Datta Meghe College of Engineering **Airoli, Navi Mumbai**	**DEPARTMENT OF ELECTRONICS ENGINEERING** **ACADEMIC YEAR : 2021 – 22 (TERM – II)** **List of Experiments** **Course Name : Artificial Intelligence** **Course Code : CSC604/CSL604**

Sr. No	**Name of experiment**	**COs Covered**	**Page No.**	**Date of Performance**	**Date of Submission**	**Marks & Signature**
1	One case study on AI applications published in IEEE/ACM/Springer or any prominent journal.	CSC 604.6				
2	Assignments on State space formulation and PEAS representation for various AI applications	CSC 604.3				
3	Program on uninformed search methods.	CSC 604.3				
4	Program on informed search methods.	CSC 604.3				
5	Program on Game playing algorithms.	CSC 604.3				
6	Program for first order Logic(Mini Project)	CSC 604.4 CSC 604.6				
7	Planning Programming	CSC 604.5				
8	Implementation for Bayes Belief Network	CSC 604.5				
9	Assignment 1	CSC 604.1, CSC 604.2 ,CSC 604.3				
10	Assignment 2	CSC 604.4, CSC 604.5, CSC 604.6				

This is to certify that Miss. **Farhat Naik** of **TE Computer-A** Roll No. **50** has performed the Experiments / Assignments / Tutorials / Case Study Work mentioned above in the premises of the institution.

____

Practical Incharge

DATTA MEGHE COLLEGE OF ENGINEERING,

AIROLI, NAVI MUMBAI

DEPARTMENT OF COMPUTER ENGINEERING

Institute Vision	:	To create value - based technocrats to fit in the world of work and research
Institute Mission	:	To adapt the best practices for creating competent human beings to work in the world of technology and research.
Department Vision	:	To provide an intellectually stimulating environment for education, technological excellence in computer engineering field and professional training along with human values.

Department Mission :

M1: To promote an educational environment that combines academics with intellectual curiosity.

M2: To develop human resource with sound knowledge of theory and practical in the discipline of Computer Engineering and the ability to apply the knowledge to the benefit of society at large.

M3: To assimilate creative research and new technologies in order to facilitate students to be a lifelong learner who will contribute positively to the economic well-being of the nation.

Program Educational Objectives (PEO):

PEO1: To explicate optimal solutions through application of innovative computer science techniques that aid towards betterment of society.

PEO2: To adapt recent emerging technologies for enhancing their career opportunity prospects.

PEO3: To effectively communicate and collaborate as a member or leader in a team to manage multidisciplinary projects

PEO4: To prepare graduates to involve in research, higher studies or to become entrepreneurs in long run.

Program Specific Outcomes (PSO):

PSO1: To apply basic and advanced computational and logical skills to provide solutions to computer engineering problems

PSO2: Ability to apply standard practices and strategies in design and development of software and hardware based systems and adapt to evolutionary changes in computing to meet the challenges of the future.

PSO3: To develop an approach for lifelong learning and utilize multi-disciplinary knowledge required for satisfying industry or global requirements.

Program Outcomes as defined by NBA (PO)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

DATTA MEGHE COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

Course Name: Artificial Intelligence Lab (R-19)

Course Code: CSC604

Year of Study: T.E., Semester: VI

Course Outcomes

CSC604.1	Ability to develop a basic understanding of AI building blocks
CSC 604.2	Ability to identify the suitable intelligent agent
CSC 604.3	Ability to choose an appropriate problem solving method and to analyze the strength and weaknesses of AI approaches to knowledge– intensive problem solving.
CSC 604.4	Ability to choose appropriate knowledge representation technique and to design models for reasoning with uncertainty as well as the use of unreliable information.
CSC 604.5	To understand the role of planning and learning in intelligent systems
CSC 604.6	Ability to design and develop AI applications in real world scenarios

DATTA MEGHE COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

ACADEMIC YEAR 2021-22 (TERM II)

SUBJECT: ARTIFICIAL INTELLIGENCE

SEM: VI

RUBRICS FOR GRADING EXPERIMENTS

Rubric Number	Rubric Title	Criteria	Marks* (out of 10)
R1	Timeliness	On-time	2
		Delayed by not more than a Week	1
R2	Knowledge	Able to answer all questions	4
		Able to answer most of the questions	3
		Able to answer some of the questions	2
		Able to answer very few of the questions	1
R3	Implementation	Correct Logic, Well formatted and Structured program	4
		Correct Logic, formatted and Structured program	3
		Correct Logic and Structured program	2
		Correct Logic	1

DATTA MEGHE COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

ACADEMIC YEAR 2021-22 (TERM II)

SUBJECT: ARTIFICIAL INTELLIGENCE

SEM: VI

RUBRICS FOR GRADING MINI PROJECT

Rubric Number	Rubric Title	Criteria	Marks* (out of 10)
R1	Design of Knowledge base and Conversion to FOL	Proper Design and Conversion with explanation	2
		Proper Design and Conversion	1
R2	Implementation in Prolog	Implementation of Complete Solution	2
		Implementation of Partial Solution	1
R3	Clarity of Concept	Clear understanding	1
		Concepts are not clear	0

DATTA MEGHE COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

ACADEMIC YEAR 2021-22 (TERM II)

SUBJECT: ARTIFICIAL INTELLIGENCE

SEM: VI

RUBRICS FOR GRADING CASE STUDY

Rubric Number	Rubric Title	Criteria	Marks* (out of 10)
R1	Understanding of Problem Definition & Objectives	Clear Understanding of Problem & Objectives	4
		Fair Understanding of Problem & Objectives	3
		Clear Understanding of Problem	2
		Fair Understanding of Problem	1
R2	Literature Review	In Depth Literature Survey with Findings	4
		In Depth Literature Survey	3
		Fair Literature Survey	2
		Not enough Literature Survey	1
R3	Presentation & Communication Skills	Presented and communicated very well	2
		Presented and communicated poorly	1

DATTA MEGHE COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

ACADEMIC YEAR 2021-22 (TERM II)

SUBJECT: ARTIFICIAL INTELLIGENCE

SEM: VI

RUBRICS FOR GRADING ASSIGNMENT

Rubric Number	Rubric Title	Criteria	Marks* (out of 10)
R1	Timeline	On-time	1
		Delayed	0
R2	Knowledge	Able to answer all questions	2
		Able to answer some of the questions	1
R3	Neatness	Well Written	2
		Fairly Written	1

EXPERIMENT NO. 1

Date of Performance :

Date of Submission :

AIM: Case study of any Artificial Intelligence Application.

THEORY:

1. To carry out case study for any application Artificial Intelligence, download the paper from IEEE/ACM/Springer/Elsevier or any other prominent journal not older than 2019.

2. Download the references given in that paper for literature survey.

3. Carry out the summarized findings and identify the gaps present in the study.

4. Provide the solutions for gaps and suggest what can be done to overcome the limitations of the present study.

5. Prepare a report containing:
Abstract
Introduction
Literature Review
Proposed system
Conclusion
References

Abstract:

Artificial Intelligence is the greatest answer for managing massive data flows and storage in IOT network. With the introduction of high-speed internet networks and numerous modern sensors that may be integrated into a microcontroller, the Internet of Things is becoming increasingly popular. Sensor data and user data will now be sent and received from workstations over the data flows internets. With the rise in the number of workstations and sensors, some data may experience issues with storage, delay, channel limitations, and network congestion. Many algorithms have been proposed in the last ten years to avoid all of these issues. Among all the algorithms, Artificial Intelligence is still the finest choice for data mining, network management, and control. The purpose of this study is to show how artificial intelligence systems can be used in the Internet of Things. The paper will emphasise the relevance of data mining and management. In addition, the methods utilized in Artificial Intelligence, such as fuzzy logics and neural networks, will be studied in connection with the IoT network in this study. The self-optimizing network and software defined network are two crucial characteristics in the AI IoT System. IoT; Artificial Intelligence; Neutral network; self-optimizing network; software defined network; IoT;

Introduction:

To manage home appliances by using AI and IOT to manage data flow and storage efficiently. As the workstation and sensors are increasing day by day we might face problems in the data storage, delay in transmission, channel limitation and high amount of network traffic. The aim of this case is to overcome all these problems and for this Artificial Intelligence is the best solution which will help to manage and control the network traffic and data mining as well.

The Internet of Things (IoT) is a new technology that allows you to send and receive sensor data via the internet. It's similar to regular data communication, with the exception that sensors and microcontrollers are frequently employed in IoT. The microcontroller and portable communication devices such as a mobile phone, communication pad, or even a smartwatch is used to send and receive data instead of a computer. Time and the free channel are normally computed and assigned to the user who wants to send and receive data when optimizing a network. A router can automatically optimise a network, and the system can update the router's table. The system will compute and determine the data's shortest path to flow. In order for the Artificial Intelligence system applied into the IoT networks, certain terms and principles must understand. For Artificial intelligence, there are two commonly used techniques - neural network and fuzzy logic [1].

Literature Review:

Artificial Intelligence research in the context of the Internet of Things isn't new. There have been numerous proposals for Artificial Intelligence applications in the IoT in the past. Making all devices communicate with one another is one of the concepts offered. This means that a user can operate house appliances from the transportation. The user can not only make calls but also operate household appliances from their smartphone.

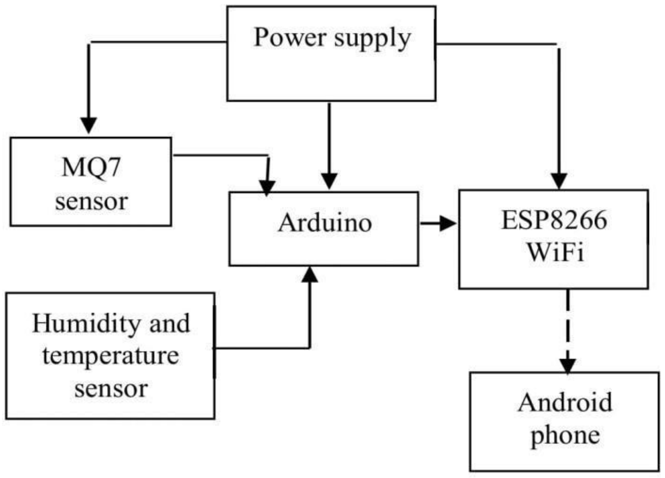
IOT devices commonly used to control home appliances through android and Iphone with the suitable app which will ON or OFF the appliances via internet connection. Aside from that, several of the devices have sensors integrated. The sensors then read the signals from nature and convert them to electrical changing voltages, which may subsequently be processed and transmitted to the receiver (Android phone). Apps are then used to display the signals. The signal information can also be viewed on the internet website.

It's hardly unexpected that many consumers currently utilize IoT voice recognition to operate their equipment. When utilizing speech to control appliances, you'll need to set up a recorder with a training system so that the voice can be recognized.[2] Alexa is a well-known IoT gadget that uses Wi-Fi to control household equipment.

Data mining is another application of Artificial Intelligence in IoT. Data mining is a data management and storage space reduction strategy. This means that as the amount of data in the network grows, there will be a greater tendency to spend more time digging out the necessary data. The data mining technique is used to shorten the time it takes to find the desired data.

Proposed Methodology:

Below is a very basic hardware implementation of IoT utilizing Artificial Intelligence. Two sensors are used in the suggested concept: a MQ7 gas sensor and a humidity sensor. The outputs of both sensors are connected to the Arduino microcontroller. After that, the Arduino is connected to the ESP8266 WiFi chip. An air interface is used to link the Arduino to the Android phone. In Figure the dotted line of the arrow represents an air interface.



The steps involved in data mining are [3]:

1. Data integration: It is the process of bringing together facts from various sources into a single, coherent view.

2. Data selection: It is the process of deciding on the proper data kind and source, as well as appropriate data collection tools.

3. Data cleaning: It is the process of eliminating or changing data that is erroneous, incomplete, irrelevant, duplicated, or incorrectly formatted in order to prepare it for analysis.

4. Data transformation: It is the process of converting data from one format to another, usually from a source system's format to a destination system's needed format.

5. Data mining: Data analysis is the process of going through big data sets to find patterns and relationships that can aid in the resolution of business challenges.

6. Pattern evaluation: Based on provided measurements, recognizing strictly growing patterns expressing knowledge.

Conclusion:

In this case Artificial Intelligence helps us to determine temp and humidity and will send the reports accordingly. All of the sensor data is automatically saved on the server and can be downloaded at any time. This is due to the fact that the user only has one account on thingspeak.com. As can be seen, a basic Arduino and ESP8266 can be used to construct a true IoT. The transmission and receiving of sensor data via the internet is possible and effective with the firmware downloaded into the ESP8266 and the Arduino board properly programmed.

