CS-204: Design and Analysis of Algorithms

220001032, 220001034, 220001036

February 13, 2024

1 Bellmann-Ford Algorithm

The Bellman-Ford algorithm is preferred over Dijkstras algorithm in scenarios where negative edge weights or cycles exist in the graph. Bellman-Ford can handle negative weights and detect negative cycles, features that Dijkstra lacks due to its reliance on a greedy choice, which is not always suitable in the presence of negativity.

Given a weighted, directed graph $G = (V, E)$ with a source vertex s and weight function $w : E \to \mathbb{R}$, the Bellman-Ford algorithm returns a boolean value indicating the presence of a reachable negative-weight cycle from the source. If such a cycle exists, the algorithm signals that no solution is possible. Conversely, if there is no negative-weight cycle, the algorithm computes the shortest paths and their corresponding weights.

1.1 Working Principles

It works on two principles:

1. Shortest path from one node to another doesn't contain any cycle.

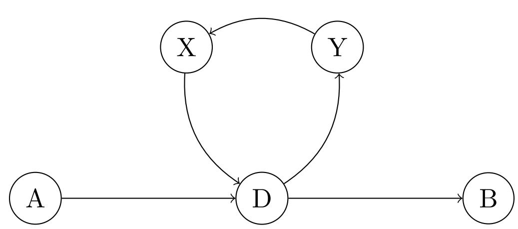
Proof: The claim is that the shortest path from one node to another doesn't contain any cycles.

Assume, for the sake of contradiction, that there exists a shortest path P from node A to node B that contains a cycle $(D \to Y \to X \to D)$ of cost C. Let $P = P_1 + C + P_2$, where P_1 is the subpath from A to a node D, and P_2 is the subpath from node D to B.

Since shortest path is undefined for negative cycles because the presence of a negative cycle in a graph allows for infinite loops, continually decreasing the path length without a well-defined minimum.

Therefore, $(D \to Y \to X \to D)$ is a cycle of +ve cost, we can eliminate it from the path without changing the start and end nodes. Therefore, $P = P_1 + P_2$, where P_1 and P_2 represent two non-overlapping subpaths from A to D and from D to B within the original path P.

Hence, the assumption that the shortest path between A and B contains a cycle is incorrect, and we conclude that the shortest path from one node to another doesn't contain any cycles.



Corollary: *There exists at-most (n-1) edges in the shortest path between two nodes.*

2. If $s \to t$ represents shortest path from s to t then $s \to v_i$ for any v_i in path from s to t also produces shortest path.

Proof: Let p, p_1 and p_2 be shortest path from s to t, subpath of p from s to v_i and subpath of p from v_i to t respectively.

$\therefore p =$p$_1$ + p_2.

Let p_1' be a shortest path from s to v_i and p' be a path from s to t through p_1' , therefore, p'=p_1' + p_2. But p is shortest path from s to t, therefore p$\leq p'$,

$\implies$ p$_1$+p_2 $\leq$p'$_1$ + p_2

$\implies$ p$_1$ $\leq$p'$_1$

But p_1' is shortest path from s to v_i, therefore, $p_1 = p_1'$.

Hence proved.

1.2 Algorithm

Algorithm 1 Bellman-Ford

Input: Graph G with vertices V and edges E, source vertex s and weight function w
Output: Shortest path from given vertex to all other vertices.
Procedure: BELLMAN-FORD(G, w, s)
for each vertex $v \in$ G.V
    $v.d$=∞
    $v.\pi$=NIL
for i = 1 **to** $|G.V|$ - 1
    for each edge $(u, v) \in G.E$
        if $v.d > u.d + w(u, v)$
            $v.d = u.d + w(u, v)$
            $v.\pi = u$
// To check if -ve cost cycle exists
for each edge $(u, v) \in G.E$
    if $v.d > u.d + w(u, v)$
        return FALSE
return TRUE
=0

1.3 Time and Space Complexity Analysis

The **Bellman-Ford** algorithm has a time complexity of **O(V*E)**, where V is the number of vertices and E is the number of edges in the graph. In the worst-case scenario, the algorithm needs to iterate through all edges for each vertex, resulting in this time complexity. The space complexity of the Bellman-Ford algorithm is O(V), where V is the number of vertices in the graph. This space complexity is mainly due to storing the distances from the source vertex to all other vertices in the graph.

