ASSIGNMENT

GAME PLAY AND UNDERSTANDING

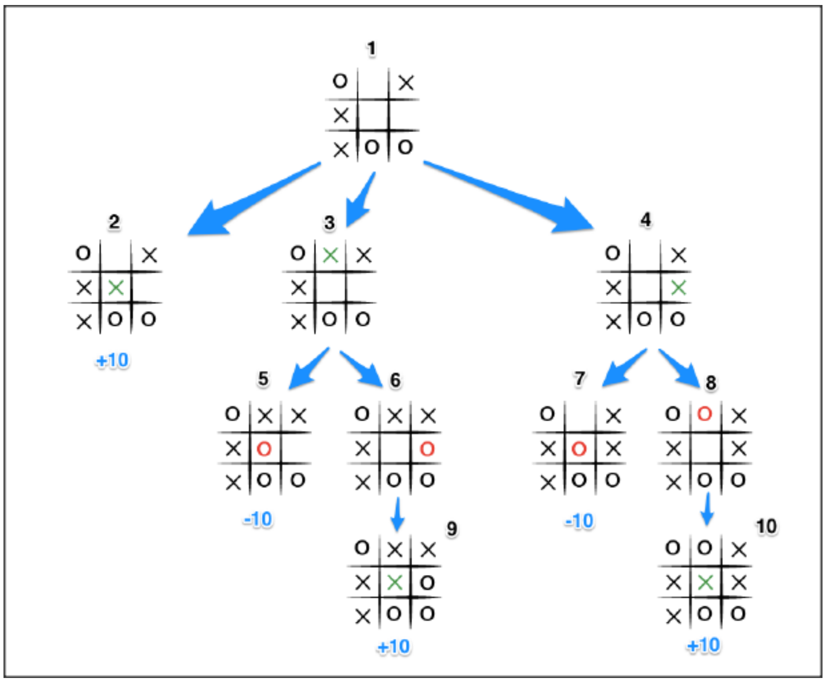
1. Explain Minimax search algorithm for the game Tic-Tac-Toe. Can we use alpha beta pruning to improve efficiency of the algorithm. Justify.

- The perfect TIC-TAC-TOE game is a game where the player either wins the game or draws the game.
- In case of playing against another perfect player the game always results in a draw.
- The "end game conditions " are as follows:-

  The player wins it gets +10 points.

  The player losses it gets -10 points as the opposition gains 10 points.

  If the game draws nobody gets any points.
- To construct a perfect game the minmax algorithm is used
- The key to the Minimax algorithm is a back and forth between the two players, where the player whose "turn it is" desires to pick the move with the maximum score.
- In turn, the scores for each of the available moves are determined by the opposing player deciding which of its available moves has the minimum score.
- And the scores for the opposing player's moves are again determined by the turn-taking player trying to maximize its score and so on all the way down the move tree to an end state.
- Taking an example of one state of the game as follows.



- It's X's turn in state 1. X generates the states 2, 3, and 4 and calls minimax on those states.
- State 2 pushes the score of +10 to state 1's score list, because the game is in an end state.
- State 3 and 4 are not in end states, so 3 generates states 5 and 6 and calls minimax on them, while state 4 generates states 7 and 8 and calls minimax on them.
- State 5 pushes a score of -10 onto state 3's score list, while the same happens for state 7 which pushes a score of -10 onto state 4's score list.
- State 6 and 8 generate the only available moves, which are end states, and so both of them add the score of +10 to the move lists of states 3 and 4.
- Because it is O's turn in both state 3 and 4, O will seek to find the minimum score, and given the choice between -10 and +10, both states 3 and 4 will yield -10.

- Finally the score list for states 2, 3, and 4 are populated with +10, -10 and -10 respectively, and state 1 seeking to maximize the score will chose the winning move with score +10, state 2.

ALPHA BETA PRUNING FOR EFFICIENCY

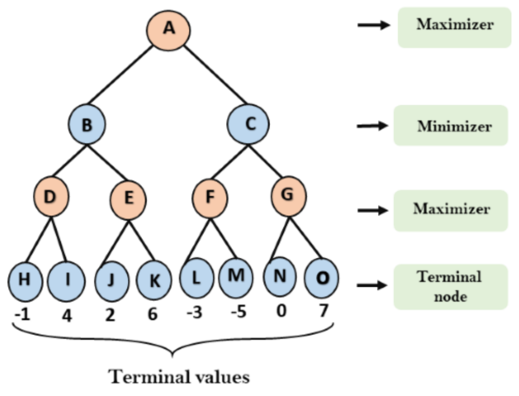
- Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree.
- It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess etc.). It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move.
- Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.
- Thus pruning away the worse cases the algorithm's efficiency is greatly improved.

