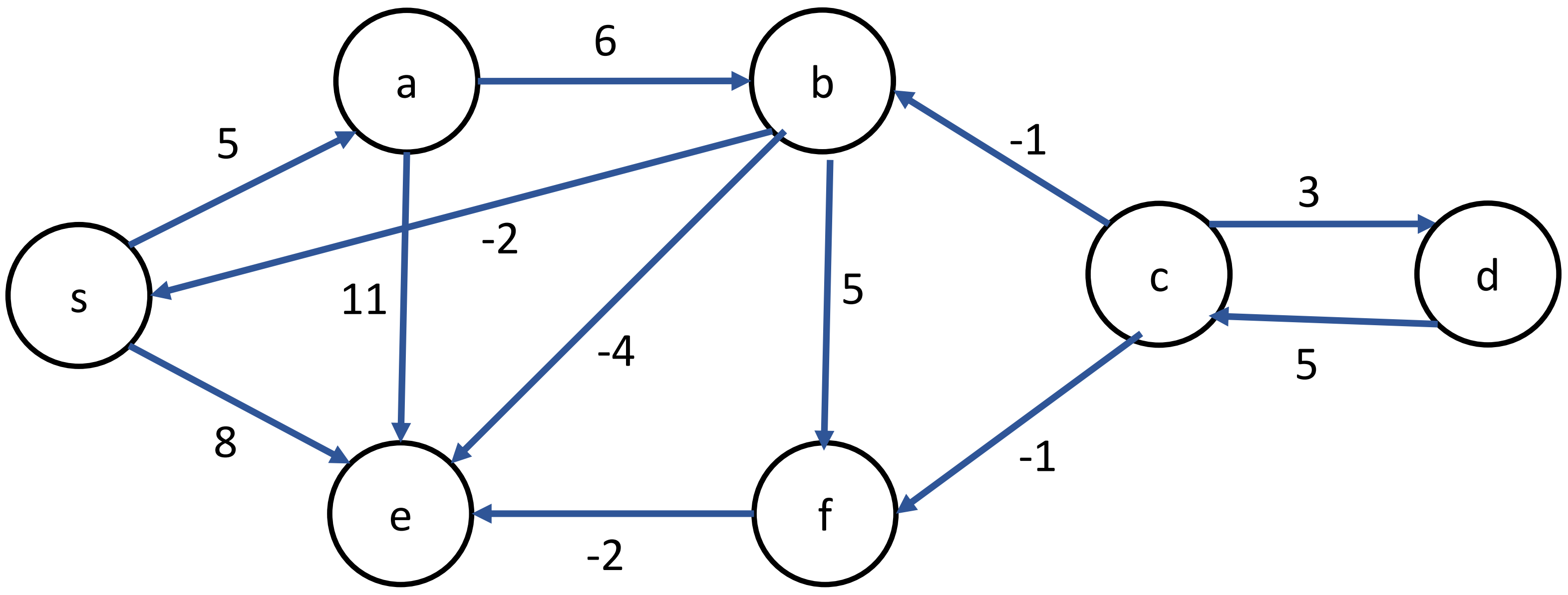
Supplemental Slides of Ch24

Sungjin Im

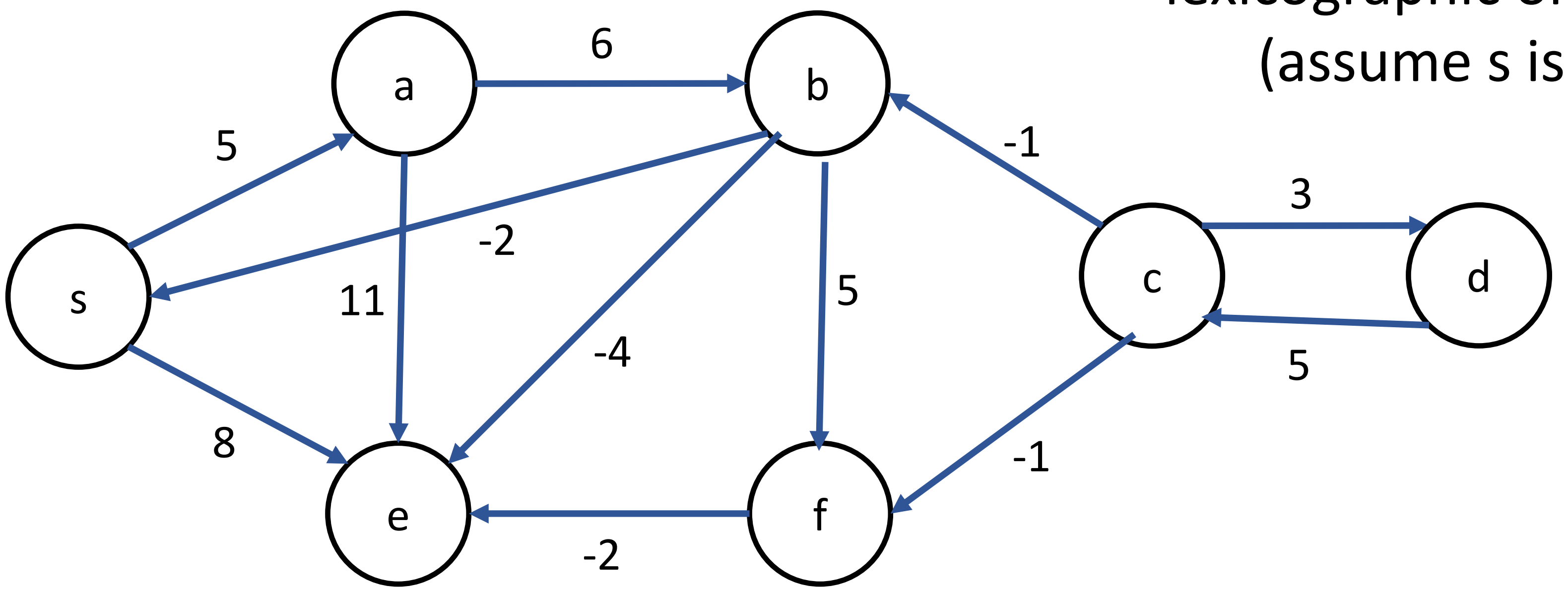
4/20/2023

Bellman-Ford Illustration

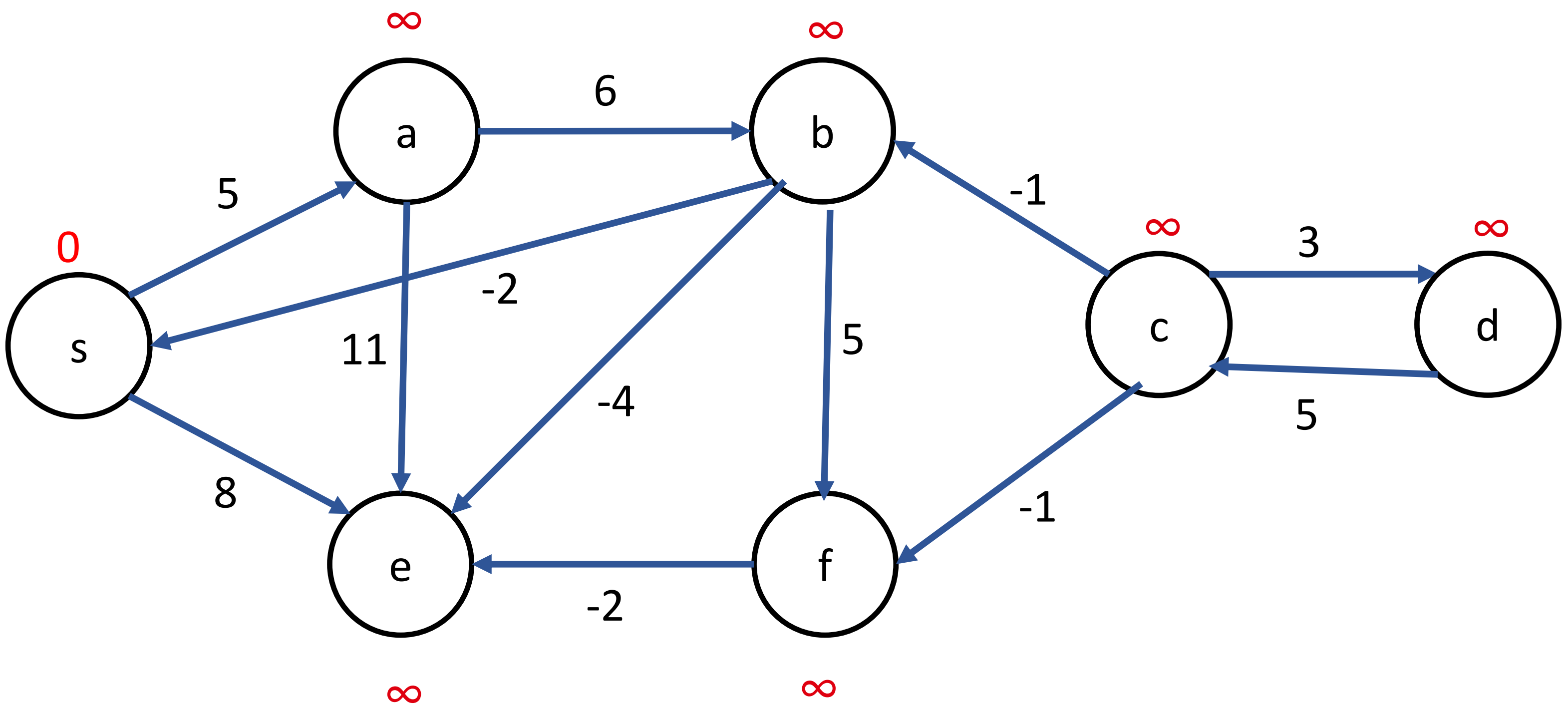


Bellman-Ford Illustration

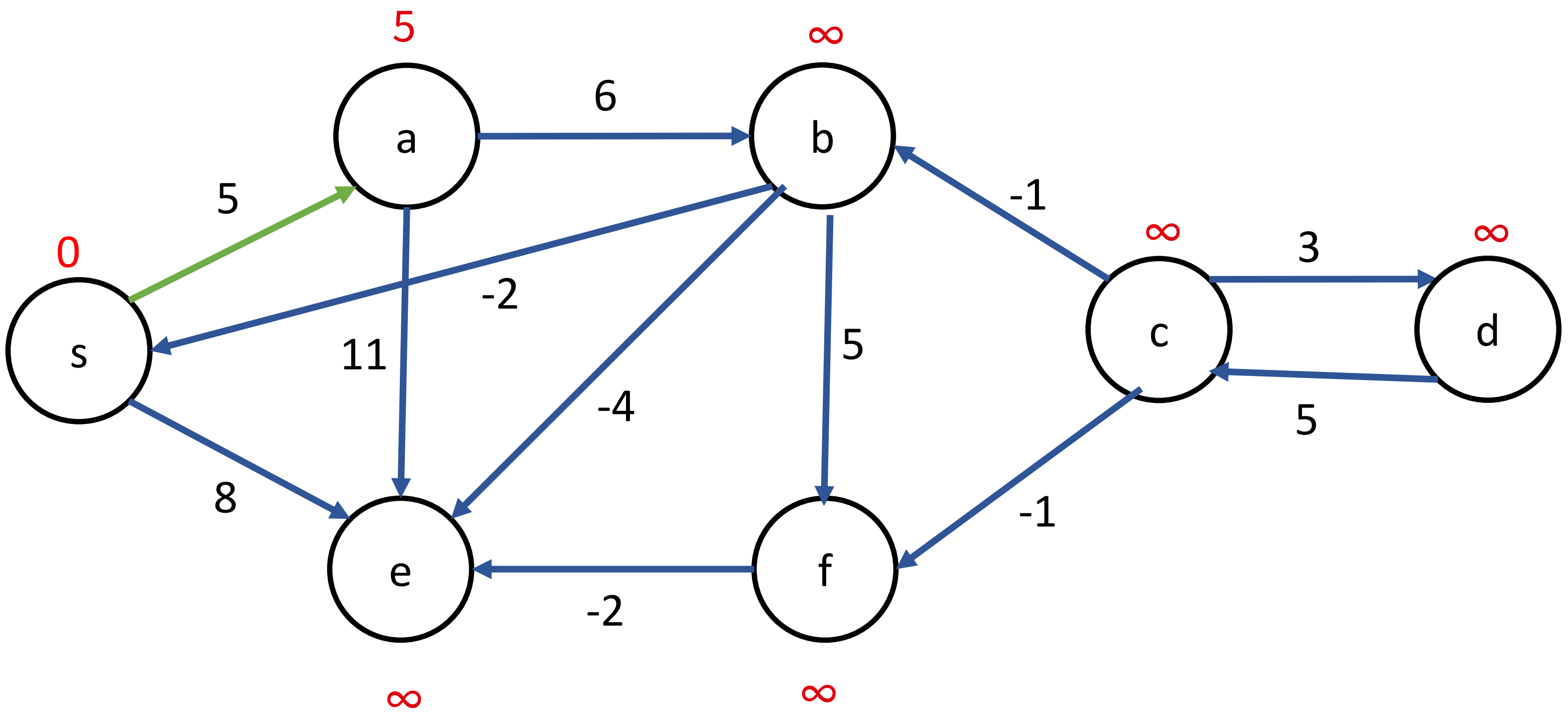
Let's assume we relax edges in lexicographic order (no need to do so)
(assume s is ahead of all chars)



