INDEX

POAI Observation

NAME: Aaditya STD.: CSE SEC.: A ROLL NO.: 003 SUB.: ~~Internet Programming~~

S. No.	Date	Title	Page No. marks	Teacher's Sign / Remarks
1	31/07/2024	Basic programs in python	9	
2.	07/08/2024	Project Description	10	
3	04/09/2024	N-Queens Problem	10	
4.	4/9/2024	DFS Searching	10	
5	11/9/2024	A*. Searching	10	
6.	18/9/2024	AO* Searching	10	
7.	18/9/2024	Min-Max ~~Algorithm~~		
8.	25/9/2024	Water-Jug Problem -DFS	10	
9.	9/10/2024	Decision-tree	10	
		Implementation of Clustering techniques K-means	10	
10	16/10/24	Artificial neural network	10	
11	23/10/24	Minimax	10	
12	30/10/24	Introduction to prolog	10	
13	6/11/24	Prolog-family tree	10	

completed

29/7/2024 Basic
Ex.no:- 01

1. Python program to print maximum element in array

Program:-

```
n= int (input("Enter the size of array"))
a=[ ]
for i in range(0, n):
        a.append (input())
c = max(a)
print (c)
```

Output

Enter the size of array: 5

34
54
69
55
23

69

2. Python program to print number evens and odds in array

Program:-

```
n= int (input ("Enter the size of array: "))
a = [ ]
even = 0
odd = 0
for i in range (0, n):
        a. append (int (input()))
for i in a:
    if i%2==0:
        even+ = 1
    if i%2!= 0:
        odd+ = 1
```

```
print(even)
print(odd)
```

Output:-

```
Enter the size of arrays: 5
23
24
56
77
69
2
3
```

3. Python program to print even and odd numbers seperately in a from an array

```
n= int(input("Enter the size of array:"))
a=[]
even=[]
odd=[]
for i in range(0,n):
    a.append(int(input()))
for i in a:
    if i%2==0:
        even.append(i)
    if i%2!=0:
        odd.append(i)
print(even)
print(odd)
```

Output:-

```
Enter the size of arrays: 5
23
24
56
77
69
```

[24, 56]

[23, 77, 69]

4. Python program to find fibonacci series.

Program:

```
def fibonacci(n):
    fib_series = [0, 1]
for i in range(2, n):
    next_fib = fib_series[-1] + fib_series[-2]
return fib_series

terms = int(input("Enter the number of terms: "))
if terms <= 0:
    print("Please enter a positive integer.")
else:
    fib_series = fibonacci(terms)
    print("Fibonacci series:")
    for num in fib_series:
        print(num, end=" ")
```

Output:

Enter the number of terms: 5

Fibonacci series:

0 1 1 2 3

Python program to print area of circle.

```
a = int(input("Enter the radius of circle:"))
area = 3.14 * a * a.
print("The area of circle is:", area)
```

Output

Enter the radius of circle: 5

The area of the circle is: 78.5

Python program to check the type of triangle.

Program

```
print("Enter the sides of triangle")
a = int(input())
b = int(input())
c = int(input())
if a==b and a==c and b==c:
    print("Equilateral triangle").
elif a==b or b==c or a==c:
    print("Isosceles triangle")
elif a!=b and a!=c and b!=c:
    print("Scalene triangle")
```

Output:

Enter the sides of the triangle

4
4
4

Equilateral triangle

7. Python program to print the duplicate element in array

Output

```
print ("Enter the size of array")
n= int (input())
a=[]
for i in range(0,n):
    a.append(input())
duplicate = set([x for x in a if a.count(x)>1])
print (duplicate)
```

Output:

Enter the size of array: 5

2 3
4 7
4 a
4)
5 3

{'4 7'}

8. Python program to remove duplicates in array

```
print ("Enter size of array")
n= int (input())
a=[]
for i in range(0,n):
    a.append(input())
duplicate = set ([x for x in a if a.count(x)>1])
for item in list (duplicate):
    while item in a:
        a.remove (item)
print (a)
```

Output:

Enter size of array 5

26
27
27
28
29

['26', '28', '29']

9. Python program to print factorial of a number.

```
def fact(n):
    if n==0:
        return 1
    else:
        return n*fact(n-1)
n = int(input("Enter the number:"))
print(fact(n))
```

