INDEX

NAME: Aravind.U ROLL NO.: 220701518

STD: A DIV./SEC.: SUBJECT: POAI / AB Mahood

S.No.	Date	Title	~~Page No.~~ Marks	Teacher's Sign/ Remarks
1	31/7	Basic Python programming Calls	9	
2	7/8	Domain : Sentiment analysis	9	
3	4/9	N Queens	9	
4	4/9	Depth first Search	10	
5	11/9	A* Algorithm	10	
6	18/9	AO* Algorithm	10	
7	25/9	Decision tree	10	
8	9/10	K-mean	10	
9	16/10	Artificial Neural Network	10	
10	23/10	Minimax	10	
11	30/10	Introduction to prolog	10	
12	6/11	Prolog - Family tree.	10	

Completed

Sky Bird

26/7/24

1) My_list = [1, 2, 3, 4, 5]

first_element = my_list[0]

last_element = my_list[-1]

my list[2] = 10

my_list.remove(10)

for element in my_list

print(element)

Output: 1
2
4
5
0

2) Import numpy as np

my_array = np.array([1, 2, 3, 4, 5])

first_element = my_array[0]

last_element = my_array[-1]

my_array[2] = 10

my_array = np.append(my_array, 6)

my_array = np.delete(my_array, np.where(array

my_array =

for element in array: print(elem

3.

```
def factorial (x):
    if x==1:
    return
    else
    return x * factorial (x-1)
num = int(input("Enter a number"))
Print (factorial (num))
```

4.

```
a = int(input ("Enter a num2")
num 1 = 0
num 2 = 1
Print (num 1)
Print (num 2)
for i in range (2, 7, 1);
    Sum = num1 + num2
    print (Sum)
    num1 = num
    num2 = Sum
```

