INDEX

AI - Obs

NAME: R. Deeksha STD.: CSE SEC.: A ROLL NO.: _____ SUB.: ~~Computer Networks~~

S. No.	Date	Title	~~Page No.~~ marks	Teacher's Sign / Remarks
1.	~~30/1~~	Basic python program	9	
2.		Content Recommendation system	9	
3.		Nqueens	9	
4.		Depth-first search	10	
5.		A* algorithm	10	
6		Water Jug problem	10	
7.		Decision tree	10	
8		kmeans	10	
9.		Artificial Neural Network	10	
10.		Minimax	10	
11.		Introduction to prolog	10	
12.		prolog - family tree	10	

completed

PYTHON PROGRAMS:

1. Reverse the array elements

```
def reverse(start, end, arr):
    while start < end:
        arr[start], arr[end] = arr[end], arr[start]
        start += 1
        end -= 1
    return arr
if __name__ == "__main__":
    arr = [1, 2, 3, 4]
    print(reverse(0, 6, arr))
```

Output:

```
[4, 3, 2, 1]
```

2. Monotonic

```
def monotonic(arr, n):
    inc = True
    dec = True
    for i in range(1, n):
        if arr[i] < arr[i-1]:
            inc = False
        if arr[i] > arr[i-1]:
            dec = False
    if inc or dec:
        print("monotonic")
    else:
        print("not monotonic")
if __name__ == "__main__":
    arr = [6, 5, 4, 2]
    n = len(arr)
    monotonic(arr, n)
```

Output:

```
monotonic
```

3. Check whether the given string is substring or not:

```
def issubstring(s1, s2):
    M = len(s1)
    N = len(s2)
    for i in range(N-M+1):
        j = 0
        while j<M and s2[i+j] == s1[j]:
            j += 1
        if j == M:
            return True
    return False

if __name__ == "__main__":
    s1 = "hello"
    s2 = "worldhello"
    print(issubstring(s1, s2))
```

Output:
True

4. Vowels count:

```
def vowelscount(s):
    count = 0
    for i in s:
        if i in 'aeiou':
            count = count + 1
    return count

if __name__ == "__main__":
    s = "geek"
    ans = vowelscount(s)
    print(ans)
```

Output:
2

5. def

5.
```
def largest(arr,n):
    max = arr[0]
    for i in range(1,n):
        if arr[i] > max:
            max = arr[i]
    return max
if __name__ == "__main__":
    arr = [10, 20, 30]
    n = len(arr)
    ans = largest(arr, n)
    print("largest = ", ans)
```

Output:

largest = 30

Content Recommendation System :

Problem statement:

Develop and diploy a highly effective content recommendation technology that generates content recommendations based on uses behaviours and their interaction with the content by the use of enhanced algosithms like collaborative filtering and content based method aimed at increasing the level of uses activity and content consumption and uses satisfation resulting from the constantly adjusting recommendation process.

Abstract:

As the users get connected with the range of technologies in the current society to access the content, there is a challenge of searching for specific information. The issue of recommendation of irrelevant piece of content is solved by a content recommendation system; which employs algosithms for the pickup and for recommending content based on the choice of the users or their usage preference. While there has been great improvement, some of the current systems of recommendation are still concerning encountering a numbes of the challenges like the data spassity, scalability and the issue of real-time recommendations. Howevers issues such as filter-bubbles, privacy considerations and handling of multiple content types introduce more complications when it comes to the design of good recommender systems. Thus, it is possible

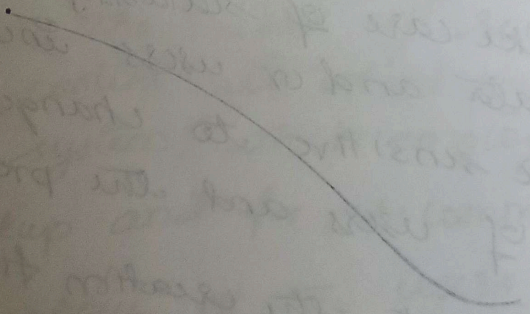
to address that the specific problem that is defined in this work relates to the main problems of the content recommendations systems and focus on the necessity of elaboration of new approaches to enhance its effectiveness, for examples in terms of UX, precision, or engagement. The purpose is to find out ways to minimise the above limitations in order to enhance: relevance, representativeness, and timeliness of the content recommended to the users and

Objective:
The goal of addressing these challenges is to develop and refine recommendation systems that can. The goal of addressing these challenges is to develop and refine recommendation systems that can:

- To provide the best possible content recommendations that are applicable to the specific users
- Take care of scalability challenges while the data and or users increases
- Be sensitive to changes in the behaviours of users and the preferences that they show.
- Prevent the creation filter bubbles and encourage the suggestions of a variety of contents
- protect the user's privacy and his or her data.