2. Explain the following terms in game playing with examples.
I. Minimax(Position, Depth, Player)
II. Deep Enough(Position, Depth)

- Minimax is the function for the algorithm.
- The pseudo algorithm is as follows:-

```
function minimax (position, depth, Player) is
if depth ==0 or node is a terminal node then
       return static evaluation of node
if MaximizingPlayer then      // for Maximizer Player
maxEva= -infinity
 for each child of node do
 eva= minimax(child, depth-1, false)
maxEva= max(maxEva,eva)       //gives Maximum of the values
return maxEva
  else                        // for Minimizer player
 minEva= +infinity
 for each child of node do
 eva= minimax(child, depth-1, true)
 minEva= min(minEva, eva)       //gives minimum of the values
 return minEva
```

- Initial call minimax(node, 3, true)



- **Step 1:-**In the above tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.

- **Step 2**:- Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

For node D	max(-1,- -∞) => max(-1,4)= 4
For Node E	max(2, -∞) => max(2, 6)= 6
For Node F	max(-3, -∞) => max(-3,-5) = -3
For node G	max(0, -∞) = max(0, 7) = 7

- **Step 3:-** In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

  For node B= min(4,6) = 4

  For node C= min (-3, 7) = -3

- **Step4:-**Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

For node A max(4, -3)= 4

- This is how the Minimax function works.

- With any recursive program, we need to decide when to stop th3e program

- There are a variety of factors that may influence the decesicio0n such as

  Has one side won?
  How many ply have we already explored? Or how much time is left?
  How stable is the configuration?
- We use DEEP ENOUGH which is assumed to evaluate all of these factors and return true if the search should be stopped at the current level and FALSE otherwise
- It takes two parameters, position and depth, it will ignore its position parameter and simply return TRUE if its depth parameter exceeds a constant cut-off value.

3. **Is the minimax procedure a depth-first or breadth-first search procedure ?Why does the search in the game playing problems always proceed forward from the current position rather than backward from a goal state?**

- Minimax procedure is a depth-first process. One path is explored as far as time allows, the static evolution function is applied to the game positions at the last step of the path.
- The efficiency of the depth-first search can improve by branch and bound technique in which partial solutions that clearly worse than known solutions can abandon early.

REASON FOR FORWARD SERCH IN GAME PLAYING

- Firstly, the main reason for forward search rather than the backward search is done due the possibility of multiple goal states.
- Taking an example of a game of Chess there are numerous ways for a player to do check mate.
- Thus for a game of chess while check mate id the ultimate goal there is no fixed starting state if moved backwards as it has multiple goal states.
- Secondly, the factor of time can be considered.
- If the goal state is too far from the current state for the search to terminate, the backward search would not give any useful information whereas forward search will though terminated would give the programmer a good amount of useful information.

4. **Explain the difference between zero sum and non-zero sum games with and without perfect information with examples.**

ZERO SUM GAME

- A zero-sum game is one in which no wealth is created or destroyed. So, in a two-player zero-sum game, whatever one player wins, the other loses.
- Therefore, the player shares no common interests. There are two general types of zero-sum games: those with perfect information and those without.

In a game with perfect information, every player knows the results of all previous moves. Such games include chess, tic-tac-toe, and Nim.

GAME WITH PERFECT INFORMATION

- In games of perfect information, there is at least one "best" way to play for each player. This best strategy does not necessarily allow him to win but will minimize his losses.
- For instance, in tic-tac-toe, there is a strategy that will allow you to never lose, but there is no strategy that will allow you to always win.
- Even though there is an optimal strategy, it is not always possible for players to find it. For instance, chess is a zero-sum game with perfect information, but the number of possible strategies is so large that it is not possible for our computers to determine the best strategy.

