Veera's Student's Choice

Name : Amrutha · B·J

STD : CSE SEC : A ROLL NO. : 25

S.No.	Date	Title	Page No.	Teacher's Sign
1	31/07/24	8 Queens problem	2	9
2	7/08/24	Depth first search	3	9
3	14/8/24	DFS-water jug problem	4	9
4	14/8/24	Minmax algorithm	10	10
5	11/9/24	A* search algorithm	3	10
6	30/10/24	Introduction to prolog	8	10
7	6/11/24	prolog family tree	9	10
8	16/10/24	Artificial neural network-regression	5	10
9	28/9/24	Decision tree	6	10
10	9/10/24	K-means	7	10
11	31/07/24	sample programs	1	10
12	16/10/24	prolog-family tree	11	10

completed

Exercise - 1

Date: 31/07/24

PROGRAM:

1) Fibanocci series

```
def fibonacci(n):
    fib_sequence = [0,1].
    for i in range(2,n):
        next_value = fib_sequence[-1] + fib_sequence[-2].
        fib_sequence.append(next_value).
    return fib_sequence.

n = 10
print(fibonacci(n)).
```

Output: 0 1 1 2 3 5.

2. Prime Number Checker:

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

num = 29.
print(f"{number} is prime: {is_prime(number)}").
```

OUTPUT:

29 is prime.

3) Merge two sorted arrays.

```
def merge_sorted_array(list1, list2)
sorted_list = []
i = j = 0
while i < len(list1) and j < len(list2):
    if (list1[i] < list2[j]):
        sorted_list.append(list1[i])
        i += 1
    else:
        sorted_list.append(list2[j])
        j += 1
sorted_list.extend(list1[i:])
sorted_list.extend(list2[j:])
return sorted_list

list1 = [1, 3, 5, 7]
list2 = [2, 4, 6, 8]
print(merge_sorted_lists(list1, list2))
```

OUTPUT: [1, 2, 3, 4, 5, 6, 7, 8]

4) Find longest substring without Repeating characters.

```
def length_of_longest_substring(s):
    char_index = {}
    max_length = start = 0
    for i, char in enumerate(s):
        if char in char_index and char_index[char] >=
            start = char_index[char] + 1
        char_index[char] = i
        max_index = max(max_length, i - start + 1)
    return max_length
string = 'ababcbcb'
print("length of longest substring without repeating characters :",
```

OUTPUT: [illegible]

5. def rotate_matrix(matrix):

```
    n = len(matrix)
    for layer in range(n//2):
        first = layer
        last = n - layer - 1
        for i in range(first, last):
            offset = i - first
            top = matrix[first][i]
            matrix[first][i] = matrix[last-offset][first]
            matrix[last-offset][first] = matrix[last][last-offset]
            matrix[last][last-offset] = matrix[i][last]
            matrix[i][last] = top
    return matrix
```

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
rotated_matrix = rotate_matrix(matrix)
for row in rotated_matrix:
    print(row)
```

OUTPUT:
[7, 4, 1]
[8, 5, 2]
[9, 6, 3]

6. Binary Search Algorithm:-

```
def binary_search(arr, target):
    left, right = 0, len(arr)-1
    while left <= right:
        mid = (left + right)//2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
```

```
sorted_list = [1,2,3,4,5,6,7,8,9]
target = 7
result = binary_search(sorted_list, target)
print(f"Element {target} is at index: {result}")
```

OUTPUT: The target index : 3.

7) find duplicates.

```
def find_duplicates(arr):
    duplicates = []
    num = nt()
    for item in arr:
        if item in num:
            duplicates.append(item)
        else
            num.add(item)
    return duplicates.
```

```
input_list = [1,2,3,4,5,3,2,1]
print(f"Duplicate in the list : {find_duplicates(input_list)}").
```

OUTPUT:
The duplicate numbers is : 2

8) ~~class Queueusing stacks~~ : find duplicate :

```
def init(self):
    self.stack1 = [].
    self.stack2 = [].
def enqueue(self, item):
    self.stack1.append.
```

```
def find_duplicate(nums):
    slow = fast = nums[0]
    while True:
        slow = nums[slow]
        fast = nums[nums[fast]]
        if slow == fast:
            break.
    slow = nums[0].
    while slow != fast:
```

```
dup in   nums [slow]
find   nums [fast]
return slow
num = [1,3, 4,2,2]
print (f" The duplicate number is {find_duplicate (num)}")
```

OUTPUT:-
The duplicate number is 2

9) To print to print maximum element in array.