5) ~~list = {~~

```
list = ['apple', 'banana']
list.add ("ois")
print (list
```

OutPut = ['apple, org, banana

6/9/24

N Queens Problem

Aim: Write a Python Program for N Queens Problem

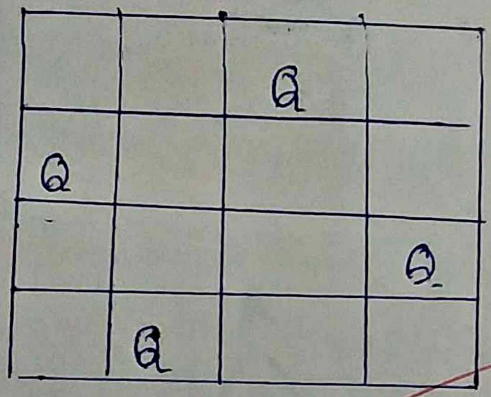
Program:

```
def Print_Solution(board):
    N = len(board)
    for i in range(N):
        row = ['Q' if board[i][j] else '.' for j in range(N)]
        Print(" ".join(row))
    Print()

def is_Safe(board, row, col, N):
    for i in range(col):
        if board[row][i]:
            return False

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

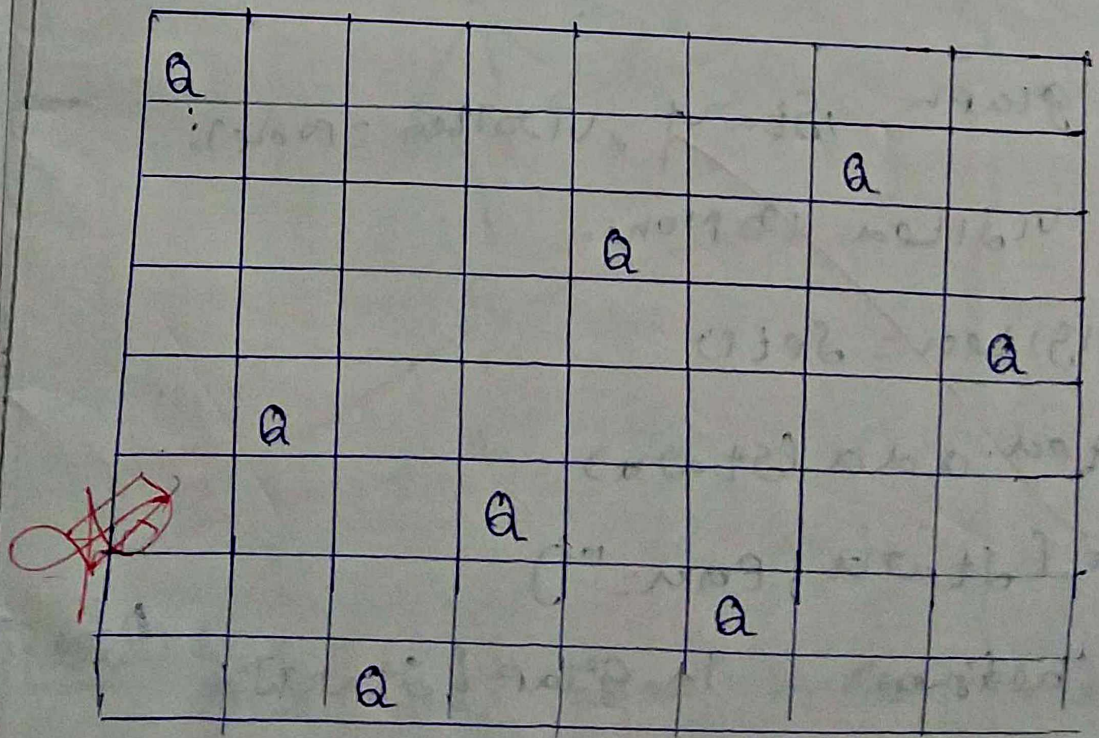
```
For i, j in zip(range(row, N, 1), range(col, -1, -1)):
    if board[i][j]:
        return False

return True

def Solve_nQueens_util(board, col, N):
    if col >= N:
        Print_Solution(board)
        return True

    res = False
    for i in range(N):
        if is_Safe(board, i, col, N):
            board[i][col] = True
    return res

def Solve_nQueens(N):
    board = [[False] * N for _ in range(N)]
    Print(f"Solution for {N}-Queens Problem:")
    Solve_nQueens_util(board, 0, N)
```

```
N_values = [4,8]

for N in N_values:
    Solve_n_queens(N)
print("\n" + "-"*20 + "\n")
```

OUTPUT : Solution for 4 queens



Solution for 8 queens



Result: Thus the above Python code executed successfully

DEPTH FIRST SEARCH (Graph)

Aim: write a Python code for DFS

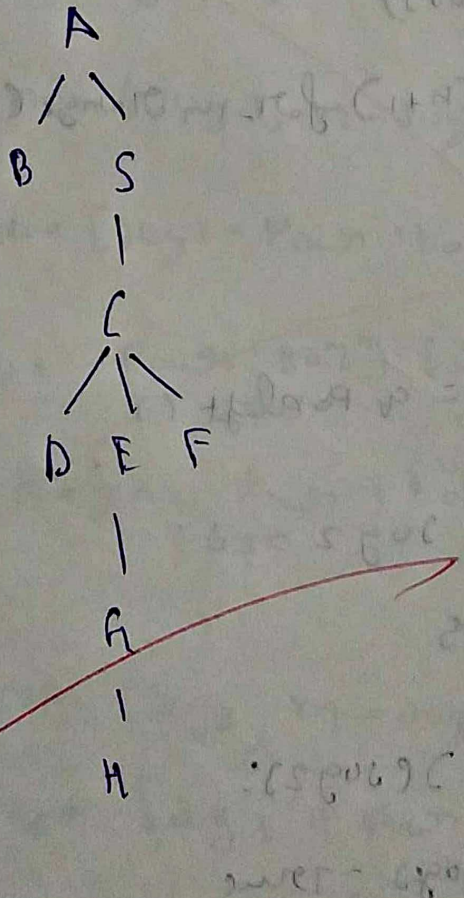
```
graph = {
    'A' : ['B', 'C'],
    'B' : ['A', 'D', 'E'],
    'C' : ['A', 'F', 'G'],
    'D' : ['B'],
    'E' : ['B'],
    'F' : ['C'],
    'G' : ['C']
}

