**cmput 355 2024  homework 3, with hints**

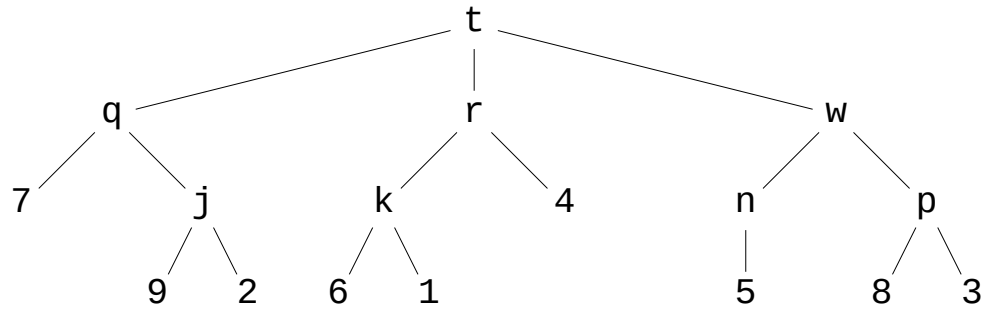
In this course, unless we say otherwise,

when discussing minimax values of game tree nodes, we assume that

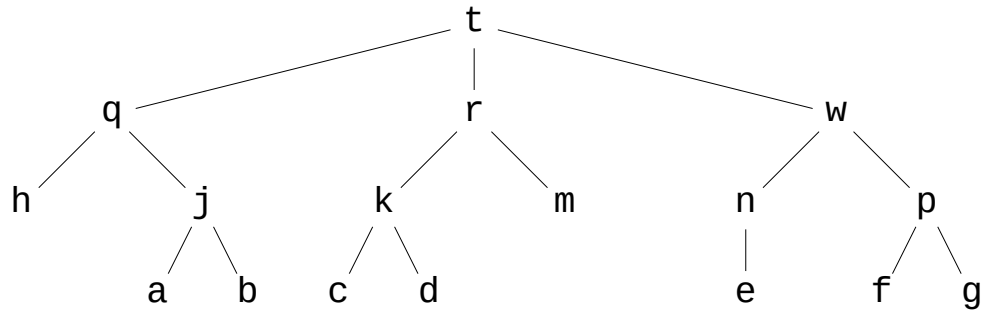
1st-player is MAX, 2nd-player is MIN, terminal nodes scores are for MAX.

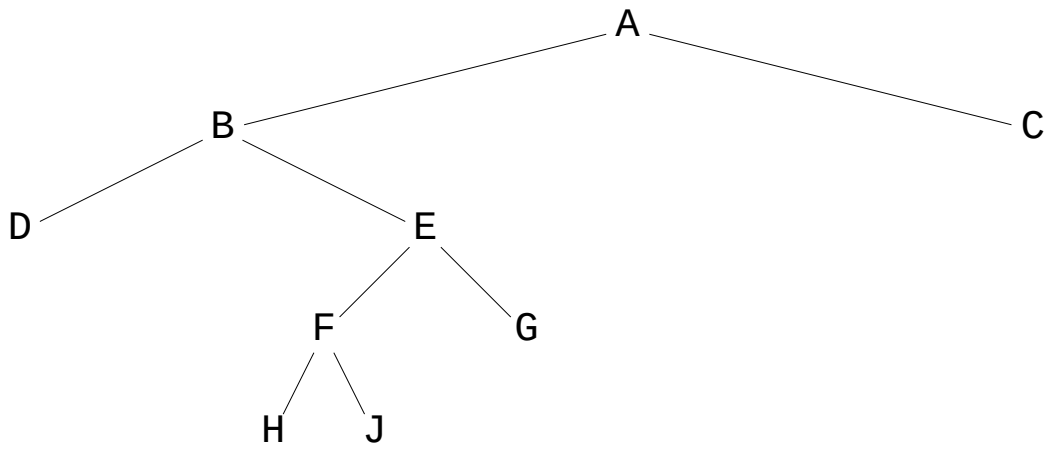
1. One measure of the effectiveness of a particular 2-player-game strategy is its average case performance. Instead, in the lectures, we discussed minimax, which gives us a player's worst (minimum) score over all possible opponent strategies. Why did we learn minimax, instead of learning how to evaluate average case performance?