Program:-

```
n = int (input ()):
a = []
for i in array (0,n):
        a.append (input ())

c = max(a)
print (c)
```

OUTPUT:- [1,2,3,4,5]
5

10) python program to find the no. of even and odd numbers

```
n = int (input (" Enter the size of array : "))
a = [1,2,3,4,5]
even = 0.
odd = 0

for i in range ("enter the no"):

a=[] a.append (int (input ())
even=0
for i in a
    if i % 2==0.
       even + = 1
    if i % 2 ! = 0
       odd + = 1.
print ('No. of odd', even).
print ('No. of even', Odd)
```

OUTPUT:

Number of even : 4

Number of odd : 3

Date : 14/8/24

```
print_solution(board):
    for row in board:
        print(" ".join("Q" if x else "." for x in row))
    print()

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

solve_n_queens_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.
            res = solve_n_queens_util(board, col+1, N) or res
            board[i][col] = false.
```

```
def solve_n_queens(N):
    board = [[False]*N for _ in range(N)]
    if not solve_n_queens_util(board, 0, N):
        print("Solution does not exist")
    else:
        print("Solution found:")

solve_n_queens(8)
```

OUTPUT :-

4

found solution [1, 3, 0, 2] in 4 steps

Found solution [2, 0, 3, 1] in 4 steps

8 :

Found solution [0, 5, 8, 4, 7, 6, 6, 3] in 115 steps

Found solution [4, 1, 3, 5, 7, 2, 0, 6] in 2 steps.

Found solution [3, 1, 6, 4, 0, 7, 5, 2] in 930 steps

Found solution [5, 2, 0, 7, 9, 1, 3, 6] in 217 steps.

Found solution [2, 6, 1, 7, 5, 3, 0, 4] in 1869 steps

Found solution [4 6, 1, 3 7, 0, 2, 5] in 5017 steps

Found solution [3, 1, 7, 4, 6, 0, 2, 5] in 451 steps.

Found solution [4, 6, 7, 3, 1, 6, 2, 5] in 936 steps.

Result :-

Thus the 8 queens problem was executed successfully

Date: 11/9/24

* algorithm:

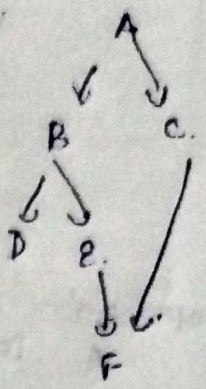
```
import heapq

class Node:
    def __init__(self, name, parent=None, g=0, h=0):
        self.name = name
        self.parent = parent
        self.g = g
        self.h = h
        self.f = g+h

    def __lt__(self, other):
        return self.f < other.f

def heuristic(node, goal):
    return abs(node[0] - goal[0]) + abs(node[1] - goal[1])

def a_star(start, goal, graph):
    open_list = []
    closed_list = set()
    start_node = Node(start, None, 0, heuristic(start, goal))
    heapq.heappush(open_list, start_node)
    while open_list:
        current_node = heapq.heappop(open_list)
        if current_node.name == goal:
            path = []
            while current_node:
                path.append(current_node.name)
                current_node = current_node.parent
            return path[::-1]
```

```
for neighbor, cost in graph[current_node.name]:
    if neighbor in closed_set:
        continue
    g_cost = current_node.g + cost
    h_cost = heuristic(neighbour, goal)
    neighbor_node = Node(neighbor, current_node, g_cost, h_cost)
    if any(node.name == neighbor and node.g <= g_cost
           for node in open_set):
        continue
    heapq.heappush(open_set, neighbor_node)

return None.
```

