ASS 1

```
# Implement depth first search algorithm and Breadth First Search algorithm, Use an undirected
# graph and develop a recursive algorithm for searching all the vertices of a graph or tree data
# structure.

import sys

visited = []
queue = []

def bfs(visited, graph, node,searchNodee):
  print("BFS: ",end='')
  visited.append(node)
  queue.append(node)

  while queue:
    m = queue.pop(0)
    print (m, end = " ")
    #v
    if(m == searchNodee):
       break



    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

dfsVisited = set()
```

```
def dfs(dfsVisited, graph, node, searchNodee):
    if node not in dfsVisited:
        #1
        print (node, end=' ')
        if(node == searchNodee):
          #a
          sys.exit()
        dfsVisited.add(node)
        for neighbour in graph[node]:
            dfs(dfsVisited, graph, neighbour, searchNodee) #c


graph = {}


while True:
    root = input("Enter the root node: [input/done]  ")


    if(root=="done"):
        break


    if root not in graph:
        graph[root] = []


        while True:
            #k
            child = input("Please enter the child nodes of "+root+": [Enter input/done]  ")


            if child == "done":
                print()
                break


            if child not in graph[root]:
```

```
        graph[root].append(child)


print("\nThe graph is:\n")
print(graph, end='\n\n')


searchNode = input("Enter the node you want to search: ",)
print()
first_key = next(iter(graph))
bfs(visited, graph, first_key, searchNode)
print()
print("DFS: ",end='')
dfs(dfsVisited, graph, first_key, searchNode)


# enter the input in following way:
# a
# b
# c
# done
# b
# d
# e
# done
# c
# f
# g
# done
# d
# done
# e
# done
# f
```

done

g

done

done

c

ASS 1 done

ass 2 tic tac toe

Implement A star Algorithm for any game search problem.

```
import numpy as np


class Node:
    def __init__(self, state, parent=None):
        self.state = state
        self.parent = parent
        self.g_score = 0 if parent is None else parent.g_score + 1      #r
        self.h_score = self.heuristic()

    def f_score(self):
        return self.g_score + self.h_score

    def path(self):
        path = [self.state]
        node = self.parent
        while node is not None:
            path.append(node.state)
            node = node.parent
        return path[::-1]

    def heuristic(self):
        winner = check_winner(self.state)
        if winner is not None:
            #3
            if winner == 1:
                return 100 - self.g_score
```

```
            else:
                return -100 + self.g_score
        else:
            return self.get_empty_spaces() - self.get_opponent_empty_spaces()

    def get_empty_spaces(self):
        return np.sum(self.state == -1)

    def get_opponent_empty_spaces(self):
        return np.sum(self.state == 1)          #x

    def __eq__(self, other):
        return np.array_equal(self.state, other.state)

    def __hash__(self):
        return hash(self.state.tostring())

def check_winner(state):
    # Check rows
    for i in range(3):
        #o
        if np.all(state[i, :] == 1):
            return 1
        elif np.all(state[i, :] == 0):
            return 0

    # Check columns
    for i in range(3):
        if np.all(state[:, i] == 1):
            return 1
        elif np.all(state[:, i] == 0):
```

```
            return 0

    # Check diagonals
    if np.all(np.diag(state) == 1) or np.all(np.diag(np.fliplr(state)) == 1):
        return 1
    elif np.all(np.diag(state) == 0) or np.all(np.diag(np.fliplr(state)) == 0):
        return 0

    # Check for tie
    if np.sum(state == -1) == 0:
        return -1

    # No winner yet     #r
    return None

def get_possible_moves(state, player):
    moves = []
    for i in range(3):
        for j in range(3):
            if state[i, j] == -1:
                new_state = state.copy()
                new_state[i, j] = player
                moves.append(new_state)
    return moves

def a_star(start_state, player):
    open_list = [Node(start_state)]
    closed_list = []

    while open_list:
        current = min(open_list, key=lambda x: x.f_score())
```

```
        open_list.remove(current)
        closed_list.append(current)

        if check_winner(current.state) is not None:
            # If the current state is a win for the AI player, return the path
            return current.path()

        for child_state in get_possible_moves(current.state, player):
            child = Node(child_state, current)
            if child in closed_list:
                continue
            if child not in open_list:
                open_list.append(child)
            else:
                # Update the existing node if this path is better
                existing_child = open_list[open_list.index(child)]
                if child.g_score < existing_child.g_score:
                    existing_child.parent = current

    # If no path is found, return None
    return None

def print_board(state):
    """
    Prints the Tic Tac Toe board in a human-readable format.
    """
    symbols = {-1: " ", 0: "O", 1: "X"}  # Map player numbers to symbols
    for i in range(3):
        print("-------------")
        row = "|"
        for j in range(3):
```

```
            row += " " + symbols[state[i, j]] + " |"
        print(row)
    print("-------------")


def main():
    # Initialize the game board
    board = np.full((3, 3), -1)
    print_board(board)

    # Game loop
    while True:
        # Player 1 (human) turn
        print("Player 1 (X) turn.")
        row = int(input("Enter row number (0-2): "))
        col = int(input("Enter column number (0-2): "))
        if board[row, col] != -1:
            print("Invalid move. Try again.")
            continue
        board[row, col] = 1
        print_board(board)
        winner = check_winner(board)
        if winner is not None:
            break

        # AI player (player 2) turn
        print("Player 2 (O) turn.")
        path = a_star(board, 0)
        if path is None:
            print("Error: AI failed to find a valid move.")
            continue
```

```
        board = path[1]
        print_board(board)
        winner = check_winner(board)
        if winner is not None:
            break

    # Print the result
    if winner == 1:
        print("Player 1 (X) wins!")
    elif winner == 0:
        print("Player 2 (O) wins!")
    else:
        print("It's a tie!")

if __name__ == "__main__":
    main()
```

