Pareenita A. Shirsath
T.E. AI & DS 52
AI

JAWAHAR EDUCATION SOCIETY'S
ANNASAHEB CHUDAMAN PATIL COLLEGE OF ENGG.
(Affiliated to the University of Mumbai)
ACPCE
Where knowledge is second nature
(Accredited by NAAC)

DOP: 10|07|24
DOS: Date : 24|07|24

Experiment No : 1

Aim: To Provide the PEAS description and TASK Environment for a given AI problem.

PEAS DESCRIPTION OF WAMPUS WORLD :-

(1) PERFORMANCE MEASURE :-

1] +1000 points for climbing out of the cave with the gold.
2] -1000 points for falling into a pit or being eaten by the wumpus.
3] -1 point for each action taken.
4] -10 points for using up the arrow.

(2) ENVIRONMENT :-

1] A 4x4 grid of rooms.
2] The agent starts in the top-left room, facing right.
3] The wumpus can be in any of the rooms, but not the starting room.
4] The gold is in one of the rooms, but not the starting room.
5] There is a pit in one of the rooms, but n the starting room.

(3) Sensors:- The agent can sense the follo
1] Breeze: Indicates that there is a pit it an a
room.
2] Stench: Indicaks that there is a wumpus in a
adjacent room.
3] Glitter: Indicates that the gold is in the current room

(4) Actuators:- The agent can perform the following actions:

1] Move forward
2] Turn left.
3] Turn right
4] Shoot the arrow.

☒ TASK ENVIRONMENT OF WAMPUS WORLD:-

1) Deterministic vs Stochastic:
- The Wumpus world is stochastic; meaning that the outcome of an action is not always certain. For example, if the agent moves forward, it may end up in a room with a pit, a room with the wumpus, or a room with neither.

2) Competitive vs Collaborative:
— The Wumpus world is competitive, meaning that the agent is trying to achieve its goal without regards for the goals of other agents. There are no other agents in the Wumpus world, so the agent is only competing with itself.

3) Single-agent vs Multi-agent:
- The Wumpus World is a single-agent task meaning that there is only one agent in the environment.

4) Static vs. Dynamic:
— The Wumpus world is static; meaning that the environment does not change over time. The wumpus, the gold, and the pits are always in the same places.

5) Discrete vs. Continuous:
— The wumpus world is discrete, meaning that the state space is finite. There are only a limited no. of rooms in the cave, & the agent can move only in limited numbe of directions.

6) Episodic vs. Sequential:
— The Wumpus world is episodic meaning that each episode begins with the agent in the starting room & ends whe the agent either reaches the goal or dies.

7) Known vs. Unknown:
— The Wumpus world is partially observable, meaning that the agent does not know the exact state of environment. The wumpu agent can only sense the presence of pits & wumpus through it sensors.

Conclusion:- Thus, I have successfully implemen to provide the PEAS description & TASK environm for a given AI problem.

4	stench		Breeze	PIT
3	 Wumpus	Breeze, stench, GOLD	PIT	Breeze
2	stench		Breeze	
1	 Agent	Breeze	PIT	Breeze

15

ACPCE
Where knowledge is second nature
(Accredited by NAAC)

JAWAHAR EDUCATION SOCIETY'S
ANNASAHEB CHUDAMAN PATIL COLLEGE OF ENGG.
(Affiliated to the University of Mumbai)

DOP: 24/07/24

Experiment No: 2

Aim: Identity suitable agent architecture for the Wompus World.

Theory: The Wumpus World game is basically Utility based agent. It aims to make choices that lead to outcomes with the highest possible value or benefit.

1) **Goal:-** A utility-based agent in "Wumpus World" aims to make choices that maxi-mize its source by assigning values to actions that lead to better outcomes.
2) **Utility functions:-** Values assigned to states (eg. gold v/s dangers).
3) **Decision Process:-** Expected utility calculated for action using probabilities & utilities.
4) **Uncertainity:-** Deals with uncertain hazard & Wampus locations.
5) **Trads-offs:-** Balances rewards & risks, in form of decisions.
6) **Complexity:-** Navigates maze, avoid hazards, finds gold —
7) **Adaption:-** Adjusts utility based on past experience.
8) Application Guides agent in Wumpus World for Optimal play.

