Artificial Intelligence

BS (CS) _SPRING_2025

Lab_09 Manual

Learning Objectives:

1. Adversarial Search

Adversarial Search

Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.

- In previous topics, we have studied the search strategies which are only associated with a single agent that aims to find the solution which often expressed in the form of a sequence of actions.
- But there might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.
- The environment with more than one agent is termed as **multi-agent environment**, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.
- So, **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games**.
- Games are modeled using Search algorithms and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

Game Playing in Artificial Intelligence

Game Playing is an important domain of artificial intelligence. Games don't require much knowledge; the only knowledge we need to provide is the rules, legal moves, and the conditions of winning or losing the game.

Both players try to win the game. So, both of them try to make the best move possible at each turn. Searching techniques like BFS(Breadth First Search) are not accurate for this as the branching factor is very high, so searching will take a lot of time. So, we need another search procedures that improve –

- **Generate procedure** so that only good moves are generated.
- **Test procedure** so that the best move can be explored first.

The most common search technique in game playing is **Minmax Search Algorithm**. It is a depth-limited depth-first search procedure. It is used for games like **chess and tic-tactoe**.

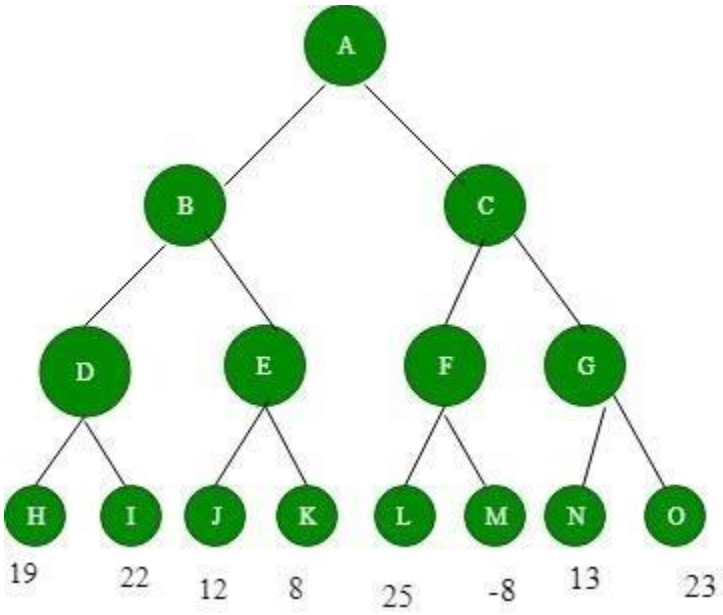
Minimax algorithm uses two functions

MOVE GEN : It generates all the possible moves that can be generated from the current position.

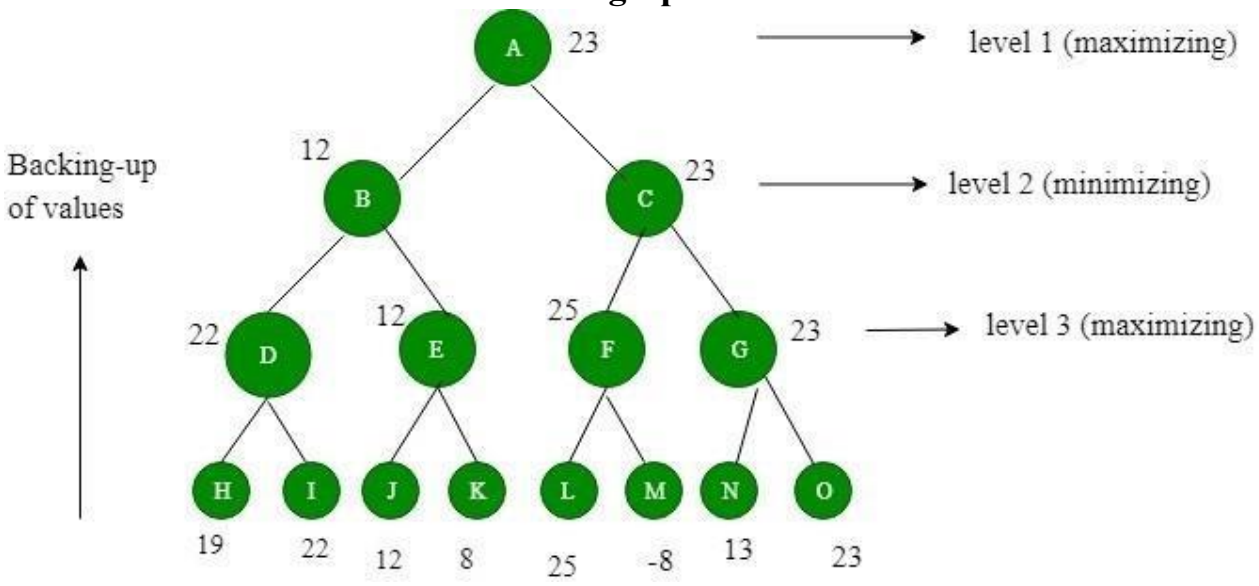
STATIC EVALUATION : It returns a value depending upon the goodness from the viewpoint of two-player.

This algorithm is a two-player game, so we call the first player as PLAYER1 and second player as PLAYER2. The value of each node is backed-up from its children. For PLAYER1 the backed-up value is the maximum value of its children and for PLAYER2 the backed-up value is the minimum value of its children. It provides most promising move to PLAYER1, assuming that the PLAYER2 has made the best move. It is a recursive algorithm, as same procedure occurs at each level.

Before Backing-up values



After Backing-up values



Pseudocode for Minimax Algorithm

```
function MINIMAX(state, depth, isMaximizingPlayer):

    if TERMINAL-TEST(state) or depth == 0:
        return EVALUATION(state)

    if isMaximizingPlayer:
        bestValue = -∞
        for each child in MOVE-GEN(state):
            value = MINIMAX(child, depth - 1, false)
            bestValue = max(bestValue, value)
        return bestValue

    else:  // Minimizing player
        bestValue = +∞
        for each child in MOVE-GEN(state):
            value = MINIMAX(child, depth - 1, true)
            bestValue = min(bestValue, value)
        return bestValue
```

Key Components:

- **MOVE-GEN(state):** Returns all legal moves from the current state.
- **EVALUATION(state):** Returns a score estimating how good the position is for the maximizing player.
- **TERMINAL-TEST(state):** Checks if the game is over.
- **depth:** Limits how far ahead the algorithm looks (prevents infinite recursion).

How Minimax Works (Step-by-Step):

- Start from the current game state.
- Generate all possible moves.
- Assume opponent plays optimally.
- Recursively compute best move using **max** for one player and **min** for the opponent.
- Return the best move for the current player based on the expected outcome.