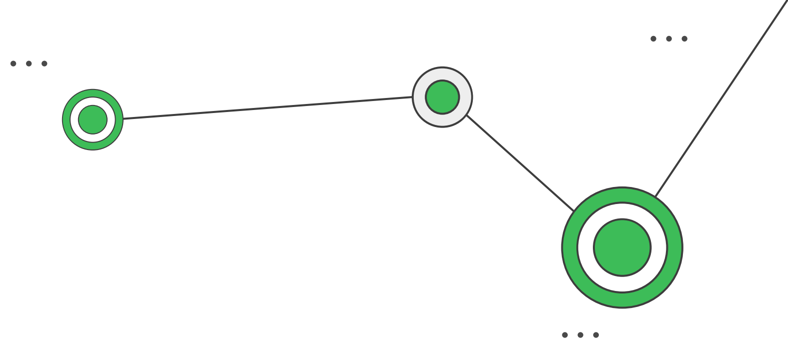
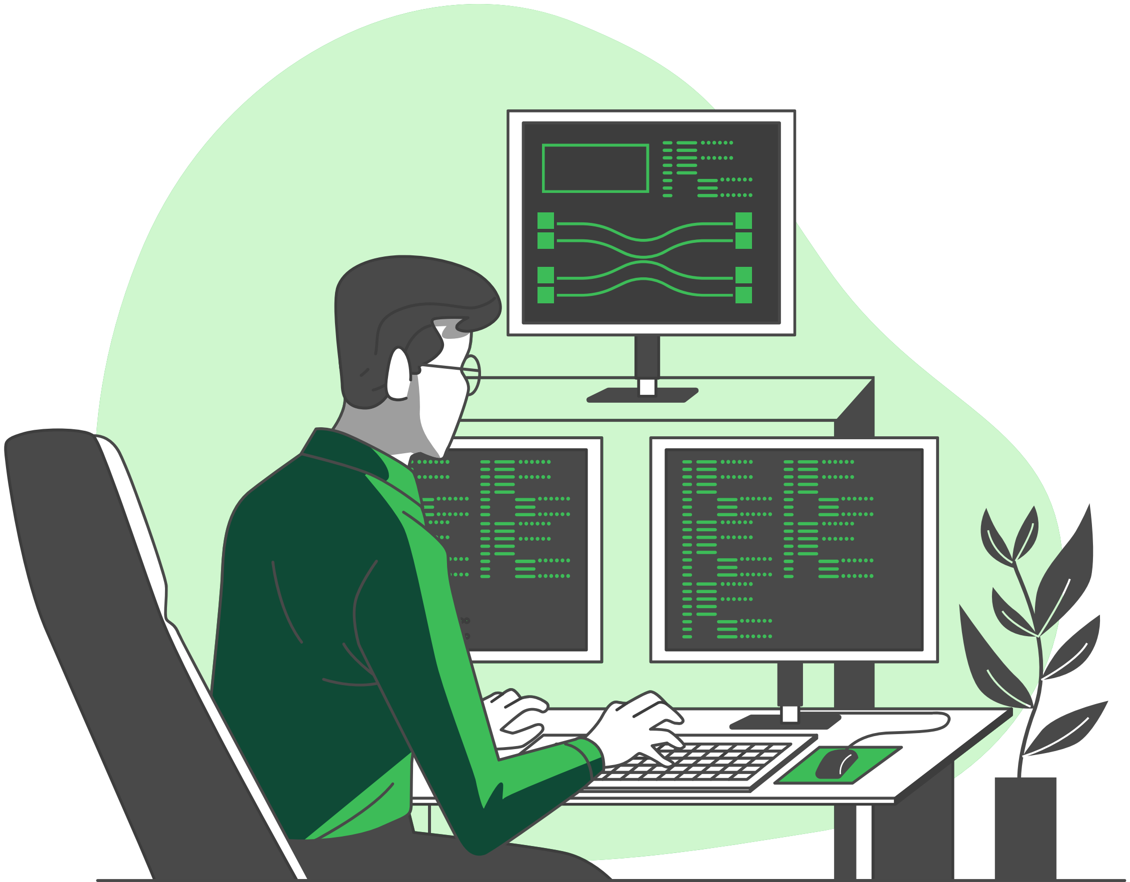