Output

Enter the number: 3
6

10. Python program to print sum of first n digits.

Program

```
n = int(input("Enter a num:"))
sum = 0
for i in range(1, n+1):
    sum = sum+i
print(sum)
```

31/07/2024

Output

Enter a num: 5
15

07/08/24

DOMAIN: FINANCIAL CRIME

Credit card fraud is a type of financial ~~crim~~ crime that involves the unauthorised use of someone else's credit card information to make transactions or purchases. The domain of credit card fraud encompasses several activities, including:

* Card Not present fraud: This occurs when credit card information is stolen and used for online or telephone transactions where physical card is not required.

* Card Present Fraud: This involves using a stolen or counterfeit physical credit card in person at a point of sale terminal

* Account Takeover: Fraudsters gain access to person's credit card account, often through phishing or data breaches, and use it to make unauthorized transactions

* Identity Theft: This broader category includes stealing personal information to open new credit card accounts or access existing ones

* Skimming: Criminals use a device to capture the magnetic stripe data from a credit card during legitimate transactions, which can then be used to create counterfeit cards

* Phising and Social Engineering:- Fraudsters trick individuals into revealing their credit card information through deceptive emails, calls or message

Problem Statement

Problem Statement: Enhancing accuracy and efficiency in credit card fraud Detection systems

As digital transactions proliferate, credit card fraud has became increasingly sophisticated, leading to substantial financial losses and compromised of consumer trust. Traditional fraud detection methods often struggle to keep pace with evolving fraud tactics, resulting in either high false positive rates or missed fraudulent transactions. There is a critical need to develop advanced detection system that can accurately identify and mitigate fraudulent activities in real-time while minimizing disruption to legitimate transactions

Algorithm:-

Machine Learning-Based Credit Card Fraud Detection

1. Data Collection:

- Input Data: Collect transaction details such as transaction amount, merchant, location, time and cardholder history

2. Pre processing:

Feature Engineering: Extract relevant features from raw data

3. Model Selection:

Algorithm Choices: Use machine learning algorithms such as Logistic Regression, Decision Trees, Random Forests, Gradient Boosting Machines or Neural Networks

4. Anomaly Detection:

Real-Time Scoring: Apply the trained model to incoming transactions to compute a fraud risk score.

5. Evaluation

Evaluate model performance using metrics like Precision, Recall, F1-Score and ROC-AUC.

6. Integration Monitoring:

* Deployment: Integrate the fraud detection model into payment processing systems.

* Real-Time Monitoring: Continuously monitor model performance and update it with new data to adapt to emerging fraud patterns

Objectives

* Improve Detection Accuracy: Enhance the precision and recall of fraud detection systems to accurately identify fraudulent transactions while minimizing false positives.

* Minimize Financial losses: Reduce financial losses incurred by cardholders and financial institutions due to fraudulent transaction.

* Adapt to Evolving Threats: Develop adaptive models that can quickly respond to new and evolving fraud tactics.

* Ensure Compliance: Meet industry regulations and standards for fraud detection and data protection.

Beneficiaries.

Consumers:

Enhanced Security, Consumers benefit from reduced risk of financial loss identity theft due to more effective fraud detection.

Financial Institutions:

Reduced Losses, Banks and credit card companies experience lower financial losses from fraud and chargebacks

4/9/24

Ex:-3

n-Queens Problem

Aim:- To write a python code to solve n queens problem

Python Code:

```
def is_safe(board, row, col, n):
    for i in range(row):
        if board[i][col] == 1:
            return False

    for i,j in zip(range(row,-1,-1), range(col,-1,-1)):
        if board[i][j]==1:
            return False

    for i,j in zip(range(row,-1,-1), range(col,n)):
        if board[i][j] == 1:
            return False

    return True

def solve_nqueens_util(board, row, n):
    if row>=n:
        return True

    for col in range(n):
        if is_safe(board, row, col,n):
            board[row][col]=1

            if solve_nqueens_util(board, row+1, n):
                return True

            board[row][col]=0

    return False
```

```
def solve_nqueens(n):
    board = [[0 for _ in range(n)] for _ in range(n)]
    if not solve_nqueens_util(board, 0, n):
        print("Solution does not exist")
        return None
    return board
def print_board(board):
    for row in board:
        print(" ".join("Q" if col else "." for col in row))
n = int(input("Enter the value of N: "))
Solution = solve_nqueens(n)
if Solution:
    print_board(Solution)
```

