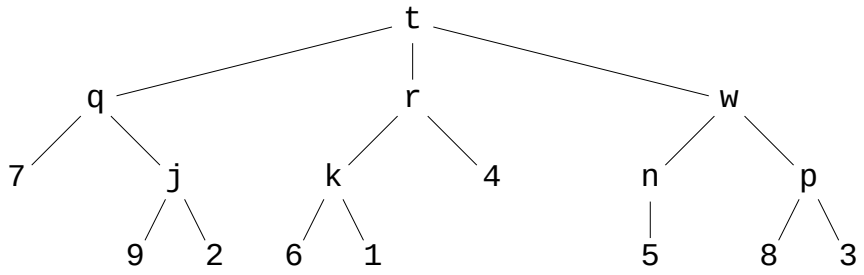
cmput 355 2025 practice questions 3

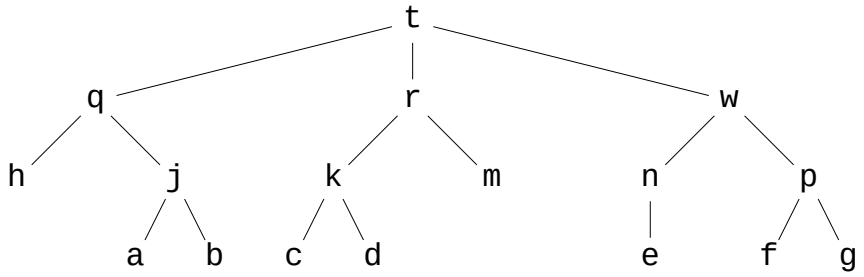
1. One measure of the effectiveness of a particular 2-player-game strategy is its average case performance. Another measure is worst-case performance. a) Explain how minimax is a worst-case performance measure. b) Explain why we learned minimax instead of average case analysis.

2. a) In `alphabeta.py`, how is the depth parameter used? b) In `negamax.py`, how is the depth parameter used?

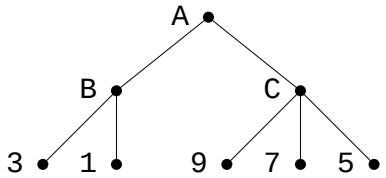
3. Give the minimax value for each non-terminal node.



4. a) For this game tree, assuming children are ordered left to right, list nodes in the order that dfs learns minimax values. b) Repeat if children are ordered right to left.



5. In each question, all node values are for MAX. Unless otherwise stated, MAX plays first.



a) For each node in the game tree, give the minimax value.

b) Repeat a) if MIN plays first (node values are for MAX).

c) For each node in the game tree, give the negamax value.

d) Repeat c), but first negate all leaf values.