9)

Stench		~Breeze~	PIT
Stench, Wumpus	~Breeze~ Stench, Wumpus	PIT	~Breeze~
Stench		~Breeze~	
Agent	~Breeze~	PIT	~Breeze~

9) Challenges: Models probabilities, define utility, values, handles changes.

Conclusion:- Thus, I have successfully identify suitable agent architecture for WAMPUS WORLD.

15

24.7.24

Pareenita A. Shirsath
T.E. AI & DS '52
AI

ACPCE
Where knowledge is second nature
(Accredited by NAAC)

JAWAHAR EDUCATION SOCIETY'S
ANNASAHEB CHUDAMAN PATIL COLLEGE OF ENGG.
(Affiliated to the University of Mumbai)

DOP: 10/24

Page No. : 1
DOS Date : / /

Experiment No : 3

Aim :- To write simple program using PROLOG as an AI programming language :-

Theory :- Prolog stands for programming in Logic. In the logic programming paradigm, prolog language is mostly widely available. Prolog is a declarative language, which means that a program consists of data based on the facts & rules (logical relationship) rather than computing how to find a solution. A logical relationship describes the relationships which hold for the given application.

To obtain the solution, the user asks a question rather than running a program. When a user asks a question, that to answer the run time system searches through the database of data facts & rules. Prolog features are 'Logical variable', which means that they behave like uniform data structure, a backtracking strategy to search for proofs, a pattern - matching facility, mathematical variable, & input & out are interchangeable.

To deduce the answer, there will be more than one way. In such case, the run time system will be asked to find another solution. To generate another solution, use the backtracking strategy.

Prolog is a weakly typed language with scope rules and dynamic type checking. Prolog is a declarative language that means can specify when problem we want to solve rather than how to solve it.

1. Atoms :-

- Anything enclosed in single quotes (e.g. 'whatever you like').
- Any sequence of characters, numbers and/or the underscore character which is preceded by a lower case character (e.g. this_IS_an_atom).
- Any continuous sequence of the symbols: +, -, *, /, \, ^, >, <, =, ', :, ., ?, @, #, $, &. (eg. ***+***+@).
- Any of the special atoms: [], {}, ;, !.

The Prolog built-in predicate atom/1 will tell you if a given term is or isn't an atom. It gives the following behaviour:

```
| ?- atom(foo).
yes.
| ?- atom(Foo).
no
| ?- atom('Foo').
yes
| ?- atom([]).
yes
```

2. Numbers :- • An integer (e.g. 99).
• A floating point number (e.g. 99.91).
The built-in predicate integer/1 succeeds if its argument is an integer. The built-in predicate real/1 succeeds if its argument is a real number.
These give the following number:

```
| ?- integer(9).
yes
| ?- integer(99.9).
no
| ?- real(99.9).
yes
| ?- real(9).
no
| ?- number(9).
yes
| ?- number(99.9)
yes
| ?- number(foo).
no.
```

3. Variables :- Any sequence of characters, numbers, or the underscore character which is preceded by an upper case character (e.g. This_IS_a_variable).

- Any sequence of characters, numbers and underscore character which is preceded underscore (e.g. _this_IS_a_variable).
- An underscore character by itself (e.g. _). The underscore character is referred to as the anonymous variable because it cannot be referenced by its name in other parts of the clause.
- The Prolog built-in predicate var/1 will tell you if a given term is or isn't a variable. It gives the following behaviour:

```
| ?- var(Foo).
Foo = _125459

yes
| ?- var(_)
yes
| ?- var(foo).
no
```

Facts, Rules & Queries — These are the building blocks of We will get some detailed knowledge about facts and rules, and also see some kind of queries that will be used in logic programming.

4. Facts:— A fact is a predicate expression that makes a declarative statement about the problem domain. Wherever a variable occurs in a Prolog expression it is assumed to be universally quantified. Note that all Prolog sentences must end with a period.

```
likes (john, susie).      /* John like Susie */
likes (X, susie).         /* Everyone like Susie */
likes (john, Y).          /* John likes everybody */
likes (john, Y), likes (Y, john)  /* John like everybody
                                     & everybody likes John */
```

likes (John, susie); likes (john, mary). /* John likes Susie or John like Mary */
not (likes (john, pizza)). /* John does not like pizza */
likes (john, susie) :- likes (john, mary). /* John likes Susies if John likes Mary.

