**Artificial Intelligence Practical**

**Practical no:-01**

**Aim:**-Write a Program to ImplementBreadth first search algorithm.

**Input:-**

from collections import deque

class Node:

def \_\_init\_\_(self,info):

self.info = info

self.left = None

self.right = None

self.level = None

def \_\_str\_\_(self):

returnstr(self.info)

def BFT(node):

node.level = 1

queue = deque([node])

output = []

current\_level = node.level

whilelen(queue)>0:

current\_node = queue.popleft()

if(current\_node.level>current\_level):

output.append("\n")

current\_level += 1

output.append(str(current\_node))

ifcurrent\_node.left != None:

current\_node.left.level = current\_level + 1

queue.append(current\_node.left)

ifcurrent\_node.right != None:

current\_node.right.level = current\_level + 1

queue.append(current\_node.right)

return ''.join(output)

root = Node(9)

root.left = Node(2)

root.right = Node(4)

root.left.left = Node(1)

root.left.right = Node(3)

root.right.left = Node(5)

root.right.right = Node(7)

print(BFT(root))

**Output:-**

>>>

9

24

1357

>>>

**Practical no:-02**

**Aim**:-Write a program to Implement depth first search.

**Input:-**

graph = {'A': set(['B','C']),

'B': set(['A','D','E']),

'C': set(['A','F']),

'D': set(['B']),

'E': set(['B','F']),

'F': set(['C','E'])}

defdfs(graph,start):

visited,stack = set(),[start]

while stack:

vertex = stack.pop()

if vertex not in visited:

visited.add(vertex)

stack.extend(graph[vertex]-visited)

return visited

print(dfs(graph,'A'))

defdfs\_paths(graph,start,goal,path=None):

if path is None:

path = [start]

if start == goal:

yield path

for next in graph[start]-set(path):

yield from dfs\_paths(graph,next,goal,path+[next])

print(list(dfs\_paths(graph,'C','F')))

**Output:-**

>>>

{'C', 'D', 'A', 'F', 'E', 'B'}

[['C', 'A', 'B', 'E', 'F'], ['C', 'F']]

>>>

**Practical no:-03 and 04**

**Aim:-**write a program to ImplementA\* search algorithm and best-ﬁrst search algorithm.

**Input:**-

fromsimpleai.search import SearchProblem, astar

GOAL= 'HELLO WORLD'

classHelloProblem(SearchProblem):

def actions(self,state):

iflen(state) <len(GOAL):

return list(' ABCDEFGHIJKLMNOPQRSTUVWXYZ')

else:

return []

def result(self,state,action):

return state + action

defis\_goal(self,state):

return state == GOAL

def heuristic(self,state):

wrong = sum([1 if state[i] != GOAL[i] else 0

for i in range(len(state))])

missing = len(GOAL) - len(state)

return wrong + missing

problem = HelloProblem(initial\_state='')

result =astar(problem)

print(result.state)

print(result.path())

**Output:-**

>>>

HELLO WORLD

[(None, ''), ('H', 'H'), ('E', 'HE'), ('L', 'HEL'), ('L', 'HELL'), ('O', 'HELLO'), (' ', 'HELLO '), ('W', 'HELLO W'), ('O', 'HELLO WO'), ('R', 'HELLO WOR'), ('L', 'HELLO WORL'), ('D', 'HELLO WORLD')]

>>>

**Practical no:-05**

**Aim:-** write a program for decision tree learning algorithm.

**Input:-**

#The program doesnt construct a decision tree, it consructs datasets and results of branches using dictionary

import pandas as pd

data = pd.DataFrame({"toothed":["True","True","True","False","True","True","True","True","True","False"],

"hair":["True","True","False","True","True","True","False","False","True","False"],

"breathes":["True","True","True","True","True","True","False","True","True","True"],

"legs":["True","True","False","True","True","True","False","False","True","True"],

