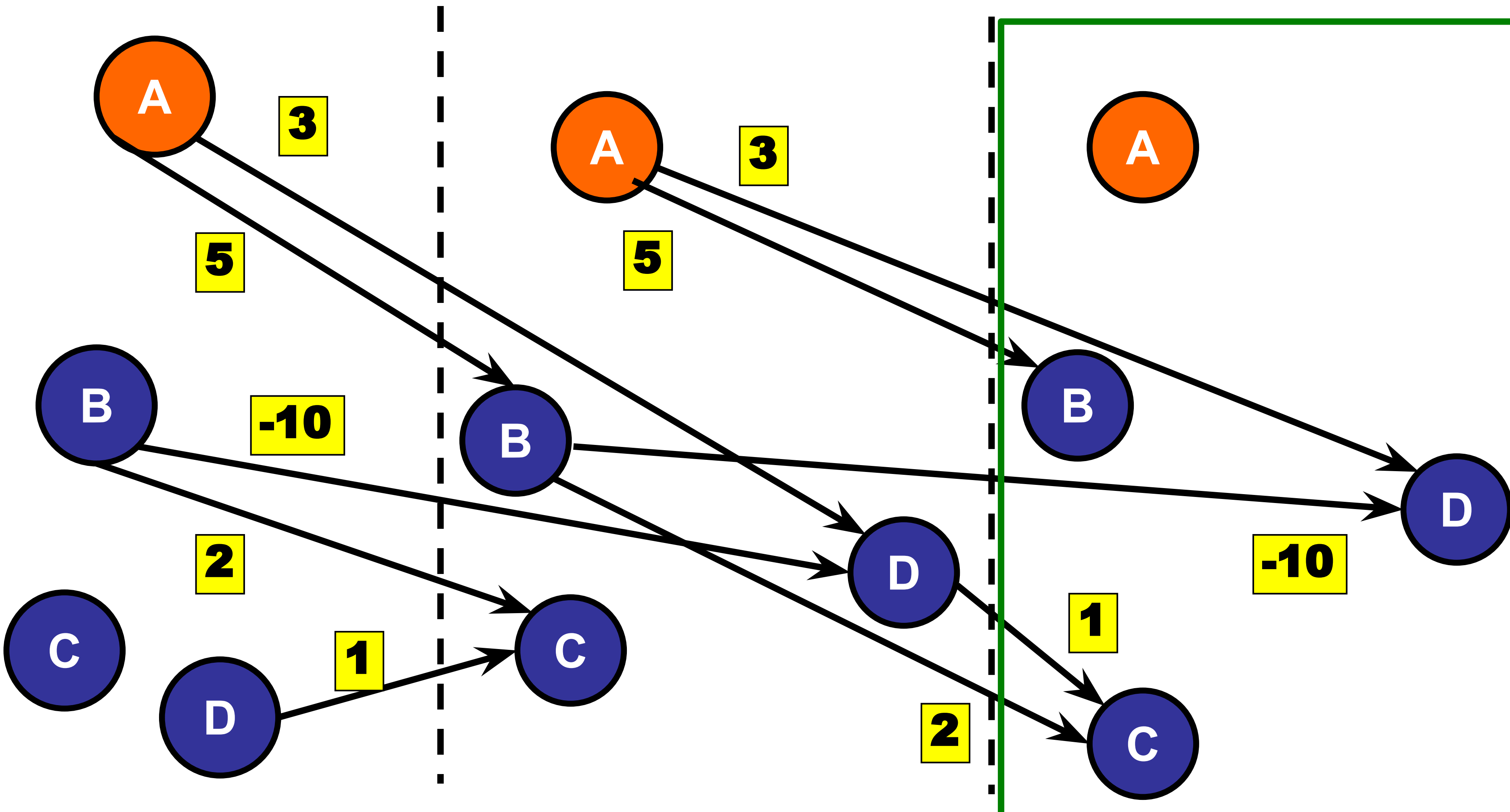
Technique: Graph Modifications

Intuition: If we are on the ith layer, we have taken exactly i steps in the original graph. 1st layer = 1 step



