NAMAL UNIVERSITY MIANWALI
DEPARTMENT OF ELECTRICAL ENGINEERING

DATA STRUCTURE AND ALGORITHM
LAB # 10
REPORT
Title : Graphs In Python

Name	Fahim-Ur-Rehman Shah
Roll No	NIM-BSEE-2021-24
Intructor	Ms. Naureen Shaukat
Date	13-June-2023
Marks	

1. Lab Objectives

The objective of this lab is to introduce students to the concept of graphs in python. In this lab, the students will enable the concepts of implementing a graph, adding the vertices and edges. Students will make the python code for removing the vertices and edges. Students will make the python code for computing the degree the vertices and edges. Students will also see the directed and undirected graphs implementation in Python.

2. Lab Outcomes

- CLO:1 Recognize the usage of fundamental Data structure using Python Programming Language.
- CLO:3 Demonstrate solution to real life problems using appropriate data structures.

3. Equipment

- Software ○ IDLE (Python 3.11)

4. Instructions

1. This is an individual lab. You will perform the tasks individually and submit a report.
2. Some of these tasks (marked as 'Example') are for practice purposes only while others (marked as 'Task') have to be answered in the report.
3. When asked to display an output in the task, either save it as jpeg or take a screenshot, in order to insert it in the report.
4. The report should be submitted on the given template, including:
    a. Code (copy and pasted, NOT a screenshot)
    b. Output figure (as instructed in 3)
    c. Explanation where required
5. The report should be properly formatted, with easy to read code and easy to see figures.
6. Plagiarism or any hint thereof will be dealt with strictly. Any incident where plagiarism is caught, both (or all) students involved will be given zero marks, regardless of who copied whom. Multiple such incidents will result in disciplinary action being taken.
7. Late submission of report is allowed within 03 days after lab with 20% deduction of marks every day.
8. You have to submit report in pdf format (Reg.X_DSA_LabReportX.pdf).

5. Background

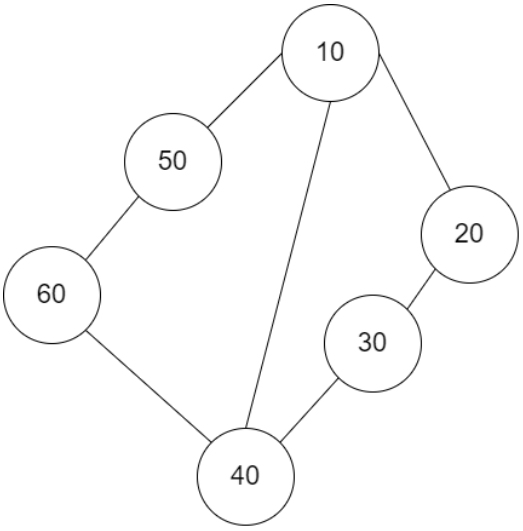
Graphs:

In data structures, a graph is a collection of nodes (also called vertices) that are connected by edges. It is a fundamental data structure used to represent relationships between different entities. Graphs are widely used in various applications, including computer networks, social networks, transportation networks, and more. A graph can be visualized as a set of points (vertices) connected by lines (edges). Each edge represents

relationship or connection between two vertices. The edges can be directed (one-way) or undirected (two-way), depending on the nature of the relationship they represent.

Example 1:

Write the python code from your lecture slides about the basic graph implementation. Implement the given graph.



Output:

Add the scree shot in the report.

6. Lab Tasks:

Task 1: Using above example code, add a method to remove the vertex, and add the method to remove the edge.

Code :

```
class Vertex:
    def __init__(self, data):
        self.data = data
        self.edges = []

    def add_edge(self, edge):
        self.edges.append(edge)


class Edge:
    def __init__(self, start, end):
        self.start = start
        self.end = end


class Graph:
    def __init__(self):
        self.vertices = []

    def add_vertex(self, data):
        vertex = Vertex(data)
        self.vertices.append(vertex)
        return vertex
```

```
    def add_edge(self, start, end):
        edge = Edge(start, end)
        start.add_edge(edge)
        end.add_edge(edge)

    def remove_vertex(self, vertex):
        for v in self.vertices:
            if vertex == v.data:
                self.vertices.remove(v)
            for edge in v.edges:
                if edge.start.data == vertex:
                    self.remove_edge(edge.start.data,edge.end.data)
                if edge.end.data == vertex:
                    self.remove_edge(edge.start.data,edge.end.data)


    def remove_edge(self, start, end):
        for vertex in self.vertices:
            for edge in vertex.edges:
                if (edge.start.data == start) and (edge.end.data == end):
                    #print(f" edges of vertix : {vertex.data} : {edge.start.data} -- {edge.end.data}")
                    vertex.edges.remove(edge)

    def print_graph(self):
        for vertex in self.vertices:
            print("Vertex:", vertex.data)
            print("Edges:")
            for edge in vertex.edges:
                print(f"{edge.start.data} --- {edge.end.data}")
            print()


graph = Graph()

vertex_a = graph.add_vertex(10)
vertex_b = graph.add_vertex(20)
vertex_c = graph.add_vertex(30)
vertex_d = graph.add_vertex(40)
vertex_e = graph.add_vertex(50)
vertex_f = graph.add_vertex(60)

graph.add_edge(vertex_a, vertex_e)
graph.add_edge(vertex_a, vertex_d)
graph.add_edge(vertex_a, vertex_b)
graph.add_edge(vertex_b, vertex_c)
graph.add_edge(vertex_c, vertex_d)
graph.add_edge(vertex_e, vertex_f)
graph.add_edge(vertex_f, vertex_d)

graph.print_graph()

graph.remove_vertex(60)
print("\nAfter removing vertex 60:")
graph.print_graph()

graph.remove_edge(20, 30)
print("\nAfter removing edge between 20 and 30 :")
graph.print_graph()
```

