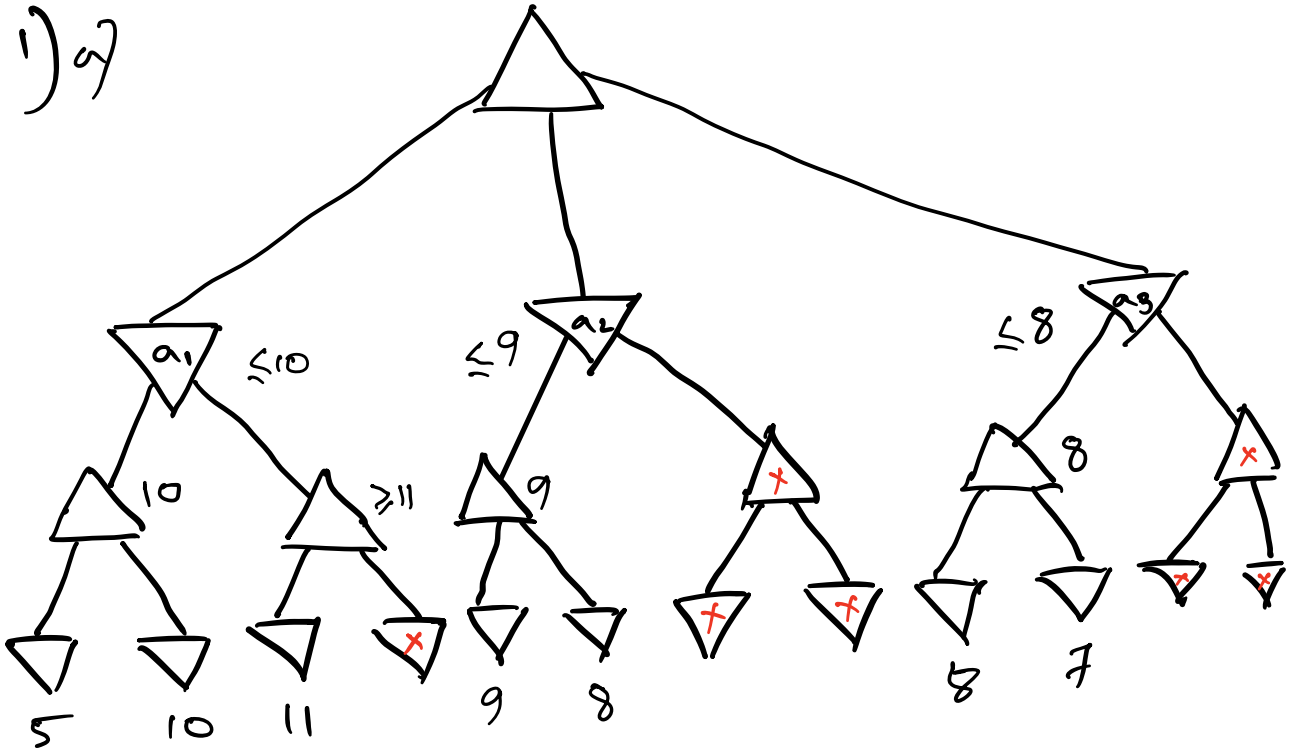
1) a)



For the maximum performance of alpha-beta pruning, the utility values of the leaf nodes are marked in the above figure. The pruned nodes are marked in 'x'. The leaf nodes marked in 'x' can have any values without affecting the performance of

alpha-beta pruning.

b) The ideal way of ordering nodes is to order them from best to worst, based on whose move it is, such that the best node would be explored first.

For example, in question 1.a, for the root of the tree (MAX node), the best move 'a_1' was explored first. Because of that right-hand subtrees of 'a_2' and 'a_3' were pruned.

Similarly, in case of 'a_1' (MIN node)

the best move (left subtree of 'a;') was explored first. Because of that one leaf node in the right hand subtree of 'a,' was pruned.

2) a) We have $Q_a = 2$ and $Q_g = 4$.

Therefore the possible acceptable values of the other variables are as follows:

Variable	Domain
Q_b	$\{5,6,7\}$
Q_c	$\{1,3,5,6,7\}$
Q_d	$\{3,6\}$
Q_e	$\{1,3,5,7\}$
Q_f	$\{1,6\}$

Table 1

Therefore, using minimum remaining value heuristics, we see that there is a tie between Q_d & Q_f. As per the question, in case of a tie, we choose the leftmost column. Therefore, the backtracking search would choose column 'd' next i.e. Q_d will be chosen next.

2.4) Now, $Q_e = 1$ has been chosen.

Therefore, we have:

$Q_a = 2, Q_e = 1, Q_g = 4$

From Table 1, we get the following remaining domains for other variables. I have shown what values are removed in AC 3, in the same Table (Table 2):

Variable	Prev Domain	Domain before AC3	Removed in AC3
Q_b	{5,6,7}	{5,6,7}	{6}
Q_c	{1,3,5,6,7}	{5,6,7}	{6}
Q_d	{3,6}	{3,6}	{6}
Q_f	{1,6}	{6}	None

Table 2 (After AC3)

{6} is removed from Q_b, Q_c, & Q_d as it is not ARC consistent with Q_f.

