LAB # 14

Open Ended Lab

Lab Tasks:-

- Apply Breadth First Search on following graph considering the initial state is A and final state is F. Show results in form of open and closed list.

Code:-

```
from collections import deque

graph = {
    'A': ['B', 'C', 'D'],
    'B': [],
    'C': ['E', 'F'],
    'D': ['F'],
    'E': [],
    'F': []
}

def bfs(start, goal):
    open_list = deque([start])
    closed_list = []

    while open_list:
        node = open_list.popleft()
        closed_list.append(node)

        print("Open List:", list(open_list))
        print("Closed List:", closed_list)
        print("---------------")

        if node == goal:
            print("Goal Found:", goal)
            return

        for neighbor in graph[node]:
            if neighbor not in open_list and neighbor not in closed_list:
                open_list.append(neighbor)

bfs('A', 'F')
```

Output:-

```
Open List: []
Closed List: ['A']
---------------
Open List: ['C', 'D']
Closed List: ['A', 'B']
---------------
Open List: ['D']
Closed List: ['A', 'B', 'C']
---------------
Open List: ['E', 'F']
Closed List: ['A', 'B', 'C', 'D']
---------------
Open List: ['F']
Closed List: ['A', 'B', 'C', 'D', 'E']
---------------
Open List: []
Closed List: ['A', 'B', 'C', 'D', 'E', 'F']
---------------
Goal Found: F
```

- Apply Depth First Search on the graph given in question 1. Considering the initial state is A and final state is F. Show results in form of open and closed list.

Code:-

```
graph = {
    'A': ['B', 'C', 'D'],
    'B': [],
    'C': ['E', 'F'],
    'D': ['F'],
    'E': [],
    'F': []
}

def dfs(start, goal):
    open_list = [start]
    closed_list = []

    while open_list:
        node = open_list.pop()
        closed_list.append(node)

        print("Open List:", open_list)
        print("Closed List:", closed_list)
        print("---------------")

        if node == goal:
            print("Goal Found:", goal)
            return

        for neighbor in reversed(graph[node]):
            if neighbor not in open_list and neighbor not in closed_list:
                open_list.append(neighbor)

dfs('A', 'F')
```

Output:-

```
Open List: []
Closed List: ['A']
---------------
Open List: ['D', 'C']
Closed List: ['A', 'B']
---------------
Open List: ['D']
Closed List: ['A', 'B', 'C']
---------------
Open List: ['D', 'F']
Closed List: ['A', 'B', 'C', 'E']
---------------
Open List: ['D']
Closed List: ['A', 'B', 'C', 'E', 'F']
---------------
Goal Found: F
```

- Create a Class “Employee”, it’s a common base class for all the employee. Then initialize employee’s parameter like empName, empID, and salary and create function like displayCount() contain total number of employee in your knowledge base and displayEmployee() contain empName, empID and their salary.

Code:-

```
class Employee:
    count = 0

    def __init__(self, empName, empID, salary):
        self.empName = empName
        self.empID = empID
        self.salary = salary
        Employee.count += 1

    @staticmethod
    def displayCount():
        print("Total Employees:", Employee.count)

    def displayEmployee(self):
        print("Name:", self.empName)
        print("ID:", self.empID)
        print("Salary:", self.salary)
        print("----------------")

e1 = Employee("Sarah", 101, 50000)
e2 = Employee("Rayyan", 102, 60000)
e3 = Employee("Wania", 103, 55000)

e1.displayEmployee()
e2.displayEmployee()
e3.displayEmployee()

Employee.displayCount()
```

Output:-

```
Name: Sarah
ID: 101
Salary: 50000
----------------
Name: Rayyan
ID: 102
Salary: 60000
----------------
Name: Wania
ID: 103
Salary: 55000
----------------
Total Employees: 3
```

- Design a Python program to perform the following tasks using dictionaries:

Create an empty dictionary named student_db to store student information. Implement a function add_student that takes student ID, name, and grade as parameters. Implement a function display_students. Implement a function get_average_grade that calculates and returns the average grade of all students in the student_db. Create instances of students in the student_db, invoke the functions you implemented, and demonstrate their functionality.

Code:-

```
student_db = {}

def add_student(student_id, name, grade):
    student_db[student_id] = {
        "Name": name,
        "Grade": grade
    }

def display_students():
    for sid, details in student_db.items():
        print("ID:", sid)
        print("Name:", details["Name"])
        print("Grade:", details["Grade"])
        print("----------------")

def get_average_grade():
    total = sum(student["Grade"] for student in student_db.values())
    return total / len(student_db) if student_db else 0

add_student(1, "Wania", 85)
add_student(2, "Rayyan", 90)
add_student(3, "Sarah", 80)

display_students()

print("Average Grade:", get_average_grade())
```

Output:-

```
ID: 1
Name: Wania
Grade: 85
----------------
ID: 2
Name: Rayyan
Grade: 90
----------------
ID: 3
Name: Sarah
Grade: 80
----------------
Average Grade: 85.0
```