CS121 – Introduction to Artificial Intelligence – Winter 2011

Homework #6
(Constraint Satisfaction, Non-Deterministic Uncertainty and Adversarial Search)
Out: 2/21/11 — Due: 2/29/11 (at noon)

How to complete this HW: First copy this file; then type your answers in the file immediately below each question; start each question on a separate page, write your name on every page. Finally, print this file and return it in the drawer marked CS121 of a file cabinet located in the entryway of the Gates building next to office Gates 182 (see red arrow in http://ai.stanford.edu/~latombe/cs121/2011/map.jpg) no later than Tuesday 2/29 at noon.

Your name: ……………………………………………………………………………………

Your email address: ………………………………………………………………………

Note on Honor Code: You must NOT look at previously published solutions of any of these problems in preparing your answers. You may discuss these problems with other students in the class (in fact, you are encouraged to do so) and/or look into other documents (books, web sites), with the exception of published solutions, without taking any written or electronic notes. If you have discussed any of the problems with other students, indicate their name(s) here:
………………………………………………………………………………………………
Any intentional transgression of these rules will be considered an honor code violation.

General information: Justify your answers, but keep explanations short and to the point. Excessive verbosity will be penalized. If you have any doubt on how to interpret a question, tell us in advance, so that we can help you understand the question, or tell us how you understand it in your returned solution.

Grading:

Problem#	Max. grade	Your grade
I	25	
II	25	
III	25	
IV	25	
Total	100	

Name …………………………………………………………………..

I. Logic and Constraint Satisfaction

Assume that the states of a world are described using n Boolean propositions P_1, ..., P_n. In any particular state, each one of these propositions is either true or false. Hence, 2^n states can be distinguished.

Let a propositional sentence over the n propositions P_1, ..., P_n be any sentence built using the Boolean operators $\wedge$ (and), $\vee$ (or), and ~ (not). An example of a sentence over 3 propositions is $(P_1 \vee {\sim}P_2) \wedge {\sim}P_3$. A given state of the world *satisfies* a sentence S if S is true in this state.

1. Consider the problem of determining if there exists at least one state of the world that satisfies k given sentences S_1, ..., S_k. How would you formulate this problem as a constraint-satisfaction problem? What would be the variables, their domains, and the constraints?

2. A sentence S is said to be *entailed* by the sentences S_1, ..., S_k., if S is true in every state of the world that satisfies all sentences S_1, ..., S_k. Suppose that a general algorithm CSP-SOLVER is available to solve constraint-satisfaction problems. How can CSP-SOLVER be used to determine if S is entailed by S_1, ..., S_k?

II. Designing a crossword puzzle

Consider the problem of constructing a 6×6 crossword puzzle. The goal is to place a valid 6-letter word in each of the six rows and each of the six columns. We are given a large dictionary that contains all the valid 6-letter words (there are 26 letters in total, A through Z). There are N words in the dictionary.

1. Formulate this problem as a constraint satisfaction problem with 12 variables. What are these variables and their domains? How many constraints are there? What are they?

2. Assume that the dictionary only allows the following type of query: Given any partial word with $0 \leq k \leq 6$ known letters at specified locations, the dictionary returns the number (possibly 0) of valid words that contain these letters at these locations. If this number is not 0, then the query also returns one such valid word. For instance, a partial word can be A-O--T (here, k = 3) and a valid word containing these letters is AMOUNT. If the same query is made several times, each query returns a new valid word if there exists one, otherwise returns that all words have already been returned.

   Let the crossword be constructed using a backtracking algorithm. After this algorithm has inserted the n^{th} word in the crossword, what should it do? How can it detect that it is in a dead-end? If no dead-end is detected, how should the algorithm select the variable that will be assigned a value next?

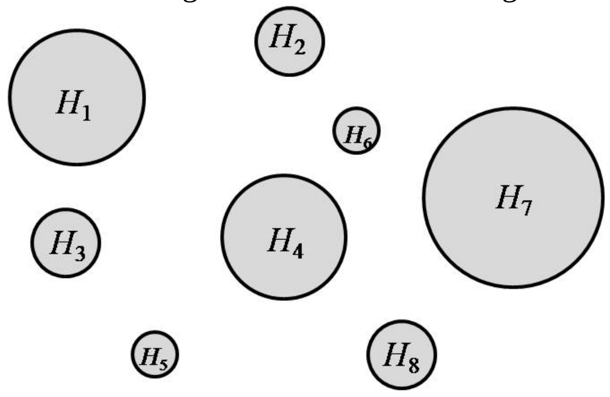
Name ……………………………………………………………………..