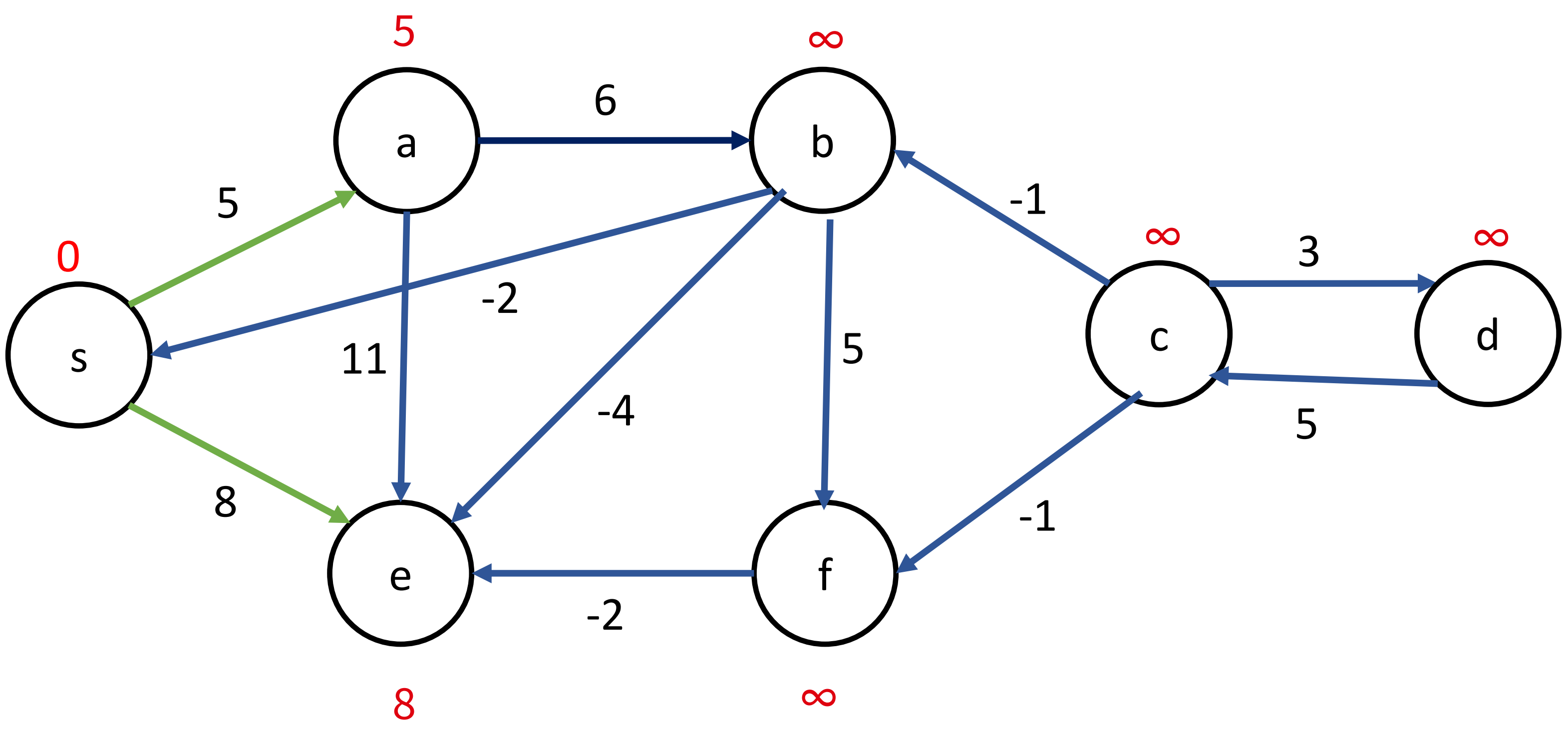
Bellman-Ford Illustration



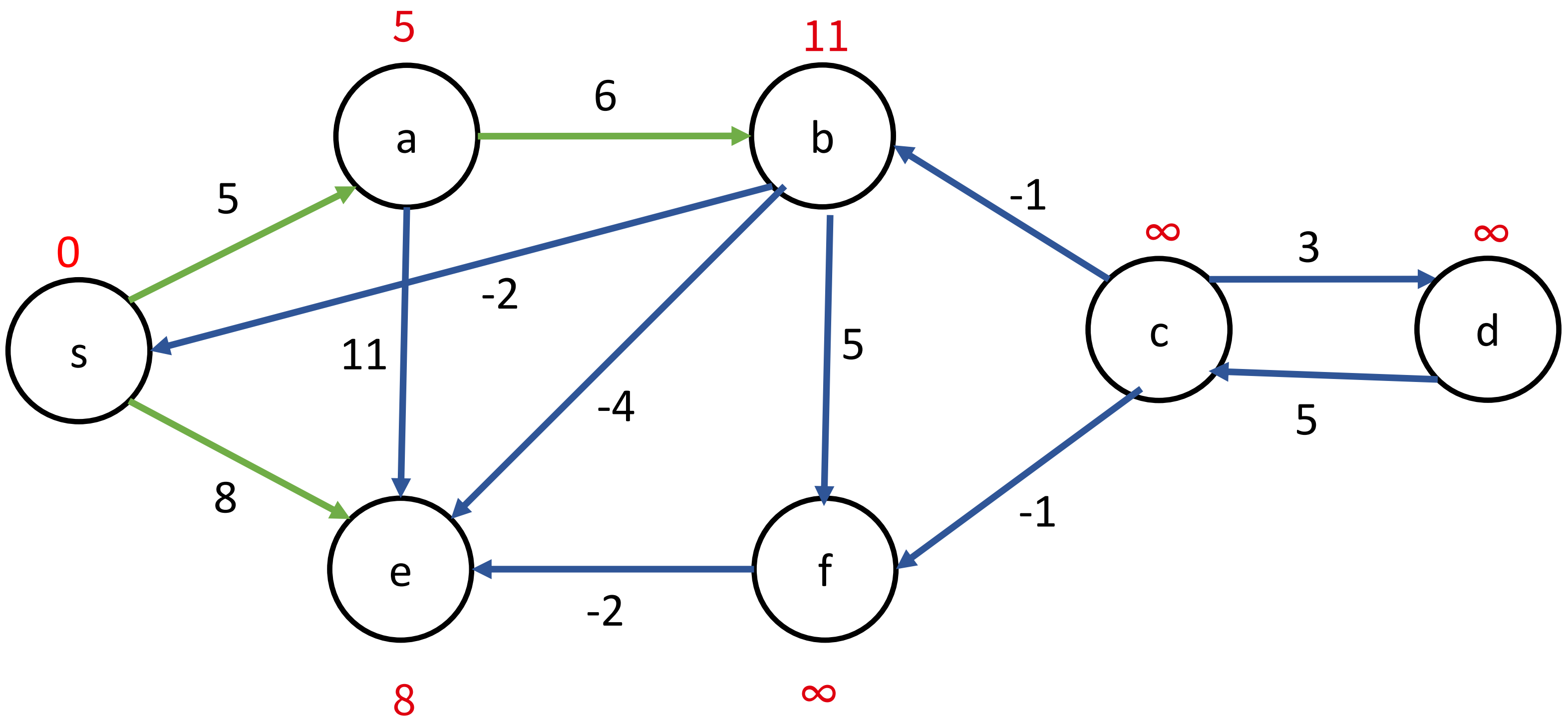
Bellman-Ford Illustration



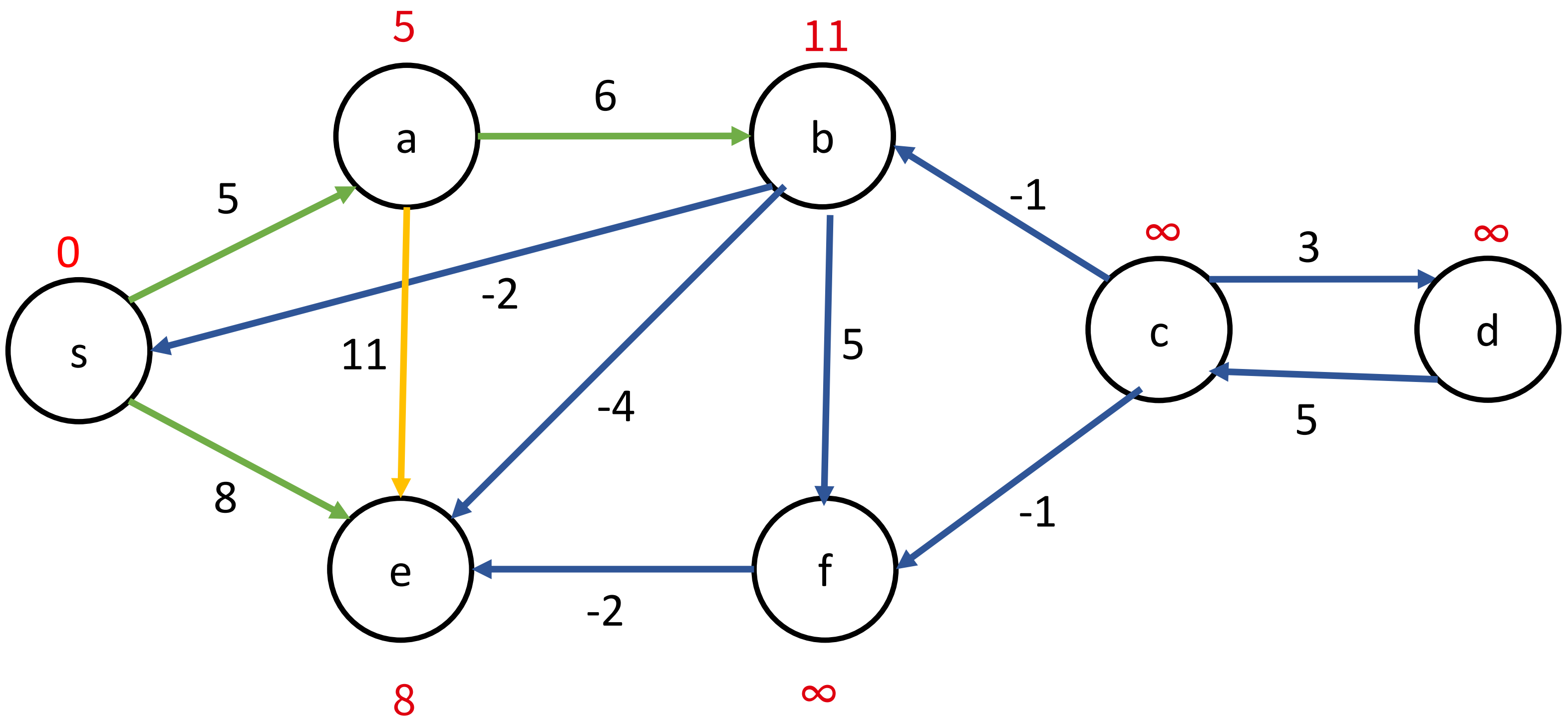
Bellman-Ford Illustration



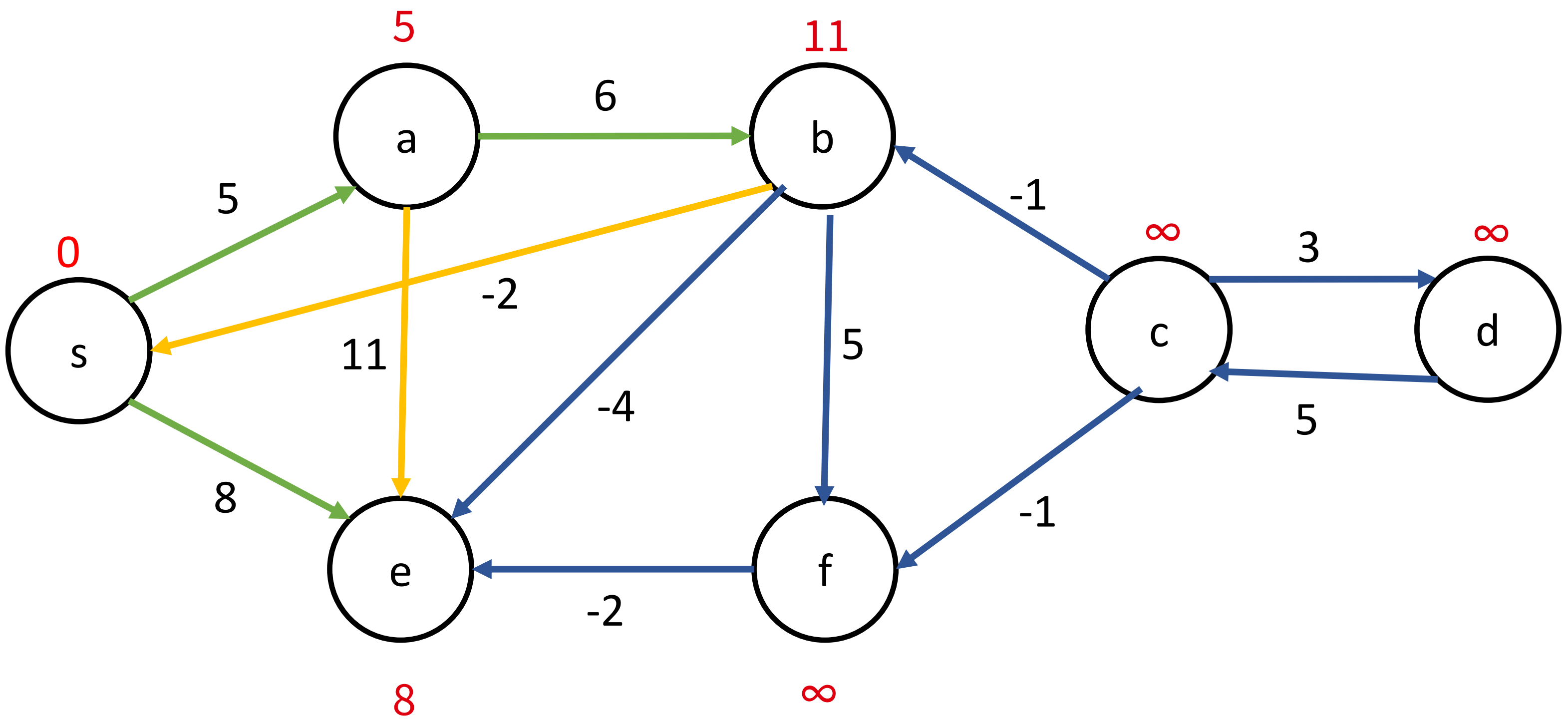
Bellman-Ford Illustration



Bellman-Ford Illustration

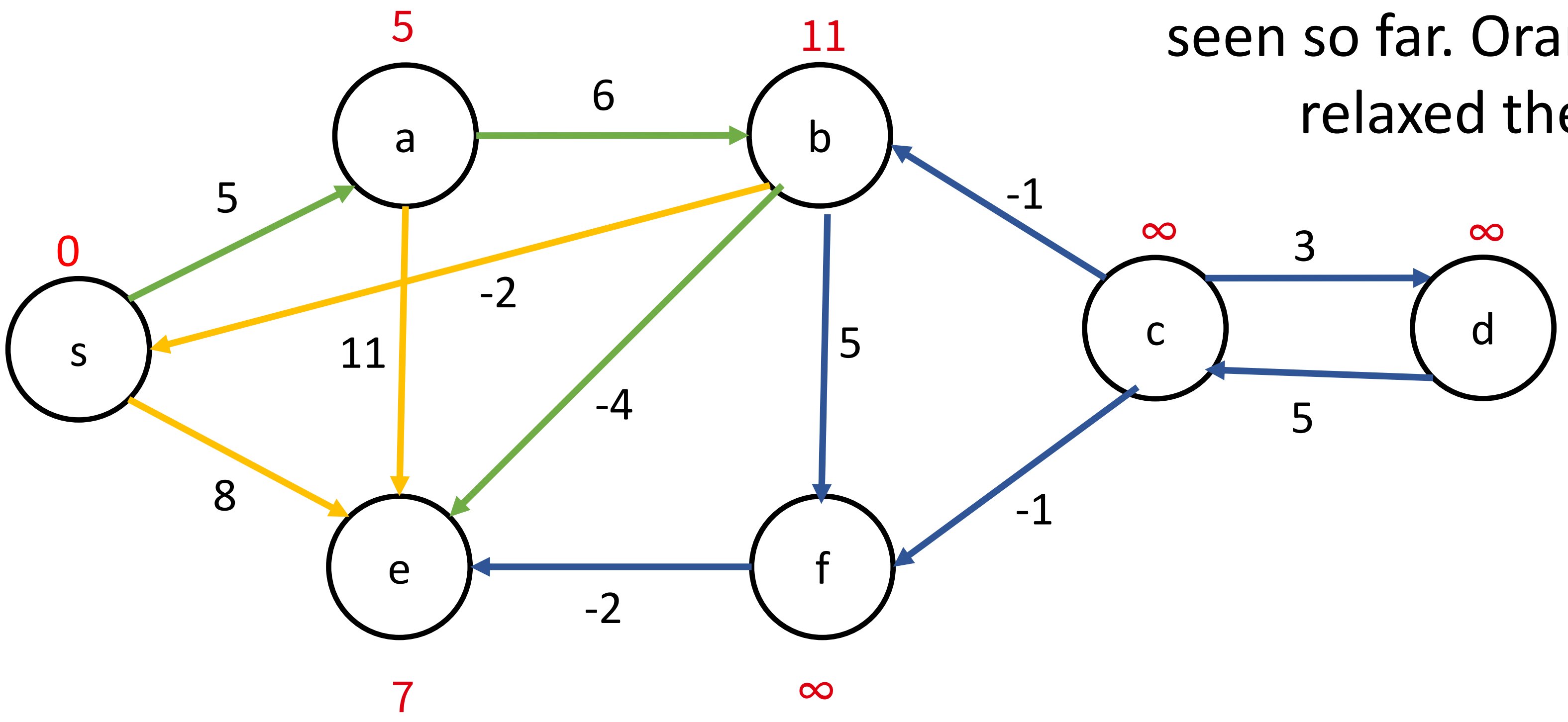


Bellman-Ford Illustration



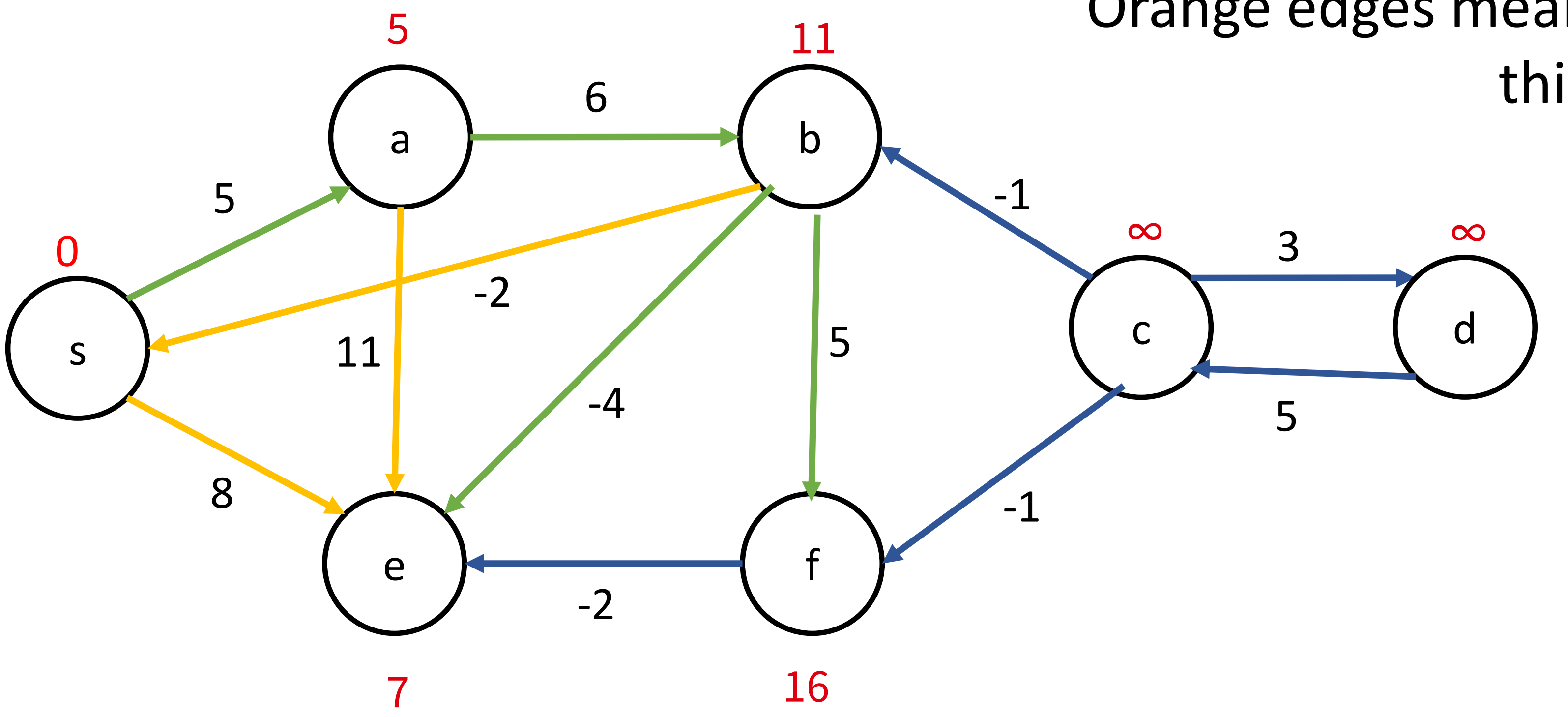
Bellman-Ford Illustration

Green edges encode shortest paths we've seen so far. Orange edges mean we've relaxed them (in this round)