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



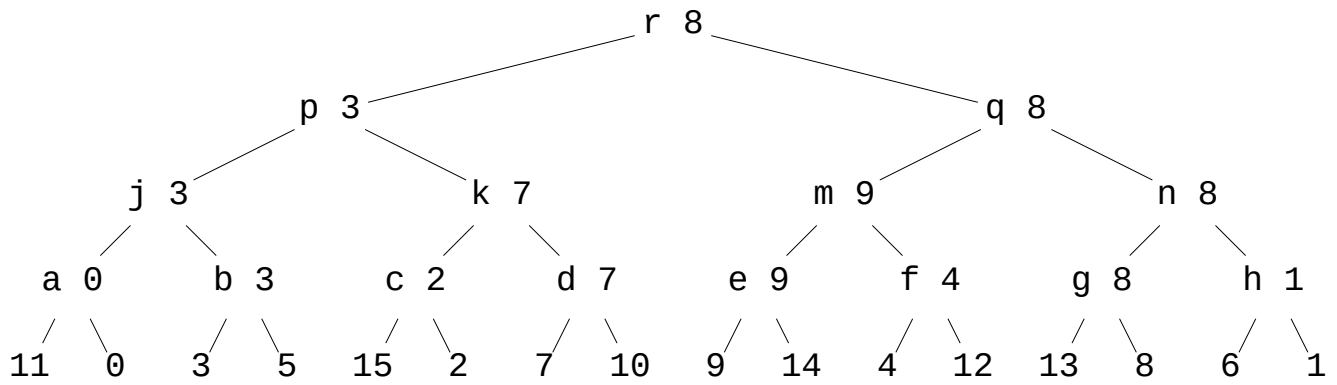

3. For this game tree, list nodes in the order that dfs learns minimax values.



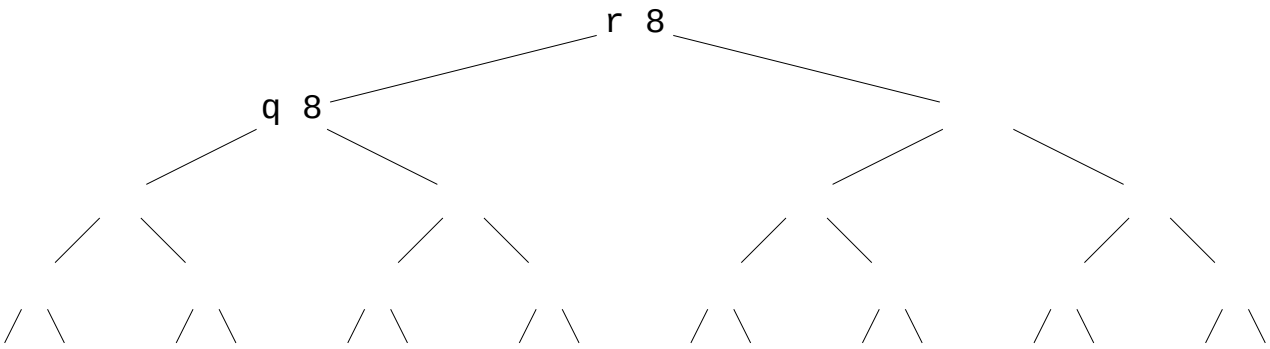



4. Complete the following table. For each node when it is first reached in alphabeta search, show the path to the root and each already-searched move option on this path.

| 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 | ---------- | ---------- |




5. 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.



