Dynamic Programming
Unit 4

Session Overview

- Greedy Strategy vs Dynamic Programming
- Principal of Optimality
- The Bellman-Ford algorithm
- Time Complexity of Bellman-Ford

Greedy v/s Dynamic Programming

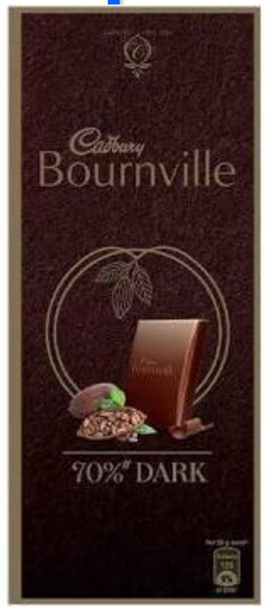
- We make whatever choice seems best at the moment in the hope that it will lead to global optimal solution.
- Sometimes there is no such guarantee of getting Optimal Solution.
- It follows the problem solving heuristic of making the locally optimal choice at each stage.
- Never look back or revise previous choices, so it is faster.

- We make decision at each step considering current problem and solution to previously solved sub problem to calculate optimal solution .
- Its guaranteed that Dynamic Programming will generate an optimal solution.
- It is an algorithmic technique which is usually based on a recurrent formula that uses some previously calculated states.
- It is generally slower as it works in iterations and looks back at previous decisions.

You have a budget of Rs.300 and box of capacity 275g

Rs.60 wt of each is 28g

Rs. 95 178g



Rs. 270 100g

Rs. 175 100g

Cadbury Dairy Milk Silk 150 G...

₹204

Principal of Optimality

- **A problem is said to satisfy the Principle of Optimality if the subsolutions of an optimal solution of the problem are themesleves optimal solutions for their subproblems.**

Shortest Path Problem

- Weighted path length (cost): The sum of the weights of all links on the path.

- The single-source shortest path problem: Given a weighted graph G and a source vertex s, find the shortest (minimum cost) path from s to every other vertex in G.

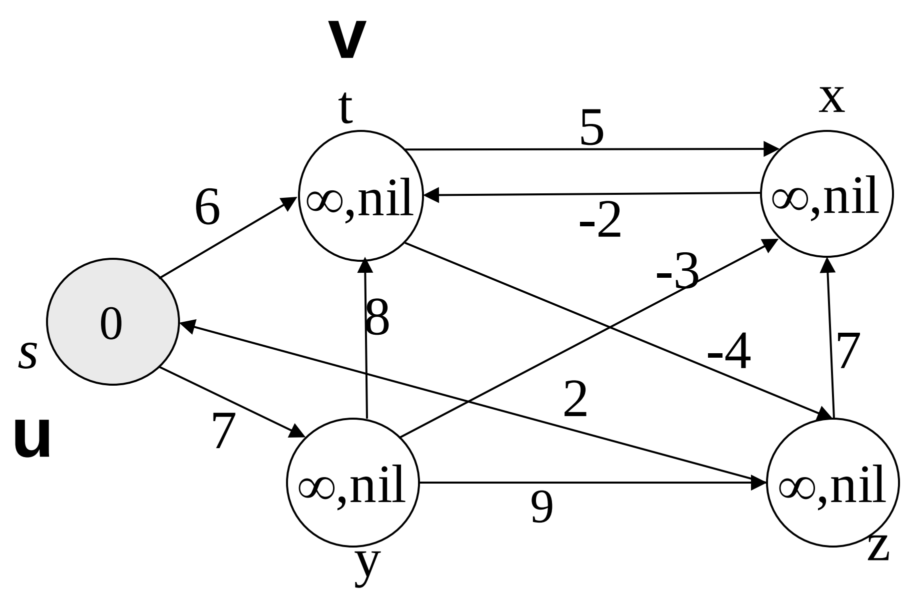
Differences

- Negative link weight: The Bellman-Ford algorithm works; Dijkstra's algorithm doesn't.

- Distributed implementation: The Bellman-Ford algorithm can be easily implemented in a distributed way. Dijkstra's algorithm cannot.

- Time complexity: The Bellman-Ford algorithm is higher than Dijkstra's algorithm.

