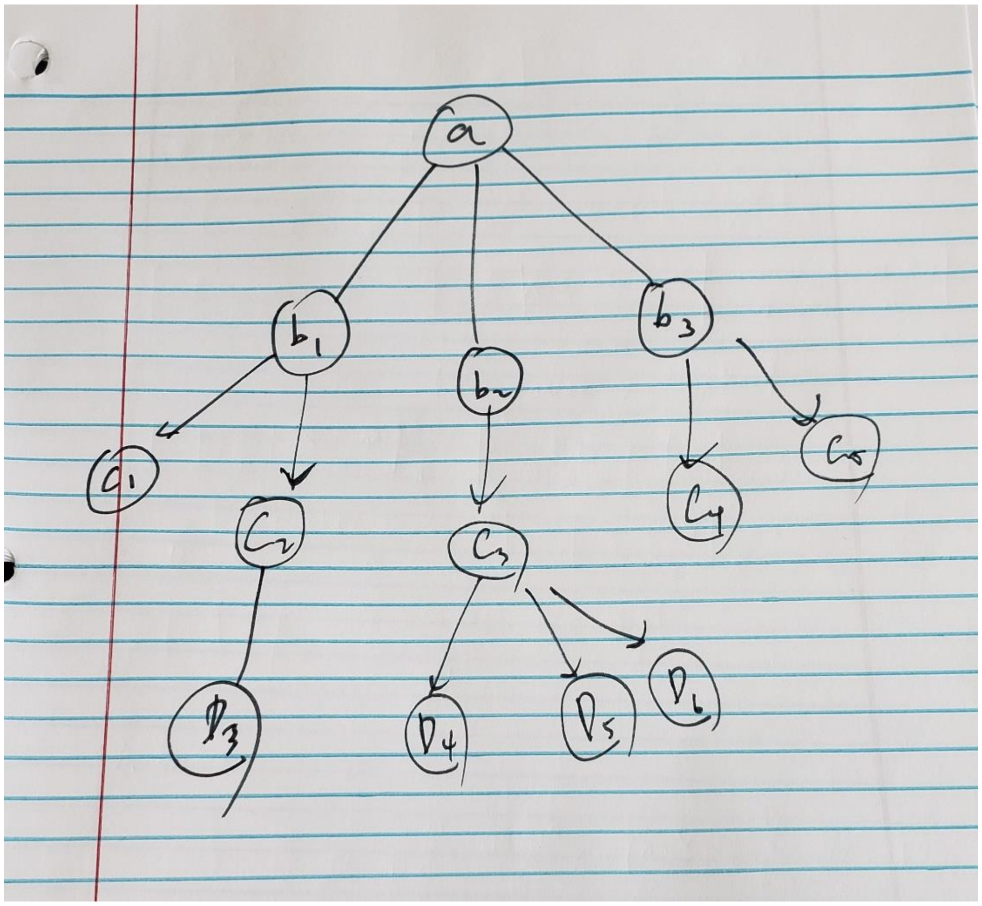
first work out how to determine the smallest vertex cover, then from that derive the vertices involved.



My observation is that, if an arbitrary node is not selected, then all of its children should be selected instead. For example, if a is not a part of the vertex cover set, then b_1, b_2, b_3 should be selected.

Subproblem: for each node v in G, the subproblem S(v) is defined as the minimum vertex cover for the subtree whose root is v.

Recurrence relations eqn: I will assign each depth(level) of the tree from the root as (a -> b -> c) where b's are the children of some nodes a's, and c's are the children of some nodes b's. Then the reccurence is given:

S(a) = min $\{(\sum_{b:child(a)} S(b) + 1) , (\sum_{c:child(b)} S(c) + 1)\}$

Define the order in which these subproblems are solved:
The base case is when the node being evaluated, do not have any children, i.e., the node is a leaf. Then, a's subtree is empty so there is no vertex to cover. The execution goes up in order of decreasing depth (... -> c -> b -> a) until S(a) is

calculated. Now with our S() function, which does not tell us the actual vertex cover set but just the number of nodes in the set, we introduce a new variable H(). H keeps track of whether the first term or the second term in the min() function gave the min value. If $\boldsymbol{(\sum_{b:child(a)} S(b) + 1)}$ was chosen, it means that the minimum vertex set cover can be given by choosing all the children of a, the b's. If $\boldsymbol{(\sum_{c:child(b)} S(c) + 1)}$ was chosen, this will mean that a will be a part of the cover set.

Runtime: For each node a in the graph, S(a) is called just once and for every node/root. And then, S() looks at the number of children of a by the number of edges going downward from a, or does the same for all children of a (all the b's). The function S() goes through all the nodes and all the edges connected from a while looking at each edge at most twice, so the runtime is 2|E| = O(n). Furthermore, getting the actual cover set via H( ) can take another O(n) time using an array and pointers to keep track. Therefore, the total runtime is still linear, O(n).