Adjacency List

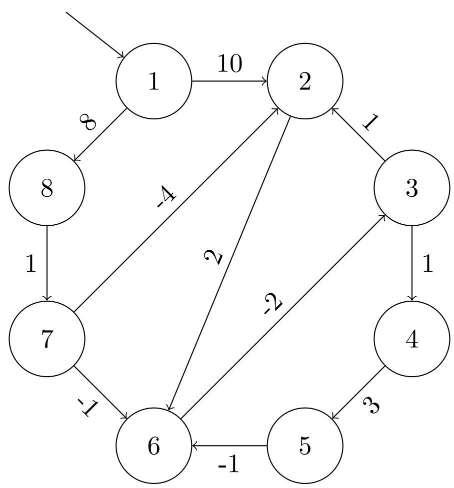
Operation	Time Complexity	Space Complexity
Initialization	O(V)	O(V)
Relaxation	O(V*E)	O(1)
Overall	O(V*E)	O(V)

Adjacency Matrix

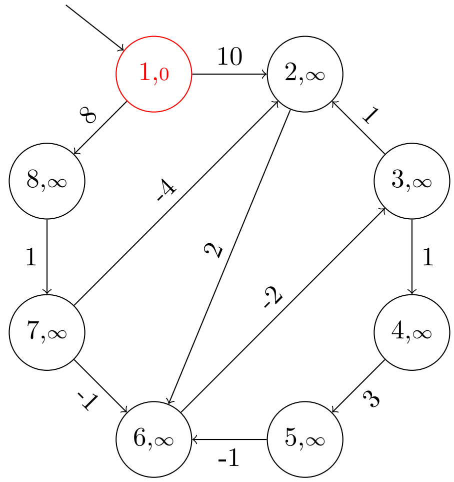
Operation	Time Complexity	Space Complexity
Initialization	O(V)	O(V)
Relaxation	O(V^3)	O(1)
Overall	O(V^3)	O(V)

1.4 Example

Lets suppose we have a graph which is given below and we want to find shortest distance from the sourcenode to every node.

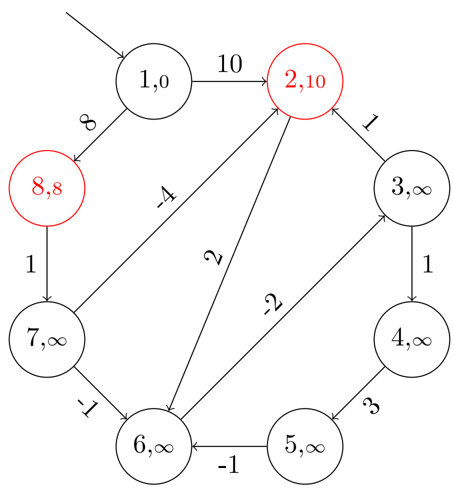


Step 1: Initialize a distance array Dist[] to store the shortest distance for each vertex from the source vertex. Initially distance of source will be 0 and Distance of other vertices will be INFINITY.



