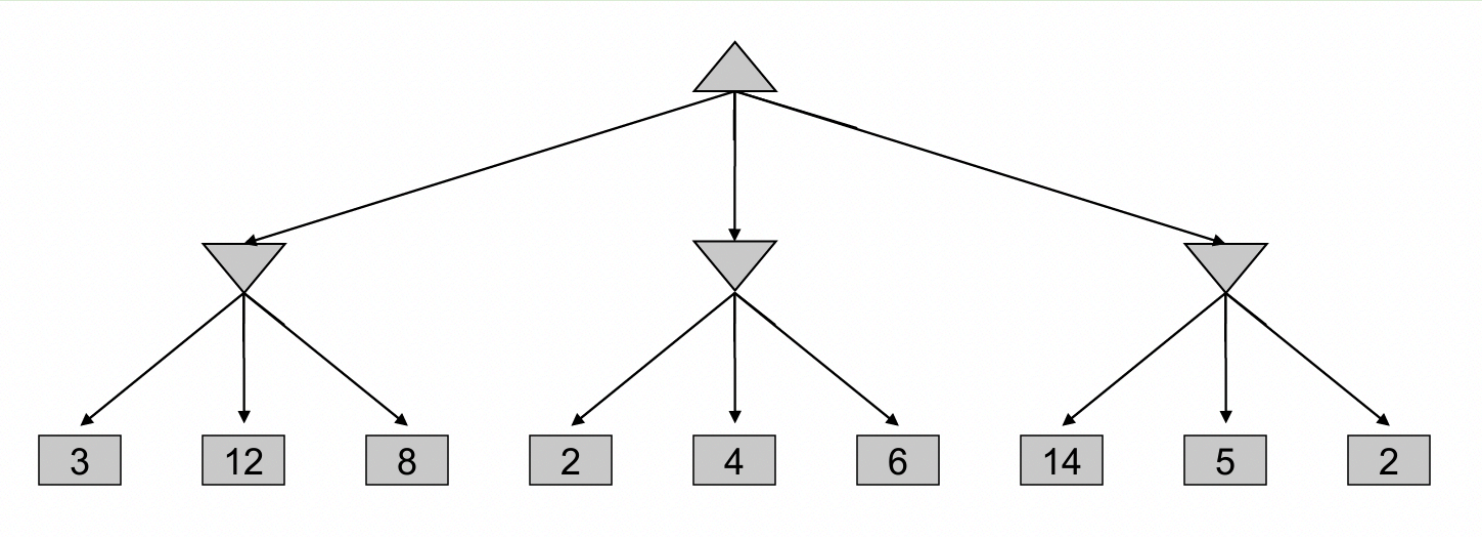
Adversarial Search - Alpha Beta Pruning	start time:

Before you start, share this document with your team member(s) and then complete the form below to assign the role of speaker.

Team Role	Team Member
Speaker: shares your team's ideas with the class.	Arogya, Makenna, Shreeya

A. Pruning Early	start time:



Q1. Run a complete Minimax search from left leaf to right leaf.

				3				
	3			2			2	
3	12	8	2	4	6	14	5	2

Q2. Do you think there are some leaves that can be skipped (pruned) during the Minimax search?

Yes, we can cut the 4 & 6 branches because we are looking for the smallest value on this current node, and we know that the value has to be at most 2, which does not qualify for the level above that (at least 3).