"species":["Mammal","Mammal","Reptile","Mammal","Mammal","Mammal","Reptile","Reptile","Mammal","Reptile"]},

columns=["toothed","hair","breathes","legs","species"])

features = data[["toothed","hair","breathes","legs"]]

target = data["species"]

print(data)

**Output:-**

>>>

toothed hair breathes legs species

0 True True True True Mammal

1 True True True True Mammal

2 True False True False Reptile

3 False True True True Mammal

4 True True True True Mammal

5 True True True True Mammal

6 True False False False Reptile

7 True False True False Reptile

8 True True True True Mammal

9 False False True True Reptile>>>

**Practical no.06**

**Aim**:-write a program for Implement Naive Bayes’ learning algorithm.

**Input:-**

# https://www.geeksforgeeks.org/naive-bayes-classifiers/

classNaiveBayes:

def \_\_init\_\_(self, f, r):

self.features = f

self.response = r

def predict(self,custcase):

anskeys = list(self.response.keys())

ansvalues = dict.fromkeys(anskeys,0)

#print(custcase)

for key in anskeys :

ansvalues[key] = self.response[key]

forikey, ival in custcase.items() :

ansvalues[key] = ansvalues[key] \* self.features[ikey][ival][key]

print(ansvalues)

#calculating MAP

maxkey=""

maxans=-1

forikey, ival in ansvalues.items():

ifival>maxans :

maxans= ival

maxkey = ikey

returnmaxkey

#precalculated values from worksheet - "naive bayes classifier working"

response = {"Wait":0.4, "Leave":0.6}

features = {

"Reservation":

{

"Yes" : {"Wait":0.5, "Leave":0.666667},

"No" : {"Wait":0.5, "Leave":0.333333}

} ,

"Time>30":

{

"Yes" : {"Wait":0.25, "Leave":0.83333},

"No" : {"Wait":0.75, "Leave":0.16667}

}

}

nb = NaiveBayes(features, response)

#print("Probability :", nb.features["Reservation"]["Yes"]["Wait"])

#print("Probability :", nb.features["Time>30"]["No"]["Leave"])

resstatus = input("Manager asks Customer, have you reserved table?(Yes/No):")

timestatus = input("Customer asks Manager, Will it take more than 30 mins?(Yes/No):")

custcase = {"Reservation":resstatus, "Time>30":timestatus}

print("Manager predicts that Customer will :" , nb.predict(custcase) )

**Output:-**

>>>

Manager asks Customer, have you reserved table?(Yes/No):Yes

Customer asks Manager, Will it take more than 30 mins?(Yes/No):Yes

{'Wait': 0.05, 'Leave': 0.33333216666599996}

Manager predicts that Customer will : Leave

>>>

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Reservation | Time>30 | Result |
| 1 | Yes | No | Wait |
| 2 | Yes | Yes | Leave |
| 3 | No | No | Wait |
| 4 | Yes | No | Wait |
| 5 | No | Yes | Leave |
| 6 | No | Yes | Wait |
| 7 | No | Yes | Leave |
| 8 | Yes | No | Leave |
| 9 | Yes | Yes | Leave |
| 10 | Yes | Yes | Leave |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Result | Ans | Value | Probability |  |  |
|  | Wait | 4 | 0.4 |  |  |
|  | Leave | 6 | 0.6 |  |  |
|  | Total | 10 |  |  |  |
|  |  |  |  |  |  |
| Reservation | Ans | Wait | Leave | Probability(Wait) | Probability(Leave) |
|  | Yes | 2 | 4 | (2/4)=0.5 | 0.666666667 |
|  | No | 2 | 2 | (2/4)=0.5 | 0.333333333 |
|  | Total | 4 | 6 | 1 | 1 |
|  |  |  |  |  |  |
| Time>30 | Ans | Wait | Leave | Probability(Wait) | Probability(Leave) |
|  | Yes | 1 | 5 | (1/4)=0.25 | 0.833333333 |
|  | No | 3 | 1 | (3/4)=0.75 | 0.166666667 |
|  | Total | 4 | 6 | 1 | 1 |