DEWAN VS GROUP OF INSTITUTIONS INDIA

Department of Artificial Intelligence

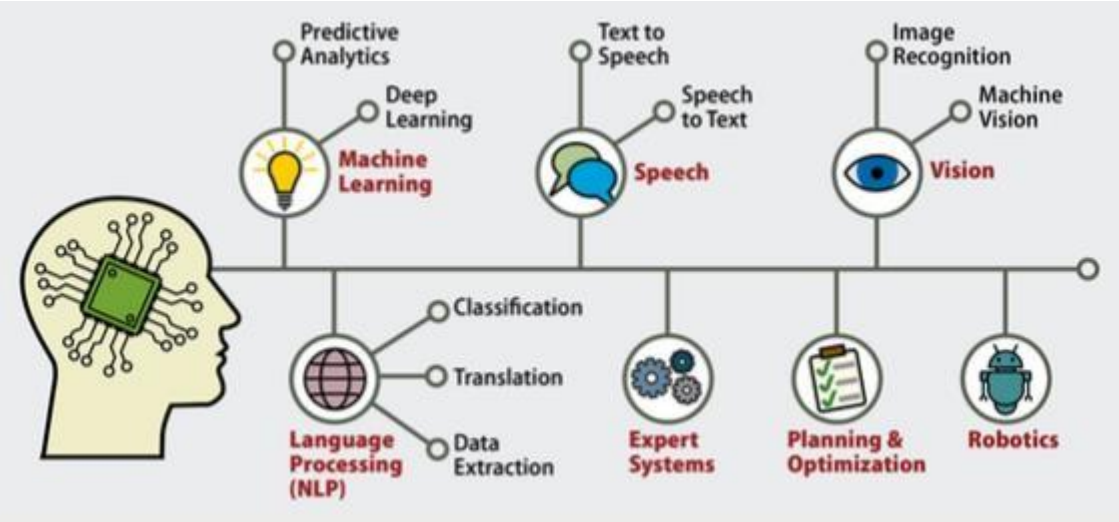
Academic Year 2025 (Session 2025-26)

LAB MANUAL

ARTIFICIAL INTELLIGENCE

BCAI-551

B. Tech. 3rd Year AI-**DS / AI-ML**



COURSE OUTCOME (CO)	
AFTER STUDYING THIS COURSE, THE STUDENT WILL BE ABLE TO	
CO1	understand and remember the basic concepts of prolog programming. Students also able to learn different logic programming languages.
CO2	Students acquire skill to identify the given problem and design the rule-based systems.
CO3	Students are able to apply and analyse various problem-solving techniques on artificial intelligent problems.

List of Programs:

S. No.	***Name of Practical***	***Date***	***Signature***
1.	STUDY OF PROLOG		
2.	WRITE SIMPLE FACT FOR FOLLOWING SENTENCES.		
3.	WRITE PREDICATES a) ONE CONVERT's CENTIGRADE TEMPERATURES TO FAHRENHEIT b) ANOTHER CHECKS IF A TEMPERATURE IS BELOW FREEZING.		
4.	WAP IN PROLOG FOR MEDICAL DIAGNOSIS AND SHOW THE ADVANTAGE AND DISADVANTAGE OF GREEN AND RED CUTS.		
5.	WAP TO IMPLEMENT FACTORIAL, FIBONACCI OF A GIVEN NUMBER.		
6.	WRITE A PROLOG PROGRAM FOR DISEASE BY INDICATION AND FINDING ITS CURE.		
7.	WRITE A PROLOG PROGRAM FOR MEDICAL DIAGNOSIS.		
8.	WRITE A PROLOG PROGRAM TO CHECK THE GIVEN NUMBER IS EVEN / ODD.		
9.	WRITE A PROLOG PROGRAM TO SOLVE MONKEY BANANA PROBLEM.		
10.	WRITE A PROLOG PROGRAM TO PERFORM THE OPERATIONS: SUM, DIFFERENCE AND MULTIPLICATION.		
11.	WRITE A PROGRAM FOR FINDING THE MAXIMUM THREE NUMBERS FROM A LIST.		
12.	WRITE A PROGRAM TO PRINT THE REVERSE OF AN INTEGER NUMBER.		
13.	PROGRAM TO CHECK WHETHER A NUMBER IS A MEMBER OF GIVEN LIST OR NOT		
14.	WRITE A PROGRAM TO SOLVE TOWER OF HANOI.		
15.	WRITE A PROLOG PROGRAM FOR ADDING DONALD + GERALD =ROBERT		
16.	WRITE A PROGRAM TO IMPLEMENT BREADTH FIRST SEARCH (BFS) USING PYTHON.		
17.	WRITE A PROGRAM TO IMPLEMENT TIC-TAC-TOE GAME USING PYTHON.		
18.	WRITE A PROGRAM TO IMPLEMENT WATER-JUG PROBLEM USING PYTHON.		
19.	WRITE A PROGRAM IN PYTHON FOR ALPHA BETA PRUNING.		

PROGRAM 1

OBJECTIVE: Study of Prolog.

PROLOG : PROGRAMMING IN LOGIC

PROLOG stands for Programming, In Logic — an idea that emerged in the early 1970's to use logic as programming language. The early developers of this idea included Robert Kowaiski at Edinburgh (on the theoretical side), Marrten van Emden at Edinburgh (experimental demonstration) and Alian Colmerauer at Marseilles (implementation). David D.H. Warren's efficient implementation at Edinburgh in the mid -1970's greatly contributed to the popularity of PROLOG. PROLOG is a programming language centred around a small set of basic mechanisms, including pattern matching, tree-based data structuring and automatic backtracking. This Small set constitutes a surprisingly powerful and flexible programming framework. PROLOG is especially well suited for problems that involve objects- in particular, structured objects- and relations between them.

