CSCI 3104: Algorithms

Lecture 20: Dynamic Programming – Bellman Ford

Rachel Cox

Department of Computer Science

What will we learn today?

- ❑ Bellman Ford

Intro to Algorithms, CLRS:
Sections 24.1

Intro to Dynamic Programming

Dynamic Programming Algorithm: Breaks a problem into subproblems (optimal substructure), avoids recomputing the answer to overlapping subproblems by storing their answer and finding a good ordering of the subproblems

Optimal Substructure: Property that the solution to a problem can be computed using the solution to subproblems

Overlapping Subproblems: When breaking problem into subproblems, at least one subproblem appears more than once.

Memoization: Storing previously computed solutions to speed up algorithm/program runtime

Single-Source Shortest Path Problem

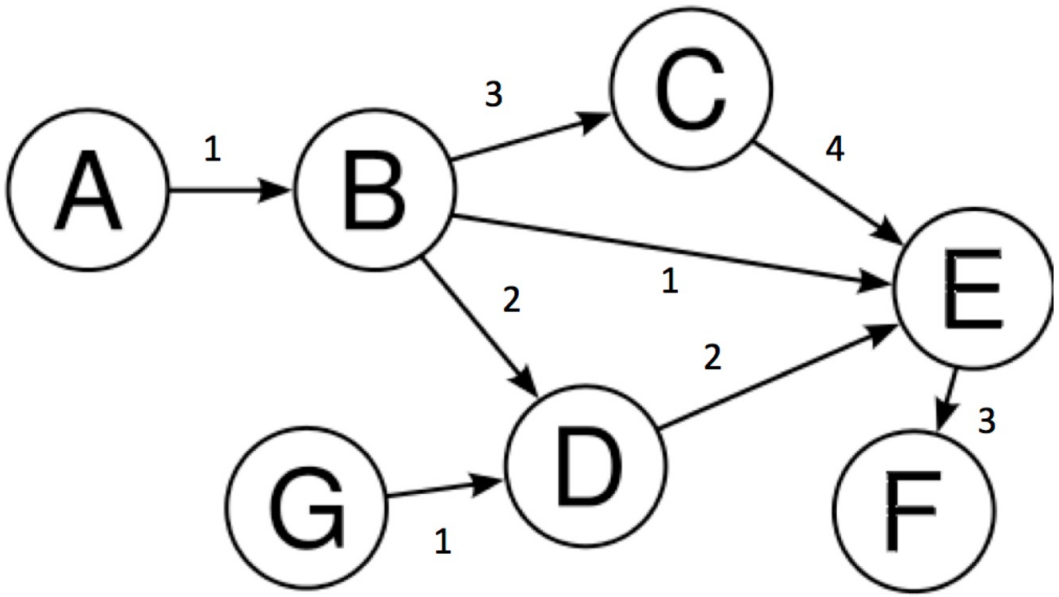
Goal: For every destination $v \in V$, compute the length (sum of edge costs) of a shortest $s - v$ path. In other words, find the length of the shortest path from some fixed vertex to every other vertex in a directed acyclic graph.

Input: A weighted DAG, $G = (V, E, w)$ and a special vertex $r \in V$

Output: A data structure that contains the length of the shortest path from r to every other vertex in V.