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

```
visited = set()
print("DFS traversal using recursive algorithm
```

```
graph = {
    (0,0) : [((0,1),1), ((1,0),1)],
    (0,1) : [((0,0),1), ((0,2),1)],
    (0,2) : [((0,1),1), ((1,2),1)],
    (1,0) : [((0,0),1), ((1,1),1)],
    (1,1) : [((1,0),1), ((1,2),1)],
    (0,2) : [((1,1),1), ((2,2),1)],
    (2,2) : [((1,2),1)],
}

start = (0,0)
goal = (2,2)
path = a_star(start, goal, graph)
print("Path from start to goal:", path)
```



OUTPUT :-

A B D E F C

Result :-

Thus the A* program was successfully executed.

```
def dfs_recursive(graph, node, visited):
    if node not in visited:
        print(node, end=" ")
        visited.add(node)
        if node in graph:
            for neighbour in graph[node]:
                dfs_recursive(graph, neighbour, visited)

graph = {
    'A': ['B', 'C', 'D'],
    'B': ['E'],
    'C': ['G', 'F'],
    'D': ['H'],
    'E': ['I'],
    'F': ['J'],
    'G': ['K'],
    'H': [],
    'I': [],
    'J': [],
    'K': []
}

visited = set()
print("DFS traversal using recursive approach:")
dfs_recursive(graph, 'A', visited)
```

OUTPUT :-

A B D E F C

RESULT :-

Thus the DFS program is executed successfully.

water jug using DFS

```
def full-4-gallon (a, y, x-max, y-max):
    return (x-max, y)

def full-3-gallon (x, y, x-max, y-max):
    return (x, y-max)

def empty-4-gallon (x, y, x-max, y-max):
    return (x, y)

def pour 4 to 3 (x, y, x-max, y-max):
    transfer = min (x, y-max-y)

def pour 3 to 4 (x, y, x-max, y-max):
    transfer = min (y-max, x)
    return (x+transfer, y-transfer)

def dfs (water jug (x-max, y-max, goal-x, visited = None, start=(0,
    if visited is None:
        visit = set()
        stack = [start]
    while stack:
        state = stack.pop()
        x, y = state
        if state in visited:
            continue
        visited.add(state)
        print (f'visited state: {state} y"y
        if x = goal-x:
            print (f"goal reached: {state} y")
            return state
```

next state :-

[fill 4-gallon (x,y) x-max, y-max),
fill 3-gallon (x,y, x-max, y-max),
empty 3-gallon (x,y, x-max, g-max),
empty 4 gallon (x,y, x-max, y-max)
empty 4 to 3 (x,y, x-max, y-max)
empty 3 to 4 (x,y), x-max, y-max 2

for new_state in next_state:
    if new_state not in visited
        state.append (new_state)

return none.

xmax = 4.
gmax = 3
goalx = 2.

dfs_water_jug (x_max, gmax, 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 : (1,3)
visiting state : (1,0)
visiting state : (0,1)
visiting state : (4,1)
visiting state : (2,3)

Goal reach(2,3)

(2,3)

Result :

Thus the water jug problem using DFS is executed successfully.

Date : 10/10/24

Artificial Neural Network

Aim :-

To implement artificial Neural network for an application using python.

CODE

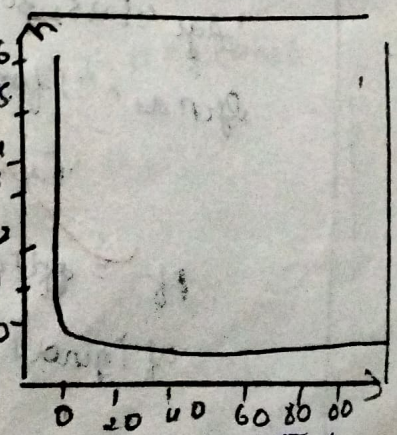
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt

np.random.seed(42)

x = np.random.rand(1000,3)

y = 3 + x[:,0] + 2 * x[:,1] + x[:,2] + 1.5 + np.sin(x[:,2]) +
    np.random.normal(0,0.1,1000)

x_train, x_test, y_train, y_test = train_test_split(x,y,
test_size = 0.2, random_state = 42)