1	2	3	4	5	6	7	8
0	∞	∞	∞	∞	∞	∞	∞

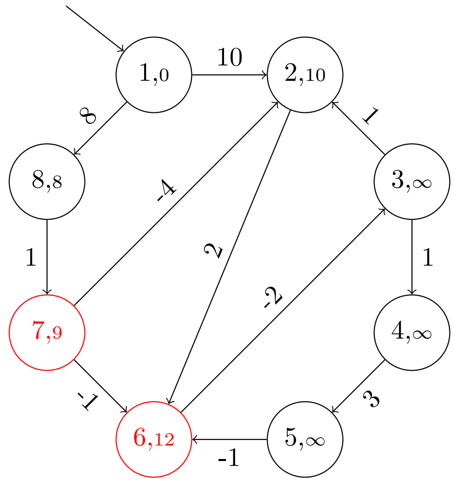
Step 2: Start relaxing the edges, during 1st Relaxation:
Current Distance of 2 > (Distance of 1) + (Weight of 1 to 2) i.e. Infinity > 0 + 10.
$\therefore$ Dist[2] = 10
Current Distance of 8 > (Distance of 1) + (Weight of 1 to 8) i.e. Infinity > 0 + 8.
$\therefore$ Dist[8] = 8



1	2	3	4	5	6	7	8
0	10	∞	∞	∞	∞	∞	8

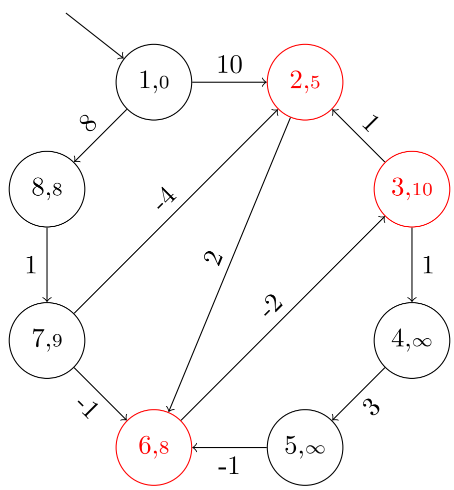
Step 3: During 2nd Relaxation:
Current Distance of 7 > (Distance of 8) + (Weight of 8 to 7) i.e. Infinity > 8 + 1

$\therefore$ Dist[7] = 9
Current Distance of 6 > (Distance of 2) + (Weight of 2 to 6) i.e. Infinity > 10 + 2
$\therefore$ Dist[6] = 12



1	2	3	4	5	6	7	8
0	10	∞	∞	∞	12	9	8

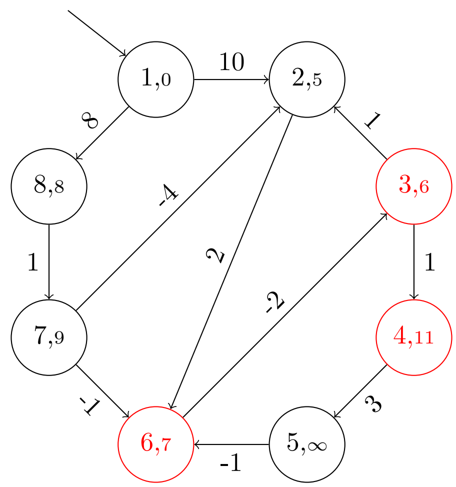
Step 4: During 3rd Relaxation:
Current Distance of 2 > (Distance of 7) + (Weight of 7 to 2) i.e. Infinity > 9 + -4
$\therefore$ Dist[2] = 5
Current Distance of 3 > (Distance of 6) + (Weight of 6 to 3) i.e. Infinity > 12 + -2
$\therefore$ Dist[3] = 10
Current Distance of 6 > (Distance of 7) + (Weight of 7 to 6) i.e. Infinity > 9 + -1
$\therefore$ Dist[6] = 8



1	2	3	4	5	6	7	8
0	5	10	∞	∞	8	9	8

Step 5: During 4th Relaxation:

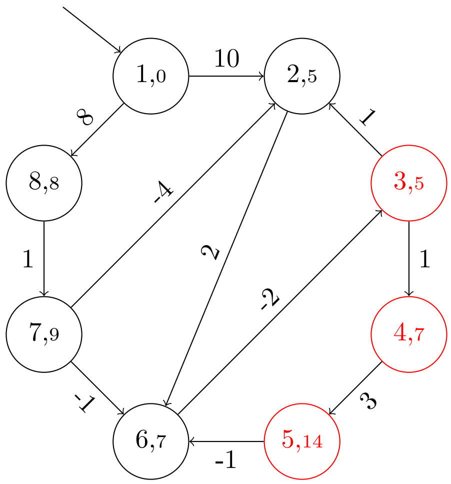
Current Distance of 3 > (Distance of 6) + (Weight of 6 to 3) i.e. Infinity > 8 + -2
$\therefore$ Dist[3] = 6
Current Distance of 4 > (Distance of 3) + (Weight of 3 to 4) i.e. Infinity > 10 + 1
$\therefore$ Dist[4] = 11
Current Distance of 6 > (Distance of 7) + (Weight of 7 to 2) + (Weight of 2 to 6) i.e.
Infinity > 9 + -4 + 2 $\therefore$ Dist[6] = 7



1	2	3	4	5	6	7	8
0	5	6	11	∞	7	9	8

Step 6: During 5th Relaxation:

Current Distance of 3 > (Distance of 6) + (Weight of 6 to 3) i.e. Infinity > 7 + -2
$\therefore$ Dist[3] = 5
Current Distance of 4 > (Distance of 3) + (Weight of 3 to 4) i.e. Infinity > 6 + 1
$\therefore$ Dist[4] = 7
Current Distance of 5 > (Distance of 4) + (Weight of 4 to 5) i.e. Infinity > 11 + 3
$\therefore$ Dist[5] = 14



1	2	3	4	5	6	7	8
0	5	5	7	14	7	9	8

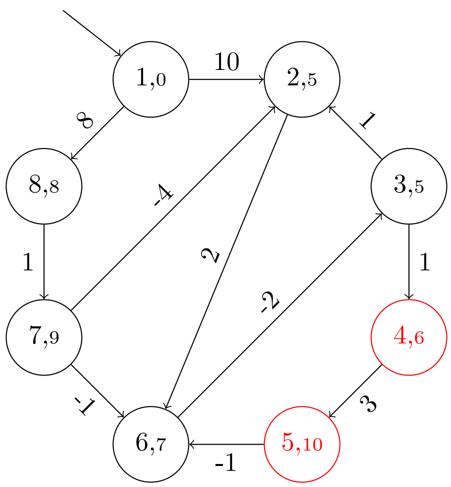
Step 7: During 6th Relaxation:

Current Distance of 4 > (Distance of 3) + (Weight of 3 to 4) i.e. Infinity > 5 + 1

$\therefore$ Dist[4] = 6

Current Distance of 5 > (Distance of 4) + (Weight of 4 to 5) i.e. Infinity > 7 + 3

$\therefore$ Dist[5] = 10

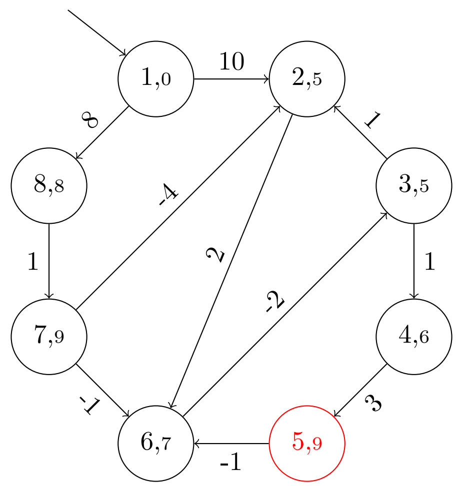


1	2	3	4	5	6	7	8
0	5	5	6	10	7	9	8

Step 8: Now the final relaxation i.e 7th relaxation:

Current Distance of 5 > (Distance of 4) + (Weight of 4 to 5) i.e. Infinity > 6 + 3

$\therefore$ Dist[5] = 9



Therefore, after 7(n-1) relaxations we got the final shortest distances from the considered source vertex.

