CS2040C Data Structures and Algorithms
Single-source Shortest Path

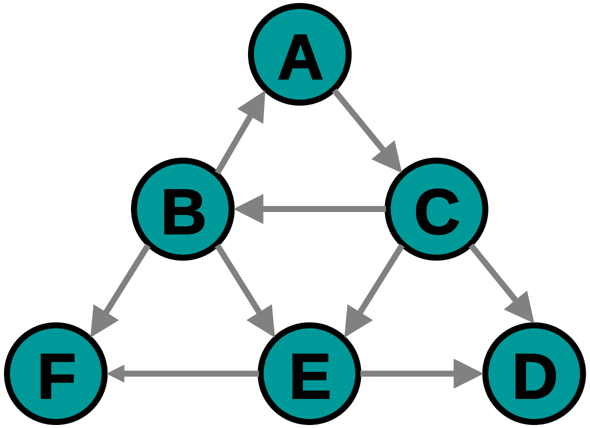
(Bellman-Ford's Algorithm)

Outline

- Definitions
- Unweighted Shortest Paths
- SSSP for positive weighted graphs
- Relax(u,v)
- Bellman-Ford’s Algorithm
- Running Time of Bellman-Ford
- Special Case/s
  - BFS for SSSP

Definitions

- A path on a graph G is a sequence of vertices v_0, v_1, v_2, .. v_n where $(v_i,v_{i+1}) \in E$
- The cost of a path is the sum of the cost of all edges in the path.



In the single-source shortest path (SSSP) problem, we are given a vertex s, and we want to find the path with minimum cost (weight) to **every other vertex**.

Definitions

distance(v): shortest distance so far from s to v

parent(v): previous node on the shortest path so far from s to v

weight(u, v): the weight (cost) of edge from u to v

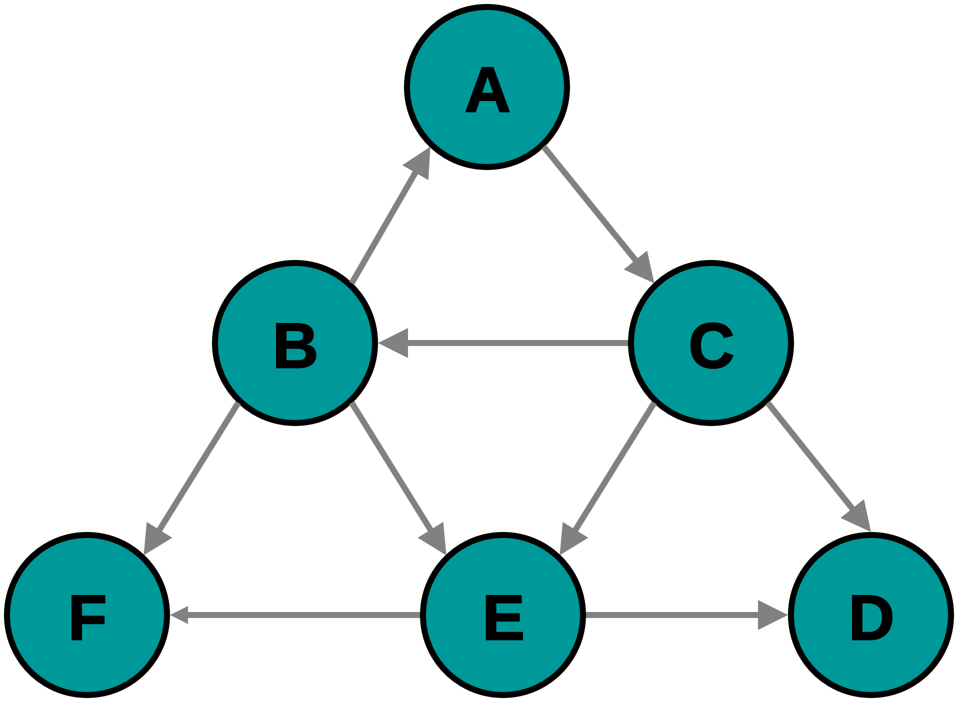
$\delta(u, v)$: actual shortest path from u to v

Note: weight(s, s) = 0

weight(s, u) where u is unreachable = +∞

Unweighted shortest path

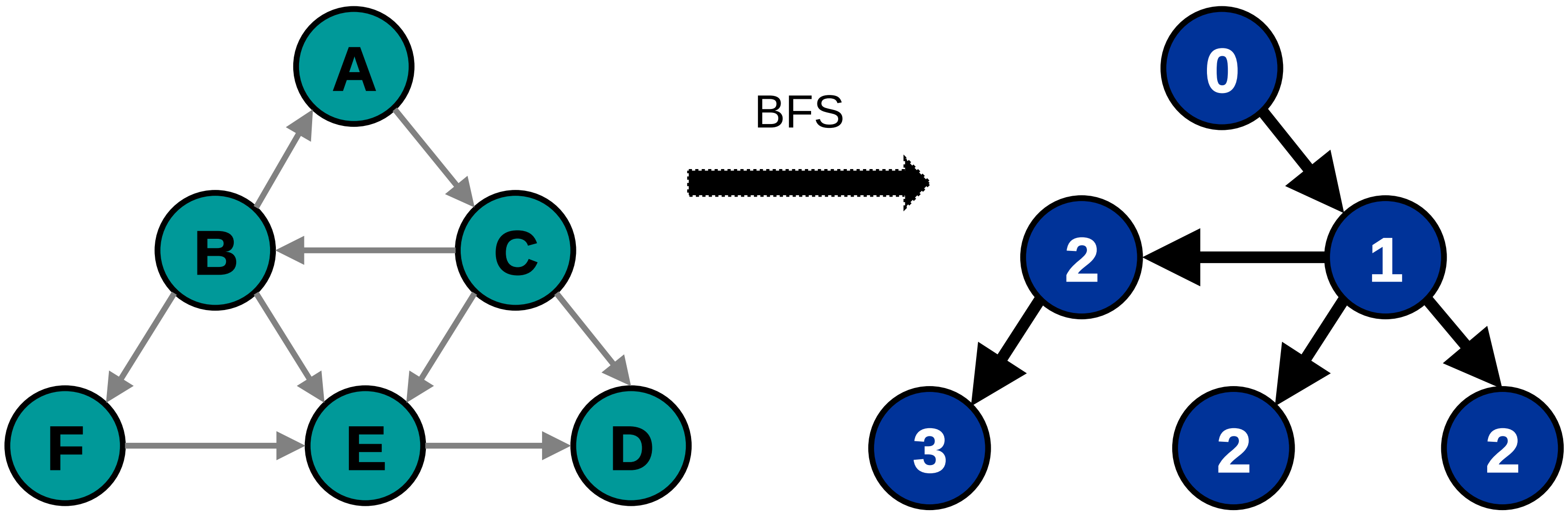
If a graph is unweighted, we can treat the cost of each edge as **1**.



ShortestPath(s)

- The shortest path for an *unweighted* graph can be found using BFS.
- Run BFS(s) where s is the chosen source node
- Trace back the parent pointer from *v* to *s* to get the shortest path
- no. of edges in the path is given by the level of a vertex in the BFS tree (or level – 1, if level of root is 1)

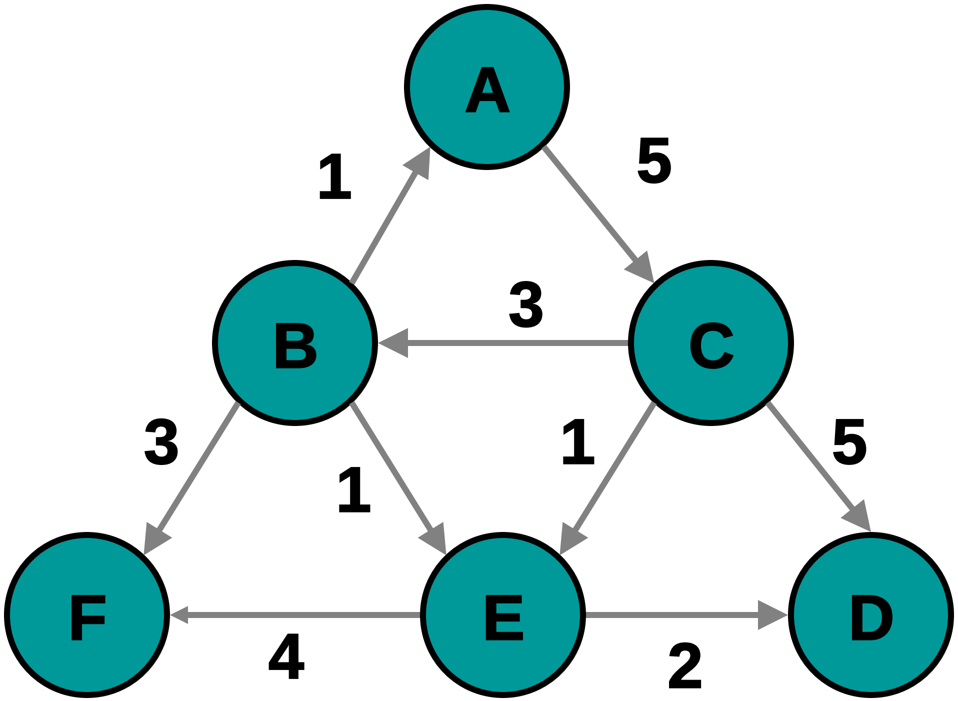
ShortestPath(s)



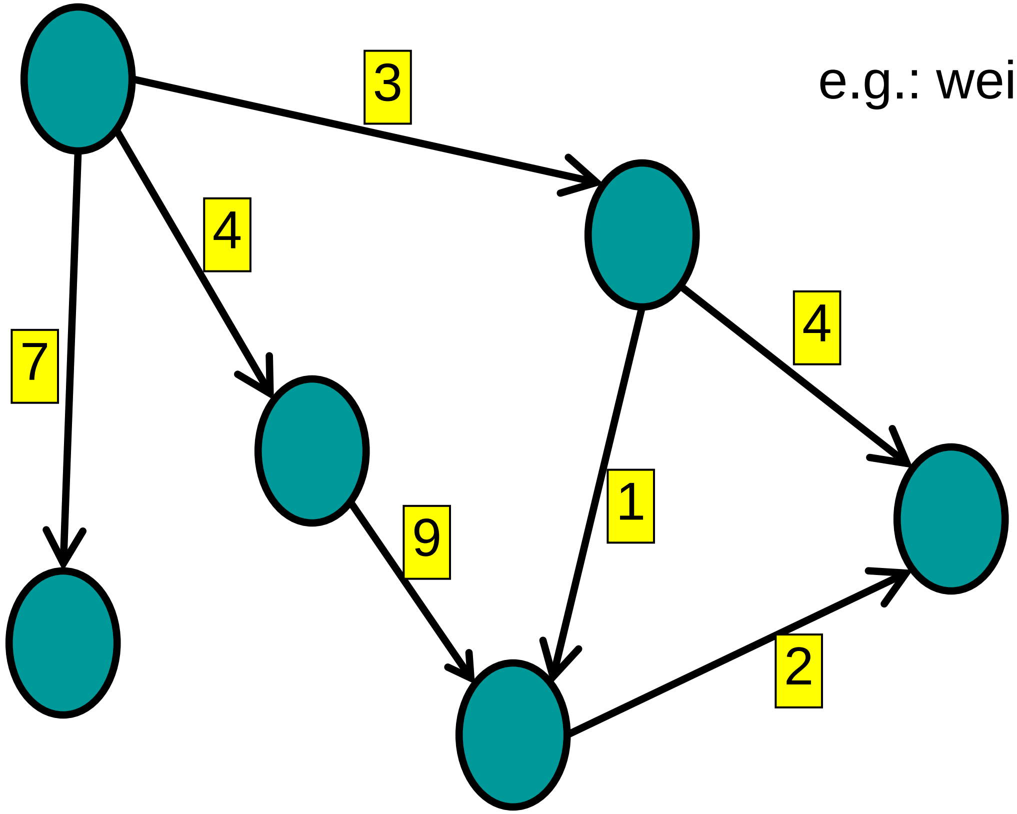
Question: Why does BFS guarantee shortest paths?

Positive weighted shortest path

Will BFS work?



Weighted graphs

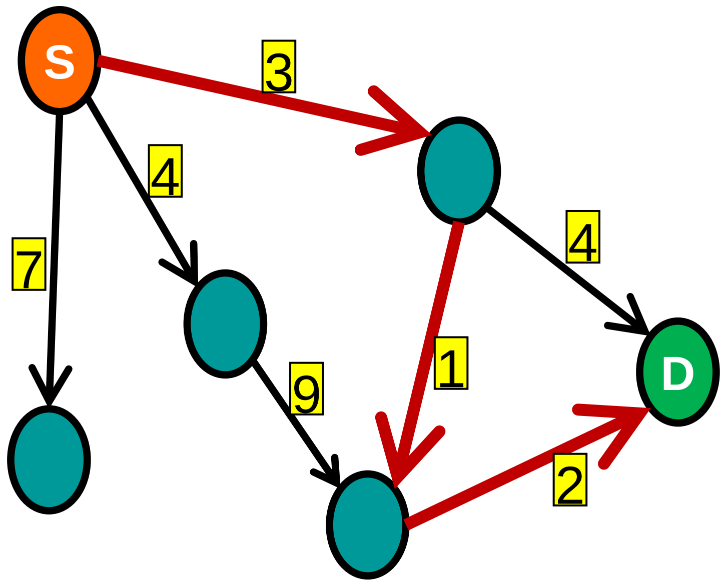


e.g.: weight = distance

Shortest paths

Questions:

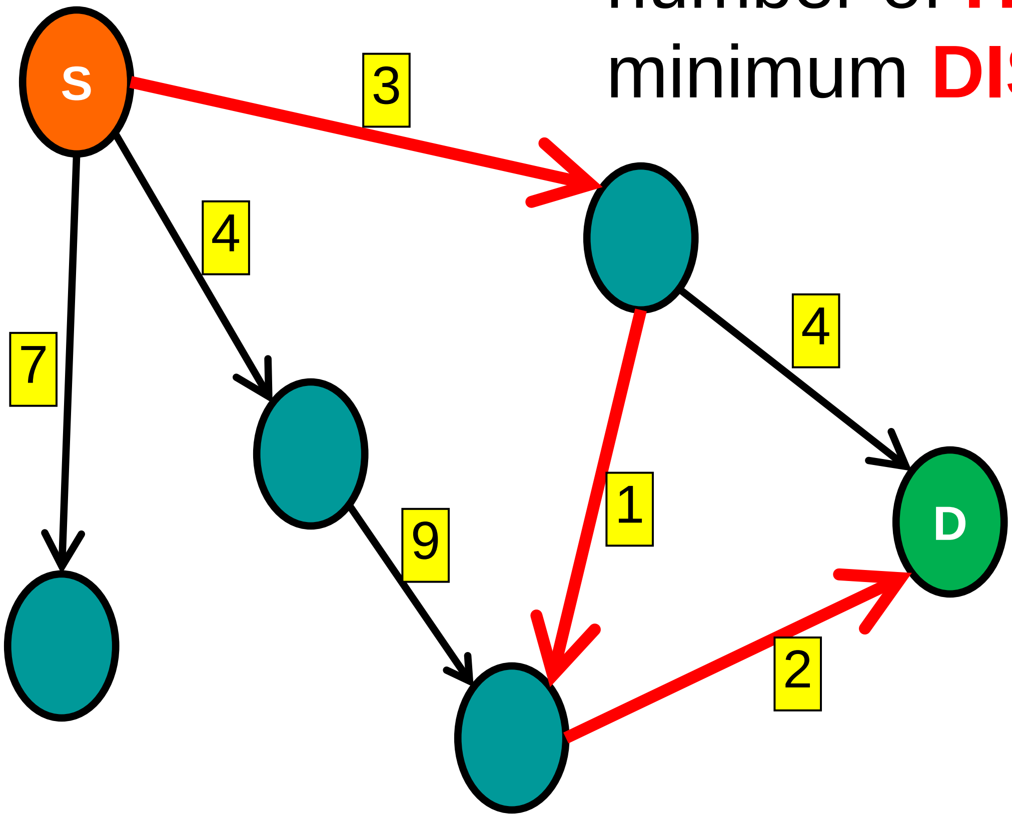
- How far is it from S to D?
- What is the shortest path from S to D?
- Find the shortest path from S to every node
- Find the shortest path between every pair of nodes



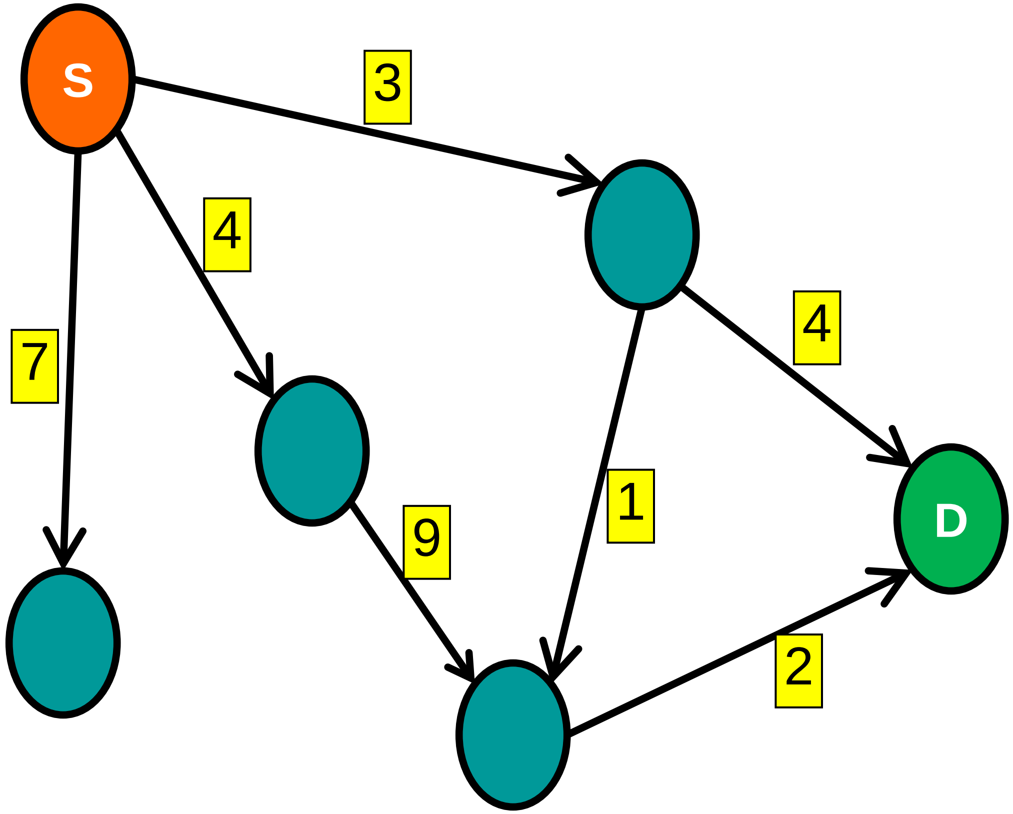
Will BFS work?

Cannot use **BFS**

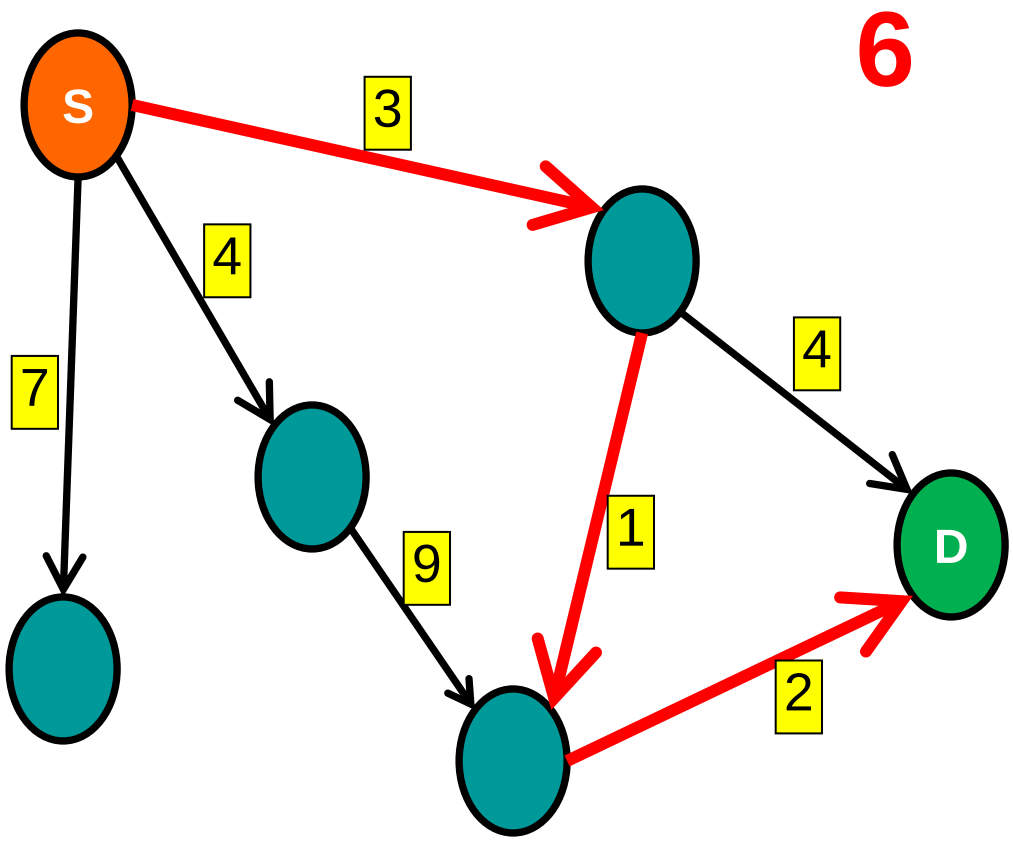
BFS finds minimum number of **HOPS** not minimum **DISTANCE**



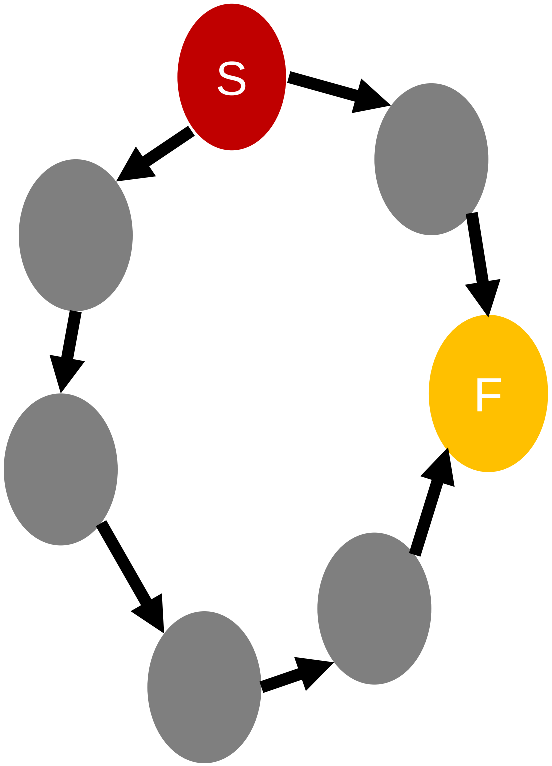
Distance from the source?



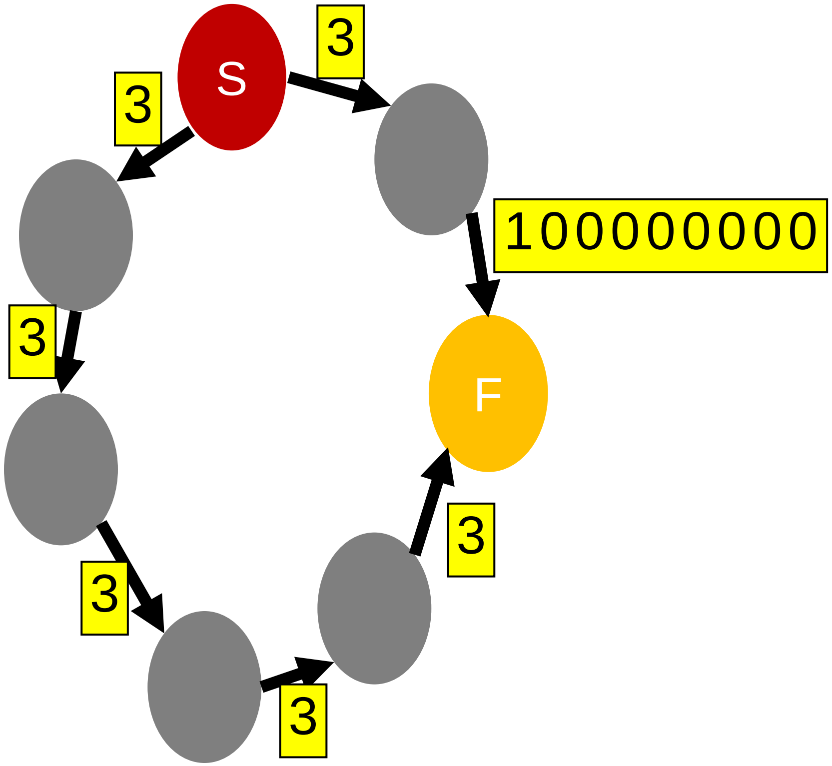
Distance from the source?