Single_Source Shortest Path Problem

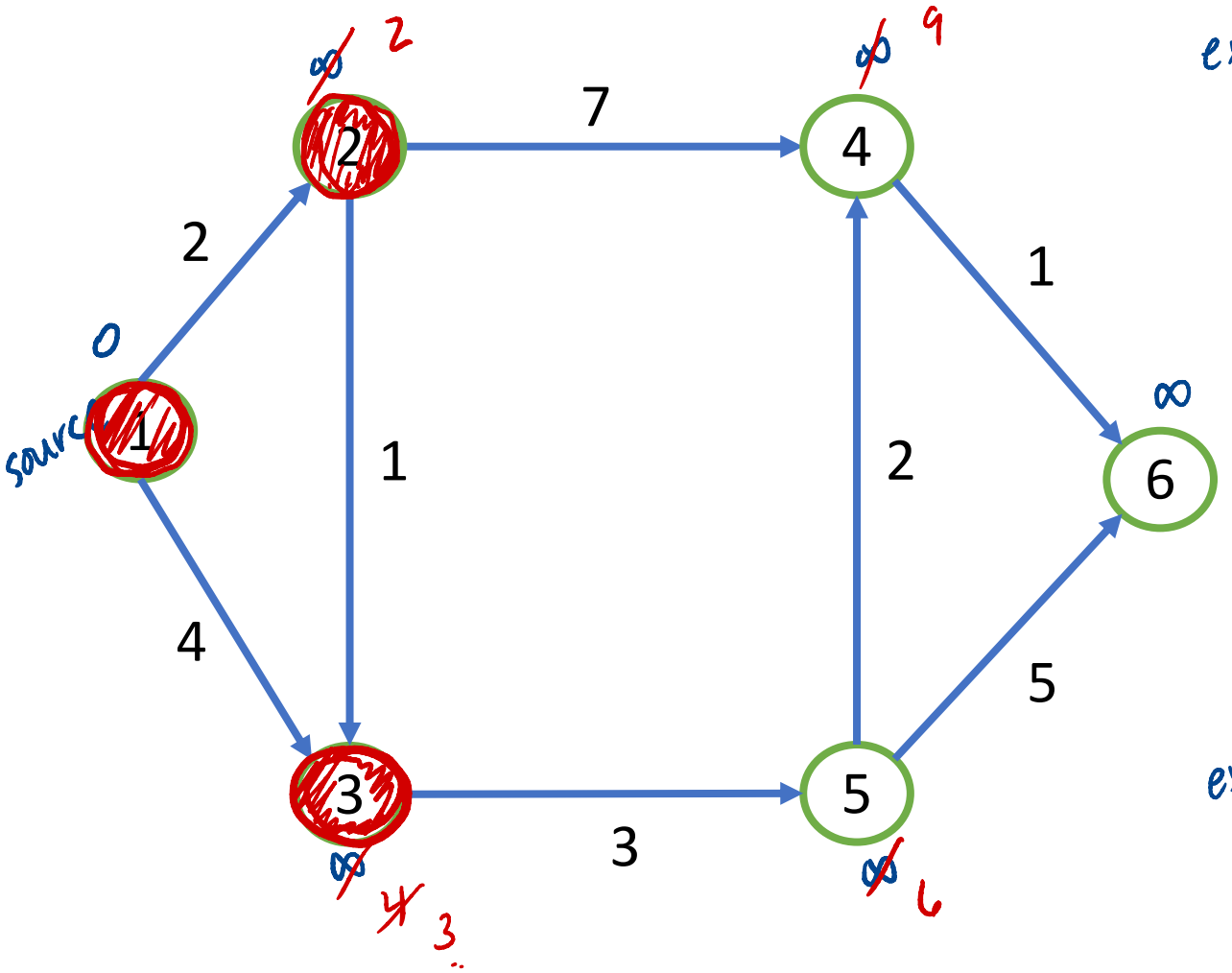
Example: Consider the directed graph $G = (V, E)$, pictured below. We define the minimum cost of a path from vertex u to vertex v to be the minimum edge weights along that path. Fill in the table below with the minimum cost to get from each node to every other node. Assume that paths start from the row node.



	A	B	C	D	E	F	G
→ A	0	1	4	3	2	5	NA
→ B	NA	0	3	2	1	4	NA
→ C	NA	NA	0	NA	4	7	NA
D	NA	NA	NA	0	2	5	NA
E	NA	NA	NA	NA	0	3	NA
F	NA	NA	NA	NA	NA	0	NA
G	NA	NA	NA	1	3	6	0

Dijkstra's Algorithm

Example: Use Dijkstra's to find the shortest path to each node.



