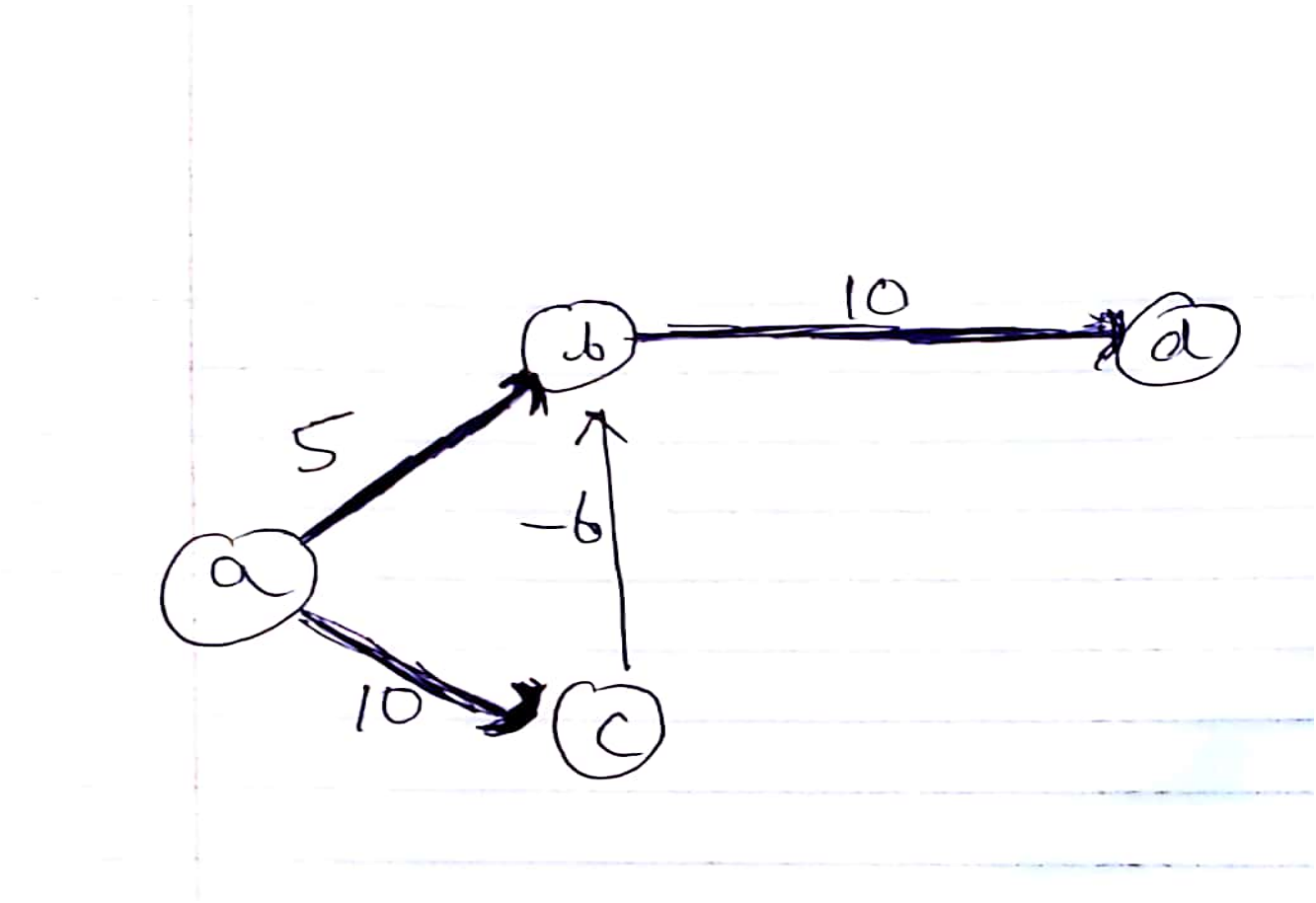
Fundamental Algorithms, Graphs, HW - 12

Tulsi Jain

1. Dijkstras algorithm

1.1 Example

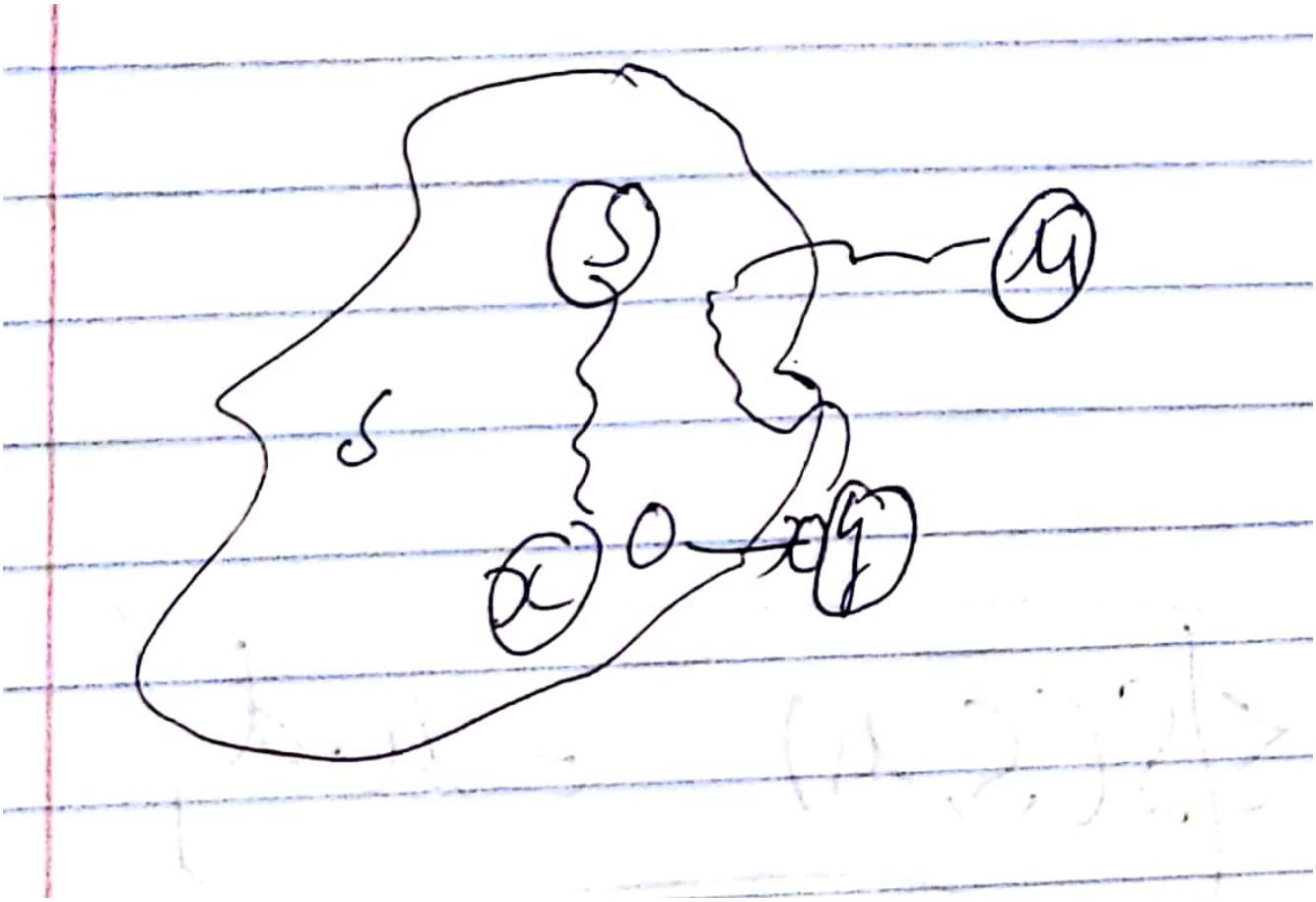
Figure 1.1: With Negative Weight



As per Dijkstras algorithm minimum distance from path a to b is 5 but because of introduction of negative weight from c to b. It possible to reach b i 4 via c. Hence Dijkstras algorithm does not work when edges weights are negative.

1.2 Dijkstras algorithm proof

Figure 1.2: Dijkstras algorithm proof



In Dijkstras algorithm proof we consider that min(s,y) is <= min(s, u) (y is parent of u in shortest path from s) but it weights can negatives than we can not apply min(s,y) is <= min(s, u). Hence proof of Dijkstras algorithm depends whether weights are negative or positive.

