


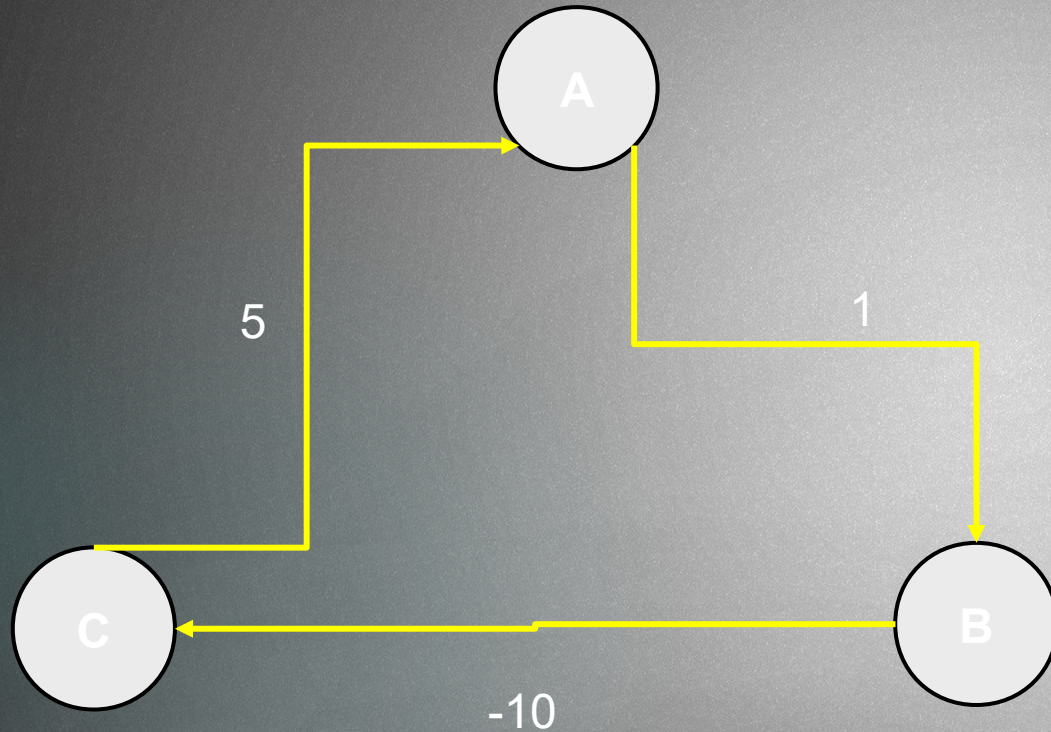


SHORTEST PATH

BELLMAN-FORD ALGORITHM

- 
- ▶ Invented in 1958 by Bellman and Ford independently
 - ▶ Slower than Dijkstra's but more robust: it can handle negative edge weights too
 - ▶ Dijkstra algorithm choose the edge greedily, with the lowest cost: Bellman-Ford relaxes all edges at the same time for $V-1$ iteration
 - ▶ Running time is $O(V \cdot E)$
 - ▶ Does $V-1$ iteration + 1 to detect cycles: if cost decreases in the V -th iteration, than there is a negative cycle, because all the paths are traversen up to the $V-1$ iteration !!!

Negative cycle:



What is the problem?

If we would like to find a path with the minimum cost we have to go $A \rightarrow B \rightarrow C \rightarrow A$ to decrease the overall cost

And a next cycle: decrease the cost again

And again ...

Real life scenarios: no negative cycles at all ... but sometimes we transform a problem into a graph with positive / negative edge weights and looking for some negative cycles !!!

Bellman-Ford: pseudocode

```
function BellmanFordAlgorithm(vertices, edges, source)
```

```
    distance[source] = 0
```

```
    for v in Graph
```

```
        distance[v] = inf
```

```
        predecessor[v] = undefined // previous node in the shortest path
```

```
    for i=1...num_vertexes-1
```

```
        for each edge (u,v) with weight w in edges
```

```
            tempDist = distance[u] + w
```

```
            if tempDist < distance[v]
```

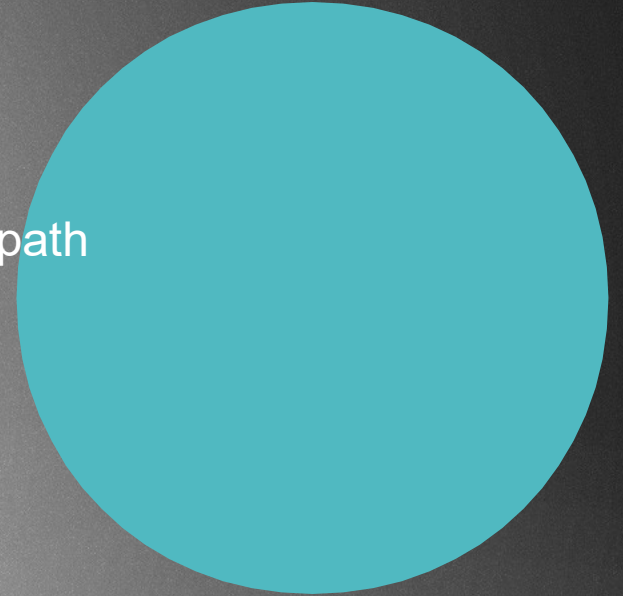
```
                distance[v] = tempDist
```

```
                predecessor[v] = u
```

```
    for each edge (u,v) with weight w in edges
```

```
        if distance[u] + w < distance[v]
```

```
            error: „Negative cycle detected”
```