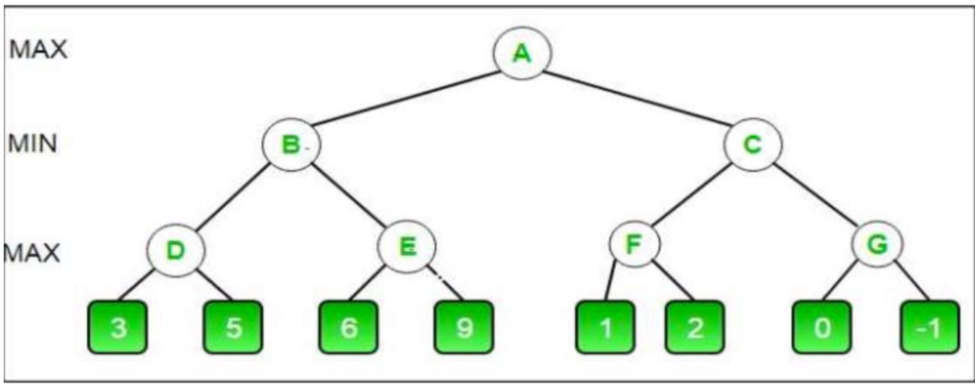
GAME WITH IMPERFECT INFORMATION

- In games with imperfect information, the players do not know all of the previous moves. Often, this occurs because the players play simultaneously. Here is an example of ROCK,SCISSORS and PAPER
- Rock, paper, scissors is an example of a zero-sum game without perfect information. Whenever one player wins, the other loses. We can express this game using a payoff matrix that explains what one player gains with each strategy the players use. In the payoff matrix below, the rows represent player 1's possible strategies while the columns represent player 2's possible strategies. 1 represents a win for player 1, 0 a tie and 1 a loss for player 1. So, for instance, the upper left entry is a 0 because if both players display rocks, they tie. The upper middle entry is a -1 because if player 1 displays a rock and player 2 displays paper, player 2 wins, and therefore, player 1 loses.

NON-ZERO SUM GAME

- Non-zero-sum games differ from zero-sum games in that there is no universally accepted solution.
- That is, there is no single optimal strategy that is preferable to all others, nor is there a predictable outcome.
- Non-zero-sum games are also non-strictly competitive, as opposed to the completely competitive zero-sum games, because such games generally have both competitive and cooperative elements.
- Players engaged in a non-zero sum conflict have some complementary interests and some interests that are completely opposed.

5. **Considering the diagram given below, what is the value of the root node considering the minimax algorithm. Which nodes will be pruned using alpha beta pruning.**

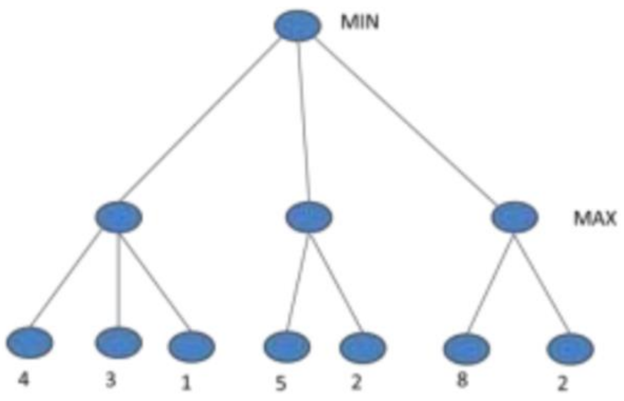


- **Step 1:-**Let the maximer take the first chance with the worst case initial value=-∞and minimizer will take the next turn with the worst case initial value=+∞. At **A** the maximizer must choose max of **B** and **C**, so **A** calls **B** first
- **Step 2:-**Comparing each value in terminal state with initial value of maximize and determine higher node values.It will be the maximum among all.
  For node D max(3,-∞)=3 α=3 and β=+∞
  To decide whether it's worth looking at its right node or not, it checks the condition β<=α which is false thus the search continues.
  Now α=max(3,5)=5
- **Step 3 :-**Now the E is called where β=5 and α=-∞ as passed from B
  For node E max(6,-∞)=6
  To decide whether it's worth looking at its right node or not, it checks the condition β<=α which is true thus the right child is truncated and value of 6 is passed to B.
- **Step 4:-**At B β=min(5,6)=5 therefore value of node B is 5
- **Step 4:-**Now at A α=max(-∞,5)=5 and then it calls the node C which calls node F.
- **Step 5:**-At F α=5 and β=+∞ it compares the terminal values.
  For α max(5,1)=5
  For node F max(1,2)=2 which is passed to C
- **Step 6:**-At C β=min(+∞,2)=2
  The condition β <= α becomes true as β = 2 and α = 5. So it breaks and it does not even have to compute the entire sub-tree of **G**.

