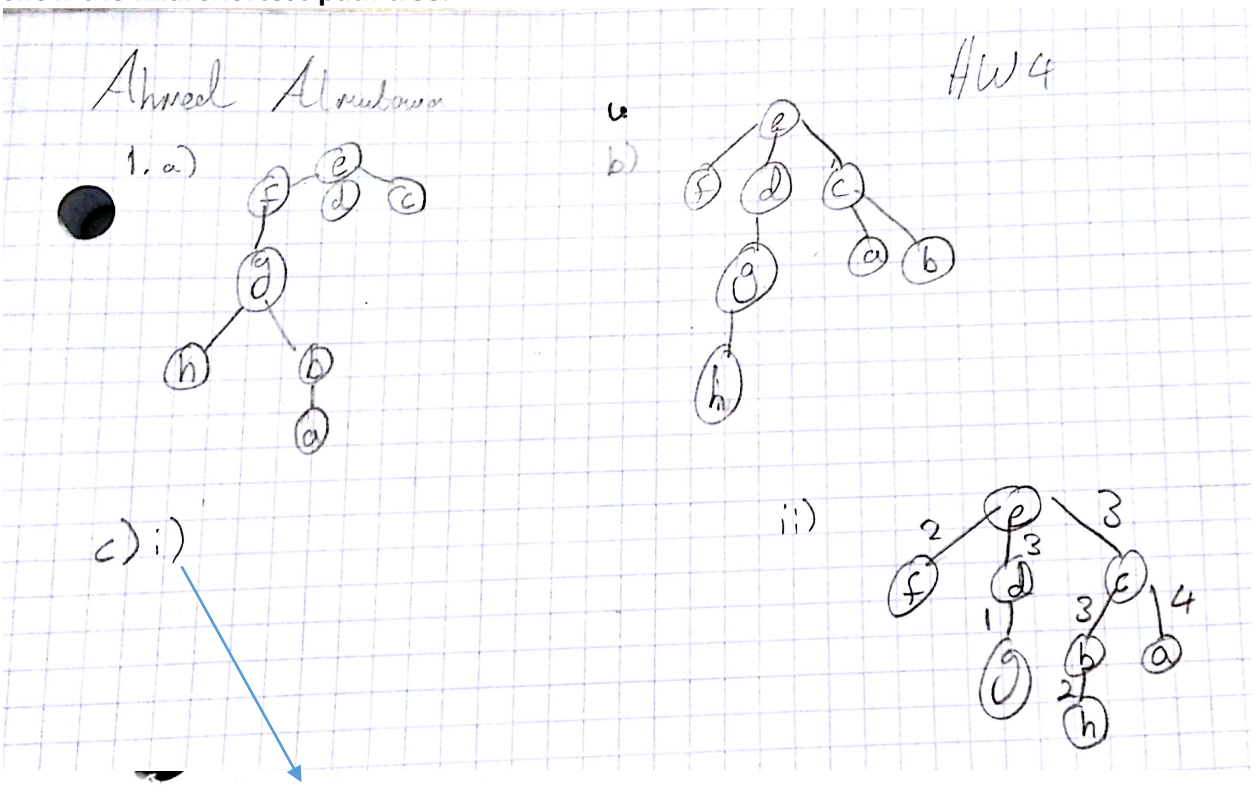
1. Answer the following questions for the graph shown below: (a) Draw the DFS search tree with starting vertex E and break ties alphabetically. (b) Assuming unit edge length (i.e., ignore edge weight), draw the BFS search tree with starting vertex E and break ties alphabetically. (c) Suppose the Dijkstras algorithm is run on the graph with starting vertex E: (i) draw a table showing the intermediate distance values of all vertices at each iteration of the algorithm; (ii) show the final shortest-path tree.



From E →	F	C	D	A	G	B	-
① E	2_A	3_A	3_A	∞	∞	∞	∞
② F	2_A	3_A	3_A	∞	6_F	∞	∞
③ C	2_A	3_A	3_A	7_C	6_F	6_C	∞
④ D	2_A	3_A	3_A	7_C	4_G	6_C	∞
⑤ G	2_A	3_A	3_A	7_C	4_G	6_C	18_G

2. Often there are multiple shortest paths between two nodes of a graph. Give a lineartime algorithm for the following task. Input : Undirected graph G = (V, E) with unit edge lengths; nodes u, v $\in$ V . Output : The number of distinct shortest paths from u to v.

To achieve linear-time, we modify a BFS to allowing us to keep track of the number of shortest paths as we go through it. Now we can either modify the struct of the Nodes to include this information, but that adds up to significant memory cost as we scale so I'll opt to just store things in a variable.
Pseudocode (from the book):

```
for all u ∈ V :
        dist(u) = ∞
        paths(u)=0
dist(s) = 0
Q = [s] (queue containing just s)
while Q is not empty:
        u = eject(Q)
        for all edges (u, v) ∈ E:
                paths(v)+=paths(u)
                if dist(v) = ∞:
                        inject(Q, v)
                        dist(v) = dist(u) + 1
                        paths(y)=paths(x)
```

3. You are given a strongly connected directed graph G = (V, E) with positive edge weights along with a particular node v0 $\in$ V . Give an efficient algorithm for finding shortest paths between all pairs of nodes, with the one restriction that these paths must all pass through v0. Describe your algorithm in words or write down the pseudo code. And analyze its time complexity

At first glance, this seems like a typical use case of Djikstra's algorithm from node a to b. Dictating that it has to pass through v_0 complicates things slightly. We can segment the path into two parts, a to v_0 and v_0 to b. If c is the shortest path between a and b, then a to v_0 has to be the shortest path between the two nodes for the first segment and in reverse, b to v_0 must also be the shortest path between those two nodes for the second segment. Then c can be pieced from a composite of those two. We run Djikstra's from a to v_0 once and then again for b to v_0, this will give us the shortest path between the two nodes. Since we ran Djikstra's twice then our O time is O(2|V|^2) or O(|v|^2).