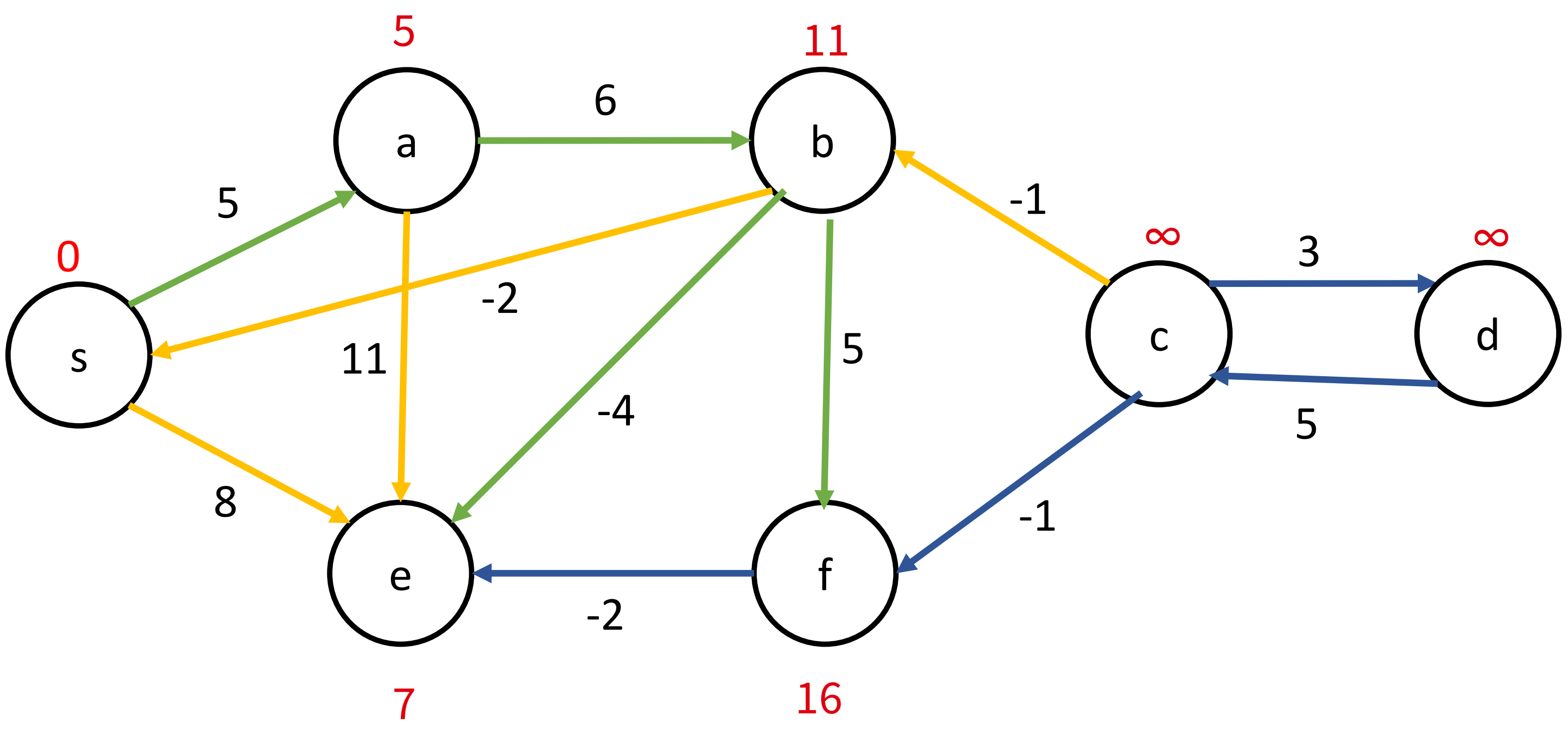


Bellman-Ford Illustration

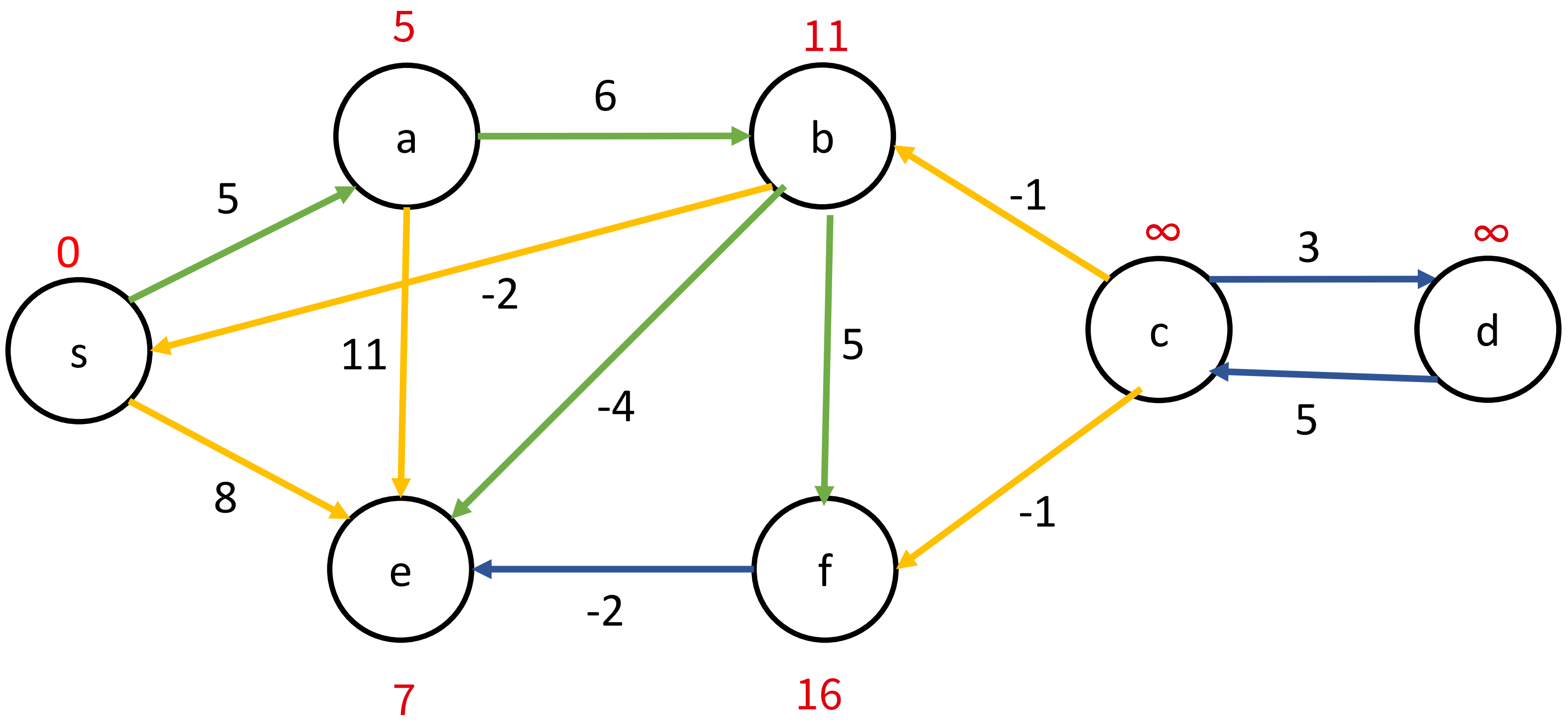
Green edges encode shortest paths we've seen so far; in implementation we use v.π
Orange edges mean we've relaxed them (in this round)



