NAME M Sri Charulata

INDEX

STD CSE DIV. A ROLL NO. 22CT1053

S.No.	Date	Topic	Page No.	Signature
1.	09/8/24	8 Queens Problem	10	
2.	16/8/24	DFS	10	
3.	23/08/24	Depth first search water jug problem	10	
4.	30/8/24	A* search Algorithm	10	
5.	06/09/24	Implementation of Decision Tree classification Technologies	10	
6.	27/9/24	Implementing ANN for an application using python Regression	10	
7.	04/10/24	Implementation of clustering Technique	10	
8	18/10/24	MINIMAX algorithm	10	
9.	25/10/24	Introduction to prolog	10	
10.	08-11-2024	Prolog Family Tree	10	

completed

Python Programs

1. Basic Calculator

```
def add(x, y):
    return x+y

def subtract(x,y):
    return x-y

def multiply(x,y):
    return x*y

def divide(x, y):
    if y!=0:
        return x/y
    else:
        return "Error!"

def main():
    print("select operation:")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")

choice = input("Enter choice:")
num1 = float(input("Enter first number:"))
num2 = float(input("Enter Second number:"))
if choice == '1':
    print(f"The result is: {add(num1, num2)}")

elif choice == '2':
    printf(f"The choice result is {subtract(num1, num2)}
```

```
print elif choice == '3':
        print (f"The result is : {multiply (num1, num2)}")
    elif choice == '4':
        print (f"The result is : {divide (num1, num2)}")
    else:
        print ("Invalid Input")

if __name__ == "__main__":
    main()
```

Output:

```
Select operation :
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice : 1
Enter first number : 3
Enter Second number : 5
The result is : 8.0
```

2. Temperature Converter.

```
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32

def fahrenheit_to_celsius (fahrenheit):
    return (fahrenheit - 32) * 5/9
```

```
def main():
    print("Temperature converter")
    print("1. celsius to fahrenheit")
    print("2. Fahrenheit to celsius")
    choice = input("Enter choice:")
if choice == '1':
    celsius = float(input("Enter temp in celsius:"))
    print(f"{celsius}°C is equal to {celsius_to_fahrenheit
                                    (celsius)}°F")

elif choice == '2':
    fahrenheit = float(input("Enter temperature in
                              Fahrenheit:"))

    print(f"{fahrenheit}°F is equal to {fahrenheit_to_
          celsius(fahrenheit)}°C")

else:
    print("Invalid choice")

if __name__ == "__main__":
    main()
```

Output:

```
Temperature Converter
1. celsius to Fahrenheit
2. Fahrenheit to celsius
Enter your choice: 1
Enter temp in celsius: 32
32.0°C is equal to 89.6°F
```

8.

```
import cmath
def solve_quadratic(a, b, c):
    discriminant = cmath.sqrt(b**2 - 4*a
    root1 = (-b + discriminant) / (2*a)
    root2 = (-b - discriminant) / (2*a
    return root1, root2

a = float(input("Enter coefficient a
b = float(input("Enter coefficient b
c = float(input("Enter Coefficient c:
    root1, root2 = solve_quadratic(a, b,c
    print(f"The roots are {root1} and {roo
```

Output:

```
Enter coefficient a: 1
Enter coefficient b: 2
Enter coefficient c: 3
The roots are (-1+1.4142j) and (-1-1.4
```

4. Prime Number within Range

```
def is_prime(n):
    if n<=1:
        return false
    for i in range (2, int (n** 0.5) +1):
    if n%.i ==0;
        return false
    return True

def primes_in_range (start, end):
    return ( n for n in range (start, end +1) if is_pri
                                                (n) ]

start = int (input ("Enter start of range :"))
end = int (input ("Enter end of rang : "));
print (f" prime numbers betw {start} and {end}:
        { primes_in_range ( start, end) } ")
```

5. Palindrome Checker.

```
def is_palindrome (s):
    s = s. lower ()
    return s == s [::-1]
string = input (" Enter a string :")
if is_palindrome ( string ):
    print (" The string is palindrome.")
else :
    print (" The string is not a Palindrome.")
```

6. Fibonacci Series

```
def fibonacci(n):
    sequence = []
    a, b = 0, 1
    while len(sequence) < n:
        sequence.append(a)
        a, b = b, a+b
    return sequence
n = int(input("Fibonacci series"))
print(fibonacci(n))
```

Mini Project :

Title : Gold Price prediction using Machine learning with Python

Problem Statement :

Gold prices are a key economic indicator that affects traders, investors and business across multiple areas. Fluctuations in the price of gold can have influence decision making and financial objectives. Predicting gold prices is difficult due to number of factors such as market trends, investor sentiment etc.