SYMBOLIC LANGUAGE

PROLOG is a programming language for symbolic, non-numeric computation. It is especially well suited for solving problems that involve objects and relations between objects. For example, it is an easy exercise in prolog to express spatial relationship between objects, such as the blue sphere is behind the green one. It is also easy to state a more general rule: if object X is closer to the observer than object Y. and object Y is closer than Z, then X must be closer than Z. PROLOG can reason about the spatial relationships and their consistency with respect to the general rule. Features like this make PROLOG a powerful language for ArtJIcia1 LanguageA1,) and non- numerical programming. There are well-known examples of symbolic computation whose implementation in other standard languages took tens of pages of indigestible code, when the same algorithms were implemented in PROLOG, the result was a crystal-clear program easily fitting on one page.

FACTS, RULES AND QUERIES

Programming in PROIOG is accomplished by creating a database of facts and rules about objects, their properties, and their relationships to other objects. Queries then can be posed about the objects and valid conclusions will be determined and returned by the program Responses to user queries are determined through a form of inference control known as

resolution.

a) FACTS:

Some facts about family relationships could be written as: Sister (sue, bill)

Parent (ann, sam) Male (jo)

Female (riya)

b) RULES:

To represent the general rule for grandfather, we write: grandfather (X 2) parent (X, Y)

parent (Y, Z) male(X)

c) QUERIES:

Given a database of facts and rules such as that above, we may make queries by typing after a query a symbol'?' statements such as:

?-parent(X,sam) Xann

?grandfather(XY) X=jo, Y=sam

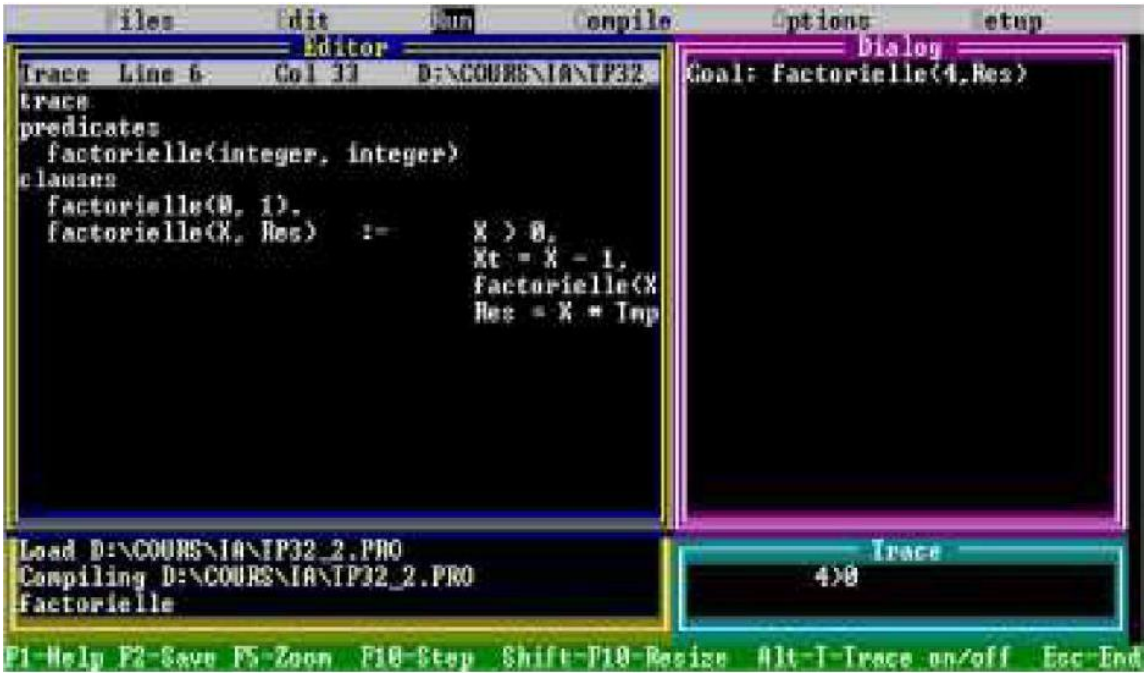
STANDARD PROCEDURES

Designing solution Define the contents in three sections of prolog program as follows.

1. Define Domain Section: Define various variables and symbols needed for problem. (Similar to definition part of conventional programming).
2. Define Predicates:
   Different relation between symbols and variables are to be declared. (Similar to defining function prototype in conventional programming).
3. Define Clauses:
   Various facts and rules supporting the predicates declared are to be defined.The format for storing facts is same as defined in predicates section.

IMPLEMENTING, COMPILING AND EXECUTING THE SOLUTION

1. Start Turbo Prolog
2. Select the Edit mode
3. Type in the program


iles
dit
un
ompile
ptions
etup
Editor
Trace Line 6 Col 33 D:\COURS\IA\TP32_
trace
predicates
factorielle(integer, integer)
clauses
factorielle(0, 1).
factorielle(X, Res) :-
X > 0,
Xt = X - 1,
factorielle(X
Res = X * Tmp
Dialog
Goal: factorielle(4,Res)
Load D:\COURS\IA\TP32_2.PRO
Compiling D:\COURS\IA\TP32_2.PRO
factorielle
Trace
4)0
F1-Help F2-Save F5-Zoom F10-Step Shift-F10-Resize Alt-T-Trace on/off Esc-End

PROGRAM 2

OBJECTIVE: Write simple fact for following:

a) Ram likes mango.
b) Seema is a girl.
c) Bill likes Cindy.
d) Rose is red.
e) John owns gold.

