**ASSIGNMENT 1: BFS DFS**

#include<bits/stdc++.h>

using namespace std;

const int n = 1e5 + 2;

bool visited[n];

vector<int>adj[n]; //adjacency list

void dfs(int root)

{

visited[root] = 1;

cout<<root<<" ";

vector<int> :: iterator it;

for(it = adj[root].begin() ; it != adj[root].end(); it++)

{

if(visited[\*it])

{

//nothing

}

else

{

dfs(\*it);

}

}

}

void bfs(queue<int>&q)

{

if(q.empty())

{

return;

}

int node = q.front();

q.pop();

cout<<node<<" ";

vector<int> :: iterator it;

for(it = adj[node].begin() ; it != adj[node].end() ; it++)

{

if(!visited[\*it])

{

visited[\*it] = true;

q.push(\*it);

}

}

bfs(q);

}

int main()

{

int v , e;

cout<<"Enter the number of vertices and edges : "<<endl;

cin>>v>>e;

for(int i = 0 ;i<v ;i++)

{

visited[i] = false;

}

cout<<"Enter the source and destination of each edge : "<<endl;

int src ;

int dest;

for(int i = 0 ;i<e ;i++)

{

cin>>src>>dest;

adj[src].push\_back(dest);

adj[dest].push\_back(src);

}

cout<<"DFS traversal : "<<endl;

cout<<"Enter the root : ";

int root;

cin>>root;

dfs(root);

cout<<endl;

for(int i = 0 ;i<v ;i++)

{

visited[i] = false;

}

cout<<"BFS traversal : "<<endl;

queue<int>q;

for(int i = 0 ;i<v ;i++)

{

if(visited[i] == false )

{

visited[i] = true;

q.push(i);

bfs(q);

}

}

return 0;

}

**ASSIGNMENT 2: A\***

import copy

final = [[1, 2, 3], [4, 5, 6], [7, 8, -1]]

initial = [[1, 2, 3], [-1, 4, 6], [7, 5, 8]]

def gn(state, finalstate):

count = 0

for i in range(3):

for j in range(3):

if (state[i][j] != -1):

if (state[i][j] != finalstate[i][j]):

count += 1

return count

def findposofblank(state):

for i in range(3):

for j in range(3):

if (state[i][j] == -1):

return [i, j]

def move\_left(state, pos):

if (pos[1] == 0):

return None

retarr = copy.deepcopy(state)

retarr[pos[0]][pos[1]], retarr[pos[0]][pos[1] -

1] = retarr[pos[0]][pos[1]-1], retarr[pos[0]][pos[1]]

return retarr

def move\_up(state, pos):

if (pos[0] == 0):

return None

retarr = copy.deepcopy(state)

retarr[pos[0]][pos[1]], retarr[pos[0]-1][pos[1]

] = retarr[pos[0]-1][pos[1]], retarr[pos[0]][pos[1]]

return retarr

def move\_right(state, pos):

if (pos[1] == 2):

return None

retarr = copy.deepcopy(state)

# for i in state:

# retarr.append(i)

retarr[pos[0]][pos[1]], retarr[pos[0]][pos[1] +

1] = retarr[pos[0]][pos[1]+1], retarr[pos[0]][pos[1]]

return retarr

def move\_down(state, pos):

if (pos[0] == 2):

return None

retarr = copy.deepcopy(state)

retarr[pos[0]][pos[1]], retarr[pos[0]+1][pos[1]

] = retarr[pos[0]+1][pos[1]], retarr[pos[0]][pos[1]]

return retarr

def printMatrix(matricesArray):

print("")

counter = 1

for matrix in matricesArray:

print("Step {}".format(counter))

for row in matrix:

print(row)

counter += 1

print("")

def eightPuzzle(initialstate, finalstate):

hn = 0

explored = []

while (True):

explored.append(initialstate)

if (initialstate == finalstate):

break

hn += 1

left = move\_left(initialstate, findposofblank(initialstate))

right = move\_right(initialstate, findposofblank(initialstate))

up = move\_up(initialstate, findposofblank(initialstate))

down = move\_down(initialstate, findposofblank(initialstate))

fnl = 1000

fnr = 1000

fnu = 1000

fnd = 1000

if (left != None):

fnl = hn + gn(left, finalstate)

if (right != None):

