```
from queue import Queue

class Graph:
    n = None
    matrix = []
    dfs_list = []
    bfs_list = []
    def __init__(self, matrix: list, verbose: bool = False, n: int = 0) -> None:
        """
        Args:
            matrix (list): adjency matrix
            verbose (bool, optional): To use terminal. Defaults to False.
            n (int, optional): number of nodes. Defaults to 0.
        """
        if verbose:
            for i in range(n):
                row = []
                for j in range(n):
                    connection = int(
                        input("Enter 0 if no connection between " + str(i) + "
and " + str(j) + " else enter 1: "))
                    row.append(connection)
                self.matrix.append(row)
        else:
            self.matrix = matrix
    def printGraph(self) -> None:
        """
        Function for displaying graph
        """
        for row in self.matrix:
            print(row)
    def dfs(self, source: int, visited: list) -> list:
        """
        DFS implementation using adjacency matrix
        Args:
            source (int): starting node of the graph
            visited (list): boolean list for visited nodes
        Returns:
            list: dfs sequence
        """
        self.dfs_list.append(source)
        visited[source] = True
        row = self.matrix[source]
        for node in range(len(row)):
            if (visited[node] == False and self.matrix[source][node] != 0):
                self.dfs(node, visited)
        return self.dfs_list
    def bfs(self, source: int, visited: list) -> list:
        """
        BFS implementation using adjacency matrix
        Args:
            source (int): starting node of the graph
            visited (list): boolean list for visited nodes
        Returns:
            list: bfs sequence
        """
        queue = Queue()
        queue.put(source)
        visited[source] = True
        while (queue.empty() != True):
            curr = queue.get()
            self.bfs_list.append(curr)
            row = self.matrix[curr]
            for node in range(len(row)):
                if (visited[node] == False and self.matrix[curr][node] != 0):
                    queue.put(node)
                    visited[node] = True
        return self.bfs_list
    def to_file(self, filename: str, language: str = "py") -> bool:
        f = open(filename, 'w')
        code = """from queue import Queue
        class Graph:
            n = None
            matrix = []
            dfs_list = []
            bfs_list = []
            def __init__(self, matrix:list, verbose:bool=False, n:int=0) ->
None:
                if verbose:
                    for i in range(n):
                        row = []
                        for j in range(n):
                            connection = int(input("Enter 0 if no connection
between " + str(i) + " and " + str(j) + " else enter 1: "))
                            row.append(connection)
                        self.matrix.append(row)
                else:
                    self.matrix = matrix
            def printGraph(self) -> None:
                for row in self.matrix:
                    print(row)
            def dfs(self, source:int, visited:list) -> list:
                self.dfs_list.append(source)
                visited[source] = True
                row = self.matrix[source]
                for node in range(len(row)):
                    if (visited[node] == False and self.matrix[source][node] !=
0):
                        self.dfs(node, visited)
                return self.dfs_list
            def bfs(self, source:int, visited:list) -> list:
                queue = Queue()
                queue.put(source)
                visited[source] = True
                while (queue.empty() != True):
                    curr = queue.get()
                    self.bfs_list.append(curr)
                    row = self.matrix[curr]
                    for node in range(len(row)):
                        if (visited[node] == False and self.matrix[curr][node] !
= 0):
                            queue.put(node)
                            visited[node] = True
                return self.bfs_list
        """
        f.write(code)
        f.close()
        return True


if __name__ == "__main__":
    matrix = [[0, 1, 1, 0],
              [1, 0, 0, 1],
              [1, 0, 0, 1],
              [1, 1, 1, 0]]
    obj = Graph(matrix)
    obj.printGraph()
    visited = [False] * 4
    print(obj.dfs(0, visited))
    visited = [False] * 4
    print(obj.bfs(0, visited))
    obj.to_file("code.py")

/*
output:-
[0, 1, 1, 0]
[1, 0, 0, 1]
[1, 0, 0, 1]
[1, 1, 1, 0]
[0, 1, 3, 2]
[0, 1, 2, 3]  */
```

```python
import copy
class Node:
    def __init__(self, data, level, fval):
        self.data = data
        self.level = level
        self.fval = fval
    def generate_child(self):
        x, y = self.find(self.data, '_')
        val_list = [[x, y - 1], [x, y + 1], [x - 1, y], [x + 1, y]]
        children = []
        for i in val_list:
            child = self.shuffle(self.data, x, y, i[0], i[1])
            if child is not None:
                child_node = Node(child, self.level + 1, 0)
                children.append(child_node)
        return children
    def shuffle(self, data, x1, y1, x2, y2):
        if x2 >= 0 and x2 < len(data) and y2 >= 0 and y2 < len(data):
            temp_data = copy.deepcopy(data)
            temp = temp_data[x2][y2]
            temp_data[x2][y2] = temp_data[x1][y1]
            temp_data[x1][y1] = temp
            return temp_data
        else:
            return None
    def find(self, data, x):
        for i in range(len(self.data)):
            for j in range(len(self.data)):
                if data[i][j] == x:
                    return i, j
class Puzzle:
    def __init__(self, size):
        self.size = size
        self.open = []
        self.closed = []

    def accept(self):
        puz = []
        for i in range(0, self.size):
            temp = input().split(" ")
            puz.append(temp)
        return puz

    def f(self, start, goal):
        return self.h(start.data, goal) + start.level

    def h(self, start, goal):
        temp = 0
        for i in range(0, self.size):
            for j in range(0, self.size):
                if start[i][j] != goal[i][j] and start[i][j] != '_':
                    temp += 1
        return temp

    def process(self):
        print("enter the start state matrix \n")
        start = self.accept()
        print("enter the goal state matrix \n")
        goal = self.accept()
        start = Node(start, 0, 0)
        start.fval = self.f(start, goal)
        self.open.append(start)
        print("\n\n")
        while True:
            cur = self.open[0]
            print("\n")
            for i in cur.data:
                for j in i:
                    print(j, end=" ")
                print("")
            if (self.h(cur.data, goal) == 0):
                break
            for i in cur.generate_child():
                i.fval = self.f(i, goal)
                self.open.append(i)
            self.closed.append(cur)
            del self.open[0]
            self.open.sort(key=lambda x: x.fval, reverse=False)
puz = Puzzle(3)
puz.process()


/*
output :-
C:\Users\anike\Scripts\python.exe C:/Users/anike/PycharmProjects/pushup/codes.py


enter the start state matrix
1 2 3
4 5 _
7 8 6

enter the goal state matrix
1 2 3
4 5 6
7 8 _


1 2 3
4 5 _
7 8 6

1 2 3
4 5 6
7 8 _
Process finished with exit code 0  */
```

