Exp 1 ; Implementation of Prolog program in the form of mini project for First Order Predicate Logic.

Theory : Prolog is a logical and declarative programming language. The name itself, Prolog, is short for PROgramming in LOGic. Prolog is a logic language that is particularly suited to programs that involve sf computation. It is a frequently used language in Artificial Intelligence where manipulation of symbols and inference about them is a common task.

Code

```
male (rohit).
male (jayesh).
male (nitin).
male (bhikubhai).
male (bhupendra).
male (suresh).
male (raman).
female (savita).
female (rangula).
female (kunta).
female (kavita).
female (varsha).
parent (nitin, rohit).
parent (nitin, varsha).
parent (kunta, varsha).
parent (nitin,jayesh).
parent (bhikubhai,nitin).
parent (savita, nitin).
parent (bhikubhai, bhupendra).
parent (savita,bhupendra).
parent (kunta, rohit).
parent (kunta,jayesh).
parent (rangula, kunta).
parent (raman, kunta).
parent (bhikubhai, suresh).
parent (kavita, suresh).
 /*Rules*/
mother (X, Y): -parent (X, Y), female (X).
father (X, Y): -parent (X, Y), male (X).
grandmother (X, Y): -mother (X,Z), parent (Z,Y).
grandfather (X, Y) :-father (X,Z), parent (2,Y).
grandparent (X, Y): -parent (X,Z), parent (2,Y).
brother (X, Y) :-parent (2,X), parent (2,Y), male (X),X\==Y.
sister (X, Y): -parent (Z, X), parent (Z, Y), female (X),X\-Y.
wife (X, Y): -parent (X, 2), parent (Y, Z), female (X), male (X).
uncle (X, Y) :-brother (X, Z), father (Z, Y). aunt (X, Y)-wife (X,Z),
uncle (Z, Y). cousin (X, Y): -parent (21,X), parent (22,Y), sibling (Z1, Z2).
sibling (X, Y): -parent (Z, X), parent (Z, Y),X\==Y.
```

Exp2 : Formulate State space and develop a two state vacuum cleaner simple reflex agent

Theory : Vacuum agent program is very small because
o Ignoring the percept history (cuts down no. of possibilities)
o when the current square is dirty, the action does not depend on the location.

● It can be implemented using condition–action rule
● E.g.
● if car-in-front-is-braking
then initiate-braking.
● Agent program: Build a general-purpose interpreter for condition–action rules and then create rule sets for specific task environments.

Code :
```
def vacuum_cleaner():
    cost = 0

    remain = 2
    vacuum_pos = input("Enter Position of Vacuum Cleaner (A/B): ")
    a = input("Is room A dirty (T/F): ")
    b = input("Is room B dirty (T/F): ")

    if(a == 'F' or b == 'F'):
        if(vacuum_pos == 'A' and a=='F'):
            print("A is already Clean\nMoving to B\nB is also Clean\nTotal Cost = 1")
        else:
            print("B is already Clean\nMoving to A\nA is also Clean\nTotal Cost = 1")
        return
    while(remain > 0):
        if(vacuum_pos == 'A'):
            print("Currently Vacuum is at location A")
            if(a == 'T'):
                print("Room A is being cleaned...")         #CLEAN PRINT
                cost += 1 #cost for suck                    #COST
                a = 'F' #mark clean                         #MARK CLEAN
            if(b == 'T'):
                print("Moving RIGHT to Room B...")          #MOVE PRINT
                vacuum_pos = 'B'                            #VACUUM POS
                cost += 1 #cost for moving RIGHT            #COST

        if(vacuum_pos == 'B'):
            print("Currently Vacuum is at location B")
            if(b == 'T'):
                print("Room B is being cleaned...")
                cost += 1 #cost for suck
                b = 'F'  #mark clean
            if(a == 'T'):
               print("Moving LEFT to Room A...")
               vacuum_pos = 'A'
               cost += 1 #cost for moving LEFT
        remain -= 1;
```

```
    print("Both rooms are cleaned\n cost: ",cost)

vacuum_cleaner()
```

