Problem1:

Q: What are the states?
A: A state represents that the situation of the search tree's nodes in a given moment. The set of all reachable state is called state space.

Q: What are the operators?
A: Operator is the action from one state to another state. For instance, add task or delete task.

Q: What is the branching factor?
A: Branching factor is the maximum number of successors (child nodes) generated by a node (parent node).

Q: Is the depth of the goal node known initially?
A: No, the depth of the goal node isn't known initially. What we know is the possible upper limit of the depth of the goal node. For instance, some search tree G has n nodes in total. So the depth of the goal node is no greater than n. The goal node might be on may level, different problem have different answer.

The upper limit of the depth of the goal node is base on the assumption that the task of each level requires X = min T.V value, and Y = min T.L time. Since the requirement of Problem1 has target value is M, so the upper bound of depth will be ceiling (M / X) when the adding each task with the value X. And the problem requires an overall time of a correct schedule must be no more than D, so the upper bound of depth will be floor (D / Y) when adding each task with the time Y. The depth must satisfy that it is no greater than ceil (M / X), floor (D / Y) and N (as mentioned above). So the depth must be no greater than the smallest value of these three values.

Problem 2:

A. Fig.1 is the graph of Problem 1 using DFS (Page 2). The correct schedule is AEB.

B. Fig.2 is the graph of Problem 2 using BFS (Page 3). The correct schedule is AEB.

Problem 3:

A: Fig.3 is the graph of Problem 1 using BFS, assuming that those repeated states are eliminated. So we don't touch the CA, EA, EC.

B: We can use hash table as explored set. The nodes that have been visited can be put into the explored set (hash table). Hash key is the frontier node that is currently visited.

Under some order form, hash value is the node formed by the ordered combination of tasks. For Instance, we can hash the combination of tasks (node) by Alphabetical order. Firstly, AC has been visited and put into hash table. When CA is visiting, the hash key is CA and hash value is AC. We check the hash table to find whether AC is in the hash table or not. If it is there, we discard CA. If not, we put it in hash table.

C: Since BFS stores all the visited state in the explored set; the solution in B can eliminated the repeated state by checking the explored set (hash table). However, DFS and ID will not store the previous states until we find the goal state. The advantage of DFS and ID is saving memory. If using hash table, the memory of storage will increase. So using hash table will not be efficient for DFS and ID.

D: The repeated states are caused by the two separated nodes without edge, like A and C, C and E, A and E. We expand the nodes by Alphabetical order initially. Take Problem 1 as example, we expand the nodes by following A, C, E. When A is expanded, the children nodes are AC and AE (B and D cannot be executed right now). When we expand C, the children are CA and CE. The generation of repeated state CA is because that A was expanded before C. To eliminate the repeated states in DFS and ID, we only add the task with the greater Alphabetical letter than current expanding node. For instance, we add task C or E to A (A is expanded right now), and add E to C (C is expanded right now). To do so, repeated states (CA, EA and EC) will be eliminated.

