CS 188
Fall 2021

Introduction to
Artificial Intelligence

Challenge Q9 HW3

- **Due:** Tuesday 9/21 at 10:59pm.
- **Policy:** Can be solved in groups (acknowledge collaborators) but must be submitted individually.
- **Make sure to show all your work and justify your answers**.
- **Note:** This is a typical exam-level question. On the exam, you would be under time pressure, and have to complete this question on your own. We strongly encourage you to first try this on your own to help you understand where you currently stand. Then feel free to have some discussion about the question with other students and/or staff, before independently writing up your solution.
- Your submission on Gradescope should be a PDF that matches this template. Each page of the PDF should align with the corresponding page of the template (page 1 has name/collaborators, question begins on page 2.). **Do not reorder, split, combine, or add extra pages**. The intention is that you print out the template, write on the page in pen/pencil, and then scan or take pictures of the pages to make your submission. You may also fill out this template digitally (e.g. using a tablet.)

First name	
Last name	
SID	
Collaborators	

For staff use only:

Q9.	Challenge Question (Multi-Agent Search)	/17

Q9. [17 pts] Challenge Question (Multi-Agent Search)

Let's consider a game played on a board with M rows. The maximizing agent (PacMax) has access to N different types of pieces while the minimizing agent (PacAdv) has access to only one type of piece. At each round, the PacAdv puts one piece to the right end of a row. After that PacMax chooses a piece to put on the left side of that row.

Assume we run alpha-beta pruning on the game tree described above for one turn.

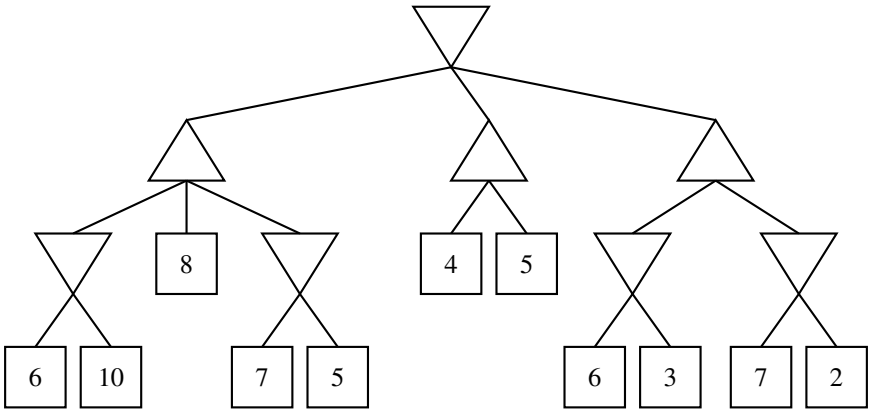
9.1) (2pts) If $M = 5$ and $N = 3$, then what are the upper and lower bounds on the number of leaf nodes we can prune?

8 and 0 respectively. See 9.2 for justification.

9.2) (1pts) If we choose $M = 6$ and $N = 10$, then what is the upper bound on the number of leaf nodes pruned?

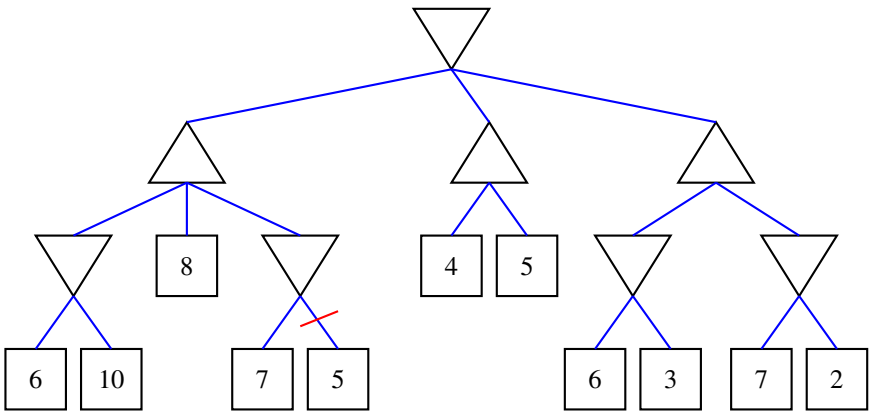
45. For both 9.1 and 9.2 the logic is the same. In minimax, it's always possible that none of the nodes can be pruned, so the minimum possible pruning is 0.
The maximum pruning for a depth 2 minimax is that except we need to see all the leaf nodes of the first branch of the root, for each other branch of the root, we prune all but the first leaf node. That is, we prune $(M-1)(N-1)$ nodes. Plug this in for your value of M and N.