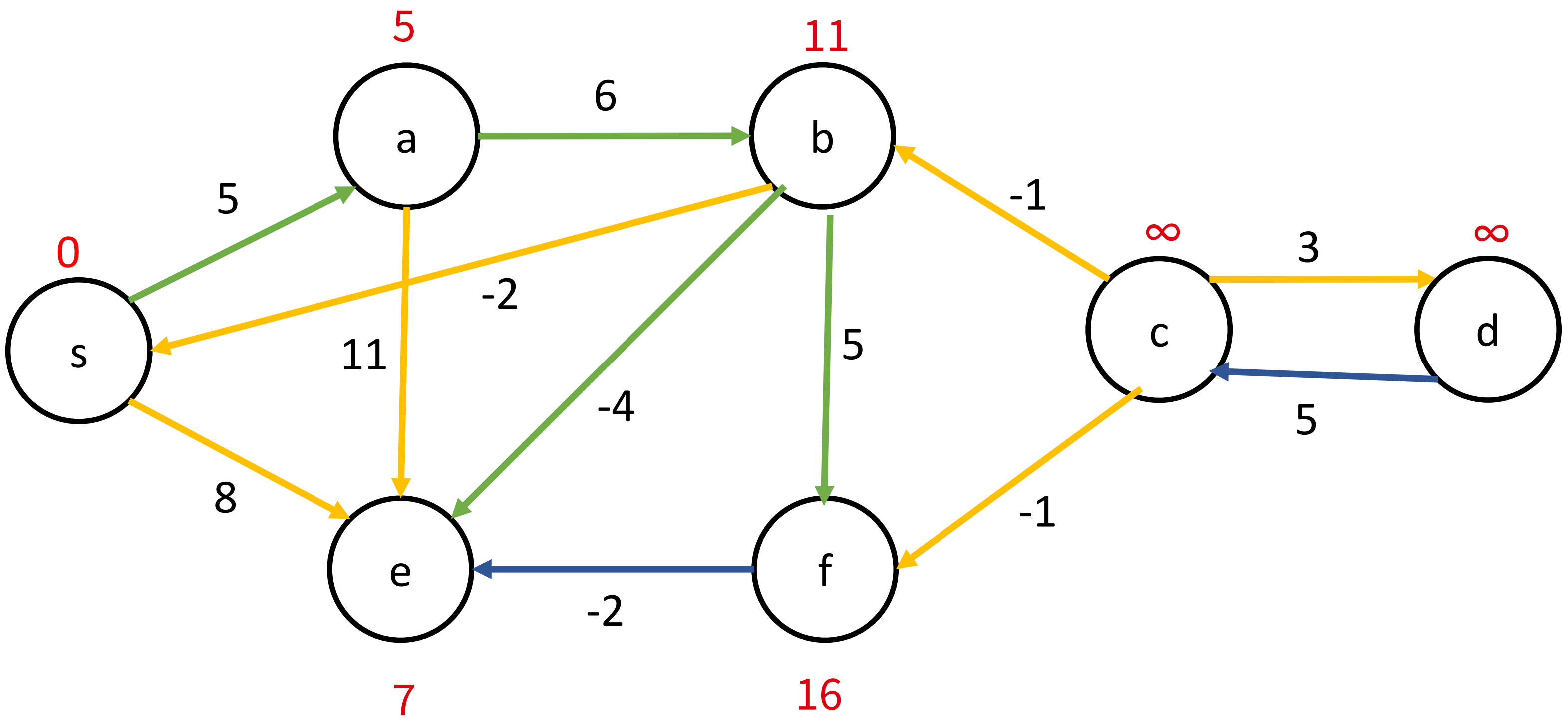
Bellman-Ford Illustration



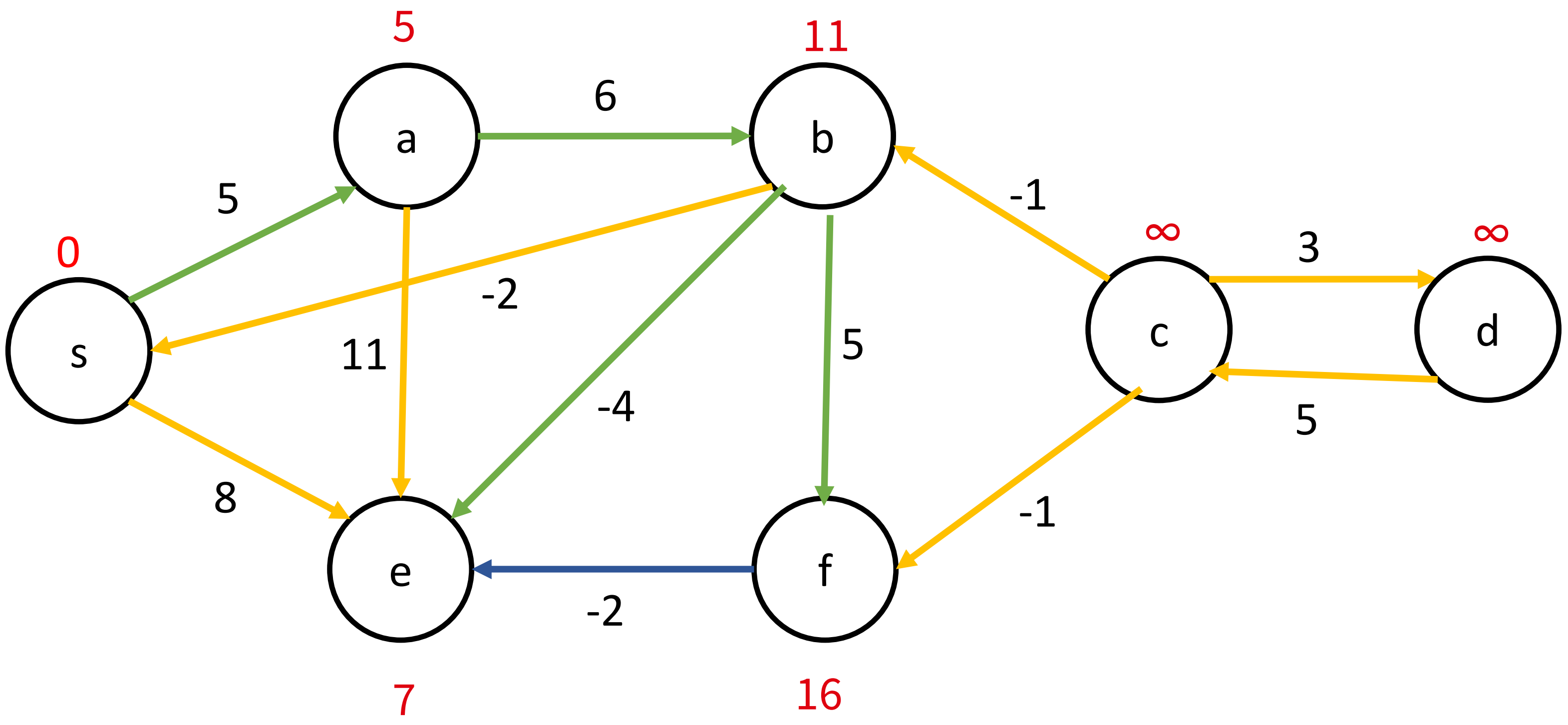
Bellman-Ford Illustration



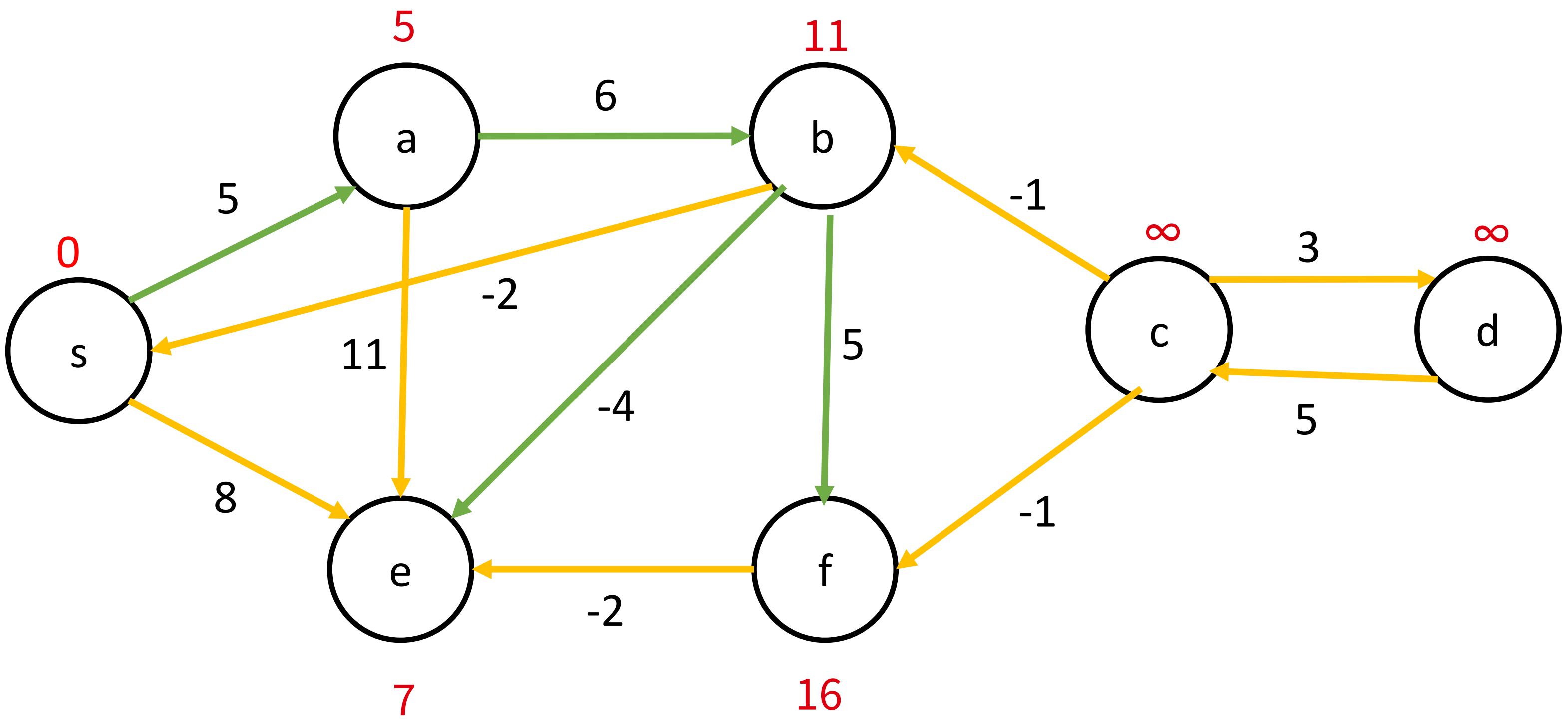
Bellman-Ford Illustration



Bellman-Ford Illustration

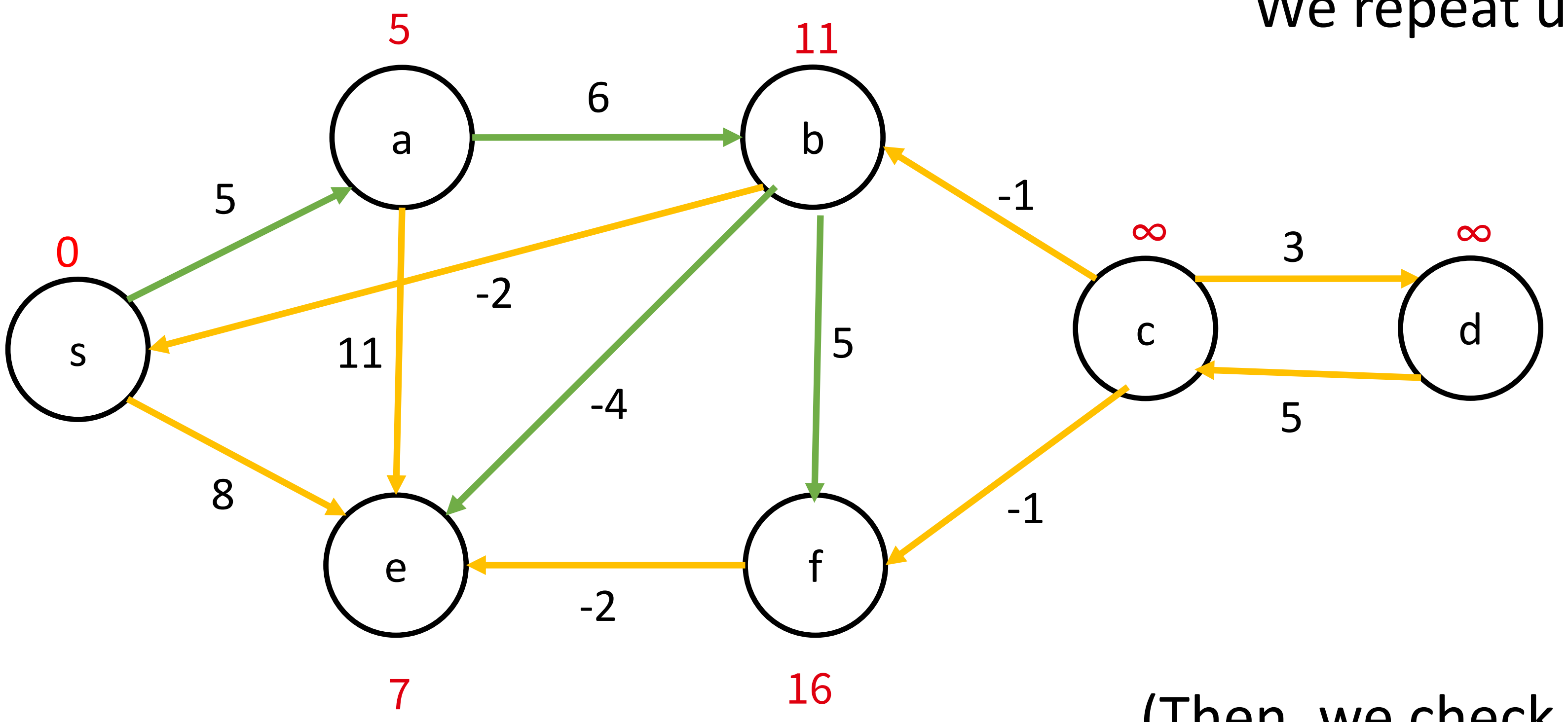


Bellman-Ford Illustration



Bellman-Ford Illustration

We relaxed each edge exactly once;
This is the end of Round 1
