2318124

WATER JUG USING DFS.

```
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 (x, y).
    def empty_3_gallon (x, y, x_max, y_max):
    return (x, 0).
    def pour_4to3_gallon (x, y, x_max, y_max)
    transfer = min (x, y_max - y).
    return (x - transfer, y + transfer)
    def pour_3to4 - (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:
    visite = set()
    stack = [start]
    while stack:
        state = stack.pop()
        x, y = state
        if state in visited:
            continue
        visited.add (state)
    print (f "visited state : {state}")
    if x = goal_x:
        print (f ("Goal reached")
```

```
next_states =
[ fill_4_gallon (x, y, x_max, y_max),
  fill_3_gallon (x, y, x_max, y_max),
  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 newstate not in visited:
        stack.append (new_state)
return None.
```

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)
(2,3).
```

RESULT:

Thus the water jug problem using DFS is executed successfully.

30/8/24

Implement A* algorithm.

CODE:

```
def astaralgo (start_node, stop_node):
    open_set = set (start_node)
    closed_set = set()
    g = {}
    parents = {}
    while len (open_set > 0):
        n = None
    for v in openset:
        if n = none or g[v] + heuristic [v] <
            g[n] + heuristic (n):
        n = v.
    If n == stop_node or graph_nodes[n] == Non
    pass.
    else
if g [m] > g[n] + weight :
    g[m] = g[n] + weight
    parents[m] = n
if m in closedset:
    closed_set_remove (m)
    open_set . add (n)
if n == None :
    print ("path does not exists");
    return none.
if n == stop node:
    path []
while parents [n] ! = n:
    path . append (n)
    n = parents [n]
path . reverse()
return path
```

```
open.set remove (n)
closed.setadd (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' = 7,
        'E' = 7
        'G' = 0 }

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

astarlo ('A', G)
```

o/p

RESULT:

Thus the program is verified and executed successfully.

27/9/24

ARTIFICIAL NEURAL NETWORKS.

AIM:

TO implement ANN for an application in regardless using python.

CODE:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import standard scaler
from keras model import sequential.
from keras.layers import Dense
from keras.optimizer import Adam
import mat plotlib pyplot as plt.

np.random.seed(42)
X = np.random.rand (1000, 3)
y = 3 * X[:, 0] + 2 * X[:, 1] ** 2 + 1.5 *
np.sin [X[:, 2] * np : p');
    + 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 = standard scaler()
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 = 'relu'))
```

```
mse = np.mean(y_test - y_pred.flatten())**2)
print(f"Mean squared Error = {mse:4f}")

plt.figure(figsize=(12,6))
plt.plot(history['loss'], label='training loss')
plt.plot(history['val_loss'], label='validation loss')
plt.title('training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

RESULT:

Thus the program to implement ANN is done successfully and output is verified.

6/9/24

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.metrics import accuracy_score,
classification_report
confusion_matrix
import matplotlib.pyplot as plt

data = {
'Height': [150, 160, 170, 180, 155, 165, 175, 185]
'weight': [50, 60, 70, 80, 55, 65, 75, 85, 45]
'Age': [25, 30, 35, 40, 28, 32, 37, 42, 24]
'Gender': ['Female', 'Female', 'Male', 'Male',
'Female', 'Female', 'Male', 'Male'].
df = pd.DataFrame(data)
df['Gender'] = df['Gender'].map({
'Female': 0, 'Male': 1}).
x = df[['Gender']]
y = df[['Height', 'weight', 'Age']]
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', classport)
Print (Plt.figure),
```

RESULT:

Thus the program to implement decision tree classification. is done successfully and output is verified.

4/10/24

K-MEANS.

Aim:

To implement a kmeans clustering tech using python language.

PROGRAM:

```
import numpy as np.
import matplotlib.pyplot as plt
from sklearn.cluster import K.Means
from sklearn.dataset import make_blob
X-y-true = make_blobs (n_samples=300,
centers = 3, cluster_std = 0.60,
random_state = 0).

K=3
kmean = kmeans (n_cluster = K, random_state=0)
y_kmeans = kmeans.fit_predict (X).

plt.scatter
plt.title ('K-means clustering Results')
plt.xlabel ('Feature1')
plt.ylabel ('Feature2')
plt.legend ()
plt.show ().
```

O/P

RESULT:

Thus the program is done successfully and output is verified.