Now let's consider the following game tree:



9.3) (1pts) Assuming that we visit branches from left to right, if we perform alpha-beta pruning how many leaf nodes will be pruned?

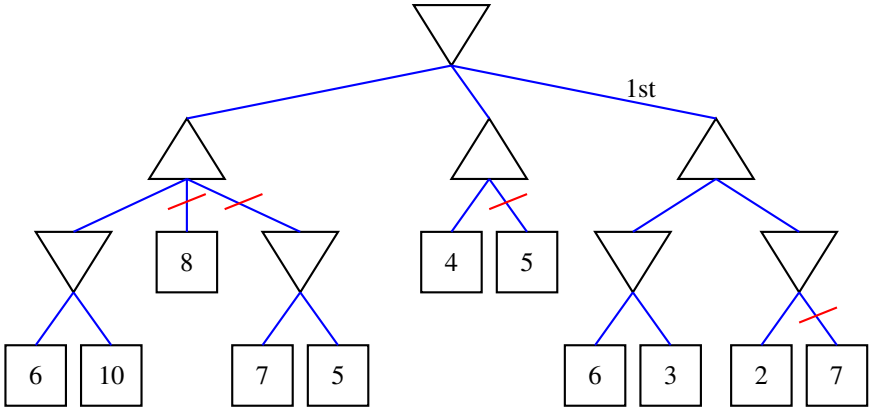
1. Only the 5th leaf node (counting from the left.)



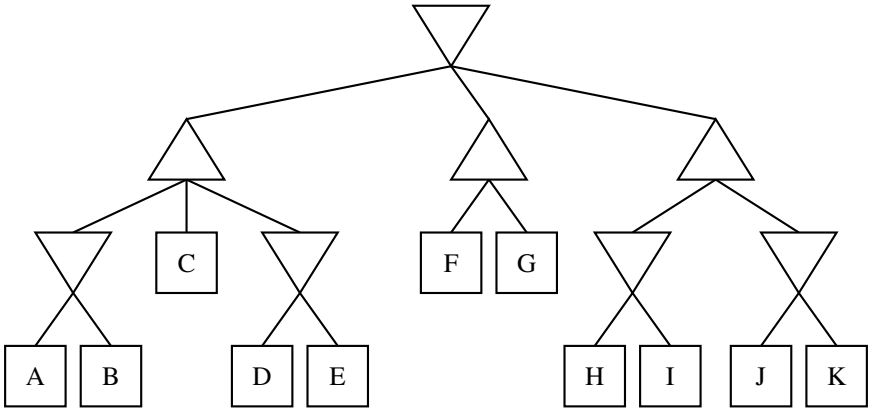
9.4) (2pts) Now let's consider a slightly more involved case. For this part, assume again that children of maximizing nodes are

visited from left to right. If we re-order the branches of minimizing nodes in order to maximize the number of leaf nodes pruned, what is the number of leaf nodes we can now prune? Assume that we do not prune on equality.

5. The right branch of the root contains the value that got propagated to the root, so let's visit that branch first. Note that this doesn't mean that visiting other branches first cannot prune maximally. For simplicity, we are just giving one ordering. In the right branch, swap "7" and "2" so that "7" gets pruned. It doesn't really matter whether we visit the middle branch next or the left branch. In the middle branch, we can prune "5" since upon seeing "4", we know that the maximizer mode is at least a 4, so the root would not choose this branch. We cannot prune more than this. In the left branch, after getting the value of the first minimizer (6), we know that the maximizer mode is at least a 4, so the root would not choose this branch. Hence, we are not checking the 3 leaf nodes corresponding to "8", "7" or "5". If we can reorder branches of max nodes, we could be able to prune 4 leaf nodes, but that's not what we can do for this problem. No reordering of branches of min nodes can prune more than these 3 leaf nodes.



