Data Structures and Algorithms

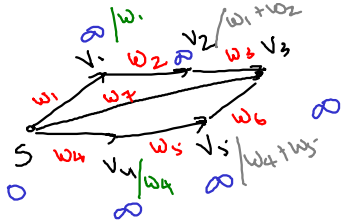
Prof. Ganesh Ramakrishnan,
Prof. Ajit Diwan,
Prof. D.B. Phatak

Department of Computer Science and Engineering
IIT Bombay

Session: Shortest Path Algorithm
(Bellman-Ford Algorithm)

Bellman-Ford Algorithm

1. Computes shortest paths from a source vertex to all other vertices in a weighted directed graph.
2. Dijkstra's Algorithm does not work for negative edges.
3. Solves the problem of negative edge weights but is slower than Dijkstra's Algorithm.
4. Running time: O(VE)



# of times to iterate over all edges
= width of graph
= max # hops from S to any vertex (V_3)
$\leq |V|-1$

$\forall$ each edge e

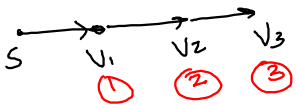
$e: u \to v$

if $(d[v] \geq d[u] + w(u,v))$

$d[v] = d[u] + w(u,v)$

$d[u]$: SP to u

$w(u,v)$: append (u,v) to SP to u



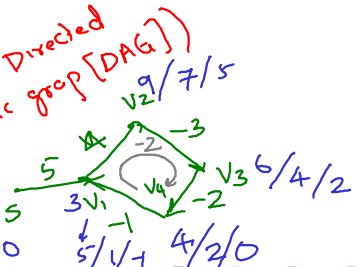
Bellman-Ford Shortest Path Algorithm

Algorithm ComputeBellman-FordSPs(G, s, w)
Output: Returns false for negative-weight cycle, else produces shortest paths with predecessor.

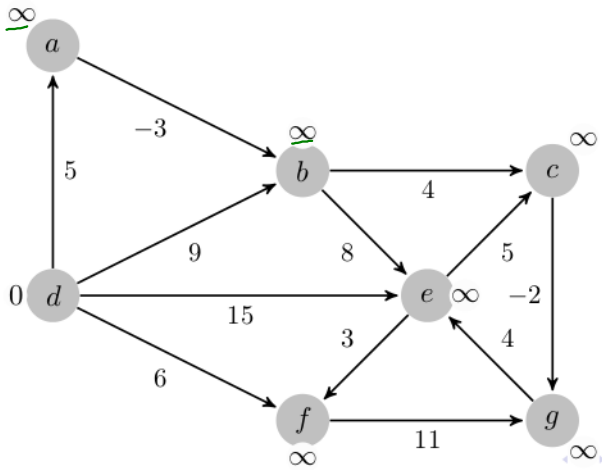
```
for v ∈ G.getVertices() do
    if v = s then
        d[v] = 0
        predecessor[v] = NULL
    else
        d[v] = ∞
    end if
end for
for i ∈ {1, ..., G.getVertices() − 1} do
    for e ∈ G.edges() do
        if d[u] + w < d[v] then
            d[v] = d[u] + w
            predecessor[v] = u
        end if
    end for
end for
for e ∈ G.edges() do
    if d[u] + w < d[v] then
        return FALSE
    end if
end for
return d[], predecessor[]
```

→ width of graph in worst case

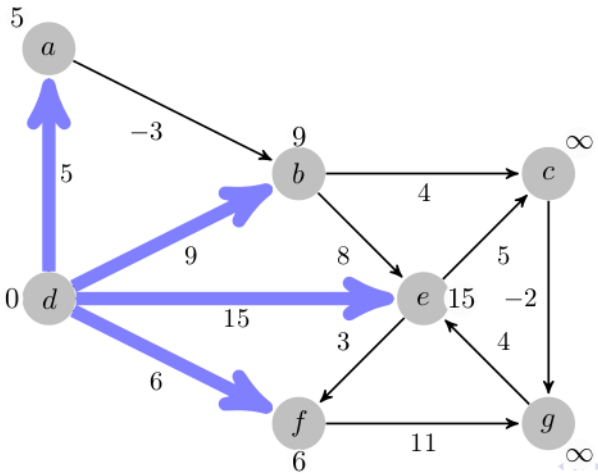
what if the graph has cycles (it is not a Directed acyclic graph [DAG])