We repeat up to Round |V| - 1

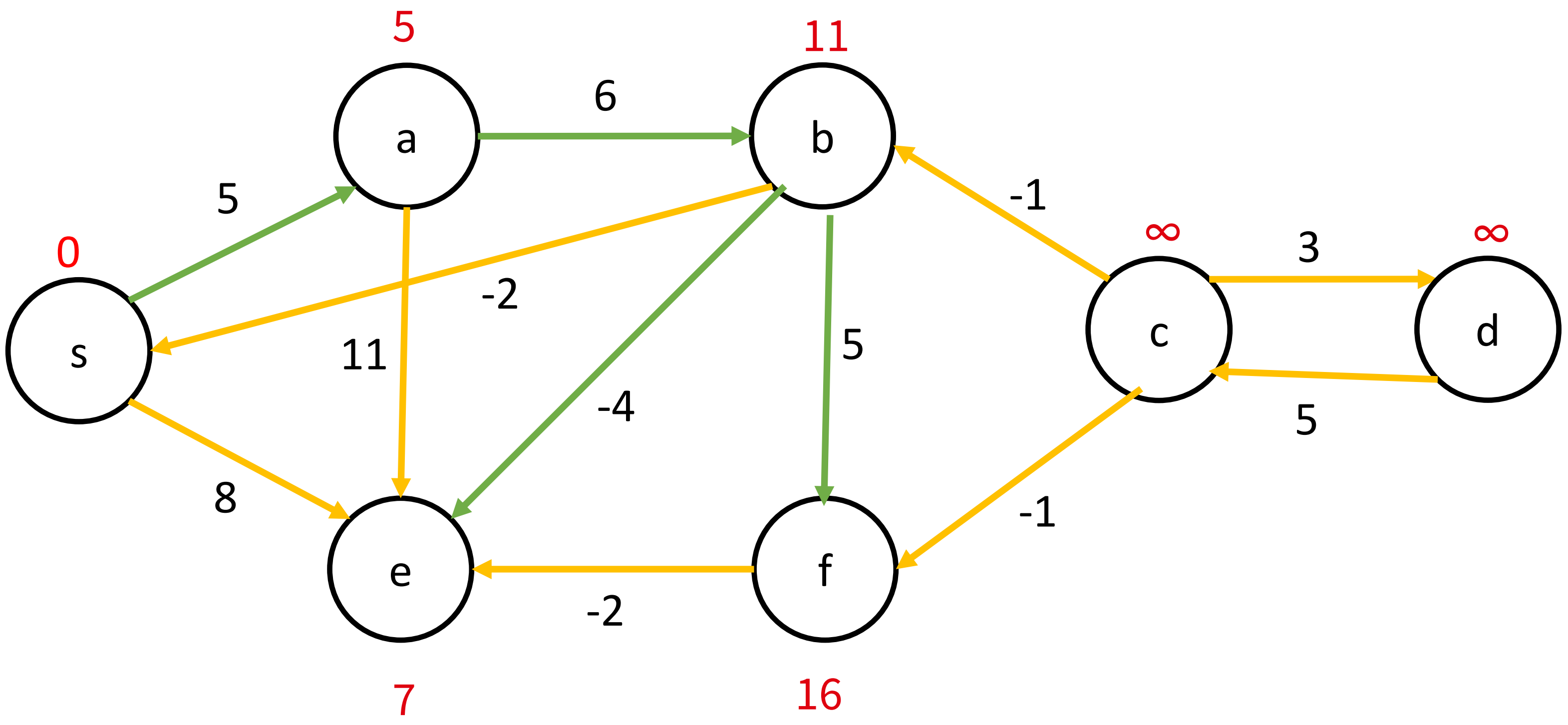


(Then, we check relaxing an edge gives an improvement. If so, we say the graph has a negative weight cycle reachable from s)

Bellman-Ford Illustration

Try Round 2

Do you see any changes?

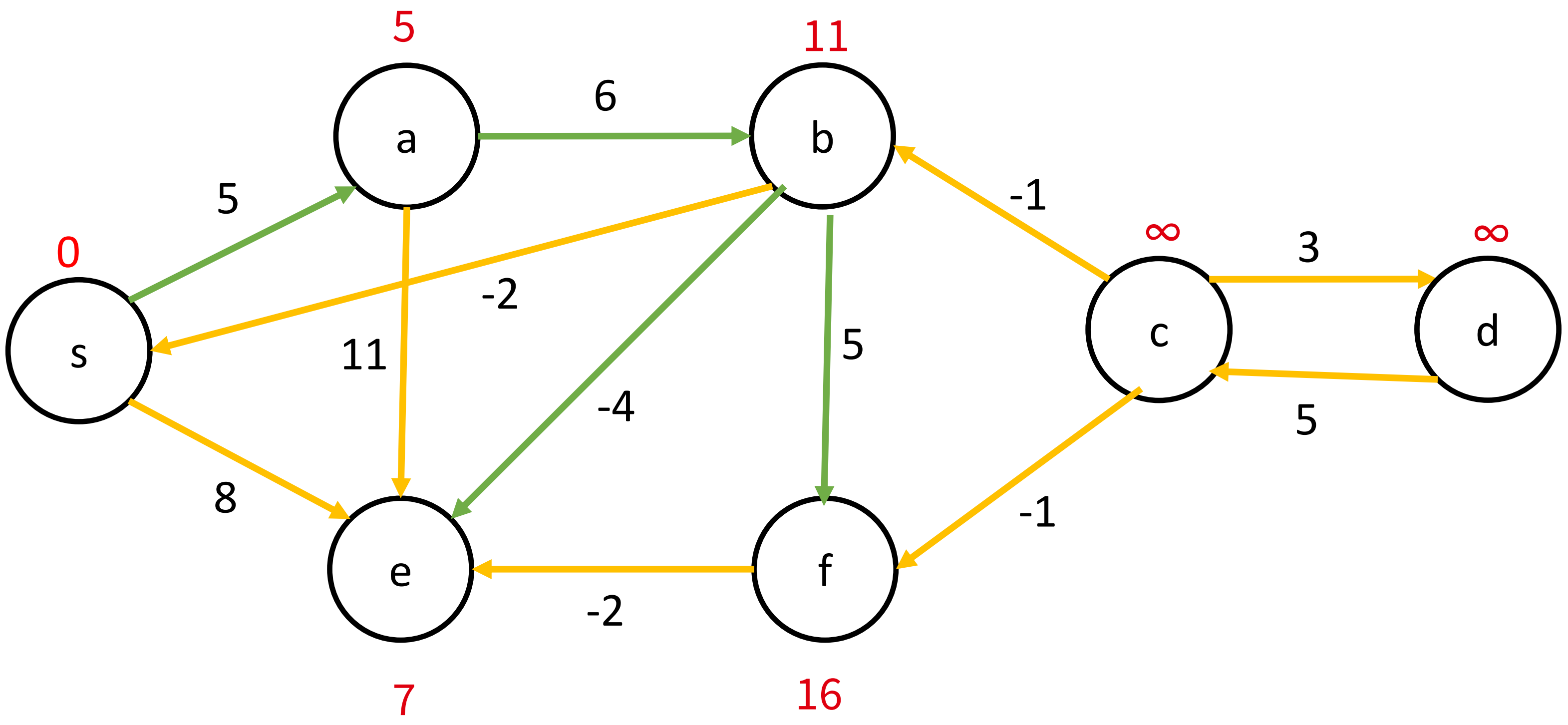


Bellman-Ford Illustration

Try Round 2

Do you see any changes?

No.