fnr = hn + gn(right, finalstate)

if (up != None):

fnu = hn + gn(up, finalstate)

if (down != None):

fnd = hn + gn(down, finalstate)

minfn = min(fnl, fnr, fnu, fnd)

if ((fnl == minfn) and (left not in explored)):

initialstate = left

elif ((fnr == minfn) and (right not in explored)):

initialstate = right

elif ((fnu == minfn) and (up not in explored)):

initialstate = up

elif ((fnd == minfn) and (down not in explored)):

initialstate = down

printMatrix(explored)

def main():

start = [[1, 2, 3], [-1, 4, 6], [7, 5, 8]]

goal = [[1, 2, 3], [4, 5, 6], [7, 8, -1]]

eightPuzzle(start, goal)

main()

**ASSIGNMENT 3: Selection sort/ Kruskal MST**

// C++ program for implementation of

// selection sort

#include <bits/stdc++.h>

using namespace std;

// Function for Selection sort

void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

// One by one move boundary of

// unsorted subarray

for (i = 0; i < n - 1; i++) {

// Find the minimum element in

// unsorted array

min\_idx = i;

for (j = i + 1; j < n; j++) {

if (arr[j] < arr[min\_idx])

min\_idx = j;

}

// Swap the found minimum element

// with the first element

if (min\_idx != i)

swap(arr[min\_idx], arr[i]);

}

}

// Function to print an array

void printArray(int arr[], int size)

{

int i;

for (i = 0; i < size; i++) {

cout << arr[i] << " ";

cout << endl;

}

}

// Driver program

int main()

{

int arr[] = { 64, 25, 12, 22, 11 };

int n = sizeof(arr) / sizeof(arr[0]);

// Function Call

selectionSort(arr, n);

cout << "Sorted array: \n";

printArray(arr, n);

return 0;

}

#include<bits/stdc++.h>

using namespace std;

class DSU{

int \*parent;

int \*rank;

public:

DSU(int n )

{

parent = new int[n];

rank = new int[n]; //to check to which larger component the subset belongs to

for(int i = 0 ;i<n ;i++)

{

parent[i] = -1;

rank[i] = 1; //at first all the nodes are at the same level

}

}

int find(int i )

{

if(parent[i] == -1)

return i;

return parent[i] = find(parent[i]);

}

void unionp(int x , int y)

{

int a1 = find(x);

int a2 = find(y);

if(a1 != a2)

{

if(rank[a1]<rank[a2])

{

parent[a1] = a2;

}else if(rank[a1]>rank[a2])

{

parent[a2] = a1;

}

else

{

parent[a2] = a1;

rank[a1] +=1;

}

}

}

};

class Graph {

vector<vector<int> > edgelist;

int V;

public:

Graph(int V) { this->V = V; }

void addEdge(int x, int y, int w)

{

edgelist.push\_back({ w, x, y }); // cost , the egdes

}

void kruskals\_mst()

{

sort(edgelist.begin(), edgelist.end());

DSU s(V);

int ans = 0;

cout << "Following are the edges in the "

"constructed MST"

<< endl;

for (auto edge : edgelist) {

int w = edge[0];

int x = edge[1];

int y = edge[2];

if (s.find(x) != s.find(y)) {

s.unionp(x, y);

ans += w;

cout << x << " -- " << y << " == " << w

<< endl;

}

}

cout << "Minimum Cost Spanning Tree: " << ans;

}

};

int main()

{

int n,i=1;

cout<<"Enter number of nodes in the graph: ";

cin>>n;

Graph g(n);

char ch='y';

while(ch!='n'){

int x,y,cost;

cout<<"\nEnter start node for "<<i<<" edge: ";

cin>>x;

cout<<"Enter end node for "<<i<<" edge: ";

cin>>y;

cout<<"Enter cost: ";

cin>>cost;

g.addEdge(x,y,cost);

i+=1;

cout<<"Do you want to add more? (y/n): ";

cin>>ch;

}

g.kruskals\_mst();

return 0;

}

// g.addEdge(0, 1, 12);

// g.addEdge(1, 3, 10);

// g.addEdge(2, 3, 4);

// g.addEdge(2, 0, 3);

// g.addEdge(0, 3, 5);