Target audience:

1. E-commerce shoppers: Recommend products based on browsing and purchase history to boost sales.
2. Streaming Users: Suggest movies, shows, and music tailored to individual tastes to enhance engagement.
3. News Readers: Personalize news feeds to match user interests and keep them informed.
4. Social Media Users: Curate posts, ads and profiles based on interactions to increase engagement.
5. Educational Consumers: Recommend courses and materials aligned with learning goals and past activities.



1. Depth
code:
def d
if

V
p

grap

dfs (

Output:
A B

Result:
Thus
DFS

1. Depth First Search

Code:

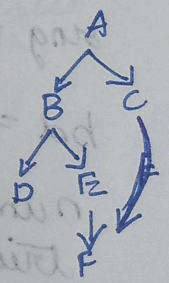
```
def dfs(graph, start, visited = None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end=" ")
    for neighbour in graph[start]:
        if neighbour not in visited:
            dfs(graph, neighbour, visited)
```

```
graph = {
    'A' : ['B','C'],
    'B' : ['D','E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}
```

```
dfs(graph,'A')
```

Output:

A B D E F C



Result:

Thus the python program for implementing DFS search algorithm was executed successfully.

2. N-Queens problem

```
def share_diagonal(x0, y0, x1, y1):
    dx = abs(x0 - x1)
    dy = abs(y0 - y1)
    return dy == dx

def col_clashes(bs, c):
    for i in range(c):
        if share_diagonal(i, bs[i], c, bs[c]):
            return True
    return False

def has_clashes(the_board):
    for col in range(1, len(the_board)):
        if col_clashes(the_board, col):
            return True
    return False

def main():
    import random
    rng = random.Random()
    bd = list(range(8))
    num_found = 0
    tries = 0
    result = []
    while num_found < 30:
        rng.shuffle(bd)
        tries += 1
        if not has_clashes(bd) and bd not in result:
            print("Found solution {0} in {1} tries.".
                  format(bd, tries))
            tries = 0
            num_found += 1
            result.append(list(bd))
```

print (result)

main()

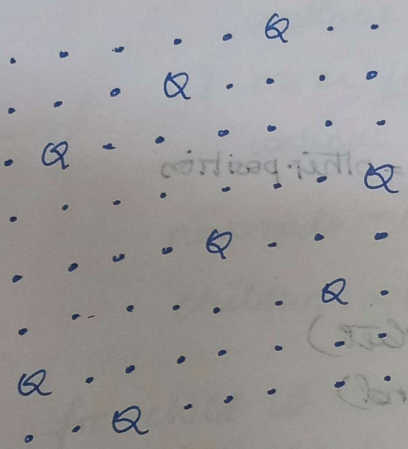
Output:

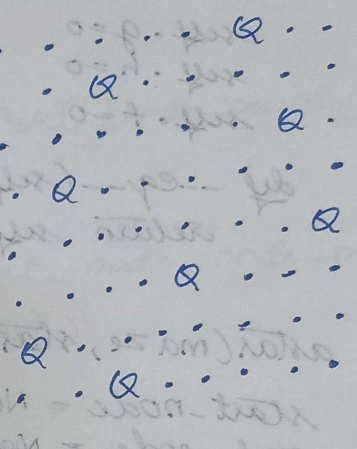
Found solution [5, 3, 1, 7, 4, 6, 0, 2] in 688 tries

Found solution [5, 2, 6, 1, 7, 4, 0, 3] in 421 tries

[[5, 3, 1, 7, 4, 6, 0, 2], [5, 2, 6, 1, 7, 4, 0, 3]]

this is the solution for 8 queens problem.





for 4 queens:

found solution [1, 3, 0, 2] in 7 tries

found solution [2, 0, 3, 1] in 32 tries

Result: Thus the python program for 8-queens and 4 queens problem was executed successfully

3. A* Algorithm

Program code:

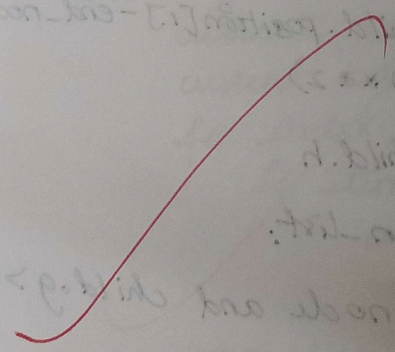
```
import heapq

Class Node:
    def __init__(self, parent=None, position=None):
        self.parent = parent
        self.position = position

        self.g = 0
        self.h = 0
        self.f = 0

    def __eq__(self, other):
        return self.position == other.position


def astar(maze, start, end):
    start_node = Node(None, start)
    end_node = Node(None, end)

    open_list = []
    closed_list = []

    open_list.append(start_node)
    while len(open_list) > 0:
        current_node = open_list[0]
        current_index = 0
        for index, item in enumerate(open_list):
            if item.f < current_node.f:
                current_node = item
                current_index = index

        open_list.pop(current_index)
        closed_list.append(current_node)

        if current_node == end_node:
            path = []
            current = current_node
            while current is not None:
                path.append(current.position)
                current = current.parent
```

```
children = []
for new_position in [(0,-1),(0,1),(-1,0),(1,0),(-1,-1),
                     (-1,1),(1,-1),(1,1)]:
    node_position = (current_node.position[0] +
                     new_position[0], current_node.
                     position[1] + new_position[1])

    if node_position[0] > (len(maze) -1) or
       node_position[0] < 0 or node_position[1] >
       (len(maze[len(maze)-1]) - 1) or
       continue
    if maze[node_position[0]][node_position[1]]!=0:
       continue

    new_node = Node(current_node, node_position)
    children.append(new_node)

for child in children:
    for closed_child in closed_list:
        if child == closed_child:
            continue

    child.g = current_node.g + 1
    child.h = ((child.position[0] - end_node.position[0])
              ** 2) + ((child.position[1] - end_node.
              position[1]) ** 2)
    child.f = child.g + child.h

    for open_node in open_list:
        if child == open_node and child.g > open_node.
                                            g:
            continue
    open_list.append(child)

def main():
    maze = [[0,0,0,0,1,0,0,0,0,0],
            [0,0,0,0,1,0,0,0,0,0],
            [0,0,0,0,1,0,0,0,0,0],
```

```
[0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0]]

start = (0,0)
end = (7,6)
path = astar(maze, start, end)
print(path)
if __name__ == "__main__":
    main()
```

Output:

```
[(0,0), (1,1), (2,2), (3,3), (4,3), (5,4), (6,5), (7,6)]
```



Result:

Thus the python program for A* algorithm is verified and executed successfully.

1 Water Jug program code:

```
def fill_4_gal
    return (
def fill_3_g
    return (
def empty
    return (
def empty_
    retu
def pour_
    transfo
    return
def pour_3
    trans
    retu
def dfs_
```

it

sta

w

1 Water Jug

program code:

```
def fill_4_gallon(x, y, x_max, y_max):
    return (x_max, y)
def fill_3_gallon(x, y, x_max, y_max):
    return (x, y_max)
def empty_4_gallon(x, y, x_max, y_max):
    return (0, y)
def empty_3_gallon(x, y, x_max, y_max):
    return (x, 0)
def pour_4_to_3(x, y, x_max, y_max):
    transfer = min(x, y_max - y)
    return (x - transfer, y + transfer)
def pour_3_to_4(x, y, x_max, y_max):
    transfer = min(y, x_max - x)
    return (x + transfer, y - transfer)
def dfs_water_jug(x_max, y_max, goal_x, visited=None,
                  start=(0,0)):
    if visited is None:
        visited = set()
    stack = [start]
    while stack:
        state = stack.pop()
        x, y = state
        if state in visited:
            continue
        visited.add(state)
        print(f"visiting state: {state}")
        if x == goal_x:
            print(f"Goal reached: {state}")
            return state
        next_states = [fill_4_gallon(x, y, x_max, y_max),
                       fill_3_gallon(x, y, x_max, y_max),
```

[7, 2]

```
        empty_4_gallon (x, y, x_max, y_max),
        empty_3_gallon (x, y, x_max, y_max),
        pour_4_to_3 (x, y, x_max, y_max),
        pour_3_to_4 (x, y, x_max, y_max)]

    for new_state in next_states:
        if new_state not in visited:
            stack.append(new_state)

    return None

x_max = 4
y_max = 3
goal_x = 2

dfs_water_jug(x_max, y_max, goal_x)
```

Output:

```
visiting state : (0, 0)
visiting state: (0, 3)
visiting state: (3, 0)
visiting state : (3, 3)
visiting state: (4, 2)
visiting state : (4, 0)
visiting state: (1, 3)
visiting state: (1, 0)
visiting state: (0, 1)
visiting state: (4, 1)
visiting state: (2, 3)
Goal reached: (2, 3)
(2, 3)
```

Result:
Thus, the python program to solve water jug problem is executed and verified successfully

5. IMPLEMEN
CLASSIFI

AIM

AIM:
To Imple
technique for
python

Explanation:

- Import
- call the
from
- Assign
- call the
on the
for
- Display

Program

```
from
x = [[
    [1
y = [
clf = c
predi
print
```

Output:
Predicte

Result:
Thus
classi

5. IMPLEMENTATION OF DECISION TREE CLASSIFICATION TECHNIQUES

AIM:

To implement a decision tree classification technique for gender classification using python

Explanation:

- Import tree from sklearn
- call the function DecisionTreeClassifier() from tree
- Assign values for x and y
- call the function predict for predicting on the basis of given random values for gender to each given feature
- Display the output.