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i

        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]



# Test the function
arr = [64, 25, 12, 22, 11]
selection_sort(arr)
print("Sorted array:", arr)


/*

OUTPUT:-
Sorted array: [11, 12, 22, 25, 64]

 */
```

```
def solveNQueens(n : int):
    col = set()
    posDiag = set() # determined by r+c
    negDiag = set() # determined by r-c

    res = []
    board = [['.' for _ in range(n)] for _ in range(n)]

    def backtrack(r):
        if (r == n):
            res.append([''.join(row) for row in board])
            return
        for c in range(n):
            if (c in col or r+c in posDiag or r-c in negDiag):
                continue
            col.add(c)
            posDiag.add(r + c)
            negDiag.add(r - c)
            board[r][c] = 'Q'
            backtrack(r + 1)
            col.remove(c)
            posDiag.remove(r + c)
            negDiag.remove(r - c)
            board[r][c] = '.'
    backtrack(0)
    return res

def printSolutions(boards):
    for board in enumerate(boards):
        print(f"Solution: {board[0]+1}")
        for row in board[1]:
            for col in row:
                print(col, end=' ')
            print()
        print()

if __name__ == "__main__":
    boards = solveNQueens(4)
    printSolutions(boards)

/*

output:-
C:\Users\anike\Scripts\python.exe C:/Users/anike/PycharmProjects/pushup/codes.py

Solution: 1
. Q . .
. . . Q
Q . . .
. . Q .