Bellman Ford

CS 251 - Data Structures and Algorithms

Note:
Slides complement the discussion in class

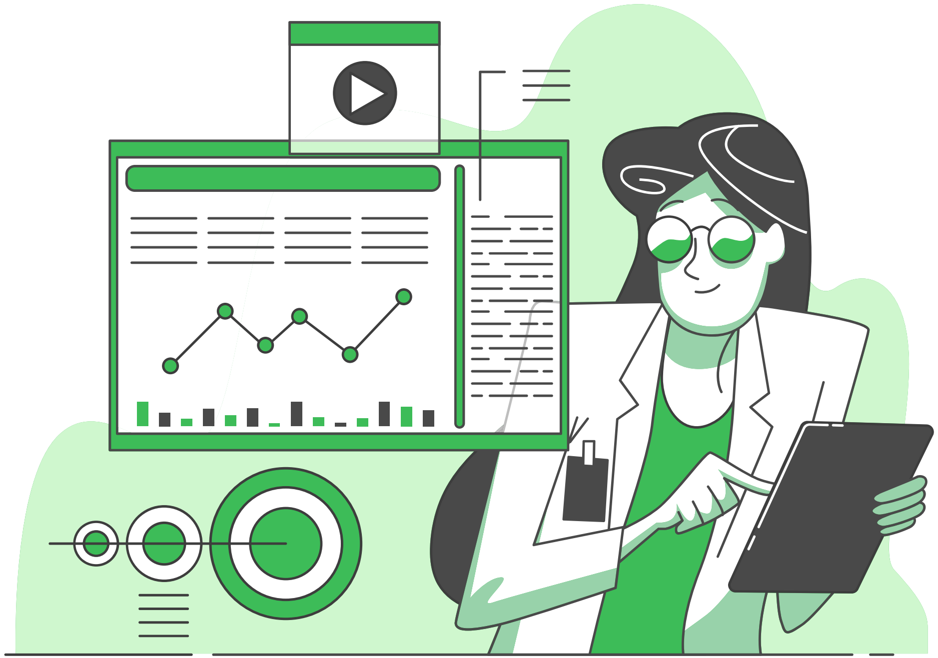
Table of Contents

01

Bellman-Ford Algorithm

Shortest paths with negative weights

...



01

Bellman–Ford Algorithm

Shortest paths with negative weights

Application: Arbitrage Opportunities

Wanna make an easy profit? Use graphs!

If you have USD $10,000...

- How many Swiss francs (CHF) can you buy?
- How many Euros (EUR) can you buy?
- How could you exchange money and make a profit?

	USD	**EUR**	**GBP**	**CHF**	**CAD**
USD	1	0.741	0.657	1.061	1.005
EUR	1.349	1	0.888	1.433	1.366
GBP	1.521	1.126	1	1.614	1.538
CHF	0.942	0.698	0.619	1	0.953
CAD	0.995	0.732	0.650	1.049	1

Application: Arbitrage Opportunities

Buy USD 10,000 in EUR: 10,000 * 0.741
Now you have **EUR 7,410**.

Buy CAD with your EUR: 7,410 * 1.366
You get **CAD 10,122**.

BUY USD with your CAD: 10,122 * 0.995
You get **USD 10,071** (**Profit!**)

	USD	EUR	GBP	CHF	CAD
USD	1	0.741	0.657	1.061	1.005
EUR	1.349	1	0.888	1.433	1.366
GBP	1.521	1.126	1	1.614	1.538
CHF	0.942	0.698	0.619	1	0.953
CAD	0.995	0.732	0.650	1.049	1

Disclaimer: It is more complicated that it looks like due to limitations, fees, regulations, and currency volatility.