5. Rules :- A rule is a predicate expression that uses logical implication (:-) to describe a relationship among facts. Thus, a Prolog rule takes the form:
left_hand_side :- right_hand_side.
This sentence is interpreted as : left_hand_side if right_hand_side. The left_hand_side is restricted to a single, positive, literal, which means it must consists of a positive atomic expression It cannot be negated and it cannot contain logical connectives.
This notation is known as as Horn clause. In Horn clause logic, the left hand side of the clause is the conclusion, and must be a single positive literal. The RHS contains the premises. The Horn clause calculus is equivalent to the first-order predicate ~~calculus~~. calculus.
Examples of valid rules :

Examples of valid rules:

```
friends (X, Y) :- likes (X, Y), likes (Y, X). /* X and Y
                                are friends if they like
                                other */
Hates (X, Y) :- not (likes (X, Y)). /* X hates Y if
                X does not like Y. */
enemies (X, Y) :- not (likes (X, Y)), not (likes (Y, X))
        /* X and Y are enemies if they don't
           like each other */.
```

Examples of invalid rules:

```
Left_of (X, Y) :- right_of (Y_X)  /* Missing a
                                     period */
likes (X, Y), likes (Y, X) :- friends (X, Y)  /*
                              LHS is not a single literal */
not (likes (X, Y)) :- Hates (X, Y). /* LHS cannot be
                                       neglected */
```

6. Queries:- The Prolog interpreter responds to queries about the facts and rules represented in its database. The database is assumed to represent what is true about a particular problem domain. In making a query you are asking Prolog whether it can prove that your query is true. If so, it answers "yes" & displays any variable bindings that it made in coming up with the answer. If it fails to prove the query true, it answers "No".

Logic of program :-

This Prolog program defines a family tree.
The first few lines of code (female (pam). male (tom). male (bob). female (liz). female (ann). female (pat). male (jim).) define the genders of the people in the family tree.
The next line (parent (pam, bob). defines that Pam is Bob's mother. The following lines of code define the order parent-child relationships in the family tree.
The next line (parent (pam, bob).) defines the Pam is Bob's mother. The following lines of code define the other parent-child relationships in the family tree.
The next three lines of code (mother (X, Y) :- parent (X, Y), female (X), father (X, Y) :- parent (X, Y), male (X).
grandparent (X, Y) :- parent (X, Z), parent (Z, Y).) define the predicates mother, father & grandparent.
The mother (X, Y) predicate states that X is the mother of Y if X is a parent of Y and X is female.
The father (X, Y) predicate states that X is the father of Y if X is a parent of Y and X is male.

The grandparent (X, Y) predicate states that grandparent of Y if X is a parent of is a parent of Y.
For example, the following query would retu true: mother (pam, bob).
This query asks if Pam is the mother of Bob.
The predicate mother (X, Y) is true if X is the mothe of Y, and the first line of code (female (pam)) states that Pam is female.
Therefore, the query returns true.

Here are some other queries that you could ask:

1] Is Tom the father of Liz (parent (tom, liz)).
2] Is Ann the grandparent of Jim?
(grandparent (ann. jim).).
3] who are the parents of Pat ? (parent (X, pat).).

Procedure :-

Step -1 = In Linux / Ubuntu Go to Text Editor & Paste This.

```
female (pam).
male (tom).
male (bob).
female (liz).
female (ann).
female (pat).
male (jim).
parent (pam, bob).
parent (tom, bob).
parent (tom, liz).
parent (bob, ann).
```

```
parent (bob, pat).
parent (pat, jim).
mother (X, Y) :- parent (X, Y), female (X).
father (X, Y) :- parent (X, Y), male (X).
grandparent (X, Y) :- parent (Y, Z), parent (Z, Y)
```

Step - 2 = SAVE This is .pl Extension in Desktop.

Step - 3 = Go to Command line Type cd Desktop.

Step - 4 = Then, swipl - s filename.pl

Step - 5 = Enter this Queries.

1] grandparent (X, Y).
2] mother (X, Y).
3] males (X) after Type Yes.

Conclusion: Thus, we have successfully implemented to write simple program using PROLOG as an AI programming Language.