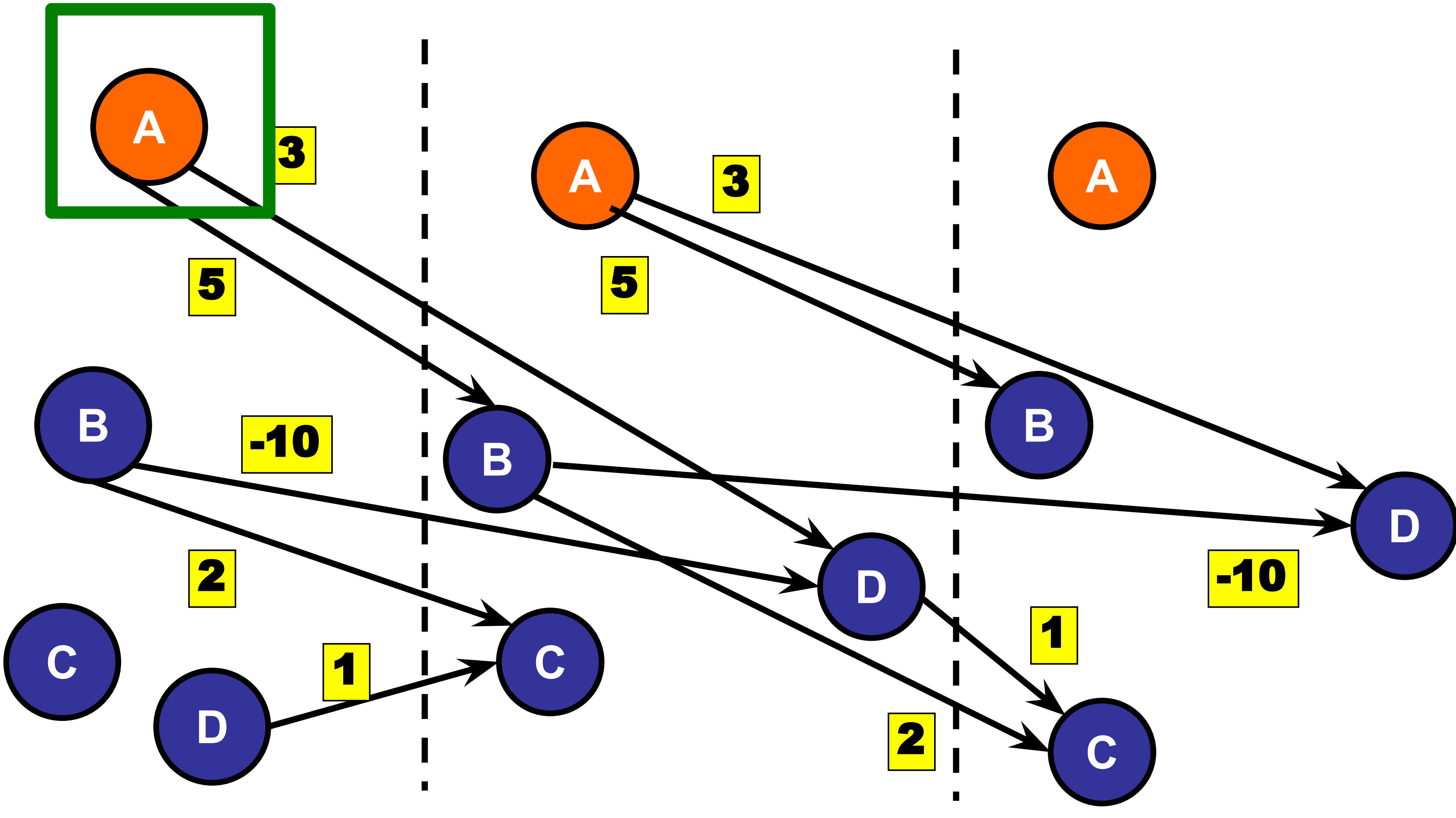
Technique: Graph Modifications

Intuition: If we are on the ith layer, we have taken exactly i steps in the original graph. 2nd layer = 2 steps