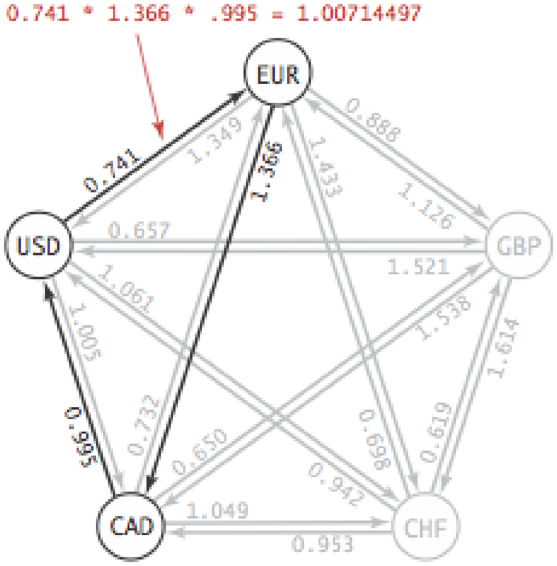
Application: Arbitrage Opportunities

Challenges:

1. Understand the problem in terms of graphs.
2. Find a cycle in the graph such that it “always increases its weight”. Is that even possible?

	USD	EUR	GBP	CHF	CAD
USD	1	0.741	0.657	1.061	1.005
EUR	1.349	1	0.888	1.433	1.366
GBP	1.521	1.126	1	1.614	1.538
CHF	0.942	0.698	0.619	1	0.953
CAD	0.995	0.732	0.650	1.049	1

Application: Arbitrage Opportunities

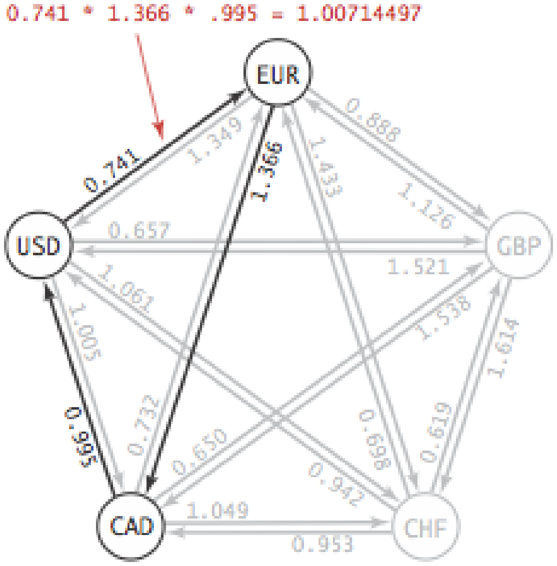


An arbitrage opportunity

	USD	EUR	GBP	CHF	CAD
USD	1	0.741	0.657	1.061	1.005
EUR	1.349	1	0.888	1.433	1.366
GBP	1.521	1.126	1	1.614	1.538
CHF	0.942	0.698	0.619	1	0.953
CAD	0.995	0.732	0.650	1.049	1

Algorithms, 4th edition, Sedgewick and Wayne. Official web site

Application: Arbitrage Opportunities



An arbitrage opportunity

Transform the problem: Let u, v, w be the weights of three edges forming a cycle in a graph.

Goal: $uvw > 1$

$$uvw > 1$$
$$\log(uvw) > \log(1)$$
$$\log(u) + \log(v) + \log(w) > 0$$
$$-1(\log(u) + \log(v) + \log(w)) < 0$$
$$-\log(u) - \log(v) - \log(w) < 0$$

Now the problem is to find a negative weight cycle in the graph. Let's use the **Bellman-Ford algorithm**.

Richard Bellman. "On a Routing Problem." Quarterly of Applied Mathematics, vol. 16, no. 1, pp. 87-90, 1958.

Lester R. Ford, Jr. "Network Flow Theory." RAND Corporation, P-923, 1956.

ON A ROUTING PROBLEM*

By RICHARD BELLMAN (*The RAND Corporation*)

Summary. Given a set of N cities, with every two linked by a road, and the times required to traverse these roads, we wish to determine the path from one given city to another given city which minimizes the travel time. The times are not directly proportional to the distances due to varying quality of roads and varying quantities of traffic.

The functional equation technique of dynamic programming, combined with approximation in policy space, yields an iterative algorithm which converges after at most $(N - 1)$ iterations.

1. Introduction. The problem we wish to treat is a combinatorial one involving the determination of an optimal route from one point to another. These problems are usually difficult when we allow a continuum, and when we admit only a discrete set of paths, as we shall do below, they are notoriously so.