Program:

Clauses

likes(ram ,mango). girl(seema). red(rose). likes(bill ,cindy). owns(john ,gold).

Output:

Goal queries

?-likes(ram,What). What= mango

?-likes(Who,cindy). Who= cindy

?-red(What). What= rose

?-owns(Who,What). Who= john
What= gold.

OUTCOME: Student will understand how to write simple facts using prolog.

META PROGRAMMING

A meta-program is a program that takes other programs as data. Interpreters and compilers are examples of mela-programs. Meta-interpreter is a particular kind of meta-program: an interpreter for a language written in that language. So, a PROLOG interpreter is an interpreter for PROLOG, itself written in PROLOG. Due to its symbol-manipulation capabilities, PROLOG is a powerful language for meta-programming. Therefore, it is often used as an implementation language for other languages. PROLOG is particularly suitable as a language for rapid prototyping where we are interested in implementing new ideas quickly. New ideas are rapidly implemented and experimented with.

OUTCOME: Students will get the basic idea of how to program in prolog and its working environment.

PROGRAM 3

OBJECTIVE: Write predicates

a) one converts centigrade temperatures to Fahrenheit

b) another checks if a temperature is below freezing.

Program

```
% ---------------------------------------------
% Temperature Conversion and Freezing Check
% ---------------------------------------------
```

Production rules

a) Predicate to convert Centigrade (C) to Fahrenheit (F)
Formula: F = (C * 9 / 5) + 32

```
centigrade_to_fahrenheit(C, F) :-
F is (C * 9 / 5) + 32.
```

b) Predicate to check if a temperature is below freezing
Freezing point is 0°C

```
below_freezing(C) :-
   C < 0.
```

Example Queries:

```
?- centigrade_to_fahrenheit(100, F).
F = 212.0.

?- below_freezing(-5).
true.
?- below_freezing(10).
false.
```

Output:

Queries:

?- c_to_f(100,X). X = 212

Yes

?- freezing(15)

.Yes

?- freezing(45). No

OUTCOME: Student will understand how to write a program using the rules.

PROGRAM 4

OBJECTIVE: WAP in prolog for medical diagnosis and show the advantage and disadvantage of green and red cuts.

Program:

```
Domains: disease, indication=symbol
name-string Predicates:
hypothesis
(name,dise
ase)
symptom(
name,indi
cation)
response(c
har)
go goonce
clauses go:- goonce
write("will you like to try again (y/n)?"),
response(Reply), Reply='n'. go. goonce:-
write("what is the patient's name"),nl, readln(Patient),
hypothesis(Patient,Disease),!, write(Patient,"probably has",Disease),!, goonce:-
write("sorry, i am not ina position to diagnose"), write("the
disease"). symptom(Patient,fever):- write("does",Patient,"has a fever
(y/n)?"),nl, response(Reply), Reply='y',nl. symptom(Patient,rash):-
write ("does", Patient,"has a rash (y/n)?" ),nl,
response(Reply), Reply='y',
symptom(Patient_body,ache):-
write("does",Patient,"has a body ache
(y/n)?"),nl, response(Reply). Reply='y',nl.
symptom(Patient,runny_nose):-
write("does",Patient,"has a runny_nose
(y/n)?"), response(Reply),
```

Reply='y' hypothesis(Patient,flu):- symptom(Patient,fever),
symptom(Patient,body_ache),

hypothesis(Patient,common_cold):-

symptom(Patient,body_ache), Symptom(Patient,runny_nose).
response(Reply):-

readchar(Reply), write(Reply).

Output:
makewindow(1,7,7"Expert Medical Diagnosis",2,2,23,70), go.

OUTCOME: Student will understand how to create a expert system using prolog.

PROGRAM 5 A

OBJECTIVE: WAP to implement factorial, fibonacci of a given number.

Program:

Factorial:

```
factorial(0,1). factorial(N,F) :-
N>0,
N1 is N-1,
factorial(N1,F1), F is N * F1.
```

Output:

Goal:

?- factorial(4,X). X=24

PROGRAM 5 B

Objective: PROGRAM TO FINDING THE FACTORIAL OF A GIVEN VALUE */

```
domains
X=integer
predicates
fact(X,X)
clauses
fact(0,1):-!.
fact(No,Fac):-
N=No-1,
Fact(N,F),
Fac=No*F.
```

Result

Goal: Fact(5,x)

X=120

1 Solution

Goal: Fact(4,x)

X=24

1 Solution

PROGRAM 5 C

Objective: PROGRAM TO FINDING THE FIBONACCI OF A GIVEN NUMBER

Fibonacci:

```
fib(0, 0).
fib(X, Y) :- X > 0, fib(X, Y, _). fib(1, 1, 0).
fib(X, Y1, Y2) :- X > 1,
X1 is X - 1, fib(X1, Y2, Y3), Y1 is Y2 + Y3.
```

Output:

Goal:

?-fib(10,X). X=55

OUTCOME: Student will understand the implementation of Fibonacci and factorial series using prolog.