Scaler = StandardScaler()

x_train = scalar.fit_transform(x_train)

x_test = scalar.transform(x_test)
```

```
model = Sequential()
model.add(Dense(10, input_dim = x_train.shape[1], activation
                                                = 'relu'))
model.add(Dense(10, activation = 'relu'))
model.add(Dense(1, activation = 'linear'))
model.compile(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, verbose = 1)

ypred = model.predict(x_test)

mse = np.mean((y_test - ypred.flatten())**2)
print(f'Mean Squared Error = {mse:.4f}')

plt.figure(figsize = (10,6))
plt.plot(history.history['loss'], label = 'Training Loss')
plt.plot(history.history['val_loss'], label = 'Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

OUTPUT:

```
20/20
Epoch 86/100
20/20
Epoch 87/100
20/20
Epoch 88/100
20/20
Epoch 89/100
20/20
Epoch 90/100
20/20
Epoch 91/100
```

```
0s loss: 0.0137
0s loss: 0.0108 val_loss: 0.0[illegible]
0s loss: 0.0132 val_loss: 0.0191
0s loss: 0.0147 val_loss: 0.0227
0s loss: 0.0135 val_loss: 0.0182
0s loss: 0.0035 val_loss: 0.0188
```



Result:-

Thus the above program was successfully executed

Decision Tree

Aim :-

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

Code :

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn import tree

data = {
'height' : [150, 160, 170, 180, 155, 165, 175, 185, 140, 145],
'Weight' : [50, 60, 70, 80, 55, 65, 75, 85, 45, 50],
'Age' : [25, 30, 35, 40, 29, 32, 37, 42, 44, 47],
'gender' : ['female', 'Female', 'Male', 'Male', 'Female',
            'Female', 'Male', 'Male', 'Female', 'Female']}

df = pd.DataFrame(data)
df['gender'] = df['gender'].map({'female': 0, 'Male': 1}
```

```
X = df[['height', 'weight', 'Age']].
Y = df['gender']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,
                                                    random_state=42,
                                                    stratify=Y).

clf = DecisionTreeClassifier().
clf.fit(X_train, y_train)

accuracy = accuracy_score(y_test, y_pred).
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred, zero_division=0)

print(f"Accuracy: {accuracy:.2f}").
print("Confusion Matrix:\n", conf_matrix).
print("Classification Report:\n", class_report)
plt.figure(figsize=(12,8))
tree.plot_tree(clf, feature_names=X.columns,
               class_names=['Female', 'Male'], filled=True).
plt.title("Decision tree for gender classification").
plt.show().
```

OUTPUT:-

Enter height (in cm) for prediction: 65
Enter weight (in kg) for prediction: 60.
Predicted gender for height 165 cm and weight 60.0 kg: Female

Result:-

Thus the above program was successfully executed

Date :- 9/10/24

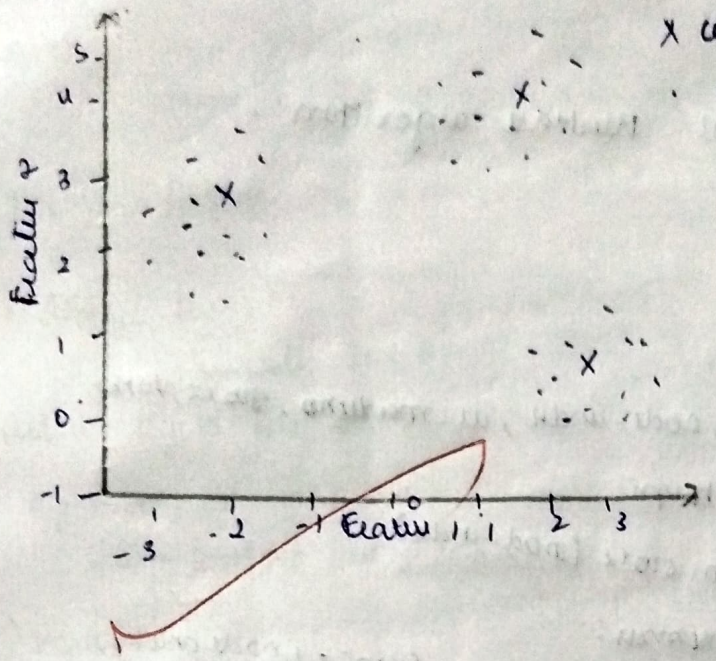
Kmeans

Aim :-

To implement a kmeans clustering technique using python language

Program..

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn cluster import k-means
from sklearn.datasets import make_blobs
x, y_true = make_blobs(n_samples=300, centers, 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(figsize=(8,6))
plt.scatter(x[:,0], x[:,1], y=kmeans.s=50,
            cmap='viridis', label='clusters').
centers = kmeans.cluster_centers_
plt.scatter(centers[:,0], centers[:,1], c='red',
            s=200, alpha=0.75, marker='x',
```



Result:

Thus the above O/P is verified

Aim :-

To Implement Minimax algorithm.