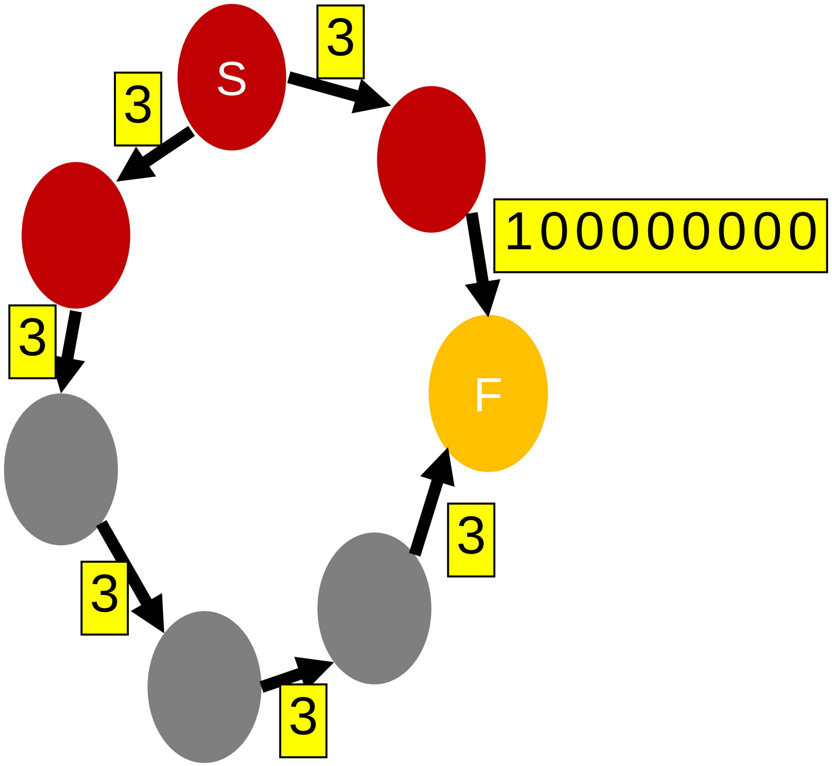
An example: BFS



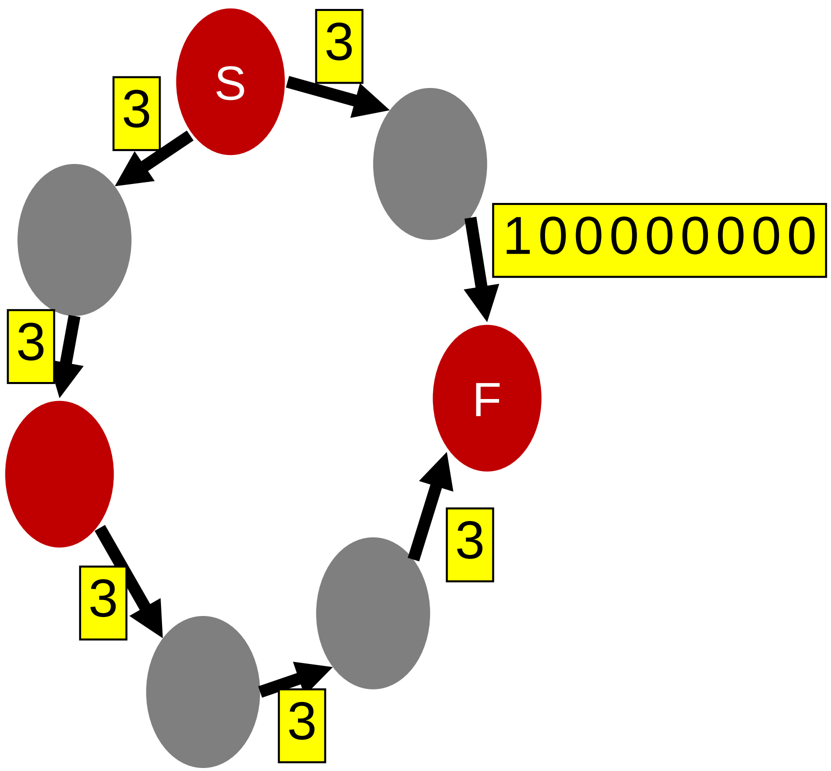
An example: BFS



An example: BFS

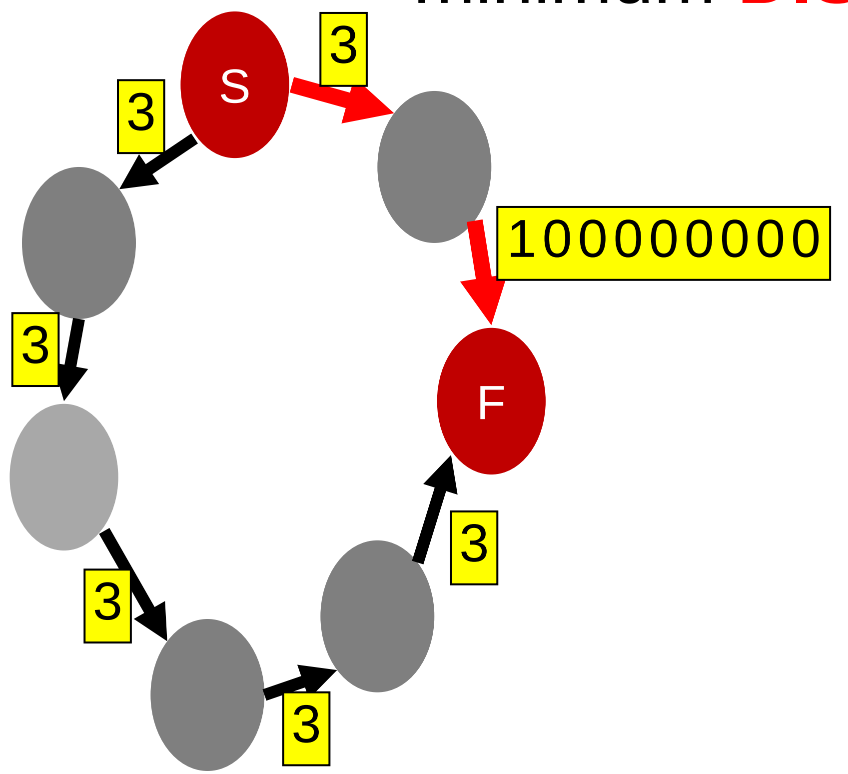


An example: BFS

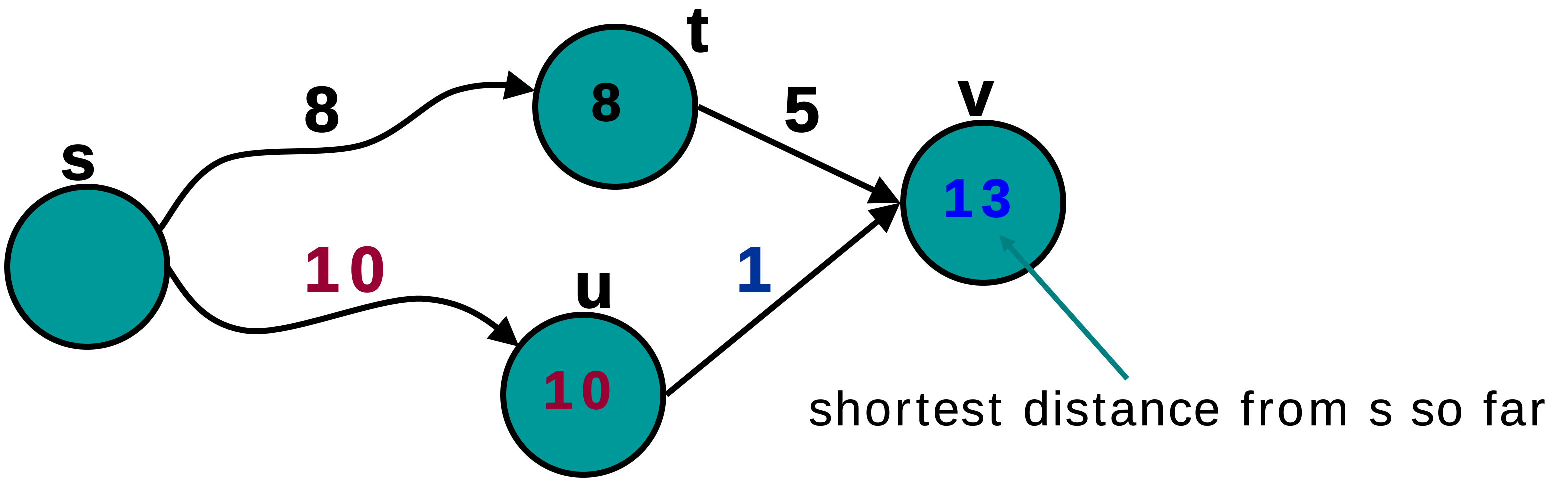


An example: BFS

BFS finds minimum number of **HOPS** not minimum **DISTANCE**



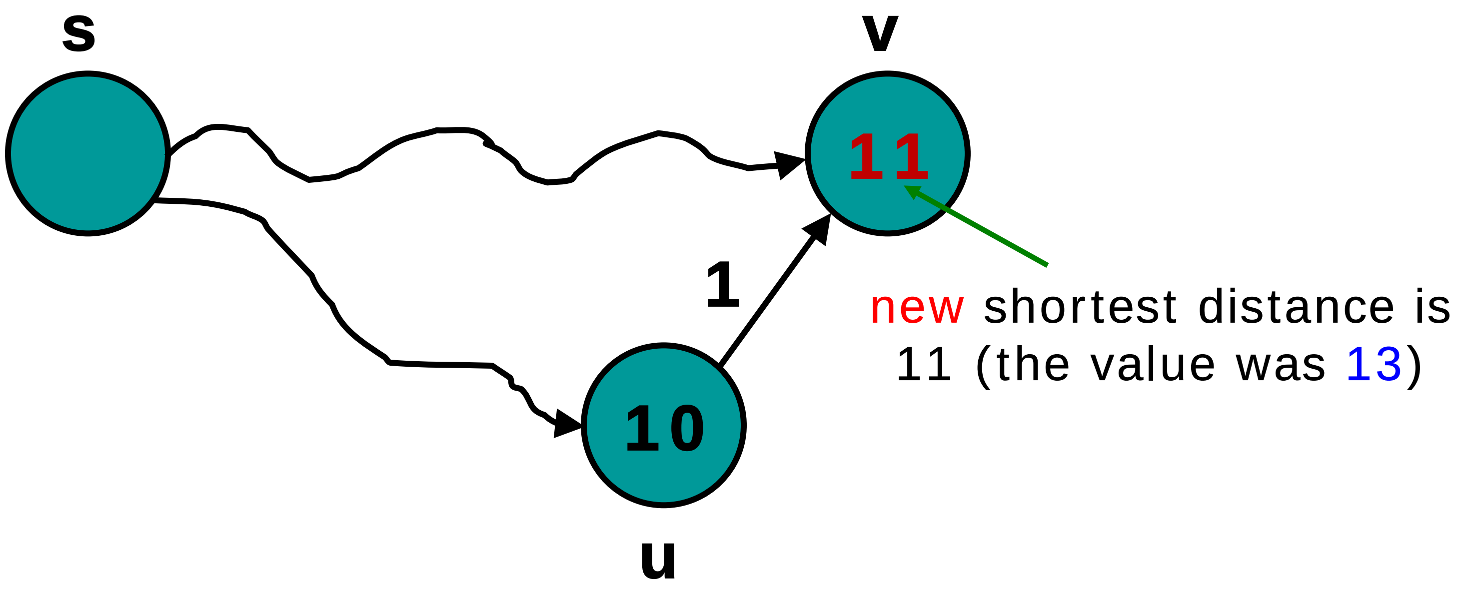
BFS(s) does not work



- Must keep track of shortest distance from the source node so far for each node
- Observation 1: If we found a new shorter path, update the distance.

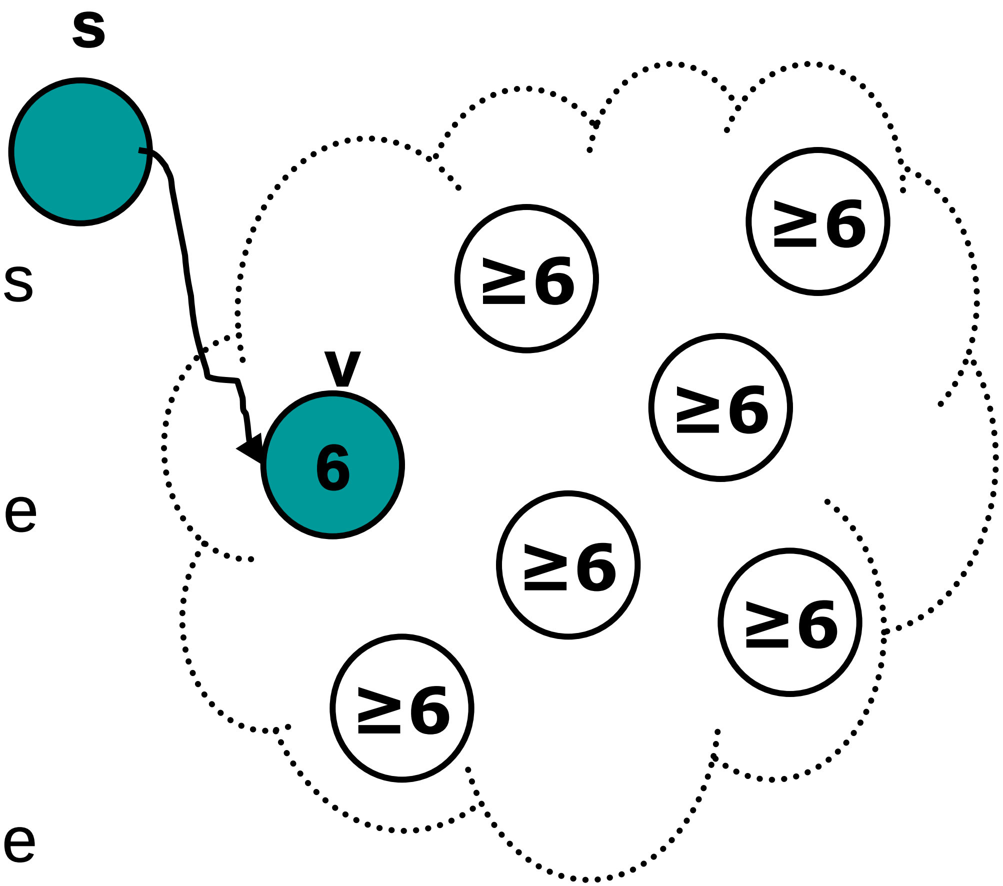
Observation 1

- In the following figures, we label a node with the shortest distance discovered so far from the source.
- Here is the basic idea that will help us solve our shortest path problem.
- If the current shortest distance from s to v is 13, to u is 10, and the cost of edge (u,v) is 1, then we have discovered a shorter path from s to v (through u).

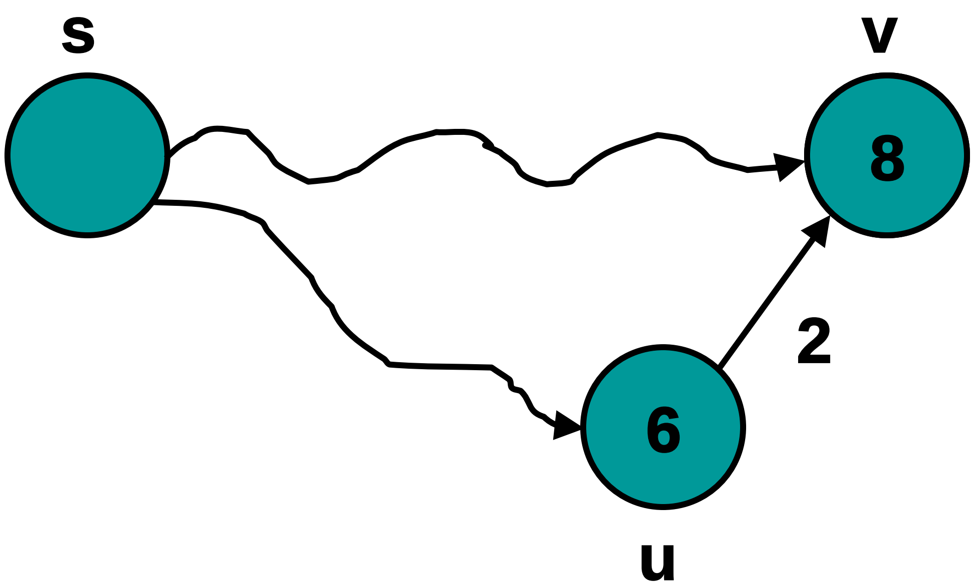


Observation 2 (for positive costs only)

- The second idea is that if we know the shortest distance so far from s to v is 6, and the shortest distance so far from s to other nodes (in white) is bigger or equal to 6, then there cannot be a shorter path to v through these other nodes.
- This is true only if the costs are positive!



