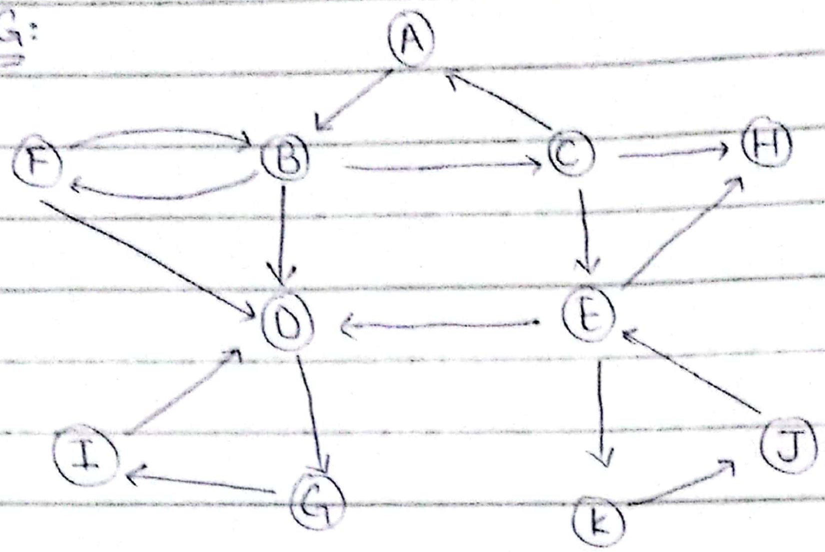
Muhammad Hamza Khan
216-5654
Algorithms

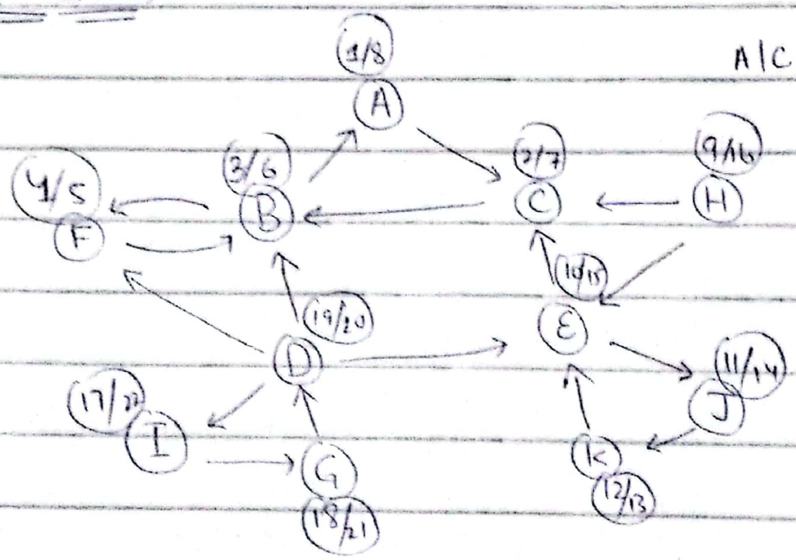
Homework # 6

Question Number Q1

G:

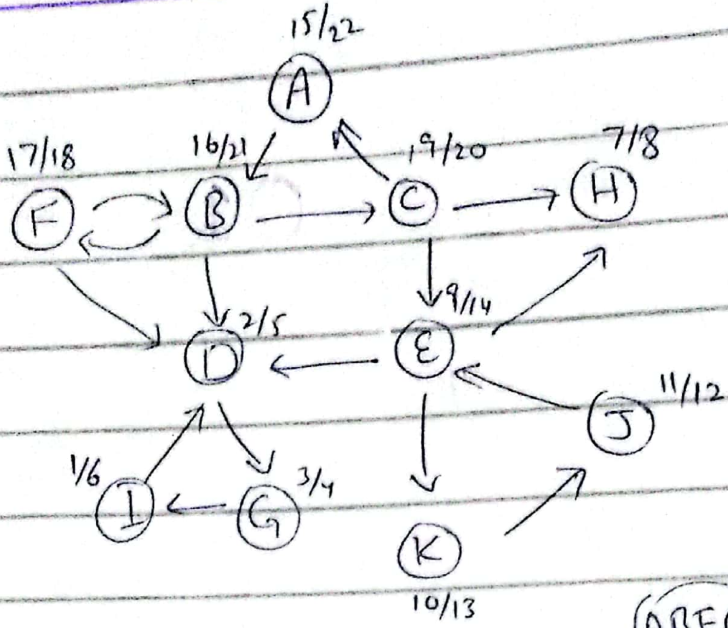


G(REVERSE):



finish time Order: I, G, D, H, E, J, K, A, C, B, F

Now, we will run DFS on original graph from the finish time order.