expand ① neighbors 2, 3

$d(1) + c(1,2) = 0 + 2 = 2 < \infty$

So "Relax" $d(2) = 2$

$d(1) + c(1,3) = 0 + 4 = 4 < \infty$

So "relax" $d(3) = 4$

expand ② neighbors 3, 4

$d(2) + c(2,3) = 2 + 1 = 3 < 4$

"Relax" $d(3) = 3$

$d(2) + c(2,4) = 2 + 7 = 9 < \infty$

$d(4) = 9$

expand ③ neighbors 5

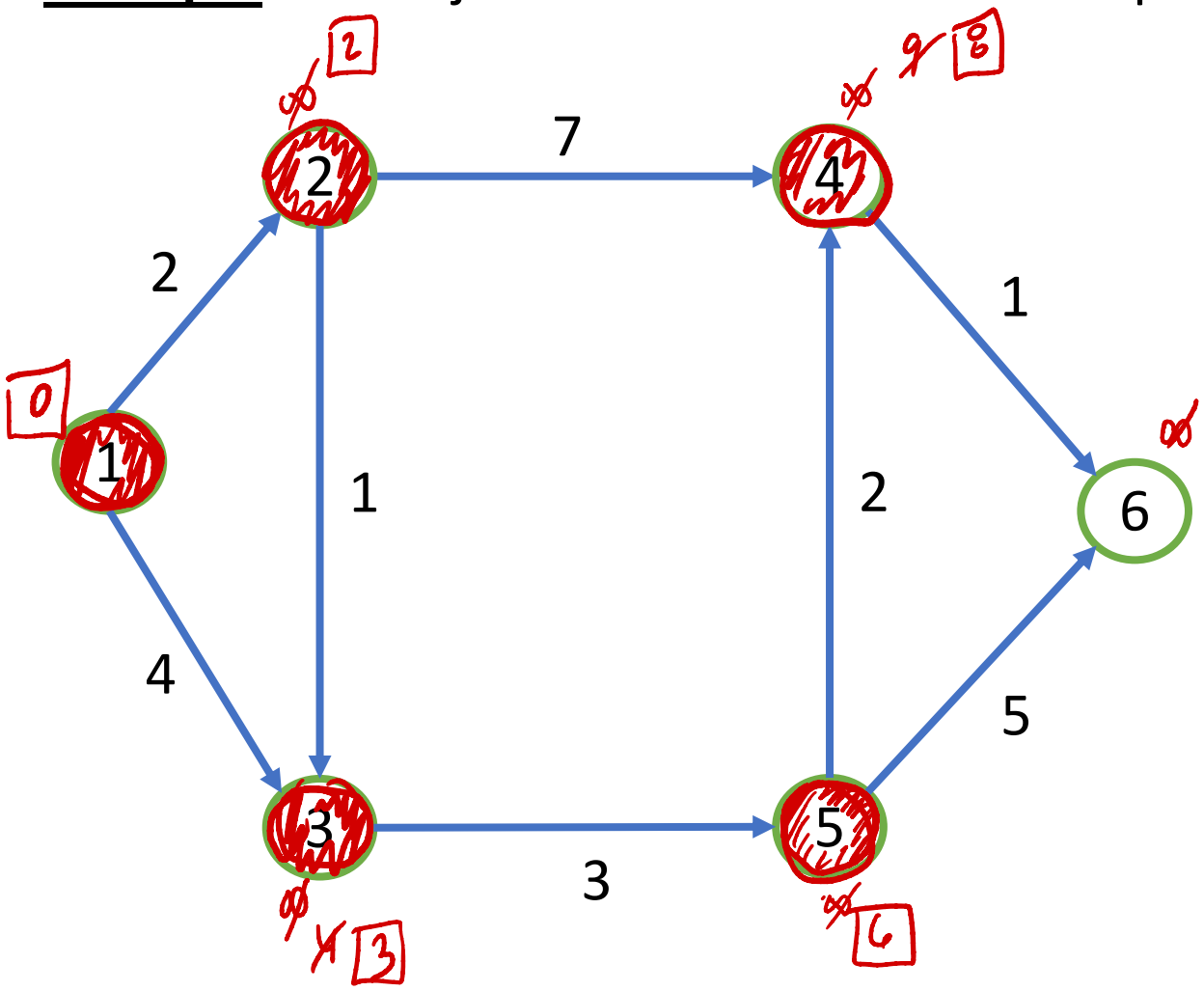
$d(3) + c(3,5) = 3 + 3 = 6 < \infty$

$d(5) = 6$

Wikipedia Example

Dijkstra's Algorithm

Example: Use Dijkstra's to find the shortest path to each node.