References:

[1] Lily, D. Chan, B. and Wang, T. G. (2013). “A Simple Explanation of Neural Network in Artificial Intelligence,” IEEE. Trans on Control System, vol. 247, pp. 1529–5651.

[2] Nicole, R. and Lee, J. H. (2015). “The IoT Concepts and Design,” International Journal of Engineering, Vol. 6, No. 7, pp. 16 - 29.

[3] Gorozu, A. Hirano, K. Okawa, K. and Tagawaki, Z. (2014). “High Speed Data Mining Technique" IEEE Trans on Electronics, Vol. 10, Vol. 17, pp. 10 - 30.

SIGN AND REMARK

DATE

R1 **(4 Marks)**	**R2** **(4 Marks)**	**R3** **(2 Marks)**	**Total** **(10 Marks)**	**Signature**

EXPERIMENT NO. 2

Date of Performance :

Date of Submission :

AIM: Assignments on State space formulation and PEAS representation for various AI applications.

THEORY:

1.1 Specifying Task Environment

Task environments, which are essentially the "problems" to which rational agents are the "solutions.” In designing an agent, the first step must always be to specify the task environment as fully as possible with the help of PEAS (Performance, Environment, Actuators, Sensors) description.

1.2 Example of PEAS

Movie Recommendation System

Problem Definition:

An online movie recommendation system helps user to identify a list of movie recommendations, which contains at least one movie that the user will start watching as their next selection depending upon the previous watch history of user.

PEAS Descriptors:

PEAS stands for Performance measure, Environment, Actuator, Sensor.

The PEAS description for the above-mentioned problem description is as follows:

Performance Measure: Similar Actor/Actress, movies by similar Studios, the movies of genreuser have watch most.

Environment: Streaming Platform, Movie Store

Actuator: Recommends the movie which user would like to watch Next.

Sensor: Keyboard, Mouse.

A goal directed agent needs to achieve certain goals. Such an agent selects its actionsbased on the goal it has. Many problems can be represented as a set of states and a set of rules of how one state is transformed to another. Each state is an abstract representation of the agent's environment. It is an abstraction that denotes a configuration of the agent.

1.1 Problem Formulation

• Formulate a problem as a state space search by showing the legal problem states, the legal operators, and the initial and goal states.

• A state is defined by the specification of the values of all attributes of interest in the world

• An operator changes one state into the other; it has a precondition which is the value of certain attributes prior to the application of the operator, and a set of effects, which are the attributes altered by the operator

• The initial state is where you start

• The goal state is the partial description of the solution

An initial state is the description of the starting configuration of the agent an action or an operator takes the agent from one state to another state which is called a successor state. A state can have a number of successor states. A plan is a sequence of actions. The cost of a plan is referred to as the path cost. The path cost is a positive.

Problem formulation means choosing a relevant set of states to consider, and a feasible set of operators for moving from one state to another. Search is the process of considering various possible sequences of operators applied to the initial state, and finding out a sequence which culminates in a goal state.

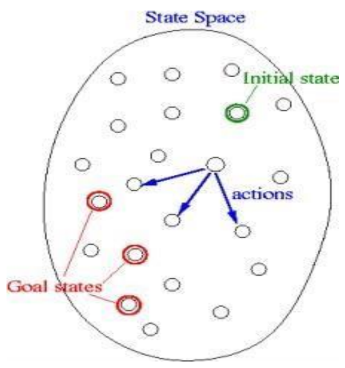
Search Problem is formally described as following:

• S: the full set of states

• S0: the initial state

• A:S→S is a set of operators

• G is the set of final states. Note that G ⊆S



The search problem is to find a sequence of actions which transforms the agent from the initial state to a goal state g ∈ G. A sequence of states is called a path. The cost of a pathis a positive number. In many cases the path cost is computed by taking the sum of the costs ofeach action.

1.2 Example of Problem formulation

Problem Formulation:

Initial state: Watched history of User.

Successor Function: Identifying Pattern in the data gathered through watched history.

Goal test: Recommend the movie suitable for a particular user and user's satisfaction.

Path cost: Time taken to recommend a movie.

CONCLUSION: Thus, we have understood the concept of PEAS description and problem formulation and have implemented it by identifying a problem and writing a PEAS descriptionand problem formulation for the same.

SIGN AND REMARK:

DATE:

R1 (2 Marks)	R2 (4 Marks)	R3 (4 Marks)	Total (10 Marks)	Signature

EXPERIMENT NO. 3

Date of Performance:

Date of Submission :

AIM: To implement Breadth First Search as Uninformed Search.

S/W USED : C/C++/Java/Prolog

THEORY:

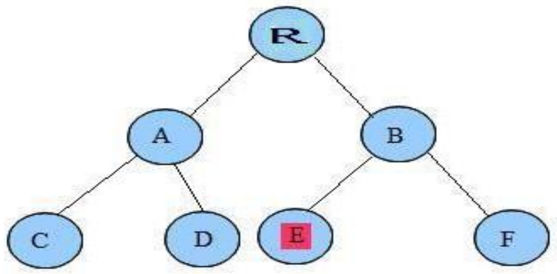
<u>**Breadth First Search**</u>

Breadth First Search (BFS) searches breadth-wise in the problem space. Breadth-First search is like traversing a tree where each node is a state which may a be a potential candidate for solution. It expands nodes from the root of the tree and then generates one level of the tree at a time until a solution is found. It is very easily implemented by maintaining a queue of nodes. Initially the queue contains just the root. In each iteration, node at the head of the queue is removed and then expanded. The generated child nodes are then added to the tail of the queue.

Algorithm: Breadth-First Search

```
1. Create a variable called NODE-LIST and set it to the initial state.
2. Loop until the goal state is found or NODE-LIST is empty.
      a. Remove the first element, say E, from the NODE-LIST. If NODE-LIST was
         empty then quit.
      b. For each way that each rule can match the state described in E do:

   i) Apply the rule to generate a new state.
   ii) If the new state is the goal state, quit and return this state.
   iii) Otherwise add this state to the end of NODE-LIST
```

Since it never generates a node in the tree until all the nodes at shallower levels have been generated, *breadth-first search* always finds a shortest path to a goal. Since each node can be generated in constant time, the amount of time used by Breadth first search is proportional to the number of nodes generated, which is a function of the branching factor b and the solution d. Since the number of nodes at level d is b^d, the total number of nodes generated in the worst case is $b + b^2 + b^3 +\ldots + b^d$ i.e. $O(b^d)$, the asymptotic time complexity of breadth first search.



Breadth First Search

Look at the above tree with nodes starting from root node, R at the first level, A and B at the second level and C, D, E and F at the third level. If we want to search for node E then BFS will search level by level. First it will check if E exists at the root. Then it will check nodes at the second level. Finally it will find E a the third level.

Advantages Of Breadth-First Search

1. Breadth first search will never get trapped exploring the useless path forever except if the no. of successors to any are infinite
2. BFS is **complete** i.e.If there is a solution, BFS will definitely find it out.
3. BFS is **optima**l i.e. If there is more than one solution then BFS can find the minimal one that requires less number of steps.

Disadvantages Of Breadth-First Search

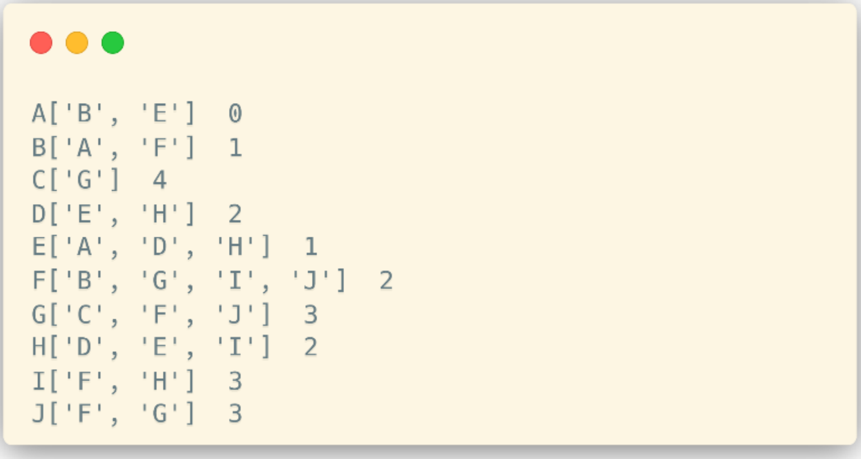
1. The main drawback of Breadth first search is its memory requirement. Since each level of the tree must be saved in order to generate the next level, and the amount of memory is proportional to the number of nodes stored, the space complexity of BFS is O(b^d). As a result, BFS is severely space-bound in practice so will exhaust the memory available on typical computers in a matter of minutes.
2. If the solution is farther away from the root, breath first search will consume lot of time.

Source Code:

```
class Vertex:
        def init (self, n): self.name = n
                self.neighbors = list()

                self.distance = 9999
                self.color = 'black'

        def add_neighbor(self, v):
                if v not in self.neighbors:
                        self.neighbors.append(v)
                        self.neighbors.sort()


class Graph:
        vertices = {}

        def add_vertex(self, vertex):
                if isinstance(vertex, Vertex) and vertex.name not inself.vertices:
                        self.vertices[vertex.name] = vertexreturn True
                else:
                        return False

        def add_edge(self, u, v):
                if u in self.vertices and v in self.vertices:for key, value in
                        self.vertices.items():
```

```
                    if key == u:
                        value.add_neighbor(v)if key
                    == v:
                        value.add_neighbor(u)return
                True
            else:

                return False

        def print_graph(self):
            for key in sorted(list(self.vertices.keys())): print(key +
                str(self.vertices[key].neighbors) +
                       " " + str(self.vertices[key].distance))
        def bfs(self, vert): q = list()
            vert.distance = 0
            vert.color = 'red'
            for v in vert.neighbors: self.vertices[v].distance = vert.distance + 1
                q.append(v)

            while len(q) > 0:u =
                q.pop(0)
                node_u = self.vertices[u]
                node_u.color = 'red'

                for v in node_u.neighbors: node_v =
                    self.vertices[v] if node_v.color ==
                    'black':
                        q.append(v)
                        if node_v.distance > node_u.distance + 1:
                            node_v.distance = node_u.distance + 1


    g = Graph()
    a = Vertex('A') g.add_vertex(a)
    g.add_vertex(Vertex('B'))
    for i in range(ord('A'), ord('K')):
        g.add_vertex(Vertex(chr(i)))


                     edges = ['AB', 'AE', 'BF', 'CG', 'DE', 'DH',
                                    'EH', 'FG', 'FI', 'FJ', 'GJ', 'HI']
    for edge in edges: g.add_edge(edge[:1],
        edge[1:])


    g.bfs(a)
    g.print_graph()
```

Output:



```
A['B', 'E']  0
B['A', 'F']  1
C['G']  4
D['E', 'H']  2
E['A', 'D', 'H']  1
F['B', 'G', 'I', 'J']  2
G['C', 'F', 'J']  3
H['D', 'E', 'I']  2
I['F', 'H']  3
J['F', 'G']  3
```

CONCLUSION:

Thus, we have studied Breadth First Search Algorithm and implemented it in Python programming language.

SIGN AND REMARK:

DATE:

R1 (2 Marks)	R2 (4 Marks)	R3 (4 Marks)	Total (10 Marks)	Signature

EXPERIMENT NO. 4

Date of Performance :

Date of Submission :

AIM: To implement A* Informed Search algorithm.

SOFTWARE USED: Java/Python

THEORY:

Informed search (Heuristics search) makes use of the fact that most problem spaces provide some information that distinguishes among states in terms of their likelihood of leading to a goal. This information is called a heuristic evaluation function. In other words, the goal of a heuristic search is to reduce the number of nodes searched in seeking a goal.

Most widely used best first search form is called A*, which is pronounced as A star. It is a heuristic searching method, and used to minimize the search cost in a given problem. It aims to find the least- cost path from a given initial node to the specific goal. It is an extended form of best-first search algorithm. Best firstsearch algorithm tries to find a solution to minimize the total cost of the search pathway, too. However, the difference from Best-First Search is that A* also takes into account the cost from the start, and not simply the local cost from the previously been node. Best-first search finds a goal state in any predetermined problem space. However, it cannot guarantee that it will choose the shortest path to the goal . For instance, if there are two options to chose from, one of which is a long way from the initial point but has a slightly shorter estimate of distance to the goal, and another that is very close to the initial state but has a slightly longer estimate of distance to the goal, best-first search will always choose to expand next the state with the shorter estimate. The A* algorithm fixes the best first search's this particular drawback.