Program 6

Objective: Program for diseases by indication and finding its cure

```
domains
diseases,indication,medicine=symbol
predicates
symptom(diseases,indication)
cure(diseases,medicine)
clauses
symptom(high_fever,body_pain).
symptom(chiken_pox,chills).
symptom(cold,body_pain).
symptom(chiken_pox,body_pain).
symptom(cold,runny_nose).
symptom(flu, runny_nose).
symptom(stomac_problem,high_pain).
cure(high_fever,paracetemol).
cure(chicken_pox,betnesol).
cure(cold,coldrine).
cure(flum,combiflom).
cure(flu,crocin).
cure(stomac_problem,meftal).

Goal: symptom(X, Cold)
No Solution

Goal: symptom(X, runny_nose)
X=Cold
X=Flu

2 Solution
```

OUTCOME: Student will understand how to create a expert system using prolog.

Program 7

Objective: PROGRAM FOR MEDICAL DIAGNOSIS.

```
Domains
X,Y,Z = symbol
Predicates
Disease
Check(X,Y,Z)
Clauses
Check(no,no,no):-write("Physically fit, mentally retarded."),nl.
Check(yes,no,no):-write("Seasonal fever, you will be fit in 4 days."),nl.
Check(no,yes,no):-write("fit/Cold"),nl.
Check(no,no,yes):-write("The Disprin + Rest"),nl.
Check(yes,yes,no):-write("Weakness"),nl.
Disease:-
Makewindow (1,7,7,"Medical Diagnosis System",0,0,25,80),
write (" Fever:"),
readln(X),
write("Shivering:"),
readln(Y),
write("Pain:"),
readln(Z),
Check(X,Y,Z),
readln(Z).
```

Output:

```
Check(no,no,no)
Physically fit, mentally retarded.
Disease
Fever : Yes
Shivering : No
Pain : No
Seasonal fever, you will be fit in 4 days.
```

OUTCOME: Student will understand how to create an expert system using prolog.

PROGRAM 8

Objective: PROGRAM TO CHECK EVEN OR ODD NUMBER

```
Domains
X=integer
Y=integer
Predicates
Checks(Y)
Clauses
Check(Y):-
X=Y mod 2,
X=0,
write (“Even number.”), nl;
write (“ODD number.”), nl.
```

Result

Goal: Check (4)

Even Number

1 Solution

Goal: Check (3)

Odd Number

1 Solution

OUTCOME: Student will understand the implementation of Even Odd using prolog.

PROGRAM 9

Objective: PROGRAM TO SOLVE MONKEY BANANA PROBLEM

Imagine a room containing a monkey, chair and some bananas. That have been hanged from the Centre of ceiling. If the monkey is clever enough, he can reach the bananas by placing the chair directly below the bananas and climb on the chair. The problem is to prove the monkey can reach the bananas. The monkey wants it, but cannot jump high enough from the floor. At the window of the room there is a box that the monkey can use. The monkey can perform the following actions: -

1) Walk on the floor.
2) Climb the box.
3) Push the box around (if it is beside the box).
4) Grasp the banana if it is standing on the box directly under the banana.

Production Rules
can_reach clever,close.
get_on: can_climb.
under in room,in_room, in_room,can_climb.
Close get_on,under| tall
Clauses:

```
in_room(bananas).
in_room(chair).
in_room(monkey).
clever(monkey).
can_climb(monkey, chair). tall(chair).
can_move(monkey, chair, bananas).
can_reach(X, Y):-
clever(X),close(X, Y).
get_on(X,Y):- can_climb(X,Y). under(Y,Z):-
in_room(X),in_room(Y),in_room(Z),can_climb(X,Y,Z). close(X,Z):-get_on(X,Y),
under(Y,Z);
tall(Y). Output:
```

Queries:

```
?- can_reach(A, B). A = monkey.
B = banana.
?- can_reach(monkey, banana).Yes.
```

OUTCOME: Student will understand how to solve monkey banana problem using rules in prolog.

PROGRAM 10

Objective: Program to perform operations: sum, difference & multiplication

```
Domains
X,Y,Z,A,B,ANS=integer
S=symbol
Predicates
        p(integer)
        check(symbol)
        sum(integer,integer,integer)
        diff(integer,integer,integer)
        mul(integer,integer,integer)
operation().
clauses
        operation():-
                clearwindow,
        write("Enter The Choice:-"),nl,
        write("1.SUM"),nl,
        write("2.DIFFERENCE"),nl,
        write("3.MULTIPLY"),nl,
        readint(X),nl,
        p(X),
        write("Want to continue....Y/N"),nl,
        readln(S),
        check(S),
        operation.
```

```
p(1):-
        clearwindow,
        write("Operation=SUM"),nl,
        write("Enter first number:"),
        readint(A),nl,
        write("Enter second number:"),
        readint(B),nl,
        sum(A,B,ANS),
        write("The Answer Is:",ANS),nl.
p(2):-
        clearwindow,
        write("Operation=DIFFERENCE"),nl,
        write("Enter the first no.:"),
        readint(A),nl,
        write("Enter the second no.:"),
        readint(B),nl,
        diff(A,B,ANS),
        write("The Answer Is:",ANS),nl.
p(3):-
        clearwindow,
        write("Operation=MULTIPLY"),nl,
        write("Entre the First No.:"),
        readint(A),nl,
        write("Enter the Second No.:"),
        readint(B),nl,
        mul(A,B,ANS),
        write("The Answer Is:",ANS),nl.
p(_):-
        write("Please Select From 1,2,3 Only...").
```

```
sum(A,B,ANS):-ANS=A+B.
diff(A,B,ANS):-ANS=A-B.
mul(A,B,ANS):-ANS=A*B.
check("n"):-
write("Thanks For Using Program"),nl,
!,fail.
check("Y").
```

Goal: operation

1. Sum

2. Difference

3. Mul

1

Enter the First value 52

Enter the Second Value 10

Sum=62

OUTCOME: Student will understand how to solve calculations using rules in prolog.