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)

dfs(graph, 'A')

Output

A B D E C F G



Result:

Thus the above Python code for dfs traversal successfully

13/9/24

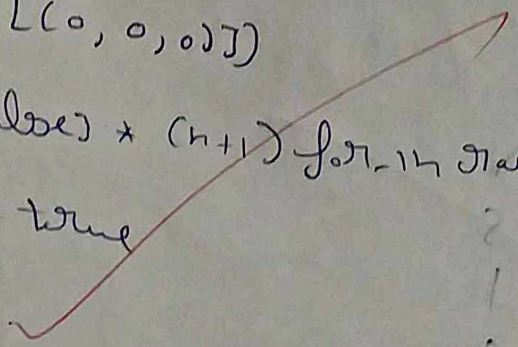
Water Jug Problem Using BFS

Aim:

To write a Pithoncode ^ water Jug Problem using BFS.

Program:

```
def min_steps(m, n, d):
    if d > max(m, n):
        return -1
q = deque([(0, 0, 0)])
visited = [[False] * (n+1) for _ in range(m+1)]
visited[0][0] = True
while q:
    jug1, jug2, steps = q.popleft()
    if jug1 == d or jug2 == d:
        return steps
    if not visited[m][jug2]:
        visited[m][jug2] = True
        q.append((m, jug2, steps + 1))
```



```
if not visited[jug1][n]:
    visited[jug1][n] = True
    q.append((jug1, n, steps+1))

if not visited[0][jug2]:
    visited[0][jug2] = True
    q.append((0, jug2, steps+1))

if not visited[jug1][0]:
    visited[jug1][0] = True
    q.append((jug1, 0, steps+1))

Pour1to2 = min(jug1, n - jug2)
if not visited[jug1 - Pour1to2][jug2 + Pour1to2]:
    visited[jug1 - Pour1to2][jug2 + Pour1to2] = True
    q.append((jug1 - Pour1to2, jug2 + Pour1to2, steps

Pour2to1 = min(jug2, m - jug1)
if not visited[jug1 + Pour2to1][jug2 - Pour2to
    visited[jug1 + Pour2to1][jug2 - Pour2to1] = Tru
    q.append((jug1 + Pour2to1, jug2 - Pour2to1, steps
```

return -1

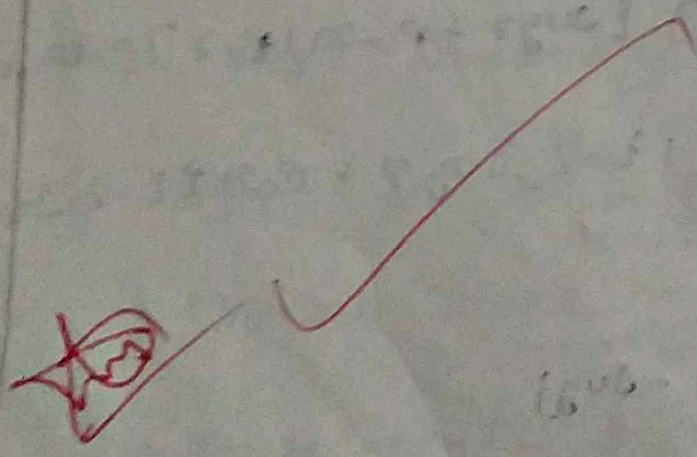
if __name__ == "__main__":

m, n, a = 4, 3, 2

print(min_steps(m, n, a))

OUTPUT

4



Result:

Thus the above Python code for water jug problem was executed successfully

A* Algorithm

Aim: To write a Python code for A* Algorithm.