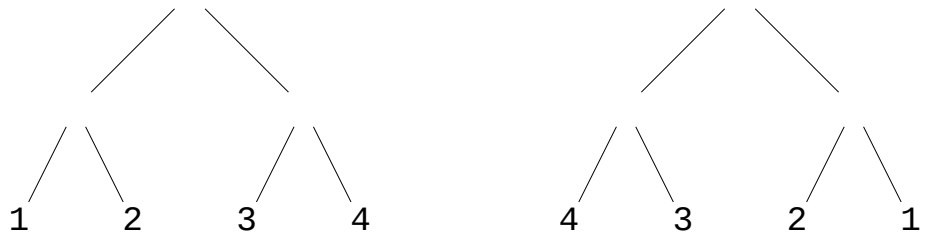

6. Trace the alphabeta example here:

   `https://webdocs.cs.ualberta.ca/~hayward/355/p.pdf`

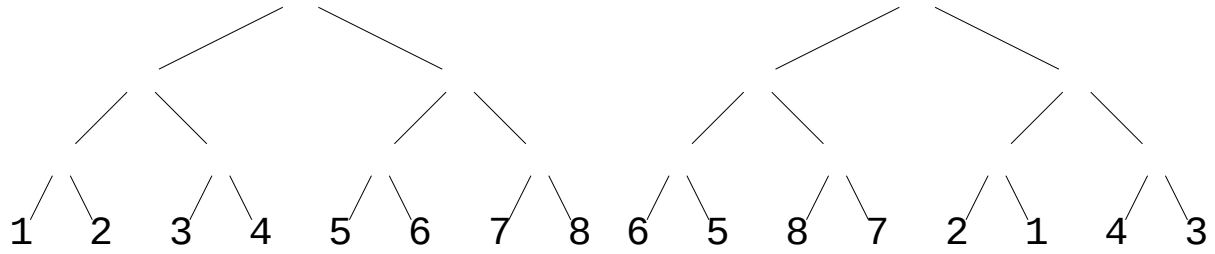
   For each change to a node's minimax, alpha or beta value, explain why that change occurred (each page of the pdf shows one change).

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

   b) At each node, give what is known about the minimax value (e.g. 11, $\geq 3$, $\leq 7$, or ? if nothing is known) after $\alpha\beta$-minimax runs.



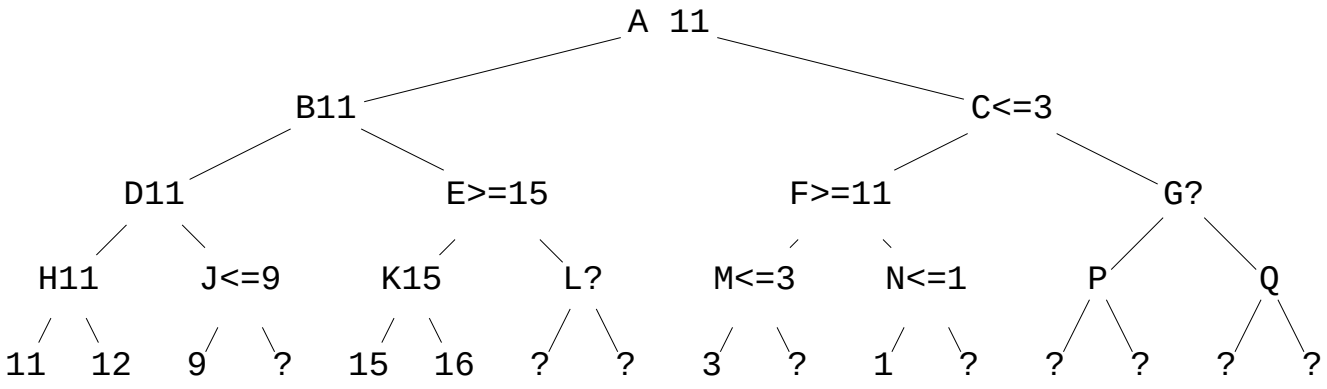

8. Repeat the previous question for these game trees.




9. (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.

10. In this alphabeta search, subtrees are cut off below A,B,C,D,E. Explain the reason for each cut.



11. At right, unscramble these lines from the start of the alphabeta function in `alphabeta.py`. Write line numbers only: indent properly. We have written the first line number for you.

```
ab = alphabeta(d+1, T, V, c, alpha, beta) #1
break                                     #2
def alphabeta(d, T, V, v, alpha, beta):   #3
for c in T[v]:                            #4
if ab > val: alpha, val = ab, ab          #5
if alpha >= beta:                          #6
if isMaxNode(v, d):                       #7
if isTerminalNode(v,V): return V[v]       #8
return val                                #9
val = NEGINF                              #10
```

(3) ___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___ ___

12. Fill in the 2 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
```

13. Show the output when `alphabeta` is called on the game trees in question 6.