III. Uncertainty and Search: Navigation through Hotspots

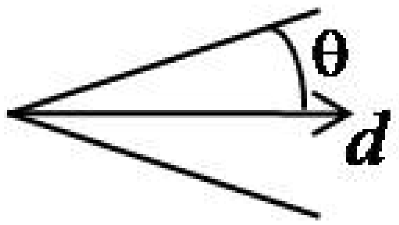
Using the position information given by your cell phone you want to go from point *S* to point *F* in a planar environment *P* without obstacle. Unfortunately, your telephone company does not provide good coverage. In covered areas, your phone gives your exact position, but in uncovered areas it gives you no information.

More formally, we model this navigation problem as follows. Your position is represented by a point *Y* in the horizontal plane *P*. The phone reception areas are represented by *n* discs H_1, ..., H_n in *P* and are called *hotspots* (see Figure 1.*a*). These discs may have different radii and no two discs overlap. Whenever you are in a hotspot H_i, you exactly know your current position *Y*. Using this information you can precisely navigate from any position to any other position within H_i. In addition you have a perfect map of the hotspots: you know the coordinates of their centers and their radii.

However, outside the hotspots, you cannot control perfectly your velocity. You may choose a direction of motion *d*, but the magnitude of your velocity is unknown and only guaranteed to be non-zero. Moreover, its angle with the chosen direction *d* is only guaranteed to be no greater than some given constant angle $\theta < \pi/2$ (see figure 1.*b*). During the motion your velocity may vary in both magnitude and orientation. Therefore you are always guaranteed to move in an uncertainty cone of half angle $\theta < \pi/2$ centered along *d*.



(*a*)



(*b*)

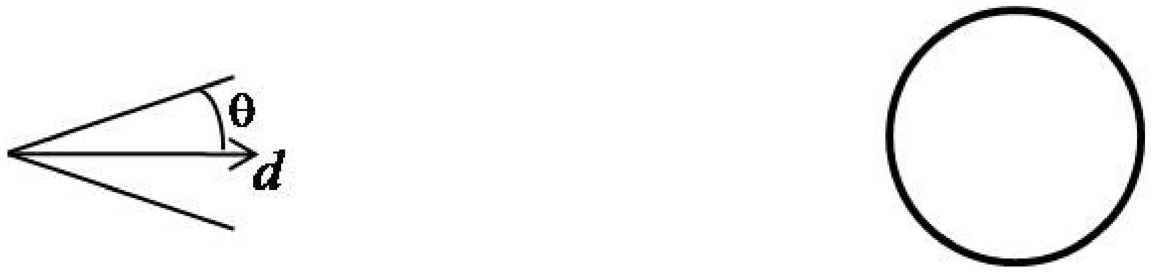
Figure 1: (*a*) A plane P with 8 hotspots. (*b*) Cone of possible velocities outside the hotspots when the chosen direction of motion is *d*.

Your start position *S* and your final position *F* are both located in hotspots, but in different ones. So, to navigate from *S* to *F*, you must hop from hotspots to hotspots. Whenever you leave a hotspot, you may select to leave it at any chosen point in its circular boundary, since you can navigate precisely to this point from within the hotspot. From this leave point you must choose a direction of motion *d* and keep moving along this direction (with the uncertainty model defined above) until you reach a hotspot of your choice. The only directions *d* that are allowed are North, East, South, and West. You cannot change *d* between two hotspots; you may pick a new direction *d* only when you leave a hotspot.