PROGRAM 11
Objective: Program to Finding the Maximum of Three Numbers*/

```
domains
    X=integer
    Y=integer
    Z=integer

predicates
    maximum(X,Y,Z)
clauses
    maximum(X,Y,Z):-
        X>Y, X>Z,
        write(X,"is maximum."),nl;

        Y>X, Y>Z,
        write(Y,"is maximum."),nl;

        Z>Y, Z>X,
        write(Z,"is maximum."),nl;

        X=Y, X>Z,
        write("Two are maximum."),nl,
        write("value=",X),nl;

        Z=Y, Z>X,
        write("Two are maximum,"),nl,
        write("value=",Y),nl;
        X=Z, X=Y,
        write("Two are maximum."),nl;
        write("Value=",Z),nl; write("All are maximum.")
```

Result:
Goal (2, 3, 4)
4 is maximum.

PROGRAM 12
Objective: Program to print the reverse of an integer

```
domains
    Z, X, Y = integer
predicates
    reverse(integer)
clauses
    reverse(0):-!.
    reverse(X):-
    Y=X mod 10,
    Z=X/10,
    write(Y),
    reverse(Z).
```

Result:

```
reverse(1234)
4321 Yes

reverse(234)
432 Yes
```

PROGRAM 13

Objective: PROGRAM TO CHECK WHETHER A NUMBER IS A MEMBER OF GIVEN LIST OR NOT

```
domains
    list=integer*

predicates
    findnum(integer,list)

clauses
findnum(X,[]):-
    write("\nNumber Is Not Found").
findnum(X,[X|Tail]):-
    write("\nNumber Is Found").
findnum(X,[Y|Tail]):-
    findnum(X,Tail).
```

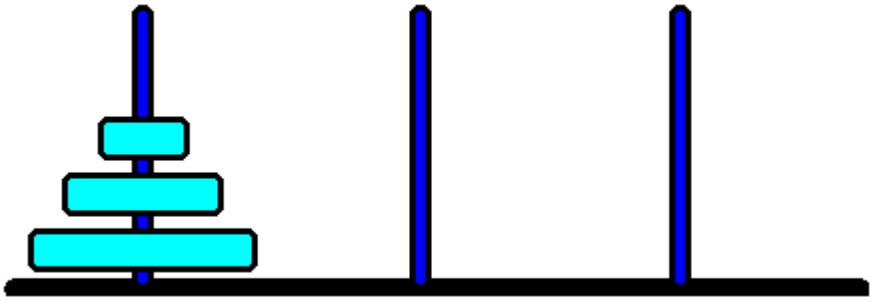
OUT PUT

```
Goal: findnum(3,[1,2,3,4,5])
Number Is Found Yes ----------------------------
Goal: findnum(6,[1,2,3,4,5])
Number Is Not Found Yes ----------------------------
Goal: findnum(2,[1,2,2,1])
Number Is Found Yes
```

PROGRAM 14

Objective: Write a program to solve Tower of Hanoi problem.

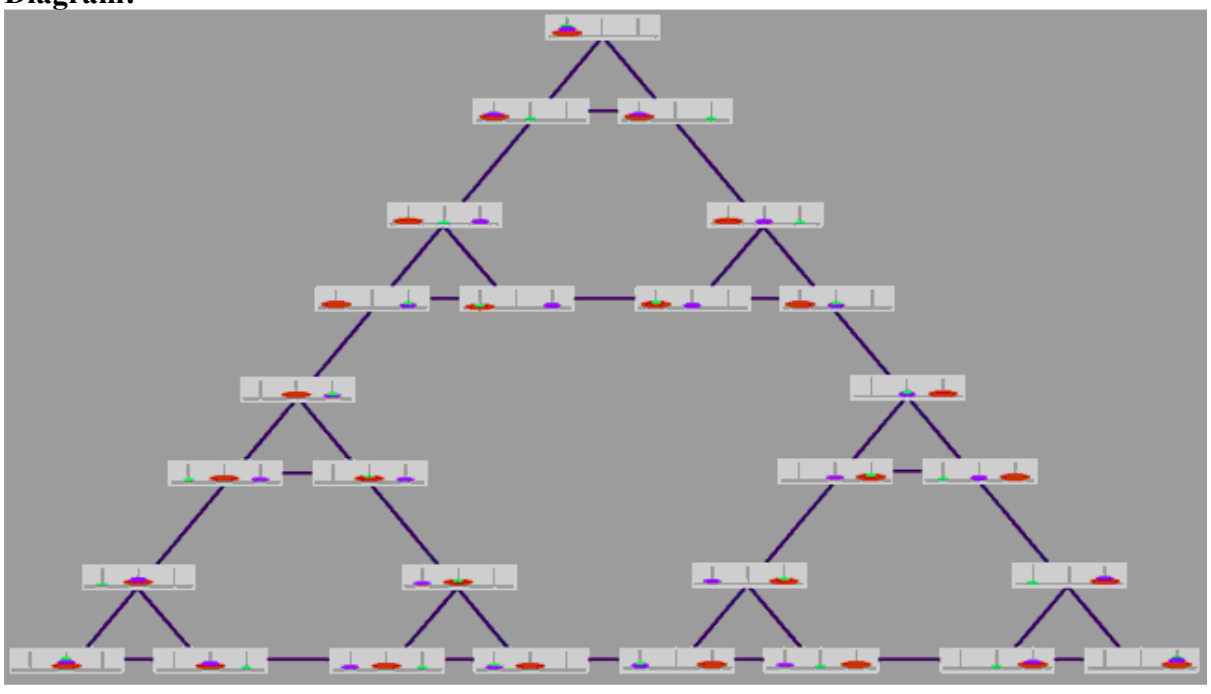
This object of this famous puzzle is to move N disks from the left peg to the right peg using the center peg as an auxiliary holding peg. At no time can a larger disk be placed upon a smaller disk. The following diagram depicts the starting setup for N=3 disks.