The purpose of this paper is to show that the functional equation technique of dynamic programming, [1, 2], combined with the concept of approximation in policy space, yields a method of successive approximations which is readily accessible to either hand or machine computation for problems of realistic magnitude. The method is distinguished by the fact that it is a method of exhaustion, i.e. it converges after a finite number of iterations, bounded in advance.

2. Formulation. Consider a set of N cities, numbered in some arbitrary fashion from 1 to N, with every two linked by a direct road. The time required to travel from i to j is not directly proportional to the distance between i and j, due to road conditions and traffic. Given the matrix $T = (t_{ij})$, not necessarily symmetric, where t_{ij} is the time required to travel from i to j, we wish to trace a path between 1 and N which consumes minimum time.

Since there are only a finite number of paths available, the problem reduces to choosing the smallest from a finite set of numbers. This direct, or enumerative, approach is impossible to execute, however, for values of N of the order of magnitude of 20.

We shall construct a search technique which greatly reduces the time required to find minimal paths.

3. Functional equation approach. Let us now introduce a dynamic programming approach. Let

$$f_i = \text{the time required to travel from } i \text{ to } N,\ i = 1, 2, \cdots, N-1, \text{ using an optimal policy,} \tag{3.1}$$

with $f_N = 0$.

Employing the principle of optimality, we see that the f_i satisfy the nonlinear system of equations

$$\begin{aligned} f_i &= \operatorname*{Min}_{j \neq i} [t_{ij} + f_j], \qquad i = 1, 2, \cdots, N-1, \\ f_N &= 0. \end{aligned} \tag{3.2}$$

*Received January 30, 1957.

This content downloaded from
73.103.76.200 on Sun, 09 Jun 2024 02:23:16 +00:00
All use subject to https://about.jstor.org/terms

https://www.jstor.org/stable/43634538

Bellman–Ford Algorithm

1. Given a digraph $G = \{V, E\}$ and a source vertex $v \in V$, Bellman-Ford's algorithm finds the shortest path from v to every other vertex in the digraph.

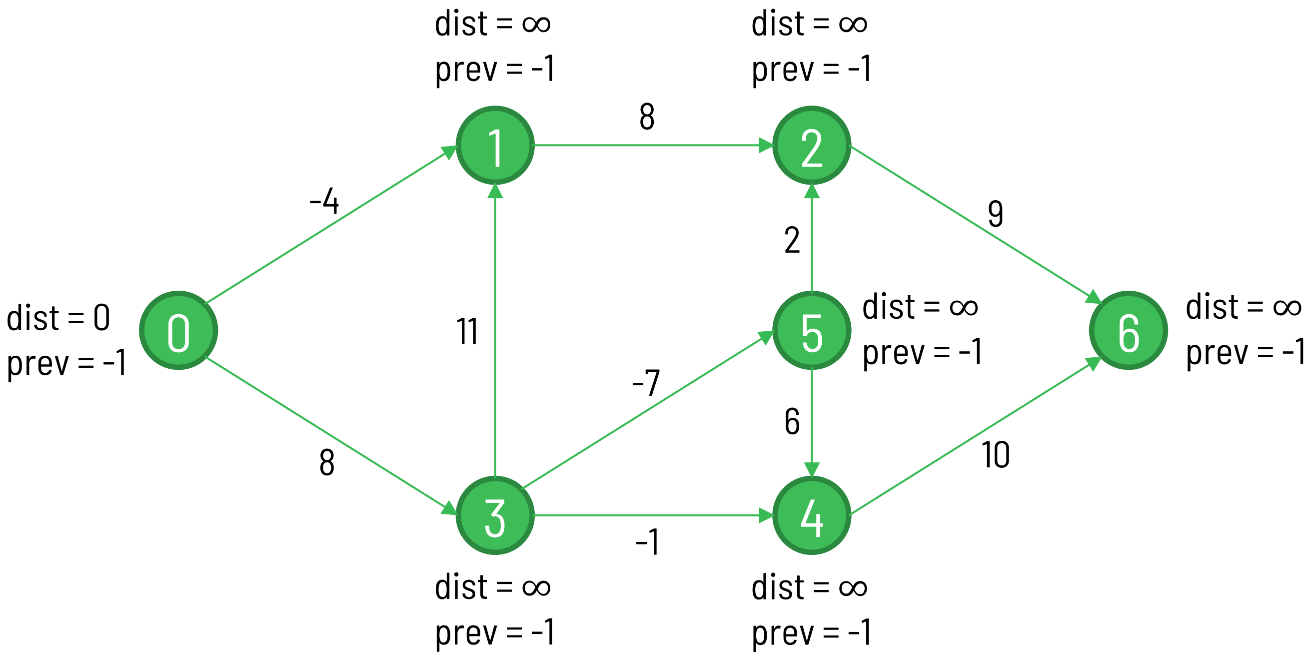
2. Bellman-Ford's algorithm works on any weighted digraph (even with negative weights).