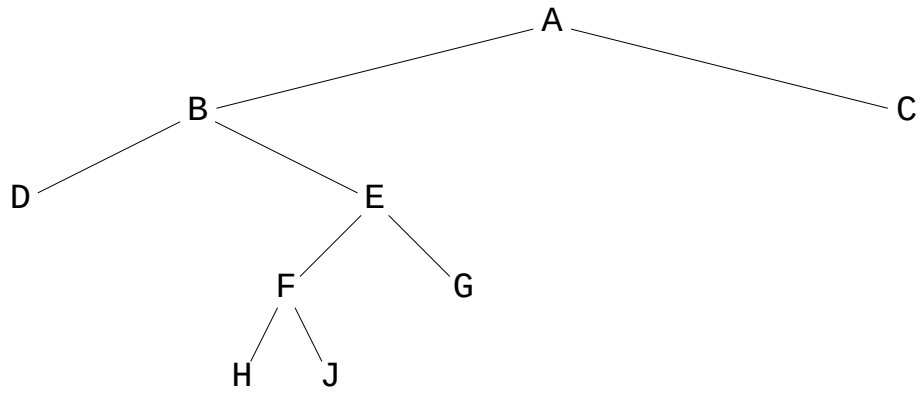
a) A ___ B ___ C ___

b) A ___ B ___ C ___

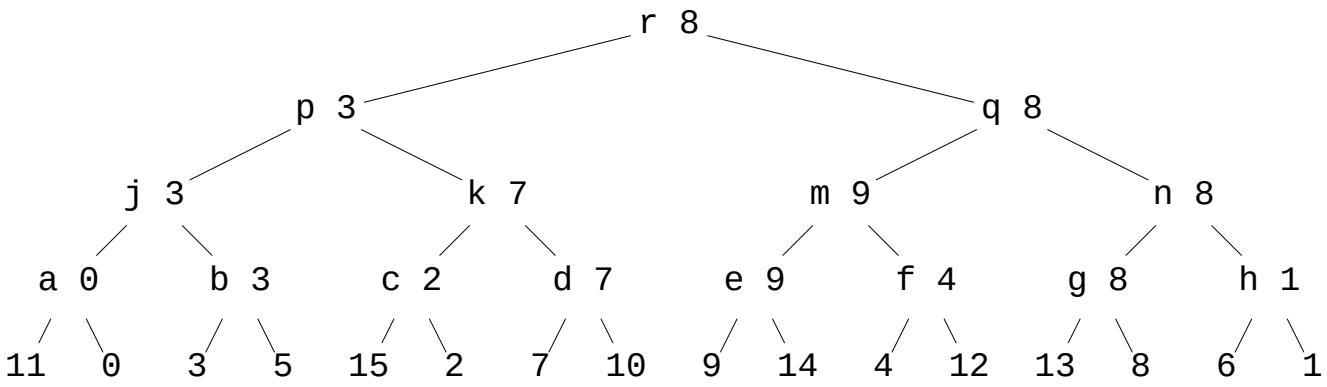
c) A ___ B ___ C ___

d) A ___ B ___ C ___

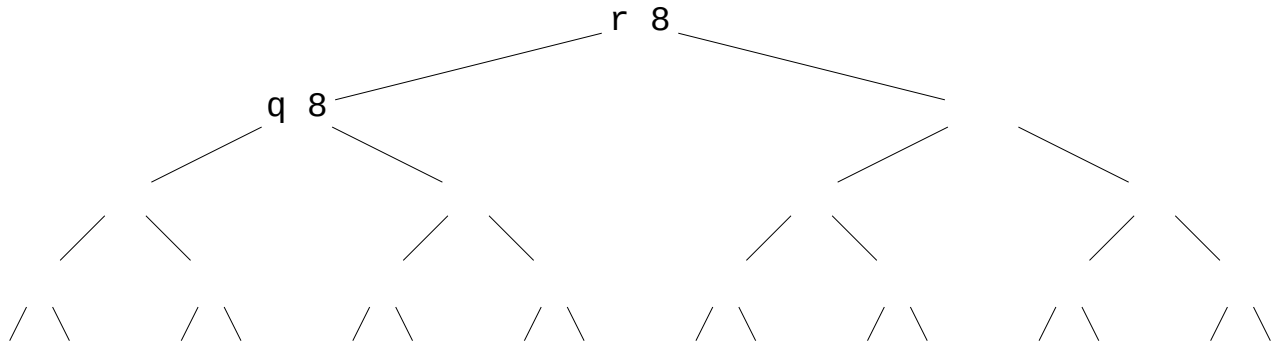
6. Complete the following table. For each node when it is first reached in alphabeta search, show the path to the root and each already-searched move option on this path.



node	path-to-root	already-searched move-options on path-to-root
A	(A)	--
B	(B, A)	--
D	(D, B, A)	--
E	(E, B, A)	B-D
F	(F, E, B, A)	B-D
H	(H, F, E, B, A)	B-D
J	(J, F, E, B, A)	B-D, F-H
G	----------	----------
C	----------	----------

