Third Year B. Tech., Sem V 2022-23

Design and Analysis of Algorithm Lab

Assignment / Journal submission

PRN/ Roll No: 21520010

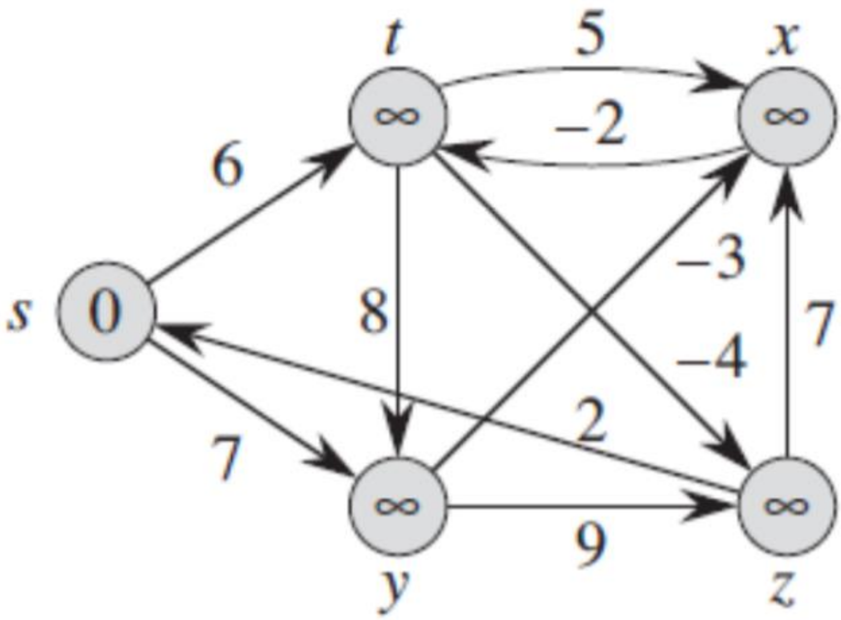
Full name: Mayur Sunil Savale

Batch: T8

Assignment: Week 9

Title of assignment: Dynamic Programming

1. From a given vertex in a weighted connected graph, implement shortest path finding Bellman-Ford algorithm.



a) Algorithm: (Pseudocode)

Input: Graph and a source vertex src

Output: Shortest distance to all vertices from src. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

- This step initializes distances from the source to all vertices as infinite and distance to the source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.
- This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.
  …..a) Do following for each edge u-v
  ………………If dist[v] > dist[u] + weight of edge uv, then update dist[v]
  ………………….dist[v] = dist[u] + weight of edge uv
- This step reports if there is a negative weight cycle in graph. Do following for each edge u-v
  ……If dist[v] > dist[u] + weight of edge uv, then “Graph contains negative weight cycle”

b) Code snapshots of implementation

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
   ios_base::sync_with_stdio(0);
   using node = tuple<int,int,int>;
   vector<node> edges;
   int n,m;
   cout<<"Enter number of vertices and edges: ";
   cin>>n>>m;
   for(int i=0;i<m;i++)
   {
```

```
        int src,dst,weight;
        cin>>src>>dst>>weight;
        edges.push_back({src,dst,weight});
    }
    vector<int> d(n+1,1000000);

    //source vertex
    d[1]=0;
    for(int i=0;i<=n-1;i++)
    {
        for(auto e:edges)
        {
            int u,v,w;
            tie(u,v,w) = e;
            if(d[v]>d[u]+w)
            {
                d[v] = d[u]+w;
            }
        }
    }

    cout<<"All Shortest Path Weights:\n";
    for(int i=1;i<=n;i++)
    {
        cout<<d[i]<<' ';
    }
    return 0;
}
```

Output:

```
Enter number of vertices and edges: 5 10
1 2 6
1 3 7
4 1 2
2 3 8
2 5 5
5 2 -2
3 5 -3
2 4 -4
3 4 9
4 5 7
All Shortest Path Weights:
0 2 7 -2 4
Process returned 0 (0x0)   execution time : 39.321 s
Press any key to continue.
```

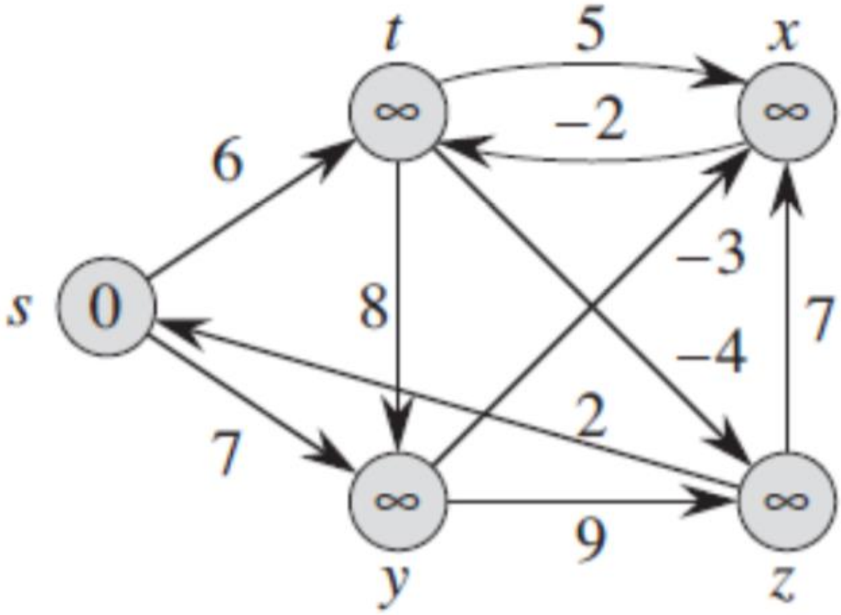
c) **Complexity of proposed algorithm (Time & Space)**

- Time Complexity: O(N^2)
- Space Complexity: O(N)

d) **Your comment (How your solution is optimal?)**

- Bellmon-Ford algorithm works in O(N^2) time which is slower than Dijkstra's algorithm but it will work for edges with negative weights.
- Bellmon-Ford algorithm can be also used to detect negative weight cycles

2. Show that Dijkstra's algorithm doesn't work for above graph



Ans:

Answer using greedy method:

Vertex	1	2	3	4	5
Distance	0	6	7	16	11
Actual Distance	0	2	7	-2	4

We can notice that greedy Dijkstra's algorithm don't work for negative edge weights.

3. Given a weighted, directed graph G = (V, E) with no negative-weight cycles, let m be the maximum over all vertices v V of the minimum number of edges in a shortest path from the source s to v. (Here, the shortest path is by weight, not the number of edges.). Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in m+1 passes, even if m is not known in advance.

Ans:

- We can simply implement this optimization of Bellmon-Ford algorithm by remembering if v was relaxed or not. If v is relaxed then we wait to see if v was updated (which means being relaxed again). If v was **not updated, then we would stop.**

- Because the greatest number of edges on any shortest path from the source is m, then the path weight in graph. By the upper-bound property, after m iterations, no d values will ever change. Therefore, no d values will change in the (m+1) st iteration. Because we do not know m in advance, we cannot make the algorithm iterate exactly m times and then terminate. But if we just make the algorithm **stop when nothing changes any more, it will stop after m + 1 iterations.**