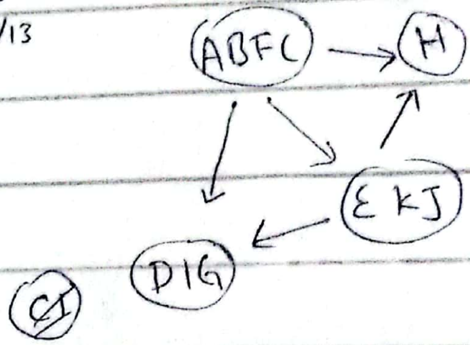
(1) I D G

(2) H

(3) E K J

~~(4) A B G F~~

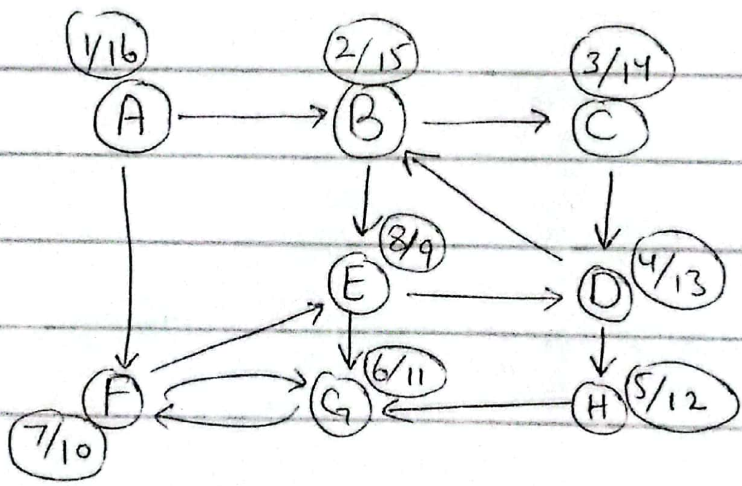
(4) A B F C



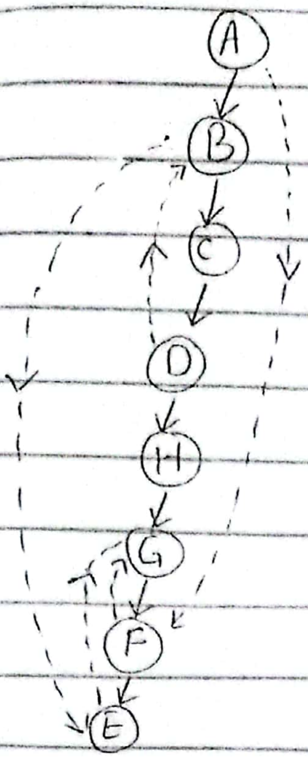
Strongly Connected Components.

Question 2:

A:



DFS Tree



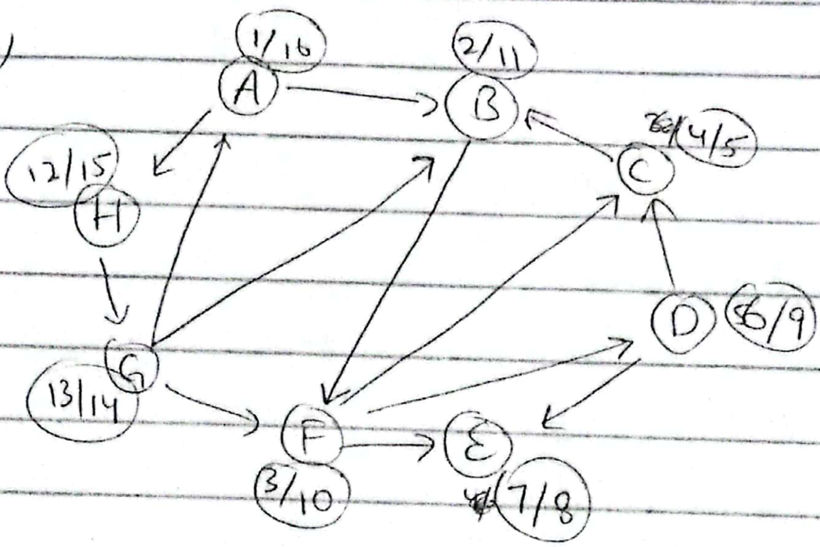
Edges =

Forward: (A,F), (B,E)