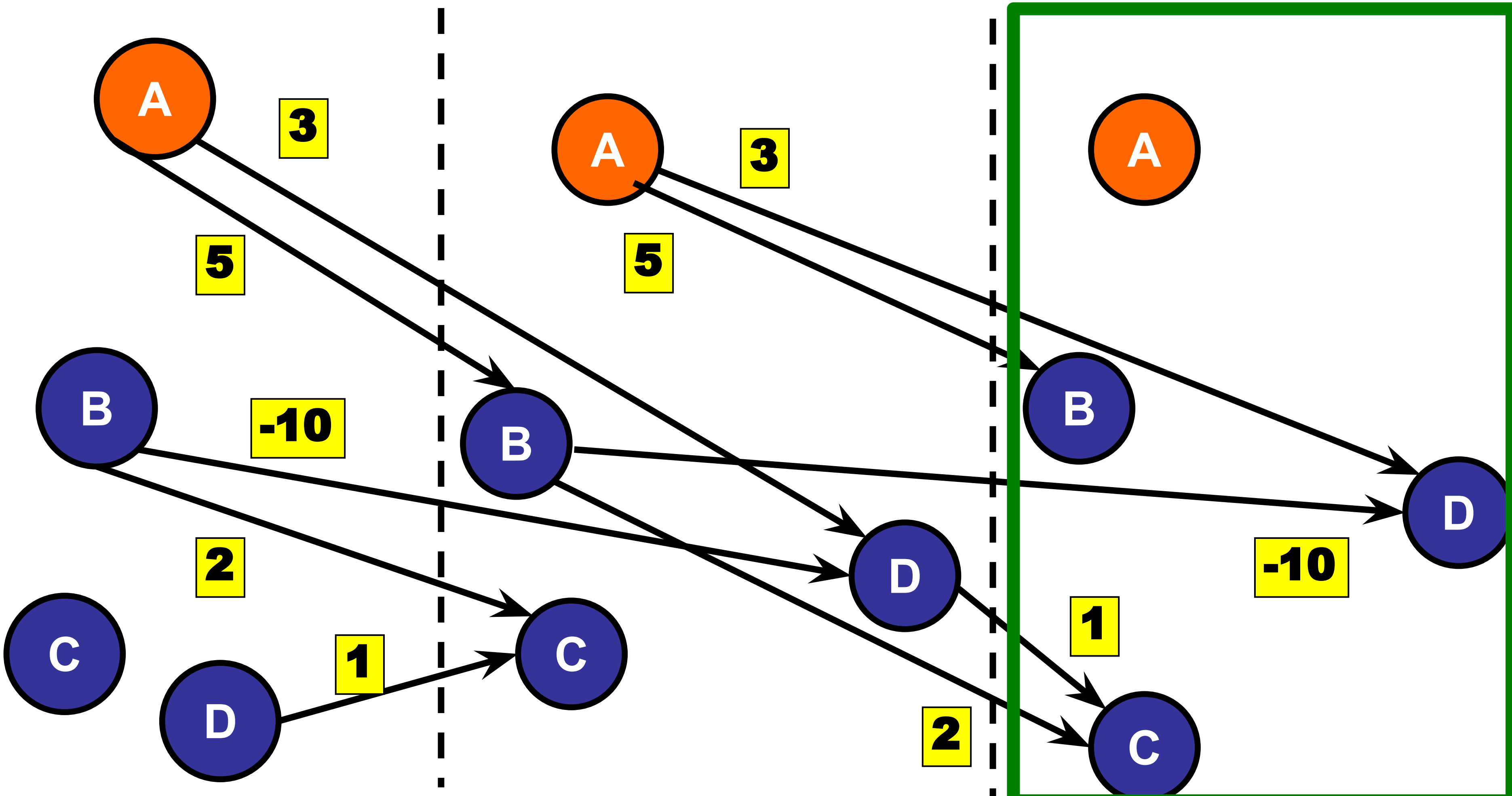
Technique: Graph Modifications

If we SSSP from source node in layer 0



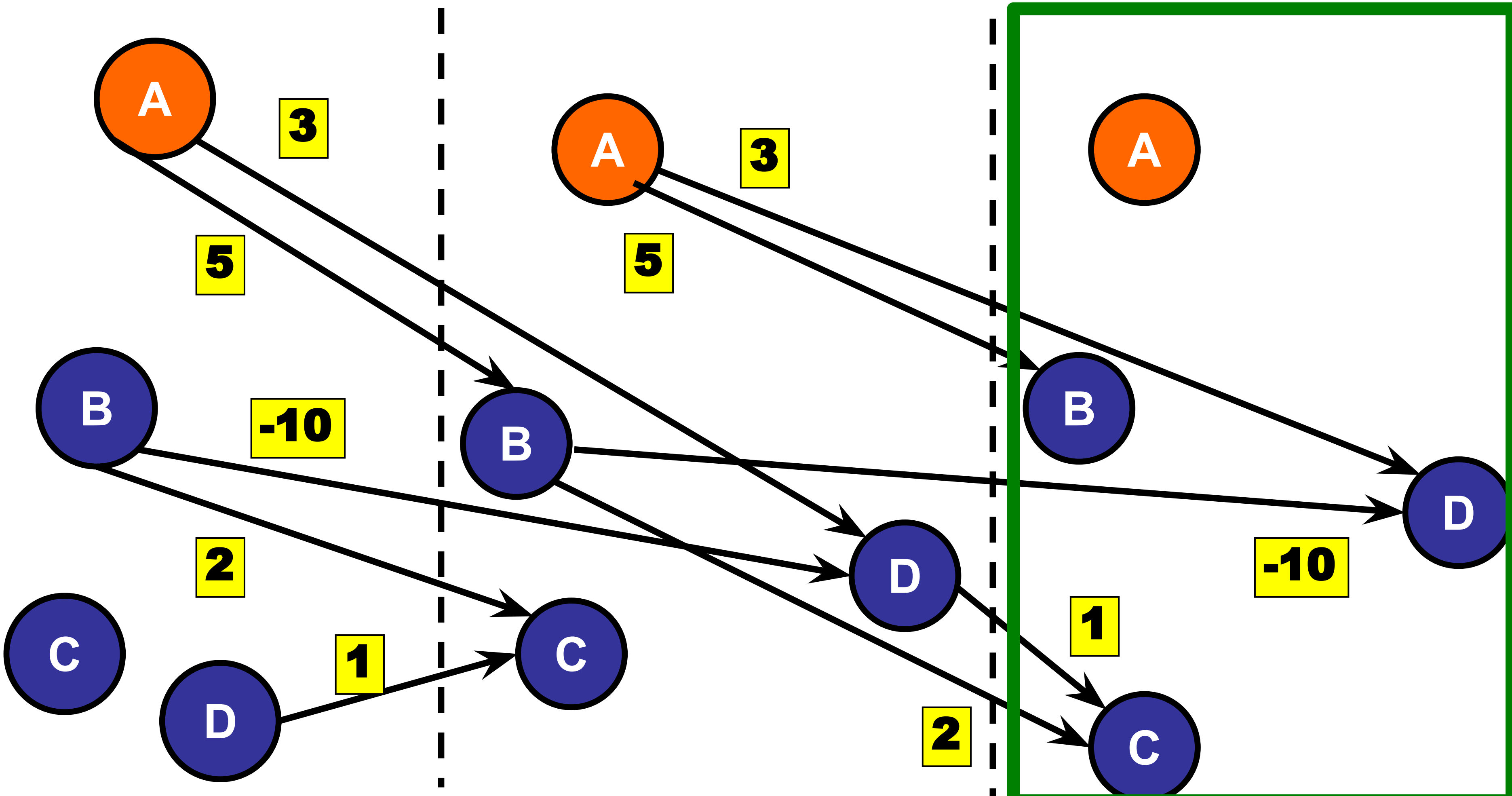
Technique: Graph Modifications

The distances to the nodes in the last layer all took EXACTLY 2 steps.



Technique: Graph Modifications

This technique of creating layers of the same graph is quite general.



Technique: Graph Modifications

This technique of creating layers of the same graph is quite general.

Effectively, we have created a *state space*. Where each node represents a kind of state.

Technique: Graph Modifications

This technique of creating layers of the same graph is quite general.

Effectively, we have created a *state space*. Where each node represents a kind of state.

E.g. Node B in layer 2 represents the state of “taking exactly 2 steps from source node to B.”

Technique: Graph Modifications

This technique of creating layers of the same graph is quite general.

Effectively, we have created a *state space*. Where each node represents a kind of state.

E.g. Node B in layer 2 represents the state of "taking exactly 2 steps from source node to B."

Can encode many things about your traversal through the graph

Technique: Graph Modifications

This technique of creating layers of the same graph is quite general.

Effectively, we have created a *state space*. Where each node represents a kind of state.

E.g. Node B in layer 2 represents the state of "taking exactly 2 steps from source node to B."

Can encode many things about your traversal through the graph

Bonus Slides: 3AM Things:

So Far:

- **Unweighted** graph
  - BFS
  - O(V + E)

- **Weighted** graphs with non-ne**gotiable**

Google slides putting words in my mouth.

Today

Single Source Shortest Paths (SSSP):

- Bellman Ford
  - SSSP on negative edge graphs
  - Negative Cycle Detection

Some graph techniques

Next Week:

MSTs and Union Find