**ASSIGNMENT 4: N-Queen**

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

bool isSafe(vector<int>& queens, int row, int col) {

for (int i = 0; i < row; i++) {

// Check if there is a queen in the same column or diagonal

if (queens[i] == col || queens[i] == col - (row - i) || queens[i] == col + (row - i))

return false;

}

return true;

}

void solveNQueensUtil(vector<int>& queens, int row, int& count) {

int n = queens.size();

if (row == n) {

// All queens have been placed, print the solution

count++;

cout << "Solution " << count << ": ";

for (int i = 0; i < n; i++) {

cout << queens[i] << " ";

}

cout << endl;

return;

}

for (int col = 0; col < n; col++) {

if (isSafe(queens, row, col)) {

queens[row] = col; // Place the queen

solveNQueensUtil(queens, row + 1, count);

queens[row] = -1; // Backtrack and remove the queen

}

}

}

void solveNQueens(int n) {

vector<int> queens(n, -1);

int count = 0;

solveNQueensUtil(queens, 0, count);

cout << "Total solutions: " << count << endl;

}

int main() {

int n;

cout << "Enter the number of queens (N): ";

cin >> n;

solveNQueens(n);

return 0;

}

**ASSIGNMENT 5: Chatbot**

import re

import random

# Define the chatbot's rules and responses

rules = {

'greeting': {

'patterns': [r'hello', r'hi', r'hey'],

'responses': ['Hello! Welcome to our food ordering service.', 'Hi there! How can I assist you with your order?']

},

'menu': {

'patterns': [r'menu', r'options'],

'responses': ['Sure! Here is our menu: ...\n 1.Pav Bhaji \n 2.Vada Pav \n 3.Maggi \n 4.Chai'],

},

'food items': {

'patterns': [r'Vada Pav', r'Pav Bhaji', r'Chai', r'Maggi'],

'responses': ['Do you want to confirm your order??'],

},

'order': {

'patterns': [r'order', r'I want to order'],

'responses': ['Great! Please let me know what items you would like to order.'],

},

'customization': {

'patterns': [r'customize', r'special request'],

'responses': ['Certainly! Let me know your specific requirements or any dietary restrictions.'],

},

'confirm\_order': {

'patterns': [r'confirm', r'place order'],

'responses': ['Perfect! Your order has been placed. The estimated delivery time is approximately 30 minutes.'],

},

'gratitude': {

'patterns': [r'thank you'],

'responses': ['My pleasure']

},

'cancel\_order': {

'patterns': [r'cancel', r'change', r'update'],

'responses': ['I apologize for the inconvenience. Please contact our customer support for order modifications.'],

},

'goodbye': {

'patterns': [r'bye', r'goodbye', r'see you'],

'responses': ['Thank you for choosing our food ordering service.', 'Goodbye!'],

},

'default': {

'responses': ['Im sorry, I didn’t understand that. Can you please rephrase?']

}

}

# Function to match user input with patterns

def match\_patterns(user\_input, patterns):

for pattern in patterns:

match = re.search(pattern, user\_input, re.IGNORECASE)

if match:

return True

return False

# Function to get chatbot's response

def get\_response(user\_input):

for intent, data in rules.items():

patterns = data.get('patterns')

if patterns and match\_patterns(user\_input, patterns):

responses = data.get('responses')

return random.choice(responses)

return random.choice(rules['default']['responses'])

# Main conversation loop

def chat():

print("Chatbot: Hello! Welcome to our food ordering service.")

while True:

user\_input = input("User: ")

response = get\_response(user\_input)

print("Chatbot:", response)

# Exit the loop if the user says goodbye

if any(re.search(pattern, user\_input) for pattern in rules['goodbye']['patterns']):

break

# Start the chatbot

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

chat()

import random

# Define the chatbot responses

greetings = ['Hello!', 'Hi!', 'Hey there!', 'Greetings!', 'Nice to see you!']

goodbyes = ['Goodbye!', 'See you later!', 'Farewell!', 'Bye!', 'Take care!']

help\_responses = ['How may I assist you?', 'What can I do for you?', 'How can I help?']

problem\_responses = ['I\'m sorry to hear that. Can you please tell me more about the problem?', 'Let me see if I can help. What seems to be the issue?', 'I\'ll do my best to help you. What\'s the problem?']

thankyou\_responses = ['You are welcome!', 'No problem!', 'It was my pleasure!', 'Glad to help!']