Program:

```
def astar(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}
    Parents = {}
    g[start_node] = start_node
while len(open_set) > 0:
    n = None
for v in open_set:
    if n == None or g[v] + heuristic(v) < g[n] + heuristic
        n = v
if n == stop_node or Graph_node[n] == None:
    pass
```

```
else:
    for (m, weight) in get_neighbors(n):
        if m not in open_set and m not in closed_set:
            open_set.add(m)
            Parents[m] = n
            g[m] = g[n] + weight
        else:
            if g[m] > g[n] + weight:
                g[m] = g[n] + weight
                Parents[m] = n

        if m in closed_set:
            closed_set.remove(m)
            open_set.add(m)

    if n == None:
        print("Path does not exist!")
        return None
```

```
If n==Stop_node:
    Path = []
While Parent[n] != n:
    Path.append(n)
    n = Parent[n]
Path.append(Start_node)
Path.reverse()
Print("Path found: {}".Format(Path))
Return

OPen_Set.remove(n)
Closed_Set.add(n)
Print("Path does not exist!")
Return None

def get_neighbours(v):
    If v in graph_nodes
    Return graph_nodes[v]
Else

Def Heuristic(n)
    H_dist = {
    'A' = 11
    'B' = 6
    'C' = 99
```

'C' = 99, 'a' = 0,

'D' = 1

'E' = 7

}

graph_nodes = [

'A' : [('B', 1), ('F', 3)]

'B' : [('C') ('G', 9)]

'C' : None

'E' : [('D', 6)]

'D' : [('G'), 1)]

}

OUTPUT

If Path found: ['A', 'E', 'D', 'G']

Result:

Thus the Python program A* Alg was executed successfully.

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 DecisionTreeClassification() from tree
* Assign values for x and y
* Call the function Predict for predicating on the basis of given random values for each given feature
* Display the output

Code:

```
Import pandas as pd
from sklearn.tree import
DecisionTreeClassifier
data = {'Height': [152, 158, 172, 185, 167, 180, 157, 190
        164, 177],
weight: [48, 55, 72, 85, 62, 73, 52, 90, 66, 80],
```

'Gender': ['Female', 'Female', 'Male', 'Male', 'Fe
'Male', 'Female', 'Male', 'Female', 'Ma

y

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]], Column = ['Height', 'Weight'])

Predicted gender = Classifier.predict(Random_Vals

Print(f" Predicted Gender for height {height}
cm and weight {weight} kg
{Predicted_gender[0]}")

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

Result:

Thus the above Python code executed
Successfully.

Implementation of clustering techniques

Aim:

To Implement a K-Mean clustering technique using Python language

EXPLANATION:

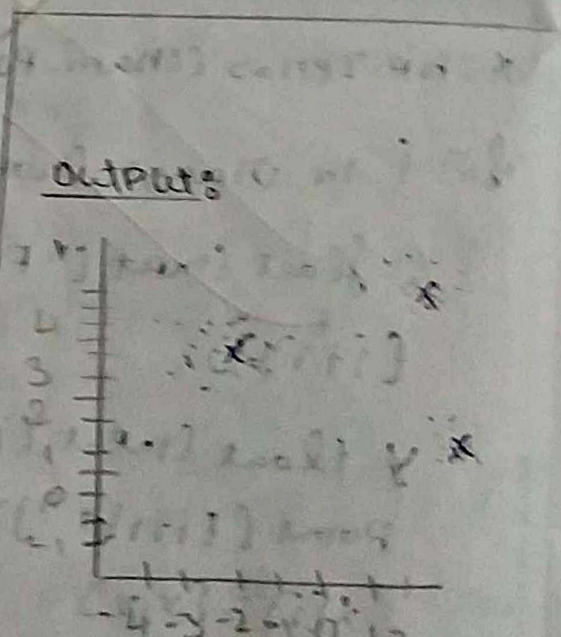
* Import kMean from Sklearn Cluster
* Assign x and y
* Calls the function kMean()
* Perform Scatter Operation and display the output