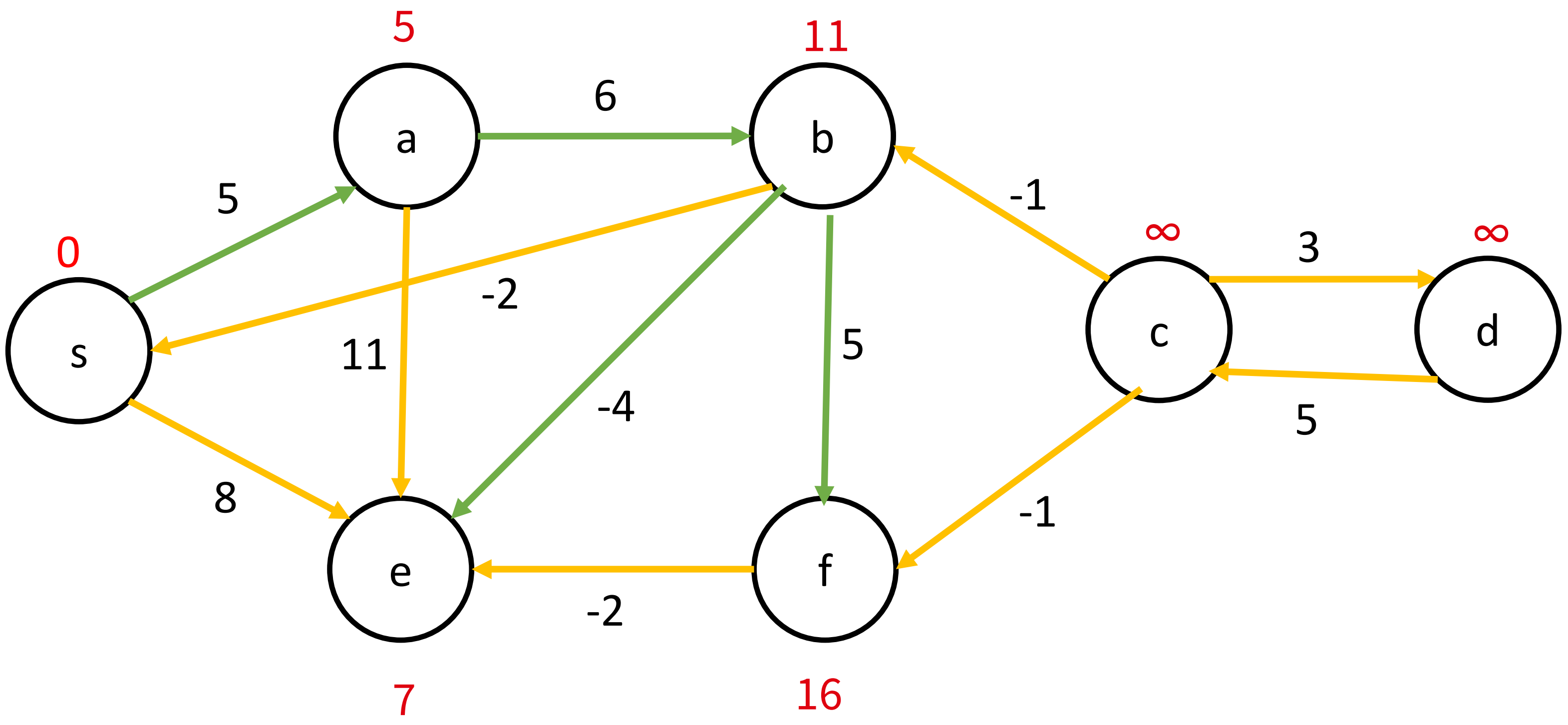


Bellman-Ford Illustration

Try Round 3

Will you see any changes?

No.

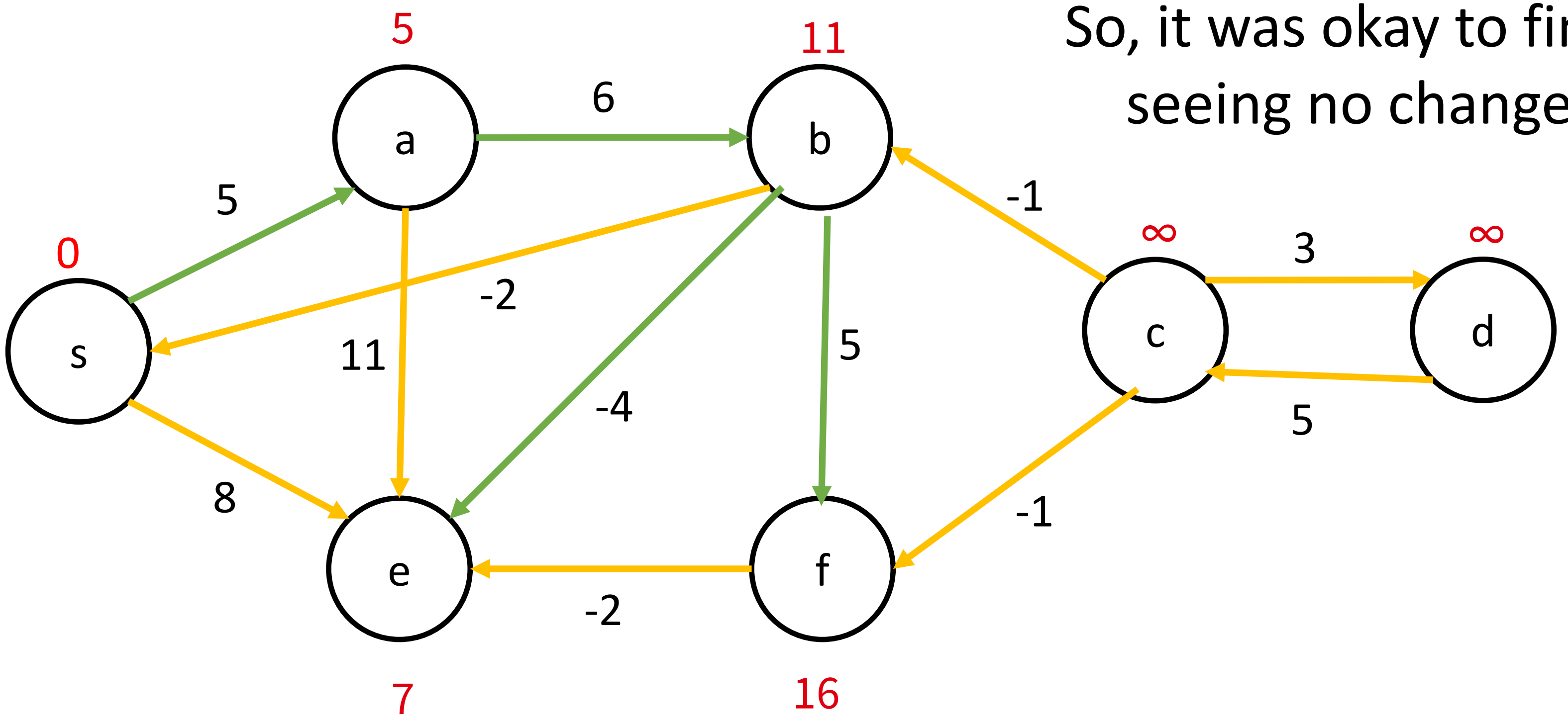


Bellman-Ford Illustration

We will have no changes in Round 2, 3, …, V-1

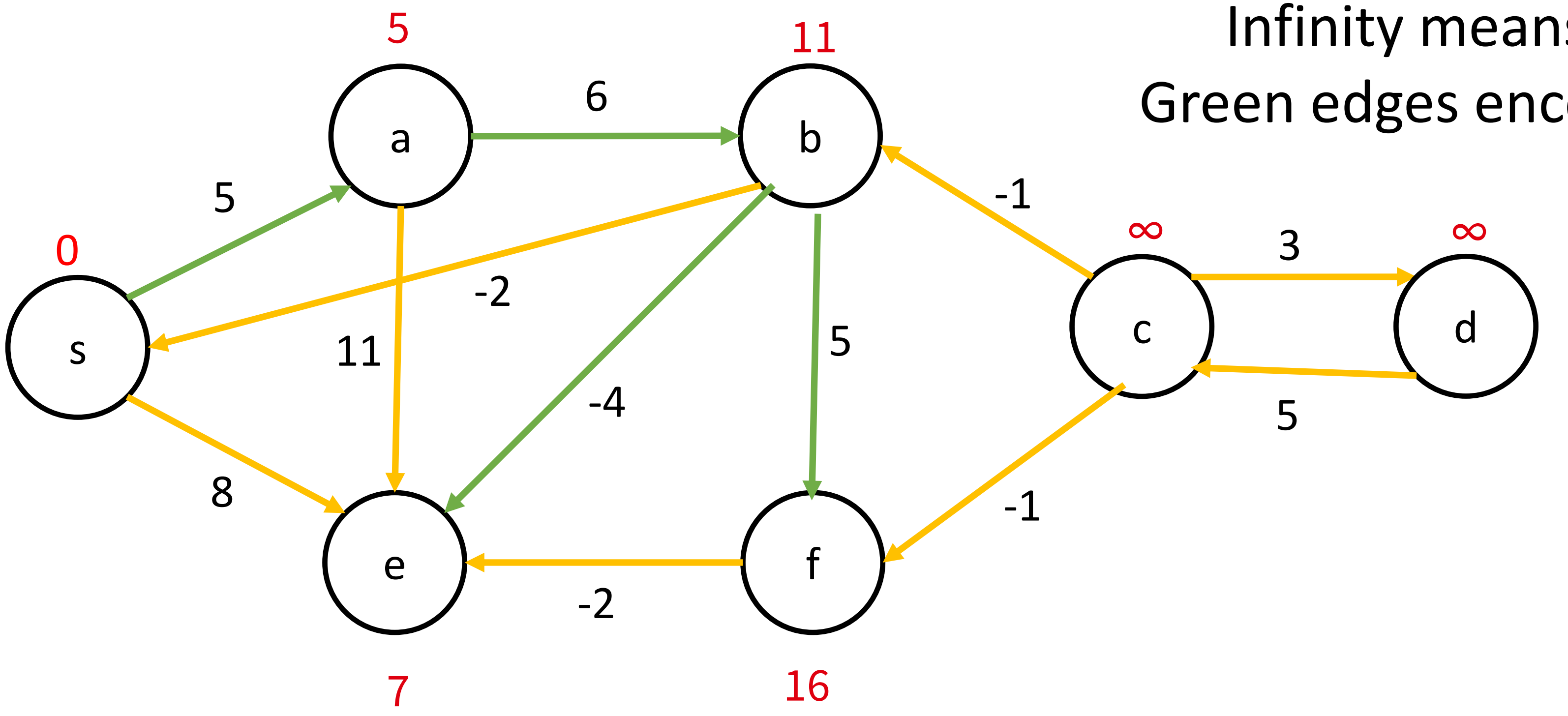
So, it was okay to finish right after round 2

seeing no changes made in the round



Bellman-Ford Illustration

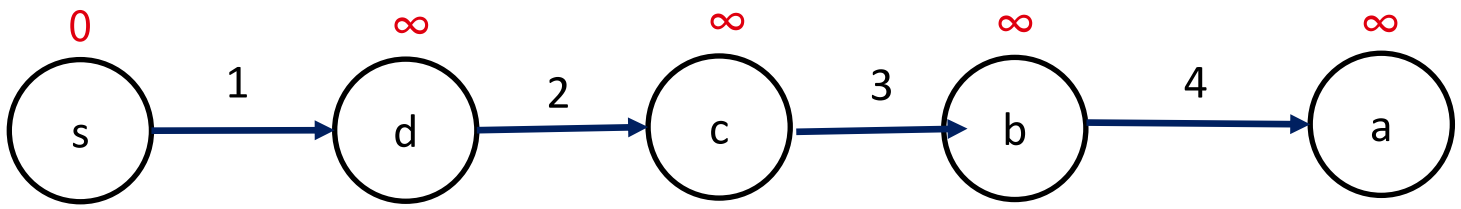
Red number implies the distance from s
Infinity means not reachable
Green edges encode shortest paths



Bellman-Ford Illustration

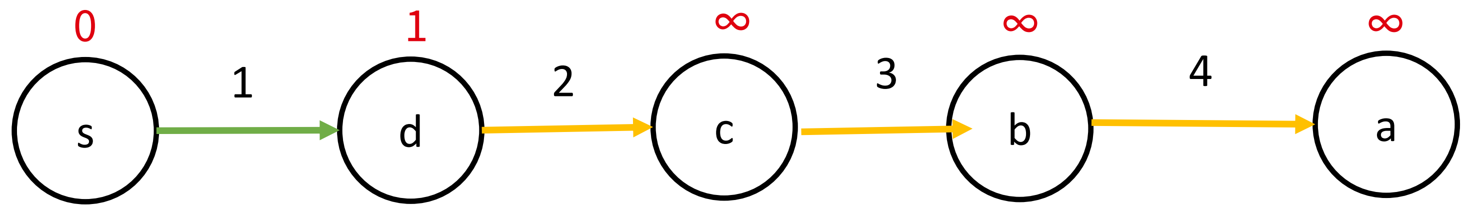
Do we really need V-1 rounds?