# Define the chatbot function

def chatbot():

print('Chatbot: ' + random.choice(greetings))

while True:

user\_input = input('User: ')

if 'hello' in user\_input.lower() or 'hi' in user\_input.lower() or 'hey' in user\_input.lower():

print('Chatbot: ' + random.choice(greetings))

elif 'bye' in user\_input.lower() or 'goodbye' in user\_input.lower() or 'see you' in user\_input.lower():

print('Chatbot: ' + random.choice(goodbyes))

break

elif 'help' in user\_input.lower() or 'support' in user\_input.lower():

print('Chatbot: ' + random.choice(help\_responses))

elif 'problem' in user\_input.lower() or 'issue' in user\_input.lower() or 'error' in user\_input.lower():

print('Chatbot: ' + random.choice(problem\_responses))

elif 'thank you' in user\_input.lower() or 'thanks' in user\_input.lower() or 'thankyou' in user\_input.lower():

print('Chatbot: ' + random.choice(thankyou\_responses))

else:

print('Chatbot: I\'m sorry, I don\'t understand. Can you please rephrase your request?')

# Test the chatbot

chatbot()

**ASSIGNMENT 6: Expert System**

# Define the rules

def rule1(symptoms):

if 'fever' in symptoms and 'cough' in symptoms and 'fatigue' in symptoms:

return 'You may have the flu.'

return None

def rule2(symptoms):

if 'fever' in symptoms and 'rash' in symptoms and 'headache' in symptoms:

return 'You may have meningitis.'

return None

def rule3(symptoms):

if 'pain' in symptoms and 'swelling' in symptoms and 'bruising' in symptoms:

return 'You may have a fracture.'

return None

def rule4(symptoms):

if 'abdominal pain' in symptoms and 'diarrhea' in symptoms and 'nausea' in symptoms:

return 'You may have food poisoning.'

return None

def rule5(symptoms):

if 'shortness of breath' in symptoms and 'chest pain' in symptoms and 'dizziness' in symptoms:

return 'You may be having a heart attack. Please seek medical attention immediately.'

return None

# Define the expert system

def diagnose(symptoms):

rules = [rule1, rule2, rule3, rule4, rule5]

results = []

for rule in rules:

result = rule(symptoms)

if result:

results.append(result)

if len(results) == 0:

return 'Sorry, we could not diagnose your condition.'

elif len(results) == 1:

return results[0]

else:

return 'You may have multiple conditions: ' + ', '.join(results)

# Test the expert system

symptoms = ['fever', 'rash', 'headache']

result = diagnose(symptoms)

print(result)

import nltk

nltk.download('wordnet')

import nltk

nltk.download('omw-1.4')

import re

from nltk.corpus import wordnet

list\_words=['hello','timings','shop','item','cost','thanks']

list\_syn={}

for word in list\_words:

synonyms=[]

for syn in wordnet.synsets(word):

for lem in syn.lemmas():

lem\_name = re.sub('[^a-zA-Z0-9 \n\.]', ' ', lem.name())

synonyms.append(lem\_name)

list\_syn[word]=set(synonyms)

print (list\_syn)

keywords={}

keywords\_dict={}

keywords['greet']=[]

for synonym in list(list\_syn['hello']):

keywords['greet'].append('.\\b'+synonym+'\\b.')

keywords['timings']=[]

for synonym in list(list\_syn['timings']):

keywords['timings'].append('.\\b'+synonym+'\\b.')

keywords['facility']=[]

for synonym in list(list\_syn['shop']):

keywords['facility'].append('.\\b'+synonym+'\\b.')

keywords['items']=[]

for synonym in list(list\_syn['item']):

keywords['items'].append('.\\b'+synonym+'\\b.')

keywords['items'].append('.\\b'+'items'+'\\b.')

keywords['price']=[]

for synonym in list(list\_syn['cost']):

keywords['price'].append('.\\b'+synonym+'\\b.')

keywords['price'].append('.\\b'+'prices'+'\\b.')

keywords['thanks']=[]

for synonym in list(list\_syn['thanks']):

keywords['thanks'].append('.\\b'+synonym+'\\b.')

for intent, keys in keywords.items():

keywords\_dict[intent]=re.compile('|'.join(keys))

print (keywords\_dict)

