Pratice Program

1. To check whether the given number is armstrong or not

```
n = 153
s = n
b = len(str(n))
sum1 = 0
while n != 0:
    r = n % 10
    sum1 = sum1 + (r ** b)
    n = n // 10
if s == sum1:
    print("The given number", s, "is armstrong number")
else:
    print("The given number", s, "is not armstrong number")
```

Output

The given number 153 is armstrong number

2. Largest number in the array

```
def largest (arr,n):
    max = arr[0]
    for i in range (1,n):
        if arr[i] > max:
            max = arr[i]
    return max

arr = [10, 24, 35, 90, 100]
n = len(arr)
ans = largest (arr, n)
print ("Largest in given array", ans)
```

Output

Largest in given array 100

3. To check given string is palindrome and symmetry

```
def palindrome(a):
    mid = (len(a)-1)//2
    start = 0
    last = len(a)-1
    flag = 0
```

```
while (start <= mid):
    if (a[start] == a[last]):
        start += 1
        last -= 1
    else:
        flag = 1
        break
if flag == 0:
    print ("string is palindrome")
else:
    print ("string is not palindrome")

string = ammaana
palindrome (string)
```

Output

The entered string is palindrome

4. To reverse the given string

```
string = "Wonderful"
s = string.split()[::-1]
l = []
for i in s:
    l.append(i)
```

print (" ". join(l))

Output

lufredno w

5. def is_arm(n):

num_str = str(n)

num_d = len(num_str)

sum = sum(int(digi) * num_d for digi in num_str)

return sum

num = 371

if is_arm(num):

print("Yes")

print ("No")

26/07/2024

Output

Yes

PROJECT :- Image-based music recommendation System

DOMAIN :- MULTIMEDIA CONTENT RECOMMENDATION

Problem statement

Develop a music recommendation system that suggest songs based on user uploaded images. The system will analyze the images to detect objects, scences and emotions and then match these with suitable tracks. Using advanced computer vision techniques, the system aims to deliver personalized

TARGET AUDIENCE

Primary users: Users of streamling platforms

Secondary users: Users of social media who want personalized music recommendation based on images they upload

OBJECTIVES

- Analyse user Images
- Detect mood and themes
- Generate recommendation
- Good user interface

- Real time performance

ALGORITHMS

- Image Feature Extraction
- Object and Scene Detection
- Emotion and Mood analysis
- Recommendation mapping
- Enhanced recommendations.

N Queens problem

```
def print_board(board):
    for row in board:
        print(" ".join(['Q' if cell else '.' for cell in row]))
    print()

def is_safe(board, row, col):
    for i in range(col):          # (left side)
        if board[row][i]:
            return false

    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j]:           # (upper diagonal left)
            return false

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

    return True

def solve_queens_util(board, col):
    if col >= len(board):
        print_board(board)
        return

    for i in range(len(board)):
        if is_safe(board, i, col):
            board[i][col] = True
            solve_queens_util(board, col+1)
            board[i][col] = false
```

```
def solve_queens():
    N = int(input("Enter the board size (N): "))
    board = [[False] * N for _ in range(N)]
    print("solutions")
    solve_queens_util(board, 0)
```

DFS

```
def add_edge(adj, s, t):
    adj[s].append(t)
    adj[t].append(s)

def dfs_rec(adj, visited, s):
    visited[s] = true
    print(s, end = " ")
    for i in adj[s]:
        if not visited[i]:
            dfs_rec(adj, visited, i)

def dfs(adj, s):
    visited = [false] * len(adj)
    dfs_rec(adj, visited, s)

if __name__ == '__main__':
    v = 5
    adj = [[] for _ in range(v)]
    edges = [[1,2],[1,0],[2,0],[2,3],[2,4]]
    for e in edges
        add_edge(adj, e[0], e[1])
```

```
source = 1
print("DFS from source", source)
dfs(adj, source)
```

Output

```
DFS from source: 1
1 2 0 3 4
```

Output for n queens

Solutions for 4 queens problem

```
. . Q .
Q . . .
. . . Q
. Q . .
```

Solutions for 8 queens problem

```
Q . . . . . . .
. . . . . . Q .
. . . . Q . . .
. . . . . . . Q
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .
```