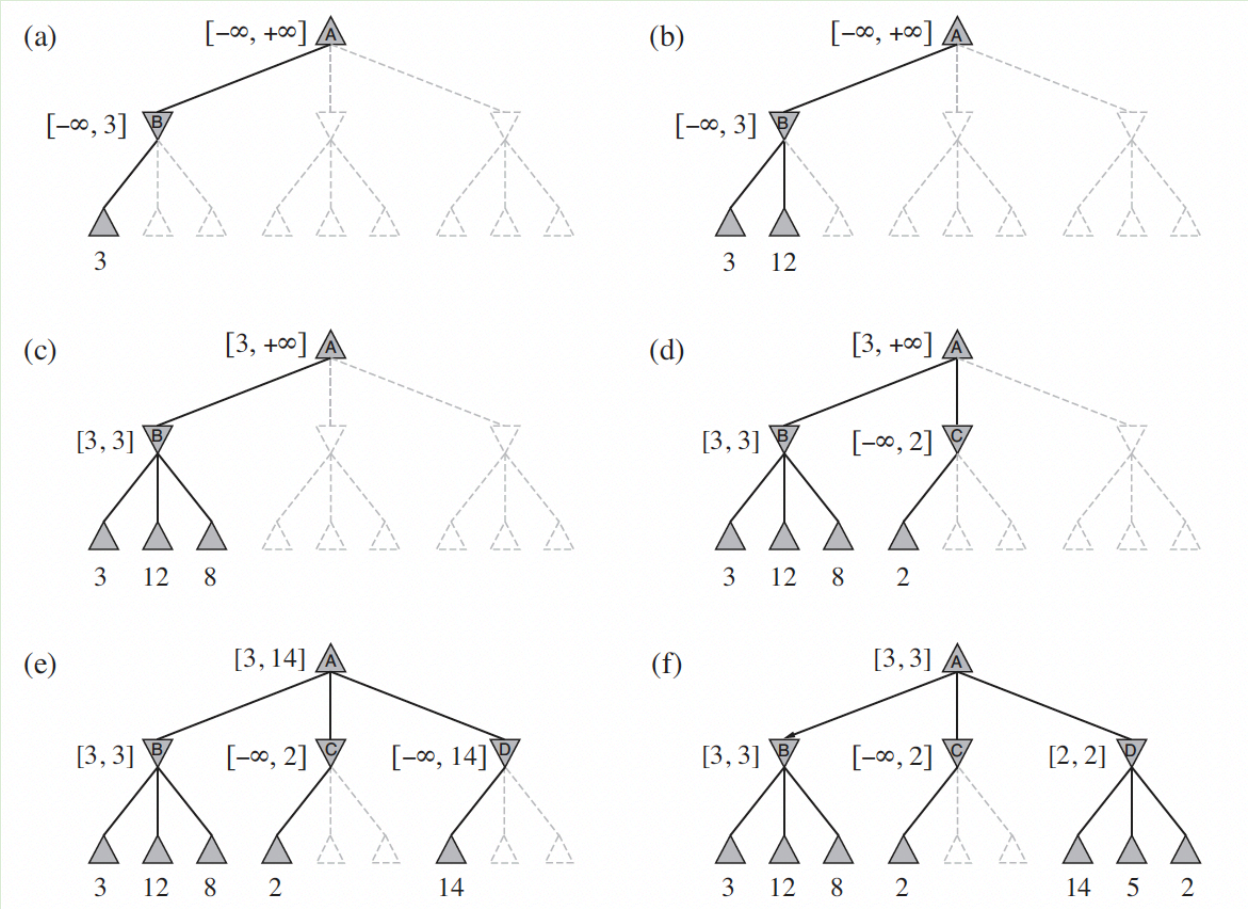
Q3. Can you provide some rules to prune the trees so that we can skip branches of trees during the search?

Current value: Current root value

Any of the nodes that's value is either larger or smaller than the current value

When exploring from left to right, in max, if we already have something maxer than current value then we can skip that branch entirely.

B. Alpha Beta Pruning	start time:



During the search in the game tree, we could not only update the minimax value, but also could estimate the bound of the value. If v is the **final** minimax value at the node after searching all its children, we have

- a lower bound α, which means v is at least α.
- an upper bound β, which means v is at most β.

Since the search follows DFS, there is only one path from root to consider during the search. We maintain two values during the search:

α = the value of the **best** (i.e., highest-value) choice we have found so far at any choice point along the path for MAX. (Think as at least)

β = the value of the **best** (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN. (Think as at most)

How They Work:

- α starts at -∞ and is updated as MAX finds a better value, which is larger than current α .
- β starts at +∞ and is updated as MIN finds a better value, which is less than current β.
- Pruning occurs when β ≤ α, meaning the current branch cannot affect the final decision.