ACPCE
Where knowledge is second nature
(Accredited by NAAC)

JAWAHAR EDUCATION SOCIETY'S
ANNASAHEB CHUDAMAN PATIL COLLEGE OF ENGG.
(Affiliated to the University of Mumbai)

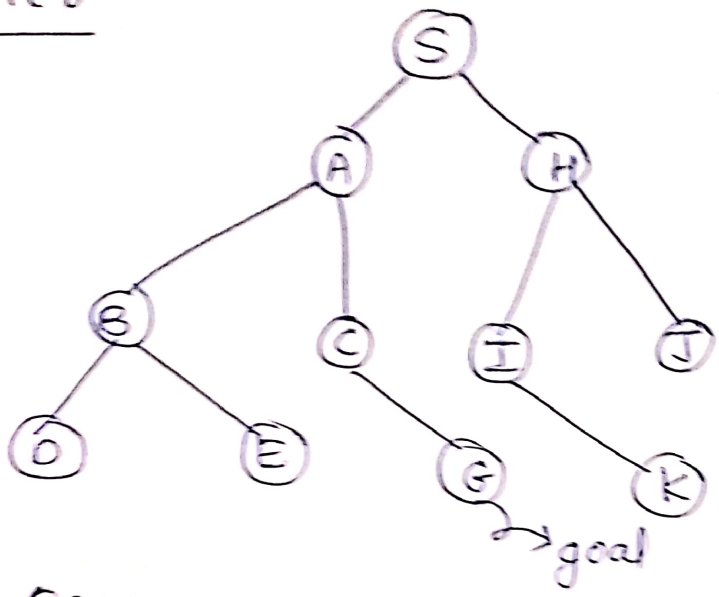
Experiment No : 4

Aim :- To implement any one of the uniformed search technique.

Theory :- Depth first search is a traversing algorithm used in tree and graph like dat structures. It generally starts by exploring the deepest node in the frontiers. Starting at the root node, the algorithm proceeds to search to the deeper level of the search to tree until nodes with no successor are reached. Suppose the node with unexpanded successors in encountered then the search back - tracks to the next deepest node to explore alternative paths. DFS explores a graph by selecting a path & traversing it as deeply as possible before backtracking

When the DFS encounters a new node, it adds it to the stack to explore its neighbours. If it reaches a node with no successors (leaf node), it works by backtracking such as popping nodes off the stack to explore the alternative paths. DFS is not cost - optimal since it does not guar-antee to find shortest path.

Example:-



Open	Close
S	—
A , H	S ,
B , C , H	S , A ,
D , E , C , H	S , A , B ,
E , C , H	S , A , B , D ,
C , H	S , A , B , D , E ,
G , H	S , A , B , D , E , C
H	S , A , B , D , E , C , G

Path = S → A → B → D → E → C → G

Conclusion:- Thus, I have successfully implemented one (DFS) of the uninformed search techniques.

ACPCE
Where knowledge is second nature
(Accredited by NAAC)

JAWAHAR EDUCATION SOCIETY'S
ANNASAHEB CHUDAMAN PATIL COLLEGE OF ENGG.
(Affiliated to the University of Mumbai)

Experiment No: 5

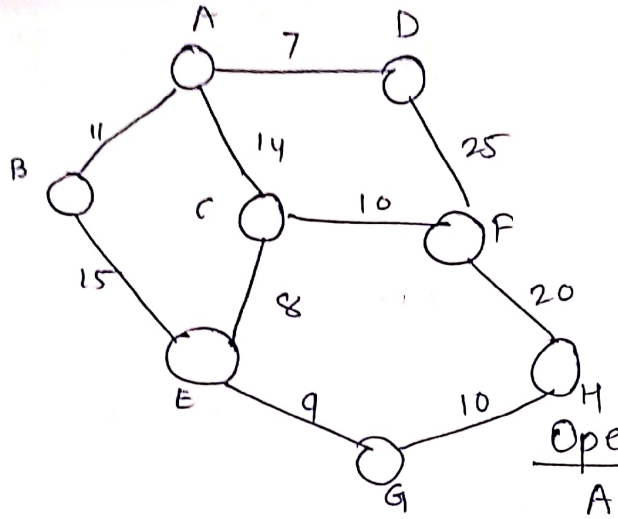
Aim:- To implement any one of the informed search technique.

Theory:- A* is a powerful graph traversal and path finding algorithm widely used in AI and computer science. It is mainly used to find the shortest path between 2 nodes in a graph, given the estimated cost of getting from the current node to the destination node. The main advantage of the algorithm is its ability to provide an optimal path by exploring the graph in a more informed way compared to traditional search algorithms such as Dijkstra's algorithm. A* ensures that the path found is a short as possible.
The main idea of A* is to evaluate each node based on 2 parameters:-

1. g(n) - The actual cost to get from the initial node to n.
2. h(n) - Heuristic cost from node n to destination node n.