RESULT

Thus, the n Queen Problem and DFS problem has been executed successfully

Water Jug using DFS

```
def solvewaterJug(c_jug1, c_jug2, des_q):
    stack=[]
    stack.append(((0,0),[]))
    visited = set((0,0))
    while stack:
        cur_state, path = stack.pop()
        if cur_state[0]== des_q or cur_state[1] == des_q:
            return cur_state, path + [cur_state]

        next_states = generatenextStates(cur_states, c_jug1, c_jug2)
        for next_states in nextstates:
            if next_state not in visited:
                stack.append((next_state, path +[cur_state]))
                visited.add(next_state)

        return "No solution found"

def generateNextStates(state, c_jug1, c_jug2):
    next_states[]
    next_states.append((c_jug1, state[1]))
    next_states.append((state[0], c_jug2))
    next_states.append((0, state[1]))
    next_states.append((state[0], 0))
    pour_amt = min(state[0], c_jug2 - state[1])
    next_states.append((state[0]- pour_amt, state[1] + pour_amt))
    pour_amt = min(state[1], c_jug1 - state[0])
    next_states.append((state[0] + pour_amt, state[1] - pour_amt))

    return next_states
```

```
def main():
    c_jug1 = int(input("Enter the capacity of Jug1: "))
    c_jug2 = int(input("Enter the capacity of Jug2: "))
    des_q = int(input("Enter the desired quanity: "))
    result, path = solveWaterJug(c_jug1, c_jug2, des_q)
    if result != "No solution found":
        print(f"The desired quanity {des_q} can be achieved:")
        for i, state in enumerate(path):
            print(f"Step {i+1}: {state}")
    else:
        print("No solution found")

if __name__ == "__main__":
```

Output

```
Enter the capacity of Jug1: 5
Enter the capacity of Jug2: 3
Enter the desired quantity: 4
The desired quanity 4 is achieved
(0,0)
(0,3)
(3,0)
(3,3)
(5,1)
(0,1)
(1,0)
(1,3)
(4,0)
```

Result

Thus the water jug problem using DFS executed successfully.

A* Algorithm

```
import math
import heapq
class cell:
    def __init__(self):
        self.parent_i = 0
        self.parent_j = 0
        self.f = float('inf')
        self.g = float('inf')
        self.h = 0

ROW = 9
COL = 10

def is_valid(row, col):
    return (row >= 0) and (row < ROW) and (col >= 0) and (col < COL)

def is_unblocked(grid, row, col):
    return grid[row][col] == 1

def is_destination(row, col, dest):
    return row == dest[0] and col == dest[1]

def cal_h_val(row, col, dest):
    return ((row - dest[0]) ** 2 + (col - dest[1]) ** 2) ** 0.5

def trace_path(cell_d, dest):
    print("The Path is")
    path = []
    row = dest[0]
    col = dest[1]

    while not (cell_d[row][col].parent_i == row and cel[row][col].parent_j ==
        path.append((row, col))
        temp_row = cel_d[row][col].parent_i
        temp_col = cell_d[row][col].parent_j
        row = temp_row
        col = temp_col
```

```
path.append((row,col))
path.reverse()
for i in path:
    print("->", i, end=" ")
print()

def a_star(grid, src, dest):
    if not is_valid(src[0], src[1]) or not (dest[0], dest[1]):
        print("source is blocked")
        return
    if is_destination(grid, src[0], src[1]):
        print("source or destination is blocked")

    if is_destination(src[0], src[1], dest):
        print("reached")
        return

    cl_list = [[false for _ in range(col)] for in range(Row)]
    cell_d = [[cell for _ in rang(oc)] for _ in range(roow)]
    i = src[0]
    j = src[1]
    cell_d[i][j].f = 0
    cell_d[i][j].g
    cell_d[i][j].h
    cell_d[i][j].p_i = i
    cell_d[i][j].p_j = j

    o_list[]
    heapq.heapush(o_list, (0.0, i, j))
    f_d = False
    while len(o_list) > 0:
        p = heapq.heappop(o_list)
        i = p[1]
        j = p[2]
        c_list[i][j] = "true"
```

```
directions = [(0,1), (0,-1), (1,0), (-1,0), (1,1), (1,-1), (-1,1), (-1,-1)]

for dir in directions:
    n_i = i + dir[0]
    n_j = j + dir[1]

    if is_valid(new_i, new_j) and not closed_list[new_i][new_j]:
        if is_destination(n_i, new_j, dest):
            cell_d[new_i][new_j].p_i = i
            cell_d[new_i][new_j].p_j = j
            print("Not found")
            t_path(cell_d, dest)
            f_dest = True
            return
        else:
            g_new = cell_d[i][j].g + 1.0
            h_new = cal_h_values[new_i, new_j, dest]
            f_new = g_new + h_new

        if cell_d[new_i][new_j].f = float('inf') or f > f_new:
            heapq.heappush(o_list, (f_n, new_i, new_j))

if not found_dest:
    print("Failed to reach")

def main():
    grid [
        [1, 0, 1, 1, 1, 1, 0, 1, 1],
        [1, 1, 1, 0, 1, 1, 0, 1, 1],
        [1, 1, 1, 0, 1, 1, 0, 1, 0],
        [1, 0, 1, 1, 1, 1, 0, 1, 0],
        [1, 0, 1, 1, 1, 1, 0, 1, 1],
        [1, 0, 0, 0, 1, 0, 0, 1]]
```

```
src = [8,0]
dest = [0,0]
a_star_search (grid, src, dest)
if __name__ == "__main__":
main()
```