Output:-

```
Enter the value of N: 4
. Q . .
. . . Q
Q . . .
. . Q .
```

Result:

Therefore the python code for solving nQueens problems is executed sucessfully

4/9/24

Ex.no: 4

Depth first Search

Aim: To implement depth first search using python

Code:

```
def dfs_recursive(graph, node, visited=None):
    if visited is None:
        visited = set()
    visited.add(node)
    print(node, end=" ")
    for neighbour in graph[node]
        if neighbour not in visited:
            dfs_recursive(graph, neighbour, visited)
    return visited
def get_graph_from_user():
    graph = {}
    num_nodes = int(input("Enter the number of
    nodes in graph"))
for _ in range(num_nodes):
    node = input("Enter node name:")
    neighbors = input("Enter neighbours of {node}
    (space-seperated):").split()
    graph[node] = neighbors
    return graph
graph = get_graph_from_user()
start_node = input("Enter the starting
                    node for DFS:")
dfs_recursive(graph, start_node)
```

Output :-

```
Enter number nodes in graph : 4
Enter node name : A
Enter neighbors of A (space-separated) : B C
Enter node name : B
Enter starting node for DFS : A
A B D C
```

Result :-

Thus the above code is executed sucessfully.

11/9/24
Ex.no:5

A* Searching

Aim:- To write a python program to execute a A* searching Algorithm

Code:

```
import heapq
def a_star(graph, start, goal):
    open_set = []
    heapq.heappush(open_set, (0+ heuristic(start,goal),0, start))
    came_from ={}
    g_score = {node: float('inf') for node in graph}
    g_score[start]=0

    f_score ={node: float('inf') for node in graph}
    f_score[start]=heuristic(start, goal)
    while open_set:
        _, current_g, current= heapq.heappop(open_set)
        if current == goal:
            return reconstruct_path(came_from, current)
        for neighbor, cost in graph[current]:
            tentative_g_score=g_score[current]+cost
            if tentative_g_score<g_score[neighbor]:
                came_from[neighbor]= current
                g_score[neighbor]=tentative_g_score
                f_score[neighbor]=g_score[neighbor]+
                                  heuristic(neighbor,goal)
        if not any(item[1] == neighbor for item in open_
```

```
                heapq.heappush(open_set, (f_score[neighbor], g_score[neighbor], neighbor))
    return None

heuristic(node, goal):
    return 0

reconstruct_path(came_from, current):
    path = [current]
    while current in came_from:
        current = came_from[current]
        path.append(current)
    path.reverse()
    return path
```

Output:-

path: ['A', 'B', 'C', 'D']

Result:-

Thus, the python program for A* searching is executed sucessfully

Exno:6
18.9.24

AO* Searching

Aim: To write a python code for AO* Searching

code:

```
Class Graph:
    def __init__(self):
        self.graph = {}
        self.heuristics = {}
    def add_node(self, node, children, cost):
        self.graph[node] = (children, cost)
    def set_heuristic(self, node):
        return self.heuristics.get(node, 0)
    def AO_star(graph, start):
        open_list = {start}
        closed_list = set()
        solution_graph = {}
        def minimum_cost_solution(node):
            children, cost = graph.graph.get(node, ([], 0))
        if not children:
            return graph.get_heuristic(node), {}
        min_cost = float('inf')
        best_solution = None
        for child_set in children:
            total_cost = cost
            sub_solution = {}
            for child in child_set:
                child_cost, child_solution = minimum_cost_
                                             solution(child)
        return min_cost, best_solution
```

```
while open_list:
    current_node = next(iter(open_list))
    open_list.remove(current_node)
    update_solution(current_node)
return solution_graph.

def main():
    graph = Graph()
    graph.add_node('A', [['B','C'],['D']], 1)
    graph.add_node('B', [['E'],['F']], 1)
    graph.add_node('C', [['G']], 1)
    graph.add_node('D', [['H']], 1)
    graph.add_node('E', [], 0)
    graph.add_node('F', [], 0)
    graph.add_node('G', [], 0)
    graph.add_node('H', [], 0)

    graph.set_heuristic('A', 2)
    graph.set_heuristic('B', 2)
    graph.set_heuristic('C', 2)
    graph.set_heuristic('D', 2)
    graph.set_heuristic('E', 0)
    graph.set_heuristic('F', 0)
    graph.set_heuristic('G', 0)
    graph.set_heuristic('H', 0)
```

```
solution = AO_star(graph, 'A')
print("Solution graph:")
for node, sub_solution in solution.items():
    print(f"{node}: {sub_solution}")
if __name__ == "__main__":
    main()
```