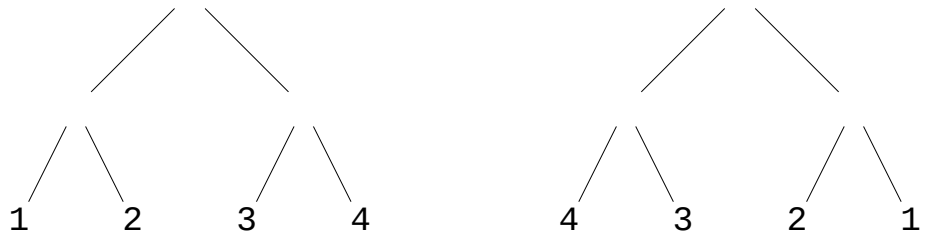


7. Below, redraw the above minimax tree so that each node's best move is its left child. E.g. at `r`, MAX's best move is to q (MAX minimax value 8), so the left child of `r` is `q` (as shown). Now you label the rest of the nodes.

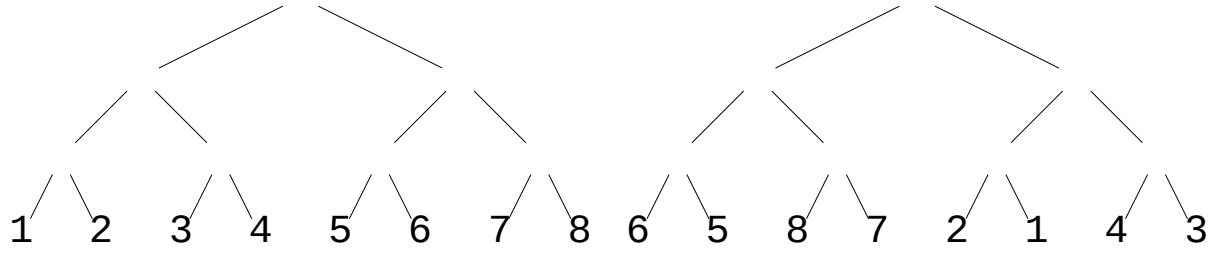


8. a) For each game tree, how many nodes are cut off by alpha-beta search?

   b) After $\alpha\beta$-minimax runs, at each node give what is known about the minimax value there, e.g. 11, ≥ 3, ≤ 7, or ? if nothing is known.



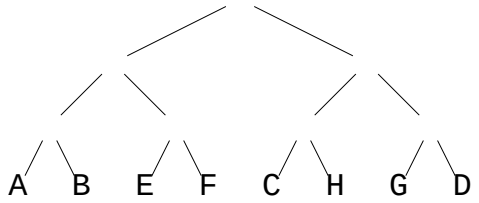
9. Repeat the previous question for these game trees.



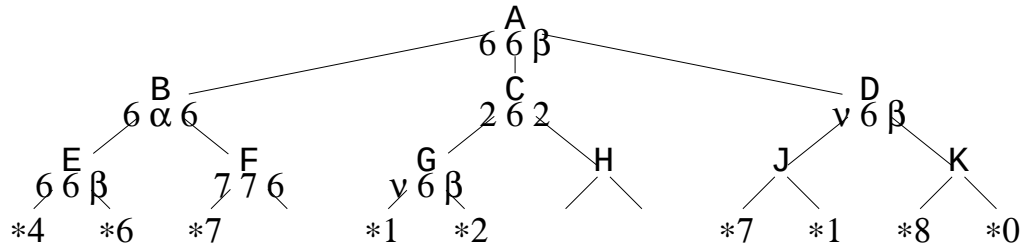
10. (a) Repeat the previous two questions if we change all leaf values to 1 and we use the test `alpha > beta ?` to check for cutoffs.

    (b) Repeat the previous two questions if we change all leaf values to 1 and we use the test `alpha >= beta ?` to check for cutoffs.

11. All leaf nodes have value 1. Which leaf nodes are reached in an alphabeta search with cutoff test $\alpha \geq \beta$?



12. This alphabeta search has just reached J. At each node, values are shown: minimax, alpha, beta. In order, in the rest of the search, at each non-leaf node where a change is made, show current mmx, alpha, beta values. We have shown you change 1 (there might be fewer than 8 changes).