Example



distance(v) = 8

weight(u,v) = 2

parent(v) = u

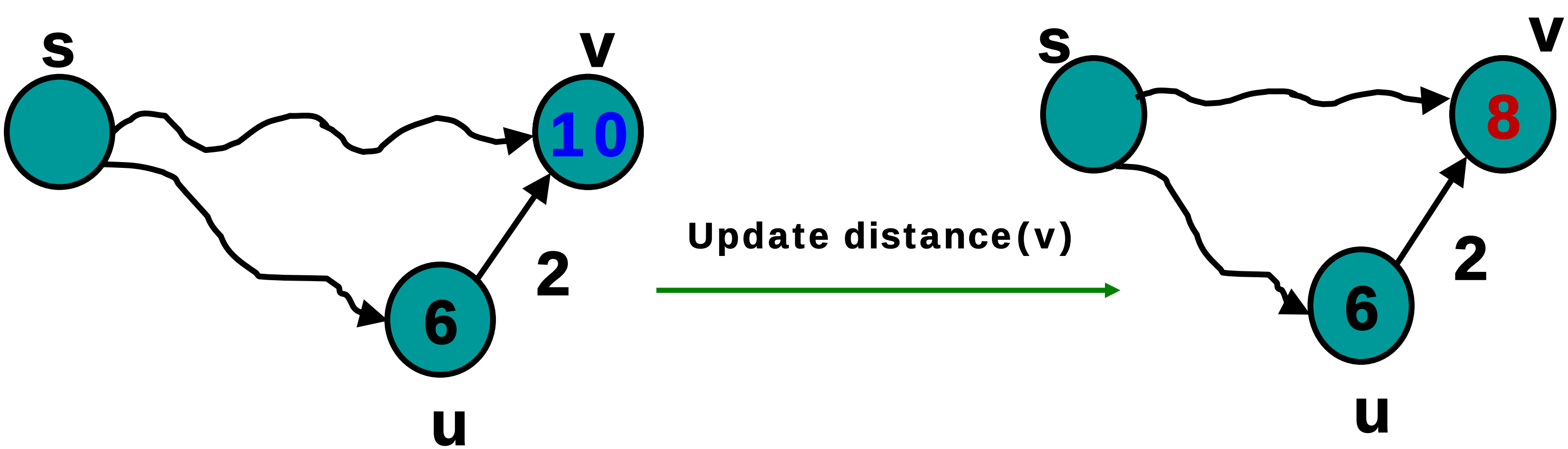
Relax(u, v) - based on observation 1

d = distance(u) + weight(u,v)

if distance(v) **>** d **then** **// found a new shorter distance**

distance(v) = d // update the distance and parent

parent(v) = u



Data structures needed

- Array/Vector **dist** of size **V** (dist: distance)

  Initially **dist[u]** = 0 if u = s; else **D[u]** = +∞ (10^9)
  - **dist[u]** decreases as we find better (shorter) paths
  - **dist[u]** ≥ **δ(s, u)** throughout the execution of SSSP algorithm
  - **dist[u]** = **δ(s, u)** at the end of SSSP algorithm
- Array/Vector **p** of size **V** (p: parent/predecessor)
  - **p[u]** = the predecessor on best path from source **s** to **u**
  - **p[u]** = NULL (initially not defined, we can use -1)

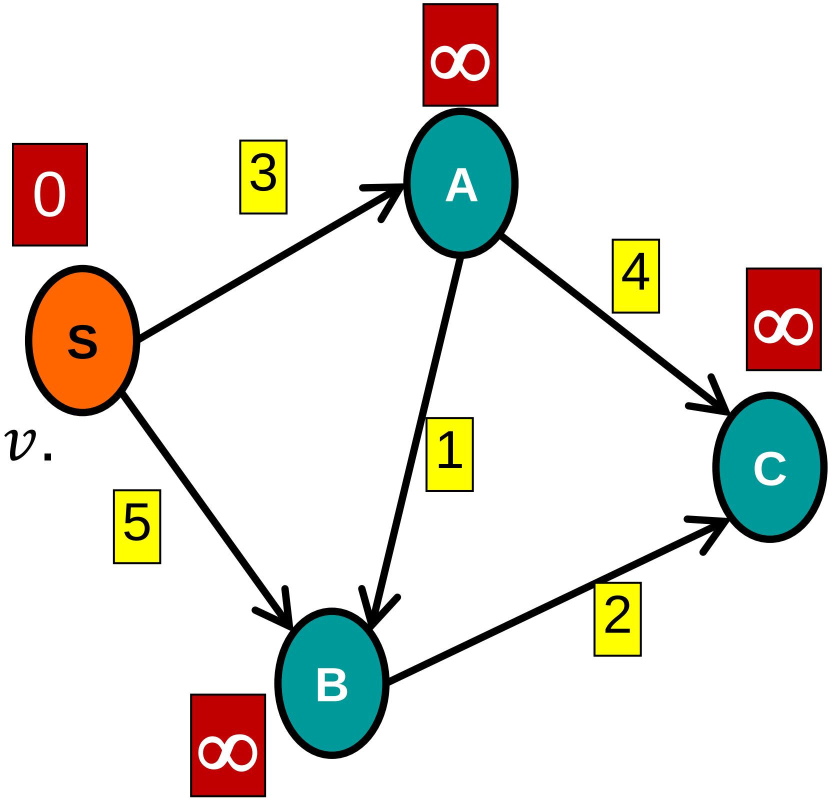
  This array/Vector **p** describes the resulting SSSP spanning tree

shortest paths

Maintain estimate for each distance:

- **Reduce** estimate
- **Invariant:** estimate ≥ shortest distance

The idea:
relax(u,v):
Test if the best way to get from $s \to v$ is to go from $s \to u$, then $u \to v$.
Update dist

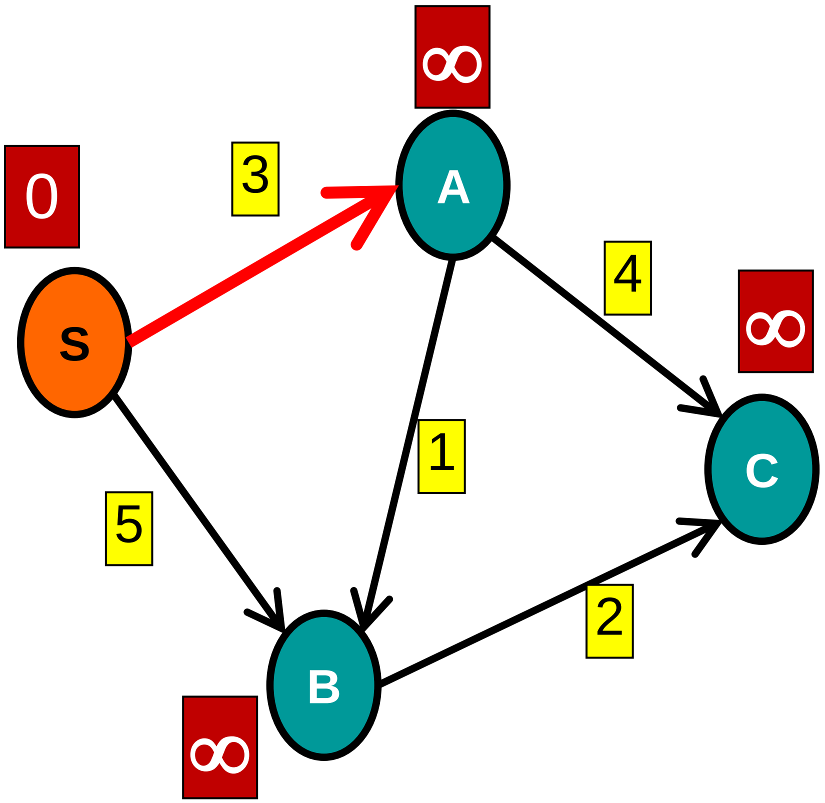


shortest paths

Maintain estimate for each distance:

relax(S, A)

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

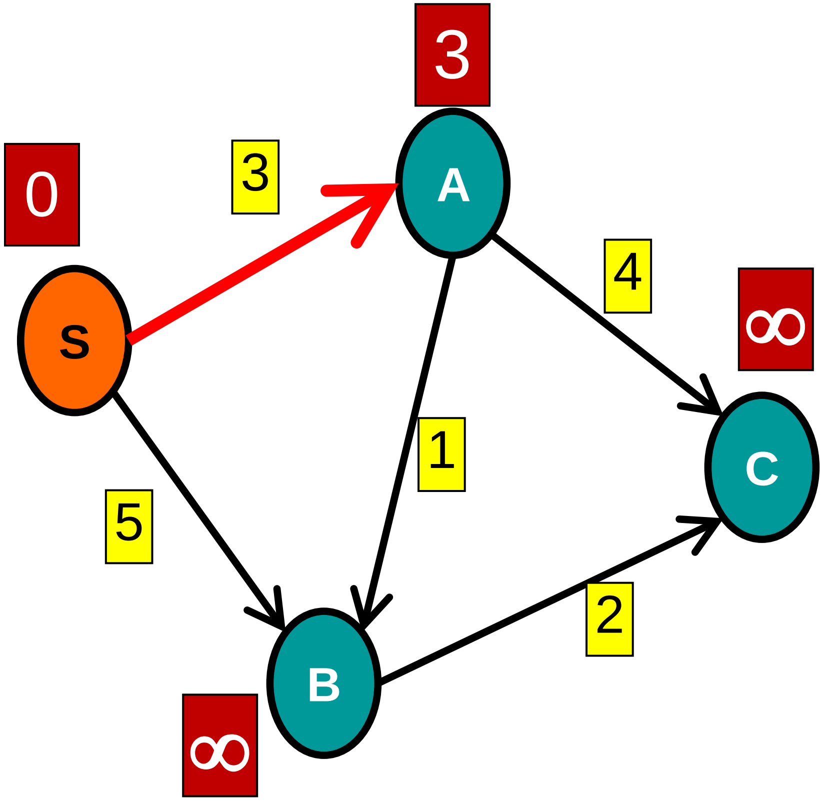


shortest paths

Maintain estimate for each distance:

relax(S, A)

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

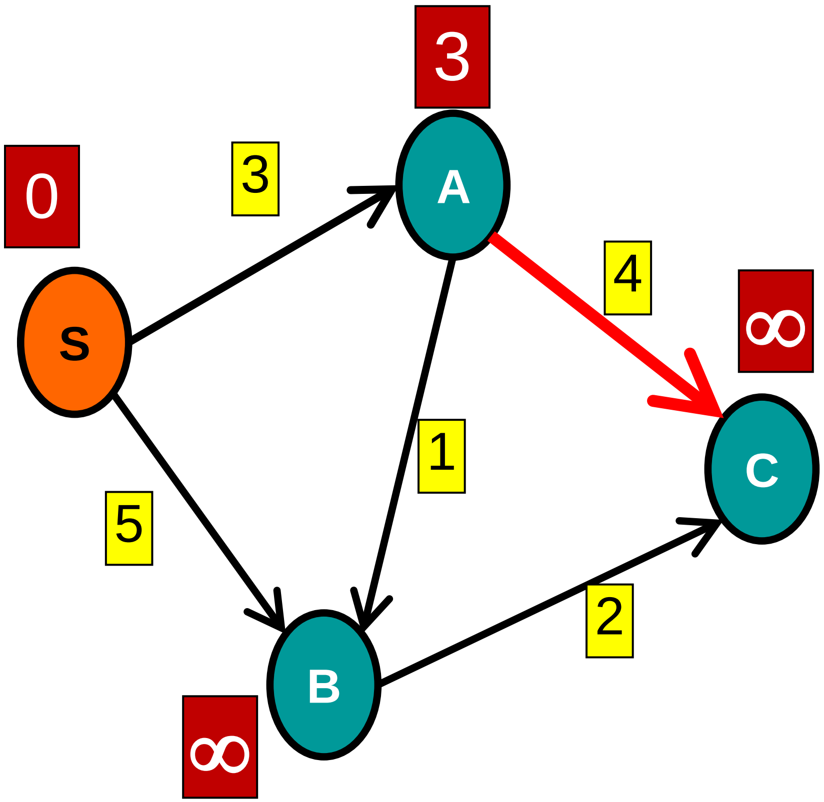


shortest paths

Maintain estimate for each distance:

relax(A, C)

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

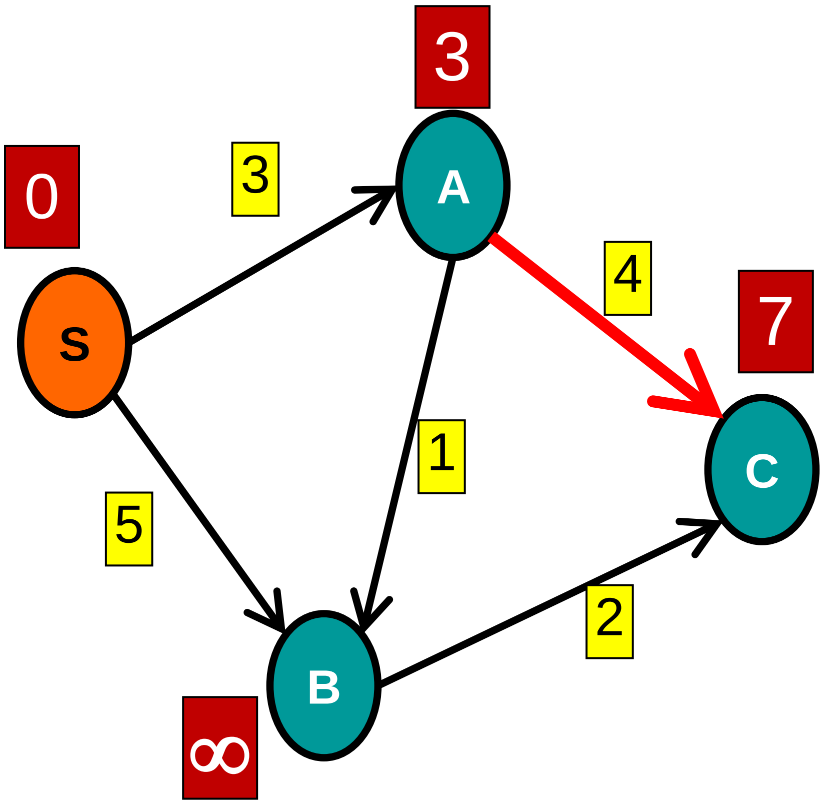


shortest paths

Maintain estimate for each distance:

relax(A, C)