Exp 3 Uninformed Search- BFS and DFS Algorithm

Theory : The aim of BFS algorithm is to traverse the graph as close as possible to the root node. Queue is used in the implementation of the breadth first search.
The aim of DFS algorithm is to traverse the graph in such a way that it tries to go far from the root node. Stack is used in the implementation of the depth first search
code :

```
from queue import Queue
# Function to perform BFS traversal
def bfs(graph, start_node):
    visited = set()  # Set to keep track of visited nodes
    q = Queue()  # Queue for BFS traversal
    visited.add(start_node)
    q.put(start_node)


    print("Path: ")
    while not q.empty():
        node = q.get() #DEQUEUE
        print(node, end=' ')
        for neighbor in graph[node]:
            if neighbor not in visited:
                visited.add(neighbor)
                q.put(neighbor)

# Function to perform DFS traversal
def dfs(graph, start_node):
    visited = set()  # Set to keep track of visited nodes
    stack = [start_node]  # Stack for DFS traversal

    print("Path: ")
    while stack:
        node = stack.pop()
        if node not in visited:
            visited.add(node)
            print(node, end=' ')
            for neighbor in graph[node]:
                stack.append(neighbor)

# Input graph
graph = {}
print("Enter the graph:")
while True:
    u, v = input().split()
    if u == "-1" and v == "-1":
        break
```

```
    if u not in graph:
        graph[u] = []
    if v not in graph:
        graph[v] = []
    graph[u].append(v)
    graph[v].append(u)
print("Graph: ",graph)

start_node = input("Enter the start node: ")
print("\n\n1.BFS TRAVERSAL")
bfs(graph, start_node)
print("\n\n1.DFS TRAVERSAL\n")
dfs(graph, start_node)
```

```
from queue import Queue
# Function to perform BFS traversal
def bfs(graph, start_node):
    visited = set()  # Set to keep track of visited nodes
    q = Queue()  # Queue for BFS traversal
    q.put(start_node)
    visited.add(start_node)

    print("Path: ")
    while not q.empty():
        node = q.get()
        print(node, end=' ')
        for neighbor in graph[node]:
            if neighbor not in visited:
                visited.add(neighbor)
                q.put(neighbor)

# Function to perform DFS traversal
def dfs(graph, start_node):
    visited = set()  # Set to keep track of visited nodes
    stack = [start_node]  # Stack for DFS traversal

    print("Path: ")
    while stack:
        node = stack.pop()
        if node not in visited:
            visited.add(node)
            print(node, end=' ')
            for neighbor in graph[node]:
                stack.append(neighbor)

# Input graph
```

```
graph = {}
print("Enter the graph:")
while True:
    u, v = input().split()
    if u == "-1" and v == "-1":
        break
    if u not in graph:
        graph[u] = []
    if v not in graph:
        graph[v] = []
    graph[u].append(v)
    graph[v].append(u)
print("Graph: ",graph)

start_node = input("Enter the start node: ")
print("\n\n1.BFS TRAVERSAL")
bfs(graph, start_node)
print("\n\n1.DFS TRAVERSAL\n")
dfs(graph, start_node)
```

```
PS C:\Users\91996\OneDrive\Desktop\Python> python -u "c:\Users\9199
6\OneDrive\Desktop\Python\DFS.py"
Enter the graph:
A B
A D
A E
B C
B E
C E
C F
C G
D E
E F
F G
-1 -1
Graph:  {'A': ['B', 'D', 'E'], 'B': ['A', 'C', 'E'], 'D': ['A', 'E'
], 'E': ['A', 'B', 'C', 'D', 'F'], 'C': ['B', 'E', 'F', 'G'], 'F':
['C', 'E', 'G'], 'G': ['C', 'F']}
Enter the start node: A


1.BFS TRAVERSAL
Path:
A B D E C F G

2.DFS TRAVERSAL
Path:
A E F G C B D
PS C:\Users\91996\OneDrive\Desktop\Python>
```