 # ass 2 done

ass 3

```
# Implement Greedy Search Algorithm for any of the following application: Prim's Minimal
# Spanning Tree Algorithm

import sys

class Graph:
    def __init__(self, vertices):          #v
        self.vertices = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printMST(self, parent):
        print("Edge \tWeight")#1
        for i in range(1, self.vertices):
            print(parent[i], "-", i, "\t", self.graph[i][parent[i]])



    def minKey(self, key, mstSet):
        min = sys.maxsize
        for v in range(self.vertices):
            if key[v] < min and mstSet[v] == False:
                min = key[v]
                min_index = v
        return min_index    #a

    def primMST(self):
        key = [sys.maxsize] * self.vertices
        parent = [None] * self.vertices
        key[0] = 0
```

```
        mstSet = [False] * self.vertices
        parent[0] = -1

        for cout in range(self.vertices):
            u = self.minKey(key, mstSet)
            mstSet[u] = True

            for v in range(self.vertices):  #c
                if self.graph[u][v] > 0 and mstSet[v] == False and key[v] > self.graph[u][v]:
                    key[v] = self.graph[u][v]
                    parent[v] = u

        self.printMST(parent)



#k
num_vertices = int(input("Enter the number of vertices in the graph: "))

graph = Graph(num_vertices)

for i in range(num_vertices):
    for j in range(num_vertices):
        if i != j and graph.graph[i][j] == 0:
            weight = int(input(f"Enter the weight of edge ({i}, {j}): "))
            graph.graph[i][j] = weight
            graph.graph[j][i] = weight

graph.primMST()
```

 # ass 3 done

ass 4

```
# Implement the solution for a Constraint Satisfaction Problem using Branch and Bound and
# Backtracking for n-queens problem or a graph coloring problem


# branch and bound


def printSolution(board):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end = " ")
        print()



def isSafe(row, col, nd, rd,rowLookup, ndLookup,rdLookup):
    if (ndLookup[nd[row][col]] or rdLookup[rd[row][col]] or rowLookup[row]):
        return False
    return True



def solveNQueensUtil(board, col, nd, rd,rowLookup, ndLookup,rdLookup):
    if(col >= N):
        return True
    for i in range(N):
        if(isSafe(i, col, nd, rd,rowLookup, ndLookup,rdLookup)):



            board[i][col] = 1
            rowLookup[i] = True
            ndLookup[nd[i][col]] = True
            rdLookup[rd[i][col]] = True
```

```
            if(solveNQueensUtil(board, col + 1,nd, rd,rowLookup, ndLookup,rdLookup)):
                return True
            board[i][col] = 0
            rowLookup[i] = False
            ndLookup[nd[i][col]] = False
            rdLookup[rd[i][col]] = False
    return False

def solveNQueens(N):

    board = [[0 for i in range(N)] for j in range(N)]



    nd = [[0 for i in range(N)] for j in range(N)]
    rd = [[0 for i in range(N)] for j in range(N)]
    rowLookup = [False] * N



    x = 2 * N - 1
    ndLookup = [False] * x
    rdLookup= [False] * x



    for r in range(N):
        for c in range(N):
            nd[r][c] = r + c
            rd[r][c] = r - c + N - 1

    if(solveNQueensUtil(board, 0, nd, rd,rowLookup, ndLookup,rdLookup) == False):
```

```
        print("Solution does not exist")
        return False


    printSolution(board)
    return True


N=int(input("Enter a Number: "))
solveNQueens(N)


# backtracking

from typing import List
boardcount=0
def isboardok(chessboard:List,row:int,col:int):
    for c in range(col):
        if(chessboard[row][c]=='Q'):
            return False
    for r,c in zip(range(row-1,-1,-1),range(col-1,-1,-1)):
        if(chessboard[r][c]=='Q'):
            return False
    for r,c in zip(range(row+1,len(chessboard),1),range(col-1,-1,-1)):
        if(chessboard[r][c]=='Q'):
            return False
    return True


def displayboard(chessboard:List):
    for row in chessboard:
```

```
        print(row)
    print()



def placenqueens(chessboard:List,col:int):
    global boardcount
    if(col>=len(chessboard)):
        boardcount+=1
        print("Board"+str(boardcount))
        print("======================")
        displayboard(chessboard)
        print("======================\n\n")
    else:
        for row in range(len(chessboard)):
            chessboard[row][col]='Q'
            if(isboardok(chessboard,row,col)==True):
                placenqueens(chessboard,col+1)
            chessboard[row][col]='.'


chessboard=[]
N=int(input("Enter chessboard size: "))
for i in range(N):
    row=["."]*N
    chessboard.append(row)


placenqueens(chessboard,0)
```