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

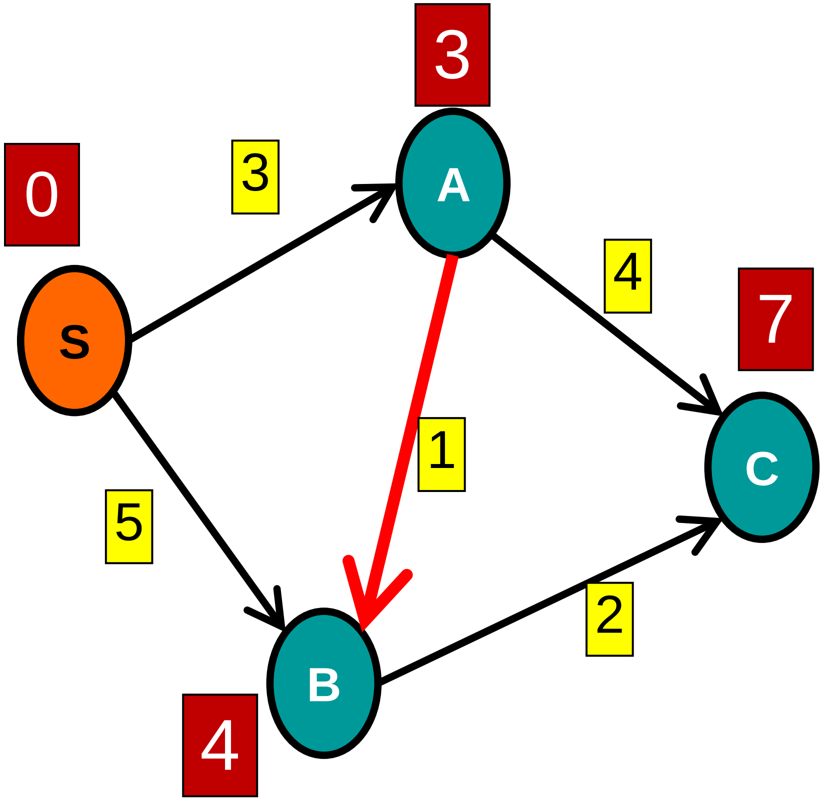


shortest paths

Maintain estimate for each distance:

relax(A, B)

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

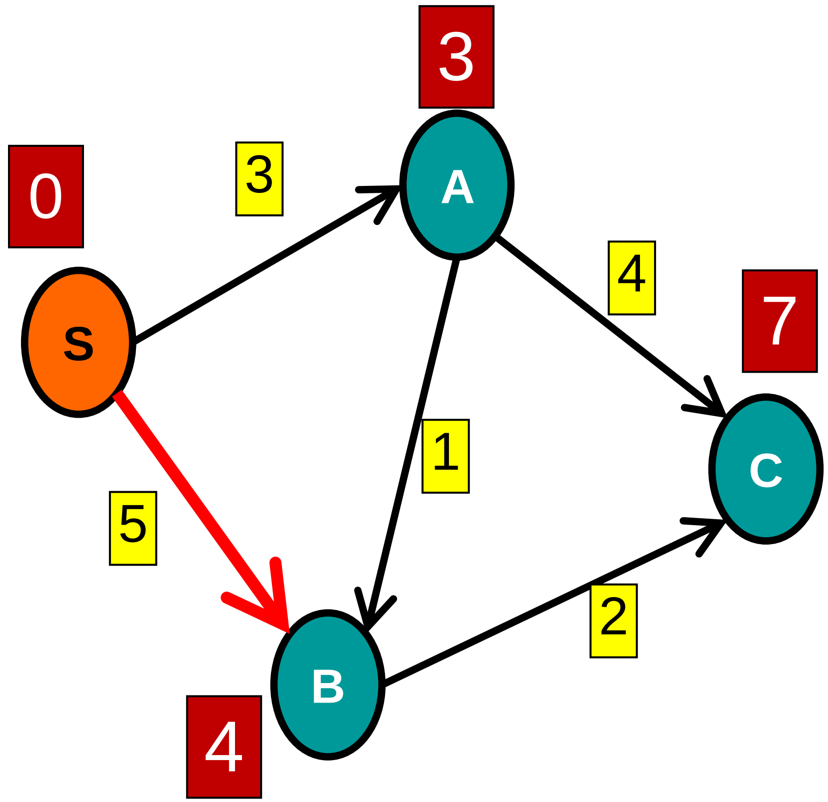


shortest paths

Maintain estimate for each distance:

relax(S, B)

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

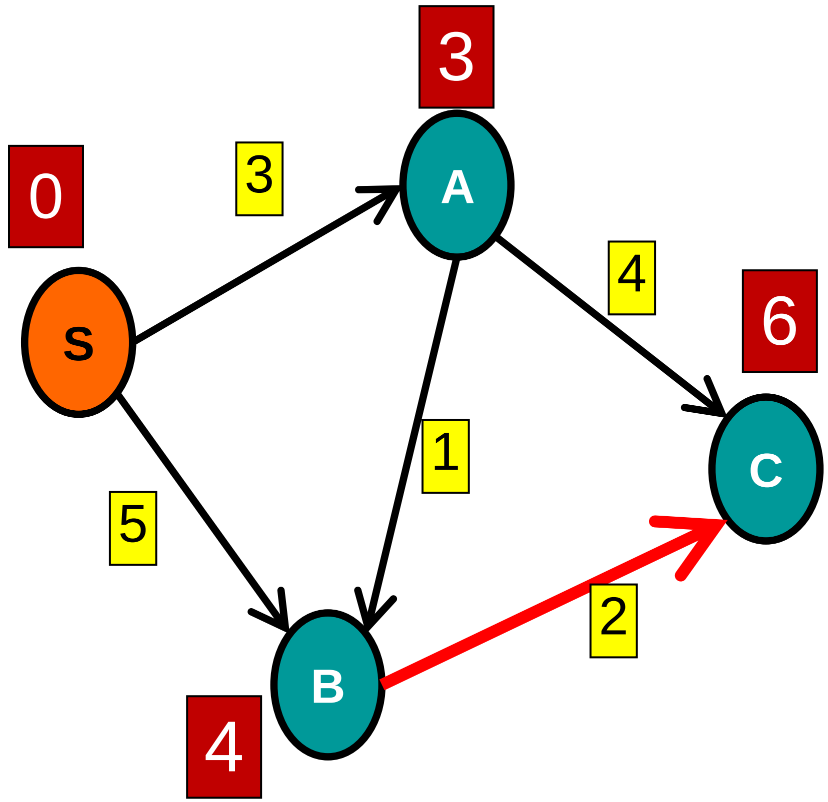


shortest paths

Maintain estimate for each distance:

relax(B, C)

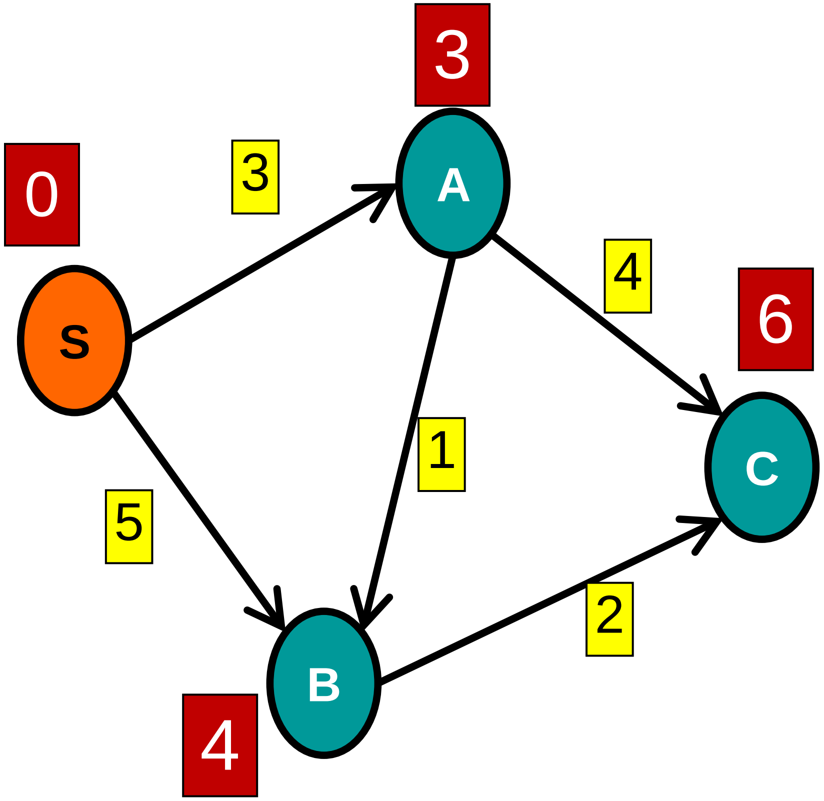
```
relax(int u, int v){
   if (dist[v] > dist[u] +
        weight(u,v))
     dist[v] = dist[u] +
        weight(u,v);
     p[v] = u;
}
```