Say we relax in lexicographic order (including s)



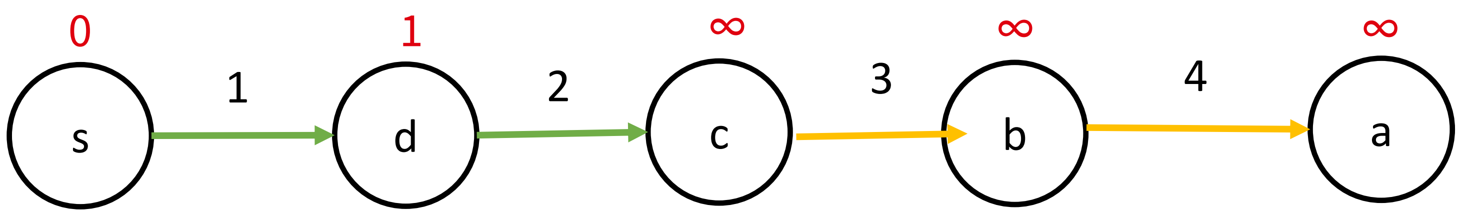
Bellman-Ford Illustration

After round 1



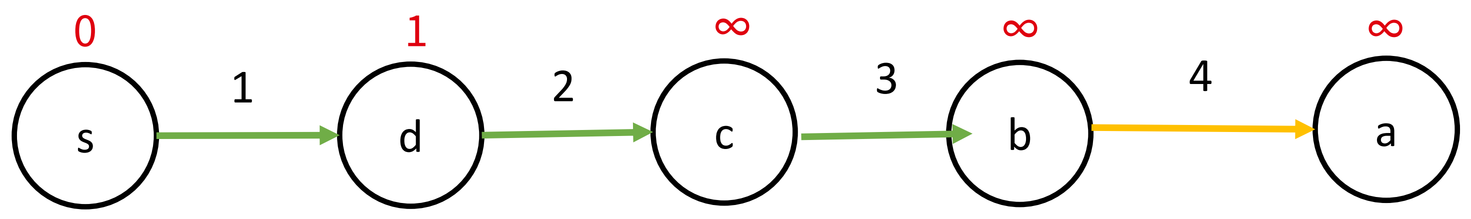
Bellman-Ford Illustration

After round 2



Bellman-Ford Illustration

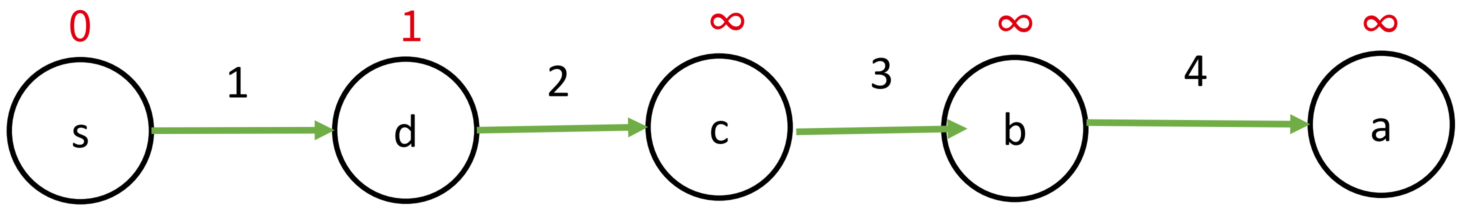
After round 3



Bellman-Ford Illustration

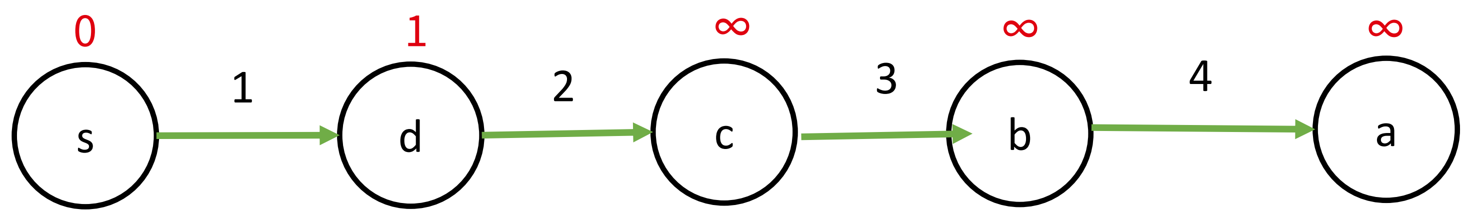
After round 4

So we did |V| - 1 = r rounds.



Bellman-Ford Illustration

After round 4
So we did |V| - 1 = r rounds.



This example is actually easy because it is DAG
However, if there are “backward” edges of high weight, we can’t assume it’s a DAG.

Why Bellman-Ford Works

Let’s consider two cases:

- True case: there is no negative weight cycle reachable from s
  - We compute a shortest path tree rooted at s
- False case: there is a negative weight cycle reachable from s
  - We are supposed to declare that it has a negative weight cycle.

Why Bellman-Ford Works (true case)

- We know
  - $v.d \geq \delta(s, v)$ and once $v.d = \delta(s, v)$, it remains so (Lemma 24.11)
  - If there's no path from s to v, then $v.d = \delta(s, v) = \infty$ always (Lemma 24.12)
- So, it suffices to show that we have $v.d = \delta(s, v)$ at the end.

Why Bellman-Ford Works (true case)

Due to path relaxation property

- It suffices to show that we have $v.d = \delta(s, v)$ at the end.
  - $v.d$ never increases
  - Say $<s = v_0, v_1, \ldots, v = v_k>$ is a shortest path from s to v. $(k \leq V - 1)$
  - In round 1, we relax edge $(s = v_0, v_1)$, so, $v_1.d \leq \delta(s, v_1)$
    - We relax every edge in each round
  - In round 2, we relax edge (v_1, v_2), so, $v_2.d \leq \delta(s, v_1) + w(v_1, v_2) = \delta(s, v_2)$
    - Due to optimality of subpaths of a shortest path
  - In round k, we relax edge (v_{k-1}, v_k), so, $v_k.d \leq \delta(s, v_{k-1}) + w(v_{k-1}, v_k) = \delta(s, v = v_k)$
    - $k \leq V - 1$; there is a shortest path that is simple
  -

Why Bellman-Ford Works (false case)

Due to path relaxation property

- False case: there is a negative weight cycle reachable from s
  - Let $c = \langle v_0, v_1, \ldots, v_k = v_0 \rangle$ be such a cycle
- Suppose BF returned true, which means no edge relaxation changed v.d.

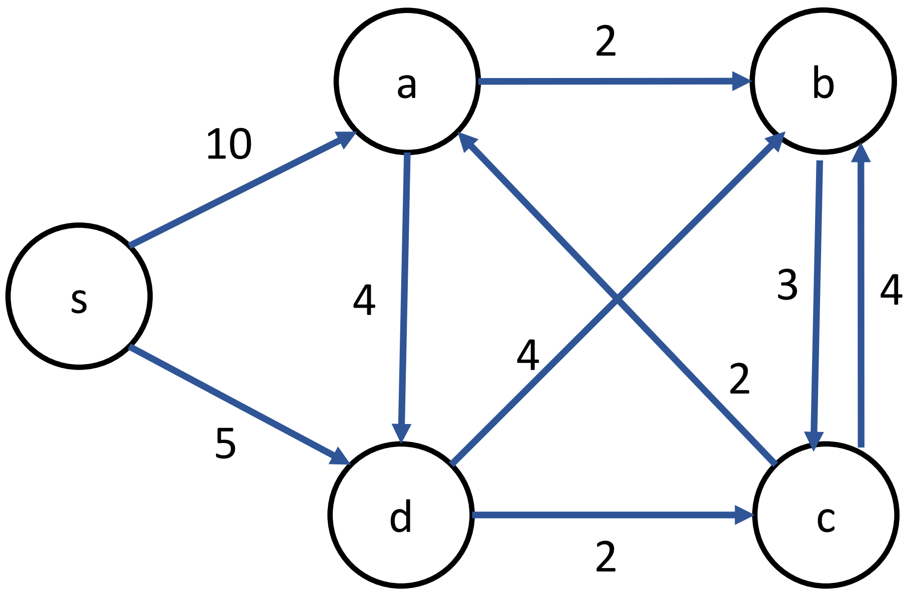
$$
\begin{aligned}
v_1.d &\leq v_0.d + w(v_0, v_1) \\
v_2.d &\leq v_1.d + w(v_1, v_2) \\
&\ldots \\
v_k.d &\leq v_{k-1}.d + w(v_{k-1}, v_k) \\
v_0.d &\leq v_k.d + w(v_k, v_0)
\end{aligned}
$$

So, we have $0 \leq w(v_0, v_1) + w(v_1, v_2) + \ldots + w(v_k, v_0)$, a contradiction.