In short, A* algorithm searches all possible routes from a starting point until it finds the shortest path or cheapest cost to a goal. The terms like shortest path, cheapest cost here refer to a general notion. It could be some other alternative term depending on the problem. A* evaluates nodes by combining g(n) and h(n).

$$\mathbf{f(n)= g(n)+h(n)}$$

f(n) is the total search cost, g(n) is actual lowest cost(shortest distance traveled) of the path from initial start point to the node n, h(n) is the estimated of cost of cheapest(distance) from the node n to a goal node. This part of the equation is also called heuristic function/estimation. At each node, the lowest f value is chosen to be the next step to expand until the goal node is chosen and reached for expansion. (Pearl & Korf, 1987). Whenever the heuristic function satisfies certain conditions, A* search is both complete and optimal (Russell & Norvig, 2003).

Characteristics of A* Search Algorithm:

Admissibility Strategies which guarantee optimal solution, if there is any solution, are called admissible. There are few items which are needed to be satisfied for A* to be admissible A* is optimal, if h(n) is an admissible heuristic. h*(n) = the true minimal cost to goal from n. A heuristic h is admissible if h(n) <= h*(n) for all states n.

Problem with A* Search Algorithm:

According to Pearl & Korf (1987) the main shortcoming of A*, and any best-first search, is its memory requirement. Because the entire open pathway list must be saved, A* is space-limited in practice and is no more practical than breadth first search. For large search spaces, A* will run out of memory

Algorithm:

A* Algorithm pseudocode

The goal node is denoted by node_goal and the source node is denoted by node_start

We maintain two lists:

OPEN and CLOSE:

OPEN consists on nodes that have been visited but not expanded (meaning that sucessors have not been explored yet). This is the list of pending tasks.

CLOSE consists on nodes that have been visited and expanded (sucessors have been explored already and included in the open list, if this was the case).

```
1 Put node_start in the OPEN list with f(node_start) = h(node_start) (initialization)
2 while the OPEN list is not empty {
3 Take from the open list the node node_current with the lowest
4 f(node_current) = g(node_current) + h(node_current)
 5 if node_current is node_goal we have found the solution; break
6 Generate each state node_successor that come after node_current
7 for each node_successor of node_current {
8 Set successor_current_cost = g(node_current) + w(node_current, node_successor)
9 if node_successor is in the OPEN list {
10 if g(node_successor) ≤ successor_current_cost continue (to line 20)
11 } else if node_successor is in the CLOSED list {
12 if g(node_successor) ≤ successor_current_cost continue (to line 20)
13 Move node_successor from the CLOSED list to the OPEN list
14 } else {
 15 Add node_successor to the OPEN list
16 Set h(node_successor) to be the heuristic distance to node_goal
17 }
18 Set g(node_successor) = successor_current_
19 Set the parent of node_successor to node_current
20 }
21 Add node_current to the CLOSED list
22 }
23 if(node_current != node_goal) exit with error (the OPEN list is empty)
```

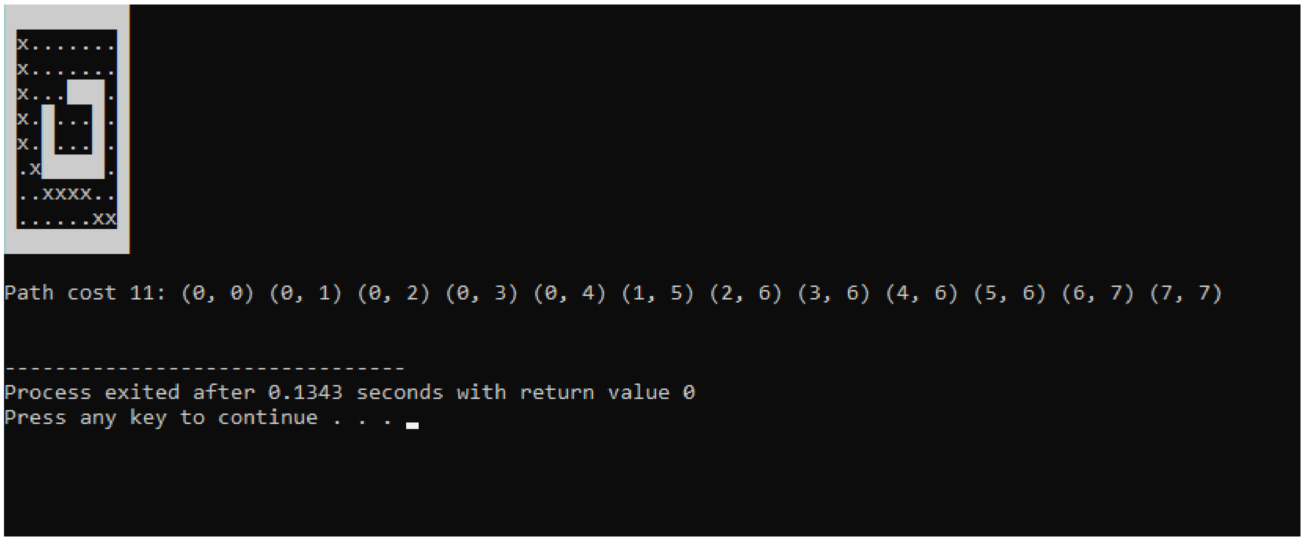
Program:

```
#include <list>
#include <algorithm>
```

```
#include <iostream>
class point
{
public:
    point(int a = 0, int b = 0)
    {
        x =
        a;y =
        b;
    }
    bool operator==(const point &o) { return  o.x == x && o.y == y;
    }point operator+(const point &o) { return point(o.x + x, o.y + y);
    }int x, y;
};
class map
{
public:
    map()
    {
        char t[8][8] = {
            {0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 1, 1, 1, 0}, {0, 0, 1, 0, 0, 0, 1, 0}, {0, 0, 1, 0,
0, 0, 1, 0}, {0, 0, 1, 1, 1, 1, 1, 0}, {0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0}};
        w = h = 8;
        for (int r = 0; r < h; r++)
            for (int s = 0; s < w;
            s++)
                m[s][r] = t[r][s];
    }
    int operator()(int x, int y) { return m[x][y]; }
    char m[8][8];
    int w, h;
};
class node
{
public:
    bool operator==(const node &o) { return pos == o.pos; }
    bool operator==(const point &o) { return pos == o; }
    bool operator<(const node &o) { return dist + cost < o.dist + o.cost;
    }point pos, parent;
    int dist, cost;
};
class aStar
{
public:
    aStar()
    {
        neighbours[0] = point(-1,-1);
        neighbours[1] = point(1, -1);
        neighbours[2] = point(-1, 1);
        neighbours[3] = point(1, 1);
    neighbours[4] = point(0, -1);
    neighbours[5] = point(-1, 0);
    neighbours[6] = point(0, 1);
    neighbours[7] = point(1, 0);
}
int calcDist(point &p)
{
    // need a better heuristic
    int x = end.x - p.x, y = end.y - p.y;
    return (x * x + y * y);
}
bool isValid(point &p)
{
    return (p.x > -1 && p.y > -1 && p.x < m.w && p.y < m.h);
}
bool existPoint(point &p, int cost)
{
    std::list<node>::iterator i;
    i = std::find(closed.begin(), closed.end(), p);
    if (i != closed.end())
    {
        if ((*i).cost + (*i).dist < cost)
            return true;
        else
        {
            closed.erase(i)
            ;return false;
        }
    }
    i = std::find(open.begin(), open.end(), p);
    if (i != open.end())
    {
        if ((*i).cost + (*i).dist < cost)
            return true;
        else
        {
            open.erase(i)
            ;return false;
        }
    }
    return false;
}
bool fillOpen(node &n)
{
    int stepCost, nc,
    dist;point
    neighbour;
    for (int x = 0; x < 8; x++)
    {
        // one can make diagonals have different cost
        stepCost = x < 4 ? 1 : 1;
        neighbour = n.pos + neighbours[x];
        if (neighbour == end)
            return true;
        if (isValid(neighbour) && m(neighbour.x, neighbour.y) != 1)
        {
            nc = stepCost + n.cost;
            dist =
            calcDist(neighbour);
            if (!existPoint(neighbour, nc + dist))
            {
                node m;
                m.cost = nc;
                m.dist =
                dist;
                m.pos = neighbour;
                m.parent = n.pos;
                open.push_back(m)
                ;
            }
        }
    }
    return false;
}
bool search(point &s, point &e, map &mp)
{
    node    n;
    end  =  e;
    start  =  s;
    m  =  mp;
    n.cost    =
    0;n.pos =
    s;
    n.parent = 0;
    n.dist = calcDist(s);
    open.push_back(n);
    while
    (!open.empty())
    {
        // open.sort();
        node n =
        open.front();
        open.pop_front();
        closed.push_back(n)
        ; if (fillOpen(n))
            return true;
    }
    return false;
}
int path(std::list<point> &path)
```

```
        {
            path.push_front(end);
            int cost = 1 + closed.back().cost;
            path.push_front(closed.back().pos);
            point parent = closed.back().parent;
            for (std::list<node>::reverse_iterator i = closed.rbegin(); i != closed.rend(); i++)
            {
                if ((*i).pos == parent && !((*i).pos == start))
                {
                    path.push_front((*i).pos);
                    parent = (*i).parent;
                }
            }
            path.push_front(start);
            return cost;
        }
        map m;
        point end, start;
        point
        neighbours[8];
        std::list<node>
        open;
        std::list<node> closed;
    };
    int main(int argc, char *argv[])
    {
        map m;
        point s, e(7, 7);
        aStar as;
        if (as.search(s, e, m))
        {
            std::list<point>
            path;int c =
            as.path(path);
            for (int y = -1; y < 9; y++)
            {
                for (int x = -1; x < 9; x++)
                {
                    if (x < 0 || y < 0 || x > 7 || y > 7 || m(x, y) == 1)
                        std::cout << char(0xdb);
                    else
                    {
                        if (std::find(path.begin(), path.end(), point(x, y)) != path.end())
                            std::cout << "x";
                        else
                            std::cout << ".";
                    }
                }
                std::cout << "\n";
            }
        std::cout << "\nPath cost " << c << ": ";
        for (std::list<point>::iterator i = path.begin(); i != path.end(); i++)
        {
            std::cout << "(" << (*i).x << ", " << (*i).y << ") ";
        }
    }
    std::cout <<
    "\n\n";return 0;
}
```

INPUT & OUTPUT:



```
x.......
x.......
x...███.
x.█...█.
x.█...█.
.x█████.
..xxxx..
......xx


Path cost 11: (0, 0) (0, 1) (0, 2) (0, 3) (0, 4) (1, 5) (2, 6) (3, 6) (4, 6) (5, 6) (6, 7) (7, 7)


--------------------------------
Process exited after 0.1343 seconds with return value 0
Press any key to continue . . .
```

CONCLUSION:

Thus, we have successfully implemented A* informed search algorithm where we have used both g(n) (actual lowest cost(shortest distance traveled) of the path from initial start point to the node n) and h(n), estimated cost from the node n to a goal node. i..e the heuristic function.

SIGN AND REMARK

DATE

R1 **(2 Marks)**	**R2** **(4 Marks)**	**R3** **(4 Marks)**	**Total** **(10 Marks)**	**Signature**

EXPERIMENT NO. 5

Date of Performance:

Date of Submission :

AIM: Program on Game playing algorithm.

SOFTWARE USED: Java/Python

THEORY:

PROGRAM:

Mini-Max Algorithm in Artificial Intelligence

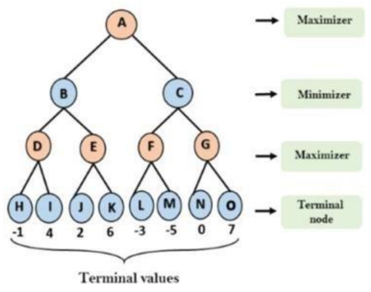
- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac- toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Working of Min-Max Algorithm:

- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.

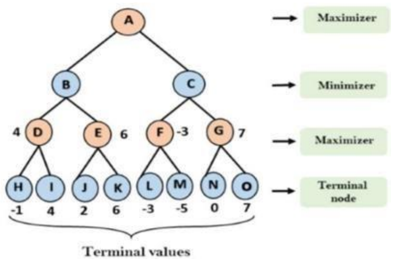
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

Step-1: In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states. In the below 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.



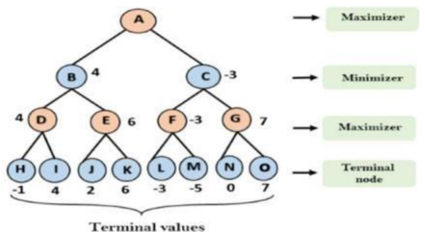
Step 2: Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

- 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



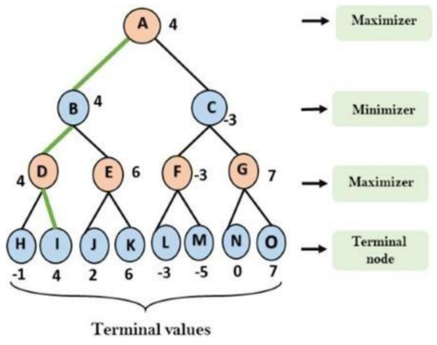
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

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



Step 4: 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



That was the complete workflow of the minimax two player game.