Output :

Solution Graph:

A: {'D': {'H': {}}}

D: {'H': {}}

H: {}

Result :-

Thus the python code for AO* Searching is executed sucessfully

9/10/2024 (Team 1).

Credit Card fraud detection:

Paper:

- Abstract
- Introduction
  - Statistics.
- Literature survey.
  - min 15-20. papers.
- System architecture.
- Algorithm
  - formula
  - mathematical calculations.
- Results.
  - images & tables.
- Conclusion
- References.

9/10/24.

Ex.no:07

Water jug using Dfs

Aim:- To write & execute code for water jug using DFS in python.

Code:

```
def is_solvable_dfs(jug1, jug2, target):
    def dfs(state, visited):
        if state[0]==target or state[1]==target:
            return True.

        visited.add(state)
        x,y = state

        possible_moves = [
                (jug1, y),
                (x, jug2),
                (0, y),
                (x, 0),
                (max(0, x-(jug2-y)), min(jug2, y+x)),
                (min(jug1, x+y), max(0, y-(jug1-x)))]

        for new_state in possible_moves:
            if new_state not in visited:
                if dfs(new_state, visited):
                    return True
        return false
```

```
initial_state = (0, 0)
visited = set()

return dfs(initial_state, visited)

jug1_capacity = 4
jug2_capacity = 3
target_amount = 2

if is_solvable_dfs(jug1_capacity, jug2_capacity, target_amount):
    print("It is possible to measure the target amount.")
else:
    print("It is not possible to measure the target amount.")
```

Output

```
Enter capacity of jug1 : 4
Enter capacity of jug2 : 3
Enter target amount to measure : 2
It is possible to measure the target amount.
```

Result :-

Thus the water-jug problem using DFS is in [illegible] is executed successfully.

Ex:-08

Implementation of Decision tree (classification techniques.

Aim:

To implement a decision tree classification technique for gender classification using Python.

Program:-

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
data = {'Height':[152,155,172,185,167,180,187,180,164,174],
'weight':[45,57,72,85,68,78,22,90,66,88],
'Gender':['Female','Female','Male','Male','Female','Male','Female',
'Male','female','Male']
}
df = pd.DataFrame(data)
x = df[['Height','weight']]
Y = df['Gender']
classifier = DecisionTreeClassifier()
classifier.fit(x,Y)

height = float(input("Enter height (in cm) for prediction:"))
weight = float(input("Enter weight (in kg) for prediction:"))
random_values = pd.DataFrame([[height,weight]],
predicted_gender = classifier.predict(random_values)
print(f"predicted gender for height {height} cm and
weight {weight} kg: {predicted_gender}")
```

Output :-

Enter height (in cm) for prediction : 169

Enter weight (in kg) for prediction : 61

Predicted gender for height 169.0 cm and weight 61.0 kg : female.

Result :

Thus the decision tree classification has been executed successfully.

Ex.no:09

Implementation of clustering techniques
k-means

Aim:-

To implemant a k-Means clustering techniques using Python language.

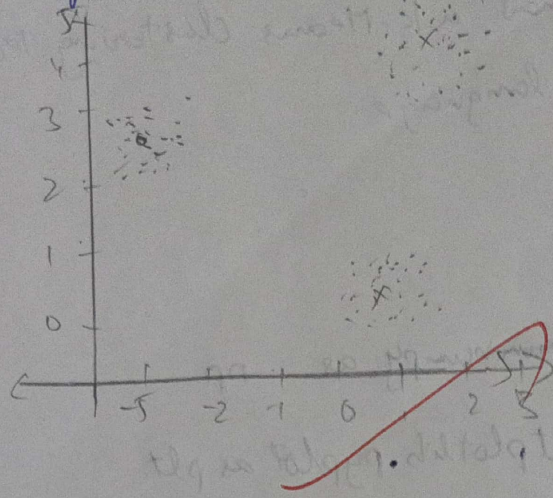
Eg

Code:-

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=300, centers=3,
    cluster_std=0.60, random_state=0)
K=3
Kmeans = kmeans(n_clusters=K, random_state=0)
y_kmeans = kmeans.fit_predict(X)
plt.figure(fig.size=(8,6))
plt.scatter(X[:,0], X[:,1], c=y_kmeans,
s=39; cmap='viridis',
    label='clusters'
centers = kmeans.cluster_centers
plt.scatter(centers[:,0], centers[:,1] c='red', s=200,
    alpha=0.75, marker='x', label='centroids')
plt.title('kmeans clustering results')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
```

