AI – 1.1 BFS Undirected Graph

Code

# BFS algorithm in Python import collections

# BFS algorithm

def bfs(graph, root): visited, queue = set(), collections.deque([root]) visited.add(root)

while queue:

# Dequeue a vertex from queue vertex = queue.popleft() print(str(vertex) + " ", end="")

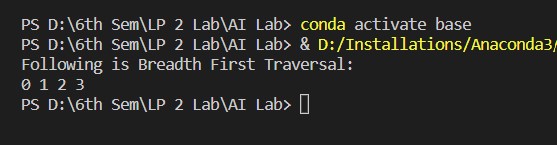
# If not visited, mark it as visited, and

# enqueue it for neighbour in graph[vertex]: if neighbour not in visited: visited.add(neighbour) queue.append(neighbour)

if \_\_name\_\_ == '\_\_main\_\_':

graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]} print("Following is Breadth First Traversal: ") bfs(graph, 0)

Output



AI- 1.2 DFS Undirected Graph

Code

# DFS algorithm in Python

# DFS algorithm def dfs(graph, start, visited=None): if visited is None: visited = set() visited.add(start) print(start)

for next in graph[start] - visited:

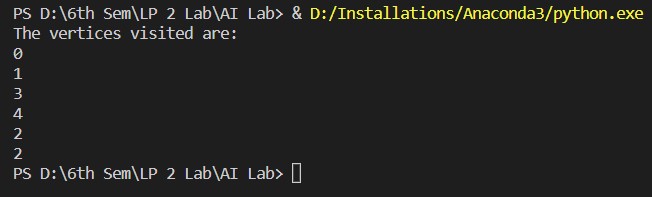
dfs(graph, next, visited) return visited

graph = {'0': set(['1', '2']), '1': set(['0', '3', '4']),

'2': set(['0']),

'3': set(['1']), '4': set(['2', '3'])} dfs(graph, '0')

Output



AI- 2 A Star algorithm Code

from collections import deque class Graph:

# example of adjacency list (or rather map) def \_\_init\_\_(self, adjacency\_list):

self.adjacency\_list = adjacency\_list

def get\_neighbors(self, v):

return self.adjacency\_list[v]

# heuristic function with equal values for all nodes def h(self, n): H = {

'A': 1,

'B': 1,

'C': 1,

'D': 1

} return H[n]

def a\_star\_algorithm(self, start\_node, stop\_node):

# open\_list is a list of nodes which have been visited, but who's neighbors haven't all been inspected, starts off with the start node

# closed\_list is a list of nodes which have been visited and who's neighbors have been inspected open\_list = set([start\_node]) closed\_list = set([])

# g contains current distances from start\_node to all other nodes the default value (if it's not found in the map) is +infinity g = {} g[start\_node] = 0

# parents contains an adjacency map of all nodes parents = {} parents[start\_node] = start\_node while len(open\_list) > 0:

n = None

# find a node with the lowest value of f() - evaluation function for v in open\_list: if n == None or g[v] + self.h(v) < g[n] + self.h(n):

n = v;

if n == None: print('Path does not exist!') return None

# if the current node is the stop\_node

# then we begin reconstructin the path from it to the start\_node if n == stop\_node: reconst\_path = [] while parents[n] != n:

reconst\_path.append(n) n = parents[n] reconst\_path.append(start\_node) reconst\_path.reverse() print('Path found: {}'.format(reconst\_path)) return reconst\_path

# for all neighbors of the current node do for (m, weight) in self.get\_neighbors(n):

# if the current node isn't in both open\_list and closed\_list

# add it to open\_list and note n as it's parentif m not in open\_list and m not in closed\_list:

open\_list.add(m) parents[m] = n g[m] = g[n] + weight # otherwise, check if it's quicker to first visit n, then m and if it is, update parent data and g data and if the node was in the closed\_list, move it to open\_list else: if g[m] > g[n] + weight: g[m] = g[n] + weight parents[m] = n

if m in closed\_list: closed\_list.remove(m) open\_list.add(m)