Output

The destination cell is reached

The path is

→ (8,0) → (7,0) → (6,0) → (5,0) → (4,1) → (3,2) → (2,1)

→ (1,0) → (0,0

Result

Thus A* algorithm has been executed successfully

AO* Algorithm

Aim: To Implement AO* algorithm

Program

```
import heapq
def heuristic(a,b):
    return abs(a[0]-b[0])+abs(a[1]-b[1])
def ao_star(grid, start,goals):
    directions=[(0,1),(1,0),(0,-1),(-1,0)]
    open_list=[(0+min(heu(start,goal) for goal in goals),0,
    g_cost, came_from={start:0},{}
    while open_list:
        if any(current==goal for goal in goals):
            path=[]
            while current in came_from:
                path.append(current)
                current= came_from[current]
            return path+[start][::-1]
        for dx, dy in directions:
            neighbor=(current[0]+dx, current[1]+dy)
            if 0<=neighbor[0]< len(grid) and 0<=neighbor[1]
            tentative_g = current_g+1
            if neighbor not in g_cost or tentative_g
                g_cost[neighbor]=tentative_g
                f_cost = tentative_g+min(heuristic(neighbor,
                goal
```

```
        heapq.heappush(open_list, (f_cost, tentative_g, neighbor))
        came_from[neighbor] = current

return None
```

Output

```
expanding : A
Best path for A: ['B', 'C']

grid = [ [0, 0, 0, 0, 0],
         [0, 1, 1, 1, 0],
         [0, 1, 0, 0, 0]
         [0, 1, 0 0, 0]
          [0,0,0,0,0] ]

start = (0 0)
goals = [(4,4), (2,2)]
path = ao_star(grid, start, goals)
print ("Path found:", path)
path found: [(2,2), (2,1), (2,0), (1, 0), (0, 0)]
```

RESULT

Thus the AO* algorithm has been executed successfully.

Implementation of decision Tree classification Techniques.