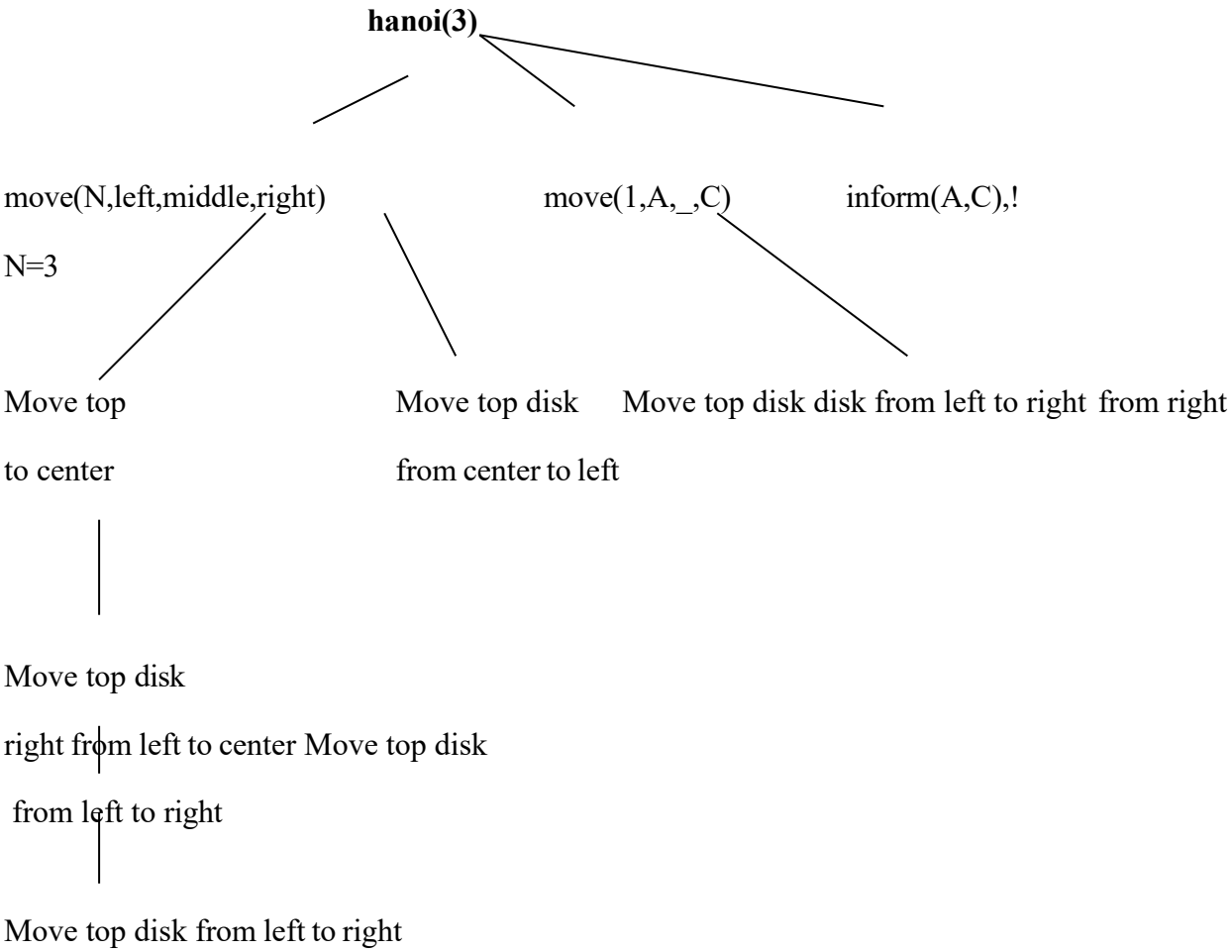
Production Rules

hanoi(N)→move(N,left,middle,right).
move(1,A,_,C)→inform(A,C),fail.
move(N,A,B,C)→N1=N-1,move(N1,A,C,B),inform(A,C),move(N1,B,A,C).

Diagram: -



Parse Tree:-


hanoi(3)
move(N,left,middle,right)
N=3
move(1,A,_,C)
inform(A,C),!
Move top
to center
Move top disk
from center to left
Move top disk disk from left to right
from right
Move top disk
right from left to center
Move top disk
from left to right
Move top disk from left to right

Solution:-

```
domains
    loc =right;middle;left
    predicates hanoi(integer)
move(integer,loc,loc,loc) inform(loc,loc)
clauses
    hanoi(N):-
    move(N,left,middle,right). move(1,A,_,C):-
    inform(A,C),!.
    move(N,A,B,C):-
    N1=N-1,
    move(N1,A,C,B),
    inform(A,C),
    move(N1,B,A,C).
    inform(Loc1, Loc2):-
    write("\nMove a disk from ", Loc1, " to ", Loc2).
```

Goal

hanoi(3).