Consider the tree from Question 9.3, in which we have shuffled the leaf node values so that we check the least number of nodes. The structure of the tree remains unchanged. The new ordering of the leaf node values is $A, B, C, \ldots$, while in the previous question the ordering was $6, 10, \ldots$. Assume again that we do not prune on equality and that we traverse from left to right.



9.5) (2pts) Consider all these possible aforementioned orderings. Which of the following are the possible root node values?

A. 2
B. 3
C. 4
D. 5
E. 6
F. 7
G. 8
H. 10

4,5,6

The maximal pruning can have E, G, J and K pruned.
The root is selecting the minimum in the three maximizers, so one of the maximizers must have the result, and the other two maximizers are larger. If that maximizer is the middle or right one, the result could be 3 (but not 2, since the maximizer is never choosing 2). But in this case, we cannot prune node E. The smallest result is therefore 4, where we have min(A, B), C, and

min(D, E) each taking a value from 2, 3 and 4.
The largest result is 6. The right branch of the root must be smaller than the maximum leaf in that branch, since the maximum leaf cannot get selected by either 2 minimizers in that branch. The best we can have is the second largest in the branch. Let one of F or G is large and C is large. If the right branch of the root has result > 6, it has two of 7, 8, 10 in the branch, and one of the left/middle branch of the root would have max 6. Else, the right branch of the root has max 6. So for the root, 7, 8, and 10 are impossible.

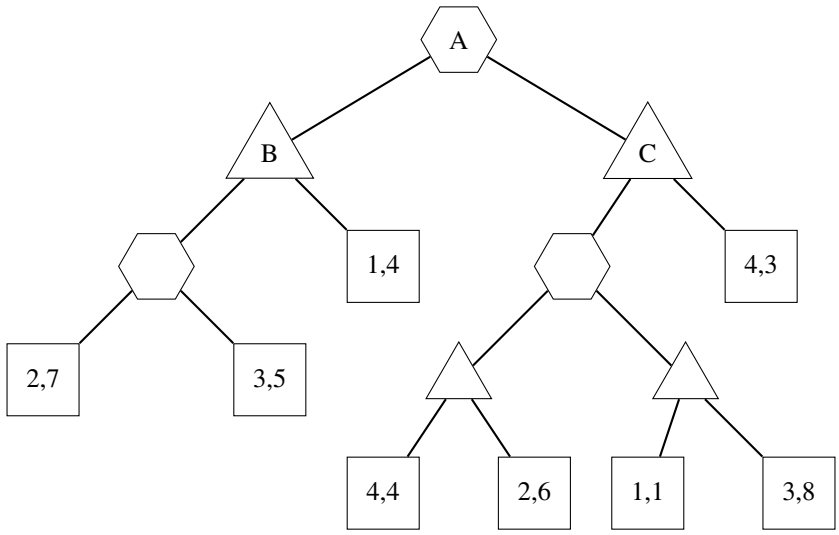
9.6) (3pts) If the value at the root is 4 for the new ordering then which of the following leaf nodes are going to have a value less than or equal to 5

A. A	E. E	I. I
B. B	F. F	J. J
C. C	G. G	K. K
D. D	H. H	L. None

C,D

For root = 4, it's sufficient and necessary to have one of min(A, B) and C to be "4", and the other is "2" or "3" while D also takes "2" or "3". So *C* and *D* are guaranteed to be ≤ 5, but *A* and *B* can take any value as long as one of them is < 5. All other nodes may take values > 5.

Assume that PacMax knows PacAdv's utility function but PacAdv does not know PacMax's utility. Now PacMax is no longer interested in matching PacAdv's move on the same row. On the contrary, Pacmax wants to put as many pieces on the board as possible. Find the PacAdv's and PacMax's utilities on the following tree. If there is a tie choose the leftmost branch. Hexagons denote when PacAdv is deciding, triangles denote Pacmax's decision nodes and squares denote the leaf nodes. The format in all nodes is a tuple ($u_{\text{PacAdv,PacMax}}$), where u denotes the utility.



9.7) (2pts) Write down the utilities at node B as a tuple ($u_{\text{PacAdv,B}}, u_{\text{PacMax,B}}$).