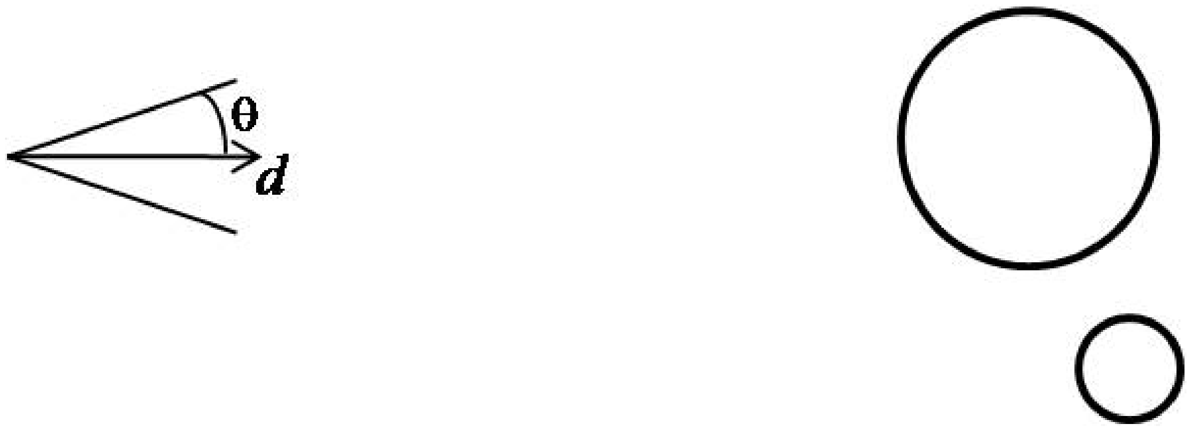
Name ………………………………………………………………..

1. We would like to build a backward planner to plan a motion from *S* to *F*. To do this, we define the **preimage** *Pr*(*G*,*d*) of a set *G* of hotspots for $d \in$ {North, East, South, and West} to be the region in the plane *P* such that a motion along *d* from any point in this region is guaranteed, under the uncertainty model defined above, to eventually reach a hotspot in *G* (anyone).

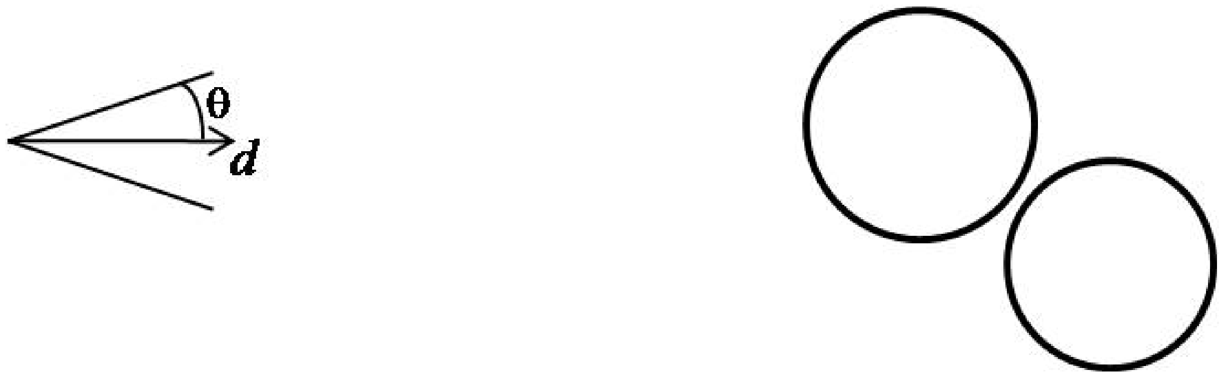
   a. Let *G* consist of the single hotspot shown below. Draw the region *Pr*(*G*,*d*) for *d* = East in the figure below. The angle θ is shown in the figure.



   b. Let *G* consist of the two hotspots shown below. Draw the region *Pr*(*G*,*d*) for *d* = East in the figure below. The angle θ is shown in the figure.



   c. Let *G* consist of the two hotspots shown below. Draw the region *Pr*(*G*,*d*) for *d* = East in the figure below. The angle θ is shown in the figure. [Hint: The answer is slightly more involved than for question b above.]



Name ………………………………………………………………..

2. Assume from now on that an algorithm is available to compute *Pr*(*G*,*d*) for any set of hotspots *G* and any for $d \in$ {North, East, South, and West}.

   a. Formulate backward planning as a search problem:
   - What would be the elements (states) of the search space?
   - What would be the "initial" node of the search tree?
   - What would be the "goal" nodes of this tree?
   - What would be the successors of a node?
   - What would be the branching factor of the tree?