# Building a dictionary of responses

responses={

'greet':'Hello! How can I help you? \n',

'timings':'We are open from 9AM to 5PM, Monday to Friday. We are closed on weekends and public holidays. \n',

'price':'Pepsi-co: 20rs \n MountainDew: 20rs \n Coke: 40rs \n Sprite:30rs \n Thumbsup: 20rs \n Mazza: 10rs \n Frooti: 10rs \n ',

'items':'THE ITEMS PROVIDED ARE \n 1.Pepsi-co \n 2.MountainDew \n 3.Coke \n 4.Sprite \n 5.Thumbsup \n 6.Mazza \n 7.Frooti \n',

'facility':'Welcome to our soft drink shop. We sell soft drink at a very cheap price and also sell items in wholesale \n',

'thanks':'Glad to help you out \n',

'fallback':'I dont quite understand. Could you repeat that? \n',

}

print ("WELCOME TO THE SOFTYSHOP!!!! \n How can i help you \n")

# While loop to run the chatbot indefinetely

while (True):

user\_input = input().lower()

if user\_input == 'quit':

print ("Thank you for visiting. For further inquiry you can also contact at 6789456767")

break

if user\_input == 'bye':

print ("Thank you for visiting. For further inquiry you can also contact at 6789456767")

break

matched\_intent = None

for intent,pattern in keywords\_dict.items():

if re.search(pattern, user\_input):

matched\_intent=intent

key='fallback'

if matched\_intent in responses:

key = matched\_intent

print (responses[key])

**ASSIGNMENT 7: AWS/AZURE**

**ASSIGNMENT 8: APEX CALCULATOR/CURRENCY CONVERTER**

public class Calculator {

public integer firstNumber {get; set;}

public integer secondNumber {get; set;}

public integer result {get; set;}

public void Addition()

{

result=firstNumber + secondNumber;

}

public void Subtraction()

{

result=firstNumber - secondNumber;

}

public void Multiplication()

{

result=firstNumber \* secondNumber;

}

public void Division()

{

result=firstNumber / secondNumber;

}

}

<apex:page controller="Calculator">

<apex:sectionHeader title="Calculator" subtitle="Basic Calculator"/>

<apex:form >

<apex:pageBlock title="Simple Calculator">

<apex:pageBlockButtons location="bottom">

<apex:commandButton value="Addition" action="{!Addition}" reRender="res" />

<apex:commandButton value="Subtraction" action="{!Subtraction}" reRender="res"/>

<apex:commandButton value="Multiplication" action="{!Multiplication}" reRender="res"/>

<apex:commandButton value="Division" action="{!Division}" reRender="res"/>

</apex:pageBlockButtons>

<apex:pageBlockSection title="Calculator">

<apex:inputText label="Enter first number:" html-placeholder="First Number" value="{!firstNumber}"/>

<apex:inputText label="Enter second number:" html-placeholder="Second Number" value="{!secondNumber}"/>

<apex:outputText label="Result" value="{!result}" id="res"/>

</apex:pageBlockSection>

</apex:pageBlock>

</apex:form>

</apex:page>

public class CurrencyConversion {

public double rupees {get; set;}

public double result {get; set;}

public void toDollars()

{

result=rupees\*0.012;

}

public void toPounds()

{

result=rupees\*0.0088;

}

public void toEuros()

{

result=rupees\*0.0105;

}

}

<apex:page controller="CurrencyConversion">

<apex:sectionHeader title="Currency Converter" subtitle="Basic Currency Convertor"/>

<apex:form>

<apex:pageBlock title="Currency Converter">

<apex:pageBlockButtons location="bottom">

<apex:commandButton value="Rupees to Dollars" action="{!toDollars}" reRender="res"/>

<apex:commandButton value="Rupees to Pounds" action="{!toPounds}" reRender="res"/>

<apex:commandButton value="Rupees to Euros" action="{!toEuros}" reRender="res"/>

</apex:pageBlockButtons>

<apex:pageBlockSection title="Currency Conversion">

<apex:inputText label="Enter amount in rupees" html-placeholder="Amount" value="{!rupees}"/>

<apex:outputText label="Result" value="{!result}" id="res" />

</apex:pageBlockSection>

</apex:pageBlock>

</apex:form>

</apex:page>

**ASSIGNMENT 9: Custom application in Salesforce**