Aim: To Implement a decision tree classification techniques for gender classification using python

Program

```
from sklearn.tree import DecisionTreeClassifier
import numpy as np

X = np.array([[170, 65, 42],
              [180, 75, 44],
              [160, 50, 38],
              [175, 70, 43],
              [165, 55, 39],
              [185, 80, 45]])

Y = np.array([0, 1, 0, 1, 0, 1])
clf = DecisionTreeClassifier()
clf.fit(X, Y)
new_data = np.array([[168, 52, 38]])
prediction = clf.predict(new_data)
print("Predicted gender", "Male" if
      prediction[0] == 1 else "Female")
```

Output: Predicted gender: Male

Result: Thus the decision tree classification techniques has been executed successfully

Implementing of clusting Techniques

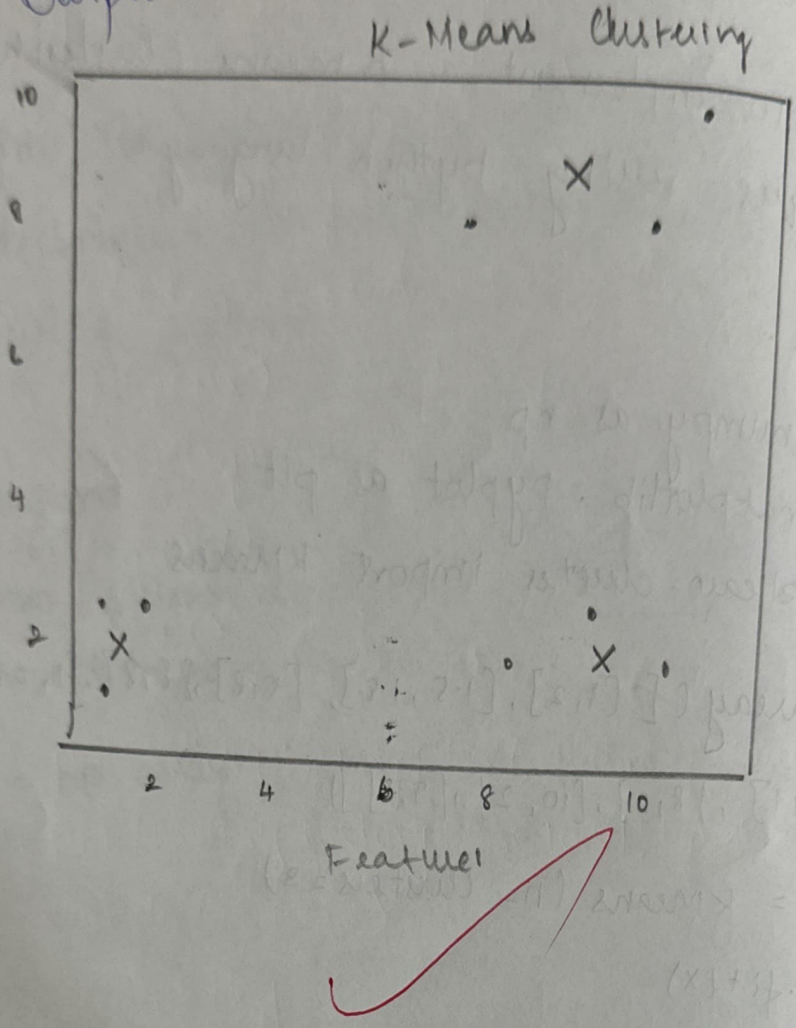
k-means

Aim :- To implement a k means clustering technique using python language

Program.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

X = np.array([[1,2],[1.5,1.8],[5,8],[8,8],[1,0.6],
    [9,11],[8,2],[10,2],[9,3]])

kmeans = KMeans(n_clusters=3)

