INDEX

Name C. Lakshmanarao Class CSE Div. 'A' Roll No. 20701A0517

School Name... PRAI - abbreviation Sub. 3rd Year

S. No.	Date	Particular	~~Page No~~ marks	Signature
~~Exno.~~ 1	31/7/24	Sample programs of python using google colab	10	
2.	7/08/24	Implement image classification abstraction	10	
3.	7/09/24	N-queens problem	10	
4.	11/9/24	A* algorithm	10	
5.	18/9/24	Depth first search	10	
6.	18/9/24	AO* algorithm	10	
7.	25/9/24	Decision tree	10	
8.	9/10/24	K-means	10	
9.	16/10/24	Artificial neural network	10	
10.	23/10/24	Min-Max	10	
11.	30/10/24	Introduction to Prolog	10	
12.	6/11/24	Prolog - family - tree	10	

completed

udemy

1) prime number

```
num = int(input("Enter a number: "))
flag = False
if num == 1:
    print(num, "is not a prime number")
elif num > 1:
    for i in range(2, num):
        if (num % i) == 0:
            flag = True
            break
    if flag:
        print(num, "is not a prime number")
    else:
        print(num, "is a prime number")

else
    print(num, "is not a valid number to
                check for prime")
```

output:

Enter a number: 10

10 is not a prime number

2) palindrome

```
word = input()
if word == word[::-1]:
    print(word, "is a palindrome")
else:
    print(word, "is not a palindrome")
```

output:

apple
apple is not a palindrome

3) Fibonacci series

```
def fibonacci(n):
    fib_sequence = [0,1]
    for i in range(2,n):
        next_number = fib_sequence[-1]
                      fib_sequence[-2]
        fib_sequence.append(next_numb
    return fib_sequence
n = int(input("Enter the number:"))
print(fibonacci(n))
```

output:

Enter the number: 12

[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

4) Factorial number

```
def factorial(n):
    if n==0:
        return 1
    else
        return n * factorial(n-1)
number = int(input("Enter a number:"))
print(f'Factorial of {number} is
{factorial(number)}.')
```

output:
Enter a number: 5
Factorial of 5 is 120.

5) Bubble Sort

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
numbers = list(map(int, input("Enter the numbers separated by spaces:").split()))
sorted_numbers = bubble_sort(numbers)
print("Sorted numbers:", sorted_numbers)
```

output:
Enter the numbers separated by spaces: 5 1 12 10
Sorted numbers: 1 5 10 12

6) Armstrong number:

```
num = int(input("Enter a number:"))
sum = 0
temp = num
while temp > 0:
    digit = temp % 10
    sum += digit ** 3
    temp //= 10
if num == sum:
    print(num, "is an Armstrong number")
else:
    print(num, "is not an Armstrong number")
```

output:-

Enter a number: 153
153 is an Armstrong number

7) Sum of n numbers:

```
num = int(input("Enter a number:"))
if num > 0:
    print("Enter a positive number")
else:
    sum = 0
    while (num > 0):
        sum += num
        num -= 1
    print("the sum is", sum)
```

output:
Enter a number: 10
the sum is 55

8) largest of three numbers

```
num1 = int(input("Enter a number1:"))
num2 = int(input("Enter a number2:"))
num3 = int(input("Enter a number3:"))
if (num1 >= num2) and (num1 >= num3):
    largest = num1
elif (num2 >= num1) and (num2 >= num3):
    largest = num2
else:
    largest = num3
print("the largest number is", largest)
```

output:
Enter a number1: 10
Enter a number2: 12
Enter a Number3: 13
the largest number is 13

9) Swapping of two numbers

```
x = input('Enter value of x:')
y = input('Enter value of y:')
temp = x
x = y
y = temp
```

print("the value of x after swapping:{x}".format(x))
print("the value of y after swapping:{y}".format(y))

output:
Enter value of x:12
Enter value of y:10
the value of x after swapping:10
the value of y after swapping:12

10) To find the LCM of the number