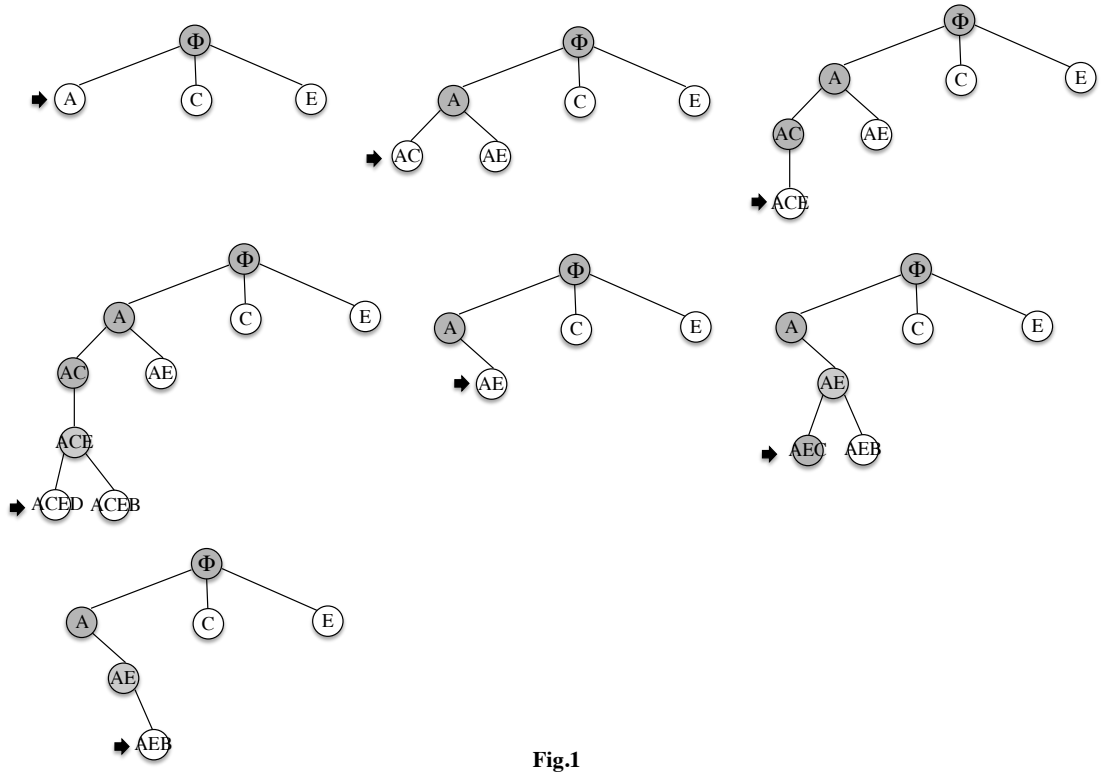

Fig.1

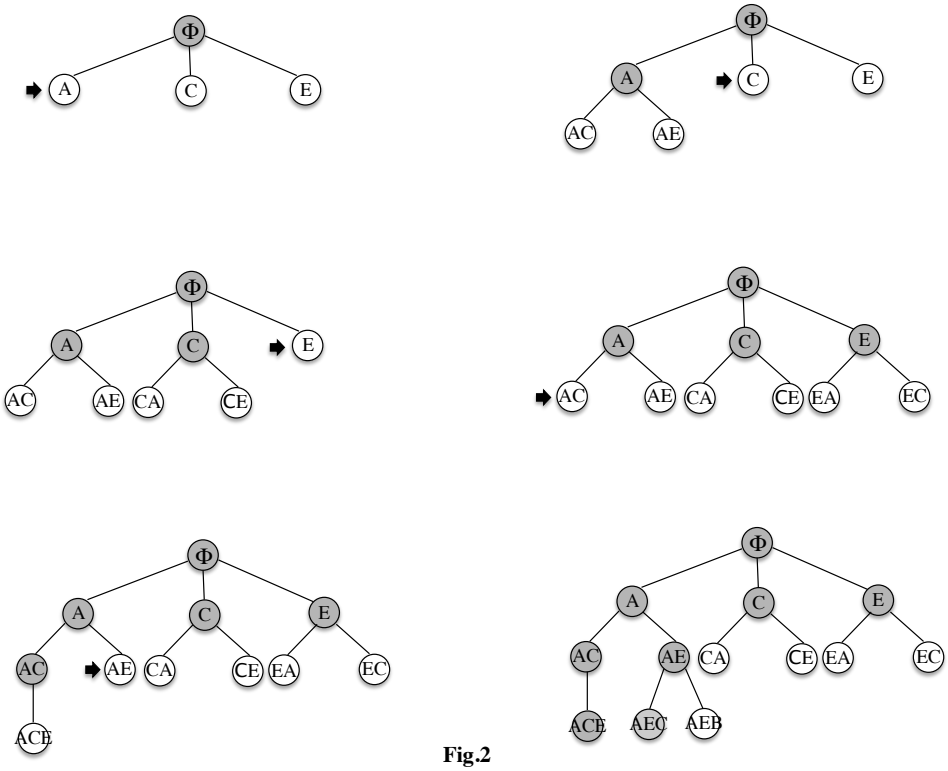

Fig.2

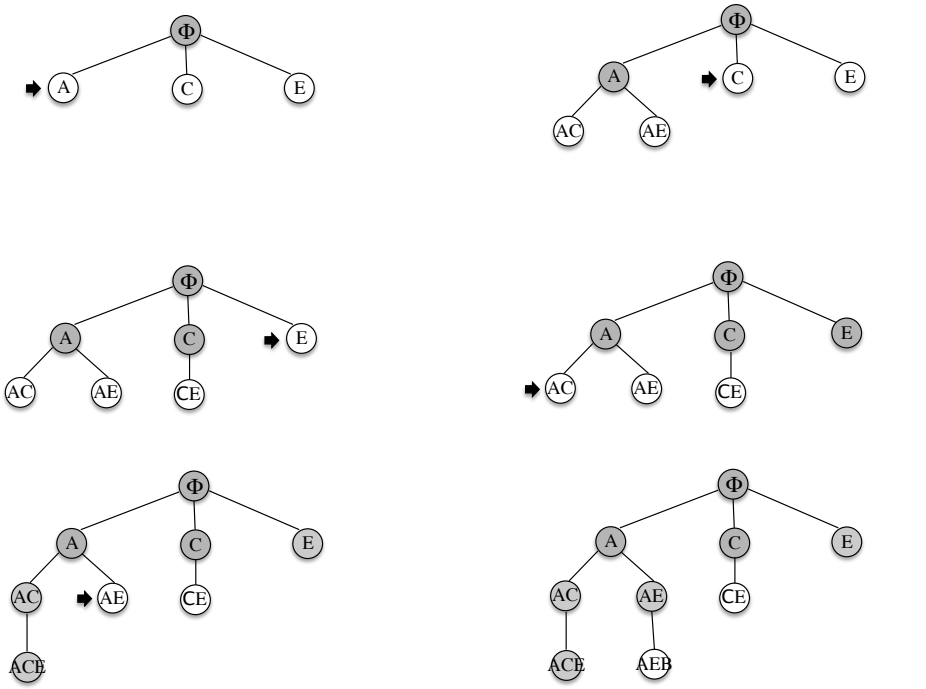

Fig.3