3. The last pass through the edges will determine if there are negative weight cycles.

Bellman–Ford Algorithm

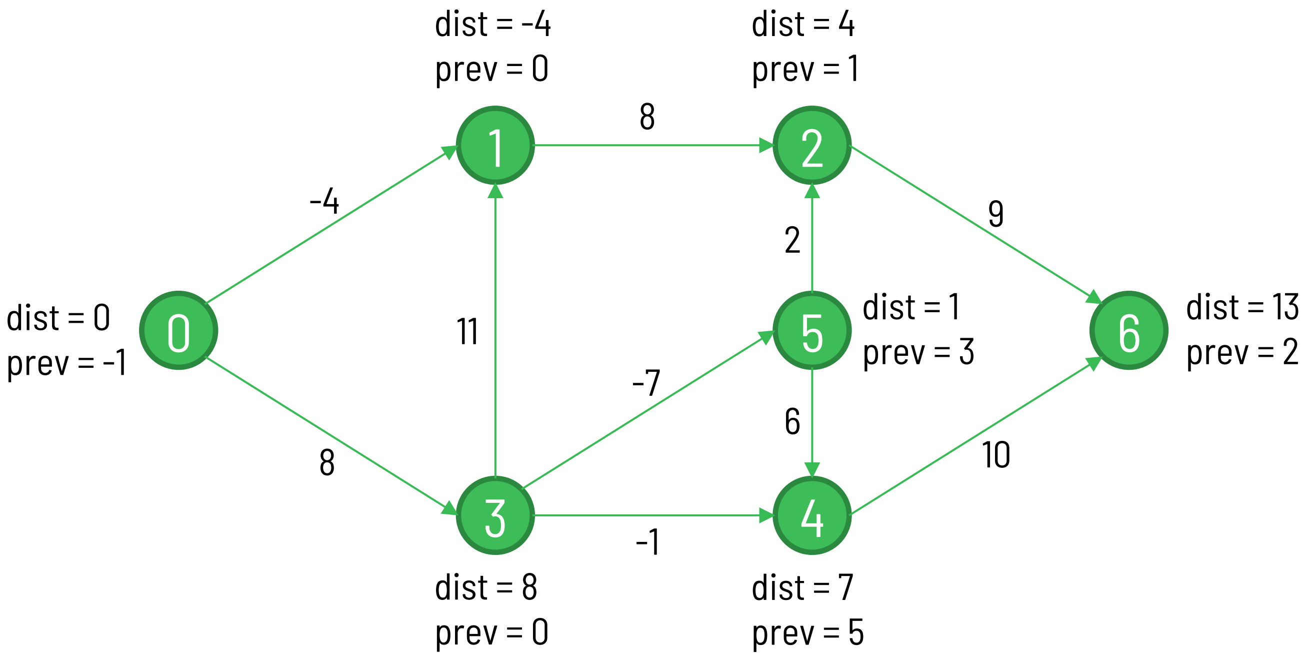
```
algorithm BellmanFord(G(V,E), s ∈ V)

   let dist:V → ℤ
   let prev:V → V

   for each v ∈ V do
      dist[v] ← ∞
      prev[v] ← -1
   end for
   dist[s] ← 0

   for i from 1 to |V| − 1 do
      for each e = (u,v) ∈ E do
         d ← dist[u] + weight(e)
         if d < dist[v] then
            dist[v] ← d
            prev[v] ← u
         end if
      end for
   end for

   for each e = (u,v) ∈ E do
      if dist[u] + weight(e) < dist[v] then
         error "Negative Weight Cycle"
      end if
   end for

   return dist, prev
end algorithm
```