1	2	3	4	5	6	7	8
0	5	5	6	9	7	9	8

Let $\delta(u, v)$ denote the shortest path weight from u to v.

Lemma 1:

Consider a weighted, directed graph $G = (V, E)$ with a source vertex s and weight function $w : E \to \mathbb{R}$. Assuming G contains no negative-weight cycles reachable from s, after the execution of the **for** loop (lines 7 to 11) in the BELLMAN-FORD algorithm for $|V| - 1$ iterations, we have $v.d = \delta(s, v)$ for all vertices v that are reachable from s.

Proof: We prove the lemma by appealing to the path-relaxation property. Consider any vertex v that is reachable from s, and let $p = \{v_0, v_1, \ldots, v_k\}$, where $v_0 = s$ and $v_k = v$, be any shortest path from s to v. Because shortest paths are simple, p has at most $|V| - 1$ edges, and so $k \leq |V| - 1$. Each of the $|V| - 1$ iterations of the **for** loop of lines 7 to 11 relaxes all $|E|$ edges. Among the edges relaxed in the ith iteration, for $i = 1, 2, \ldots, k$, is (v_{i-1}, v_i). By the path-relaxation property, therefore,

$$v.d = v_k.d = \delta(s, v_k) = \delta(s, v).$$

Lemma 2:

For a weighted, directed graph $G = (V, E)$ with a source vertex s and weight function $w : E \to \mathbb{R}$, the BELLMAN-FORD algorithm terminates with $v.d < \infty$ for each vertex $v \in V$ if and only if there exists a path from s to v.

1.5 Correctness of the Bellman-Ford algorithm

Let **BELLMAN-FORD** be run on a weighted, directed graph $G = (V, E)$ with source s and weight function $w : E \to \mathbb{R}$. If G contains no negative-weight cycles that are reachable from s, then the algorithm returns TRUE, where $v.d = \delta(s, v)$ for all vertices $v \in V$, and the predecessor sub-graph G_π is a shortest-paths tree rooted at s. If G contains a negative-weight cycle reachable from s, then the algorithm returns FALSE.

Proof: Suppose that graph G contains no negative-weight cycles that are reachable from the source s. We first prove the claim that at termination, $v.d = \delta(s, v)$ for all vertices $v \in V$. If vertex v is reachable from s, then Lemma 1 proves this claim. If v is not reachable from s, then the claim follows from the no-path property. Thus, the claim is proven. The predecessor-subgraph property, along with the claim, implies that G_π is a shortest-paths tree. Now we use the claim to show that BELLMAN-FORD returns TRUE. At termination, for all edges $(u, v) \in E$ we have

$$v.d = \delta(s, v) \leq \delta(s, u) + w(u, v) = u.d + w(u, v),$$

and so none of the tests in line 14 causes BELLMAN-FORD to return FALSE. Therefore, it returns TRUE.

Now, suppose that graph G contains a negative-weight cycle reachable from the source s. Let this cycle be $c = \{v_0, v_1, \ldots, v_k\}$, where $v_0 = v_k$, in which case we have

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0 \quad \text{— (1)}$$

Assume for the purpose of contradiction that the Bellman-Ford algorithm returns TRUE. Thus, $v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$ for $i = 1, 2, \ldots, k$. Summing the inequalities around cycle c gives

$$\sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} (v_{i-1}.d + w(v_{i-1}, v_i)) = \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Since $v_0 = v_k$, each vertex in c appears exactly once in each of the summations. $\sum_{i=1}^{k} v_{i-1}.d = \sum_{i=1}^{k} v_i.d$.

Moreover, by Lemma 2, $v_{i-1}.d$ is finite for $i = 1, 2, \ldots, k$. Thus, $0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i)$, which contradicts inequality (1). We conclude that the Bellman-Ford algorithm returns TRUE if graph G contains no negative-weight cycles reachable from the source, and FALSE otherwise.