Bellman-Ford: pseudocode

```
function BellmanFordAlgorithm(vertices, edges, source)
```

```
    distance[source] = 0
```

```
    for v in Graph
```

```
        distance[v] = inf
```

```
        predecessor[v] = undefined // previous node in the shortest path
```

```
    for i=1...num_vertexes-1
```

```
        for each edge (u,v) with weight w in edges
```

```
            tempDist = distance[u] + w
```

```
            if tempDist < distance[v]
```

```
                distance[v] = tempDist
```

```
                predecessor[v] = u
```

```
    for each edge (u,v) with weight w in edges
```

```
        if distance[u] + w < distance[v]
```

```
            error: „Negative cycle detected”
```



Bellman-Ford: pseudocode

```
function BellmanFordAlgorithm(vertices, edges, source)
```

```
    distance[source] = 0
```

```
    for v in Graph
```

```
        distance[v] = inf
```

```
        predecessor[v] = undefined // previous node in the shortest path
```

```
    for i=1...num_vertexes-1
```

```
        for each edge (u,v) with weight w in edges
```

```
            tempDist = distance[u] + w
```

```
            if tempDist < distance[v]
```

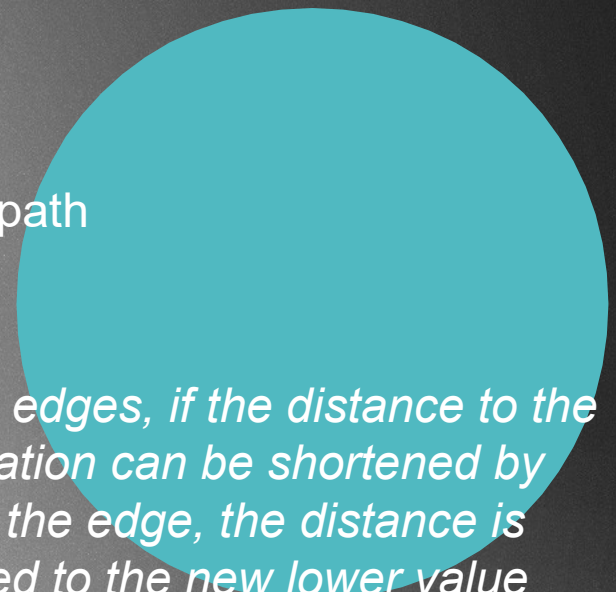
```
                distance[v] = tempDist
```

```
                predecessor[v] = u
```

```
    for each edge (u,v) with weight w in edges
```

```
        if distance[u] + w < distance[v]
```

```
            error: „Negative cycle detected”
```



For all edges, if the distance to the destination can be shortened by taking the edge, the distance is updated to the new lower value

V-1 times → we make relaxation

Bellman-Ford: pseudocode

```
function BellmanFordAlgorithm(vertices, edges, source)
```

```
    distance[source] = 0
```

```
    for v in Graph
```

```
        distance[v] = inf
```

```
        predecessor[v] = undefined // previous node in the shortest path
```

```
    for i=1...num_vertexes-1
```

```
        for each edge (u,v) with weight w in edges
```

```
            tempDist = distance[u] + w
```

```
            if tempDist < distance[v]
```

```
                distance[v] = tempDist
```

```
                predecessor[v] = u
```

```
    for each edge (u,v) with weight w in edges
```

```
        if distance[u] + w < distance[v]
```

```
            error: „Negative cycle detected”
```

Since the longest possible path without a cycle can be $V-1$ edges, the edges must be scanned $V-1$ times to ensure the shortest path has been found for all nodes

Bellman-Ford: pseudocode

```
function BellmanFordAlgorithm(vertices, edges, source)
```

```
    distance[source] = 0
```

```
    for v in Graph
```

```
        distance[v] = inf
```

```
        predecessor[v] = undefined // previous node in the shortest path
```

```
    for i=1...num_vertexes-1
```

```
        for each edge (u,v) with weight w in edges
```

```
            tempDist = distance[u] + w
```

```
            if tempDist < distance[v]
```

```
                distance[v] = tempDist
```

```
                predecessor[v] = u
```

```
    for each edge (u,v) with weight w in edges
```

```
        if distance[u] + w < distance[v]
```

```
            error: „Negative cycle detected”
```

A final scan of all the edges is performed
and if any distance is updated → means
there is a negative cycle !!!



1970: Yen optimization

- ▶ Yen algorithm: it is the Bellman-Ford algorithm with some optimization.
- ▶ We can terminate the algorithm if there is no change in the distances between two iterations !!!
- ▶ (we use the same technique in bubble sort)

Applications

- ▶ Cycle detection can prove to be very important
- ▶ Negative cycles as well → we have to run the Bellman-Ford algorithm that can handle negative edge weights by default
- ▶ On the FOREX market it can detect arbitrage situations !!!