plt.legend()
plt.show()

Output.



Result :-

Thus the K-means clustering technique has been implemented successfully

Exno:- 10

Implementation of Artificial Neural Network for application using python regression.

Aim:-

To implementing artificial neural network for an application in regression using python.

Program:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import standard scaler
from keras.models import sequential
from keras.layers import Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt
np.random.seed(42)
X=np.random.rand(1000, 3)
y= 3*X[0]+ 2*[:,1] ** .2+1.5* np.sin(X[:,2]*np.pi)
    + np.random.normal(0, 0.1, 1000)
X_train, x_test, Y_train, Y_test = train_test_split(
        X, Y, test_score = 0.2, random_state)
scales = standard scales()
X_train = scales.fit_transform(x_train)
X_test = scales.transform(x_test)
model = sequential()
modeladd( pense (10, input_dim=X_train.shape[1],
                activation 'relu'))
mod.add( Dense (10, activation='linear'))
mod.compiler(optimizer=Adam(learning_rate=0.01),
                loss = 'mean_squared_error')
```

```
history=model.fit(x_train, y_train, epochs=100, batch_size=32,
validation_split=0.2, verpose=1)
y_pred= model.predict(x_test)
mse=np.mean(y_test-y_pred.flatten()**2)
print(f'mean Squared Error : {mse:.4f}')
plt.figure(figsize=(12,6))
plt.plot(history.history['loss'],label='Training loss')
plt.plot(history.history['val_loss'], label = validation
Loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

output.

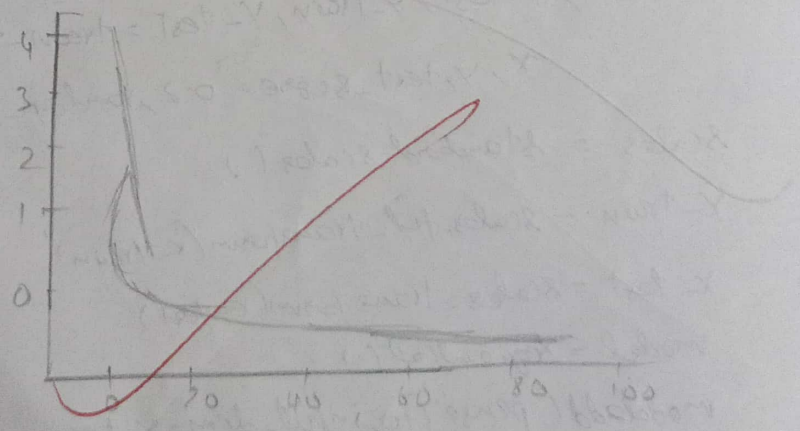
Epoch 1/100

20/20

⋮

Epoch 100/100

20/20



Result:-

Thus the artificial neural network for regression has been implemented successfully

Exno: 4

MiniMax Algorithm

Aim:-

To implement MiniMax Algorithm

Program:-

```
import math
def minimax(depth, node_index, is_maximizer, scores,
            height):
    if depth == height:
        return scores[node_index]
    if is_maximizer:
        return max(minimax(depth+1, node_index*2, False,
                   scores, height), minimax(depth+1, node_
                   index*2+1, False, scores, height))
    else:
        return min(minimax(depth-1, node_index*2, True, scores, height),
                   minimax(depth-1, node_index*2+1, True, scores, height))
def calculate_tree_height(num_leaves):
    return math.ceil(math.log2(num_leaves))
scores = [3, 5, 6, 9, 1, 2, 0, -1]
tree_height = calculate_tree_height(len(scores))
optimal_score = minimax(0, 0, True, scores, tree_height)
print(f"The optimal score is {optimal_score}")
```

Output:

The optimal score is 5

Result:-

Thus the decision tree program has been executed successfully

Aim:-

To learn of PROLOG terminologies and write basic programs.