```
def compute_lcm(x,y):
    if x>y:
        greater=x
    else:
        greater=y
    while(True):
        if((greater%x==0) and (greater%y==0)):
            lcm=greater
            break
        greater+=1
    return lcm
num1=int(input("Enter a number1:"))
num2=int(input("Enter a number2:"))
print("The L.C.M is",compute_lcm(num1,num2))
```

output:
Enter a Number1: 5
Enter a Number2: 10
The LCM is 10

31/07/2024

Dog Sentiment Image classification object

Domain:-
The Dog Sentiment Image classification Project sits at the intersection of computer vision and animal behaviour analysis, utilizing machine learning techniques to analyse and interpret visual data for understanding the emotional states of dogs. This interdisciplinary approach merges cutting-edge advancements in artificial intelligence with the principles of animal psychology, aimal to enhance human-animal interactions by providing deeper insights into canine emotions. By accurately classifying dog sentiments through image analysis, the project seeks to improve the well-being of dogs and foster stronger bonds between dogs and their owners.

objective:

The primary objective of the Dog Sentiment Image classification Project is to develop a machine learning model capable of accurately identifying and categorizing the emotional states of dogs based on their images. By analyzing dog's facial expressions and body language, the model aims to classify sentiments such as happiness, sadness, anger and neutrality. This advanced tool will enable pet owners, veterinarians, and animal behaviorists to gain a deeper understanding of dog's emotional needs, fostering better responses and interactions that enhance the overall well-being of dogs.

Target people:

The target audience for this Project includes:

1. Pet owners: To help them understand their pet's emotion and improve their well-being
2. Veterinarians: To assist in diagnosing and treating behavioral issues in dogs
3. Animal Behaviorists: To provide insights into canine emotions and behaviors for research and training purposes.

4. Animal Shelters and Rescue: To better organizations care for and rehabilitate dogs by understanding their emotional states.
5. Developers of pet-related products and services: To integrate sentiment analysis features into their offerings.

Algorithm:

The project employs a Convolutional Neural Network (CNN) for image classification. The chosen model architecture involves:

* Pre-trained Models: Utilizing models such as VGG16, ResNet50, or MobileNet, which have been pre-trained on large image datasets like ImageNet, and fine-tuning them for specific task of dog sentiment classification.
* Transfer Learning: Leveraging the feature extraction capabilities of pre-trained and retraining the final layers on the dog sentiment dataset

Data set:-
The data set for this project includes images of dogs labeled with their corresponding emotional states. Key aspects of the dataset are:

* Sources: Images collected from public datasets, social media, pet photography websites, and user submissions.
* Labeling: Experts annotators label the images with sentiment categories such as happy, sad, angry and neutral.

* Preprocessing: Images are resized, normalized, and augmented to ensure uniformity and increase variability
* Size and Diversity: A diverse and comprehensive dataset covering different breeds, ages and environments to ensure robustness and generalizability of the model.

4/9/24 n-queens

Program:-

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

def solve_n_queens(board, col):
    if col >= len(board):
        return True
```

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

        if solve_n_queens(board, col+1):
            return True

        board[i][col] = 0

return False

def print_board(board):
    for row in board:
        print(" ".join(str(x) for x in row))

def solve():
    n = 4
    board = [[0 for _ in range(n)] for _ in range(n)]

    if not solve_n_queens(board, 0):
        print("Solution does not exist")
        return False

    print_board(board)
    return True

solve()
```

Output:-

```
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

True

Result:- Thus the program can be executed successfully

A* program

Program:-

```
def astar(graph, start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()

    g = {}
    parents = {}
    g[start_node] = 0
    parents[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):
                n = v
        if n == stop_node or Graph_nodes[n]
                                    == none:
            pass
        else:
            for (m, weight) in get_neighbours(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 parents[n]!=n:
        path.append(n)
        n=parents[n]

    path.append(start_node)

    path.reverse()

    print("path found:{}".format(path))
    return path

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:
        return none

def heuristic(n):
    H_dist={
        'A': 11,
        'B': 6,
```

```
    'c' : 99,
    'D' : 1,
    'E' : 7,
    'G' : 0,
}
return H_distSnJ

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

output:

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

Result:-

Thus the program can be executed successfully.

Program:

```
def a
  a
  a
def
```

Depth First Search 18/09/24

Program:

```
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

Result:

Thus the program executed Successfully.