Program code:

```
from sklearn.tree import DecisionTreeClassifier
X = [[170, 65, 40], [160, 55, 38], [180, 80, 42],
     [175, 75, 41], [158, 52, 37]]

Y = [1, 0, 1, 1, 0]

clf = clf.fit(X, Y)
prediction = clf.predict([[165, 60, 39]])
print("Predicted Gender:", "Male" if prediction[0]
                    == 1 else "Female")
```

Output:

Predicted Gender: Female

Result:

Thus the python program to implement a decision tree classification technique for gender classification is executed successfully.

6. IMPLEMENTATION OF K-MEANS CLUSTERING TECHNIQUE

AIM:

To implement a k-Means clustering technique using python language

Explanation:

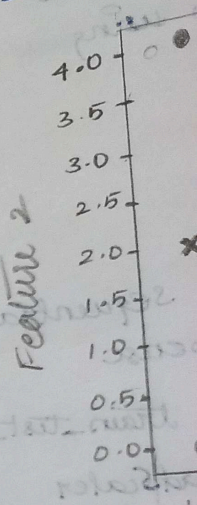
- Import KMeans from sklearn.cluster
- Assign x and y
- Call the function KMeans()
- Perform scatter operation and display the output

Program Code:

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
x = [[1,2], [1,4], [1,0], [4,2], [4,4],
     [4,0]]
kmeans = KMeans(n_clusters = 2)
kmeans.fit(x)
y = kmeans.labels_
centers = kmeans.cluster_centers_
for i, label in enumerate(y):
    plt.scatter(x[i][0], x[i][1], color='blue'
                if label == 0 else 'green')
plt.scatter(centers[:,0], centers[:,1],
            color='red', marker='x', s=100,
            label='centroids')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
```

plt.title('K-
plt.legend
plt.show()

Output:

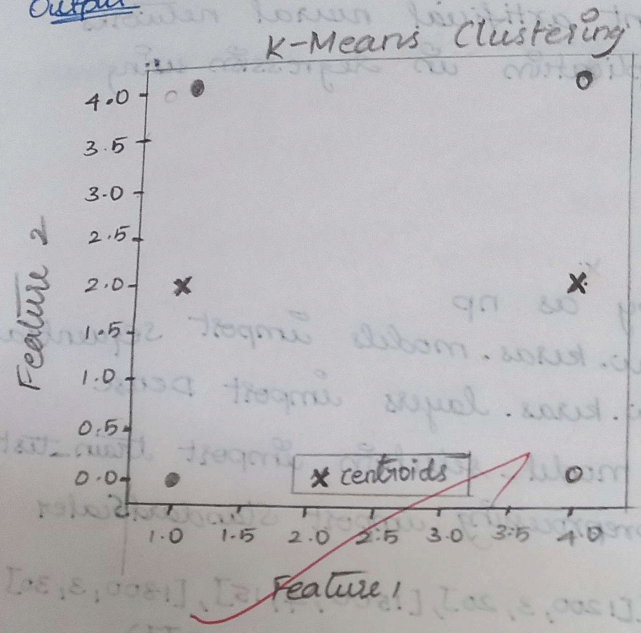


Result:

Thus
K-M
su

```
plt.title('K-Means Clustering')
plt.legend()
plt.show()
```

Output:



Result:

Thus the python program to implement K-Means clustering technique is executed successfully.

7. IMPLEMENTATION OF ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION IN REGRESSION

AIM:

To implement artificial neural networks for an application in regression using Python.

Program code:

```
import numpy as np
from tensorflow.keras.models import sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

x = np.array([[1200, 3, 20], [1500, 4, 15], [1800, 3, 30],
              [2000, 5, 10], [1600, 4, 25]])

Y = np.array([200000, 250000, 270000, 300000, 240000])

X_train, x_test, Y_train, Y_test = train_test_split(x, y,
                                   test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(x_train)
X_test = scaler.transform(x_test)
model = sequential()
model.add(Dense(32, input_dim=x.shape[1],
                activation='relu'))

model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, Y_train, epochs=10, batch_size=5,
          verbose=1)

predictions = model.predict(x_test)
```

for i in
print(

Output:
predicted

Result:
Thus
neu
is