Remember: $\text{dist}(v) = \min\left(\text{dist}(v), \text{dist}(u) + \text{weight}(u, v)\right)$



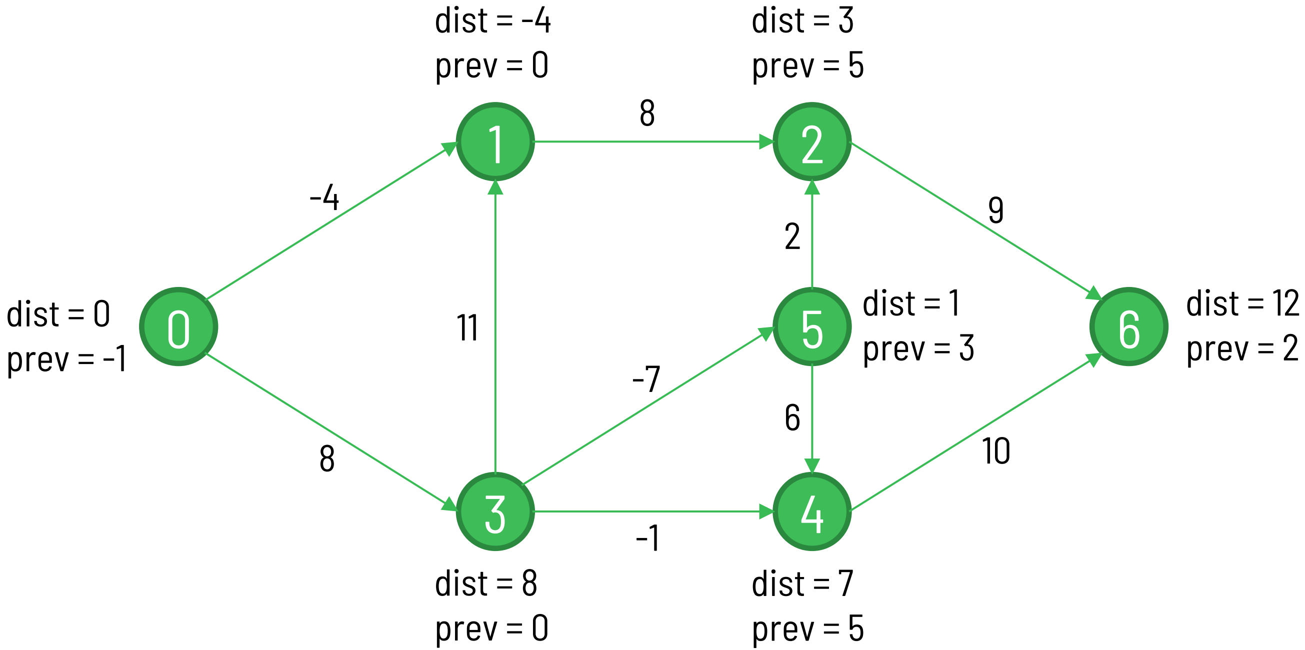
Edge	i = 1	i = 2	i = 3
(0, 1)			
(0, 3)			
(1, 2)			
(3, 1)			
(5, 2)			
(3, 5)			
(5, 4)			
(3, 4)			
(4, 6)			
(2, 6)			

Remember: $\text{dist}(v) = \min\big(\text{dist}(v), \text{dist}(u) + \text{weight}(u, v)\big)$