Exno: 7 Implementation of Decision Tree classification Techniques
Date: 25/9/24

Aim:
To implement a decision tree classification technique for gender classification using Python.

Explanation:
(i) Import tree from Sklearn
(ii) call the function DecisionTreeClassifier() from tree
(iii) Assign values for X and y
(iv) call the function predict on the basis of given random values for each given feature
(v) Display the output.

Program:

```
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, 150, 160,
175, 180, 190, 165], 'weight': [50, 60, 70, 80,
60, 75, 85, 105, 50], 'Age': [25, 30, 35, 40,
33, 37, 40, 21, 29], '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)
Y_pred = clf.predict(X_test)
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('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 Garden Classification')
plt.show()

output:

Enter height (in cm): 150
Enter weight (in kg): 50
Predicted garden for height 150.0cm and weight 50.0kg : Female

Result:

Thus the program can be executed successfully.

Exno: 8 Implementation of clustering
Date: 9/10/24 Techniques K-means

Aim:
To implement a K-means clustering technique using Python language.

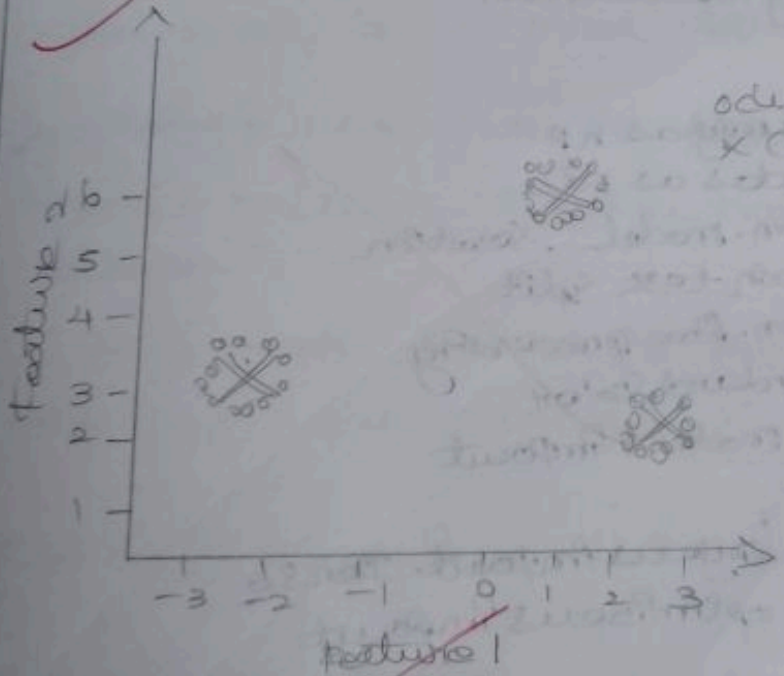
Explanation:
(i) Import kmeans from sklearn.cluster
