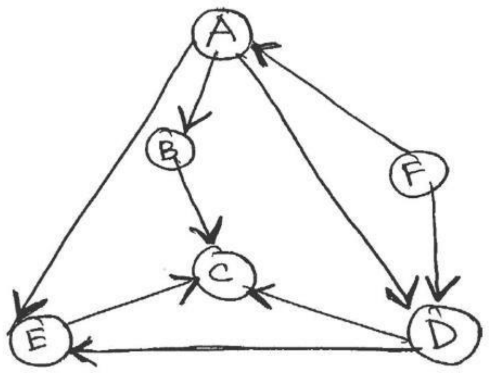
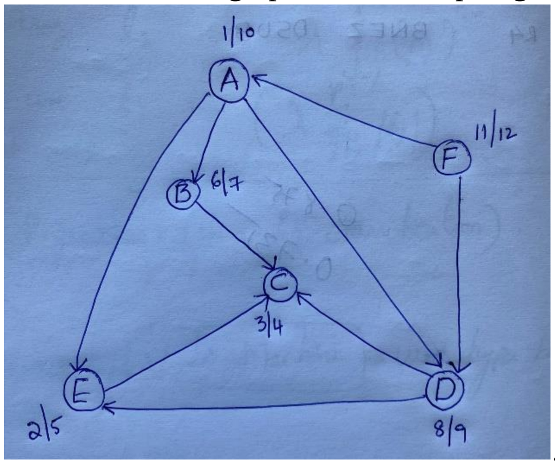
CS721 Homework - 2

WsuID: n659a624

Question 1. Consider the following directed acyclic graph.



(a) (20 points) Compute pre and post number for each node of the above graph starting from node *A* and draw graph after it is topologically sorted (linearized)

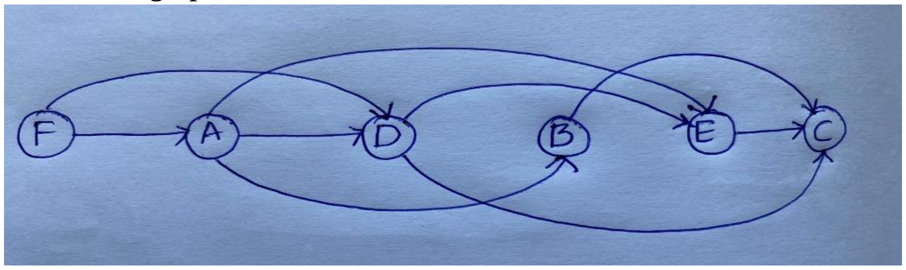


.

pre and post number for each node of the above graph starting from node *A* are A(1,10), E(2/5), C(3/4), B(6/7), D(8/9), F(11/12).

Topological sort of ordering is linear ordering of vertices such that post visit number are decreasing in order. Therefore it is F,A,D,B,E,C.

Linearized graph looks like as follows:-



(b) (20 points) Give a linear-time algorithm that takes as input a directed acyclic graph *G= (V, E)* and two vertices *s* and *t* and returns the number of simple paths (i.e., does not contain cycles) from *s* to *t* in *G*. For example, in the above graph there are four simple paths from vertex A to C: $A \rightarrow B \rightarrow C$, $A \rightarrow E \rightarrow C$, $A \rightarrow D \rightarrow C$ and $A \rightarrow D \rightarrow E \rightarrow C$. Your algorithm needs only to count the number of paths not list them.
Hint: First linearize the graph and locate nodes *s* and *t*. Which nodes will appear in simple paths from *s* to *t*? You need to consider only those nodes that may appear in simple paths between *s* and *t* (including *s* and *t*) in linearized order, maintain an array for whose indices are linearized ordering of the nodes and update the array elements as you process each node in the linearized order to get your final answer. What should this array contain? How will you update this array?

Answer:

- Let's consider a DAG Graph G =(V,E) and nodes u and v. Number of paths from node u gives number of simple between u and v.
- Suppose we fix the node v in the process. To find number of paths count, we add the paths that leave from neighbors of each u.
- As graph don't have cycles, we shouldn't add partial paths.
- We shouldn't consider same edge more than once in recursive calls.
- Therefore, total number of execution for-loop is O(V+E) from Number_of_Paths(s,t)

```
Number_of_Paths(s,t):
if(s==t)
return 1
else if s.paths != NIL
return s.paths
else
for each w in Adj[s]
 s.paths=s.paths+ Number_of_Paths(s,t) (w,t)
return s.paths
```

Question 2. We are given a directed graph *G= (V, E)* on which each edge *(u, v)*∈ *E* has an associated value *r(u, v)* which is a real number in the range $0 \leq r(u, v) \leq 1$ that represents the reliability of a communication channel from vertex *u* to vertex *v*. We interpret *r(u, v)* as the probability that the channel from *u* to *v* will not fail, and we assume that these probabilities are independent.

(a) (20 points) Give an efficient algorithm to find the most reliable path between two given vertices of this graph.

Hint: Since probabilities are independent, this means we want find a path for which product of the reliabilities will be maximum.

Answer:

Dijkstra's algorithm is used to find the most reliable shortest path between s,t and v. In which paths reliability is the product of the reliabilities of its edges.

So that we need to a path between s and t such that the product of r(s,t) is maximized.

Algorithm

```
Dijkstras (G,r,S)
Initialize_single_source(G,s)
S=0
Q=G.V

While Q != 0
U=Extract_Max(Q);
S=S u {u};

For each vertex v in  G.Adj[u];
If v.d < u.d * r(u,v) then
        v.d= u.d*r(u,v)
        v.π = u;
```

We find maximum of edges instead of minimum of edges.

Similarly, we find most reliable path by product of edges instead of sum of adjacent edges. Shortest path from s to t is the most reliable path and reliability of the path is product of reliabilities of each edge. Therefore, the time complexity of algorithm would be O(Elog V).

(b) (20 points) If the directed graph does not contain a cycle, can you give a better algorithm? Explain how your algorithm will work. What is the running time of your algorithm?

Answer:

Let us consider the graph as Acyclic graph(DAG) so that we can apply DFS algorithm by topologically sorting.

Therefore we can write as follows:

```
DAG G=(V,E)

for all u ∈ V:

dist(u) = ∞

prev(u) = nil

dist(s) = 0

Linearize   G
```

```
For each u∈ V, in linearized order
For all edges (u,v) ∈ V
Update(u,v)
```

In any path of DAG vertices appear to be in increasing the linearized order. By this we can say that graph is linearized in topological order by DFS. As we don't need any positive edges, we can also use negative edges of the original graph. If all edges are negative the longest path in converted graph will be the shortest path in original graph. As we repeat the process of topological order for V-1 times so that we find most reliable path.

The Running time complexity of the algorithm will be (V-1) times of log(V+E)

i.e O((V-1) Log(V+E))

Question 3. (20 points) Let *G= (V, E)* be an undirected connected graph, where all edge weights are positive and equal. Describe an algorithm (no need to provide pseudo-code) that finds MST of *G* and is asymptotically more efficient than Prim's and Kruskal's algorithm. What is the running time of your algorithm?

Answer:

Let weight of edge is 'w'.
Then total weight of the tree is w*(v-1) as graph has V-1 edges.
By applying BFS we get a tree which will be a part of minimum spanning tree(as weights are equal every tree of G will be MST). Therefore, runtime becomes O(V+E) which is more efficient than Prim's and Kruskal's algorithm.

As every spanning is a minimum spanning tree we can use either BFS or DFS.