Code :-

```
import math

def minimax (depth, node_index, is_maximizing, scores, height):

    if depth == height:
        return scores[node_index]

    if is_maximizing:
        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.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("Optimal score is:", optimal_score)
```

O/P :-

Optimal score is : 5

Result :- Thus the o/p is verified.

Introduction to PROLOG.

Date: 30/10/24

Aim

To learn Prolog terminologies and write basic programs

Terminologies:-

1) Atomic terms:-

Usually the strings made up of lower and upper case letters digits the underscore starting with lowercase.

Eg: dog
ab-c-321

2) Variables:-

Strings of letters, digits and underscore, starting with capital letter or an underscore.

Eg:- X
Apple-Wood.

3. Compound terms:-

Made up of a PROLOG atoms and a number of arguments enclosed in a parentheses and separated by a comma.

Eg:- is_bigger(elephant, X)
f(g(X1-), 7)

4) facts:-

predicate. follow by a dot.

Eg:- bigger_animal(X) :-
tall_is_beautiful.

5. Rules:-

consists of head & body

eg: is_small(X, Y) :- is_bigger(Y, X).

KB1

Women(mia)

Woman(jody)

Woman(yolanda).

play Air Guitar(jody)

Party.

Query 1: ?. women(mia) True

Query 2: ?. play Air Guitar(mia) False

Query 3: ? party True.

Query 4 ? Root consent procedure doesn't exist.

KB2:

woman(yolanda)

listens 2 Music(mia)

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

play Air Guitar(mia) :- listen2music(mia)

play Air Guitar(yolanda) :- listen music(yolanda)

Query1: ?- plays Air guitar(mia.) true

Query2: ?: plays Air guitar(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)

Query 1 : ? likes(dan, X).

Y sally.

Query 2: married(dans, sally). false

Query 3: married(john, brittany) false

EB4

food (burger)
food (sandwich)
food (pizza)
lunch (sandwich)
dinner (pizza)
meal (x) : food (x)

Query 1 : ? meal (x), lunch (x)
x : sandwich

Query 2 : ? food (pizza) dinner

Query 3 : ? dinner (sandwich) fail.

X 86

owns (jack, car (bmw)).
owns (john, car (chevy))
owns (olivia, car (civic))
owns (jane, car (chevy))
rich (car (audi))
rich (car (bmw))

Query 1 : ? owns (john, x)
x = car (chevy)

Query 2 : ? owns (john, _)
true.

Query 3 : ? owns (who, car (chevy))
who = john

Query 4 : ! owns (jane, x), rich (x) fail

Result

Thus the prolog programs are executed successfully.

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)

parent of (chris, peter)
parent of (chris, betty)
parent of (helen, peter)
parent of (helen, betty)
```

parent of (steven, charles)

parent of (steven, diana)

parent of (jerry, john)

parent of (jerry, helen)

/* RULES : */

/* son, parent

* son, grandparent */

father (x, y) :- male (y), parent (x, y)

OUTPUT

x = chen

y = peter

mother (x, y) :- female (y), parent of (x, y)

OUTPUT :-

x = chris

y = betty

grandfather (x, y) :- male (y), parent of (x, z), parent of (z,)

OUTPUT

x = kevin

y = pete

z = chris

grandmother(x, y) :- female(x), parent of (x, z) parent of(z, y).

OUTPUT :-

X = keren

Y = kelly

Z = ann

brother (x, y) :- male(y), father (x, z), father (x, w) :-

OUTPUT :-

procedure :- father (A, B) does not exist

sister (x, y) :- female(y), father(x, z), father(y, w)

z = = w)

OUTPUT :-

procedure 'father(A, B)' does not exist;

OUTPUT :-

This the program was executed successfully and o/p was verified.