Code:

```
Import numpy as np
Import MatPlotlib.pyplot as plt

num_Points = int (Input ("Enter the number of data Points: "))

x = np.Zeros ((num_Points, 2))

for i in Range (num_Points):

    x = float (Input ("Enter x-coordinate for data Point {i+1}"))

    y = float (Input (f"Enter y-coordinate for data Point {i+1}: "))

    x[i] = [x, y]

num_clusters = int (Input ("Enter the number of clusters: "))
```

```
def kmeans (x, num_clusters, max_iters):
    centroids = x[np.random.choice(x.shape[0],
    num_clusters, replace=False)]
    for _ in range(max_iters):
        distances = np.linalg.norm(x(
    labels, centroids = kmeans(x, num_clusters
    num_clusters)
print("Cluster Labels:\n", labels)
print("Centroids:\n", centroids)
plt.figure(figsize=(8,6))
plt.scatter(x[:,0], x[:,1], c=labels,
cmap='viridis', marker='o' label='Data Points')
plt.scatter(centroids[:,0], centroids[:,1]
c='red', s=200, marker='x',
label='Centroids')
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.legend()
plt.grid(True)
plt.show()
```

Output:



Results

Thus the kmeans clustering technique has been implemented successfully.

Implementation artificial neural network for an application using Python - Regression

Aim :

To Implementing artificial neural networks for an application in regression using Python

Algorithm : Generate Data : Create synthetic data for training

Prepare Data : Normalize or Scale the data.

Split Data : Divide data into training and testing sets.

Build Model : Define the neural network structure (layers and nodes).

Compile Model : Select optimizer and loss function

Train Model Fit the model to the training data over Multiple epochs

Evaluate Model : Test model performance using the Testing set

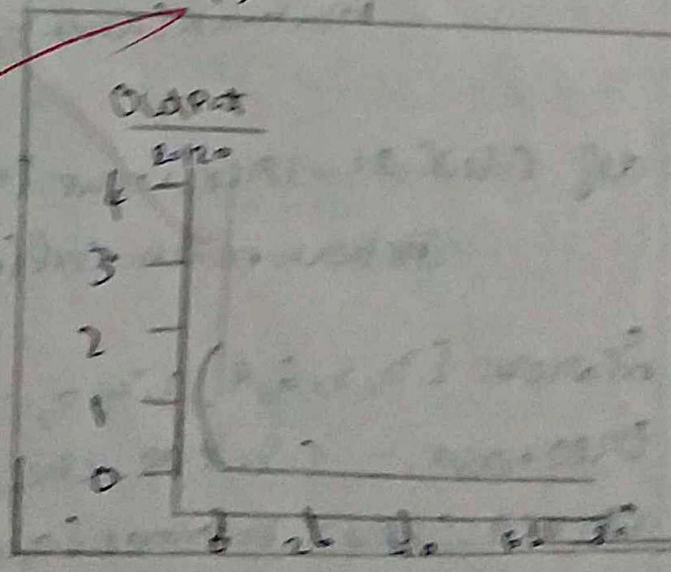
Predict : Use the model to make predictions on new data.

Visualize : Compare predictions with actual results.

