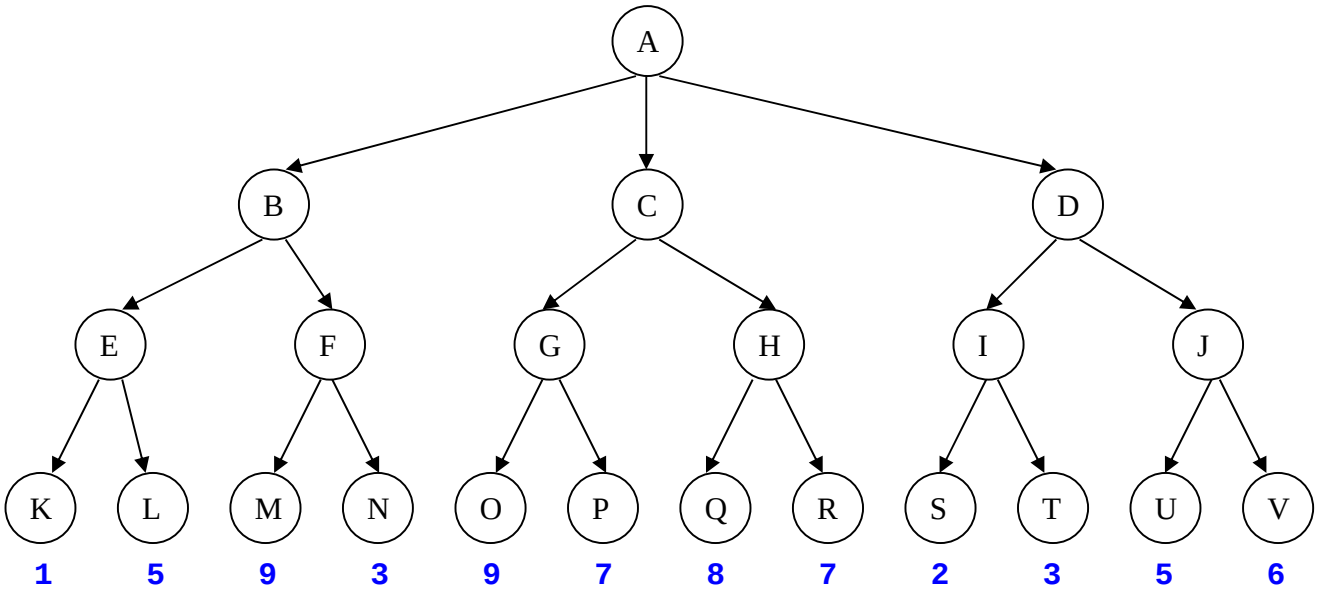
COS30019 - Introduction to Artificial Intelligence
Tutorial Problems Week 5

Task 1: For the following game tree:

(a) The first player (MAX) is trying to maximise the final score. Clearly indicate the max and min layers as part of your answer.
(b) Use minimax to determine the best move for MAX.
(c) Which nodes will not be examined if the alpha-beta procedure is used?
(d) In which order will the nodes be examined by the alpha-beta procedure?
(e) Did the alpha-beta procedure give the same best move (for MAX) as minimax?
(f) Draw a new game tree by re-ordering the children of each internal node, such that the new game tree is equivalent to the tree above, but alpha-beta pruning will prune as many nodes as possible. Which nodes will be pruned by the alpha-beta procedure in this case?