In other words, if the bound is (α, β) at current node and the value from current child is v, it means the final value at current node is between α and β. We would like to update the bound based on v:

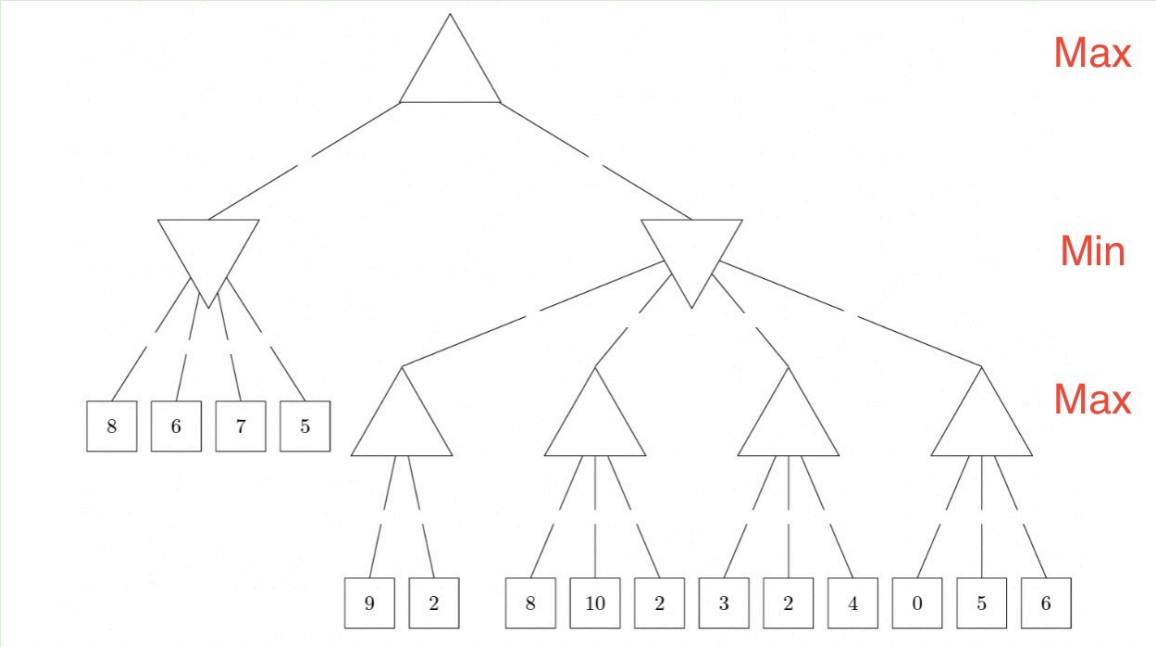
- MAX node:
  - if v <= α, ignore it.
  - if v > α, update α as v. We find a better α.
  - if v > β, prune current branch since the current branch won't be considered by the root.
- MIN node:
  - if v < α, prune current branch since the current branch won't be considered by the root. (Look at the example in tree d above.)
  - if v < β, update β as v. We find a better β.
  - if v >= β, ignore it.

Q4. What are the initial values of alpha and beta?

α = -∞
β = +∞

Q5. Explain how alpha and beta values update from (a) to (f)

a) B's first child is 3 which means at most the value of B will be 3 (beta)
b) B's second child is 12, 12 > 3 and we want beta to be as small as possible, so beta remains the same
c) B's third child is 8, 8 is greater than 3 and we want beta to be as small as possible so we can prune this branch.
   B has no more children, so we know that there is not a node will a smaller value than 3. Thus the alpha should be 3
d) Go to C, C's first child is 2 which means at most the value of C will be 2 (beta). Because 2 is less than 3 and the next level up will use max, we do not need to expand more



Q6. This is the same exercise we had in the previous activity. Can any edges be pruned? Explain.

- 2 in the second branch of the root's second branch can be pruned
- the fourth branch of the second branch of root can be pruned
- *explained in the paper activity*