Code:

```
#Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import
    train_test_split
    StandardScaler
from tensorflow import keras
from tensorflow.keras import layers

# Step: Data Gen
np.random.seed(0)
X = 2 * np.random.rand(1000, 1)
y = 4 + 3 * X + np.random.rand(1000, 1)

Scaler_X = StandardScaler()
Scaler_y = StandardScaler()
X_Scaled = Scaler_X.fit_transform(X)
y_Scaled = Scaler_y.fit_transform(y)

X_train, X_test, y_train, y_test = test_test_split
(X_Scaled, y_Scaled, test_size = 0.2 random_state=1)

Model.compile(optimizer = 'adam', loss = 'mean_squared
    error')
```

```
Model.fit(X_train, y_train, epochs=100, batch_size=
    verbose=0)

loss = model.evaluate(X_test, y_test)
print("Test Loss: {loss}")

y_pred_scaled = model.predict(X_test)
y_pred = scaler_y.inverse_transform(y_pred_scaled)

plt.figure(figsize=(6,6))
plt.scatter(scaler_y.inverse_transform(y_test), scaler_
y_inverse_transform(y_test), label='Actual Data',
color='blue', alpha=0.5)
plt.scatter(scaler_y.inverse_transform(y_test), y_
label='Predictions', color='red',
alpha=0.5)
plt.title('Actual vs Predicted')
plt.xlabel('y')
plt.ylabel('y')
plt.legend()
plt.show()
```



Result: Thus the artificial neural network regression model successfully.

MinMax Algorithm

Aim:-

To Implement MINMAX algorithm using Python

Code:

```
Import math
def minimax(depth, node_index, is_maximizer, scores,
  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,
             minimax(depth+1, node_index*2+1, True,

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, score, tree height)
Print(f"The Optimal Score is: {Optimal_score}")
```

Output:

The Optimal Score is: 5

Result:

Thus the minimax algorithm has been implemented Successfully.

Introduction To Prolog

Aim:-

To learn 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 lower case letter

eg. dog, ab-C_321

2) Variables:-

They are string of letters, digits and the underscore, starting with capital letter or an underscore

3) Compound terms:

Compound terms are made up of a prolog atom and a number of arguments enclosed in parentheses and seperated 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). like_a_beetle.

5) Rules

A rule consist of a head and body

eg: is_smaller (x,y_bigger (y,x)

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

Output:

?- woman(mia)

true

?- playsAirGuitar(mia)

false

?- concert

Error = Unknown procedure: concert/0

KB2

happy(yolanda)

listens2Music(mia)

listens2Music(yolanda) :- happy(yolanda)

playsAirGuitar(mia) :- listens2Music(mia)

playsAirGuitar(yolanda) :- listens2Music(yolanda)

output :-

? playsguitar (mia)

true

? - plays guitar (X, dana)

True

KB3:

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

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

KBS

Owns(Jack, Car(bmw))

Owns(John, Car(horse))

Own(Olivia, Car(Wild))

Own(Jane, Car(hers))

Seden(Car(bmw))

Truck(Car(heavy))

Output:-

?- Owns(John, x)

x = Car(heavy)

?- Owns(John, -)

True

Results

The basic Prolog Program has been executed successfully.

Prolog-family tree

Aim:

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

Source Code:

Knowledge base:

/* facts : : */

male (Peter)
male (John)
male (Hugh)
male (Kevin)
female (Jen)
female (Kelly)
female (lisa)
female (Helen)
parent of (Chris, Peter)
Parent of (Chris, kelly)
Parent of (Helen, Peter)
Parent of (Helen, Kelly)
Parent of (Kevin, Chris)

parent of ([illegible] , [illegible])

parent of (Jeny , [illegible])

parent of (Jeny , helen)

/* RULES = */

/* Son, Parent *

* Son, grandParent */

father (X, Y) :- male (Y), Parent (X, Y)

mother (X, Y) :- female (Y), Parent (X, Y)

grandfather (X, Y) :- male (Y), Parent (X, Z), Parent (Z, Y)

grandmother (X, Y) :- female (Y), Parent (X, Z), Parent (Z, Y)

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

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

Outputs

male (Peter)

true

father (Chris, betty)

false

father (Chris, betty)

mother (Chris, X)

X = betty

brother (Chris, helen)

false

Result:

This prolog for family tree prog has been executed successfully.