(ii) Assign X and y
(iii) Call the function kmeans()
(iv) Perform scatter operation and display the output.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import
kmeans
from sklearn.datasets import
make_blobs
import kmeans from sklearn.cluster
X, y_true = make_blobs(n_samples=300,
centers=3, cluster_std=0.60,
random_state=0)
kmeans = KMeans(n_clusters=4, random_
                                state=0)
y_kmeans = kmeans.fit_predict(X)
```

```
plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X[:,1],
c=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',
label='Centroids')
plt.title('K-means clustering Results')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

output:



Result:
thus the program can be executed successfully.

Exno: 4 Implementing Artificial Neural networks for an Application using Python - Regression.
Date: 16/10/2024

Aim:
To implementing artificial neural network for an application in Regression using Python

Explanation:
(i) Import MLPRegressor from sklearn.neural network
(ii) Define the input features (X) and target values (y)
(iii) Initialize the MLPRegressor model
(iv) Fit the model to the data
(v) Predict values using the trained model
(vi) Display the output.

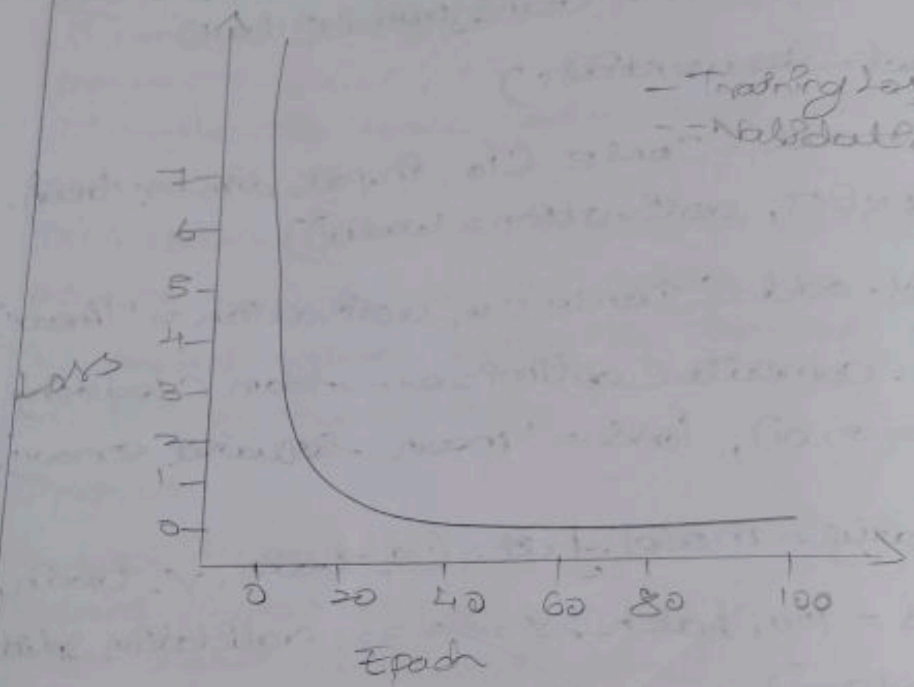
Program:

```
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(4,2)
X = np.random.rand(1000,3)
y = 3 * X[:, 0] + 2 * X[:, 1] **2
```

```
+ 1.5 * np.sin(X[:,0]) * np.pi) +
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 = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = Sequential()
model.add(Dense(10, input_dim=X_train.
shape[1], activation='relu'))
model.add(Dense(10, 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)
y_pred = model.predict(X_test)
mse = np.mean((y_test - y_pred.flatten())
                **2)
print(f'Mean Squared Error: {mse:.4f}')

plt.figure(figsize=(12,6))
plt.plot(history.history['loss'],
label='Training loss')
plt.plot(history.history['val_loss'],
label='validation loss')
```

plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

output:-



Result:

Thus the program can be executed successfully.

Exp No: Introduction to Prolog
Date: Monday

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.

Ex: dog
ab_c_321

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

Ex: Dog
Apple_1000

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

Ex: is_bigger(elephant,x)
flag(X,_),f)

4. Facts:-
A facts is a predicate followed by a dot

Ex:
bigger_animal(cobra).
life_is_beautiful.

5. Rules:-
A rule consists of a head and a body

Ex:
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).
plays_air_guitar(jody).
party.
query1 : ?- woman(mia).
query2 : ?- plays_air_guitar(mia).
query3 : ?- party.
query4 : ?- concert.

output:
?- woman(mia).
true
?- plays_air_guitar(mia).
false

?- party.
true.
?- concert.
Error: Unknown

KB2:
happy(yolanda).
listens_to_music(mia).
listens_to_music(yolanda).
plays_air_guitar(mia).
plays_air_guitar(yolanda).

output:
?- plays_air_guitar(mia).
true
?- plays_air_guitar(yolanda).
true

KB3

?- party
true.
?- carcard
Error: Unknown Procedure: carcard/0 (DWIM
could not correct goal)

KB1:

happy(yolanda).
listens_music(mia).
listens_music(yolanda) :- happy(yolanda).
plays_air_guitar(mia) :- listens_music(mia).
plays_air_guitar(yolanda) :- listens_music(yolanda).

output:
?- plays_air_guitar(mia)
true
?- plays_air_guitar(yolanda)
true.

KB2:

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

?- traveled(John, Scattergy)
false

KB1:
food(burger).
food(sandwich).
food(pizza).
lunch(sandwich).
dinner(pizza).
meal(x) :- food(x).

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.

Result:
thus the programs can be executed successfully

Exno: 12
Date: 6/11/24

Prolog – Family Tree

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

Source code:
Knowledge Base:

```
male(peter).
male(john).
male(charles).
male(robin).

female(betty).
female(jony).
female(lisa).
female(helen).

parentof(charles,peter).
parentof(charles,betty).
parentof(helen,peter).
parentof(helen,betty).
parentof(robin,charles).
parentof(robin,lisa).
parentof(jony,john).
parentof(jony,helen).

father(X,Y):- male(X), parentof(X,Y).
```

output:-

X = charles,
Y = peter

mother(x,y) :- female(y), parentof(x,y).

output:

X = charles,
Y = betty

grandfather(x,y) :- male(y), parentof(x,z),
parentof(z,y).

output:

X = kevin,
Y = peter,
Z = charles

granddaughter(x,y) :- female(y), parentof(x,z),
parentof(z,y).

output:

X = kevin,
Y = betty,
Z = charles

brother(x,y) :- male(y), father(x,z), father(y,z),
x \= y

output:

Procedure 'father(A,B)' does not exist.

Result:

Thus the program executed successfully.

EXNO: 10
Date: 23/10/24

Min- Max Tree

Aim:
To write a Min-max tree and execute its output.

program:

```
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.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)

Printf("the optimal score is: %d", optimal_score);

output:

the optimal score is: 5

Result:

thus the program can be executed successfully and output verified.

EXPNO: 6
Date: 18/9/24

AO* Algorithm

Aim:

To write the AO* algorithm and executes with output.

Program:

```
class Graph:
    def __init__(self, graph, heuristic):
        self.graph = graph
        self.heuristic = heuristic
        self.solutions = {}

    def ao_star(self, node):
        print(f"Expanding: {node}")

        if node not in self.graph or not
                                self.graph[node]:
            return

        children = self.graph[node]
        best_path = None
        min_cost = float('inf')

        for group in children:
            cost = sum(self.heuristic[child] for
                        child in group)
            if cost < min_cost:
                min_cost = cost
                best_path = group

        self.solutions[node] = best_path
        print(f"Best path for {node}: {best_path}
                with cost {min_cost}")
```

```
        for child in best_path:
            self.ao_star(child)
    def get_solution(self):
        return self.solution

graph = {
    'A': [['B', 'C'], ['D']],
    'B': [['E']],
    'C': [['G']],
    'D': [['H']],
    'E': [],
    'G': [],
    'H': []
}
heuristic = {'A': 0, 'B': 1, 'C': 2, 'D': 4, 'E': 1, 'G': 2, 'H': 5}

graph_obj = Graph(graph, heuristic)
graph_obj.ao_star('A')
solution = graph_obj.get_solution()
print("Solution:", solution)
```

Output:

Expanding: A

Best path for A: ['B', 'C'] with cost 3

Expanding: B

Best path for B: ['E'] with cost 1

Expanding: E

Expanding: C

Best path for C: ['G'] with cost 3.

Result:

Thus the Program executed successfully and output verified.