Solution: 2
. . Q .
Q . . .
. . . Q
. Q . .
Process finished with exit code 0 */
```

```
import streamlit as st

bot_name = "College Buddy"

knowledge_base = {

    "what is your name?" : [
        f"My name is {bot_name}! \n Happy to help you out with your College
enquiries!"
    ],

    "hello": [
        f"Hello my name is {bot_name}! \n Happy to help you out with your
College enquiries!"
    ],

    "what are the best colleges from pune?": [
        "COEP",
        "PICT",
        "VIT",
        "CUMMINS",
        "PCCOE"
    ],

    "which are the best engineering branches?" : [
        "Computer Engineering",
        "IT Engineering",
        "ENTC Engineering"
    ],

    "what are the top branch cut-offs for coep?" : [
        "Computer Engineering : 99.8 percentile",
        "Does not have IT branch",
        "ENTC Engineering: 99.2 percentile",
    ],

    "what are the top branch cut-offs for pict?" : [
        "Computer Engineering : 99.4 percentile",
        "IT Engineering : 98.6 percentile",
        "ENTC Engineering: 97.2 percentile",
    ],

    "what are the top branch cut-offs for vit?" : [
        "Computer Engineering : 99.8 percentile",
        "IT Engineering: 97.1 percentile",
        "ENTC Engineering: 96.2 percentile",
    ],

    "what are the top branch cut-offs for cummins?" : [
        "Computer Engineering : 99.8 percentile",
        "Does not have IT branch",
        "ENTC Engineering: 99.2",
    ],

    "what are the top branch cut-offs for pccoe?" : [
        "Computer Engineering : 99.8 percentile",
        "Does not have IT branch",
        "ENTC Engineering: 99.2",
    ],

    "When do college admissions start?": [
        "Admissions generally start around August",
    ],
}

st.header("College Enquiry Rule Based Chatbot")

def respond(input: str):
    if (input in knowledge_base):
        print(input)
        values = knowledge_base[input]
        for value in values:
            st.write(value)
    else:
        print(input)
        key = input
        st.write("Question is not present in the knowledge base!\nCould you
please enter the appropriate answer for the question below-")
        answer = st.text_input("Answer")
        add = st.button("Add answer")
        if (add):
            knowledge_base[key] = [answer]

if __name__ == "__main__":
    input = st.text_input("Enter a query here-")
    input = input.lower()
    col1, col2 = st.columns([1,0.1])
    with col1:
        ask = st.button("Ask")
    with col2:
        quit = st.button("Quit")
    if (ask):
        respond(input)
    if (quit):
        st.write("Thank you for using the Chatbot")



"""

output:-


College Enquiry Rule Based Chatbot

Enter a query here- hello

Hello my name is College Buddy!
Happy to help you out with your College enquiries!

[Buttons: "Ask" and "Quit"]

Ask


College Enquiry Rule Based Chatbot

Enter a query here- What are the top branch cut-offs for NMIET?

Computer Science: 90 percentile

[Buttons: "Ask" and "Quit"]

Quit

College Enquiry Rule Based Chatbot
```

```
Thank you for using the Chatbot
"""
```

```
import streamlit as st
from typing import List

knowledge_base = {
    "cold" : [
        "1: Tylenol",
        "2: Panadol",
        "3: Nasal spray",
        "4: Please wear warm clothes!"
    ],

    "influenza": [
        "1: Tamiflu"
        "2: Panadol",
        "2: Zanamivir",
        "4: Please take a warm bath and do salt gargling!"
    ],

    "typhoid": [
        "1: Chloramphenicol",
        "2: Amoxicillin",
        "3: Ciproflaxacin",
        "4: Azithromycin",
        "5: Please do complete bed rest and take soft diet!"
    ],

    "chicken pox" : [
        "1: Varicella vaccine",
        "2: Immunoglobulin",
        "3: Acetomenaphin",
        "4: Acyclovir",
        "5: Please do have have oatmeal and stay at home!"
    ],

    "measles" : [
        "1: Tylenol",
        "2: Aleve",
        "3: Advil",
        "4: Vitamin A",
        "5: Please get rest and use more liquid!"
    ],