Alpha-Beta Pruning

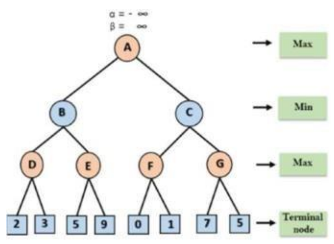
- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha- beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:
    1. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.
    2. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.

- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Working of Alpha-Beta Pruning:

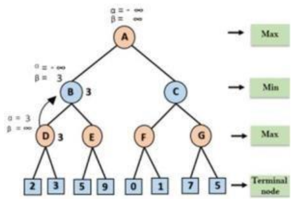
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

Step 1: At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



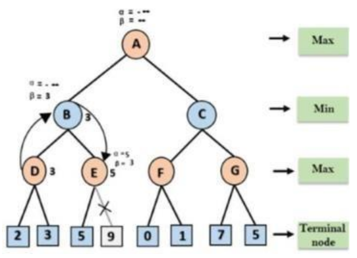
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

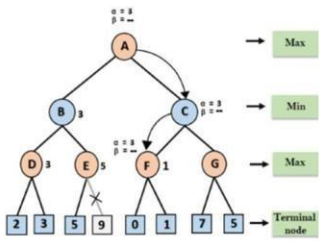
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



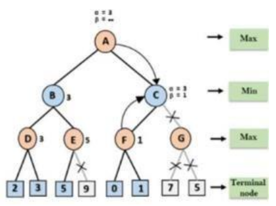
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

At node C, α=3 and β= +∞, and the same values will be passed on to node F.

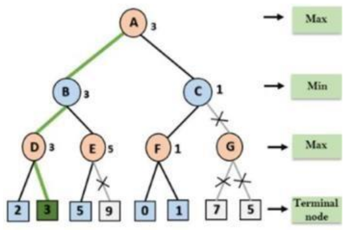
Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



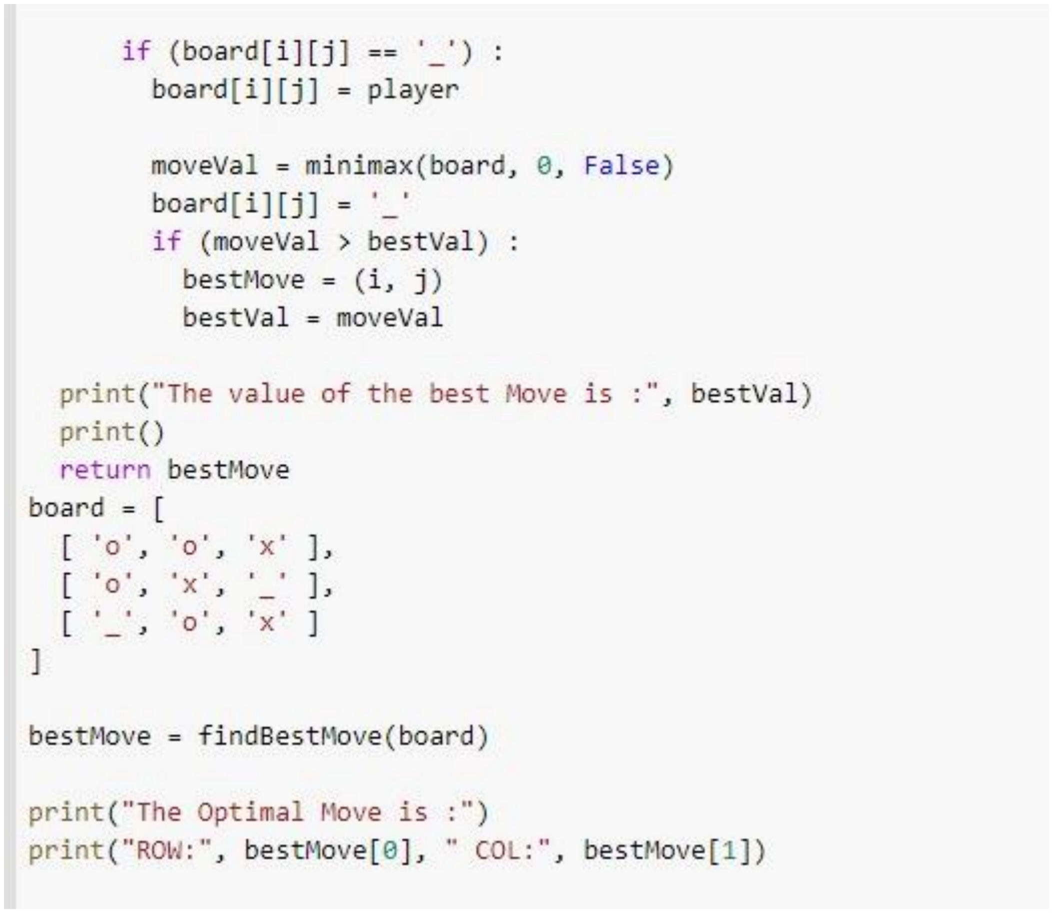
Step 7: Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.

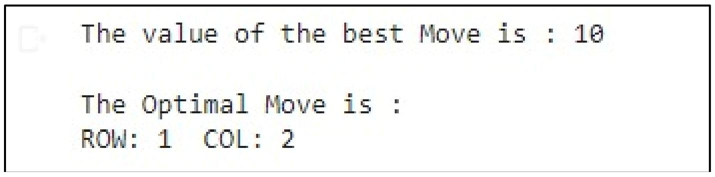


INPUT :



```
      if (board[i][j] == '_') :
        board[i][j] = player

        moveVal = minimax(board, 0, False)
        board[i][j] = '_'
        if (moveVal > bestVal) :
          bestMove = (i, j)
          bestVal = moveVal

  print("The value of the best Move is :", bestVal)
  print()
  return bestMove
board = [
  [ 'o', 'o', 'x' ],
  [ 'o', 'x', '_' ],
  [ '_', 'o', 'x' ]
]

bestMove = findBestMove(board)

print("The Optimal Move is :")
print("ROW:", bestMove[0], " COL:", bestMove[1])
```

OUTPUT:



```
The value of the best Move is : 10

The Optimal Move is :
ROW: 1  COL: 2
```

CONCLUSION:

We have successfully studied Min-Max algorithm and implemented Min-Max Algorithm in Artificial Intelligence using Alpha-Beta Pruning on a Tic – Tac – Toe AI game which is programed using python programming language.

SIGN AND REMARK

DATE

R1 **(2 Marks)**	**R2** **(4 Marks)**	**R3** **(4 Marks)**	**Total** **(10 Marks)**	**Signature**

EXPERIMENT NO. 6

Date of Performance:

Date of Submission :

AIM: Program for first order Logic.(Mini Project)

SOFTWARE USED: Prolog

THEORY:

Knowledge engineering is the base that helped in the creation of expert systems where knowledge is transformed into computer programs. Expert systems have a huge and flexible knowledge base which is integrated with mechanisms that specify how to use the information of the knowledge base and apply it to a variety of situations. These expert systems also use machine learning and deep learning algorithms in order to learn as humans do. Nowadays these Expert systems are used in the education field, healthcare, financial services, manufacturing, etc.

Process of Knowledge Engineering

Knowledge Engineering for different domains is different but it follows, the same set of rules/procedures in order to create expert systems.

1. Task Identification

This is the first initial stage where the task to be performed is defined. In a domain, a specific problem or a combination of several problems would be taken. This task must be realistic and the subject matter expert shall have a clear picture of what it is so that further process can be carried out.

2. Acquisition of Knowledge

Once the problem is well defined then the next step is to gather relevant knowledge and information about the problem. For some problems standard data is used that must be collected, for example, a problem on heat exchanger requires the standard steam table data at x temperature and y pressure what will be the value of enthalpy.

3. Prepare a road map

Once the goal and knowledge base are available the next step is to get the roadmap ready by breaking the goal down into small steps by questionnaires and relevant knowledge base. Here subject matter expert puts his thoughts on how would he make decisions and what parameters would be considered at all stages. There could be several ways to solve some problems, and all should be considered.

4. Encode

Now it’s time to convert this knowledge into computer language. Here the knowledge is encoded by using different functions as well as in some cases, for a specific task, the algorithm is used to create a model. These models are able to make decisions based on available parameters as an expert does, surely the model must be trained and tested on a sufficient amount of data.

5. Evaluation and Debugging

In the process of creating an expert system, at each step, the model should be evaluated and debugged and then added to workflow. Once all small tasks are evaluated, they are assembled to create one whole expert system. This system is again evaluated on similar problems and Debugged if any issue is there.

6. Justification & Explanation

Here the model is justified for the given task and working is explained.

First-Order Logic in Artificial intelligence

In the topic of Propositional logic, we have seen that how to represent statements using propositional logic. But unfortunately, in propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

- **"Some humans are intelligent", or**
- **"Sachin likes cricket."**

To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

First-Order logic:

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
  - **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus, ...
  - **Relations: It can be unary relation such as:** red, round, is adjacent, **or n-any relation such as:** the sister of, brother of, has color, comes between
  - **Function:** Father of, best friend, third inning of, end of, ...
- As a natural language, first-order logic also has two main parts:
  - **Syntax**
  - **Semantics**

Syntax of First-Order logic:

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as **Predicate (term1, term2,......., term n)**.

Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky).

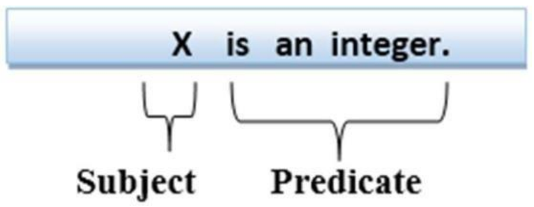
Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- **Subject:** Subject is the main part of the statement.
- **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:

1. **Universal Quantifier, (for all, everyone, everything)**
2. **Existential quantifier, (for some, at least one).**

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

Note: In universal quantifier we use implication "→".

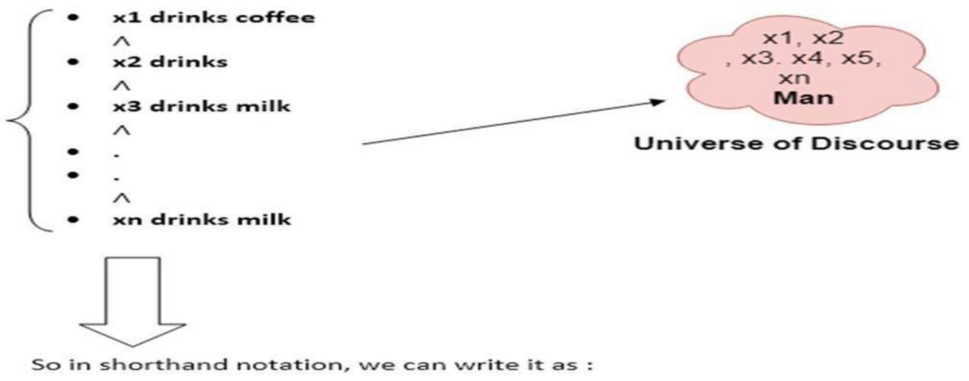
If x is a variable, then ∀x is read as:

- **For all x**
- **For each x**
- **For every x.**

Example:

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:



∀x man(x) → drink (x, coffee).

It will be read as: There are all x where x is a man who drink coffee.

Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

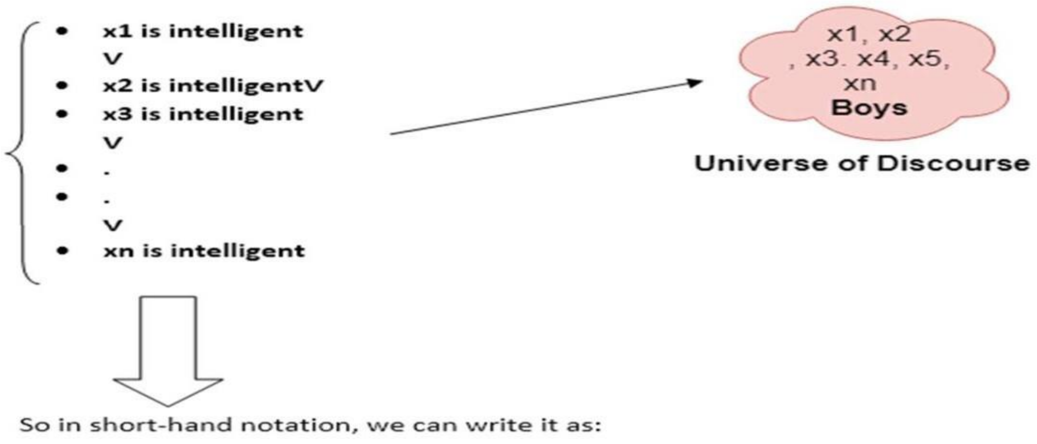
Note: In Existential quantifier we always use AND or Conjunction symbol (∧).