A* selects the nodes to be explored based on the lowest value of f(n).

Example:-



A = 40
B = 32
C = 25
D = 35
E = 19
F = 17
H = 10
G = 0

Open	Closed
A	-
C, B, D	{A}
B, D	{A, C}
F, E, B, D	{A, C}
E, B, D	{A, C, F}
G, E, B, D	{A, C, F}
E, B, D	{A, C, F, G}

Path = A → C → F → G

Conclusion:- Thus, I have successfully implemented one of the Informed search technique.

ACPCE
Where knowledge is second nature
(Accredited by NAAC)

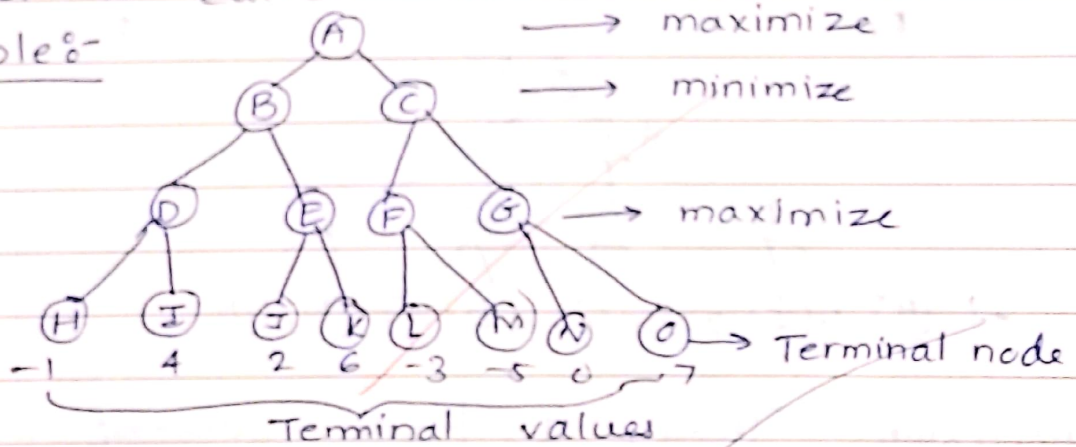
JAWAHAR EDUCATION SOCIETY'S
ANNASAHEB CHUDAMAN PATIL COLLEGE OF ENGG.
(Affiliated to the University of Mumbai)

Experiment No: 6

Aim:- To implement adversarial search using min - max algorithm.

Theory:- Min- max algorithm is a recursive or backtracking algorithm which is used in decision making and game - theory. It provides an optimal move for the player assuming that opponent is also playing optimally. It uses recursion to search through game-tree min - max algo is mostly used for game playing in AI. Such as chess, checkers tic - tac - toe, go, and various 2 players game. The algorithm computes the minimax decision for current state. In this, algo, 2 players play the game, one is called max and other is called min