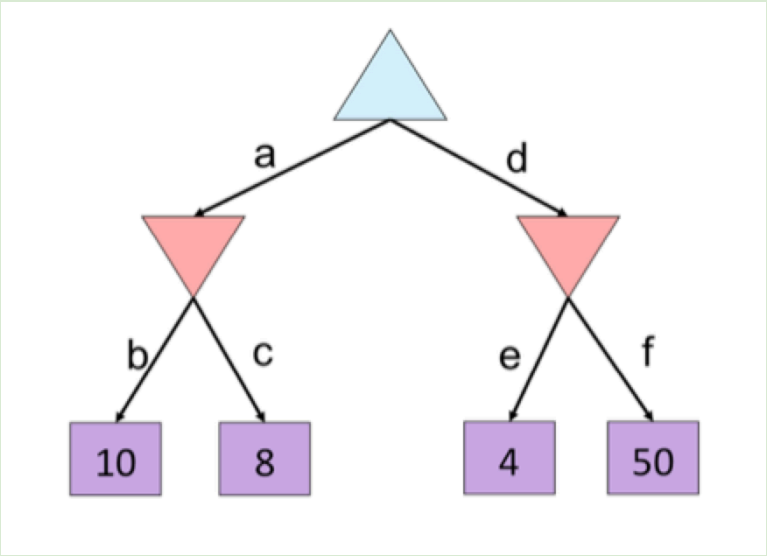
C. Code of Alpha Beta Pruning	start time:

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Examine this graph:



Q7.1. At the start of the algorithm when max-value is called on the root node, what are alpha and beta's initial values?

- [float(-inf) & float(inf)]

Q7.2. There is a for-loop in max-value.

a. List the first method call that the root node will call.

- MAX_VALUE()

b. In that method call, what will alpha and beta's values start as?

- -inf and inf

c. Trace the execution of this method call and what v, alpha, and beta will be changed to.

- alpha will start with -inf
- v becomes the value of the child (10)
- beta would get the value from the min of the child:
  - thus beta would be 10
- [-inf, 10] and v is 10

d. At the end of the method call, what will v, alpha and beta's values be?

- beta will start with 10
  - beta would then be 8 since 8 < 10
- v becomes the value of the child (8)
- alpha would get the value from the min of the child:
  - thus alpha would be 8
- [8, 8] and v is 8

e. Which of these three values are returned to the root node?

- 8 will be returned which is alpha

f. What will the root node's v, alpha, and beta's values be?

- node's v is [8, +inf]
- alpha is 8
- beta is +inf

Q7.3. Repeat for the second iteration of the root node's for-loop.

- alpha will start at 8
- beta will still be +inf
- v starts at 8

- Move to branch d
- Now v becomes the value of the child -> 4
- beta becomes 4

- now the beta value of the right branch (d) is less than the alpha value of the root
- next branch is not visited (pruned)

Q7.4. Which branches were pruned?

- Prune f

Q7.5. What was the final path that will be selected by the minimax algorithm with alpha-beta pruning?

- a - c is selected

Q7.6. Consider what would have happened if the original minimax algorithm was executed:

a. Was the final value for the root node the same, with and without alpha-beta pruning?

- Yes

b. Was the final value for the interior nodes correct, with and without alpha-beta pruning?

- Yes

c. Are you guaranteed that (a) and (b) will be the same / different? Explain your reasoning.

- Yes? maybe no?

D. Some technique details	start time:

Q8. When running alpha-beta pruning starting from the root, which represents the initial state of the game, is the last leaf (node 50) actually expanded? In other words, do you think the code still builds a full size tree when using alpha-beta pruning?

- No, we are pruning the branches based on our alpha beta values of branches we have already seen/visited

Q9. At any point of the game, minimax is called to choose the best action for MAX or MIN. MAX and MIN take turns playing the game. For example, after MAX calls minimax to choose action a, MIN will call minimax search again to choose b or c. Do you think the best choice of MIN was actually already calculated in MAX?

- Yes, without pruning, we know all the best actions in the game. Minimax will find the value of the next child, but the max gets the max of the value in memory and the minimax.

- No with pruning.

Q10. We use a technique called Transposition Table or cache to save already-computed positions to avoid redundant calculations. Please open the tictactoe assignment to learn the decorator cache.

Q11. In alpha-beta pruning, do we need to cache (state, alpha, beta) or just (state)?

- No, we only need (state); state remains the same, alpha and beta changes for every step but they will end up being the same best action as what can be seen from the state.

E. Evaluation/Heuristic Function	start time:

Q12. In Activity 8, we know that if the maximum depth of the tree is m and there are b legal moves at each point, the time complexity of the minimax search is O(b^m). What is the worst case and best case if using alpha-beta pruning?

- Best case - O(b^(m/2)) (we prune half of the branches at each level)
- Worst case - O(b^m) (no pruning)

Q13. If a game tree is too deep, even with alpha-beta pruning, the search remains impractical. Can you think of a method to reduce the computation time while still making a near-optimal decision?