(3,5)

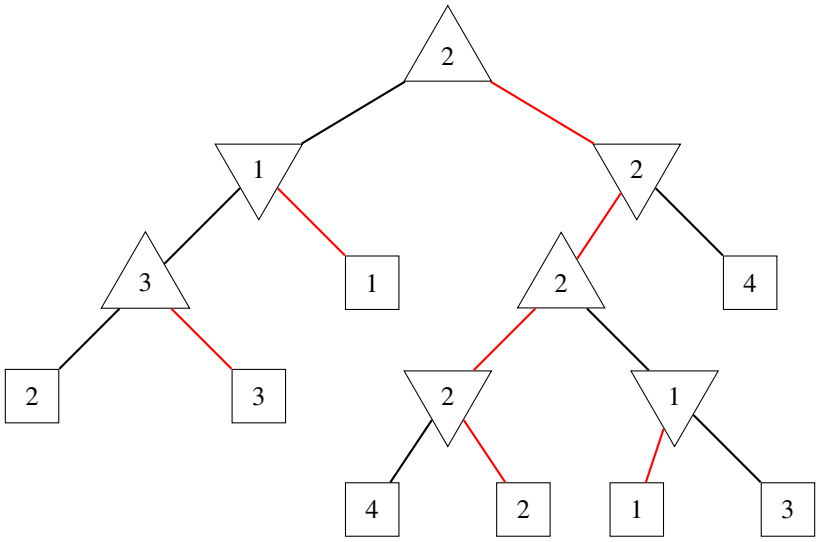
9.8) (2pts) Write down the utilities at node C as a tuple ($u_{\text{PacAdv,C}}, u_{\text{PacMax,C}}$).

(2,6)

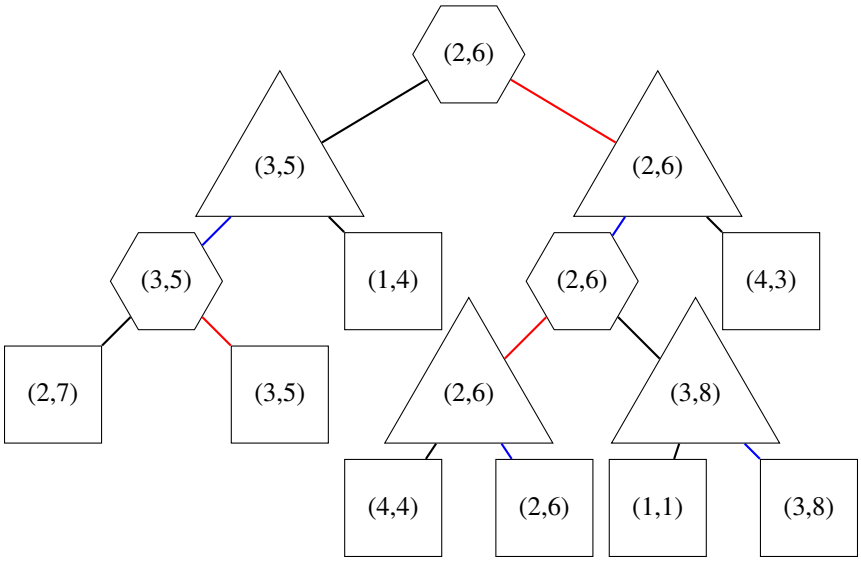
9.9) (2pts) Write down the utilities at node A as a tuple ($u_{\text{PacAdv,A}}, u_{\text{PacMax,A}}$).

(2,6)

Justification for all: In PacAdvs' perspective, the tree is a minimax, as follows:



The red branches are the ones being selected by the agents.
PacMax knows this, so all he need to do is to maximize his utility with the information of the branches that are going to be selected by PacAdv.



Essentially, to solve this problem, we first solved a minimax, then solved a general-sum game where one of the agent's moves are known. You might have observed some peculiar behaviors of this tree: despite both agents being rational, sometimes PacAdv ends up selecting a branch that does not maximize his own utility, but maximizes PacMax's utility; sometimes PacAdv selects a branch that minimizes both his own utility and PacMax's. That's a result of not having complete information.