Example:-

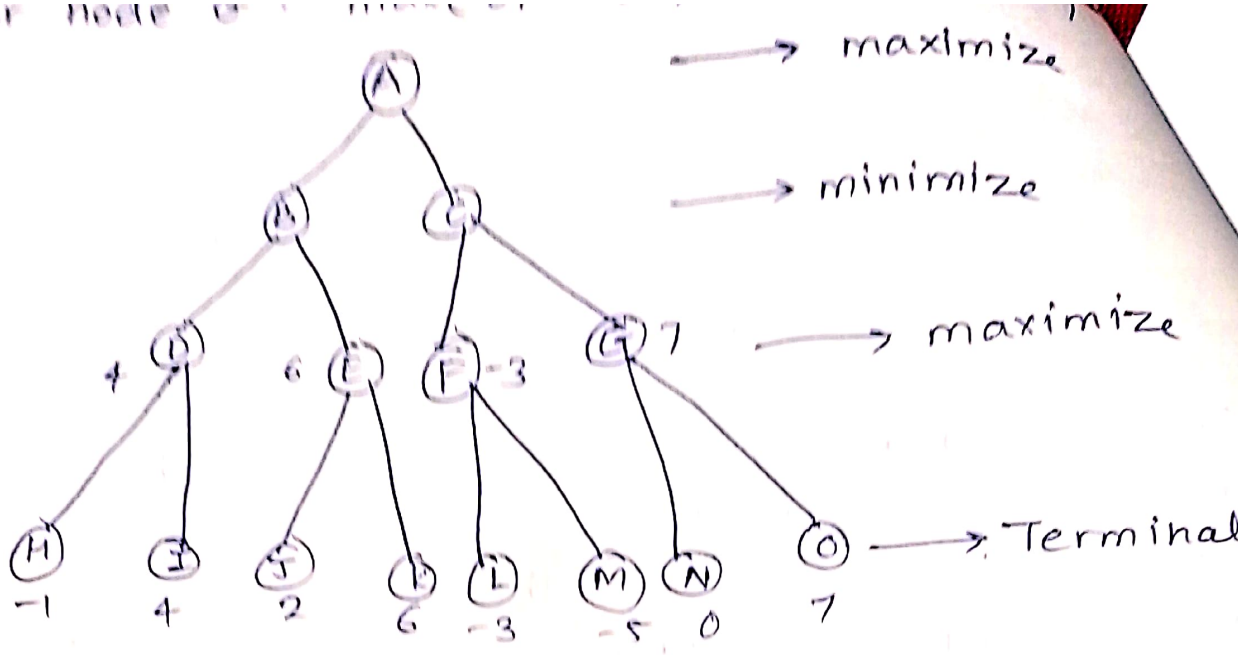


For node D : max (-1, -∞) ⇒ max (-1, 4) = 4

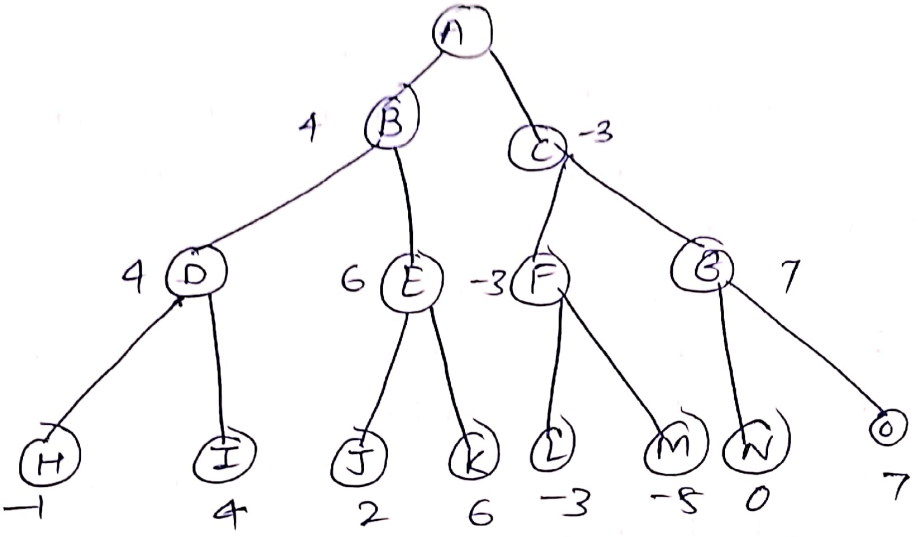
For node E : max (2, -∞) ⇒ max (2 , 6) = 6

For node F : max (-3 , -∞) ⇒ max (-3 , -5) = -3

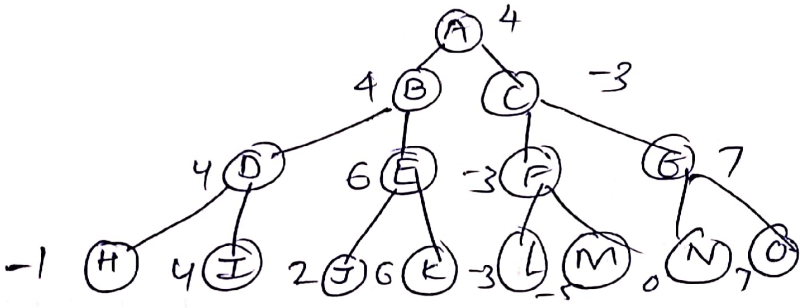
For node



For node B ⇒ min (4, 6) = 4
C ⇒ min (-3, 7) = -3



For node A ⇒ max (4, -3) = 4



Conclusion:- Thus, I have successfully implemented adversarial search using min-max algorithm.