8. Lab 7: Hash Tables

8.1. Objectives

- Understand and implement Hash table

8.2. Problem

HashDoubleApp

- Display the key sequence for the initial filling of the table
- Display the hash value, the step, and the probe sequence for insert and find.
- Display the probe length for each find and insert
- Display the average probe length for the initial filling of the table
- Investigate how the load factor affects the average probe length
- Demonstrate the importance of using a prime number for the table size

HashChainApp

- Display the key sequence for the initial filling of the table
- Display the probe length for each find and insert
- Display the average probe length for the initial filling of the table
- Investigate how the load factor affects the average probe length

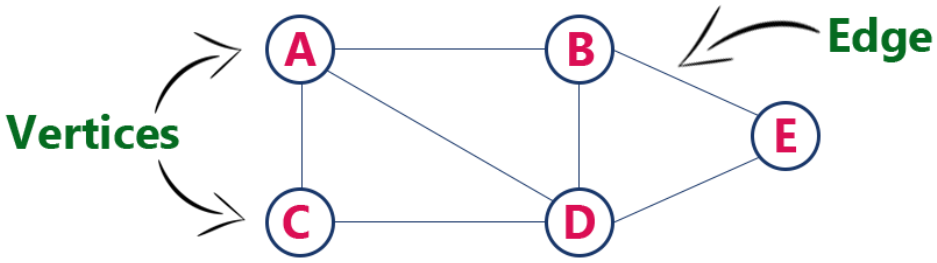
9. Lab 8: Graph

9.1. Objectives

- Understanding Graph Data Structures
- Understanding Dijkstra's Algorithm
- Know how to use Graph and Dijkstra's Algorithm to solve problems.

9.2. Problems

A **Graph** data structure can be understood as a network of connections between numerous points. These points are termed **Vertices** and the links between points are called **Edges**. As a result, a graph g is defined as a collection of vertices V and edges E that connect them.



Graphs are commonly used to visualise networks such as computer networks, social networks, maps (a network of locations), social relationships, data relationships, etc. They can also be used to indicate various software or architecture dependencies.

Useful References:

- https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/
- https://www.tutorialspoint.com/data_structures_algorithms/graph_data_structure.htm

Dijkstra's algorithm is a well-known algorithm used to find the shortest distance from the source node to all the other nodes in a weighted graph.

As a result of running Dijkstra's algorithm on a graph, we obtain the shortest path tree (SPT) with the source vertex as the root.

In Dijkstra's algorithm, we maintain two sets or lists. One contains the vertices that are a part of the shortest path tree (SPT), and the other contains vertices that are being evaluated to be included in SPT. Hence for every iteration, we find a vertex from the second list with the shortest path.

Useful References:

- https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
- https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/

Simple Map Modeling Problem

One of the most useful applications on mobile devices is Map (Google Maps/Apple Maps/Open Maps/Bing Maps…). Every day, we use maps to navigate from place to place, to find the fastest route, to locate our destination, or to look for gas stations, shopping malls, coffee shops… But have you

ever wondered how those maps are developed? After learning about Graph, you may consider a map as a network of connected locations (each location may have another internal network of related information). In this section, we will use Graph and other knowledge you learned during this course to create a very naive version of Map and then try to solve the common cases: finding the shortest path, the fastest path, and all locations near a specific location.

Task 1: Read the provided references and textbook to understand the Graph data structure.

Task 2: Implement Graph data structure in Java. Your project should have at least four classes: **Graph**, **Node** (or **Vertex)**, **Edge**, and **MapApp** (to test your solution).

Note: For the basic problem, we assume that every path has a fixed length and is a two-way path. Therefore, we will use an ***undirected and weighted graph****.*

Task 3: Use your implementation to form the graph in Figure 1

Task 4:

- Print out the number of paths to go from **A** to **K**
- Print out all the paths with the smallest and largest number of nodes from **A** to **K** and the corresponding cost (total weight).

Hint: use Depth First Search/Breadth First Search to go through the map

Task 5:

Apply *Dijkstra's Algorithm* to*:*

- Find the shortest path to go from **A** to **J**
- Find the shortest path to go from **B** to **L**