# remove n from the open\_list, and add it to closed\_list because all of his neighbors were inspected open\_list.remove(n) closed\_list.add(n) print('Path does not exist!') return None

adjacency\_list = {

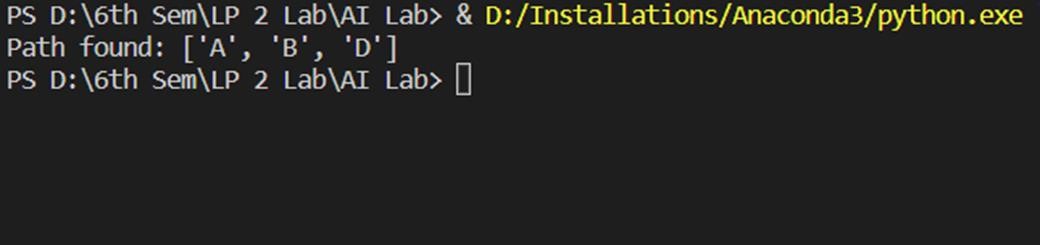
'A': [('B', 1), ('C', 3), ('D', 7)],

'B': [('D', 5)],

'C': [('D', 12)]

} graph1 = Graph(adjacency\_list) graph1.a\_star\_algorithm('A','D')

Output



AI- 3 Greedy Search Algorithm- Job Scheduling Problem Code

# Program to find the maximum profit

# job sequence from a given array

# of jobs with deadlines and profit

# function to schedule the jobs take 2 # arguments array and no of jobs to schedule def printJobScheduling(arr, t):

# length of array n = len(arr)

# Sort all jobs according to # decreasing order of profit for i in range(n): for j in range(n - 1 - i): if arr[j][2] < arr[j + 1][2]:

arr[j], arr[j + 1] = arr[j + 1], arr[j]

# To keep track of free time slots result = [False] \* t

# To store result (Sequence of jobs) job = ['-1'] \* t

# Iterate through all given jobs for i in range(len(arr)): # Find a free slot for this job

# (Note that we start from the # last possible slot) for j in range(min(t - 1, arr[i][1] - 1), -1, -1):

# Free slot found

if result[j] is False: result[j] = True job[j] = arr[i][0]

break

# print the sequence print(job)

# Driver COde arr = [['a', 2, 100], # Job Array

['b', 1, 19],

['c', 2, 27],

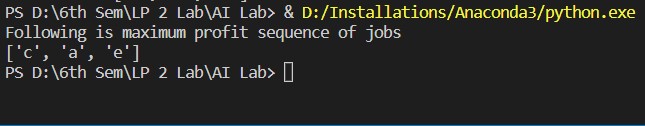
['d', 1, 25],

['e', 3, 15]]

print("Following is maximum profit sequence of jobs")

# Function Call printJobScheduling(arr, 3)

Output



AI- 4. N-queens problem Code

import io import random import string import warnings import numpy as np from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.metrics.pairwise import cosine\_similarity import warnings warnings.filterwarnings('ignore') import nltk from nltk.stem import WordNetLemmatizer

# nltk.download('popular', quiet=True)

# nltk.download('punkt')

# nltk.download('wordnet')

with open('chatbot.txt','r', encoding='utf8', errors ='ignore') as fin:

raw = fin.read().lower()

#Tokenisation sent\_tokens = nltk.sent\_tokenize(raw) word\_tokens = nltk.word\_tokenize(raw)

# Preprocessing lemmer = WordNetLemmatizer() def LemTokens(tokens):

return [lemmer.lemmatize(token) for token in tokens] remove\_punct\_dict = dict((ord(punct), None) for punct in string.punctuation) def LemNormalize(text):

return

LemTokens(nltk.word\_tokenize(text.lower().translate(remove\_punct\_dict))) # Keyword Matching

GREETING\_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey","Helo")

GREETING\_RESPONSES = ["hi", "hey", "hi there", "hello", "I am glad! You are talking to me"]

def greeting(sentence): for word in sentence.split(): if word.lower() in GREETING\_INPUTS:

return random.choice(GREETING\_RESPONSES)

def response(user\_response):

robo\_response=''

sent\_tokens.append(user\_response)

TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop\_words='english') tfidf = TfidfVec.fit\_transform(sent\_tokens) vals = cosine\_similarity(tfidf[-1], tfidf) idx=vals.argsort()[0][-2] flat = vals.flatten() flat.sort() req\_tfidf = flat[-2] if(req\_tfidf==0):

robo\_response=robo\_response+"I am sorry! I don't understand you" return robo\_response else:

robo\_response = robo\_response+sent\_tokens[idx] return robo\_response

flag=True

print("ROBO: My name is Robo. I will answer your queries about Investments. If you want to exit, type Bye!") while(flag==True):

user\_response = input()

user\_response=user\_response.lower() if(user\_response!='bye'): if(user\_response=='thanks' or user\_response=='thank you' ):

flag=False print("ROBO: You are welcome..") else: if(greeting(user\_response)!=None):

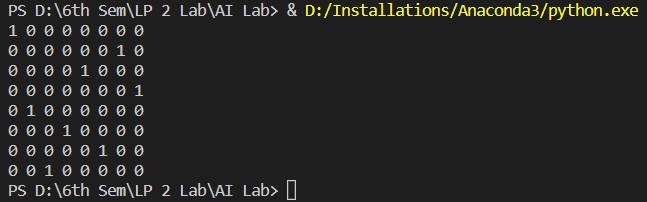
print("ROBO: "+greeting(user\_response)) else:

print("ROBO: ",end="") res = response(user\_response) nlines = res.count('\n') if nlines > 0:

res = res.split("\n",1)[1] print(res) sent\_tokens.remove(user\_response) else:

flag=False print("ROBO: Bye! take care..")

Output



AI-5 Chatbot Application in Python

Code

import io import random import string import warnings import numpy as np from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.metrics.pairwise import cosine\_similarity import warnings warnings.filterwarnings('ignore') import nltk from nltk.stem import WordNetLemmatizer

# nltk.download('popular', quiet=True)

# nltk.download('punkt')

# nltk.download('wordnet')

with open('chatbot.txt','r', encoding='utf8', errors ='ignore') as fin:

raw = fin.read().lower()

#Tokenisation sent\_tokens = nltk.sent\_tokenize(raw) word\_tokens = nltk.word\_tokenize(raw)

# Preprocessing lemmer = WordNetLemmatizer() def LemTokens(tokens):

return [lemmer.lemmatize(token) for token in tokens] remove\_punct\_dict = dict((ord(punct), None) for punct in string.punctuation) def LemNormalize(text):

return

LemTokens(nltk.word\_tokenize(text.lower().translate(remove\_punct\_dict))) # Keyword Matching

GREETING\_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey","Helo")

GREETING\_RESPONSES = ["hi", "hey", "hi there", "hello", "I am glad! You are talking to me"]

def greeting(sentence): for word in sentence.split(): if word.lower() in GREETING\_INPUTS:

return random.choice(GREETING\_RESPONSES)

def response(user\_response):

robo\_response=''

sent\_tokens.append(user\_response)

TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop\_words='english') tfidf = TfidfVec.fit\_transform(sent\_tokens) vals = cosine\_similarity(tfidf[-1], tfidf) idx=vals.argsort()[0][-2] flat = vals.flatten() flat.sort() req\_tfidf = flat[-2] if(req\_tfidf==0):

robo\_response=robo\_response+"I am sorry! I don't understand you" return robo\_response else:

robo\_response = robo\_response+sent\_tokens[idx] return robo\_response

flag=True

print("ROBO: My name is Robo. I will answer your queries about Investments. If you want to exit, type Bye!") while(flag==True):

user\_response = input() user\_response=user\_response.lower() if(user\_response!='bye'): if(user\_response=='thanks' or user\_response=='thank you' ):

flag=False print("ROBO: You are welcome..") else: if(greeting(user\_response)!=None):

print("ROBO: "+greeting(user\_response)) else:

print("ROBO: ",end="") res = response(user\_response) nlines = res.count('\n') if nlines > 0:

res = res.split("\n",1)[1] print(res) sent\_tokens.remove(user\_response) else:

flag=False print("ROBO: Bye! take care..") Output