2.c) I believe this formulation (7 variable with each having a domain size of 49) is computationally more expensive.

Reason: This representation has $(N^2)^N$ states where $N = 7$. Therefore, with this formulation there are $(7^2)^7$ different value assignments in the search space of CSP.

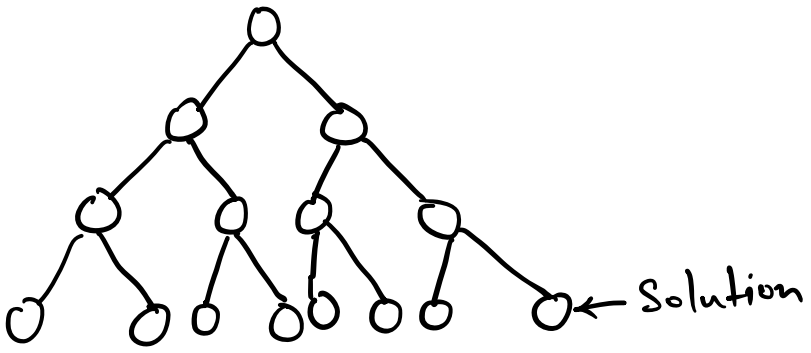
On the other hand, in the previous formulation, we have only N^N states where $N=7$. Therefore, we have only 7^7 different possible value assignments

in the search space of CSP. This is much better compared to $(7^2)^7$ different states.

3.a) No.

Reason: The problem with iterative deepening is that it generates some states multiple times, once for each value of depth limit, before reaching the solution. For example, the nodes on the bottom level (depth d) are

generated once, those on next to bottom level (depth d-1) are generated twice, and so on, upto the children of the root which are generated d times. Now, consider a graph (or tree) which has the solution at the deepest level at the last explored node by regular depth-first search. Consider the following example:



Example Tree

In this case, time complexity of iterative deepening depth-first search would be more compared to regular depth-first search.

Note that, this is a very special case. In most of the other cases, when there are multiple solutions at different depths, and these depths are unknown, iterative deepening performs equal or better in terms of optimality, completeness (assuming finite branching factor).

3.b) No.

Reason: Let us consider two heuristics h_1 & h_2. Assume that both are admissible & consistent, i.e.

$$h_1(n) \leq h^*(n) \quad \text{----- (Admissibility)}$$
$$h_2(n) \leq h^*(n),$$

for all node n in the graph, where $h^*(n)$ is the optimal cost to reach a goal from n.

For consistency we have,

$$h_1(n) \leq c(n, a, n') + h_1(n') \quad \text{---- (consistency)}$$
$$h_2(n) \leq c(n, a, n') + h_2(n'),$$

for every node n & every successor n' of n generated by an action a.

It is straight forward to show the

heuristic that is an average of $h_1(\cdot)$ & $h_2(\cdot)$ is also admissible & consistent. Let us denote the average heuristic by $h_3(\cdot)$. Therefore, we have,

$$h_3(n) = \frac{h_1(n) + h_2(n)}{2}, \text{ for all } n.$$

Suppose, $h_1(n) > h_2(n)$ $\forall$ non-goal node n. Therefore, in terms of speed, any algorithm that uses $h_1(\cdot)$, will be faster than the one that uses $h_2(\cdot)$. This is because of the fact that the more accurate the heuristic is in comparison with $h^*(\cdot)$, the less nodes will be explored to find a solution.

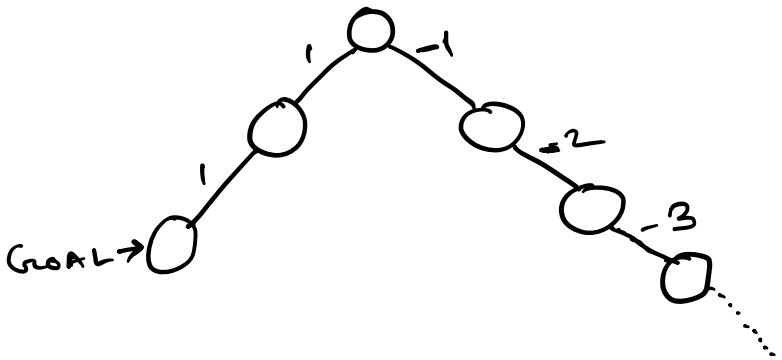
Now, note that, for all non-goal node n,

$$h_1(n) > h_3(n) = \frac{h_1(n) + h_2(n)}{2} > h_2(n)$$

$\therefore h_3(\cdot)$ is better than $h_2(\cdot)$ but worse than $h_1(\cdot)$ in terms of speed.

3.c) No.

Consider the following example -



In this case, one branch of the tree has all negative path costs. Here, A*

will start exploring the right hand branch and continue along that because on that branch, $f(n_R)$ is always lower than $f(n_L)$, where n_R is a node in right branch & n_L is a node in left branch. A* will never find the goal even if the heuristic is perfect.

3.d) Yes.

We can use the utility values of each node as the path cost to reach that node from it's parent &

use heuristic value of zero for all nodes to find optimal move using A*, considering the agent plays as a MIN agent. It means, had the agent been a MIN agent in a MINIMAX algorithm, it would have found the same solution, as given by A*, using the 'predict(·)' function in both the cases.