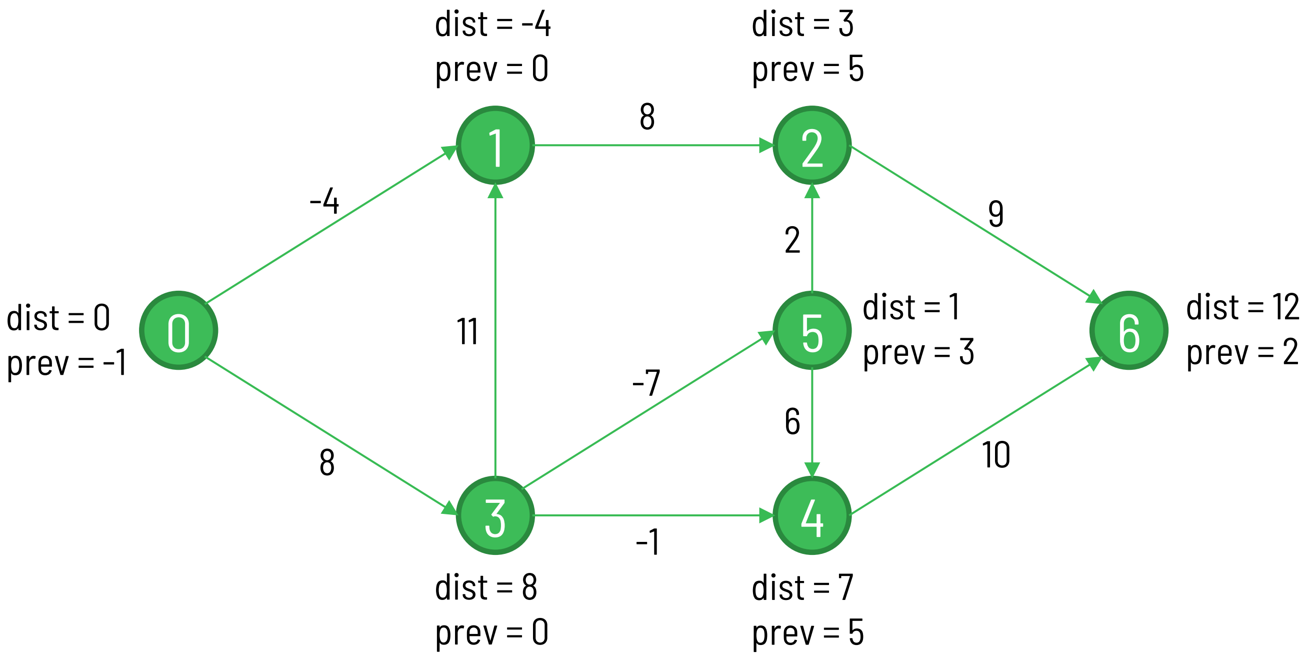
Edge	i = 1	i = 2	i = 3
(0, 1)	✓		
(0, 3)	✓		
(1, 2)	✓		
(3, 1)	X		
(5, 2)	X		
(3, 5)	✓		
(5, 4)	✓		
(3, 4)	X		
(4, 6)	✓		
(2, 6)	✓		

Remember: $\mathrm{dist}(v) = \min\big(\mathrm{dist}(v), \mathrm{dist}(u) + \mathrm{weight}(u, v)\big)$