    "malaria" : [
        "1: Aralen",
        "2: Qualaquin",
        "3: Plaquenil",
        "4: Mefloquine",
        "5: Please do not sleep in open air and cover your full skin!"
    ]
}

st.header("Medical Diagnosis Expert System")

def respond(input: List[str]):
    symptoms = input
    if (len(symptoms) == 3):
        if (symptoms[0] == "rash" and symptoms[1] == "body ache" and symptoms[2]
== "fever"):
            st.write("You have chicken pox!")
            st.write("Please take the following medicines and precautions-")
            for i in knowledge_base["chicken pox"]:
                st.write(i)
        else:
            st.write("Question is not present in the knowledge base!\nCould you
please enter the appropriate answer for the question below-")
            answer = st.text_input("Answer")
            add = st.button("Add answer")
            if (add):
                for key in symptoms:
                    knowledge_base[key] = [answer]
    elif (len(symptoms) == 4):
        if (symptoms[0] == "headache" and symptoms[1] == "runny nose" and
symptoms[2] == "sneezing" and symptoms[3] == "sore throat"):
            st.write("You have cold!")
            st.write("Please take the following medicines and precautions-")
            for i in knowledge_base["cold"]:
                st.write(i)
        elif (symptoms[0] == "headache" and symptoms[1] == "abdominal pain" and
symptoms[2] == "poor appetite" and symptoms[3] == "fever"):
            st.write("You have typhoid!")
            st.write("Please take the following medicines and precautions-")
            for i in knowledge_base["typhoid"]:
                st.write(i)
        elif (symptoms[0] == "fever" and symptoms[1] == "runny nose" and
symptoms[2] == "rash" and symptoms[3] == "conjunctivitis"):
            st.write("You have measles!")
            st.write("Please take the following medicines and precautions-")
            for i in knowledge_base["measles"]:
                st.write(i)
        else:
            st.write("Question is not present in the knowledge base!\nCould you
please enter the appropriate answer for the question below-")
            answer = st.text_input("Answer")
            add = st.button("Add answer")
            if (add):
                for key in symptoms:
                    knowledge_base[key] = [answer]
    elif (len(symptoms) == 5):
        if (symptoms[0] == "sore throat" and symptoms[1] == "fever" and
symptoms[2] == "headache" and symptoms[3] == "chills" and symptoms[4] == "body
ache"):
            st.write("You have influenza!")
            st.write("Please take the following medicines and precautions-")
            for i in knowledge_base["influenza"]:
                st.write(i)
        else:
            st.write("Question is not present in the knowledge base!\nCould you
please enter the appropriate answer for the question below-")
            answer = st.text_input("Answer")
            add = st.button("Add answer")
            if (add):
                for key in symptoms:
                    knowledge_base[key] = [answer]
    elif (len(symptoms) == 6):
        if (symptoms[0] == "fever" and symptoms[1] == "sweating" and symptoms[2]
== "headache" and symptoms[3] == "nausea" and symptoms[4] == "vomiting" and
symptoms[5] == "diahrrea"):
            st.write("You have malaria!")
            st.write("Please take the following medicines and precautions-")
            for i in knowledge_base["malaria"]:
                st.write(i)
        else:
            st.write("Question is not present in the knowledge base!\nCould you
please enter the appropriate answer for the question below-")
            answer = st.text_input("Answer")
            add = st.button("Add answer")
            if (add):
                for key in symptoms:
                    knowledge_base[key] = [answer]

if __name__ == "__main__":
    options = st.multiselect(
    'What are your symptoms?',
    ["headache", "runny nose", "sneezing", "sore throat", "fever", "chills",
"body ache", "abdominal pain", "poor appetite", "rash", "conjunctivitis",
"sweating", "nausea", "vomiting", "diahrrea"],
    [])

    col1, col2 = st.columns([1,0.1])
    with col1:
        ask = st.button("Ask")
    with col2:
        quit = st.button("Quit")
    if (ask):
        respond(options)
    if (quit):
        st.write("Thank you for using the Expert system!")


"""
output:-

Medical Diagnosis Expert System

[Multi-select options for symptoms: User selects symptoms]

[Buttons: "Ask" and "Quit"]

Ask


Medical Diagnosis Expert System

What are your symptoms?

- headache
- runny nose
- fever

[Buttons: "Ask" and "Quit"]

Quit

Thank you for using the Expert system!
```