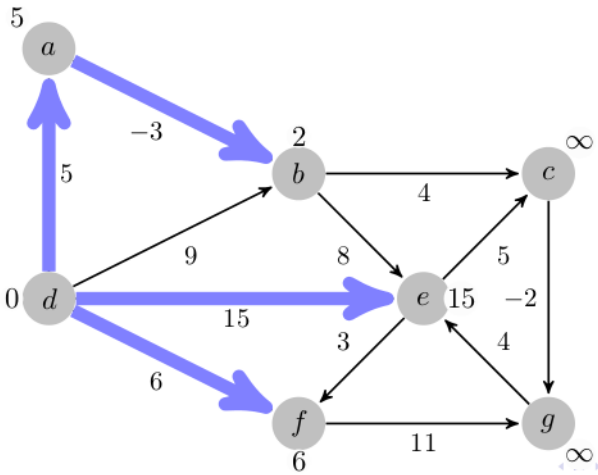
Bellman-Ford



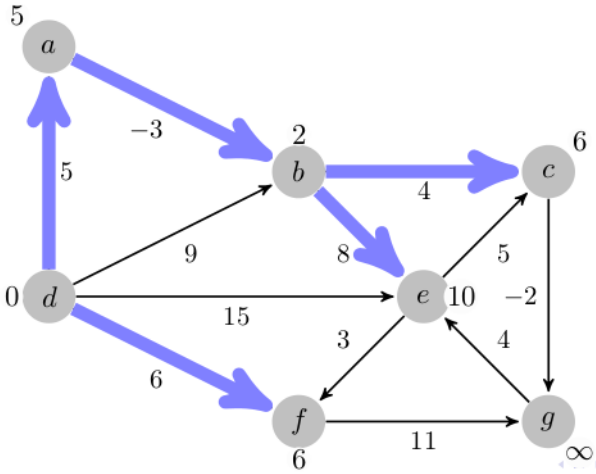
Bellman-Ford



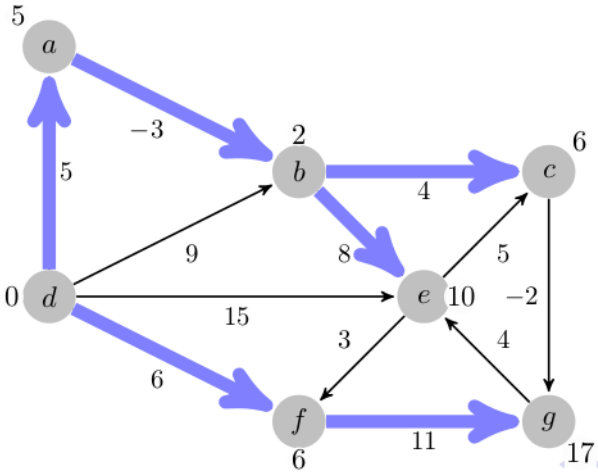
Bellman-Ford



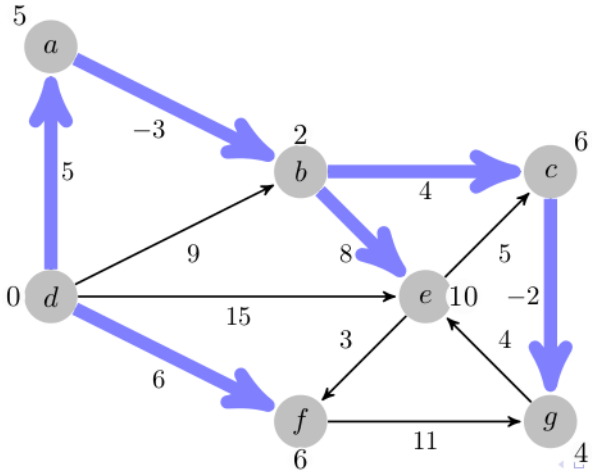
Bellman-Ford



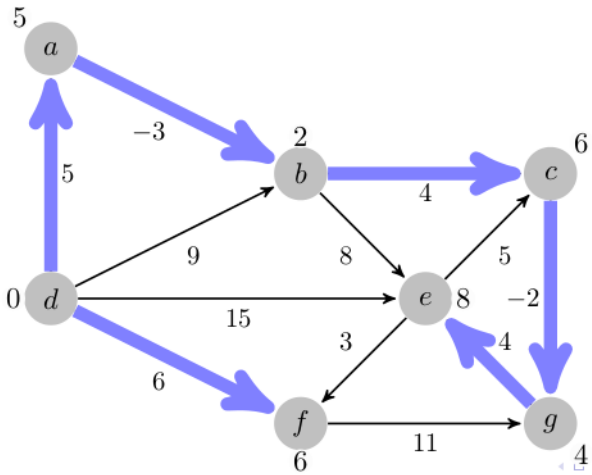
Bellman-Ford



Bellman-Ford



Bellman-Ford



Analysis of Bellman-Ford Shortest Path Algorithm

Algorithm ComputeBellman-FordSPs(G, s, w)
Output: Returns false for negative-weight cycle, else produces shortest paths with predecessor.

```
for v ∈ G.getVertices() do
    if v = s then
        d[v] = 0
        predecessor[v] = NULL
    else
        d[v] = ∞
    end if
end for          ⟹   c1   × |V| times
for i ∈ {1, ..., G.getVertices() − 1} do   [k^th iteration]
    for e ∈ G.edges() do
        if d[u] + w < d[v] then
            d[v] = d[u] + w
            predecessor[v] = u
        end if   ⟹   c2   × |1| times
    end for      ⟹   c3   × |E| times
end for          ⟹   c4   × |V| times
for e ∈ G.edges() do
    if d[u] + w < d[v] then
        return FALSE
    end if
end for          ⟹   c5   × |E| times
return d[], predecessor[]
```

|V|

|E|

Loop invariant:
For every vertex v within k hops from "s", the value d[v] is indeed the shortest path from s after k outer iterations

Maintainance: Key observation that a vertex at k+1 hops from s is reached through vertex at k hops from s

$$T(n) = c_1|V| + c_2c_3c_4|V||E| + c_5|E| = O(|V||E|)$$

Thank you