The Bellman-Ford Algorithm



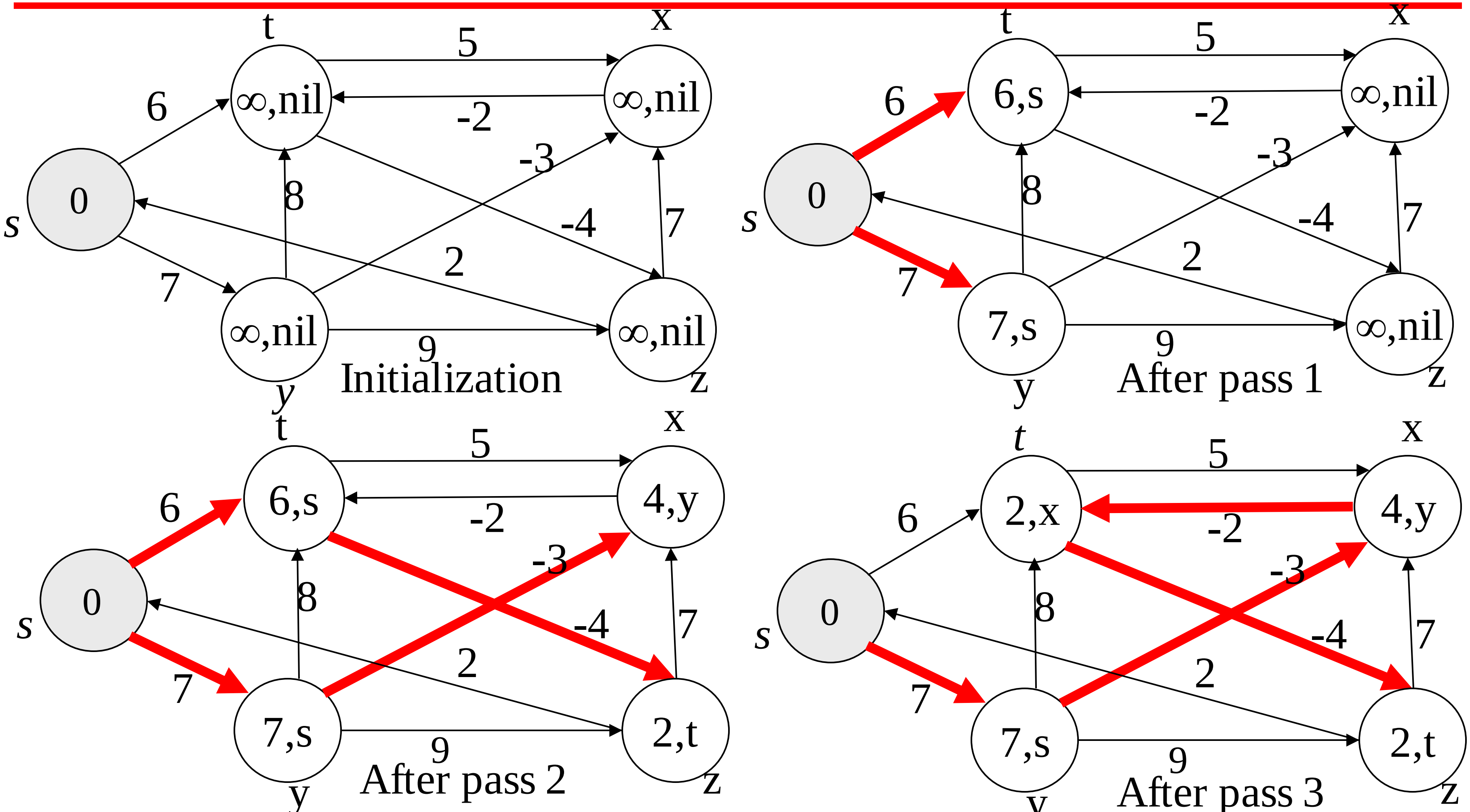
Relax(u, v, w)

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

then d[v] := d[u] + w(u, v)

parent[v] := u

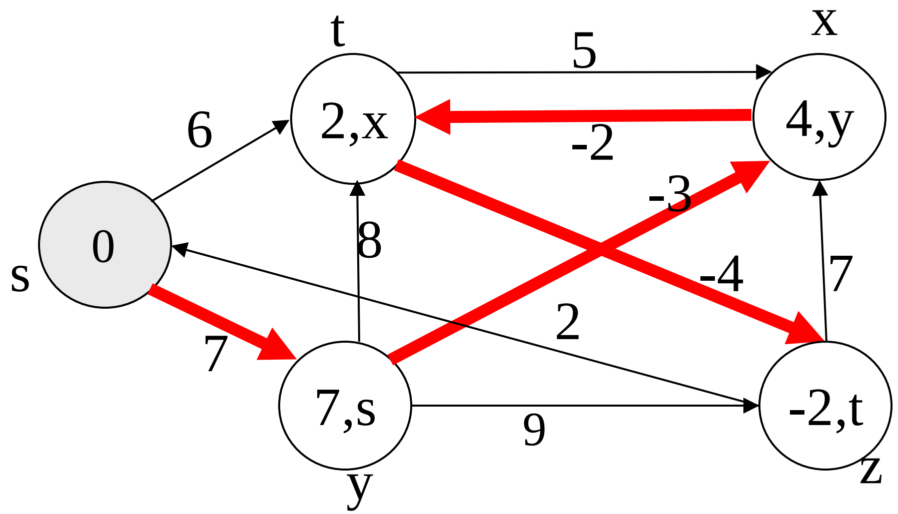
The Bellman-Ford Algorithm



The order of edges examined in each pass:

(t, x), (t, z), (x, t), (y, x), (y, t), (y, z), (z, x), (z, s), (s, t), (s, y)

The Bellman-Ford Algorithm



After pass 4

The order of edges examined in each pass:

(t, x), (t, z), (x, t), (y, x), (y, t), (y, z), (z, x), (z, s), (s, t), (s, y)

The Bellman-Ford Algorithm

```
Bellman-Ford(G, w, s)
    Initialize-Single-Source(G, s)
    for i := 1 to |V| - 1 do
        for each edge (u, v) ∈ E do
            Relax(u, v, w)
    for each vertex v ∈ u.adj do
        if d[v] > d[u] + w(u, v)
            then return False  // there is a negative cycle
    return True
```

```
Relax(u, v, w)
if d[v] > d[u] + w(u, v)
   then  d[v] := d[u] + w(u, v)
         parent[v] := u
```

Time Complexity

Bellman-Ford(G, w, s)

```
Initialize-Single-Source(G, s)                      → O(|V|)
for i := 1 to |V| - 1 do
    for each edge (u, v) ∈ E do                     → O(|V||E|)
        Relax(u, v, w)
for each vertex v ∈ u.adj do                        → O(|E|)
    if d[v] > d[u] + w(u, v)
        then return False  // there is a negative cycle
return True
```

Time complexity: $O(|V||E|)$