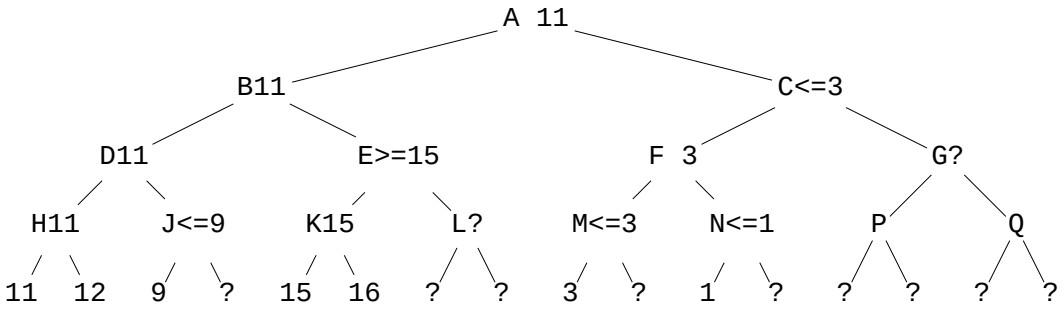
change	node	minimax	alpha	beta	change	node	minimax	alpha	beta
1	J	-	6	-	5	---	---	---	---
2	---	---	---	---	6	---	---	---	---
3	---	---	---	---	7	---	---	---	---
4	---	---	---	---	8	---	---	---	---

13. Fill in the blanks (missing code from `negamax.py`).

```
def negamax(d, T, V, v): # leaf scores for player-to-move
  if isTerminalNode(v,V): return V[v]
  val = NEGINF
  for c in T[v]: # for each child c of v

    val = max(__________________ , ________________________)
  return val
```

14. In this alphabeta search, right subtrees of J,E,M,N,C were pruned. Explain the reason for each prune by filling in the blanks. We have given the answer for node J.



J: on path-to-root J-D-B-A, MAX has move option D-H so alpha >= 11,
MIN has move option J-9 so beta <= 9, so alpha >= beta

E: on path-to-root __________, MAX has move option _________ so alpha ____
MIN has move option _________ so beta _____ so alpha >= beta

M: on path-to-root __________, MAX has move option _________ so alpha ____
MIN has move option _________ so beta _____ so alpha >= beta

N: on path-to-root __________, MAX has move option _________ so alpha ____
MIN has move option _________ so beta _____ so alpha >= beta

C: on path-to-root __________, MAX has move option _________ so alpha ____
MIN has move option _________ so beta _____ so alpha >= beta

15. Show the output when `negamax` is called on the game tree in question 5.

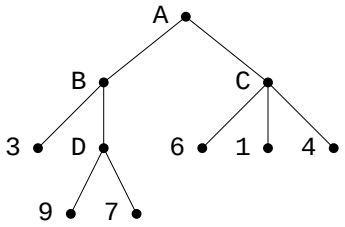
```
def negamax(d, T, V, v): # leaf scores for player-to-move
  print(d*'  ', v)
  if isTerminalNode(v,V):
    val = V[v]; print(d*'  ', v, 'leaf', val); return val
  val = NEGINF
  for c in T[v]: # for each child c of v
    val = max(val, -negamax(d+1, T, V, c))
  print(d*'  ', v, val)
  return val
```

16. Run `alphabeta.py` on `t6.in`: it return a minimax value of 8 (last line of output: `A 8 8 999`). But run `negamax.py` on `t6.in`: it returns a minimax value of -1 (last line of output: `A -1`).

    a) Why did `negamax.py` not print out the alpha and beta values?

    b) Why did the two programs return different minimax values for the above input? Is one of them wrong? Explain.