Output :

```
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 10\8 may 2023> python -u "e:\Semester 4\Data Structure and Algorithm\Lab\Lab 10\8 may 2023\Class_code.py"
Vertex: 10
Edges:
10 --- 50
10 --- 40
10 --- 20

Vertex: 20
Edges:
10 --- 20
20 --- 30

Vertex: 30
Edges:
20 --- 30
30 --- 40

Vertex: 40
Edges:
10 --- 40
30 --- 40
60 --- 40

Vertex: 50
Edges:
10 --- 50
50 --- 60

Vertex: 60
Edges:
50 --- 60
60 --- 40


After removing vertex 60:
Vertex: 10
Edges:
10 --- 50
10 --- 40
10 --- 20

Vertex: 20
Edges:
10 --- 20
20 --- 30

Vertex: 30
Edges:
20 --- 30
30 --- 40

Vertex: 40
Edges:
```

```
10 --- 40
30 --- 40

Vertex: 50
Edges:
10 --- 50


After removing edge between 20 and 30 :
Vertex: 10
Edges:
10 --- 50
10 --- 40
10 --- 20

Vertex: 20
Edges:
10 --- 20

Vertex: 30
Edges:
30 --- 40

Vertex: 40
Edges:
10 --- 40
30 --- 40

Vertex: 50
Edges:
10 --- 50

PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 10\8 may 2023>
```

Task 2: Make the graph in given figure using only one class of graph. Make a method to search for specific data inside graph.

Code :

```
class Graph:
    def __init__(self):
        self.vertices = []

    def add_vertex(self, data):
        vertex = {"data": data, "edges": []}
        self.vertices.append(vertex)
        return vertex

    def add_edge(self, start, end):
        edge = {"start": start, "end": end}
        start["edges"].append(edge)
        end["edges"].append(edge)

    def remove_vertex(self, vertex_data):
```

```
        for vertex in self.vertices:
            if vertex["data"] == vertex_data:
                for edge in vertex["edges"]:
                    edge["start"]["edges"].remove(edge)
                    edge["end"]["edges"].remove(edge)
                self.vertices.remove(vertex)
                break

    def remove_edge(self, start_data, end_data):
        for vertex in self.vertices:
            for edge in vertex["edges"]:
                if (edge["start"]["data"] == start_data) and (edge["end"]["data"] == end_data):
                    vertex["edges"].remove(edge)
                    break

    def print_graph(self):
        for vertex in self.vertices:
            print("Vertex:", vertex["data"])
            print("Edges:")
            for edge in vertex["edges"]:
                print(f"{edge['start']['data']} --- {edge['end']['data']}")
            print()

    def search_data(self, vertex_data):
        found = False
        for vertex in self.vertices:
            if vertex["data"] == vertex_data:
                found = True
                print(f"Vertex {vertex_data} found")
                print("Its edges are:")
                for edge in vertex["edges"]:
                    print(f"{edge['start']['data']} --- {edge['end']['data']}")
                break
        if not found:
            print("Vertex not found")


graph = Graph()

vertex_a = graph.add_vertex(10)
vertex_b = graph.add_vertex(20)
vertex_c = graph.add_vertex(30)
vertex_d = graph.add_vertex(40)
vertex_e = graph.add_vertex(50)
vertex_f = graph.add_vertex(60)

graph.add_edge(vertex_a, vertex_e)
graph.add_edge(vertex_a, vertex_d)
graph.add_edge(vertex_a, vertex_b)
graph.add_edge(vertex_b, vertex_c)
graph.add_edge(vertex_c, vertex_d)
graph.add_edge(vertex_e, vertex_f)
graph.add_edge(vertex_f, vertex_d)
```

```
graph.print_graph()
print("\n")
print("\n")
# Search for specific vertex
graph.search_data(30)
print("\n")
graph.search_data(40)
```

Output :

```
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 10\8 may 2023> python -u "e:\Semester 4\Data Structure and Algorithm\Lab\Lab 10\8 may 2023\task_222.py"
Vertex: 10
Edges:
10 --- 50
10 --- 40
10 --- 20

Vertex: 20
Edges:
10 --- 20
20 --- 30

Vertex: 30
Edges:
20 --- 30
30 --- 40

Vertex: 40
Edges:
10 --- 40
30 --- 40
60 --- 40

Vertex: 50
Edges:
10 --- 50
50 --- 60

Vertex: 60
Edges:
50 --- 60
60 --- 40



Vertex 30 found
Its edges are:
20 --- 30
30 --- 40


Vertex 40 found
Its edges are:
10 --- 40
30 --- 40
60 --- 40
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 10\8 may 2023
```

Task 3: Make a graph given in figure using adjacency matrix structure.