kmeans.fit(X)
centroids = kmeans.cluster_centers_
labels = kmeans.labels_
plt.scatter(X[:,0], X[:,1], c=labels, cmap='viridis')
plt.scatter(centroids[:,0], centroids[:,1]
plt.title('K-Means clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show
```

Output



Result

Thus, implementing of clusting techniques k-means executed successfully.

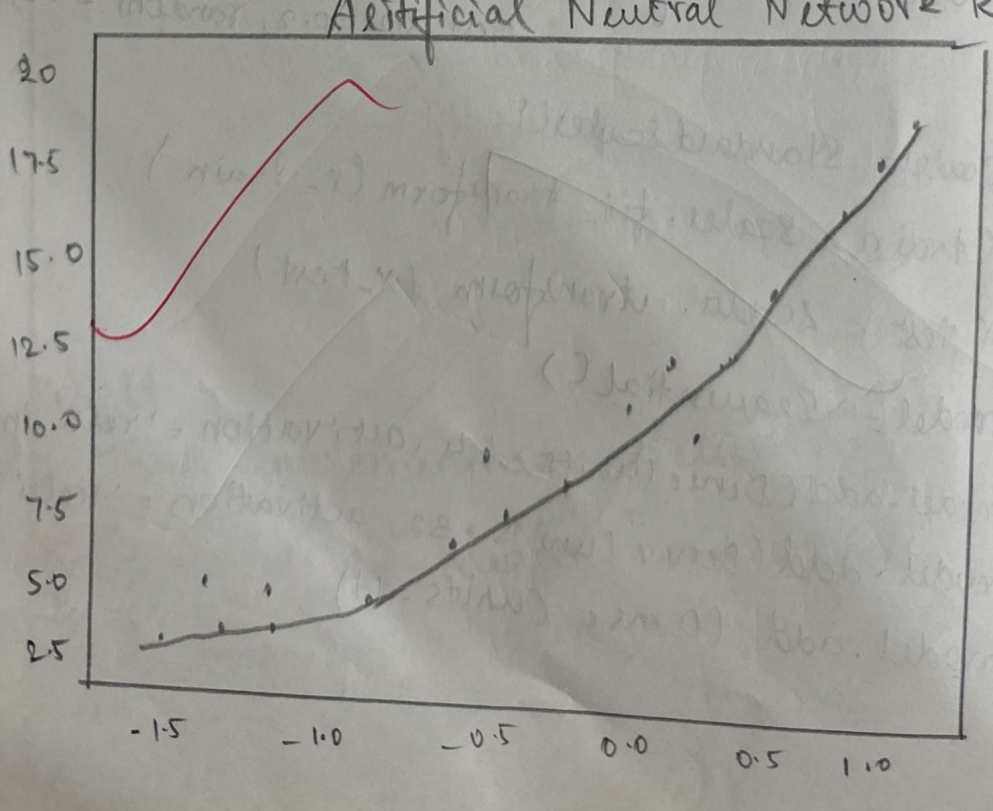
Aim: To Implementing artificial neutral for an application in regression using python

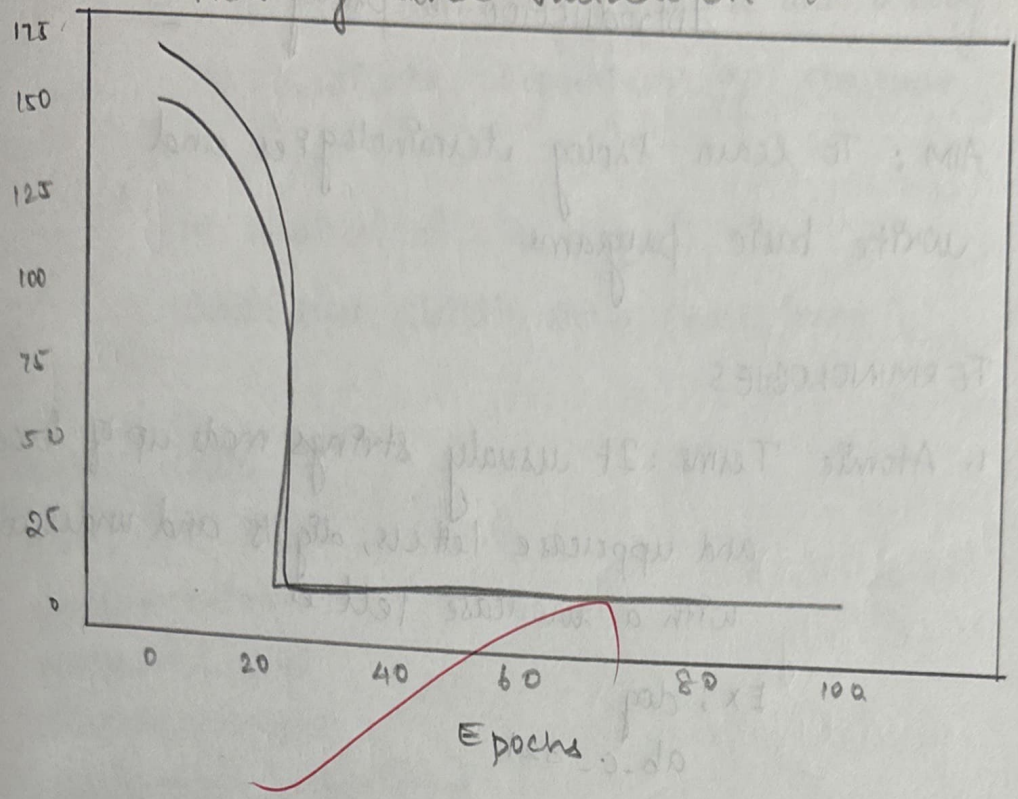
Program

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.preprocessing import StandardScaler

np.random.seed(42)
x = np.linspace(0,10,100)
y = 2*x + 1 + np.random.normal(0,1,100)
x = x.reshape(-1,1)
x_train, x_test, y_train, y_test = train_test_split(x,y,
                                   test_size=0.2, random=42)

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
model = Sequential()
model.add(Dense(units=64, activation='relu', input)
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=1))
```

```
model.compile(optimizer='adam', loss='mean_squared_error')
history = model.fit(x_train, y_train, epochs=100)
y_pred = model.predict(x_test)
plt.scatter(x_test, y_test, color='blue', label='True values')
plt.plot(x_test, y_pred, color='red', linewidth=2)
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabels('Epochs')
plt.legend()
plt.show()
```