17. The root is a MAX node. Leaf scores are for MAX. Give minimax values below.

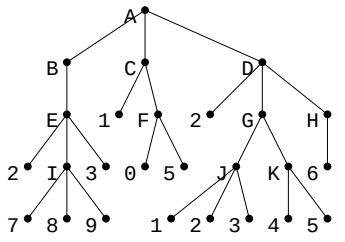


    A ___ B ___ C ___ D ___

18. Repeat the previous question if leaf scores are for MAX but root is MIN.

19. Explain how to use `alphabeta.py` to answer the previous two questions.

20. The root is a MAX node. Leaf scores are for MAX. Give minimax values below.



    A ___ B ___ C ___ D ___

    E ___ F ___ G ___ H ___

    I ___ J ___ K ___

21. Repeat the previous question if leaf scores are for MAX but root is MIN.

22. a) For a two-player game, a *principal variation* is a sequence of moves from the start of the game to a terminal position, where each move by each player is best possible. For question 18, give a principal variation. E.g. for question 17, a principal variation is 1.MAX A-B, 2.MIN B-3. Repeat for questions b) 20 and c) 21.

23. Using the blank-to-0, x-to-1, o-to-2 representation, express each board position below as a 9-digit number base 2 and then as a decimal number. Recall that $3^9 = 19683$. Show your work.

```
- - -        - - -        o o o
- - -        - - -        o o o
- - -        x - o        o o o
```

24. In tic-tac-toe assume `x` moves first: thus for any reachable position, we can determine the player-to-move from the number of `x`'s and `o`'s. For each position below, give player-to-move minimax value (win, loss, draw). Use any program from class to answer this question.

```
   a b c
1  x - -    - x -    - - -    - - o    - x o    - - o
2  - - -    - - -    - x -    x - -    x - -    x o -
3  - - -    - - -    - - -    - - -    - - -    - - x
```

25. For this tic-tac-toe position with `x` to play, how many nodes are in a proof tree that shows that `o` can at least draw? Give the number of nodes at each level of the tree: there will be one node at the root, corresponding to the position in the diagram below. (Such a tree will have all `x`-moves from the root, then for each a winning `o`-reply, then all `x`-moves, then for each a winning `o`-reply, and so on.)

```
. x .
. o o
. x .
```

26. a) For this position with `x` to play, draw the next two levels of a proof tree that shows that `x` can at least draw: at level 1 of the tree you can prune isomorphic positions. Below each leaf of your drawing, give the x-minimax value (win/lose/draw) of that subtree.

    b) Repeat the question for a proof tree that shows that `o` can at least draw.

```
o . .
. x .
. . .
```

27. When running `tt.py`, you can call function `see_positions` by typing `#`. Here is some output:

```
3139 psns with x to move            2907 psns with o to move
occupied cells   number psns        occupied cells   number psns
    0               1                   0               -
    1               -                   1               9
    2               72                  2               -
    3               -                   3               252
```

    a) Explain why the numbers at levels 0,1,2,...start 1,9,72,...: where have we seen this pattern before, and what does it represent?

    b) From a), we might expect that the number at level 3 will be 504, but it is 252: why?

    c) In `tt.py`, solving empty board returns `x minimax result 0 nodes 549946`. Explain message.

    d) Modify `tt.py`: in function `negamax`, uncomment line below.

```
for cell in L:
    psn.brd[cell] = ptm
    nmx, c = negamax(use_tt, use_iso, MMX, 0, d+1, psn, opponent(ptm))
    so_far, calls = max(so_far, -nmx), calls + c
    psn.brd[cell] = Cell.e
    # if so_far == 1: break
```

    Now solving returns `x minimax result 0 nodes 94978`. Explain.

28. In `tt.py`, how is `Isos` used?

```
Isos = ((0,1,2,3,4,5,6,7,8), (0,3,6,1,4,7,2,5,8),
        (2,1,0,5,4,3,8,7,6), (2,5,8,1,4,7,0,3,6),
        (8,7,6,5,4,3,2,1,0), (8,5,2,7,4,1,6,3,0),
        (6,7,8,3,4,5,0,1,2), (6,3,0,7,4,1,8,5,2))
```

---

end of practice questions

extra questions (will not appear on the quiz)

1. Using `stp_search2.py`, find all hardest 3×3 solvable STPs. Explain.