Zakeriya Muhumed
Program #5 Writeup
CS163

Graph data structures are essential because they may be used to represent a variety of real-world situations. Vertices and edges make up a graph, with edges signifying a relationship between two vertices. Graphs can be undirected or directed, with unidirectional edges in the former and bidirectional edges in the latter. I developed a program that builds and modifies a straightforward graph data structure. Users can add vertices and edges, and the program displays the graph in different ways. I now have a better understanding of the fundamental ideas underlying graph data structures and the algorithms employed to manage them thanks to the program.

A blank graph is first created by the application. After that, users can add vertices by giving each a special identification. Each vertex is saved by the application in a hash table for quick retrieval. By supplying the identities of the vertices that the edge connects, users can also insert edges. A new edge object is created by the software, and it is added to a list of edges. How to present the graph in a way that was user-friendly was one of the difficulties I ran into when putting the program into practice. The adjacency matrix representation, which shows the graph as a two-dimensional array, is what I choose to employ. The entries in the array show if there is an edge between two vertices, and the rows and columns of the array correspond to the vertices.

The program loops through the list of edges in order to display the graph, setting the relevant entries in the adjacency matrix. The matrix is then printed to the console by the software. Users can easily determine which vertices are related and how with this depiction. The adjacency list was a different graph representation I used. Each vertex in this form has a list of neighbors linked with it. The software builds a dictionary with a list of neighbors for each vertex as the value for each key. The software just iterates through the dictionary and publishes each vertex together with a list of its neighbors to display the graph as an adjacency list.

I now have a better grasp of graph data structures and their uses as a result of working on this program. The curriculum emphasizes the significance of selecting the appropriate

representation for a particular problem because various representations might provide various advantages in terms of effectiveness and usability. The assumption that algorithms and data structures are fundamental ideas in computer science and that a thorough comprehension of them is necessary for efficient problem-solving and programming has also been reaffirmed. The program's inability to cope with complex graphs—those with more than one edge connecting any two vertices—is one of its drawbacks. Further data structures and algorithms would be required to accommodate more complicated graphs, such as those with multiple edges or loops. Nonetheless, the program gives me a better understanding of other data structures.

In conclusion, anyone wishing to hone their programming and data structure skills will find that creating a program for a straightforward graph data structure that adds vertices and edges and displays them is a great exercise. Geographical maps, social networks, and computer networks are just a few of the many uses for graphs. For creating applications that use graphs, it is crucial to comprehend the fundamental ideas of graph data structures, such as vertices, edges, and adjacency matrices. A great exercise for people wishing to improve their programming abilities is to implement a graph data structure program, which necessitates a solid grasp of programming languages, algorithms, and data structures.Overall, one may learn more about how graphs function and how to use them in practical applications by creating a program for a straightforward graph data structure.