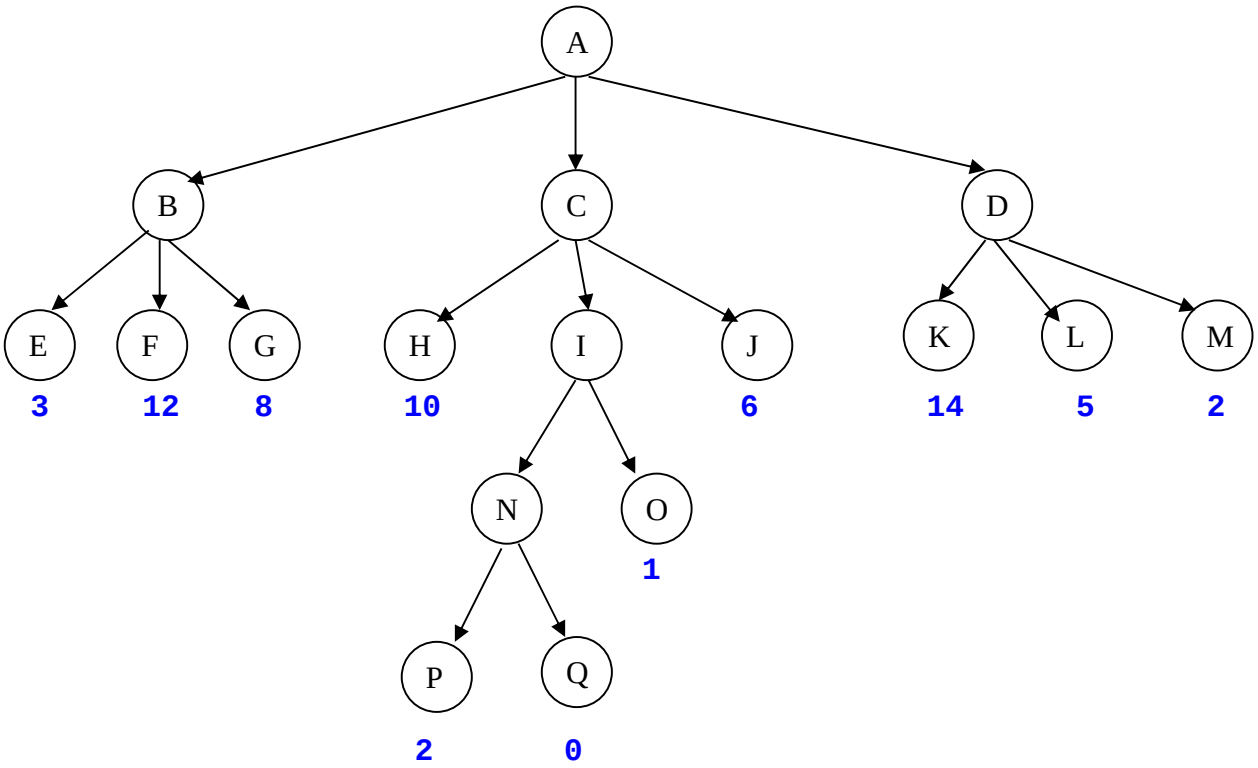


Task 2: In the following, a "max" tree consists only of max nodes, whereas an "expectimax" tree consists of a max node at the root with alternating layers of chance and max nodes. At chance nodes, all outcome probabilities are non-zero. The goal is to find the value of the root with a bounded-depth search.

a) Assuming that leaf values are finite but unbounded, is pruning (as in alpha-beta) ever possible in a max tree? Give an example, or explain why not.
b) Is pruning ever possible in an expectimax tree under the same conditions? Give an example, or explain why not.

Task 3: For the following game tree:

(a) The first player (MAX) is trying to maximise the final score. Clearly indicate the max and min layers as part of your answer.
(b) Use minimax to determine the best move for MAX.
(c) Use alpha-beta pruning to determine the best move for MAX.



Programming Task – Adversarial Search for Tic-Tac-Toe

You are tasked with creating a perfect AI Tic Tac Toe player. To meet the criterion of being perfect, the AI must never lose a game against the human player, regardless of who goes first. Draws are acceptable.
A portion of the code is already provided, including:

- Basic boiler plate code that starts the program.
- A GUI that displays the game board, and allows the player to make moves.
- A basic State class that will act as the data structure that represents the game world.
- A function to determine if the passed in state has a winning player.

To complete the lab, the following steps should be taken:

1. Complete the *State* data structure. Think about all of the relevant information that makes up each snapshot of the game world.

2. Think of an appropriate data structure to represent the frontier. Keep in mind that you will be using an Adversarial Search.
3. In the *TicTacToeLogic* class, create a function that will determine the possible moves given the current game world.
4. Find a way to ensure that the AI always picks the best move when it is its turn.
    - What are leaf nodes, and what would they consist of in Tic Tac Toe? How should we handle them?
    - How does a parent determine which of its children are the best option?
    - If the current state does not constitute a leaf, how should it be handled?

Extensions:

1. Provide your program with a rigged board, similar to the one depicted below:

	X	**X**
X		
X		

Assuming that the AI will always add nodes to the frontier from top left to bottom right, how do you believe the AI will handle the next move assuming the player is X and the AI is O? How could your program be altered to have the AI make a more appropriate move?