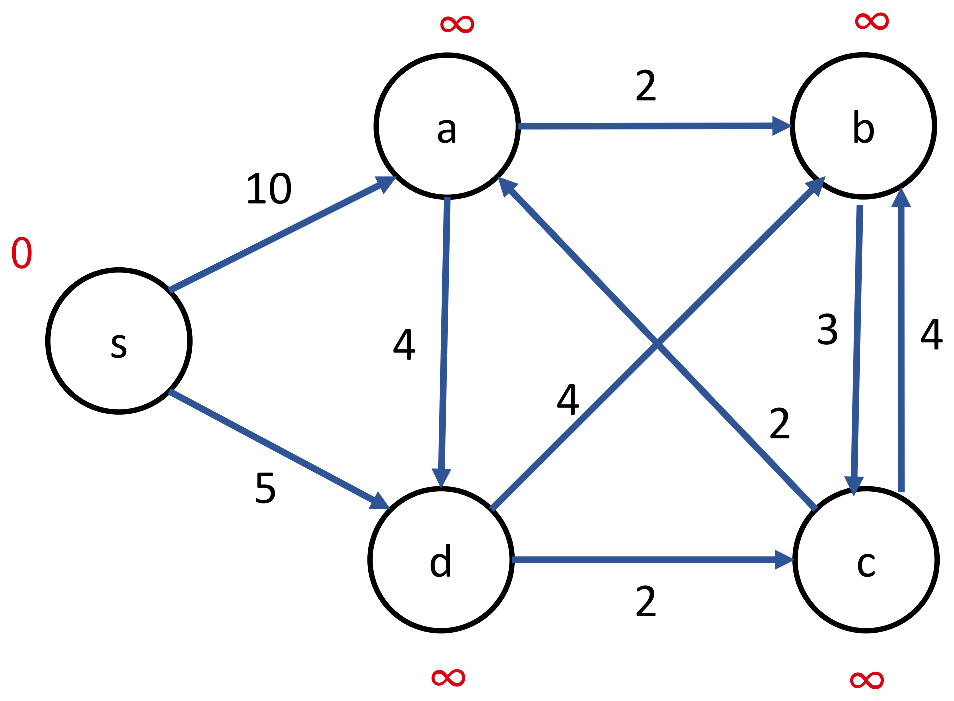
Dijkstra Algorithm

- Can be used when edges have non-negative weights

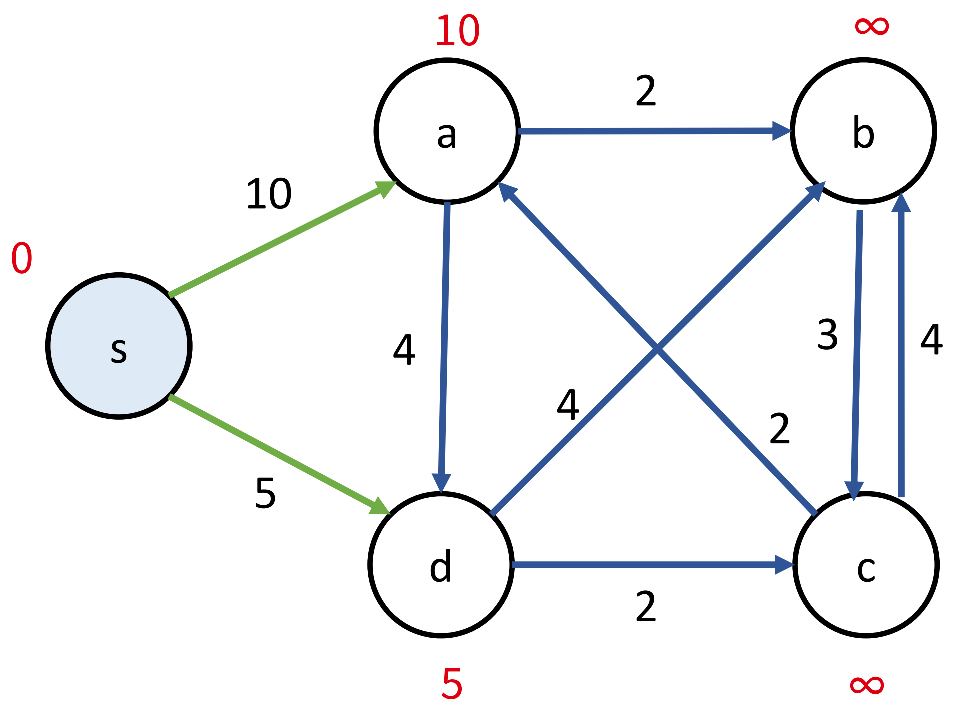
Dijkstra Algorithm



Dijkstra Algorithm



Dijkstra Algorithm



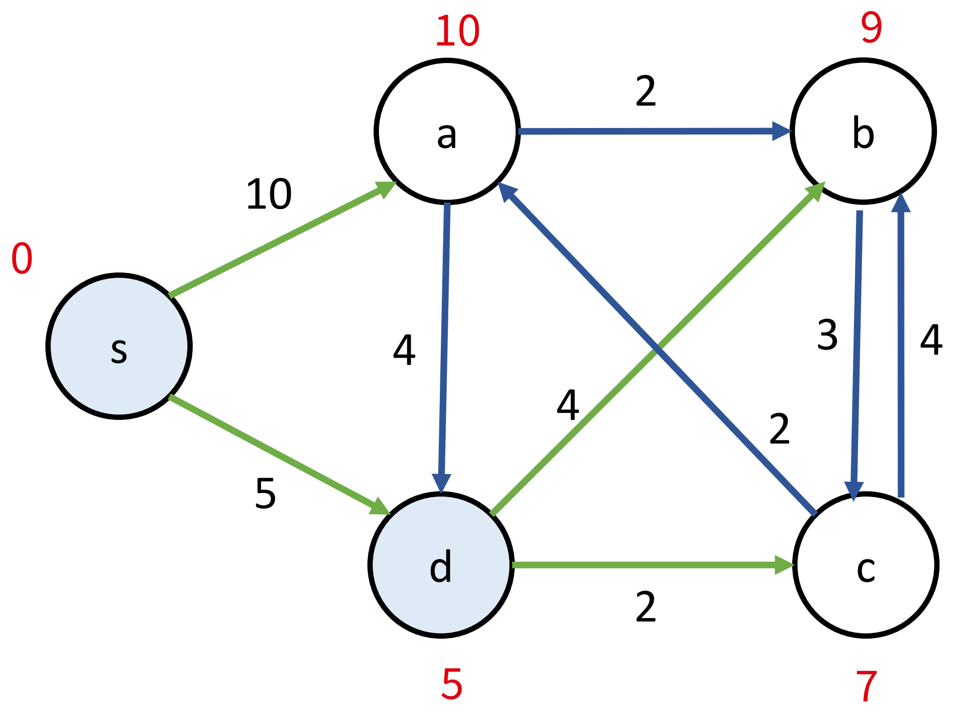
In each iteration, consider v with min v.d, and relax its out-going edges

Color v blue

Green edges encode shortest paths

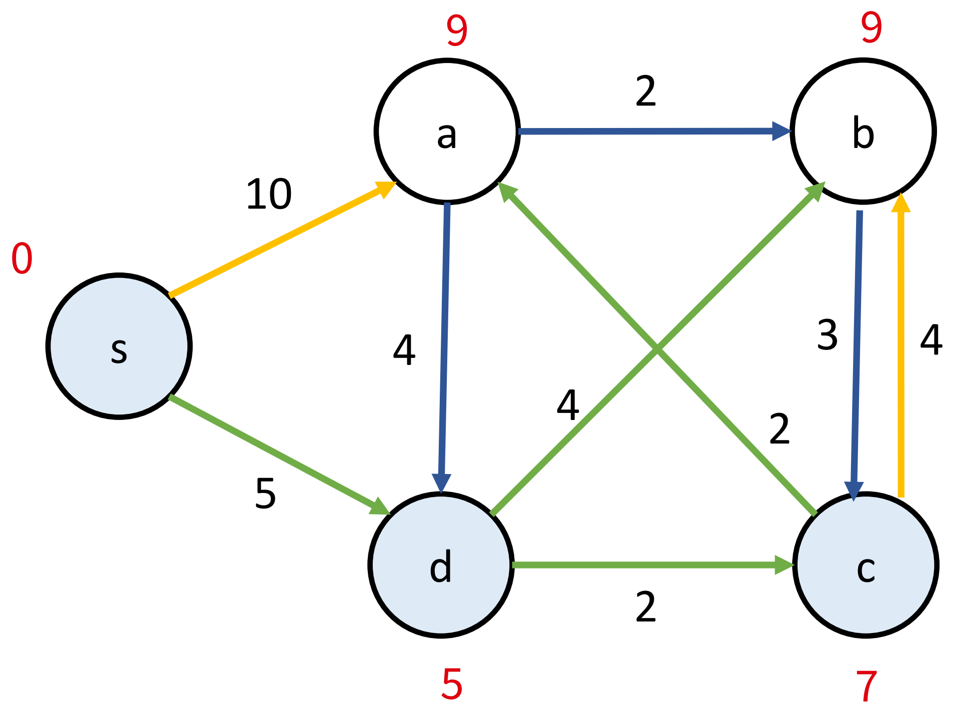
Orange edges mean we relaxed them

Dijkstra Algorithm



In each iteration, consider v with min v.d, and relax its out-going edges
Color v blue
Green edges encode shortest paths
Orange edges mean we relaxed them

Dijkstra Algorithm



In each iteration, consider v with min v.d, and relax its out-going edges

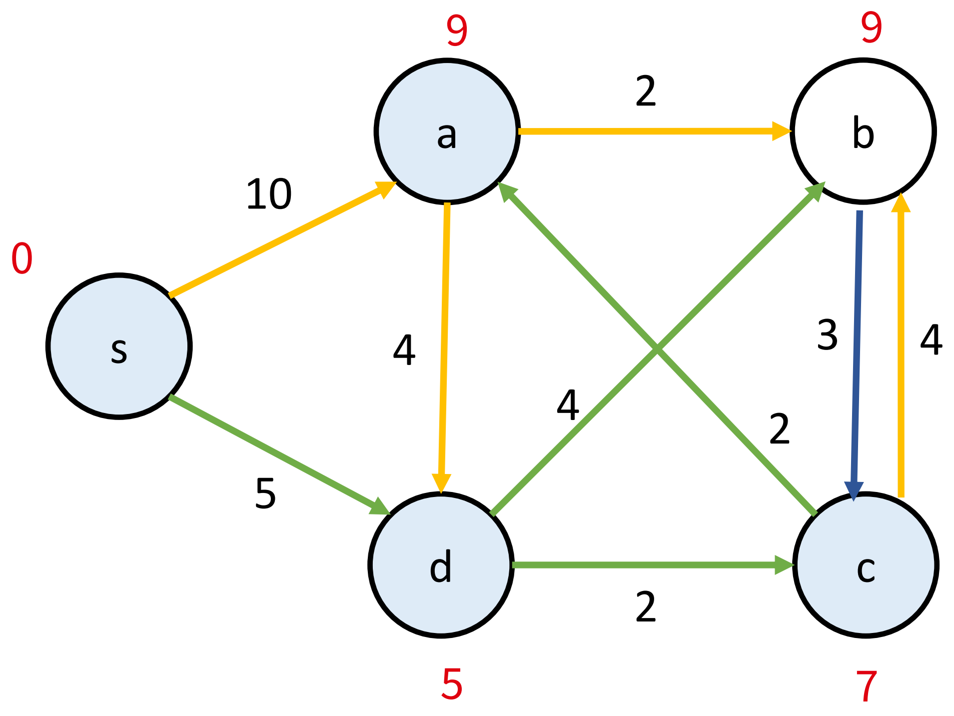
Color v blue

Green edges encode shortest paths

(in implementation we use v.π)

Orange edges mean we relaxed them

Dijkstra Algorithm



In each iteration, consider v with min v.d, and relax its out-going edges

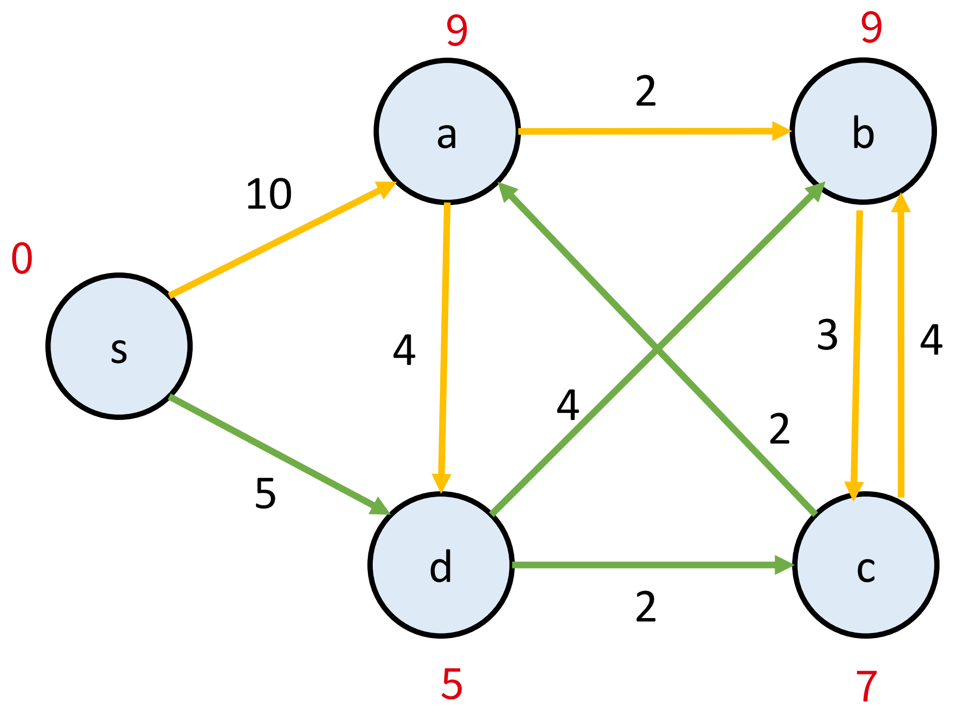
Color v blue

Green edges encode shortest paths

(in implementation we use v.π)

Orange edges mean we relaxed them

Dijkstra Algorithm



In each iteration, consider v with min v.d, and relax its out-going edges

Color v blue

Green edges encode shortest paths

(in implementation we use v.π)

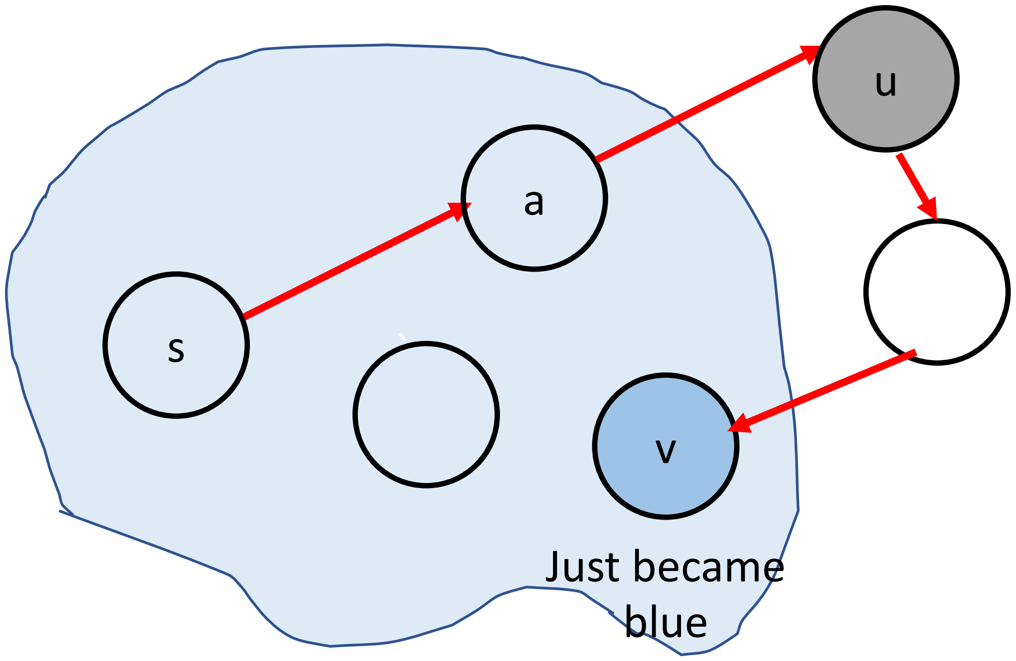
Orange edges mean we relaxed them

Why Dijkstra Algorithm Works

Blue vertex means we've found its shortest path from s

Once a vertex become blue, it remains blue

Why can't the following happen?



What if there is a shorter path using non-blue vertices?

- v :First vertex that became blue, but $v.d > \delta(s, v)$
- Red path is a shortest path to v
- u is the first non-blue vertex on the red path
- $u.d \le \delta(s, a) + w(a, u)$
  - $a.d = \delta(s, a)$, and we relaxed (a,u) when a became blue
- $u.d = \delta(s, u)$; due to optimality of subpaths
- We had $v.d > \delta(s, v) \ge \delta(s, u) = u.d$ just before coloring v blue. We must have chosen u over v.

Comparison of Algorithms

- Bellman ford:
  - Can handle negative weight edges
  - RT: O(EV)
- Dijkstra:
  - Can only handle non-negative weight edges
  - RT: O(E log V)
- DAG
  - Must have no cycle; can have negative weight edges
  - RT: O(E + V)