Move(3, left, right, center).

Move top disk from left to right

Move top disk from left to center

Move top disk from right to center

Move top disk from left to right

Move top disk from center to left

Move top disk from center to right

Move top disk from left to right Yes

PROGRAM 15

Objective: Write a prolog program for Adding **DONALD + GERALD =ROBERT**.

Predicates

```
nondeterm digit(integer). nondeterm
good(integer,integer,integer,integer,integer,integer,integer,integer,integer,integer
). nondeterm
sum(integer,integer,integer,integer,integer,integer,integer,integer,integer,integer.
```

Clauses

```
digit(1).
digit(2).
digit(3).
digit(4).
digit(5).
digit(6).
digit(7).
digit(8).
digit(9).
digit(0).

good(D,O,N,A,L,G,E,R,B,T) :-
digit(D),D<>0,
digit(O),O<>D,
digit(N),N<>O,N<>D,
digit(A),A<>N,A<>D,A<>O,
digit(L),L<>D,L<>N,L<>O,L<>A,
digit(G),G<>D,G<>N,G<>O,G<>A, G<>L,
digit(R),R<>O,R<>N,R<>D,R<>L,R<>A,R<>G,
digit(E),E<>O,E<>N,E<>D,E<>L,E<>A,E<>G, E<>R,
digit(B),B<>O,B<>N,B<>D,B<>L,B<>A,B<>G, B<>R, B<>E,
digit(T),T<>O,T<>N,T<>D,T<>L,T<>A,T<>G, T<>R, T<>E, T<>B,

sum(D,O,N,A,L,G,E,R,B,T).

sum(D,O,N,A,L,G,E,R,B,T) :- N1 = 100000*D+10000*O+1000*N+100*A+10*L+D,
N2 = 100000*G+10000*E+1000*R+100*A+10*L+D,

N3=100000*R+10000*O+1000*B+100*E+10*R+T, N3 = N1+N2.
```

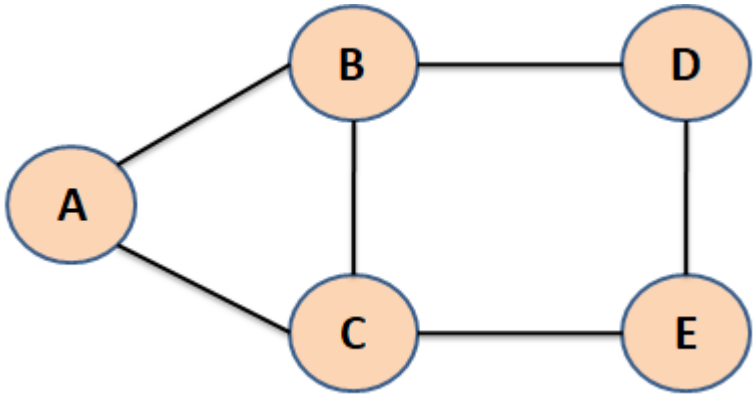
Goal good
(D,O,N,A,L,G,E,R,B,T).

```
D=4, O=3, N=8, A=0, L=9
G=1, E=6, R=5, B=7, T=2
D=4, O=5, N=8, A=1, L=0
G=3, E=6, R=7, B=9, T=2
D=5, O=3, N=8, A=1, L=0
G=2, E=6, R=7, B=9, T=4
3 SOLUTIONS
```

PROGRAM 16

Write a Program to implement Breadth First Search (BFS) using Python.

Input Graph



SOURCE CODE :

```
# Input Graph
graph = {
'A' : ['B','C'],
'B' : ['A','C','D'],
'C' : ['A','B','E'],
'D' : ['B','E'],
'E' : ['C','D']
}
# To store visited nodes.
visitedNodes = []
# To store nodes in queue
queueNodes = []
# function
def bfs(visitedNodes, graph, snode):
    visitedNodes.append(snode)
    queueNodes.append(snode)
    print()
    print("RESULT :")
    while queueNodes:
            s = queueNodes.pop(0)
            print (s, end = " ")
            for neighbour in graph[s]:
                    if neighbour not in visitedNodes:
                            visitedNodes.append(neighbour)
                            queueNodes.append(neighbour)

# Main Code
snode = input("Enter Starting Node(A, B, C, D, or E) :").upper()
# calling bfs function
bfs(visitedNodes, graph, snode)
```

OUTPUT :

Sample Output 1:

Enter Starting Node(A, B, C, D, or E) :A

RESULT :

A B C D E

Sample Output 2:

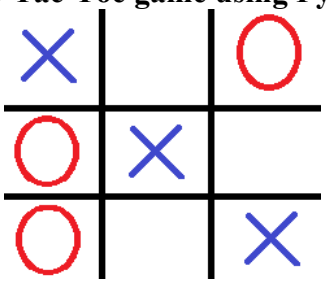
Enter Starting Node(A, B, C, D, or E) :B

RESULT :

B A C D E

PROGRAM 17

Write a Program to implement Tic-Tac-Toe game using Python.