If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

- **There exists a 'x.'**
- **For some 'x.'**
- **For at least one 'x.'**

Example:

Some boys are intelligent.



∃x: boys(x) ∧ intelligent(x)

It will be read as: There are some x where x is a boy who is intelligent.

Points to remember:

- The main connective for universal quantifier ∀ is implication →.
- The main connective for existential quantifier ∃ is and ∧.

Properties of Quantifiers:

- In universal quantifier, ∀x∀y is similar to ∀y∀x.
- In Existential quantifier, ∃x∃y is similar to ∃y∃x.
- ∃x∀y is not similar to ∀y∃x.

Some Examples of FOL using quantifier:

1. All birds fly.
In this question the predicate is "**fly(bird)**."
And since there are all birds who fly so it will be represented as follows.
∀x bird(x) →fly(x).

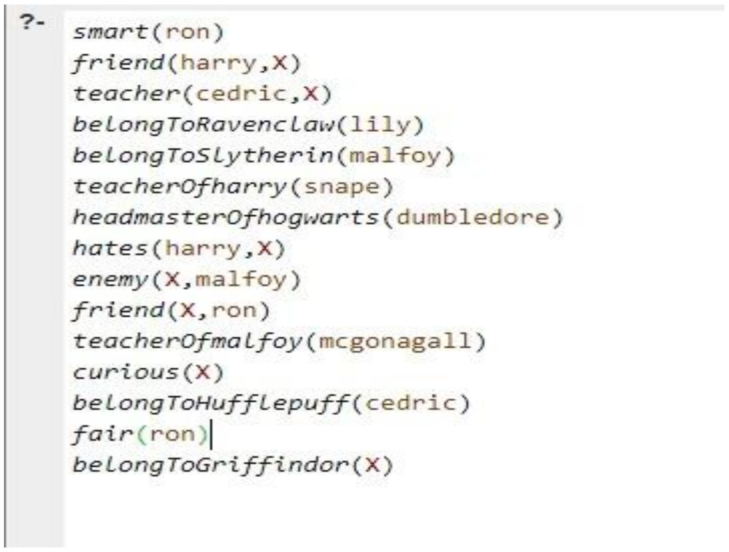
2. Every man respects his parent.
In this question, the predicate is "**respect(x, y)," where x=man, and y= parent**.
Since there is every man so will use ∀, and it will be represented as follows:
∀x man(x) → respects (x, parent).

3. Some boys play cricket.
In this question, the predicate is "**play(x, y)**," where x= boys, and y= game. Since there are some boys so we will use ∃**, and it will be represented as**:
∃x boys(x) → play(x, cricket).

4. Not all students like both Mathematics and Science.
In this question, the predicate is "**like(x, y)," where x= student, and y= subject**.
Since there are not all students, so we will use ∀ **with negation, so** following representation for this:
¬∀ (x) [student(x) → like(x, Mathematics) ∧like(x, Science)].

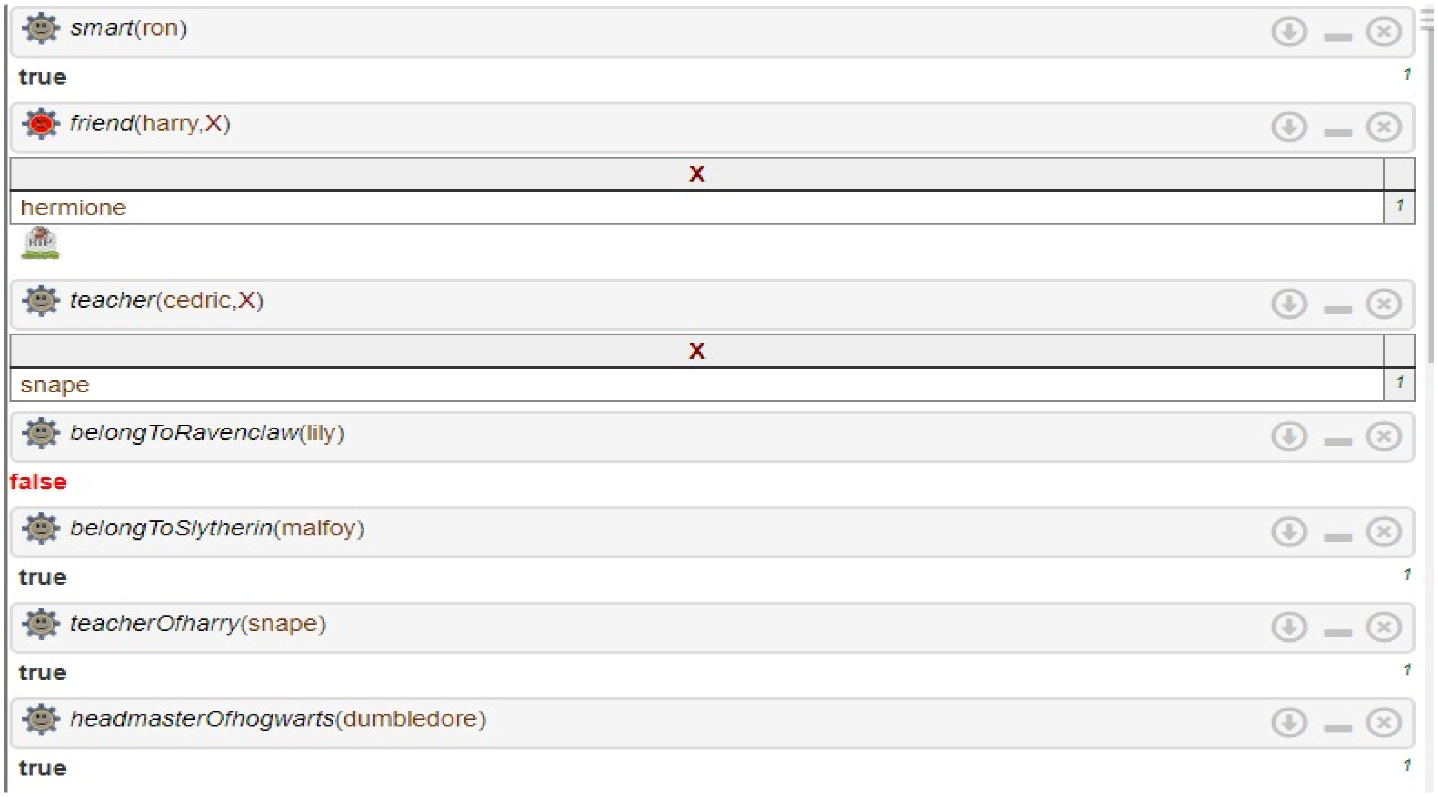
5. Only one student failed in Mathematics.
In this question, the predicate is "**failed(x, y)," where x= student, and y= subject**.
Since there is only one student who failed in Mathematics, so we will use following representation for this:
∃(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)].

PROGRAM:

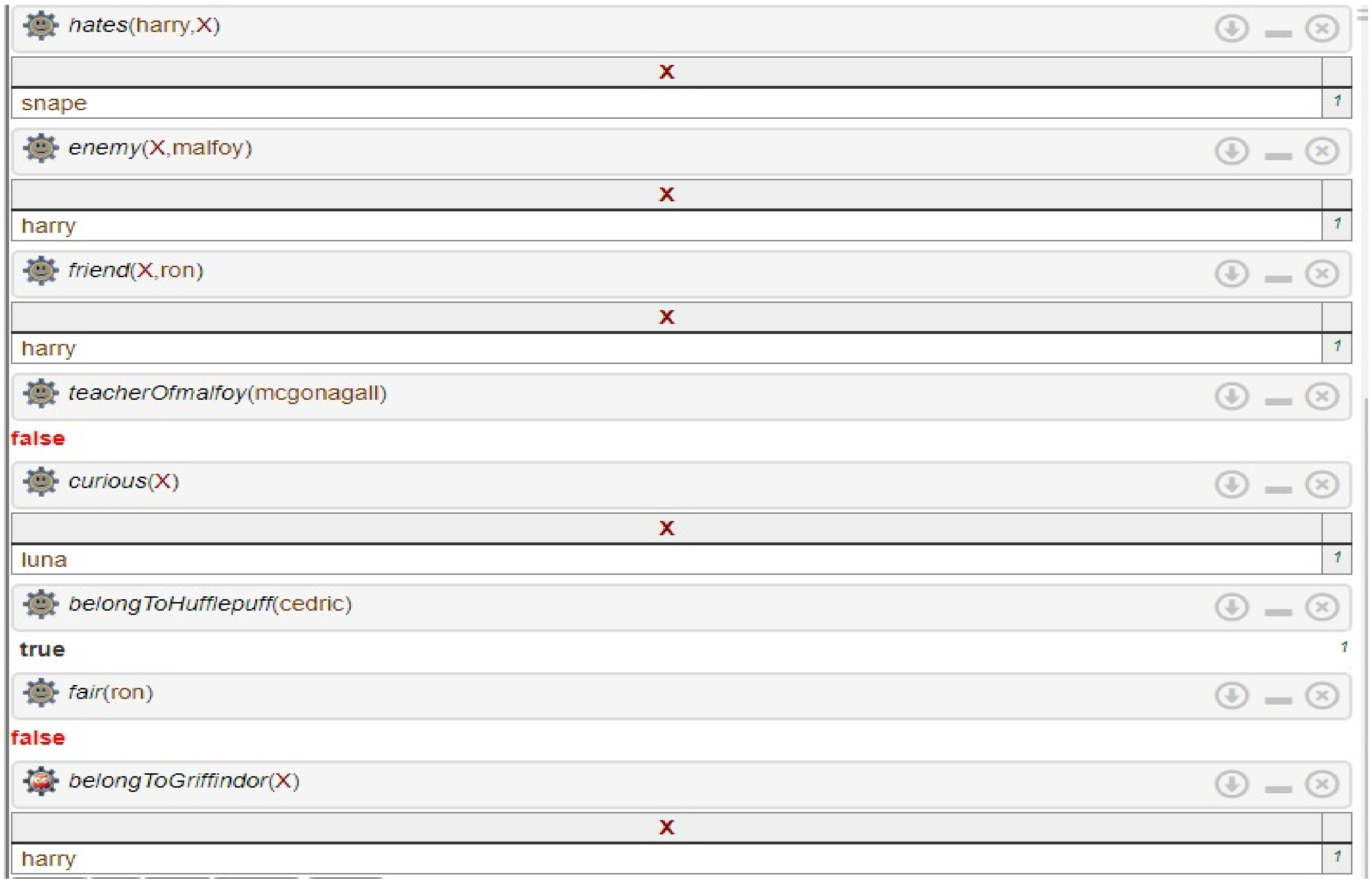


```
?-  smart(ron)
    friend(harry,X)
    teacher(cedric,X)
    belongToRavenclaw(lily)
    belongToSlytherin(malfoy)
    teacherOfharry(snape)
    headmasterOfhogwarts(dumbledore)
    hates(harry,X)
    enemy(X,malfoy)
    friend(X,ron)
    teacherOfmalfoy(mcgonagall)
    curious(X)
    belongToHufflepuff(cedric)
    fair(ron)
    belongToGriffindor(X)
```

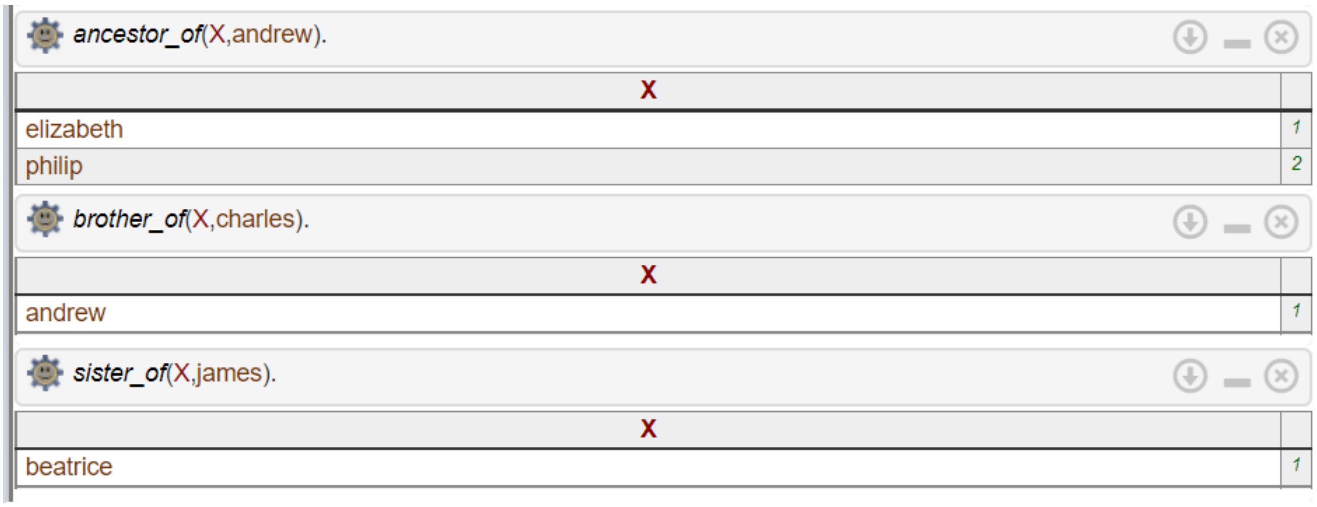
OUTPUT:



```
smart(ron)
true
friend(harry,X)
X
hermione
teacher(cedric,X)
X
snape
belongToRavenclaw(lily)
false
belongToSlytherin(malfoy)
true
teacherOfharry(snape)
true
headmasterOfhogwarts(dumbledore)
true
```



```
hates(harry,X)
X
snape
enemy(X,malfoy)
X
harry
friend(X,ron)
X
harry
teacherOfmalfoy(mcgonagall)
false
curious(X)
X
luna
belongToHufflepuff(cedric)
true
fair(ron)
false
belongToGriffindor(X)
X
harry
```



```
ancestor_of(X,andrew).
X
elizabeth    1
philip       2
brother_of(X,charles).
X
andrew       1
sister_of(X,james).
X
beatrice     1
```

CONCLUSION:

Thus, here we successfully implemented family tree using Prolog. Here we can conclude that First- order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.