```
def alphabeta(d, T, V, v, alpha, beta): # leaf scores for MAX (root player)
  print(d*'  ', v,'MAX' if isMaxNode(v,d) else 'MIN','?',alpha,beta)
  if isTerminalNode(v,V):
    val = V[v]; print(d*'  ', v, 'leaf', val); return val
  if isMaxNode(v, d):
    val = NEGINF
    for c in T[v]:
      ab = alphabeta(d+1, T, V, c, alpha, beta)
      if ab > val: alpha, val = ab, ab
      if alpha >= beta:
        print((d+1)*'  ','prune rem. ch. of', v); break
    print(d*'  ', v, val, alpha, beta)
    return val
  val = INF
  for c in T[v]:
    ab = alphabeta(d+1, T, V, c, alpha, beta)
    if ab < val: beta, val = ab, ab
    if alpha >= beta:
      print((d+1)*'  ','prune rem. ch. of', v); break
  print(d*'  ', v, val, alpha, beta)
  return val
```

14. Show the output when `negamax` is called on the game trees in question 6.

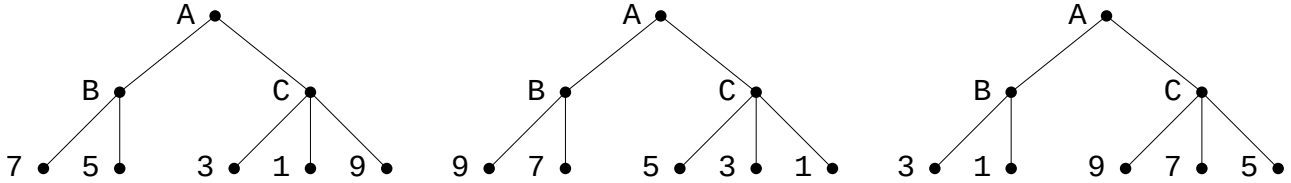
```
def negamax(d, T, V, v): # leaf scores for player-to-move
  print(d*'  ', v)
  if isTerminalNode(v,V):
    val = V[v]; print(d*'  ', v, 'leaf', val); return val
  val = NEGINF
  for c in T[v]: # for each child c of v
    val = max(val, -negamax(d+1, T, V, c))
  print(d*'  ', v, val)
  return val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

15. When I run `alphabeta.py` on `t6.in`, it return a minimax value of 8 (last line of output: `A 8 8 999`). But when I 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.

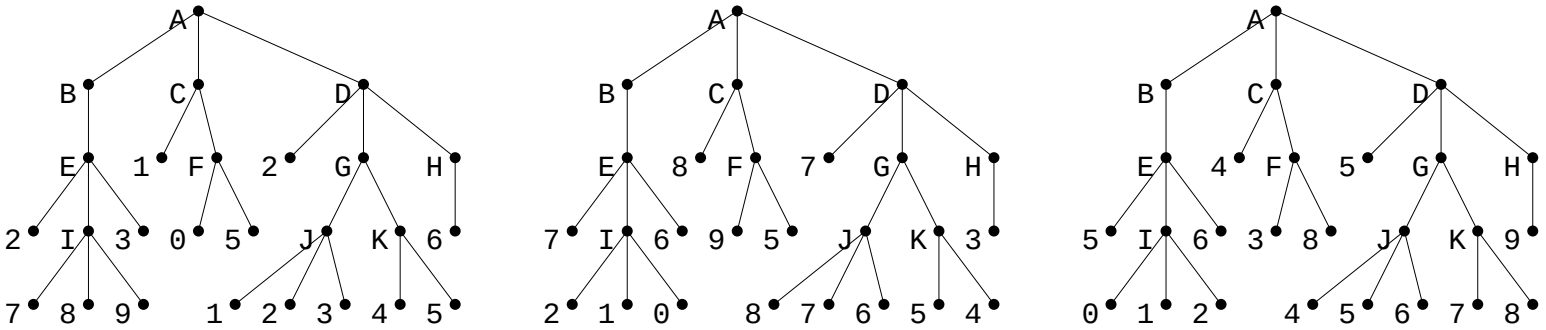
16. The root of each tree is a MAX node. For each leaf, the minimax value for MAX is shown. For each tree, give the minimax value for MAX for each node A,B,C. Answer like this: A 2, B 8, C 4




17. Repeat the previous question with this change: the root of the tree is a MIN node, each child of the root is a MAX node, each grandchild of the root is a MIN node, each leaf is labelled with the score for MAX.

18. Explain briefly how to use class program `alphabeta.py` to answer each of the previous two questions. (One of these cases is trivial.) Explain clearly.

19. The root of each tree is a MAX node. For each tree, for each leaf, the minimax value for MAX is shown. In alphabetic order, give the minimax value for MAX for each node A B C ... K. Answer like this: 3 5 7 8 9 0 1 3 3 6 6 .




20. Repeat the previous question with this change: the root of the tree is a MIN node, each child of the root is a MAX node, each grandchild of the root is a MIN node, each leaf is labelled with the score for MAX.

21. (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 19, give a principal variation. E.g. for question 16, the principal variation for the trees, from left to right, are respectively

    1.MAX to B, 2.MIN to 5;  1.MAX to B, 2.MIN to 7;  1.MAX to C, 2.MIN to 5.

    (b) Repeat (a) for question 20.