Code :

```
class Graph:
    def __init__(self):
        self.vertices = []

    def add_vertex(self, data):
        vertex = {"data": data, "edges": []}
        self.vertices.append(vertex)
        return vertex

    def add_edge(self, start, end):
        edge = {"start": start, "end": end}
        start["edges"].append(edge)
        end["edges"].append(edge)

    def print_graph(self):
        num_vertices = len(self.vertices)
        adjacency_matrix = [[0] * num_vertices for _ in range(num_vertices)]

        # Create a mapping from vertex to its index in the adjacency matrix
        vertex_to_index = {vertex["data"]: index for index, vertex in enumerate(self.vertices)}

        for vertex in self.vertices:
            for edge in vertex["edges"]:
                start_index = vertex_to_index[edge['start']['data']]
                end_index = vertex_to_index[edge['end']['data']]
                adjacency_matrix[start_index][end_index] = 1
                adjacency_matrix[end_index][start_index] = 1

        # Print the adjacency matrix
        print("Adjacency Matrix:")
        for row in adjacency_matrix:
            print(row)

        print()

graph = Graph()

vertex_a = graph.add_vertex(10)
vertex_b = graph.add_vertex(20)
vertex_c = graph.add_vertex(30)
vertex_d = graph.add_vertex(40)
vertex_e = graph.add_vertex(50)
vertex_f = graph.add_vertex(60)

graph.add_edge(vertex_a, vertex_e)
graph.add_edge(vertex_a, vertex_d)
graph.add_edge(vertex_a, vertex_b)
graph.add_edge(vertex_b, vertex_c)
graph.add_edge(vertex_c, vertex_d)
graph.add_edge(vertex_e, vertex_f)
graph.add_edge(vertex_f, vertex_d)

graph.print_graph()
```

Output :

```
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 10\8 may 2023> python -u "e:\Semester 4\Data Structure and Algorithm\Lab\Lab 10\8 may 2023\Task_3.py"

Adjacency Matrix:
[0, 1, 0, 1, 1, 0]
[1, 0, 1, 0, 0, 0]
[0, 1, 0, 1, 0, 0]
[1, 0, 1, 0, 0, 1]
[1, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 1, 0]

PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 10\8 may 2023>
```

Lab Evaluation Rubrics							
Domain	**CLOs/ Rubric**	**Performance Indicator**	**Unsatisfactory 0-2**	**Marginal 3-5**	**Satisfactory 6-8**	**Exemplary 9-10**	**Allocate d Marks**
Psychomotor	**CLO:1 R2**	Implementation with Results **(P)**	Does not try to solve problems. Many mistakes in code and difficult to comprehend for the instructor. There is not result of the problem.	Does not suggests or refine solutions but is willing to try out solutions suggested by others. Few mistakes in code, but done along with comments, and easy to comprehend for the instructor. Few mistake in result.	Refines solutions suggested by others. Complete and error-free code is done. No comments in the code, but easy to comprehend for the instructor. Results are correctly produced.	Actively looks for and suggests solution to problems. Complete and error free code is done, easy to comprehend for the instructor. Results are correctly produced. Student incorporated comments in the code.	
Affective	**CLO:3 R3**	Lab Report **(A)**	Code of the problem is not given. Outputs are not provided. Explanation of the solution is not stated.	Code of the problem is given. Output is not complete. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is satisfactory.	
	CLO:1 R5	Discipline and Behavior **(A)**	Got and wandered around. More than two incidents of talking non-lab related stuff in lab and/or any talk with other groups, voice level exceeding the appropriate level, use of cell phones and involvement in any non-lab activity.	Got out of seat and wander around for some time. No more than two incidents of talking non-lab related stuff in lab. Voice level exceeding the appropriate level, use of cell phones and involvement in any non-lab related activity.	Stayed in seat and got up for a specific lab related reason, but took more time than required to do the job. No more than one incidents of talking non-lab related stuff in lab. Voice level exceeding the appropriate level, use of cell phones and involvement in any non-lab related activity.	Stayed in seat and got up for a specific lab related reason. Took care of lab related business and sat down right away. Voice level kept appropriate. Not used cell phones or involved in any non- lab related activity.	

V1: Designed by: Naureen

Shaukat