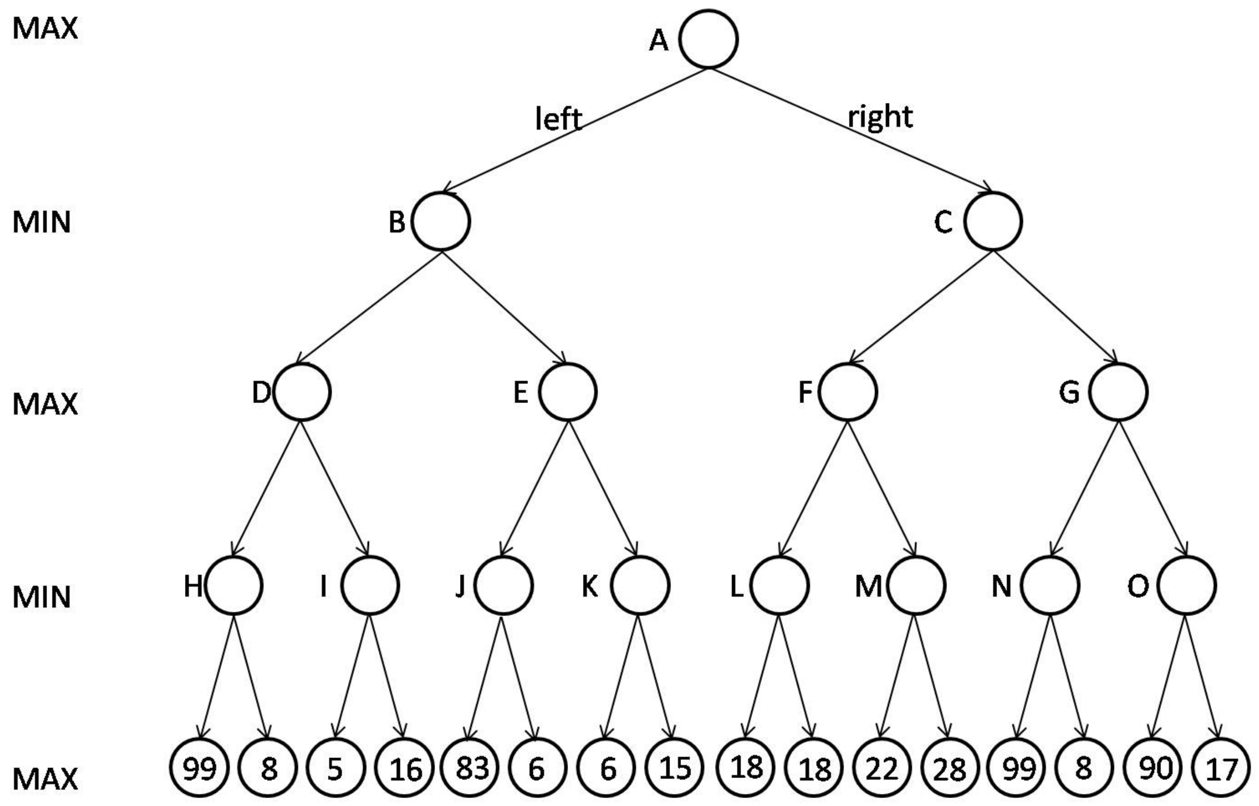
   To answer these questions, you should first form a good idea of how backward planning will work on this problem.

   b. Assume that the search succeeds. Explain in English how a plan computed by the backward planner will be executed.

Name …………………………………………………………………..