22. (Recall: In alphabeta search, at each MAX node, alpha tracks the best (so, maximum) score so far that MAX can get on the path from the current node to the root. Similarly, at each MIN node beta tracks the best (so, minimum) score so far that MIN can get on the path from the current node to the root.

    Assume that $x$ is a MIN node with current beta value $\beta$ obtained by picking a child $y$ of $x$. Assume that minimax search now starts at child $z$ of $x$, so a MAX node. If at $z$ the alpha value $\alpha$ grows so that $\alpha \geq \beta$ then we can stop searching at $z$, since MIN can always prefer child $y$ (with value $\beta$) to child $z$ (with value $\geq \alpha \geq \beta$).

    **( fill in the blanks)** Assume that $x$ is a MAX node with current alpha value $\alpha$ obtained by picking a child $y$ of $x$. Assume that minimax search now starts at child $z$ of $x$, so a _____ node. If at $z$ the beta value $\beta$ drops so that $\beta \leq \alpha$ then we can stop searching at $z$, since

---

23. In tic-tac-toe, assume that a player scores 1, $-1$, 0 respectively for a win, loss or draw. For the three non-isomorphic first moves in tic-tac-toe, what is the first player's minimax value for each move? (Use any of the programs from class to answer this question.)

24. Recall that a proof tree is a subtree of the minimax tree that is sufficient to prove an upper bound or lower bound on the minimax value. i) For `x` to play from the postion below left, give a proof tree that shows that `x` can win. ii) For `x` to play from the postion below middle, give a proof that shows that `o` can win. (Give all possible first moves for `x`, then for each give a winning reply for `o`, then for each give all next moves for `x`, then for each a winning reply for `o`, etc.) Check your answer using a program from `/simple/ttt/` .

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

25. a) Modify the win condition for tic-tac-toe: 1st-player `x` wins with 3-in-a-row, otherwise 2nd player `o` wins. (It doesn't matter whether 2nd player gets 3-in-a-row.) What is the first-player minimax value for this game (win, lose)? For this game, give a proof tree for `x` to play from the position above right.

    b) Repeat a) for this win condition: 2nd-player `x` wins with 3-in-a-row, otherwise 1st-player `o` wins.

**hints**

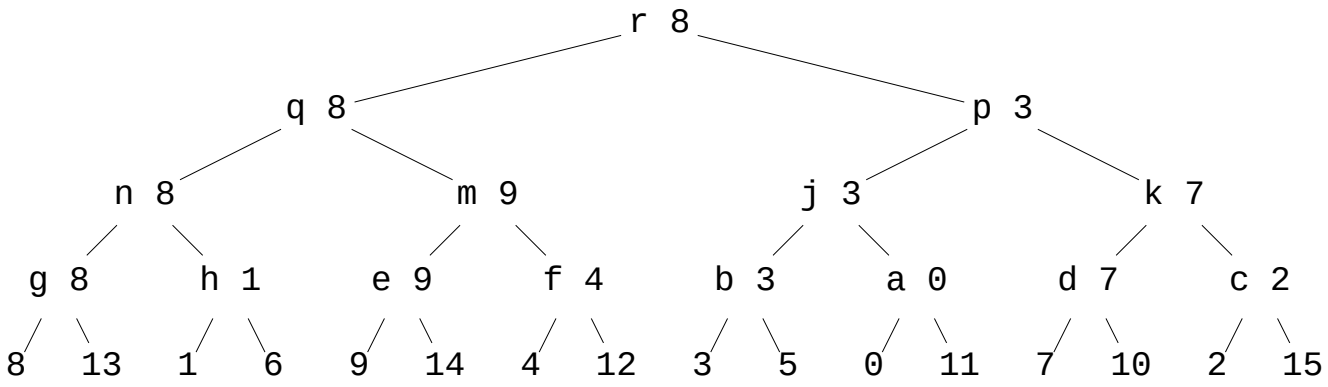
1. Minimax gives us a guarantee (we are guaranteed to score at least the minimax value, which is always good to know. Also, minimax is easier to compute than average case performance. We can find the minimax value of a strategy in time that proportional to the number of nodes in the game dag. Average case analysis usually requires more computation time.

