The Java programs created a customized data structure for storing various points in the Halifax map into it in the form of various nodes of coordinates. It defines the path summary of the formatting requirements is in the CSCI 3901 course assignment #3 information in the course's bright space.

The Java program provides a path between two points with the map and gives the shortest path between two given points. The data structure used adjacency list

Program flow:-

- **new Intersections**(Node) in the map are created based on the given coordinates and stored in a map.
- **define Road(**points) is used to define the route between nodes that are already present.
- **navigate(**source, destination**)** method is used to print the shortest path between the two given source and destination nodes.

External Java files:-

- **City.java -** used to store the coordinates of the node in the map. Each node is stored as an object of type City
- **Edge.java -** Used to store the route between the two nodes and the distance between them. All the routes are stored in the form of type Edge with a source node and destination node.
- **Graph.java** - Used to store the map of the Halifax city. It contains all the routes between nodes.

Modules of Program development:-

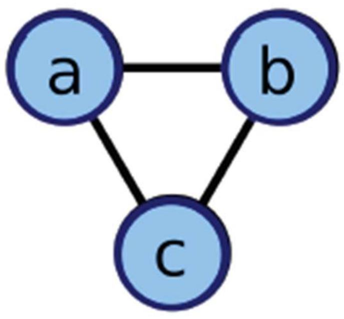
- Store all the given points/coordinates in the form of City class type and stored in ArrayList.
- Accept two node points of the city and define the road between the nodes.
- Calculate the distance between two node points and store source and destination.
- Store the paths in a Linked List array.
- Each node in the array has a linked list containing the paths.
- Traverse all the nodes and get the shortest path to all the nodes from every node.
- Accept the source and destination nodes and find all the possible paths and total.
- Print the shortest path between paths along with the destination.

Assumptions:-

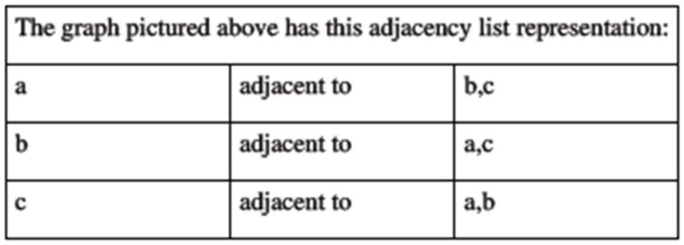
- Individual nodes without any routes defined should not have any path.
- Coordinates can be negative numbers.
- Integer coordinates are given as inputs.
- Duplicate nodes are not accepted.
- The identical point as source and destination has no path.

Data Structure and design:-

In graph theory, an **adjacency list** is the representation of all edges or arcs in a graph as a list. If the graph is undirected, every entry is a set (or multiset) of two nodes containing the two ends of the corresponding edge; if it is directed, every entry is a tuple of two nodes, one denoting the source node and the other denoting the destination node of the corresponding arc. Typically, adjacency lists are unordered.



This undirected cyclic graph can be described by the list {a,b}, {a,c}, {b,c}.



The graph pictured above has this adjacency list representation:		
a	adjacent to	b,c
b	adjacent to	a,c
c	adjacent to	a,b

In computer science, an adjacency list is a data structure for representing graphs. In an adjacency list representation, we keep, for each vertex in the graph, a list of all other vertices which it has an edge to (that vertex's "adjacency list"). For instance, the representation, in which a hash table is used to associate each vertex with an array of adjacent vertices, can be seen as an example of this type of representation. Another example is the representation in which an array indexed by vertex numbers points to a singly-linked list of the neighbors of each vertex. One difficulty with the adjacency list structure is that it has no obvious place to store data associated with the edges of a graph, such as the lengths or costs of the edges. So we used other classes to store the weights

Limitation:-

- It accepts only coordinates.
- Prints only the shortest path, but not all other paths.
- Prints the path but not the individual distance for each node.
- Keys are not case sensitive.