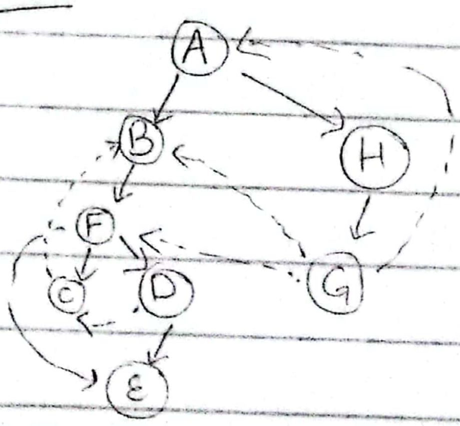
Backward: (E,G) (E,D)
(D,B) (F,G)

Cross: None

(b)



DFS Tree



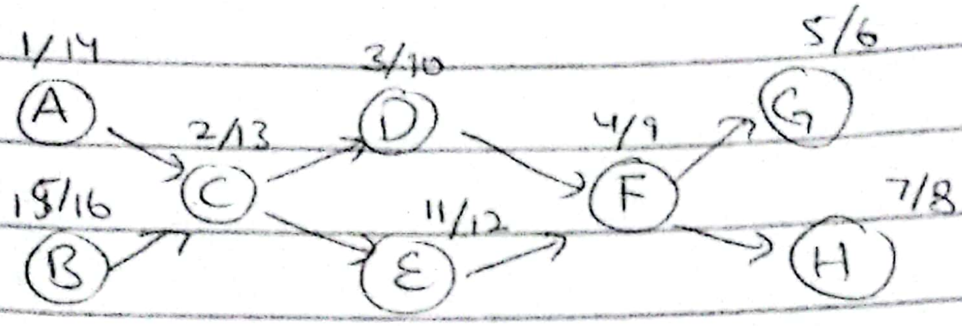
Cross:
~~(G,A)~~ (G,B), (G,F), (D,C)

Forward:
(F,E)

Backward:
(G,A), (C,B)

Question Number 3

a:



(b)

Sources for the graph is (A, B)

Sinks for the graph is (G, H)

(c)

~~B(i)~~ "BACEDFHG"

(~~(ii) A~~)

(d) For alphabetical : 2 topological ordering is possible.

whereas, if alphabetical order is not neccessary, then 8 orders are possible.

return distance;
}

Question 4:

```
int * ReverseGraph(adj[], V){
    //given adj[], we form a reversed list(adj).
    // (u,v) -> (v,u)
    int reversed_adj[V];
        //traverse the adj[] and reverse the
        //edges
        for( i =0 to V){
            for( j= 0 to len(adj[i])){
                edge= adj[i][j];
                vertex=i ;
                reversed_adj[edge].append(vertex);
            }
        }
    return ~~adj;~~ reversed_adj;
}
```

5 Question 5

(a) The time complexity will not be $O(|V| \lg |V|)$ rather it will be $O(E \lg V)$ because the actual time complexity would be

$$E + E \lg V + V \lg V$$

and we know that edges are greater than V in graph so $E \lg V >> V \lg V$

$$\Rightarrow O(E \lg V)$$

First we initialize distance array with '∞': traverse for each vertex, pick ~~minimum~~ vertex with minimum weight and then compare with crossing edges and insert minimum weighted vertex.

(b) SpKorchoff (G(V,E) s, r)

```
{
    distance[V] = [INT_MAX, ------ INT_MAX];
    distance[s] = 0;
    queue initialized with V.
    while (queue not empty) {
        U = Extract Min from Heap
        for(each V in ∈ U) {
            if distance [V]
            if ( distance[V] + G(U,V) < dist[V])
            {   distance[V] = G(U,V) + distance[V];
                decreasekey(queue, V, distance[V])
                queue insert(V, dist[V]),
            } // else { distance[V] remain same }
    }
    return distance;
}
```

Question 6 :

[A] house   n → agents.
hotels → h1, h2 · · · hn

On given source and graph we can run dijkstras algorithm to minimize risk.

function riskmin(G, A) {
    cost [V] = ~~∅~~ {∞ ∞ ..... ∞}
    parent[V] = ~~∅~~ {Null ...... Null}
    cost[A] = 0;
  Min Heap Q is initialized with V
    while (Q !empty)
    {
        U = Extract Min from Q.
        for (each V ∈ U) {   edge weight
        // ~~cost d[U]~~ ~~total cost = cost[U] + G(U,W)~~;
            if (cost d[V] > cost[V] + G(V,w))
            {   cost[V] = cost[V] + G(V,w);
                parent[V] = U;
                decreasekey(Q, V, d[V]);
            }
    return (cost, parent)
}

// for calculating all paths
function path(G) { path cost ~~[V]~~ ~~[all hotels count]~~
    for V in G :
        cost , parent = riskmin (G, V)
        pathcost [V] = (cost, parent)