2. j 9  q 7  k 6  r 4  n 5  p 8  w 5  t 7

3. dfs postorder: h a b j q c d k m r e n f g p w t

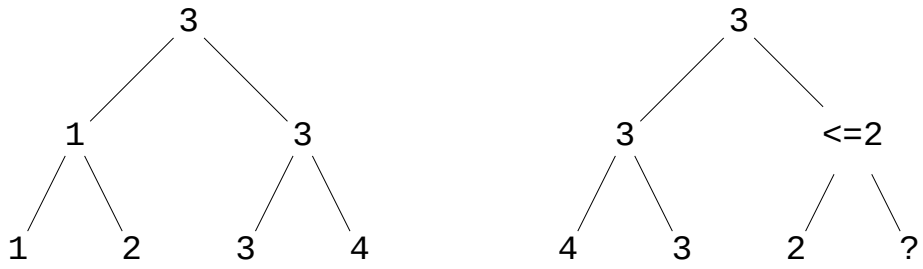
4. `G  (G, E, B, A)  E-F, B-D`
   `C  (C, A)  A-B`

   (To check these answers, you can give each leaf a value and trace alphabeta search.)

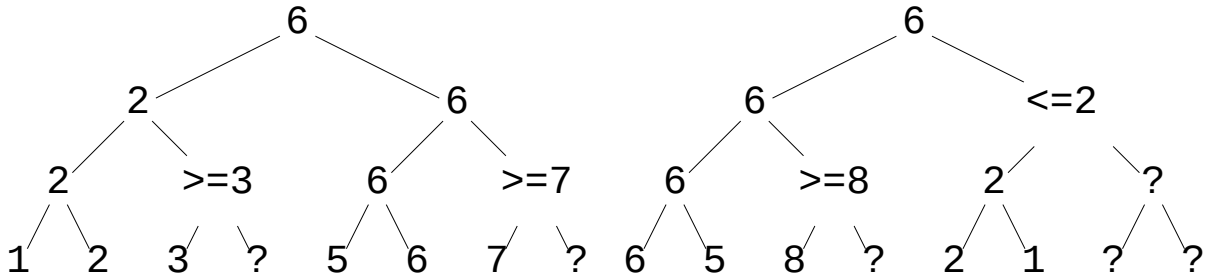
5. 


6. see the pdf

7. a) 0 (left tree) 1 (right)  b) see below




8. a) 2 (left tree) 4 (right)  b) see below




9. (a) 0 cutoffs  (b) same number of cutoffs as with perfect move ordering

10. The first cut occurs at the MIN node that has children labelled `9, ?`. Call this node $v$. In the dfs alphabeta execution, after backing up from the left subtree (terminal node, MAX score 9), $v$ has $\alpha = 11$ (from parent of $v$, MAX can move to the left subtree) and $\beta = 9$ (from $v$, MIN can move to the left subtree). Whenever a node has $\alpha$ greater than $\beta$, we know neither player will ever move there, so execution backtracks from $v$ to the parent of $v$, and the right subtree of $v$ is never explored (which is why the right child of $v$ has label `?`).

11. See the class github repo.

12. See the class github repo.

13. From the github code repo, run `abterse.py` with input `t9.in` and `t8.in`. One of your outputs should look like this:

```
A MAX ? -999 999
  B MIN ? -999 999
    D MAX ? -999 999
    D leaf 4
    E MAX ? -999 4
    E leaf 3
  B 3 -999 3
  C MIN ? 3 999
    F MAX ? 3 999
    F leaf 2
    prune rem. ch. of C
  C 2 3 2
A 3 3 999
```

14. Run `negamax.py` on `t9.in` and `t8.in`. One output looks like this:

```
A
   B
     D
     D leaf 4
     E
     E leaf 3
   B -3
   C
     F
     F leaf 2
     G
     G leaf 1
   C -1
 A 3
```

15. (a) `negamax.py` is a version of minimax, not a version of alpha-beta minimax: it does not use alpha/beta values (so it does not cut off any subtrees).

    (b) `alphabeta.py` assumes that all leaf node scores are for MAX. `negamax.py` assumes that all leaf node scores are for the player-to-move, which in t6.in will always be MIN. In `t6nega.in`, the leaf scores have been converted for player-to-move. When you run `negamax.py` on `t6nega.in`, you get the same minimax value as when you run `alphabeta.py` on `t6.in`.

16. Run a minimax or alphabeta or negamax program from the class repo to check your answers (you will need to first create the input files).

17. see the next answer

18. you can change the input file in one of two ways: a) add a new root, make its only child A b) keep the tree but negate the values of all leaves; run the program; negate the answer

19. Run a minimax or alphabeta or negamax program from the class repo.

20. See the answer to 18

21.

22.

23. hint given in question

24. hint given in question

25. modify the winning condition part of a ttt solver from class to check your answer