Exp 4 Informed Search- A* algorithm

Theory : A * algorithm is a searching algorithm that searches for the shortest path between the initial and the final state. It is used in various applications, such as maps

class Node:

```
    def __init__(self,data,level,fval):
        """ Initialize the node with the data, level of the node and the calculated fvalue """
        self.data = data
        self.level = level
        self.fval = fval
    def generate_child(self):
        """ Generate child nodes from the given node by moving the blank space
            either in the four directions {up,down,left,right} """
        x,y = self.find(self.data,'_')
        """ val_list contains position values for moving the blank space in either of
            the 4 directions [up,down,left,right] respectively. """
        val_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]
        children = []
        for i in val_list:
            child = self.shuffle(self.data,x,y,i[0],i[1])
            if child is not None:
                child_node = Node(child,self.level+1,0)
                children.append(child_node)
        return children
    def shuffle(self,puz,x1,y1,x2,y2):
        if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):
            temp_puz = []
            temp_puz = self.copy(puz)
            temp = temp_puz[x2][y2]
            temp_puz[x2][y2] = temp_puz[x1][y1]
            temp_puz[x1][y1] = temp
            return temp_puz
        else:
            return None
    def copy(self,root):
        """ Copy function to create a similar matrix of the given node"""
        temp = []
        for i in root:
            t = []
            for j in i:
                t.append(j)
            temp.append(t)
        return temp
    def find(self,puz,x):
        for i in range(0,len(self.data)):
            for j in range(0,len(self.data)):
                if puz[i][j] == x:
                    return i,j
class Puzzle:
    def __init__(self,size):
        """ Initialize the puzzle size by the specified size,open and closed lists to empty """
        self.n = size
        self.open = []
        self.closed = []
```

```
    def accept(self):
        """ Accepts the puzzle from the user """
        puz = []
        for i in range(0,self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz
    def f(self,start,goal):
        """ Heuristic Function to calculate hueristic value f(x) = h(x) + g(x) """
        return self.h(start.data,goal)+start.level
    def h(self,start,goal):
        """ Calculates the different between the given puzzles """
        temp = 0
        for i in range(0,self.n):
            for j in range(0,self.n):
                if start[i][j] != goal[i][j] and start[i][j] != '_':
                    temp += 1
        return temp
    def process(self):
        """ Accept Start and Goal Puzzle state"""
        print("Enter the start state matrix \n")
        start = self.accept()
        print("Enter the goal state matrix \n")
        goal = self.accept()
        start = Node(start,0,0)
        start.fval = self.f(start,goal)
        """ Put the start node in the open list"""
        self.open.append(start)
        print("\n\n")
        while True:
            cur = self.open[0]
            print("")
            for i in cur.data:
                for j in i:
                    print(j,end=" ")
                print("")
            """ If the difference between current and goal node is 0 we have reached the goal
node"""
            if(self.h(cur.data,goal) == 0):
                break
            for i in cur.generate_child():
                i.fval = self.f(i,goal)
                self.open.append(i)
            self.closed.append(cur)
            del self.open[0]
            self.open.sort(key = lambda x:x.fval,reverse=False)
puz = Puzzle(3)
puz.process()
```

```
PS C:\Users\PC-04\Documents\ai> python expp5.py
Enter the start state matrix

2 8 3
1 6 4
7 _ 5
Enter the goal state matrix

1 2 3
8 _ 4
7 6 5
```