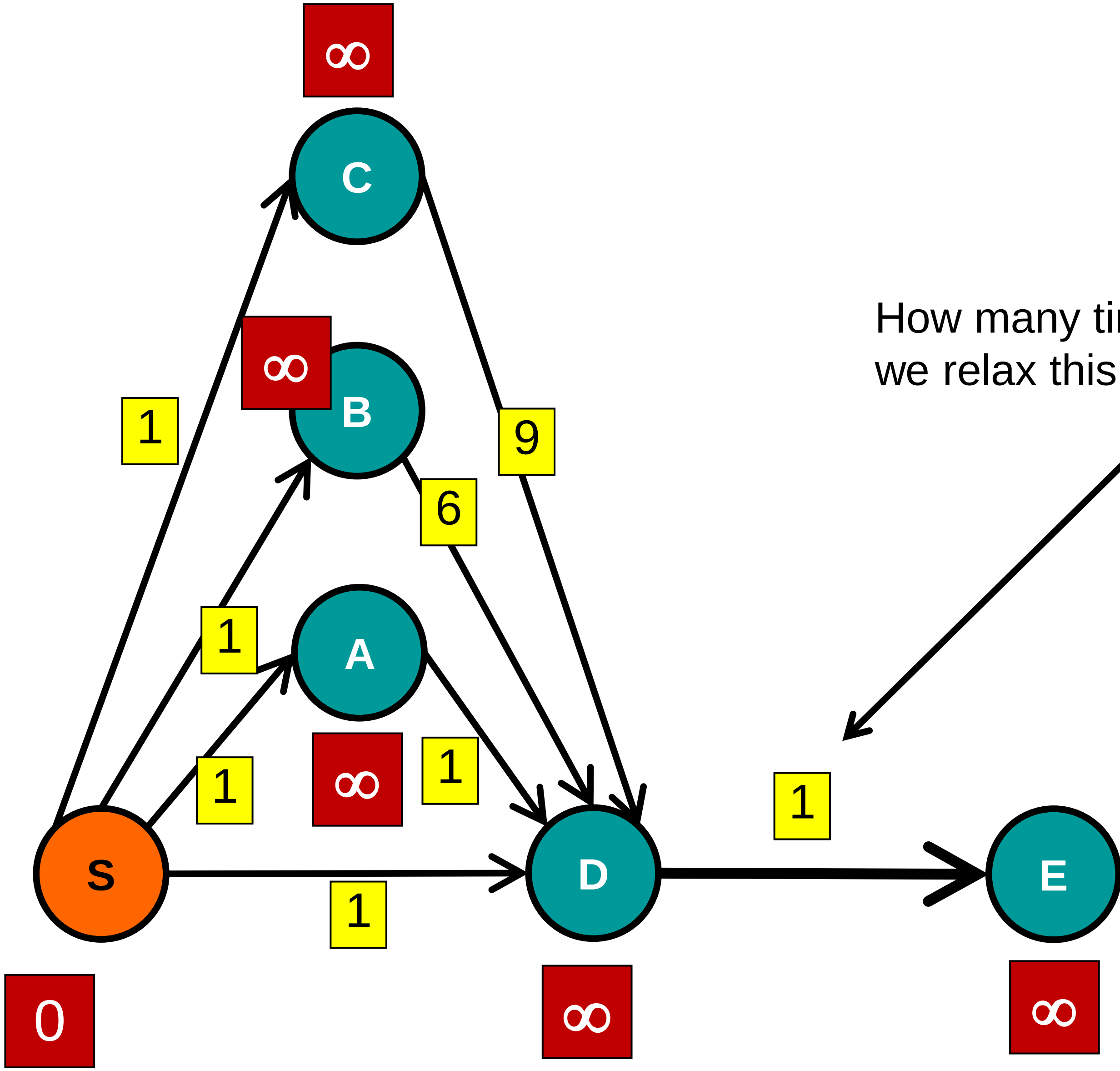
shortest paths

Maintain estimate for each distance:

```
for Edge e in graph
    relax(e)
```

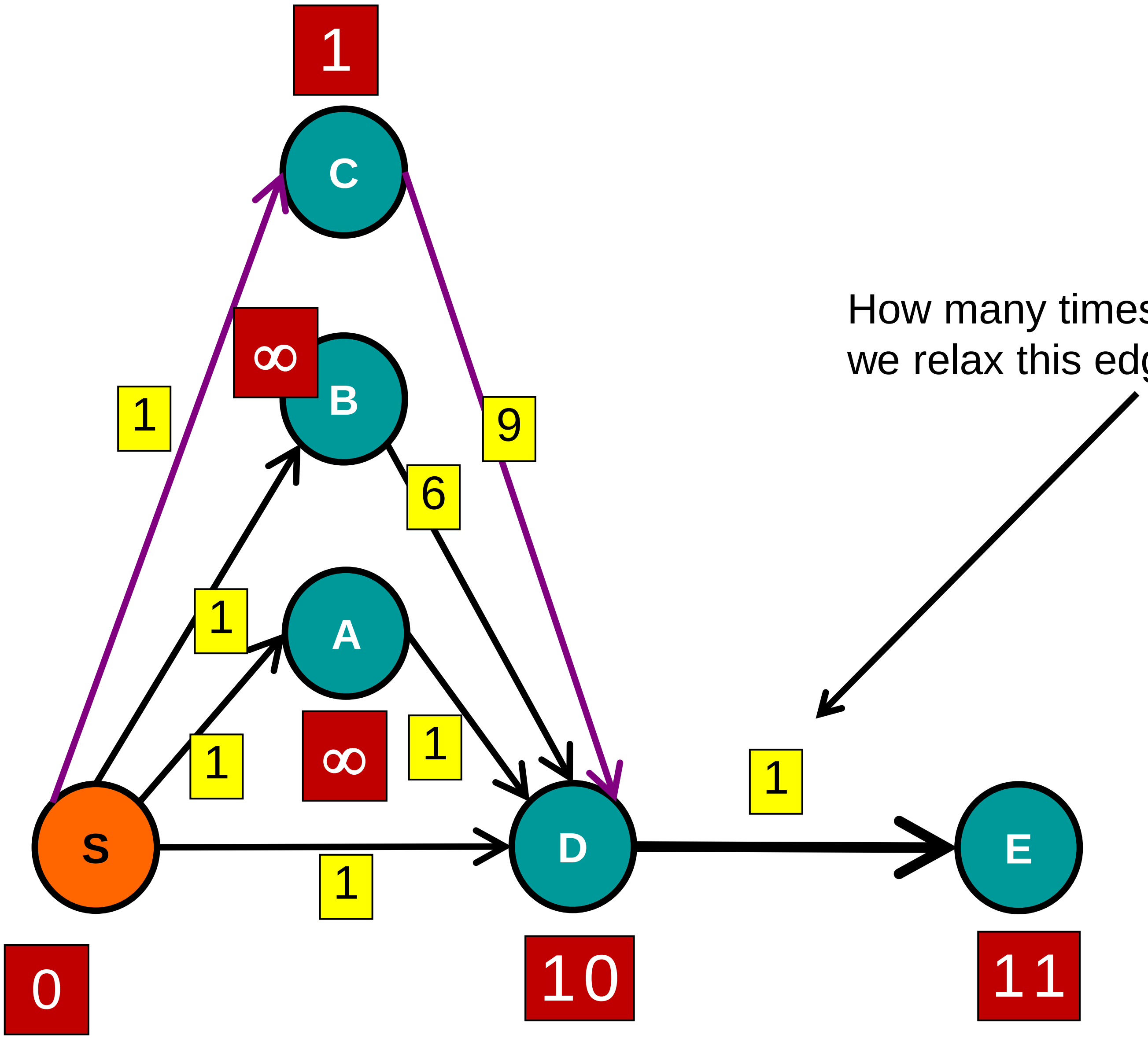


relaxation



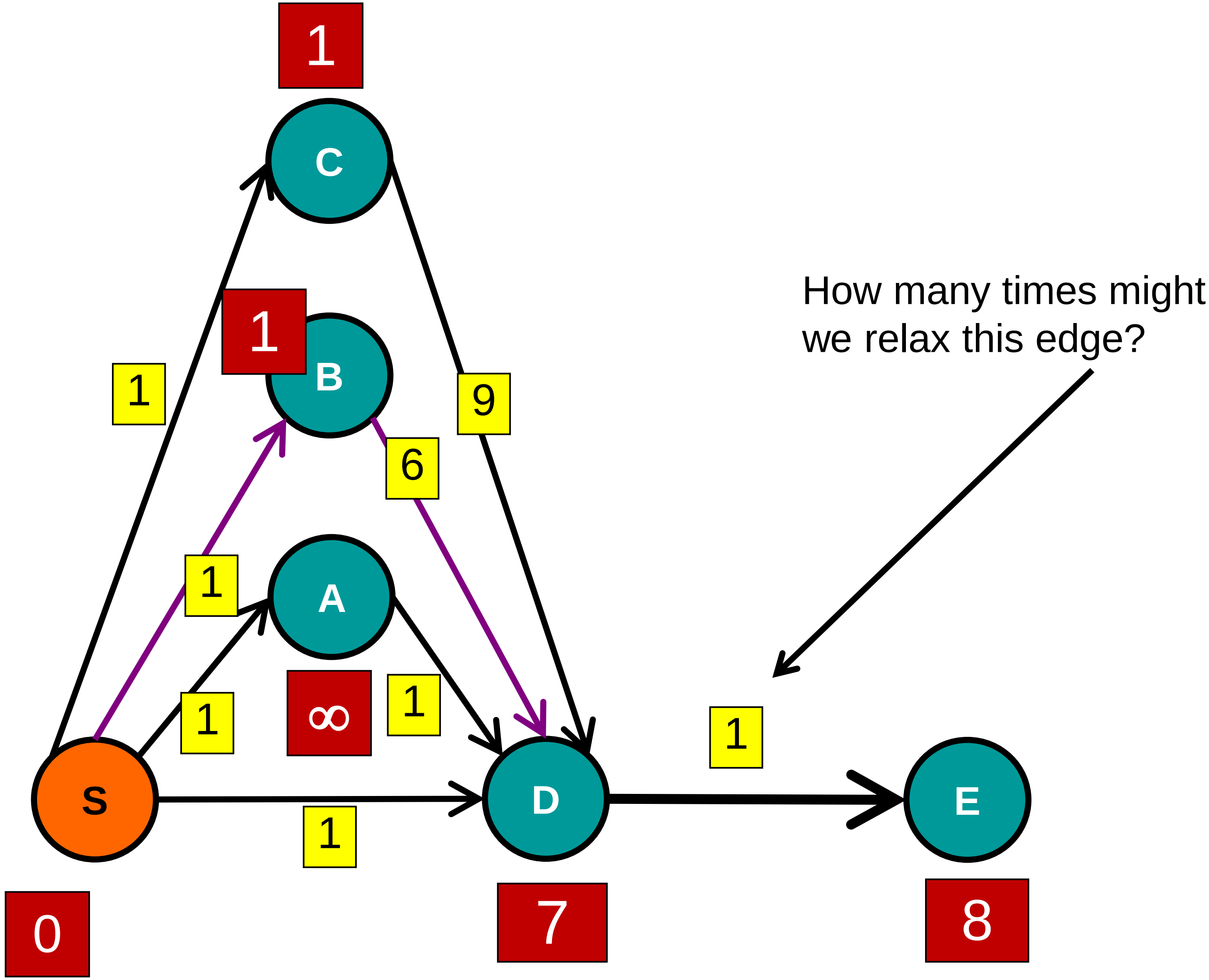
How many times might we relax this edge?

relaxation



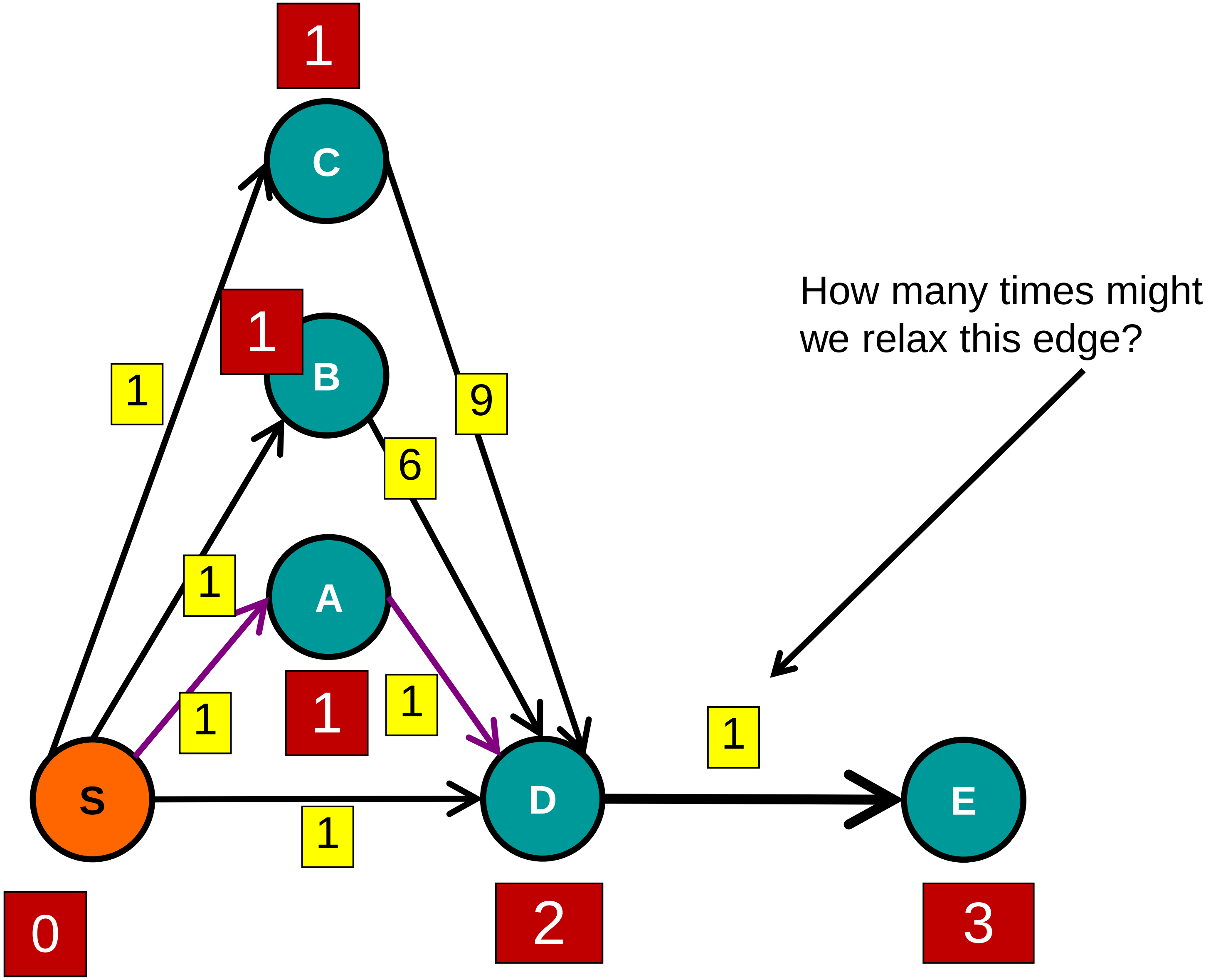
How many times might we relax this edge?

relaxation



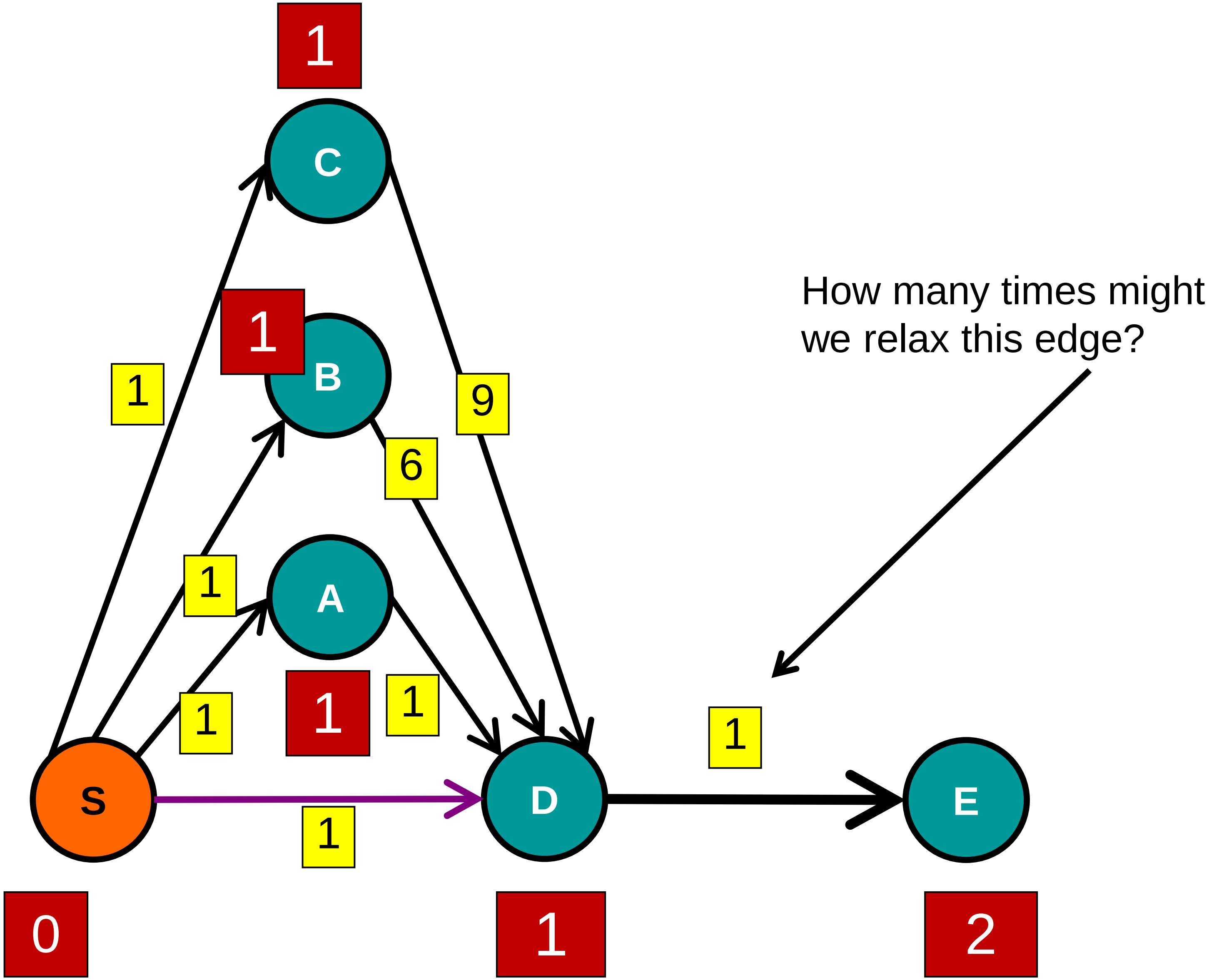
1
C
1
B
1
9
6
1
A
1
∞
1
S
D
1
E
0
7
8
How many times might
we relax this edge?