- **C** now returns a value of 2 to **A**. Therefore the best value at **A** is max(5, 2) which is a 5.

  Hence the optimal value that the maximizer can get is 5
  Thus the root node has value 5

6. **Consider the following game tree in which the evaluation function values are shown below each leaf node for the max player. Assume that the root node corresponds to the minimizing player. Assume that the search always visits children left-to-right.**
   a) Compute the backed-up values computed by the minimax algorithm by writing values at the appropriate nodes in the tree given.
   b) Which nodes will not be examined by the alpha-beta pruning algorithm?
   c) In general, if the search always visits children right-to-left instead of left-to-right,
   i)The minimax value computed at the root will be changed.(write true/false)
   ii) The number of nodes pruned will be changed.(write true/false)



a) **The backed up values are as follows:-**

- **Step 1:-**Let the minimizer take the first chance with the worst case initial value=+∞and maximizer will take the next turn with the worst case initial value=-∞. At **A** the minimizer must choose max of **B** and **C or D**, so **A** calls **B** first.
- **Step 2:-**Comparing each value in terminal state with initial value of maximize and determine higher node values. It will be the maximum among all.
  For node B max(4,-∞)=4 α=4 and β=+∞
  As 4 is the maximum value the value of α remains 4 ass well the value of node B remains 4
  Now the value of the root node are α=-∞ and β=4 as it is MIN

- **Step 3 :-**Now the value of A are passed to both C and D is where β=4 and α=-∞ as passed from A
  For node C max(5,-∞)=5 by comparing terminal values.
  Now β=4 and α=5

  To decide whether it's worth looking at its right node or not, it checks the condition β<=α which is true thus the right child is truncated and value of the node is 5.

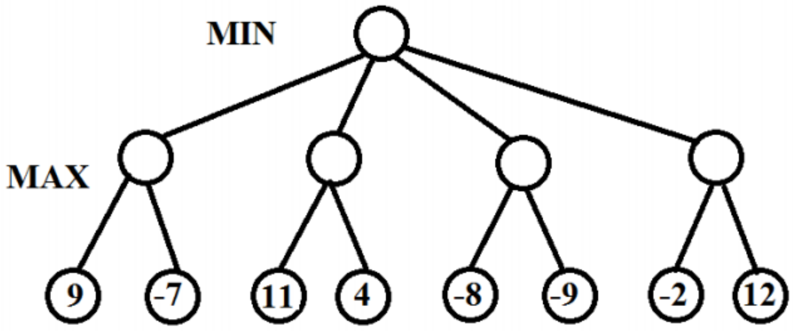
- **Step 4:-**At D β=4 and α=-∞
  For node C max(8,-∞)=8 by comparing terminal values.
  Now β=4 and α=8

To decide whether it's worth looking at its right node or not, it checks the condition β<=α which is true thus the right child is truncated and value of the node is 8.

- **Step 4:-**Now at A MIN has value the minimum of the nodes B, C, or D which is 4
  Hence the optimal value that the minimizer can get is 4
  Thus the root node has value 4

b) There are two right branches that are truncated by alpha beta pruning
c) For the searching method where the child are explored right-to-left
   I. False .The value of the root does not change.
   II. True .The value of branches truncated changes.

7. **We have two players: MIN who plays first and can make 4 moves, MAX who plays second and can make 2 moves. Suppose that after 1 turn, the values of the leaves are as in the figure :**



Compute (with the algorithm minimax) the value of the root of the tree, than say which is the most convenient move for MIN. Then tell with the reason, which parts of the tree are not generated if we perform an alpha beta pruning.

- **Step 1:-**Let the minimizer take the first chance with the worst case initial value=+∞and maximizer will take the next turn with the worst case initial value=-∞. At **A** the minimizer must choose max of **B** and **C D** or **E**, so **A** calls **B** first.
- **Step 2:-**Comparing each value in terminal state with initial value of maximize and determine higher node values. It will be the maximum among all.
  For node B max(9,-∞)=9 α=9 and β=+∞
  As 9 is the maximum value the value of α remains 9 as well the value of node B remains 9
  Now the value of the root node are α=-∞ and β=9 as it is MIN

- **Step 3 :-**Now the value of A are passed to C D and E is where β=9 and α=-∞ as passed from A
  For node C max(11,-∞)=11 by comparing terminal values.
  Now β=9 and α=11

To decide whether it's worth looking at its right node or not, it checks the condition β<=α which is true thus the right child is truncated and value of the node is 11.