SOURCE CODE :

```
 # Tuple to store winning positions.
win_positions = (
    (0, 1, 2), (3, 4, 5), (6, 7, 8),
    (0, 3, 6), (1, 4, 7), (2, 5, 8),
    (0, 4, 8), (2, 4, 6)
)

def game(player):
    # diplay current mesh
    print("\n", " | ".join(mesh[:3]))
    print("---+---+---")
    print("", " | ".join(mesh[3:6]))
    print("---+---+---")
    print("", " | ".join(mesh[6:]))

    # Loop until player valid input cell number.
    while True:
        try:
            ch = int(input(f"Enter player {player}'s choice : "))
            if str(ch) not in mesh:
                raise ValueError
            mesh[ch-1] = player
            break
        except ValueError:
            print("Invalid position number.")

    # Return winning positions if player wins, else None.
    for wp in win_positions:
        if all(mesh[pos] == player for pos in wp):
            return wp
    return None

player1 = "X"
player2 = "O"
player = player1
mesh = list("123456789")

for i in range(9):
    won = game(player)
```

```
    if won:
            print("\n", " | ".join(mesh[:3]))
            print("---+---+---")
            print("", " | ".join(mesh[3:6]))
            print("---+---+---")
            print("", " | ".join(mesh[6:]))
            print(f"*** Player {player} won! ***")
            break
    player = player1 if player == player2 else player2
else:
   # 9 moves without a win is a draw.
   print("Game ends in a draw.")
```

OUTPUT :

Sample Output:

```
1 | 2 | 3
---+---+---
4 | 5 | 6
---+---+---
7 | 8 | 9
Enter player X's choice : 5

1 | 2 | 3
---+---+---
4 | X | 6
---+---+---
7 | 8 | 9
Enter player O's choice : 3

1 | 2 | O
---+---+---
4 | X | 6
---+---+---
7 | 8 | 9
Enter player X's choice : 1

X | 2 | O
---+---+---
4 | X | 6
---+---+---
7 | 8 | 9
Enter player O's choice : 6

X | 2 | O
---+---+---
4 | X | O
---+---+---
7 | 8 | 9
Enter player X's choice : 9
```

```
X | 2 | O
---+---+---
4 | X | O
---+---+---
7 | 8 | X
*** Player X won! ***
```

PROGRAM 18

Write a Program to Implement Water-Jug problem using Python.

SOURCE CODE :

```
# jug1 and jug2 contain the value
jug1, jug2, goal = 4, 3, 2

# Initialize a 2D list for visited states
# The list will have dimensions (jug1+1) x (jug2+1) to cover all possible states
visited = [[False for _ in range(jug2 + 1)] for _ in range(jug1 + 1)]

def waterJug(vol1, vol2):
    # Check if we reached the goal state
    if (vol1 == goal and vol2 == 0) or (vol2 == goal and vol1 == 0):
            print(vol1,"\t", vol2)
            print("Solution Found")
            return True

  # If this state has been visited, return False
    if visited[vol1][vol2]:
            return False
    # Mark this state as visited
    visited[vol1][vol2] = True
    # Print the current state
    print(vol1,"\t", vol2)
    # Try all possible moves:
    return (
      waterJug(0, vol2) or  # Empty jug1
      waterJug(vol1, 0) or  # Empty jug2
      waterJug(jug1, vol2) or  # Fill jug1
      waterJug(vol1, jug2) or  # Fill jug2
      waterJug(vol1 + min(vol2, (jug1 - vol1)), vol2 - min(vol2, (jug1 - vol1))) or  # Pour water
from jug2 to jug1
      waterJug(vol1 - min(vol1, (jug2 - vol2)), vol2 + min(vol1, (jug2 - vol2)))  # Pour water from
jug1 to jug2
  )

print("Steps: ")
print("Jug1 \t Jug2 ")
print("----- \t ------")
waterJug(0, 0)
```

OUTPUT :

```
Steps:
Jug1        Jug2
-----       ------
0   0
4   0
4   3
0   3
```

3 0
3 3
4 2
0 2
Solution Found

PROGRAM 19

WAP in Python for alpha beta pruning.

SOURCE CODE :

```
 """
          Alpha Beta Pruning :
          -------------------
   depth (int): Current depth in the game tree.
   node_index (int): Index of the current node in the values array.
   maximizing_player (bool): True if the current player is maximizing, False otherwise.
   values (list): List of leaf node values.
   alpha (float): Best value for the maximizing player.
   beta (float): Best value for the minimizing player.

   Returns:
   int: The optimal value for the current player.
   """
import math

def alpha_beta_pruning(depth, node_index, maximizing_player, values, alpha, beta):
   # Base case: leaf node
   if depth == 0 or node_index >= len(values):
      return values[node_index]

   if maximizing_player:
      max_eval = -math.inf
      for i in range(2):  # Each node has two children
         eval = alpha_beta_pruning(depth - 1, node_index * 2 + i, False, values, alpha, beta)
         max_eval = max(max_eval, eval)
         alpha = max(alpha, eval)
         if beta <= alpha:
            break  # Beta cutoff
      return max_eval
   else:
      min_eval = math.inf
      for i in range(2):  # Each node has two children
         eval = alpha_beta_pruning(depth - 1, node_index * 2 + i, True, values, alpha, beta)
         min_eval = min(min_eval, eval)
         beta = min(beta, eval)
         if beta <= alpha:
            break  # Alpha cutoff
      return min_eval

# Example usage
if __name__ == "__main__":
   # Leaf node values for a complete binary tree
   values = [3, 5, 6, 9, 1, 2, 0, -1]
   depth = 3  # Height of the tree
   optimal_value = alpha_beta_pruning(depth, 0, True, values, -math.inf, math.inf)
   print(f"The optimal value is: {optimal_value}")
```

OUTPUT :

The optimal value is: 5