expand (5) neighbors 4, 6

$d(5) + c(5,4) = 6 + 2 = 8 < 9$

$d(4) = 8$

$d(5) + c(5,6) = 6 + 5 = 11 < \infty$

$d(6) = 11$

expand (4) neighbor 6

$d(4) + c(4,6) = 8 + 1 = 9 < 11$

$d(6) = 9$

expand (6)

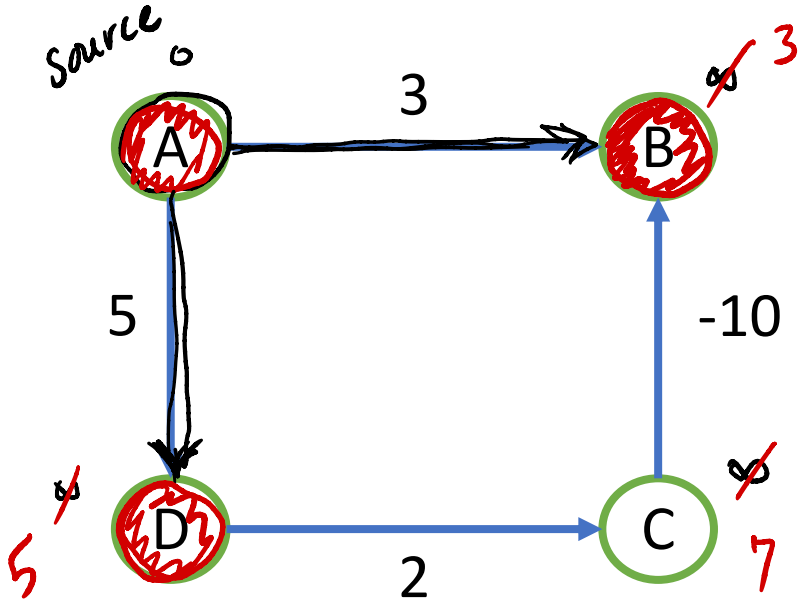
$d(6) = 9$

[Wikipedia Example]

Dijkstra's Algorithm

1. Not necessarily correct with negative edge length
2. If the weights change, we need to re-run the whole thing.

Example: Dijkstra's with negative edge weights.



Correct shortest path : A → D → C → B

length = -3

Dijkstras : A → B

length = 3

* This version of Dijkstra's fails to find the shortest path
* Assumes path can only get longer when considering more edges.

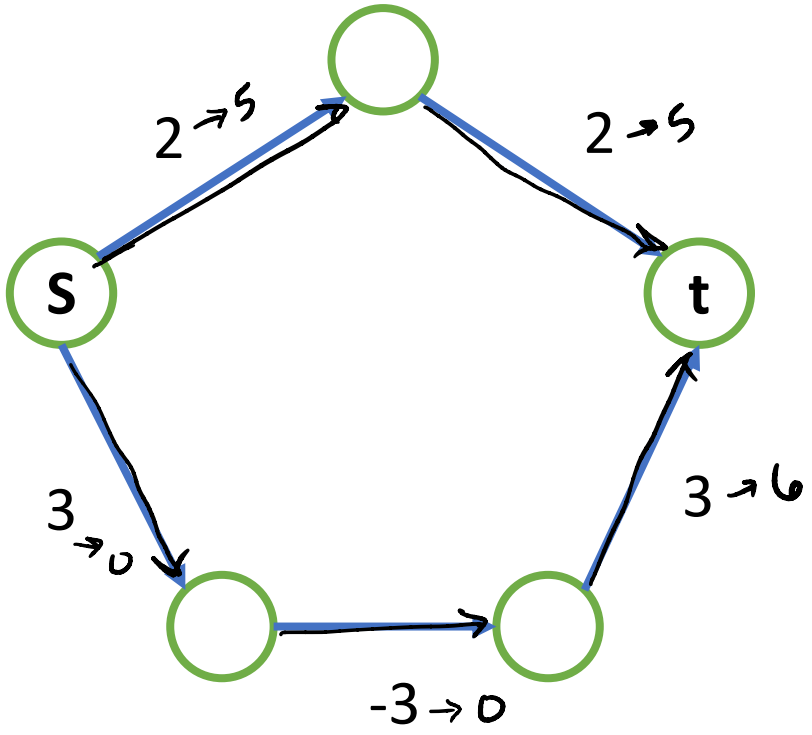
Abdul Bari Examples

Dijkstra's Algorithm

1. Not correct with negative edge length
2. If the weights change, we need to re-run the whole thing.

In the case of negative edge weights, what if we try adding weight to each edge?