- **Step 4:-**At D β=9 and α=-∞
  For node D max (-8,-∞)=-8 by comparing terminal values.
  Now β=9 and α=-8

  To decide whether it's worth looking at its right node or not, it checks the condition β<=α which is false thus the search continues.
  max(-8,-9)=-8 by comparing terminal values, thus value of α=-8

- **Step 5:-**At E β=9 and α=-∞
  For node E max(-2,-∞)=-2 by comparing terminal values.
  Now β=9 and α=-2

  To decide whether it's worth looking at its right node or not, it checks the condition β<=α which is false thus the search continues.
  max(-2,12)=12 by comparing terminal values, thus value of α=12 as well as of node.

- **Step 6:-**Now at A MIN has value the minimum of the nodes B, C, D or E which is -8
  Hence the optimal value that the minimizer can get is -8
  Thus the root node has value -8

8. What is Understanding? What makes understanding hard?

- Understanding to transform from one representation to another.
- The succe3ss or failure of an understanding program can rarely be measured in an absolute sense3.
- The success of understanding is required to be measured with respect to particular task being performed
- Language understanding and understanding in vision.
- There are many factors that make the understanding hard. They are as follows:-

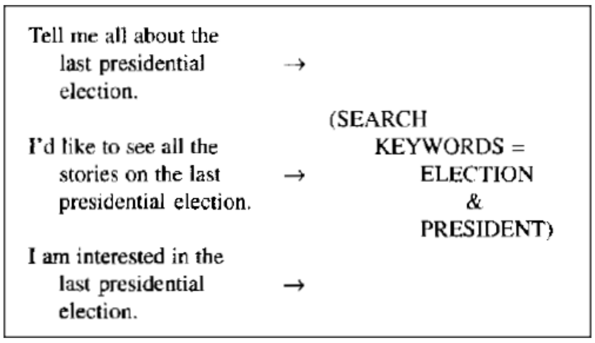
  1). The complexity of the target representation into which matching is being done.
  2).The type of mapping: one-one, many-one, one-many or many-many.
  3).The level of interaction of the components of the source representation.
  4). The presence of noise in input.

1.COMPLEXITY OF TARGET REPRESENTATION

- E.g. keyword based data retrieval system.
- -i want to read about last Presidential elections
- -Keywords:
- E.g. Conceptual dependency

2. Types of Mapping

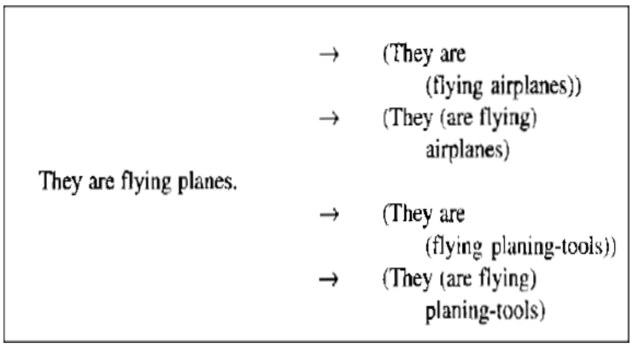
- – One-one e.g. A:=B+C*D
- – One-one mapping are the simplest to perform but rare
- – E.g. if images are interpreted, size and perspective will
- change as a function of the viewing position
- – E.g. "tall"
- – Many – One e.g.



Tell me all about the last presidential election. →

I'd like to see all the stories on the last presidential election. → (SEARCH KEYWORDS = ELECTION & PRESIDENT)

I am interested in the last presidential election. →



They are flying planes.

→ (They are (flying airplanes))

→ (They (are flying) airplanes)

→ (They are (flying planing-tools))

→ (They (are flying) planing-tools)

3. Level of intersection among Components

- - More or less interaction among components in a parse
- tree

4. Noise in Input

- - Interference before the basic input reaches the
- understander
- Background noise/images
- Typed language is less susceptible to noise.