 # ass 4 done

ass 6 expert system

Implement any one of the following Expert System

I. Information management

II. Hospitals and medical facilities

III. Help desks management

IV. Employee performance evaluation

V. Stock market trading

VI. Airline scheduling and cargo schedules

```
go:-
hypothesis(Disease),
write('It is suggested that the patient has '),
write(Disease),
nl,
undo;
write('Sorry, the system is unable to identify the disease'),nl,undo.


hypothesis(cold) :-
symptom(headache),
symptom(runny_nose),
symptom(sneezing),
symptom(sore_throat),
nl,
write('Advices and Sugestions:'),
nl,
write('1: Tylenol'),
nl,
write('2: Panadol'),
nl,
```

```
write('3: Nasal spray'),
nl,
write('Please weare warm cloths because'),
nl,!.


hypothesis(influenza) :-
symptom(sore_throat),
symptom(fever),
symptom(headache),
symptom(chills),
symptom(body_ache),
nl,
write('Advices and Sugestions:'),
nl,
write('1: Tamiflu'),
nl,
write('2: Panadol'),
nl,
write('3: Zanamivir'),
nl,
write('Please take a warm bath and do salt gargling because'),
nl,!.


hypothesis(typhoid) :-
symptom(headache),
symptom(abdominal_pain),
symptom(poor_appetite),
symptom(fever),
nl,
```

```
write('Advices and Sugestions:'),
nl,
write('1: Chloramphenicol'),
nl,
write('2: Amoxicillin'),
nl,
write('3: Ciprofloxacin'),
nl,
write('4: Azithromycin'),
nl,
write('Please do complete bed rest and take soft diet because'),
nl,!.


hypothesis(chicken_pox) :-
symptom(rash),
symptom(body_ache),
symptom(fever),
nl,
write('Advices and Sugestions:'),
nl,
write('1: Varicella vaccine'),
nl,
write('2: Immunoglobulin'),
nl,
write('3: Acetomenaphin'),
nl,
write('4: Acyclovir'),
nl,
write('Please do have oatmeal bath and stay at home because'),
nl.
```

```
hypothesis(measles) :-
symptom(fever),
symptom(runny_nose),
symptom(rash),
symptom(conjunctivitis),
nl,
write('Advices and Sugestions:'),
nl,
write('1: Tylenol'),
nl,
write('2: Aleve'),
nl,
write('3: Advil'),
nl,
write('4: Vitamin A'),
nl,
write('Please get rest and use more liquid because'),
nl,!.


hypothesis(malaria) :-
symptom(fever),
symptom(sweating),
symptom(headache),
symptom(nausea),
symptom(vomiting),
symptom(diarrhea),
nl,
write('Advices and Sugestions:'),
```

```
nl,
write('1: Aralen'),
nl,
write('2: Qualaquin'),
nl,
write('3: Plaquenil'),
nl,
write('4: Mefloquine'),
nl,
write('Please do not sleep in open air and cover your full skin because'),
nl,!.


ask(Question) :-
write('Does the patient has the symptom '),
write(Question),
write('? : '),
read(Response),
nl,
( (Response == yes ; Response == y)
->
assert(yes(Question)) ;
assert(no(Question)), fail).
:- dynamic yes/1,no/1.

symptom(S) :-
(yes(S)
->
true ;
(no(S)
->
```

```
fail ;
ask(S))).

undo :- retract(yes(_)),fail.
undo :- retract(no(_)),fail.
undo.

/*
Output-
vnd@vnd-Lenovo-H520e:~$ swipl -s medicalExpert.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- go.
Does the patient has the symptom headache? : y.

Does the patient has the symptom runny_nose? : |: n.

Does the patient has the symptom sore_throat? : |: y.

Does the patient has the symptom fever? : |: y.

Does the patient has the symptom chills? : |: y.

Does the patient has the symptom body_ache? : |: y.
```

Advices and Sugestions:

1: Tamiflu

2: Panadol

3: Zanamivir

Please take a warm bath and do salt gargling because

It is suggested that the patient has influenza

true .

?-

*/

 # ass 6 done