Exp 5 Implementation for Bayes Belief Network

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

```
def main():
    print("Bayessian Network\n")
    l = int(input("Enter the total no. of levels in your network: "))
    level = {}
    dependency = {}
    event = {}
    for i in range(l):
        level[i] = list(input(f"Enter nodes at level {i}: ").strip().split())
        if i != 0:
            for j in level[i]:
                dependency[j] = [x for x in level[i-1]]
        else:
            for j in level[i]:
                dependency[j] = []
    probability = {}
    for key, value in dependency.items():
        event[key] = bool
        if len(value) == 0:
            print(f"Enter True probability of {key}:")
            probability[key] = list(map(float, input().strip().split()))
        if len(value) == 1:
            print(f"Enter T and F probabilty of {dependency[key]} for True of {key}")
            probability[key] = list(map(float, input().strip().split()))
        if len(value) == 2:
            print(f"Enter TT, TF, FT, FF probability of {dependency[key]} for True of {key}:")
            probability[key] = list(map(float,input().strip().split()))
    while(True):
        print("\nEnter events values: (True/False)")
        for i in event:
            event[i] = input(f"Event of {i}: ") == "True"
        cal = {}
        for key, value in dependency.items():
            if len(value) == 0:
```

```
            if event[key]:
                cal[key] = probability[key][0]
            elif not event[key]:
                cal[key] = 1.0 - probability[key][0]
        if len(value) == 1:
            if event[key] and event[value[0]]:
                cal[key] = probability[key][0]
            elif event[key] and not event[value[0]]:
                cal[key] = probability[key][1]
            elif not event[key] and event[value[0]]:
                cal[key] = 1 - probability[key][0]
            elif not event[key] and not event[value[0]]:
                cal[key] = 1 - probability[key][1]
        if len(value) == 2:
            if event[key] and event[value[0]] and event[value[1]]:
                cal[key] = probability[key][0]
            elif event[key] and event[value[0]] and not event[value[1]]:cal[key] = probability[key][1]
            elif event[key] and not event[value[0]] and event[value[1]]:cal[key] = probability[key][2]
            elif event[key] and not event[value[0]] and not event[value[1]]:cal[key] = probability[key][3]
            elif not event[key] and event[value[0]] and event[value[1]]:cal[key] = 1 - probability[key][0]
            elif not event[key] and event[value[0]] and not event[value[1]]:cal[key] = 1 -
probability[key][1]
            elif not event[key] and not event[value[0]] and event[value[1]]:cal[key] = 1 -
probability[key][2]
            elif not event[key] and not event[value[0]] and not event[value[1]]:cal[key] = 1 -
probability[key][3]
    print(f"Probaility: {cal}")
    solution = 1.0
    for val in cal.values():
        solution *= val
    print(f"The probability for given scenario is {solution}")
    stop = input("Do you wish to stop?(y/n)").lower().strip() == "y"
    if stop:
        break
main()
```

```
PS C:\Users\swara\OneDrive\Desktop\Swarali\IP> & C:/Users/swara/AppData/Local/Programs/Python/Python310/py
ayesnetwork.py
Bayessian Network

Enter the total no. of levels in your network: 3
Enter nodes at level 0: Burglary Earthquake
Enter nodes at level 1: Alarm
Enter nodes at level 2: JohnCalls MaryCalls
Enter True probability of Burglary:
0.001
Enter True probability of Earthquake:
0.002
Enter TT, TF, FT, FF probability of ['Burglary', 'Earthquake'] for True of Alarm:
0.95 0.94 0.29 0.001
Enter T and F probabilty of ['Alarm'] for True of JohnCalls
0.90 0.05
Enter T and F probabilty of ['Alarm'] for True of MaryCalls
0.70 0.01

Enter events values: (True/False)
Event of Burglary: False
Event of Earthquake: False
Event of Alarm: True
Event of JohnCalls: True
Event of MaryCalls: True
Probaility: {'Burglary': 0.999, 'Earthquake': 0.998, 'Alarm': 0.001, 'JohnCalls': 0.9, 'MaryCalls': 0.7}
The probability for given scenario is 0.0006281112599999999
Do you wish to stop?(y/n)n
```

```
Enter events values: (True/False)
Event of Burglary: True
Event of Earthquake: False
Event of Alarm: True
Event of JohnCalls: False
Event of MaryCalls: True
Probaility: {'Burglary': 0.001, 'Earthquake': 0.998, 'Alarm': 0.94, 'JohnCalls': 0.09999999999999998, 'MaryCalls': 0.7}
Enter events values: (True/False)
Event of Burglary: True
Event of Earthquake: False
Event of Alarm: True
Event of JohnCalls: True
Event of MaryCalls: True
Probaility: {'Burglary': 0.001, 'Earthquake': 0.998, 'Alarm': 0.94, 'JohnCalls': 0.9, 'MaryCalls': 0.7}
The probability for given scenario is 0.0005910156
Do you wish to stop?(y/n)n

Enter events values: (True/False)
Event of Burglary: False
Event of Earthquake: False
Event of Alarm: False
Event of JohnCalls: False
Event of MaryCalls: False
Probaility: {'Burglary': 0.999, 'Earthquake': 0.998, 'Alarm': 0.999, 'JohnCalls': 0.95, 'MaryCalls': 0.99}The probability for given scenario is 0.9367427006189
999
Do you wish to stop?(y/n)y
PS C:\Users\swara\OneDrive\Desktop\Swarali\IP>
```