IV. Game Playing

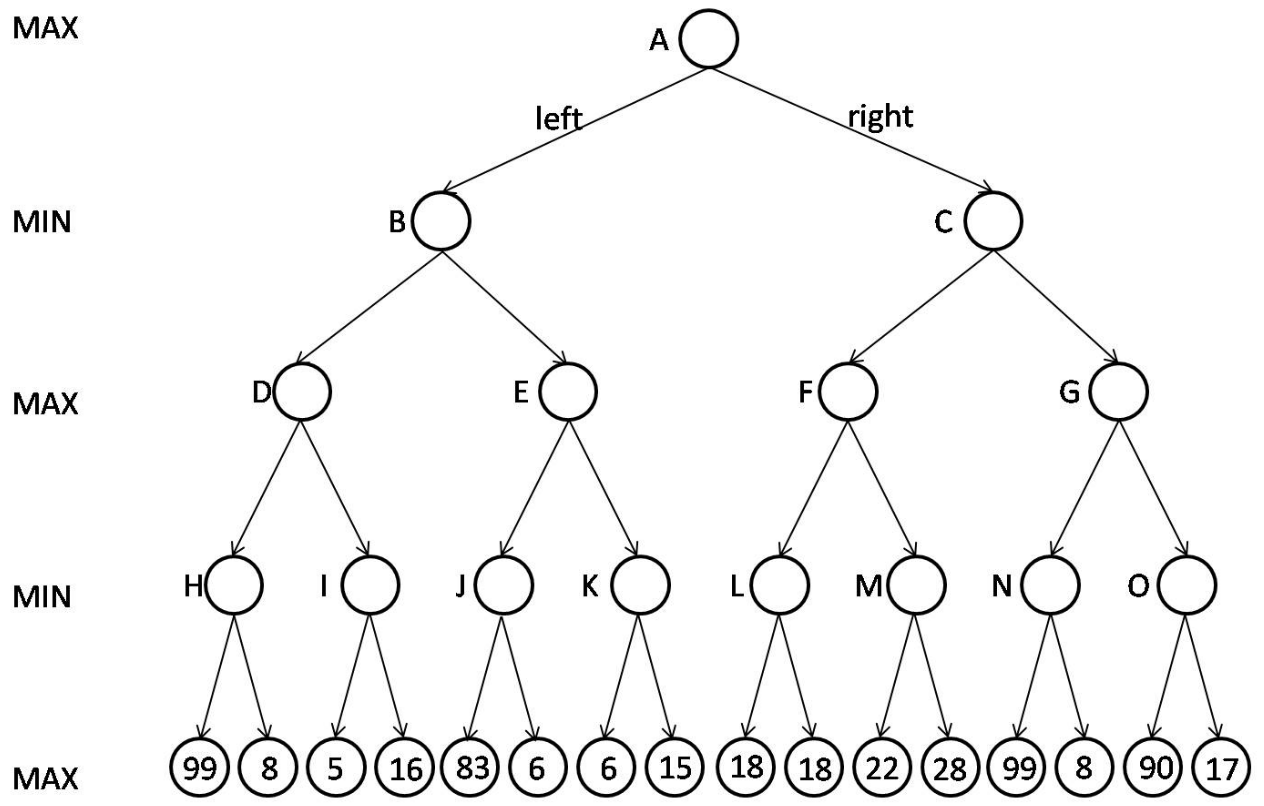
Consider the two-player game tree shown below. This tree is a binary tree and its leaf nodes are all at depth 4. The two players, MAX and MIN, play in turn. At each turn the playing player must choose between moving "left" or "right". MAX is the first player and uses an evaluation function *e* that maps each state of the game (node of the tree) to a positive integer. A larger *e* at a node, the more promising this node is for MAX. The value of *e* at each leaf is shown inside the leaf node. For the purpose of this problem, we give a label to each intermediate node: the root is labeled by A, its two children by B and C, etc. The leaves at depth 4 have not been assigned any labels.



1. In the above figure fill each intermediate node with the value that is backed-up by the Minimax algorithm. According to the backed-up values, what move MAX should play: left or right?

Name …………………………………………………………………..

2. The same game tree is reproduced below.

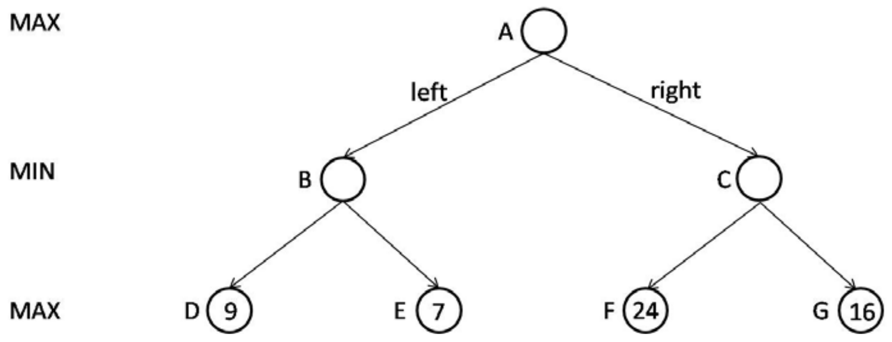


   a. Now fill each intermediate node with the alpha and beta values computed by the Alpha-Beta pruning algorithm. In addition, clearly mark all the nodes that are not examined by this algorithm by filling them in black (or any color available to you) or with a clearly visible **X**. Non-examined nodes may include leaf nodes and intermediate nodes.

   b. According to the alpha and beta values what move MAX should play: left or right? In general, is the best move computed by the Alpha-Beta pruning algorithm guaranteed to always be the same as the one computed by the Minimax algorithm?