2. Dijkstras algorithm verification

First of all verify the following step in order
1. s.d = 0 and s.parent = NIL
2. v.d = v.parent.d + w (v.parent, v) for all vertex
3. v.d = ∞ if and only if it is not connected.

If any of the above verification tests fail, it means output is incorrect. Otherwise, run one pass of Bellman-Ford, i.e., relax each edge. If any values of v.d change, then declare the output to be incorrect; otherwise, declare the output to be correct.
Running time of this is O($|V| + |E|$).

3. Reliability Probability

Problem statement is to maximize the probability between two points. To solve this problem, we would apply certain modifications to the graph as Dijkstras algorithm works on addition of weights and output shortest path between two points. Following modifications are made to the graph.

1. New Weights(w_1) are obtained by replacing reliability probability with (1 - Reliabilityprobability). Now, problem statement is changed to finding a path corresponding minimum product of weights.

2. Now, New Weights(w_2) are obtained by, (w_1) is multiply by a number, **a** such that it becomes greater than (w_1)*a >= 1 and taking log of the weight. This would ensures that new weights(w_2) are not negative. We need to minimize the product, now converting it to log we need to minimize the sum of weight (w_2).
Now Dijkstras algorithm can be run to find the minimum path.

Algorithm 1 DIJKSTRA

```
function DIJKSTRA(G, w, s)                    ▷ (G Graph, w weight, s source)
    INITIALIZE-SINGLE-SOURCE(G, s)
    S = ∅;
    Queue Q = G:V
    while Q is not empty do
        u = EXTRACT-MIN(Q)
        S = S + {u}
        for v in adj (G.u) do
            RELAX(u, v , w);
        end for
    end while
end function
```

Algorithm 2 RELAX

```
function RELAX(u, v, w)                       ▷ (u start, v end, w weight)
    if v.d > u.d + w(u, v) then
        v.d = u.d + w(u, v);
        v.parent = u;
    end if
end function
```

4. Arbitrage

4.1 a

We need to modify this problem statement. To do this we take the negative log of all the values c_i that are on the edges between the currencies. Then, we detect the presence or absence of a negative weight cycle by applying Bellman Ford. To see that the existence of an arbitrage situation is equivalent to there being a negative weight cycle in the original graph, consider the following sequence of steps:

Algorithm 3 BELLMAN-FORD

```
function BELLMAN-FORD(G, s, w)                    ▷ (G Graph, s source, w weight)
    INITIALIZE-SINGLE-SOURCE(G, s)
    for  i = 1 to |G.V| 1 do
        for each edge (u, v) belongs to G.E do
            RELAX(u, v, w);
        end for
    end for
    for each edge (u, v) belongs to G.E do
        if v.d > u.d + w(u, v)  then
            return false;
        end if
    end for
end function
```

Algorithm 4 RELAX

```
function RELAX(u, v, w)                           ▷ (u start, v end, w weight)
    if v.d > u.d + w(u, v)  then
        v.d = u.d + w(u, v);
        v.parent = u;
    end if
end function
```

If there is cycle for which arbitrage is possible than summation of log of c_i would return negative than can be found when BELLMAN-FORD algorithms returns false. **Running time of this is $\mathbf{O(|V|^2)}$.**

4.2 b

Algorithm 5 BELLMAN-FORD

```
function BELLMAN-FORD(G, s, w)                    ▷ (G Graph, s source, w weight)
    INITIALIZE-SINGLE-SOURCE(G, s)
    for  i = 1 to |G.V| 1 do
        for each edge (u, v) belongs to G.E do
            RELAX(u, v, w);
        end for
    end for
    G = ∅
    for each edge (u, v) belongs to G.E do
        if v.d > u.d + w(u, v)  then
            G = G + v;
        end if
    end for
    return G;
end function
```

First repeat the steps of part a. Bellman-Ford returns false. We relax all the edges $|V|$ 1 many times. Then, we record all of the d values of the vertices. Then, we relax all the edges —V— more times. Then, we check to see which vertices had their d value decrease since we recorded them. We call this set a G. It means all these vertices are reachable from the negative weights cycles. To find one of these cycles in particular, we can pick any vertex from this set and greedily keep picking any vertex that it has an edge to that is also in this set. Then if we came at the same vertex we can say we have found a cycle. **Running time of this is O($|V|^2$).**