$w'(e) = w(e) + 3$



shortest path: s → o → o → t length = 3

now shortest path: (s) → o → (t) length = 10

- This doesn't work - can't get correct shortest path from original graph
- Not taking into account number of edges.

Bellman–Ford Algorithm

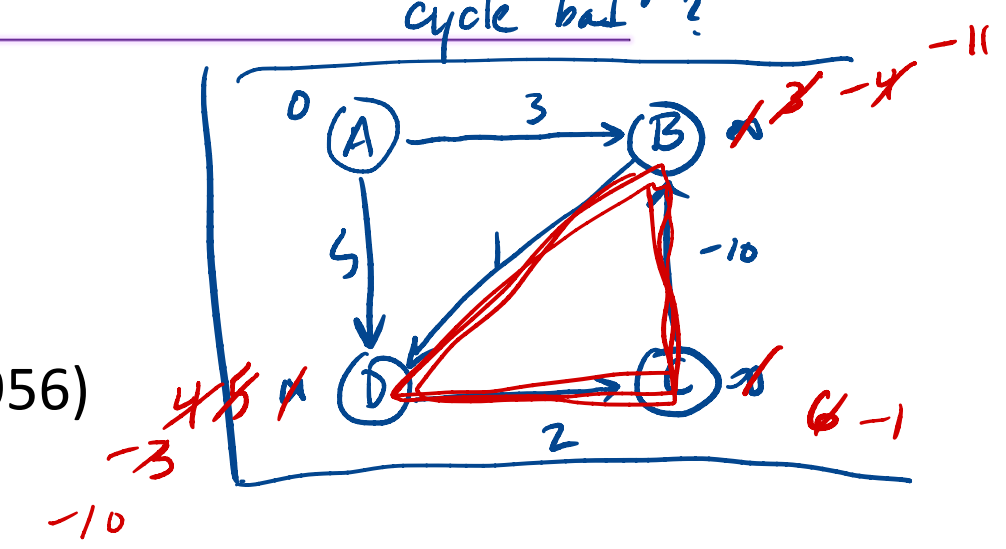
- Adding weights to each edge doesn't work.

How do we solve the SSSP problem when there are negative edges?

How do we detect negative cycles?

Bellman–Ford Algorithm

Why is a negative cycle bad?



History of the Bellman-Ford Algorithm:

- First proposed by Alfonso Shimbel in 1955
- Published separately by Bellman (1958) and Ford (1956)

- Dijkstra - published in 1959

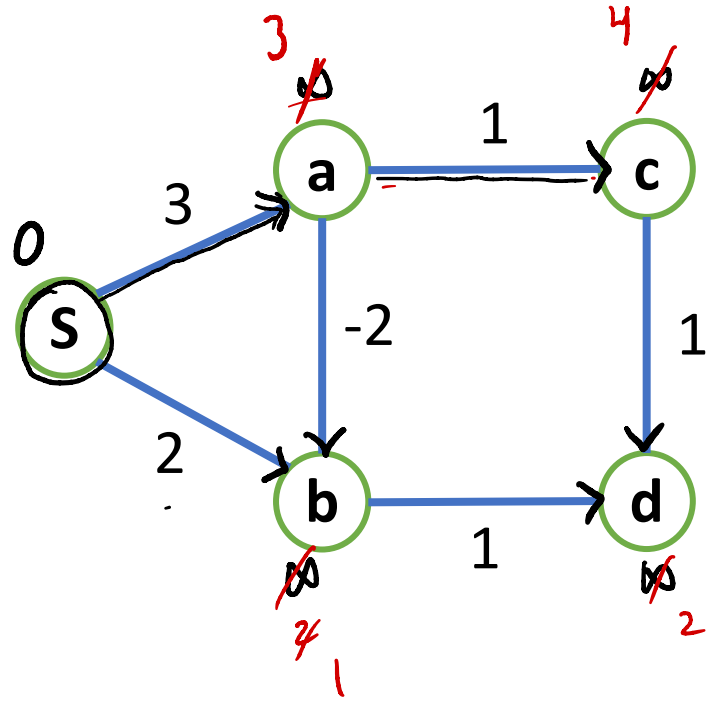
Main idea: Any simple path from source to another vertex is going to have a maximum of $n-1$ edges (n = number of nodes.)