Output





Result

Thus artificial neural for an application in regression using python is implemented and executed successfully

Introduction to prolog

AIM: To learn Prolog terminologies and write basic programs

TERMINOLOGIES

1. Atomic Terms: It usualy strings made up of lower and uppercase letters, digits and underscore with a lowercase letter

Ex: dog
ab_c_321

2. Variables: Variables are strings of letters, digits and the underscore starting with a capital letter

Ex: Dog
Apple_420

3. Compound Terms: Compound terms are made up of a PROLOG atom and a number of arguments enclosed in parentheses and separated by commas

Ex: is bigger (elephant, x)
f(g(x,_),7)

4. Facts:

A fact is a predicate followed by a dot

Ex: bigger_animal (whale)
life_is_beautiful

5. Rules - A rule consists of a head and a body of predicate separated by commas

Ex is_smaller(x,y) :- is_bigger(y, x)
aunt(Aunt, Child) :- sister(Aunt, Parent)

Source code

KB1:
woman(mia)
woman(jody)
woman(yolanda)
playsAirGuitar(jody)
party

Output
woman(mia) - true
playsAirGuitar(mia) - false

KB2
happy(yolanda)
listens2music(mia)
listens2music(yolanda) :- happy(yolanda)
playsAirGuitar(mia) :- listens2music(mia)
playsAirGuitar(yolanda) :- listens2music(yolanda)

Output
playsAirGuitar(mia) - true
playsAirGuitar(yolanda) - true

KB3

likes(dan, sally)

likes(sally, dan)

likes(john, brittney)

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

KB4

food(burger)

food(sandwich)

food(pizza)

lunch(sandwich)

dinner(pizza)

meal(X) :- food(X)

Output

food(pizza) - true

meal(X), lunch(X)

X = sandwich.

KB5

owns (jack, car (bmw))

owns (john, car (chevy))

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

→ owns (who, car (Chevy))

who = john

Result

Thus, the programs written using Prolog terminologies are written and executed successfully.

Prolog - Family Tree

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

Source Code

```
male(peter)
male(john)
male(chris)
male(kelvin)

female(betty)
female(jeny)
female(lisa)
female(helen)

parentof(chris, peter)
parentof(chris, betty)
parentof(helen, peter)
parentof(helen, betty)
parentof(kelvin, chris)
parentof(kelvin, lisa)
parentof(kevin, john)
parentof(jeny, helen)
```

RULES

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) :- female(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

→ parentof(kelvin, x)
  x = chris

→ father(x, chris)
  x = kelvin

→ father(x, chris)
  x = kelvin

→ sister(x, chris)
  false

Result

Thus the family prolog tree has been executed successfully.