Terminologies:-

1) Atomic terms:

They are usually strings made up of lower and upper case letters, digits and the underscore, starting with a lowercase letter.

eg: dog · ab2c-321

2) Variables:

They are strings of letters, digits and the underscore, starting with a capital letter or an underscore.

Eg: Dog, Apple_420

3) Compound terms:

compound terms are made up of a PROLOG atom and a number of arguments enclosed in parenthesis and separated by commas.

eg: is_bigger(elephant, x) . f(g(x,-),7)

4). fact

A fact is a predicate followed by a dot

eg: bigger_animal(whale). life_a_beautiful.

5) Rules

A rule consists of a head and body

Source code:

KB1.

woman(mia)

woman(yolanda)

Plays Air Guitar(jody)

party.

Query1 = ?- woman(mia).

Query 2 = ? - plays Air guitar(mia)

Query 3 = ? - party.

Query 4 = ? - concert.

Output:

? - woman(mia)

true

? - plays air guitar(mia)

false

? - concert

Error = unknown procedure. convert

KB2

happy(yolanda).

listens2music(mia).

playerair guitar(mia) = listens2 music(mia)

playsairguitar(yolanda): listens 2 music(yolanda)

Listens 2 music(yolanda): - happy(yolanda)

output-

? plays air guitar(mia)

true

? - plays air guitar(yolanda)

true.

Lab 3:

likes (dan, sally).

likes (sally, dan).

likes (john, britney).

married (X, Y) :- likes (X, Y), likes (Y, X).

friends (X, Y) :- likes (X, Y), likes (Y, X).

Output

?- likes (dan, X).

X = sally.

?- married (dan, sally).

true.

? married (john, britney).

false.

Lab 4.

food (burger).

food (sandwich).

food (pizza).

lunch (sandwich).

dinner (pizza).

meal (X) :- food (X).

?- food (pizza).

true.

?- meal (X), lunch (X).

X = sandwich.

Lab 5

owns (jack, car (bmw)).

owns (john, car (chevy)).

own (olivia, car (civic)).

~~owns (olivia, car (civic))~~ owns (jane, car (chevy)).

sedan (car (bmw)).

sedan (car (civic)).

truck (car (chevy)).

output :-

? owns (john, X).

X = car (chevy).

?- owns (john, _).

true.

Result: Thus the basic prolog programs has executed successfully

Ex.No:13

Prolog - Family Tree

Aim:-

To develop a family tree program using PROLOG with all possible facts, rules and queries.

Source code:

```
knowledge base:
/* facts: */
male(peter)
male(john)
male(chris)
male(kevin)
female(betty).
female(jeny).
female(lisa)
female(helen)
parentof(chris, betty)
parentof(chris, peter)
parentof(helen, peter)
parentof(helen, betty)
parentof(kevin, chris)
parentof(kevin, lisa)
parentof(jeny, john)
parentof(jeny, helen)
/* RULE == */
/* son, parent
/* son, grandparent */

father(X,Y) :- male(Y), parentof(X,Y)
mother(X,Y) = female(Y), parentof(X,Y)
grandfather(X,Y) :- male(Y), parentof(X,Z), parentof(Z,Y)
grandmother(X,Y) = male(Y), parentof(X,Z), parentof(Z,Y)
brother(X,Y) == male(Y), father(X,Y), parentof(X,Z) parentof(Z,Y)
```

brother(X,Y):- male(Y), father(X,Z), father(Y,W), Z=W.

sister(X,Y):- female(Y), father(X,Z), father(Y,W), Z=W.

Output

male(peter)

true.

father(chris,peter)

true.

father(chris,betty)

false.

mother(chris,X)

X=betty

brother(Chris,helen)

false

Result:-

Thus PROLOG for family tree program has been executed successfully