9/8/24

N QUEEN'S PROBLEM

Aim:

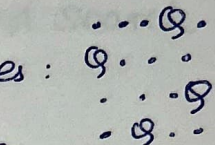
To write a program to solve 8 QUEEN'S PROBLEM.

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

def cd_clashes(bs, c):
    for i in range(c):
        if share_digonal(i, bs[i], c, bs[c]):
            return true:
    return false.

def has_dashes(the_board):
    for col in range(1, len(the_board)):
        if col_dashes(board, col):
            return true.
    return false

def main(E):
    import random
    rng = random.Random()
    bal = list(range(8))
    num_found = 0.
    tries = 0.
    result = []
    while num_found < 10:
        rang.shuffled
        tries += 1
        if not has_desked(bd) and.

    Print(result)

main()
```

Output:

4

found solution [1,3,0,2] in 4 tries:



found solution [2,0,3,1] in 4 tries.

8

found solution [2,5,1,4,7,0,6,3] in 115 tries

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

found solution [3,1,6,4,0,7,5,2] in 480 tries

found solution [5,2,0,7,4,1,3,6] in 217 tries

found solution [2,6,1,7,5,3,0,4] in 1847 tries

found solution [3,6,2,0,5,7,1,4] in 96 tries.

RESULT:

Thus the 8 Queen's problem has been executed successfully.

16/8/24

DFS.

AIM:

TO solve problem using Depth first search

CODE:

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

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

OUTPUT:

A B C D E F C

RESULT:

Thus Dfs program is executed successfully.

SOURCE CODE :

KB1

like (dan, sally)

like (sally, dan)

likes (john, birthney)

married (x,y) : likes (x,y), likes (y,x)

friends (x,y) : likes (x,y) : likes (y,y)

query 1 : ? likes (dan, x)

Query 2 : ? playcenter (trias) - false

Query 3 ? party - True.

KB5 :

owns (jack, car (bmw)).

owns (john, car (chevy))

owns (olivia, car (civic))

sedan (car (bmw)

sedan car (civic))

truck (can (chevy)).

Query 1 ? : own (john, X)

X = car (chevy)

Query 2 ? owns (john).

true

Query 3 ? owns (who can (chevy))

who = John.

RESULT :

Thus the Prolog Program are executed successfully.

8/11/24

PROLOG FAMILY TREE

AIM:

To develop a family tree program using Prolog.

Knowledge Base:

```
male (peter)
male (John)
male (chris)
male (helvin)
female (betty)
female (jenny)
female (lisa)
female (helen)

parentof (chris, Peter)
parentof (chris, betty)
parentof (helen, Peter)
parentof (helen, betty).
parentof (kelvin, chris)
parentof (kelvin, lisa)
parentof (lisa, John)
parentof (jeny, kelen)
```

*/ Rules */

a) father(x,y) = male(x) parentof(x,y)

x	y
Peter	chris
Peter	Helen
John	jeny
chris	Kevin.

b) female(Y) parentof(X,Y)

betty	chris
betty	Helen
Lisa	kelvin
helen	Jeny.

c) male(Y), parentof(X,Z), parentof(Z,Y)

Y	X	Z
Peter	kelvin	chris
Peter	Jeny	Helen.

d) female(Y) parent(X,Z) parentof(Z,Y).

Y	X	Z
betty	kelvin	chris
betty	Jeny	Helen.

RESULT:

Thus the Prolog family tree is executed successfully.

18/10/24

MINMAX ALGORITHM.

Aim:

To write a simple minmax algorithm.

PROGRAM:

```
import math
def minmax (depth, node_index, is_maximized,
    scores, height).
 if depth == height
        return scores [node_index]
    if is_maximized:
        return max (minmax (depth+1,
nodeindex * 2, false, scores, height),
minmax (depth+1, node_index * 2+1, false score,
 height)).
    else:
      return min (minimax (depth+1, node-2,
false, score, height),
minimax (depth+1, node_index * 2+1,
false, scores, height))
```

```
def calculate_tree_height(num_leaves):
    return math.ceil(math.log2(num_leaves))
scores = list(map(int, input("Enter the
score seperated by spaces: ").split()))
print("The optimal score is: {optimal
    score}").
```

OUTPUT:

Enter the score seperated by spaces

-1 4 2 6 -3 -5 0 7

The optimal score is 4.

RESULT:

Thus the minimax algorithm successfully determined the optimal move for players.