```
for i in range(min(5, len(predictions))):
    print(f"Predicted: {predictions[i][0]:.2f}, Actual: {Y_test[i]}")
```

Output:

Predicted: 0.06, Actual: 250000

Result:

Thus the python program to implement artificial neural networks for an application in regression is executed successfully.

8. Introduction to PROLOG

AIM:

To learn PROLOG terminologies and write basic programs.

TERMINOLOGIES:

1. Atomic Terms:-

Atomic terms are usually strings made up of lower and uppercase letters, digits and the underscore, starting with a lowercase letter.

Eg:

dog
ab_c_321

2. Variables:

Variables are strings of letters, digits and 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 no. of arguments enclosed in parantheses and separated by commas

Eg:

is_bigger(elephant, X)
f(g(X, _), 7)

4. Facts:

A fact is

Eg: Bigger

5. Rules:

A rule co

body

Eg: is_sma

aunt(

Source Code:

KB1:

woman(mia).
woman(jody).
woman(yolan
playsAirGuit
party.

Query 1: ?- w
Query 2: ?-
Query 3: ?-
Query 4: ?-
Query 5.

Output:

?- woman(m
true

?- playsAir
false

?- party
true

?- concert
ERROR:

4. Facts:

A fact is a predicate followed by a dot.

Eg:

Bigger_animal (whale).
life_is_beautiful.

5. Rules:

A rule consists of a head and a body:

Eg:

is_smaller (X, Y) :- is_bigger (Y, X)
aunt (Aunt, Child) :- sister (Aunt, parent),
parent (Parent, child).

Source Code:

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

Query 1: ?- woman(mia).
Query 2: ?- playsAirGuitar(mia).
Query 3: ?- party
Query 4: ?- concert
Query 5.
```

Output:

```
?- woman(mia).
true

?- playsAirGuitar(mia).
false

?- party
true

?- concert
ERROR: Unknown procedure: concert/0 (DWIM could not correct goal)
```

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, brittany).
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, brittney).
false

KB4:

food(burger).
food(sandwich).
food(pizza).
lunch(sandwich).
dinner(pizza)
meal(X):- food(X).

Output:
?- food(pizza)
true
?- meal(X),
X = sandwich
?- dinner(C
false.

KB5:
owns(jack,
owns(john,
owns(olivia
owns(jane
sedan(car
sedan(car
truck(car

Output:
?- owns(Jo
X = car(C
?- owns(
true
?- owns(
who = Jo
?- ow
false
?- owns
X = car

Result:
Thus

Output:

?- food(pizza).

true

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

x = sandwich

?- dinner(sandwich)

false.

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.

?- owns(jane, x), sedan(x).

false

?- owns(jane, x), truck(x).

x = car(chevy).

Result:

Thus the basic programs on PROLOG terminologies were successfully run and executed.

MINIMAX ALGORITHM

AIM:

To implement minimax algorithm:

Program:

```
import math
def minimax(depth, node_index, is_maximizes,
                    scores, height):
    if depth == height:
        return scores[node_index]
    if is_maximizes:
        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 develop
with all po
source codes:
knowledge ba
/* Facts : *
male(pet
male(joh
male(chri
male(kri
female(bel
female(j
female(
female(
parent of
parent of
parent of
parent of
parent of
parent of
parent of
parent of
/* RULES
/* son, p
* son, g
father(
mother(
grandf
grandm

Prolog - Family Tree

AIM:

To develop a family tree program using Prolog with all possible facts, rules and queens.

Source codes:

Knowledge base:

```
/* Facts : */
male(peter).
male(john).
male(chris).
male(kevin).
female(betty).
female(jeny).
female(lisa).
female(helen).
parentof(chris, peter).
parentof(chris, betty).
parentof(helen, peter).
parentof(helen, betty).
parentof(kevin, chris).
parentof(kevin, lisa).
parentof(jeny, john).
parentof(jeny, helen).
/* RULES :: */
/* 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) :- female(Y), parentof(X,Z), parentof(Z,Y).
```

brothis (X,Y):- male (Y), fathis (X,Z), fathis (Y,W), Z==W
sistis (X,Y):- female (Y), fathis (X,Z), fathis (Y,W), Z==W

Output

male (Petis)
true
fathis (chris - petes)
true
fathis (chris, betty)
false
mothis (chris, X)
X = betty
brothis (chris, helen)
false

Result:
Thus prolog for family tree program has been executed successfully.