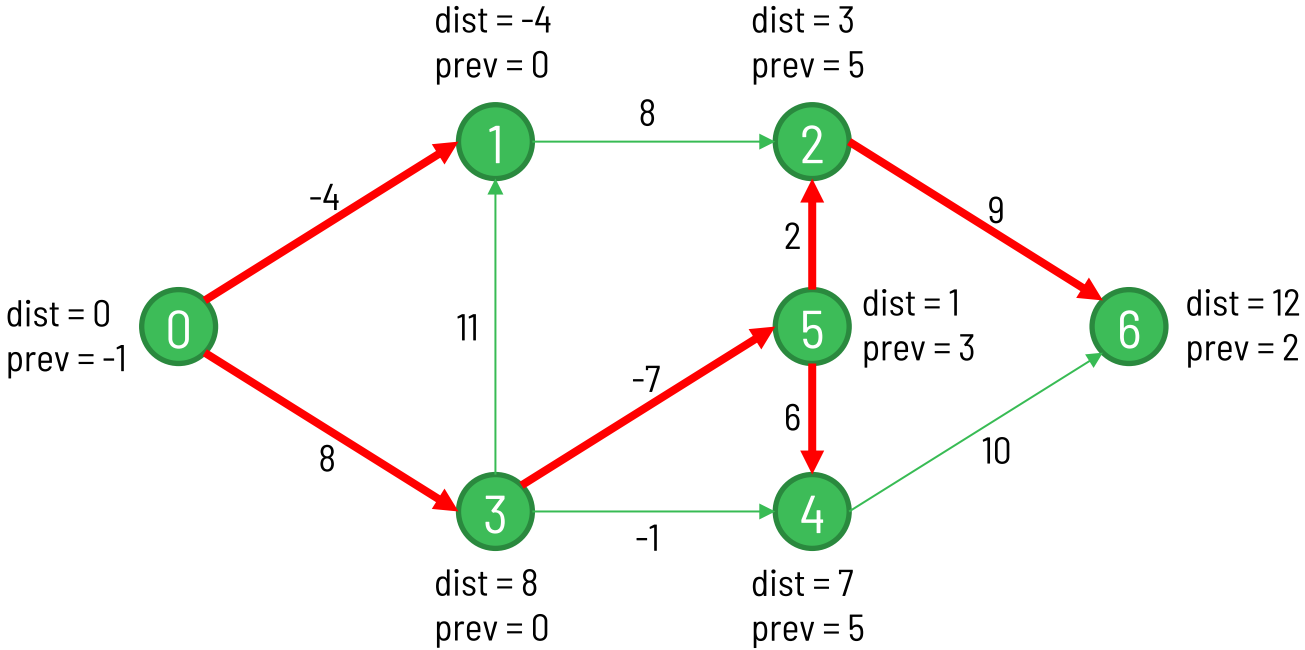
Edge	i = 1	i = 2	i = 3
(0, 1)	✓	X	
(0, 3)	✓	X	
(1, 2)	✓	X	
(3, 1)	X	X	
(5, 2)	X	✓	
(3, 5)	✓	X	
(5, 4)	✓	X	
(3, 4)	X	X	
(4, 6)	✓	X	
(2, 6)	✓	✓	

Remember: $\text{dist}(v) = \min\big(\text{dist}(v), \text{dist}(u) + \text{weight}(u, v)\big)$



Edge	i = 1	i = 2	i = 3
(0, 1)	✓	X	X
(0, 3)	✓	X	X
(1, 2)	✓	X	X
(3, 1)	X	X	X
(5, 2)	X	✓	X
(3, 5)	✓	X	X
(5, 4)	✓	X	X
(3, 4)	X	X	X
(4, 6)	✓	X	X
(2, 6)	✓	✓	X

Remember: $\mathrm{dist}(v) = \min\big(\mathrm{dist}(v), \mathrm{dist}(u) + \mathrm{weight}(u, v)\big)$



Edge	i = 1	i = 2	i = 3
(0, 1)	✓	X	X
(0, 3)	✓	X	X
(1, 2)	✓	X	X
(3, 1)	X	X	X
(5, 2)	X	✓	X
(3, 5)	✓	X	X
(5, 4)	✓	X	X
(3, 4)	X	X	X
(4, 6)	✓	X	X
(2, 6)	✓	✓	X

```
algorithm BellmanFord(G(V,E), s ∈ V)

   let dist:V → ℤ
   let prev:V → V

   for each v ∈ V do
      dist[v] ← ∞
      prev[v] ← -1
   end for
   dist[s] ← 0

   for i from 1 to |V| − 1 do
      for each e = (u,v) ∈ E do
         d ← dist[u] + weight(e)
         if d < dist[v] then
            dist[v] ← d
            prev[v] ← u
         end if
      end for
   end for

   for each e = (u,v) ∈ E do
      if dist[u] + weight(e) < dist[v] then
         error "Negative Weight Cycle"
      end if
   end for

   return dist, prev
end algorithm
```

Runtime:

- Initializing arrays: $O(|V|)$
- Resetting values in the arrays (aka. Edge relaxation): $O(|V||E|)$
- Checking for negative weight cycles: $O(|E|)$

Bellman-Ford's Runtime: $O(|V||E|)$

We’re Done!

Do you have any questions?

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, infographics & images by Freepik and illustrations by Stories