- Run $|v|-1$ times – Relaxing each edge $|V|-1$ times
- by the end of $|v|-1$ relaxations, all vertices have the correct shortest path distance
- Can detect & report negative cycles.

Bellman–Ford Algorithm

- some edge orderings allow for finding shortest path faster
  but we always iterate |V| − 1 times and end with shortest paths.

Example: Use the Bellman-Ford Algorithm to find the shortest length paths from s to each node.



make an edge list – any order of edges is fine

(s,a), (a,c), (s,b), (a,b), (b,d), (c,d)

S	0
a	∞, 3
b	∞, 2, 1
c	∞, 4
d	∞, 2

- we iterate through the edge list a total of 5 − 1 = 4 times looking for any further relaxation

Bellman–Ford Algorithm

$\Theta(|V| \cdot |E|)$

BELLMAN-FORD(Graph, weights, source):

$initialize( )$

→ for $i = 1$ to $|V| - 1$:

  for each edge $(u, v) \in E$:

    relax (u, v, w)

$|V| \cdot |E|$

for each edge $(u, v) \in E$:

  if $d[v] > d[v] + w(u, v)$:

    report that a negative cycle exists

$|E|$

Complexity: V*E - first nested for loop

second for loop is O(E)

- substantially slower than dijkstras in practice
- use Dijkstra when you can, when forced to due to negative weights, change to bellman ford

- we've checked all the $|V| - 1$ number of edges in a path

- if we do an update here ⇒ we've visited a node more than once
  ⇒ cycle
  ⇒ negative cycle

Bellman–Ford Algorithm

Algorithm 7 The Real Bellman–Ford algorithm.

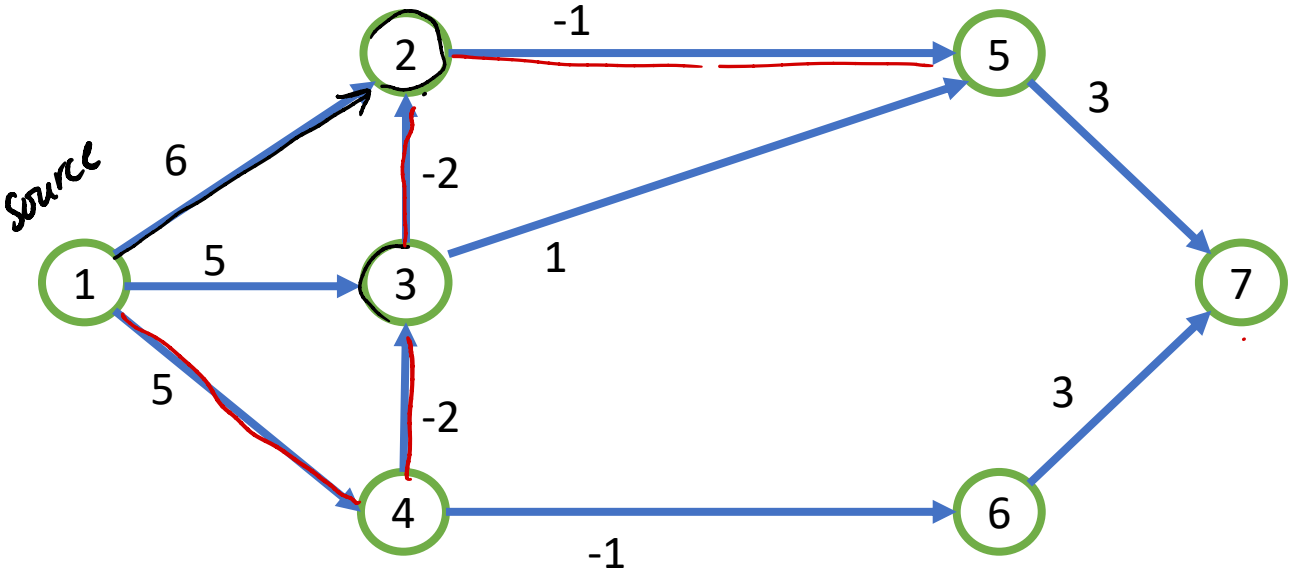
```
procedure BellmanFord(G = (V, E, w), r)
    let P be an array of length n containing the element NULL
    let D be an array of length n containing the element ∞
    D[r] = 0

    for i in 1..|V| − 1 :                          #i.e. repeat |V| − 1 times
        for every edge (u, v) in E :
            if D[v] > D[u] + w(u, v) :              #if (u, v) is tense
                P[v] = u                            #relax (u, v) by updating P
                D[v] = D[u] + w(u, v)               #and by updating D
    for every edge (u, v) in E :
        if D[v] > D[u] + w(u, v) :                  #if (u, v) is still tense
            throw "Negative cycle!"
```