Target Audience :

1. Investors : Individual looking to invest in gold seek for predictions to make detailed decisions.
2. Jewellery buyers : Consumers who purchase gold for personal / industrial purpose may use predictions to manage costs.
3. Financial Advisors : Professionals who provide investment advices to clients use predictions to guide their Recommendations.

4. Traders: Retail traders who buy and gold in various markets can make predictions to time their trades.

5. Financial and News media: journalists media outlets could use this prediction system for reporting on gold markets

MINI Project

TITLE: Gold Price Prediction using Machine Learning with python

Problem Statement:

Gold price are a key economic indicates that affects traders, investors and business across multiple areas. Fluctuations in the price of Gold can influence decision making and financial objectives, Predicting gold prices is difficult due to the number of factors such as market trends, investors sentiments, etc.

Target Audience:

1. Investors: Individuals looking to invest in gold seek for predictions to make detailed decisions.
2. Jewellery buyers: Consumers who purchase gold for personal/industrial purpose may use predictions to manage costs
3. Financial Advisors: Proffessionals who provide investment advices to clients use prediction to guide their recommendations

4. Traders: Retail traders who buy and sell gold in various markets can rely on prediction to time their trades.

5. Financial News Media: journalists and media outlets could use their prediction system for reporting on gold market

objectives:

1. User Experience: The interface provides clear and actionable insights like visualizations help users make decisions quickly.

2. Accuracy: Refining the model to maintain high accuracy and reliable under different conditions.

3. Efficiency: Use efficient techniques to handle large datasets and reduce time required for training and predictions.

ABSTRACT :

This project uses machine learning to predict gold prices. This process includes applying algorithms like random forest and KNN to make accurate predictions. The result is a user-friendly tool that provides reliable predictions and visualizations to help users make informed decisions.

DATA SET :

1. Yahoo Finance Gold Data
2. Geeksforgeeks
3. World Gold council's data.

ALGORITHMS USED :

1. Random Forest :

This algorithm is an ensemble learning method that combines multiple decision trees to improve prediction accuracy. It works by training on random subsets of data and aggregating their predictions.

2. KNN (K-Nearest Neighbours): simple, instance based ML method used for classification. for classification it assigns the most common class among the neighbours, while for regression it averages the value of neighbours.

6/9/24

N Queens

Aim: To implement N Queens problem using python.

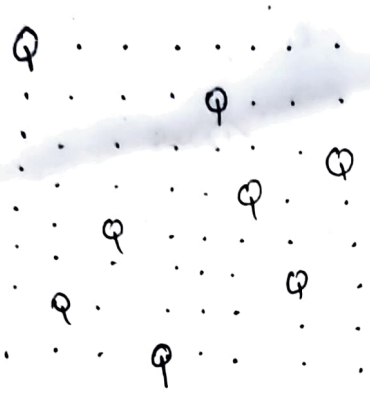
Code:

```
def print_board(board):
    for row in board:
        print(" ".join("Q" if col else "." for col in row))
    print()

def is_safe(board, row, col, N):
    for i in range(row):
        if board[i][col]:
            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, -1, -1), range(col, N)):
        if board[i][j]:
            return False
    return True

def solve_n_queens_util(board, row, N):
    if row >= N:
        return True
```

```
    for col in range(N):
        if is_safe(board, row, col, N):
            board[row][col] = True
            if solve_n_queens_util(board, row+1,
                                   N):
                return True
            board[row][col] = False
    return False

def solve_n_queens(N):
    board = [[False] * N for _ in range(N)]
    if not solve_n_queens_util(board, 0, N):
        print("No solution exists")
    else:
        print_board(board)

def main():
    try:
        N = int(input("Enter the number of Queens(N): "))
        if N <= 0:
            print("The number of queens must be
                  a positive integer.")
        else:
            solve_n_queens(N)
    except ValueError:
        print("Invalid input. Please enter a
              positive integer.")

if __name__ == "__main__":
    main()
```

Output :

Enter the number of queens (N): 8



Enter the number of queens (N) : 2

NO solution exists.

Enter the number of queens (N) : 4

Result :

Thus, the program of 8 N Queens Algorithm using Python has been executed successfully.

10/9/24

Depth First Search

Aim:

To write a program of Depth first search using python.

Code:

```
class Node:
    def __init__(self, value):
        self.value = value
        self.neighbours = []
    def add_neighbours(self, neighbor):
        self.neighbors.append(neighbor)

class graph:
    def __init__(self):
        self.nodes = {}
    def add_node(self, value):
        if value not in self.nodes:
            self.nodes[value] = Node(value)
    def add_edge(self, from_value, to_value):
        if from_value in self.nodes and to_value
                in self.nodes:
            self.nodes[from_value].add_neighbors(self.
                nodes[to_value])
```

```
        self.nodes[to_value].add_neighbor(self.nodes[from_value])

    def dfs(self, start_value):
        visited = set()
        stack = [self.nodes.get(start_value)]

        while stack:
            node = stack.pop()
            if node and node.value not in visited:
                print(node.value)
                visited.add(node.value)

                for neighbor in reversed(node.neighbors):
                    if neighbor.value not in visited:
                        stack.append(neighbor)

graph = Graph()
graph.add_node('A')
graph.add_node('B')
graph.add_node('C')
graph.add_node('D')
graph.add_node('E')
graph.add_node('F')

graph.add_edge('A', 'B')
graph.add_edge('A', 'C')
graph.add_edge('B', 'D')
graph.add_edge('B', 'E')
graph.add_edge('C', 'F')
graph.add_edge('E', 'F')
```

```
print ("DFS:")
graph.dfs('A')
```

