Experiment No: 6

Aim : Implement Any one advanced search algorithm ( min-max)

Theory:

* **Mini-Max Algorithm:**

The Mini-Max algorithm is used in decision-making, primarily in two-player games, aiming to minimize the possible loss (min) while maximizing the potential gain (max). It involves two players: the Maximizer (trying to maximize their score) and the Minimizer (trying to minimize the Maximizer's score).

* **Steps Involved**:

1. **Generate the Game Tree:** Represent all possible moves from the current game state in a tree structure.
2. **Evaluate Terminal States:** Assign utility values to terminal nodes (win, lose, or draw).
3. **Propagate Utility Values:** Propagate the utility values up the tree:
4. **Maximizing Player:** Chooses the maximum value from child nodes.
5. **Minimizing Player:** Chooses the minimum value from child nodes.
6. **Select Optimal Move:** At the root node, the Maximizer selects the move that maximizes their utility.

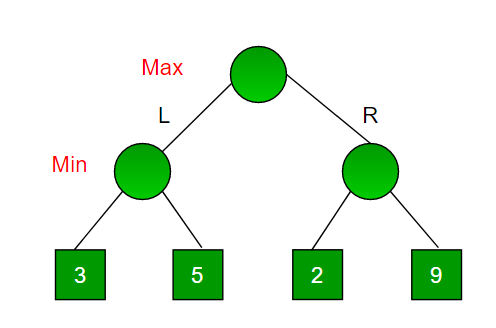
* **Mini-Max Formula**:
* **Maximizing Player's Turn**:  
  Max(s)=max⁡a∈A(s)Min(Result(s,a))\text{Max}(s) = \max\_{a \in A(s)} \text{Min}(\text{Result}(s, a))Max(s)=maxa∈A(s)​Min(Result(s,a))
* **Minimizing Player's Turn**:  
  Min(s)=min⁡a∈A(s)Max(Result(s,a))\text{Min}(s) = \min\_{a \in A(s)} \text{Max}(\text{Result}(s, a))Min(s)=mina∈A(s)​Max(Result(s,a))
* **Terminal States**:

Utility values are assigned based on the outcome (win, draw, or loss):

* + 111 for the Maximizer's win.
  + 000 for a draw.
  + −1-1−1 for the Minimizer's win.
* **Alpha-Beta Pruning**:  
  An optimization for the Mini-Max algorithm that eliminates unnecessary branches:
* **Alpha (α)**: Best value for the Maximizer.
* **Beta (β)**: Best value for the Minimizer.
* **Pruning**: If α≥β\alpha \geq \betaα≥β, further exploration of the branch is unnecessary.

This improves efficiency by reducing the number of nodes evaluated.

* Eg:

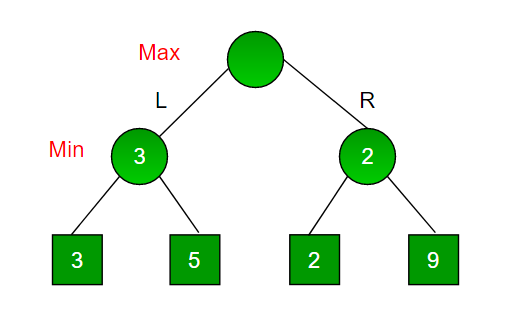


Since this is a backtracking based algorithm, it tries all possible moves, then backtracks and makes a decision.

* Maximizer goes LEFT: It is now the minimizers turn. The minimizer now has a choice between 3 and 5. Being the minimizer it will definitely choose the least among both, that is 3
* Maximizer goes RIGHT: It is now the minimizers turn. The minimizer now has a choice between 2 and 9. He will choose 2 as it is the least among the two values.

Being the maximizer you would choose the larger value that is 3. Hence the optimal move for the maximizer is to go LEFT and the optimal value is 3.

Now the game tree looks like below :



Program :

import random

def fitness(individual):

return sum(individual)

def hill\_climb(individual):

best = individual

best\_fitness = fitness(best)

for i in range(len(individual)):

neighbor = best[:]

neighbor[i] = random.randint(0, 1)

neighbor\_fitness = fitness(neighbor)

if neighbor\_fitness > best\_fitness:

best = neighbor

best\_fitness = neighbor\_fitness

return best

def genetic\_algorithm\_with\_hill\_climbing(pop\_size, generations, gene\_length, mutation\_rate):

population = [[random.randint(0, 1) for \_ in range(gene\_length)] for \_ in range(pop\_size)]

for generation in range(generations):

population = [hill\_climb(ind) for ind in population]

op :