negative cycle check

Bellman–Ford Algorithm

Example: The Bellman-Ford algorithm can find the shortest paths of this graph by iteratively relaxing all edges. Given the order of edges below, show all of the updates that Bellman-Ford would make to the cost of each vertex in the graph.

edge list: (1,2), (2,5), (5,7), (3,2), (3,5), (4,3), (4,6), (6,7), (1,3), (1,4)



1	0				
2	∞, 6	3	1		
3	∞, 5	3			
4	∞, 5				
5	∞, 5		2	0	
6	∞,	4			
7	∞, 8	7	5	3	

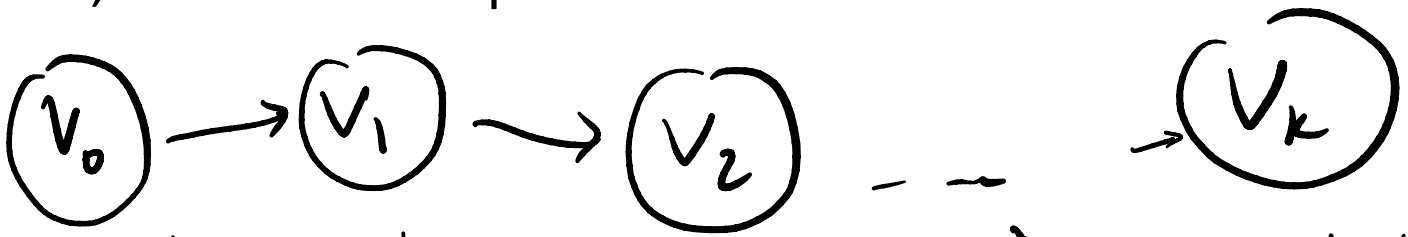
no more updates

d(1) = 0
d(2) = 1
d(3) = 3
d(4) = 5
d(5) = 0
d(6) = 4
d(7) = 3

[Abdul Bari Video]

Bellman–Ford Algorithm

Theorem: If $G = (V, E)$ contains no negative weight cycles, then after Bellman-Ford executes, the minimum path for all $v \in V$ will have been found.



How many edges in this path (worst case) : $= |V| - 1$

- If your path is longer, you've visited a vertex more than once
  - $\Rightarrow$ not a simple path
  - $\Rightarrow$ $k \leq |V| - 1$

else cycle

let $v \in V$. Let there be a path from s to v

This path is assumed to be the shortest path with the min. number of edges.

After one pass of B-F $d(v_1) =$ min. path (s, v_1)

After two passes of B-F $d(v_2) =$ min. path (s, v_2)

If (v_0, v_1) is not shortest path, we've violated optimal substructure property

Bellman–Ford Algorithm

Theorem: If $G = (V, E)$ contains no negative weight cycles, then after Bellman-Ford executes, the minimum path for all $v \in V$ will have been found.

After K passes, $d(v_k)$ = min. path s to v_k

After $|V|-1$, all reachable vertices from the source have their min. path weights.

Bellman–Ford Algorithm

Corollary: If a value $d(v)$ fails to converge after $|V| - 1$ passes, then there exists a negative weight cycle.

Suppose after $|V|-1$ passes we find an edge that can be relaxed

$\Rightarrow$ not a simple path

$\Rightarrow$ Found a cycle

$\Rightarrow$ must be a negative weight cycle.