Output

A
B
D
E
F
C

Result

Thus, the program to find Depth first search using python has been executed and verified successfully

13/9/24

Exercise - A*

Aim: To write a program using A* Algo

Code:

```
def ast (start, stop):
    open = set (start)
    close = set()
    g = {}
    parents = {}
    g[start] = 0
    parents [start] = start
while len (open) > 0:
    n = None
    for v in open:
        if n == None or g[v] + heuristic
                                    g[n]
            n = v
    if n == stop or Graph[n] == None:
        pass
    else:
        for (m, weight in get_neigh (n)
            if m not in open and m no
                open.add(m)
                parents[m] = n
                g[m] = g[n]
```

```
else:
    if g[m] > g[n] + weight:
        g[m] = g[n] + weight
        parents[m] = n

    if m in close:
        close.remove(m)
        open.add(m)

if n == None:
    print("path does not exist")
    return None

if n == stop:
    path = []

    while parents[n] != n:
        path.append(n)
        n = parents[n]
    path.append(start)
    path.reverse()
    print("Path found {}".format(path))
    return

open.remove(n)
close.add(n)
print("Path does not exist")
return None
def getWeigh(v):
    if v in Graph:
        return Graph[v]
```

```
else:
    return None

def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 99,
        'D': 1,
        'E': 7,
        'G': 0,
    }
    return H_dist[n]

Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1), ('G', 9)],
    'C': None,
    'E': [('D', 6)],
    'D': [('G', 1)],
}
ast('A', 'G')
```

Output:

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

Result:

Thus, the program using A* algorithm has been executed and verified successfully

Water jug problem using DFS

Aim:

To write a code to implement Water jug problem using Depth first search Algorithm.

Code:

```
def water_jug_prb(jug1, jug2, target):
    j1 = 0
    j2 = 0
act = [("fill", 1), ("fill", 2), ("empty", 1), ("empty", 2),
       ("pour", 1, 2), ("pour", 2, 1)]
Visit = set()
stack = [(j1, j2, [])]
while stack:
    j1, j2, seq = stack.pop()
    if (j1, j2) not in Visit:
        Visit.add((j1, j2))
    if j1 == target or j2 == target:
        return seq
    for action in act:
        if action[0] = "fill":
            if action[1] == 1:
                next_state = (jug1, j2)
            else
                next_state = (j1, jug2)
```

```
elif action[0] == "empty":
    if action[1] == 1:
        next_state = (0, j2)
    else:
        next_state = (j1, 0)
else:
    if action[1] == 1:
        amt = min(j1, jug2 - j2)
        next_state = (j1 - amt, j2 + amt
    else:
        amt = min(j2, jug1 - j1)
        next_state = (j1 + amt, j2 - amt
if next_state not in visit:
    next_seq = seq + [action]
    stack.append((next_state[0], next_state[1]
                  next_s
return none
res = water_jug_prb(3, 2, 1)
print(res)
```

Output:

[('fill', 2), ('pour', 2, 1), ('fill', 2), ('pou

Result:

Thus, the program to implement water jug problem using DFS Algorithm has been executed successfully.

Aim:

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

Explanation:

→ Import tree from sklearn
→ Call the function DecisionTree Classifier() from tree
→ Assign values for x and y
→ Call the function predict for predicting on the basis of given random values for each given feature
→ Display the output

code:

```
import pandas as pd
from sklearn.tree import
Decision Tree Classifier
data = {
    'Height': [152, 155, 172, 185, 167, 180, 157, 180, 164,
        177],
    '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 = DecisionTreeClassification()
classifier.fit(x, y)

height = float(input("Enter height (in cm) fo
weight = float(input("Enter weight (in kg) f
random_values = pd.DataFrame([[height, wei
columns = ['Height', 'Weight']])
predicted_gender = classifier.predict(random_va
print(f"Predicted gender for height {height} cm
    weight {weight} kg: {predicted_gender[0]
```

Output: Enter height (in cm) for prediction: 170

Enter weight (in kg) for prediction: 75

Predicted gender for height 170.0 cm and weight 75.0 kg: Male.

Result: Thus, a decision tree classification algorithm for gender classification using python has been executed.