SIGN AND REMARK

DATE

R1 (2 Marks)	R2 (2 Marks)	R3 (1 Marks)	Total (5 Marks)	Signature

EXPERIMENT NO. 7

Date of Performance:

Date of Submission :

AIM: Planning Programming

SOFTWARE USED: STRIP

THEORY:

What is planning in AI?

- The planning in Artificial Intelligence is about the decision making tasks performed by the robots or computer programs to achieve a specific goal.
- The execution of planning is about choosing a sequence of actions with a high likelihood to complete the specific task.

Blocks-World planning problem

- The blocks-world problem is known as **Sussman Anomaly.**
- Noninterleaved planners of the early 1970s were unable to solve this problem, hence it is considered as anomalous.
- When two subgoals G1 and G2 are given, a noninterleaved planner produces either a plan for G1 concatenated with a plan for G2, or vice-versa.
- In blocks-world problem, three blocks labeled as 'A', 'B', 'C' are allowed to rest on the flat surface. The given condition is that only one block can be moved at a time to achieve the goal.
- The start state and goal state are shown in the following diagram.

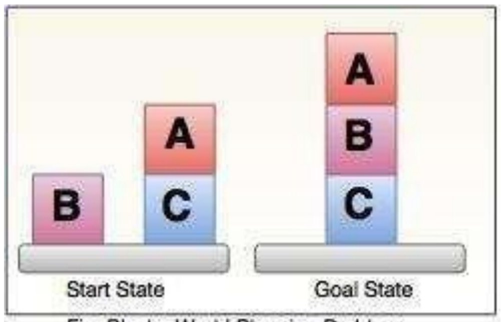


Fig: Blocks-World Planning Problem

Components of Planning System

The planning consists of following important steps:

- Choose the best rule for applying the next rule based on the best available heuristics.
- Apply the chosen rule for computing the new problem state.
- Detect when a solution has been found.

- Detect dead ends so that they can be abandoned and the system's effort is directed in more fruitful directions.
- Detect when an almost correct solution has been found.

Goal stack planning

This is one of the most important planning algorithms, which is specifically used by **STRIPS.**

- The stack is used in an algorithm to hold the action and satisfy the goal. A knowledge base is used to hold the current state, actions.
- Goal stack is similar to a node in a search tree, where the branches are created if there is a choice of an action.

The important steps of the algorithm are as stated below:

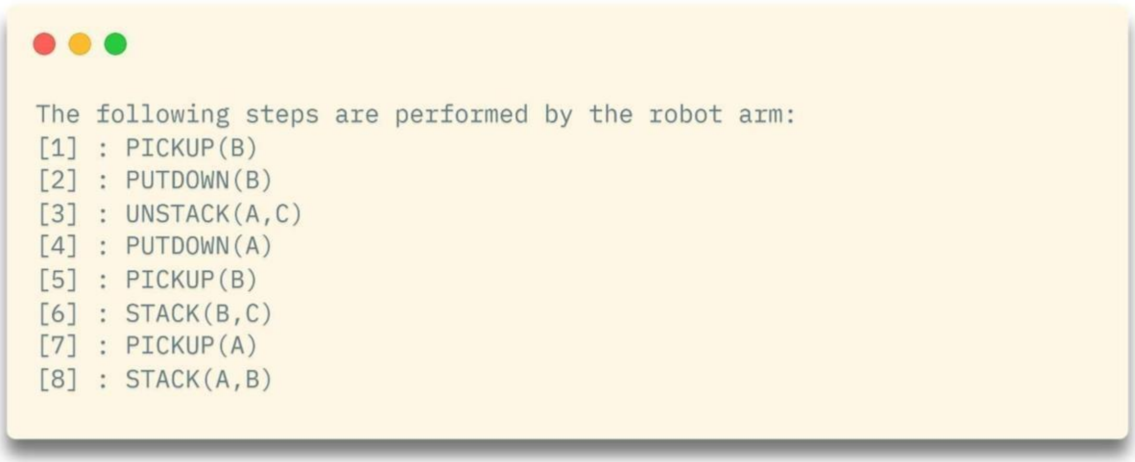
i. Start by pushing the original goal on the stack. Repeat this until the stack becomes empty. If stack top is a compound goal, then push its unsatisfied subgoals on the stack.
ii. If stack top is a single unsatisfied goal then, replace it by an action and push the action's precondition on the stack to satisfy the condition.
iii. If stack top is an action, pop it from the stack, execute it and change the knowledge base by the effects of the action.
iv. If stack top is a satisfied goal, pop it from the stack.

PROGRAM:

INPUT :

```
#PREDICATE - ON, ONTABLE, CLEAR, HOLDING, ARMEMPTY
class PREDICATE:
    def_str_(self):
        pass
    def __repr__(self):
        pass
    def __eq_(self, other):
        pass
    def __hash__(self):
        pass
    def get_action(self, world_state):
        pass

#OPERATIONS - Stack, Unstack, Pickup, Putdown
class Operation:
    def_str_(self):
        pass
    def __repr__(self):
        pass
    def __eq_(self, other):
        pass
    def precondition(self):
        pass
    def delete(self):
        pass
    def add(self):
        pass

class ON(PREDICATE):
    def_init_(self, X, Y):
        self.X = X
        self.Y  =  Y
      def_str_(self):
        return "ON({X},{Y})".format(X=self.X, Y=self.Y)
    def __repr__(self):
        return self._str_()
    def __eq__(self, other):
        return self.__dict__== other.__dict___and self.__class___== other.__class___
    def __hash__(self):
        return hash(str(self))
    def get_action(self, world_state):
        return StackOp(self.X, self.Y)

class ONTABLE(PREDICATE):
    def __init__(self, X):
        self.X =  X
    def_str_(self):
        return "ONTABLE({X})".format(X=self.X)
    def __repr__(self):
        return self._str_()
    def __eq__(self, other):
        return self.__dict__== other.__dict___and self.__class___== other.__class___
    def __hash__(self):
        return hash(str(self))
    def get_action(self, world_state):
        return PutdownOp(self.X)

class CLEAR(PREDICATE):
    def __init__(self, X):
        self.X =  X
    def_str_(self):
        return "CLEAR({X})".format(X=self.X)
        self.X = X
    def __repr__(self):
        return self._str_()
    def __eq__(self, other):
        return self.__dict__== other.__dict___and self.__class___== other.__class___
    def __hash__(self):
        return hash(str(self))
    def get_action(self, world_state):
        for predicate in world_state:
            # If Block is on another block, unstack
            if isinstance(predicate, ON) and predicate.Y == self.X:
                return UnstackOp(predicate.X, predicate.Y)
        return None

class HOLDING(PREDICATE):
    def __init__(self, X):
        self.X =  X
    def_str_(self):
        return "HOLDING({X})".format(X=self.X)
    def __repr__(self):
        return self._str_()
    def __eq__(self, other):
        return self.__dict__== other.__dict___and self.__class___== other.__class___
    def __hash__(self):
        return hash(str(self))
    def get_action(self, world_state):
        X = self.X
        # If block is on table, pick up
        if ONTABLE(X) in world_state:
            return PickupOp(X)
        # If block is on another block, unstack
        else:
            for predicate in world_state:
                if isinstance(predicate, ON) and predicate.X == X:
                    return UnstackOp(X, predicate.Y)

class ARMEMPTY(PREDICATE):
    def __init__(self):
        pass
    def_str_(self): return
        "ARMEMPTY"
    def __repr__(self):
        return self._str_()
    def __eq__(self, other):
        return self.__dict__== other.__dict___and self.__class___== other.__class___
    def __hash__(self):
        return hash(str(self))
    def get_action(self, world_state=[]):
        for predicate in world_state:
            if isinstance(predicate, HOLDING):
                return PutdownOp(predicate.X)
        return None

class StackOp(Operation):
    def_init_(self, X, Y):
        self.X = X
        self.Y  =  Y
     def_str_(self):
        return "STACK({X},{Y})".format(X=self.X, Y=self.Y)
    def __repr__(self):
        return self._str_()
    def __eq__(self, other):
        return self.__dict__== other.__dict___and self.__class___== other.__class___
    def precondition(self):
        return [CLEAR(self.Y), HOLDING(self.X)]
    def delete(self):
        return [CLEAR(self.Y), HOLDING(self.X)]
    def add(self):
        return [ARMEMPTY(), ON(self.X, self.Y)]

class UnstackOp(Operation):
    def_init_(self, X, Y):
         self.X = X
         self.Y =  Y
     def_str_(self):
        return "UNSTACK({X},{Y})".format(X=self.X, Y=self.Y)
    def __repr__(self):
        return self._str_()
    def __eq__(self, other):
        return self.__dict__== other.__dict___and self.__class___== other.__class___
    def precondition(self):
        return [ARMEMPTY(), ON(self.X, self.Y), CLEAR(self.X)]
    def delete(self):
        return [ARMEMPTY(), ON(self.X, self.Y)]
    def add(self):
        return [CLEAR(self.Y), HOLDING(self.X)]

class PickupOp(Operation):
    def __init__(self, X):
        self.X  = X
    def_str_(self):
        return "PICKUP({X})".format(X=self.X)
    def __repr__(self):
        return self._str_()
    def __eq__(self, other):
        return self.__dict__== other.__dict___and self.__class___== other.__class___
    def precondition(self):
        return [CLEAR(self.X), ONTABLE(self.X), ARMEMPTY()]
    def delete(self):
        return [ARMEMPTY(), ONTABLE(self.X)]
    def add(self):
        return [HOLDING(self.X)]

class PutdownOp(Operation):
    def __init__(self, X):
        self.X  = X
    def_str_(self):
        return "PUTDOWN({X})".format(X=self.X)
    def __repr__(self):
        return self._str_()
    def __eq__(self, other):
        return self.__dict__== other.__dict___and self.__class___== other.__class___
    def precondition(self):
        return [HOLDING(self.X)]
    def delete(self):
        return [HOLDING(self.X)]
    def add(self):
        return [ARMEMPTY(), ONTABLE(self.X)]

def isPredicate(obj):
    predicates = [ON, ONTABLE, CLEAR, HOLDING, ARMEMPTY]
    for predicate in predicates:
        if isinstance(obj, predicate):
            return True
    return False

def isOperation(obj):
    operations = [StackOp, UnstackOp, PickupOp, PutdownOp]
    for operation in operations:
        if isinstance(obj, operation):
            return True
    return False

def arm_status(world_state):
    for predicate in world_state:
        if isinstance(predicate, HOLDING):
            return predicate
    return ARMEMPTY()

class GoalStackPlanner:
    def __init_(self, initial_state, goal_state):
        self.initial_state = initial_state
        self.goal_state = goal_state
    def get_steps(self):
        # Store Steps
        steps = []
        # Program Stack
        stack = []
        # World State/Knowledge Base
        world_state = self.initial_state.copy()
        # Initially push the goal_state as compound goal onto the stack
        stack.append(self.goal_state.copy())
        # Repeat until the stack is empty
        while len(stack) != 0:
            # Get the top of the stack
            stack_top = stack[-1]
            # If Stack Top is Compound Goal, push its unsatisfied goals onto stack
            if type(stack_top) is list:
                compound_goal = stack.pop()
                for goal in compound_goal:
                    if goal not in world_state:
                        stack.append(goal)
            # If Stack Top is an action
            elif isOperation(stack_top):
                # Peek the operation
               operation = stack[-1]
                all_preconditions_satisfied = True
                # Check if any precondition is unsatisfied and push it onto program stack
                for predicate in operation.delete():
                    if predicate not in world_state:
                        all_preconditions_satisfied = False
                        stack.append(predicate)
                # If all preconditions are satisfied, pop operation from stack and execute
it
                if all_preconditions_satisfied:
                    stack.pop()
                    steps.append(operation)
                    for predicate in operation.delete():
                        world_state.remove(predicate)
                    for predicate in operation.add():
                        world_state.append(predicate)
            # If Stack Top is a single satisfied goal
            elif stack_top in world_state:
                stack.pop()
            # If Stack Top is a single unsatisfied goal
            else:
                unsatisfied_goal = stack.pop()
                # Replace Unsatisfied Goal with an action that can complete it
                action = unsatisfied_goal.get_action(world_state)
                stack.append(action)
                # Push Precondition on the stack
                for predicate in action.precondition():
                    if predicate not in world_state:
                        stack.append(predicate)
            return steps

if __name__== '__main__':
    initial_state = [
        ON('A', 'C'),
        ONTABLE('B'),
        ONTABLE('C'),
        CLEAR('A'),
        CLEAR('B'),
        ARMEMPTY()
    ]
    goal_state = [
        ON('A', 'B'),
        ON('B', 'C'),
        ONTABLE('C'),
        CLEAR('A'),
        ARMEMPTY()
    ]
    goal_stack = GoalStackPlanner(
        initial_state=initial_state, goal_state=goal_state)
    steps = goal_stack.get_steps()
    print("The following steps are performed by the robot arm:")
    for i in range(len(steps)):
        print(f"[{i+1}] : {steps[i]}")
```