- Depth limited searching

Q14. Does depth limit work? It means we cut off the search at a pre-defined depth.

- Yes

Q15. If we set the depth limit at 9, what minimax value should be returned before reaching to the leaf or being cut off by alpha-beta pruning?

- Return a heuristic value or evaluation score: the likely value of win

Q16. If we use an evaluation or heuristic function to estimate the minimax value at a state on the depth limited level, What characteristics make a good heuristic function? (Hint: Consider accuracy, efficiency, and consistency.)

- The input of the evaluation function needs to be the current state of the node
- Need to know how far away from a terminal node we are
- Accuracy: if game is likely to win, return a large value, and if a game is likely to lose, return a small value. We need to accurately show if a game is likely to win or lose.
- Efficiency: quickly calculate and evaluate the state. simple code as possible
- Consistency: for two similar board states, the evaluation function should give a similar value for both

Q17. Imagine you are designing an AI to play a simple version of Connect Four (connect 4). The game board is a 6x7 grid, and the goal is to get four pieces in a line. However, due to computational limits, your AI can only search to depth 4. Please propose an evaluation function to assess board at any point during the game.

- Guidelines for the evaluation function is it needs to summarize the state of the board in a singular value
- If the first state has a large score, that state is more likely to win
- Input is board, output is score

- check how many same colors are in a line with open adjacent spot, if there is more, return a large value, if not, return low
- if the AI bot is yellow, and the red tokens are winning, return a low value because it is likely a loss
    - for any action, return a score
- if 3 in row with open spot : (3*number of those)
- if 2 in row with open spot: (2*number of those)

- Check open case: if 3 in a row. Find those open cases
    - design evaluation function to give those cases a special score. If the opponent has 3 in a row, give a higher score to the action that places your token in the 4th row (to not lose)

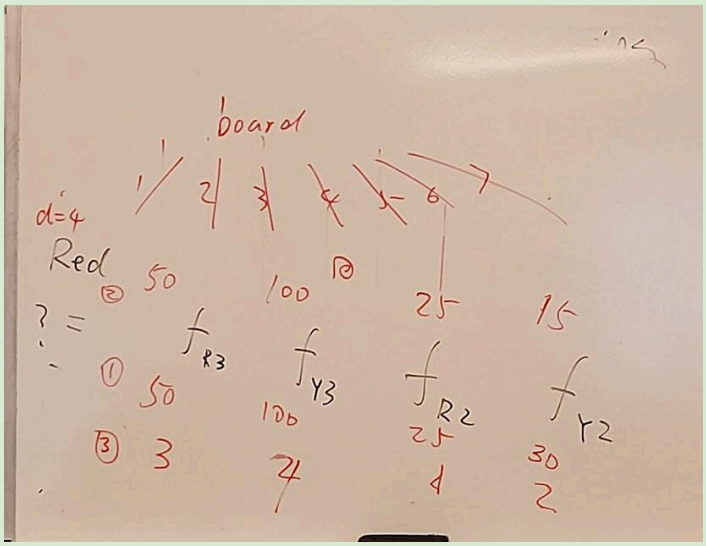
Q18. The most common design for an evaluation function is a linear combination of features.

$$Eval(s) = w1\ f1(s)+w2\ f2(s)+...+wn\ fn(s)$$

Each fi(s) corresponds to a feature extracted from the input state s, and each feature is assigned a corresponding weight wi. Features are simply some element of a game state that we can extract and assign a numerical value.

Based on this formula, please design an evaluation/heuristic function for connect-4 at depth d.

- weight - the number of adjacent opponent tokens - 2 in a row open case



Shows how the state of board is drawn, but it is wrong.

- penalty or award
- big penalty - large deficit
- avoiding large penalties will give a higher score for the evaluation function

Q19. Please read this paper and summarize the idea of the design of the heuristic function.

https://www.scirp.org/journal/paperinformation?paperid=125554

- avoiding large penalties will give a higher score for the evaluation function

Q20. If the utility is 1 and -1 at the termination leaf for win and lose in a zero sum game, and the evaluation score at a node is 100, do we need to normalize the evaluation score between -1 and 1?

- No, we do not have to follow the range.