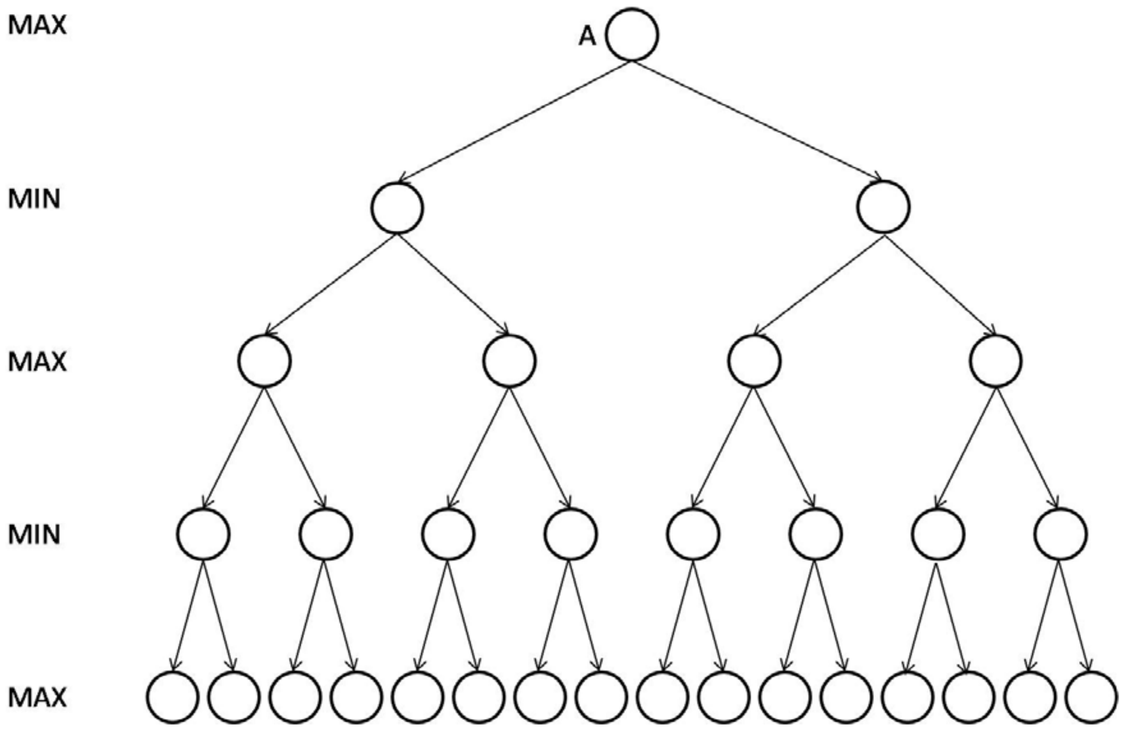
3. In this question, we still consider the same game tree. But now MAX would like to increase the number of nodes not examined by the Alpha-Beta pruning algorithm. For that purpose, MAX first runs the Minimax algorithm down to depth 2, not to select a move, but only to acquire some information that may help the Alpha-Beta pruning algorithm. So, the tree generated at depth 2 by Minimax consists of nodes A, B, ..., F and G. The figure below shows this tree along with the values of *e* at the leaf nodes. Minimax backs-up these values at nodes B and C, but the backed-up values are not shown.

Name …………………………………………………………………..



Now, MAX runs the Alpha-Beta pruning algorithm down to depth 4. Here, this algorithm uses both the values of e at depth 2 and the backed-up values computed by Minimax at nodes B and C to reorder the nodes of the depth-4 game tree at depths 1 and 2, with the goal to maximize the number of nodes that will not be examined. [Note that, after reordering the nodes, the "left" arc from a node may be on the right, and vice-versa.]

a. How should the Alpha-Beta pruning algorithm reorder the nodes, given the information generated by Minimax? To answer this question, assign the labels B, C, ..., N and O to the nodes of the tree shown below, assuming that the Alpha-Beta pruning algorithm will perform a depth-first traversal of the tree from left to right. In addition, inside each leaf node give the value of the evaluation function.



b. How many nodes will not be examined by the Alpha-Beta pruning algorithm?