OUTPUT :



```
The following steps are performed by the robot arm:
[1] : PICKUP(B)
[2] : PUTDOWN(B)
[3] : UNSTACK(A,C)
[4] : PUTDOWN(A)
[5] : PICKUP(B)
[6] : STACK(B,C)
[7] : PICKUP(A)
[8] : STACK(A,B)
```

CONCLUSION:

We have studied and understood the concept of Planning programming and have implemented it by solving the Blocks-World planning problem by solving it in STRIPS in python programming language.

SIGN AND REMARK :

DATE :

R1 **(2 Marks)**	**R2** **(4 Marks)**	**R3** **(4 Marks)**	**Total** **(10 Marks)**	**Signature**

EXPERIMENT NO. 8

Date of Performance:

Date of Submission :

AIM: Implementation for Bayes Belief Network

SOFTWARE USED: Java/Python

THEORY:

Bayesian Belief Network in artificial intelligence. Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as:

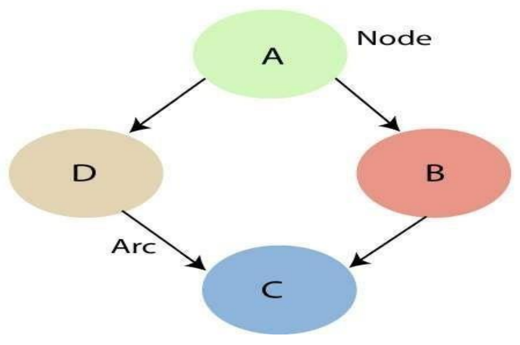
"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**. Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts: **Directed Acyclic Graph, Table of conditional probabilities.**

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram**. **A Bayesian network graph is made up of nodes and Arcs (directed links), where:**



- Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.
- **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph. These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other
    - **In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.**
    - **If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.**
    - **Node C is independent of node A.**

The Bayesian network has mainly two components: **Causal Component and Actual numbers**

Each node in the Bayesian network has condition probability distribution **$P(X_i | Parent(X_i))$**, which determines the effect of the parent on that node. Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables x1, x2, x3, , xn, then the probabilities of a different combination of x1, x2, x3.. xn, are known as Joint probability distribution.

$P[x_1, x_2, x_3, , x_n]$, it can be written as the following way in terms of the joint probability distribution.

$= P[x_1| x_2, x_3,....., x_n]P[x_2, x_3, , x_n]$

$= P[x_1| x_2, x_3,....., x_n]P[x_2|x_3,....., x_n] P[x_{n-1}|x_n]P[x_n]$.

In general for each variable Xi, we can write the equation as:

$P(X_i|X_{i-1}, \quad , X_1) = P(X_i |Parents(X_i))$

Explanation of Bayesian network:

Let's understand the Bayesian network through an example by creating a directed acyclic graph:

Example: Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

Problem:

Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.

Solution:

- The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.
- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- The conditional distributions for each node are given as conditional probabilities table or CPT.
- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
- In CPT, a boolean variable with k boolean parents contains 2^K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

List of all events occurring in this network:

- **Burglary (B)**
- **Earthquake(E)**
- **Alarm(A)**
- **David Calls(D)**
- **Sophia calls(S)**

We can write the events of problem statement in the form of probability: **P[D, S, A, B, E]**, can rewrite the above probability statement using joint probability distribution:

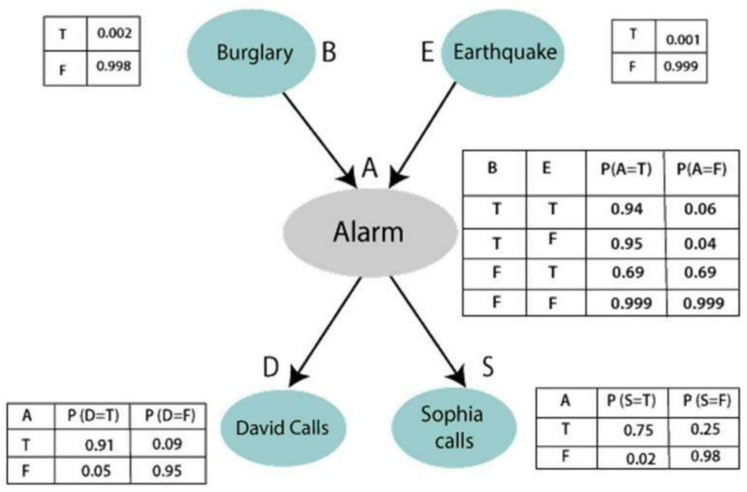
P[D, S, A, B, E]= P[D | S, A, B, E]. P[S, A, B, E]

=P[D | S, A, B, E]. P[S | A, B, E]. P[A, B, E]

= P [D| A]. P [S| A, B, E]. P[A, B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B |E]. P[E]



Let's take the observed probability for the Burglary and earthquake component:

P(B= True) = 0.002, which is the probability of burglary.

P(B= False)= 0.998, which is the probability of no burglary.

P(E= True)= 0.001, which is the probability of a minor earthquake

P(E= False)= 0.999, Which is the probability that an earthquake not occurred.

We can provide the conditional probabilities as per the below tables:

Conditional probability table for Alarm A:

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	**E**	**P(A= True)**	**P(A= False)**
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

$P(S, D, A, \neg B, \neg E) = P(S|A) * P(D|A) * P(A|\neg B \wedge \neg E) * P(\neg B) * P(\neg E)$.

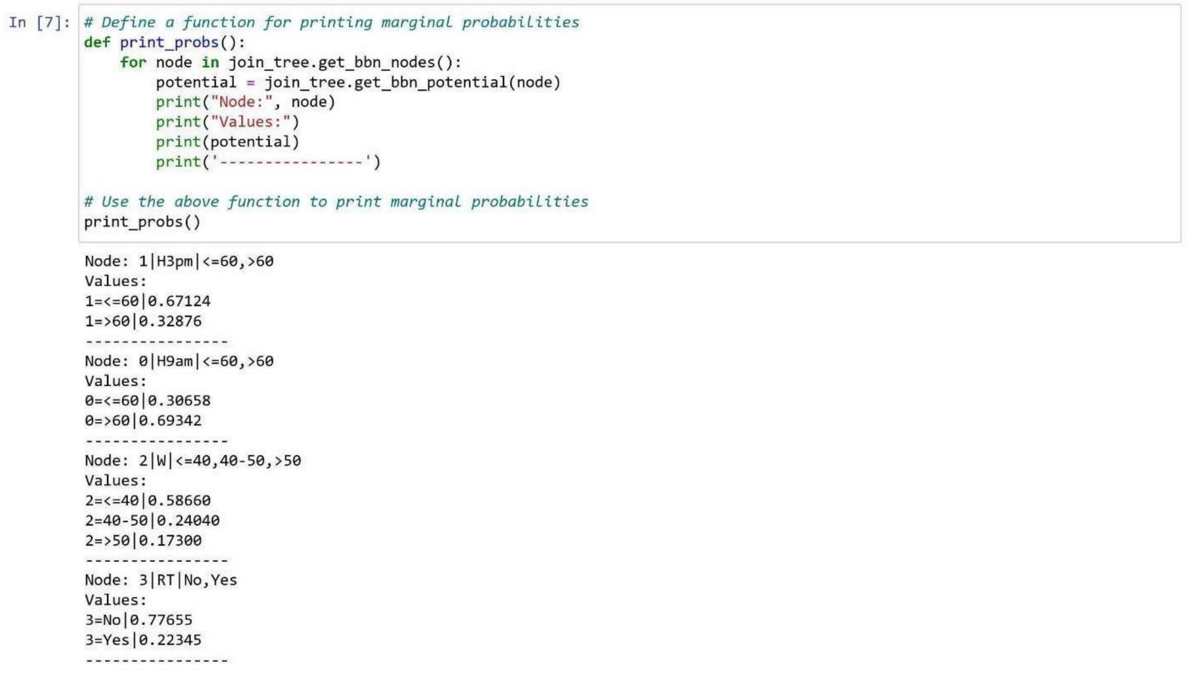
$= 0.75 * 0.91 * 0.001 * 0.998 * 0.999$

$= 0.00068045$.

Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

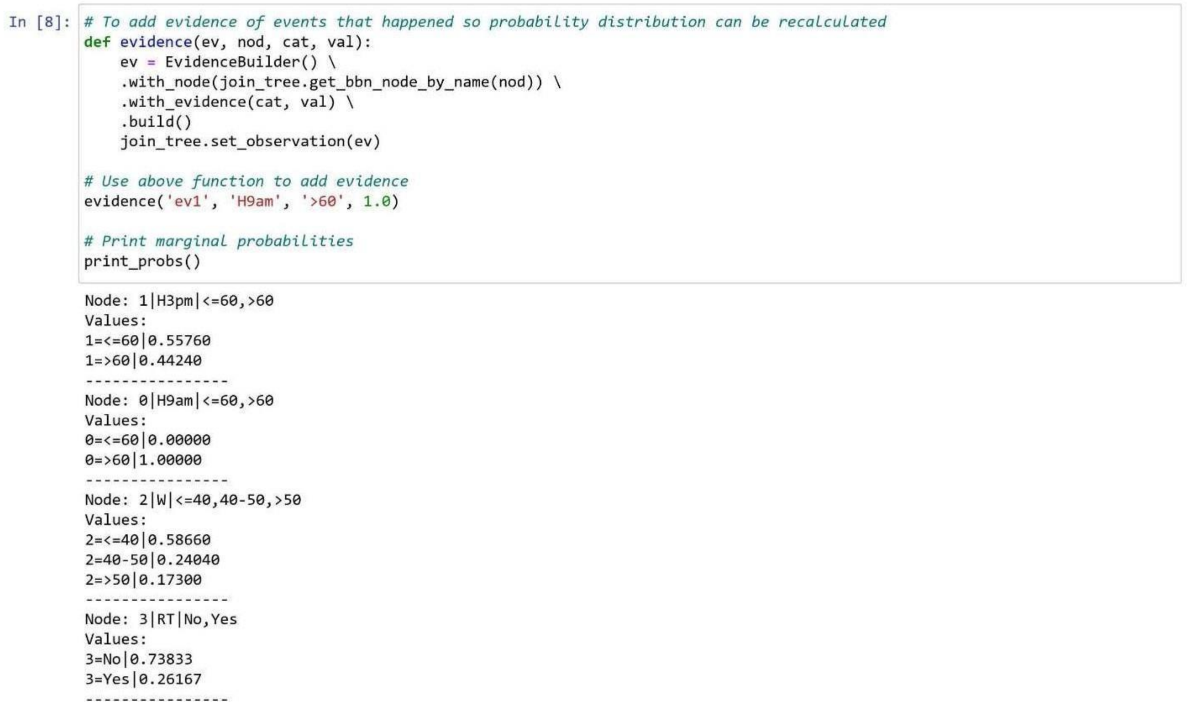
PROGRAM:

INPUT & OUTPUT:



```
In [7]: # Define a function for printing marginal probabilities
        def print_probs():
            for node in join_tree.get_bbn_nodes():
                potential = join_tree.get_bbn_potential(node)
                print("Node:", node)
                print("Values:")
                print(potential)
                print('----------------')

        # Use the above function to print marginal probabilities
        print_probs()
```

```
Node: 1|H3pm|<=60,>60
Values:
1=<=60|0.67124
1=>60|0.32876
----------------
Node: 0|H9am|<=60,>60
Values:
0=<=60|0.30658
0=>60|0.69342
----------------
Node: 2|W|<=40,40-50,>50
Values:
2=<=40|0.58660
2=40-50|0.24040
2=>50|0.17300
----------------
Node: 3|RT|No,Yes
Values:
3=No|0.77655
3=Yes|0.22345
----------------
```



```
In [8]: # To add evidence of events that happened so probability distribution can be recalculated
        def evidence(ev, nod, cat, val):
            ev = EvidenceBuilder() \
            .with_node(join_tree.get_bbn_node_by_name(nod)) \
            .with_evidence(cat, val) \
            .build()
            join_tree.set_observation(ev)

        # Use above function to add evidence
        evidence('ev1', 'H9am', '>60', 1.0)

        # Print marginal probabilities
        print_probs()
```

```
Node: 1|H3pm|<=60,>60
Values:
1=<=60|0.55760
1=>60|0.44240
----------------
Node: 0|H9am|<=60,>60
Values:
0=<=60|0.00000
0=>60|1.00000
----------------
Node: 2|W|<=40,40-50,>50
Values:
2=<=40|0.58660
2=40-50|0.24040
2=>50|0.17300
----------------
Node: 3|RT|No,Yes
Values:
3=No|0.73833
3=Yes|0.26167
----------------
```

```
Tn §1] : Tmpoo* pandas as pd # jot data mon tpuLaMon
            import netuorkx as nx # for dro+yt ng graphs
import mat plot1:tb. pyplot as pit # jar dnaHing graphs

# -for creating Bayes! an BeLief NeI+yorbs (BBN)
from pybbn . graph. dag :tmport Bbn
foam pybbn . graph . edge TmpooT Edge, EdgeType
from pybdn . graph. jot ntree Report E vldenceBuTT der
-firom pybbn . graph. node import BbnNode
 firoei pybbn . graph . variable import Variable
from pybbn . pptc. i nferencecontrolTer import EnferenceControlter
```

```
In [2]  i ¥  Pendos op i ons ro disp£eg more co Lm os
        pd . opt hons . di splay . ma xcc or umns=SO

        # Pead in tfie ueothen dado csv
        d-F=pd . read_c sv( ' ueatherAuS. c sv ' , encodlng= ' ut F-8 ' )

        # Drop ne cords where torget 8o tn Tomorrop=Noñ
        df=df t pd . is nu IN ( df [ ' RaSn7omo rro u ' § ) ==False§

        # For other columns with missing values, fill them in with column mean
        df=df.fillna(df.mean())

        k Crethe bands -for ver i at›£ es that we nant 8o use in the mode £
        df-[ ' DindGu s tS peedCat ' ) -df[ ' h!ind6u s I Speed ' ] . apply lambda x : ' 0 . ‹ -48 '    i-I- x< -40 els e
                                                                         ' 1 . AO— 50 ' i£ 4B• • • =D8 else ' 2. • 50 ' )
        df t ' I4umid ity9anCat ' ]=df[ ' Huntd ity9am ' § . apply ( lembda x: ' 1. › 66 ' T-fi x›6B eLse '6. < = 66 ' )
        df ' I4umld1ty3poCat ' j=dT[ ' Humid ity 3pm' § . apply ( lambda x: ' T . › 66 ' TT x>66 eLse '6. <=66 ' )

        It Show a snophsot oJ dole d
        F-. head( )

        c• \Users\rfshf\AppDa La \Loca1\Temp/1pykerne1 sycy/¥1s98sae2e. py:zz : FutupeMar•ning: Dropp Ing oT nut sance columns in Dat.app ame
        reductions (u:ttfi ' numeric_on1y=None ' ) :ts deprecated; in a future vers:ton this u1L1 raise TypeError . Select only va1 id
        columns before ca11Eng the reduction.
          dT=d-F. fiill na(dT . mean ( ) )

Out[2]:
```

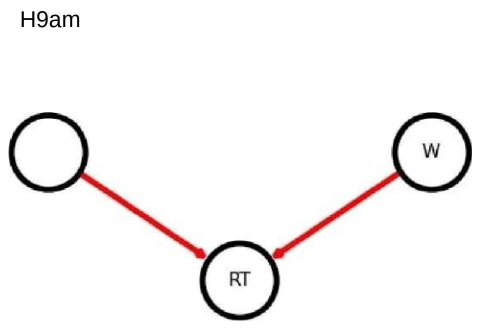
0	2008-12-01	Albury	13.4	22.9	0.6	5.469824	7.624853	W	44.0	W	WNW	20.0
,	1 % 2	Albury	7.4	25.1	0.0	6.469B24	7.6248*i3	WNW	44.0	new	WSW	4.0
2	2008-12-03	Albury	12.0	25.7	0.0	5.469824	7.624853	WSW	46.0	W	WSW	19.0
3	2008-12-04	Albury	9.2	28.0	0.0	5.469824	7.624853	NE	24.0	sE	E	11.0
J	g p§	Albury	17,d	32.3	1.0	6.469824	7.624803	W	41.0	ENE	NW	7.0

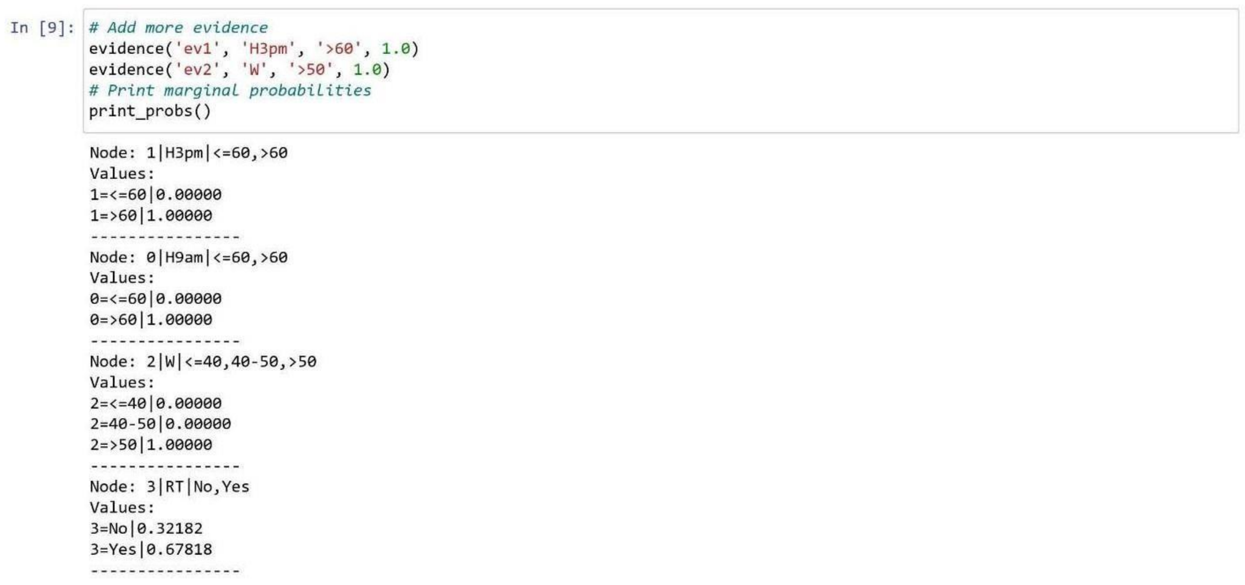
```
En [3]: a Create nodes 6y monuoLly typtng in probabi Lil:ies
        H9am = B bnNode(Va r:table(6, ' H9 an ' , [ ' < =6B ' , ' › 66 ' § }, §8. 36658, 0. 69342] )
        H3pm = BbnNode(Va rTab be(I, ' H3 pn ' ( ' < =6B, '       ' › 66 ' ] ) , §6 . 92827, B . 87173,
                                                                          6. 55760, B . 44246] )

        N - BbnNode (Va rTabte (2, ' N ' , [ ' < =4B ' , ' 4B- 5B ' , ' > SB ' ] ) , §8. 5866B, B . 2S04B, B • I'38B 3
        RT = 8bnNode ( Var i ab 1 e ( 3,  ' RT ' ,  § ' IJo ' ,  ' Yes ' § ) ,  [6. 923t4, e.67686,
                                                               6. 89B72, 0. 10928,
                                                               6.76668, 6.23992,
                                                               6. 64256, 6. 35756,

                                                               6. 32182, 0. 678T8] )
```

```
tn t4 § : # Tfiis ;{-un c I i on he Lps to co I cu L ate pnodod I I Sty di s I r i but i on ant en goes into BBN (no I e, con hond L e up to 2 ponen Is)
        de+ probs ( dat a, chtld, parent1=Non e, pa rent2=None) •

                # €aL curate probo6t £tstes
                prob=pd. crosstab(data[child§,  ' Empty '   margTns=FaLse, noraalize=' columns ' ) . sort_:Index() .to_numpy ( ) . reshape( -I}  . to1
            e1:tT parent:I! =None:
                    # Checb i- - chi Ld node has 1 pareni- or 2 parent:s
                    T+ parent2==Nene:

                        prob=pd . crosstab(data[ pa rent:I], data[ ch:IN d] , marglns=FaLse, normalize='1ndex').soptpindex().to numpy().re sfl

                     '* # Cac I ucote probabi I i I i e s
                        prob=pd . c ros stab( tdata [ parents § , data[parent *I ] , data [ chi Id § , margins=Fatse norma1i ze= ' index ' ) . so rt_index ( )
            else: print ( " Error in Probab i lity Frequency Ca1cu1at:tons" )
```

```
H9am = BbnNode(Va r:Table(0, ' H9 an ' , [ ' <=60 ' , ' › 60 ' ] ) , pnob s ( df, ch:ltd= ' FlumT dity9amC at ' } )
H3pm = BbnNode(Var:t able(1, 'H3pn', [ '<=68 ', '›68']), probs(d F, chTJd='Hunidity3pmCat ', parent:I='Humidity 9amCat'))
N = BbnNode(Var:tabie ( 2 'N', [ '‹=4B', '48-58' '›58'§), probs(df, chi Id ='Ni ndGust SpeedCat '))
RT = 8bnNode(Variable ( 3, ' RR ' , § ' No ' , ' Yes ' § ) , probs(dT, child= ' Ra:tn7omo r row ' , parents= ' Humidity3priC at ' , parents= ' Ni ndGust



     . add_node(H9atr) \
     . add_node(H3 pm) \
     . add_node(kl) \
     . add_node ( RT) \
     . add_edge ( Edge(H9am, H3pri, E dgeType . 0I RECTED ) ) \
     . add edge( E dge Id3pm, RT, Edge Type . DIRECTED) } \
     .add_edge(Edge(M, RT, EdgeType.DIRECTED) }

# Con verb the BBtJ to o yoin tree
```

```
1n [ 6] • , # Set    code posi I iozs
pos - Je: ( - i, z) , i : ( - i, e . s) , z: (i, e . s), 3: (e, —i jJ

# Sei- opl tons /or graph l oaks
opt:ton s =
     " -Font_s iz e " : 16,
     " node_s:l z e " : 4808,

     "edgecolors": "black",
     "edge_color": "red",
     "linewidths": 5,
     "width": 5,}

# Generate graph
n, d = bbn. t.o_nx@ raph ( )
nx. draw(n, uith_1abe1s=True, 1abe1s=d, pos=pos, " ”options )

# Updote margins and print tfie grope

ax.margins(0.10)

plt.show()
```





```
In [9]: # Add more evidence
        evidence('ev1', 'H3pm', '>60', 1.0)
        evidence('ev2', 'W', '>50', 1.0)
        # Print marginal probabilities
        print_probs()
```

```
Node: 1|H3pm|<=60,>60
Values:
1=<=60|0.00000
1=>60|1.00000
----------------
Node: 0|H9am|<=60,>60
Values:
0=<=60|0.00000
0=>60|1.00000
----------------
Node: 2|W|<=40,40-50,>50
Values:
2=<=40|0.00000
2=40-50|0.00000
2=>50|1.00000
----------------
Node: 3|RT|No,Yes
Values:
3=No|0.32182
3=Yes|0.67818
----------------
```

CONCLUSION:

We have successfully studied and understood Bayes Belief Network and implemented the same in python programming language.

SIGN AND REMARK

DATE

R1 **(2 Marks)**	**R2** **(4 Marks)**	**R3** **(4 Marks)**	**Total** **(10 Marks)**	**Signature**