relaxation



1
C
1
B
1
9
6
1
A
1
1
1
1
S
1
D
E
0
2
3
How many times might
we relax this edge?

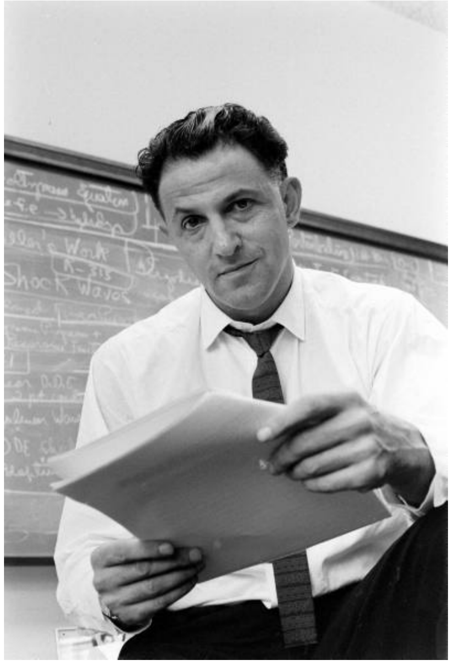
relaxation



1
C
1
B
1
9
6
1
A
1
1
1
S
D
E
1
0
1
2
How many times might we relax this edge?

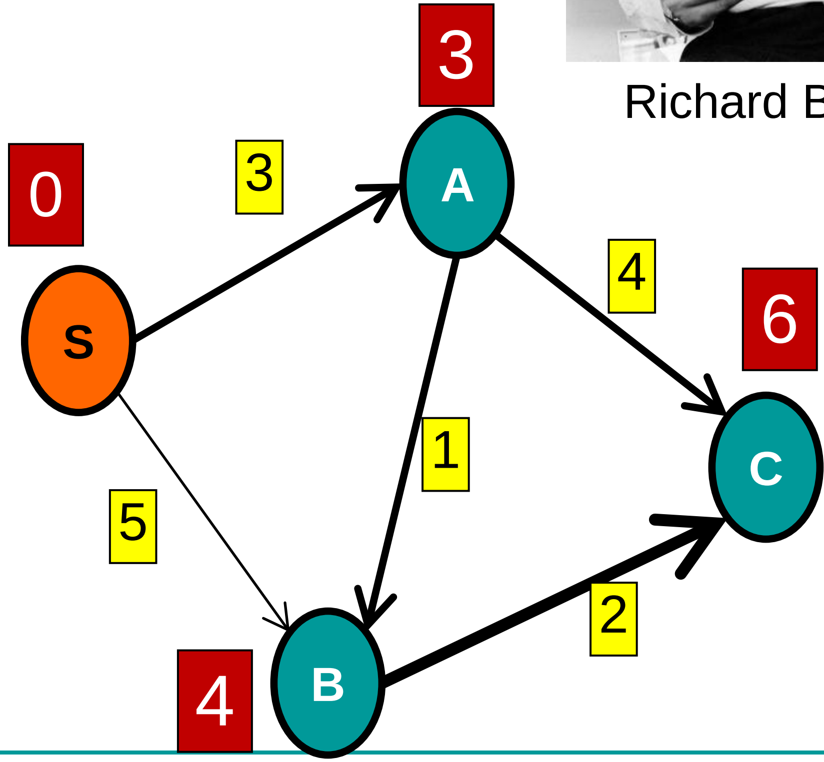
Bellman-Ford algorithm

```
n = V.length
for i = 1 to n-1
    for Edge e in Graph
        relax(e)
```



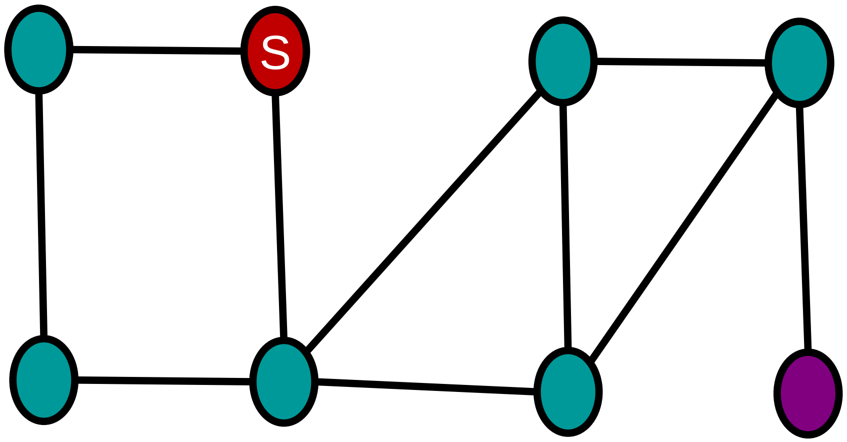
Richard Bellman

Does Bellman-Ford always work?

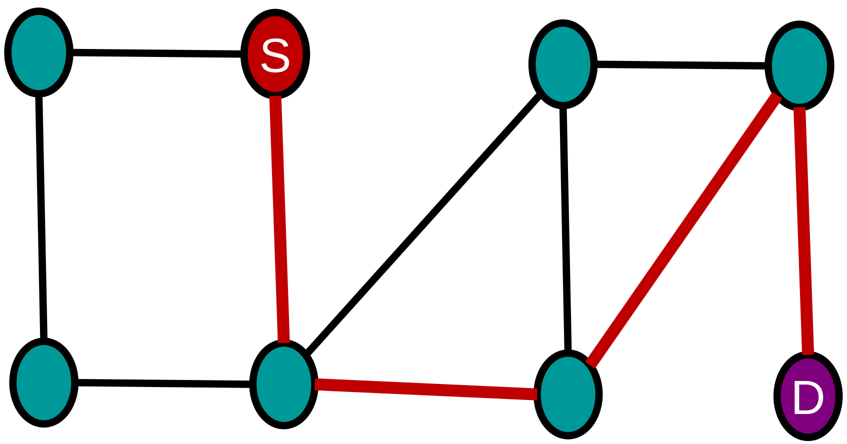


Yes! Proof by Induction (in Visualgo)

why does Bellman-Ford work?



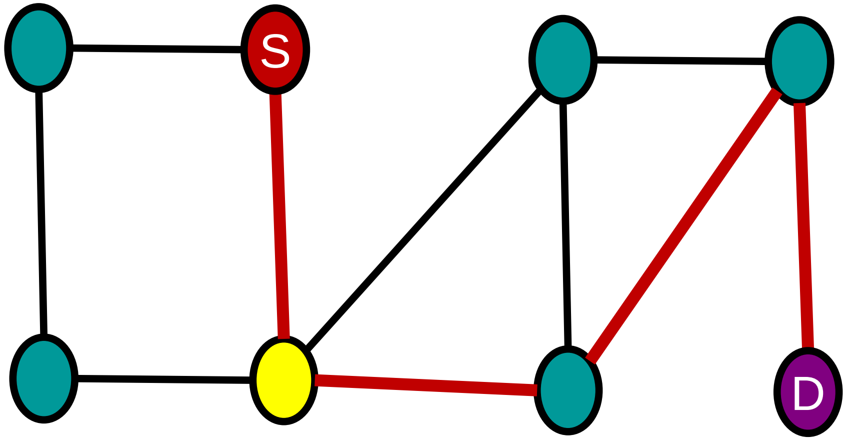
why does Bellman-Ford work?



Look at minimum weight path from S to D.

(Path is simple: no loops)

why does Bellman-Ford work?

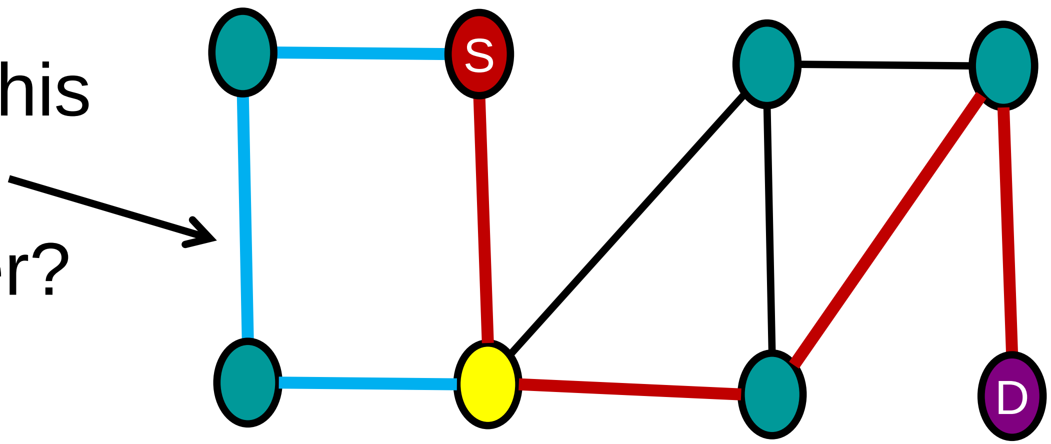


After 1 iteration, 1 hop estimate is correct.

meaning: All shortest paths that are 1 hop long are now correct

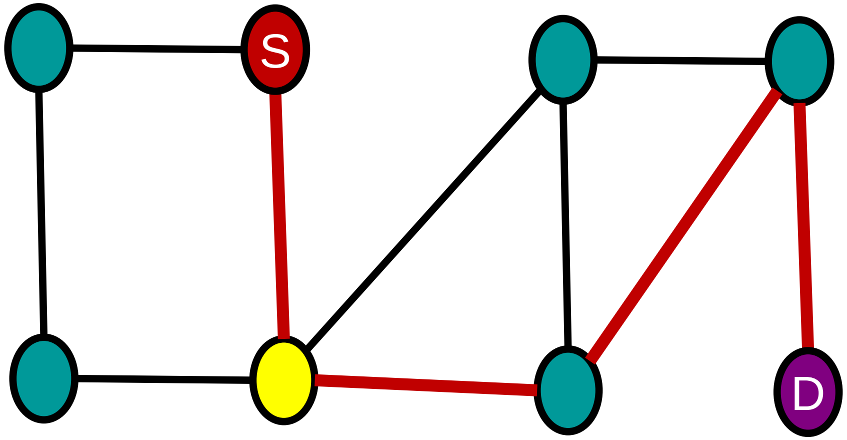
why does Bellman-Ford work?

What if this
path
is shorter?



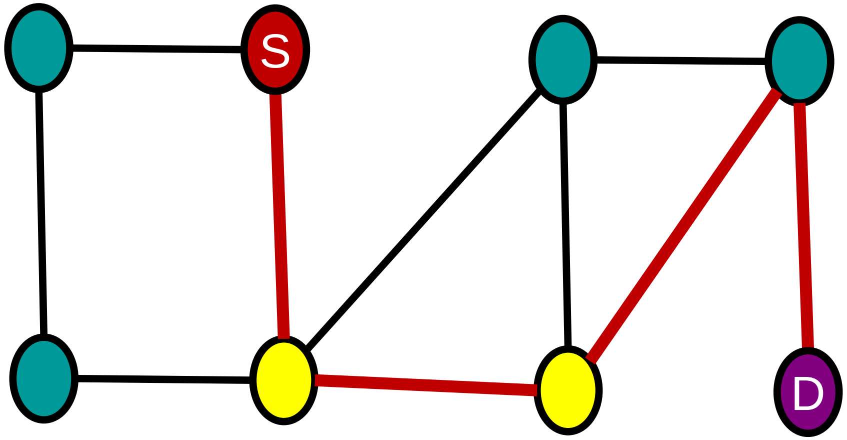
After 1 iteration, 1 hop estimate is correct.

why does Bellman-Ford work?



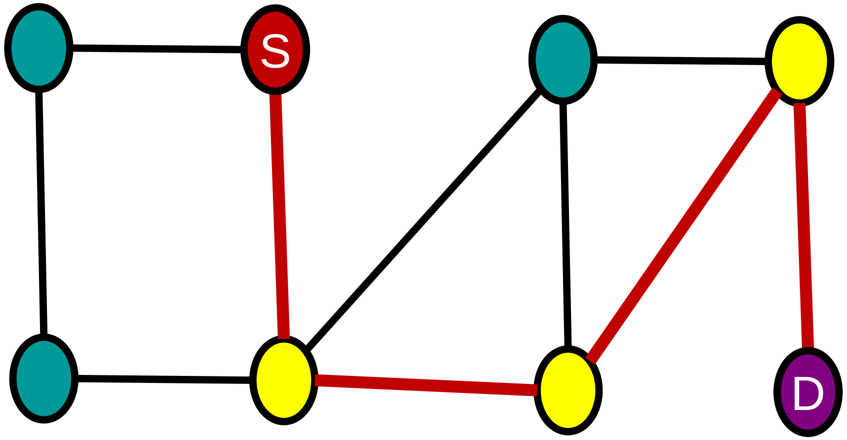
After 1 iteration, 1 hop estimate is correct.

why does Bellman-Ford work?



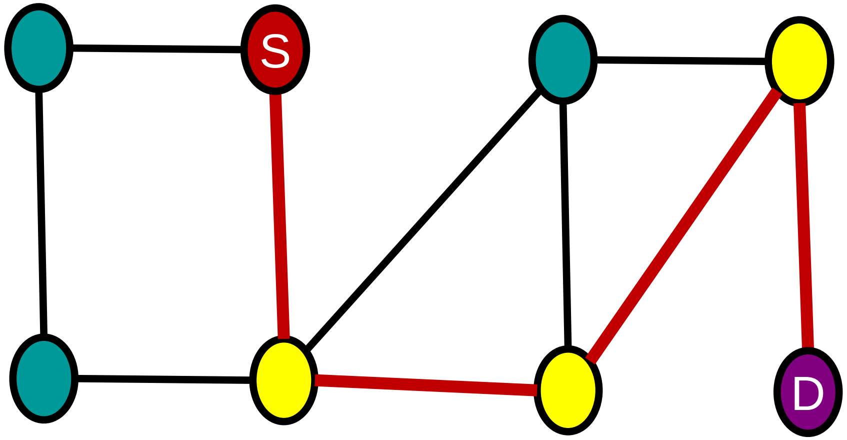
After 2 iterations, 2 hop estimate is correct.

why does Bellman-Ford work?



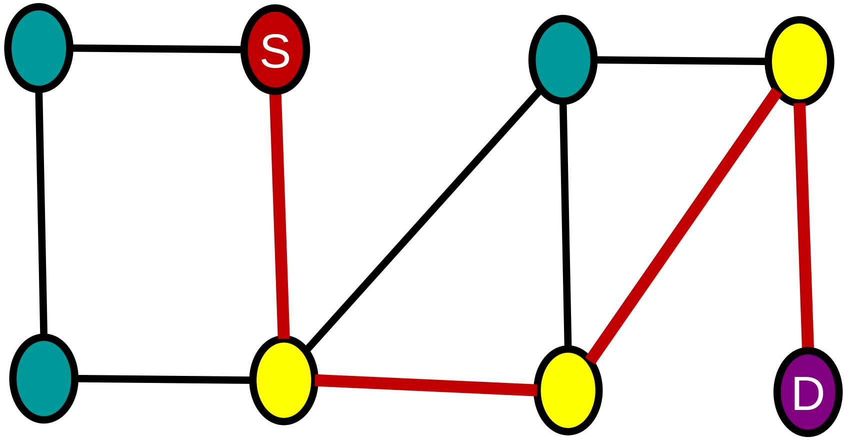
After 3 iterations, 3 hop estimate is correct.

why does Bellman-Ford work?



After 4 iterations, D estimate is correct.

Why does Bellman-Ford work?



Keep running till V-1 and Bellman-Ford finds shortest paths from s to all other nodes!

What is the running time of Bellman-Ford?

```
n = V.length
for i = 1 to n-1
    for Edge e in Graph
        relax(e)
```

What is the running time of Bellman-Ford?

A. $O(V)$
B. $O(E)$
C. $O(V+E)$
D. $O(VE)$
E. $O(E \log V)$
F. I have no idea.

Early termination?

```
n = V.length
for i = 1 to n-1
    for Edge e in Graph
        relax(e)
```

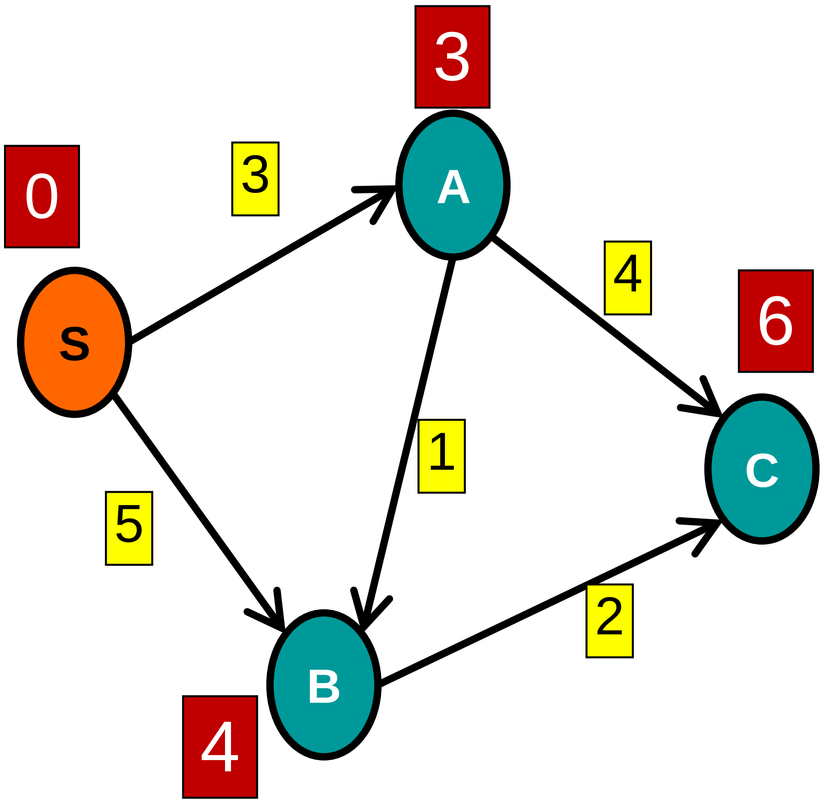
When can we terminate early?

A. When a relax operation has no effect.
B. When two consecutive relax operations have no effect.
C. When an entire sequence of |E| relax operations have no effect.
D. Never. Only after |V| complete iterations.

Shortest paths

Maintain estimate for each distance:

```
for Edge e in graph
      relax(e)
```



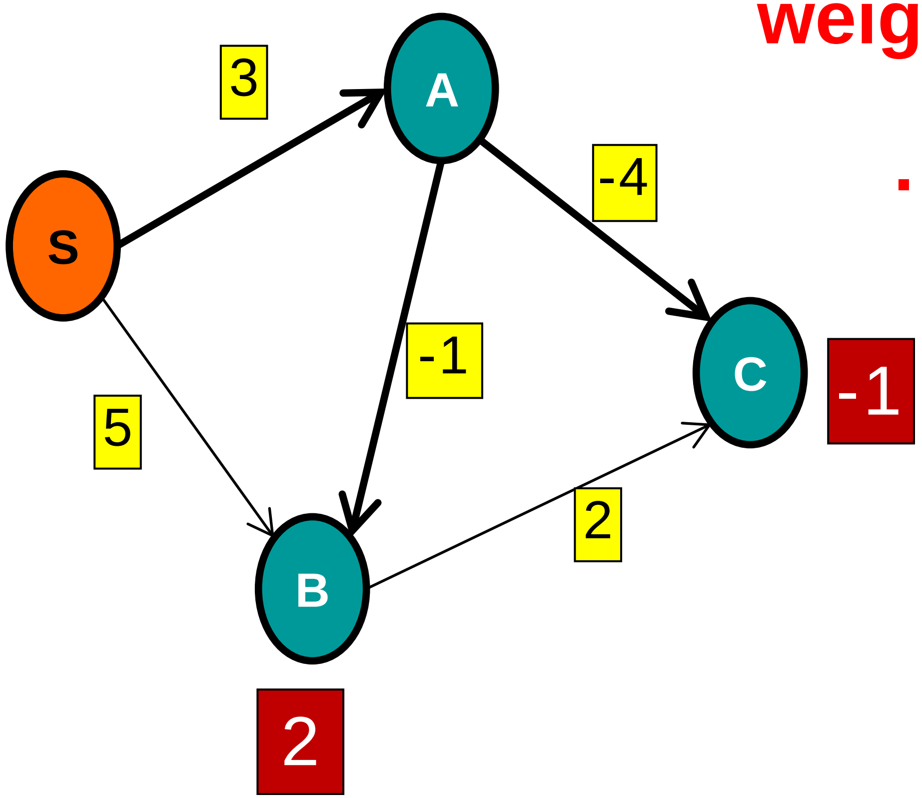
If we relax all the edges and there is no faster way to get to any node, we have the shortest paths!

Negative edge weights?

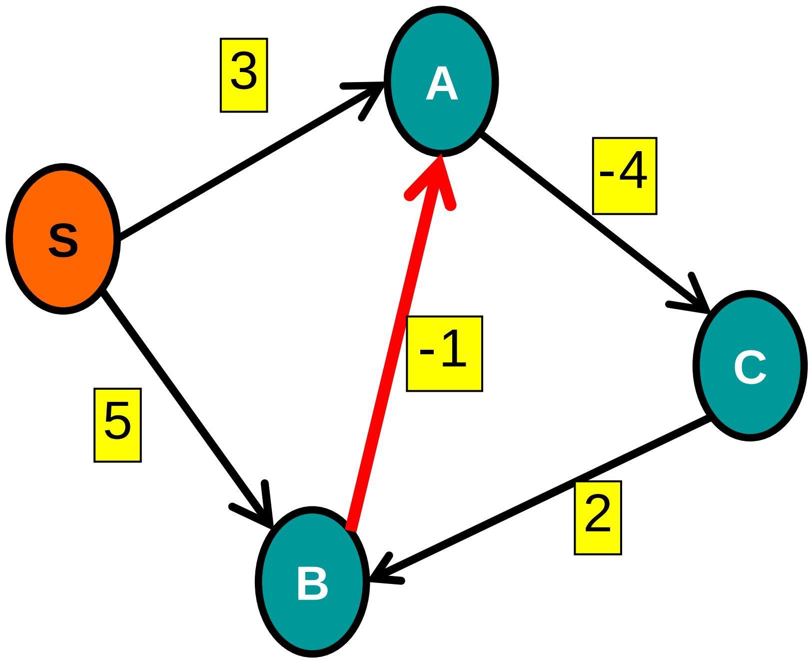
Will Bellman-Ford algorithm still work?

Bellman-Ford has no problems with negative edge weights!

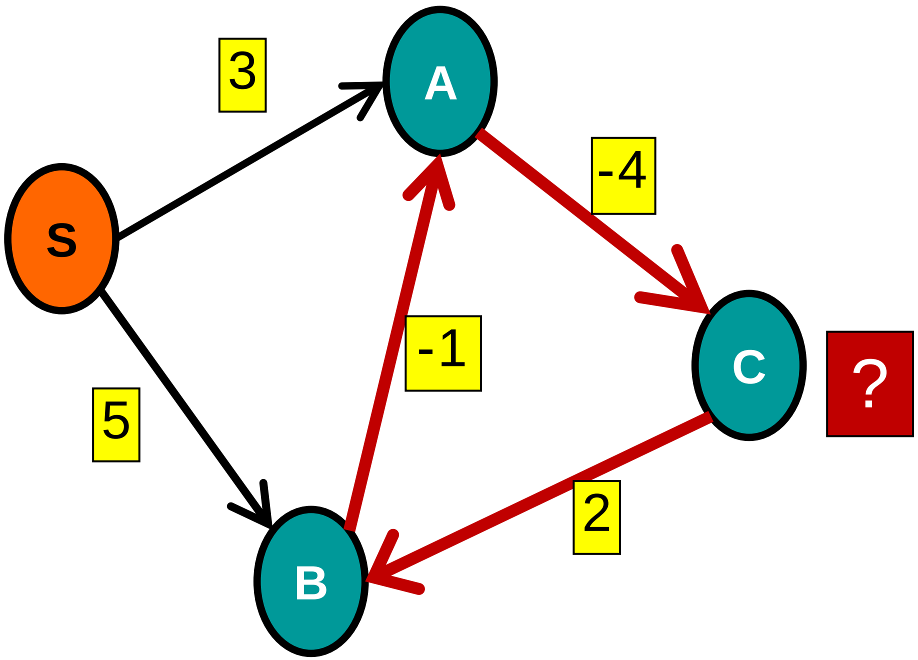
….almost…



What if the graph looks like this?



negative weight **cycle**

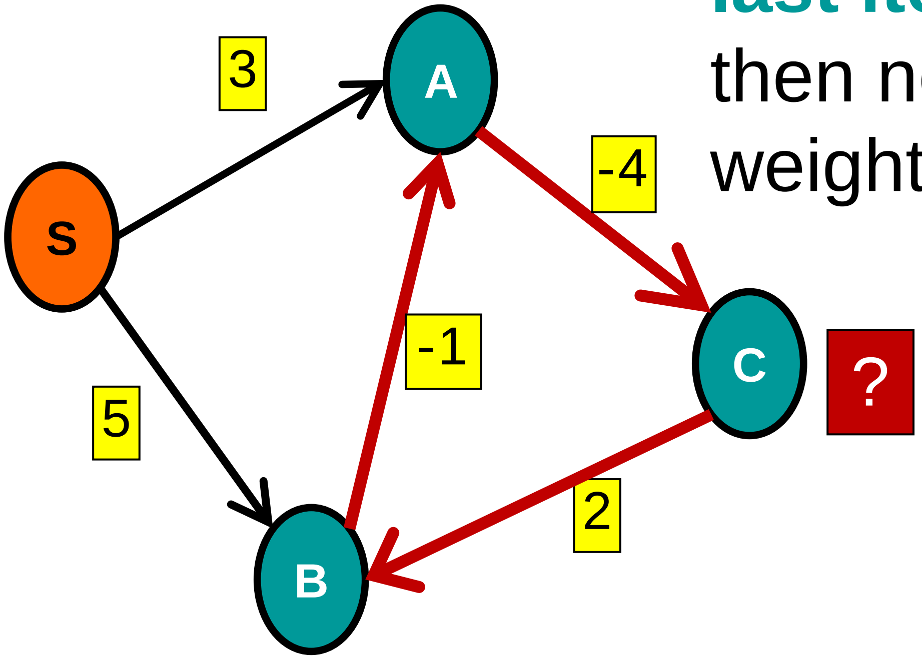


d(S,C) is infinitely negative!

negative weight **cycle**

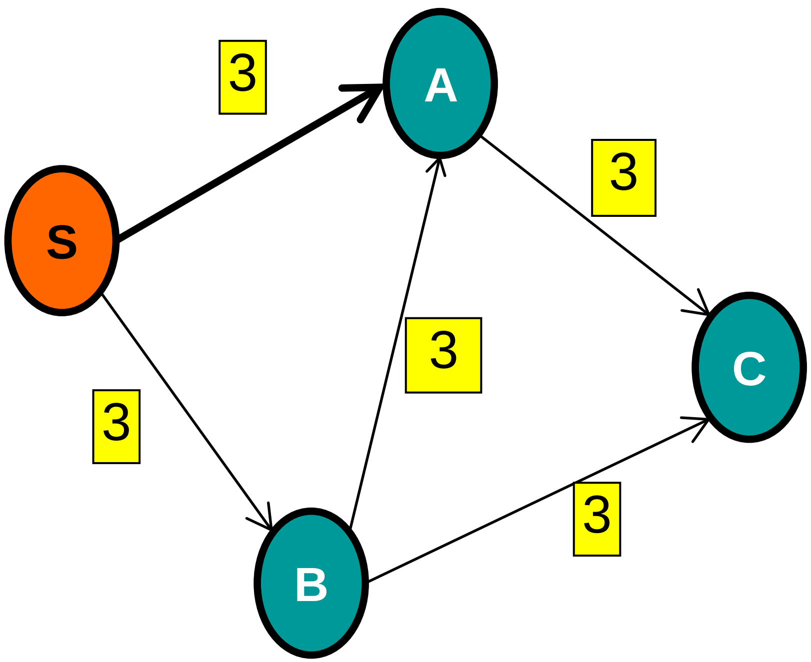
Run Bellman-Ford for |V| iterations.

If an **estimate changes in the last iteration** then negative weight cycle.



How to **detect** negative weight cycles?

Special case:



all edges have the **same weight**: **What can we use?**

BFS!!!

BFS for SSSP

We need to perform some simple modifications to BFS for it to be able to solve the unweighted version (or equal weights) of the SSSP problem:

1. we change the Boolean array **visited** into an Integer array **D**.
2. At the start of BFS, instead of setting visited[u] = false, we set D[u] = 1e9 (a large number to symbolise +∞ or even -1 to symbolise 'unvisited' state, but we cannot use 0 as D[0] = 0) $\forall$**u** $\in$ **V\{s}**; Then we set D[s] = 0
3. We change the BFS main loop from
   if (visited[v] == 0) { visited[v] = 1 … } // v is unvisited
   to
   if (D[v] == 1e9) { D[v] = D[u]+1 … } // v is 1 step (or whatever
   // weight) away from u

Other special cases

Condition	Algorithm	Time Complexity
No Negative Weight Cycles	Bellman-Ford Algorithm	$O(VE)$
On Unweighted Graph (or equal weights)	BFS	$O(V+E)$
No Negative Weights	Dijkstra's Algorithm	$O((V+E)\log V)$
On Tree	BFS / DFS	$O(V)$
On DAG	Dynamic Programming	$O(V+E)$

Summary

- Single source shortest path (very common Computer Science problem)
- SSSP: Given source vertex s, want to find shortest path weights path to every other vertex v
- Bellman-Ford can be used in graphs that contain negative weights, but not negative weight cycles
- BFS can be used for unweighted or paths with same weights

*Acknowledgement: some slides courtesy of Dr Harold Soh

Programming Exam

- Date: 3 Nov 2018 (week 11)
- Time: 1pm to 3pm
- Venues: PL1 and PL2
- Lab allocation has been uploaded in IVLE
- Format: 2 problems each with subtasks
- Total 12%
- Open Book (allowed to bring hard copy material)
- No internet access available during PE (i.e. cannot use online IDE or sunfire)