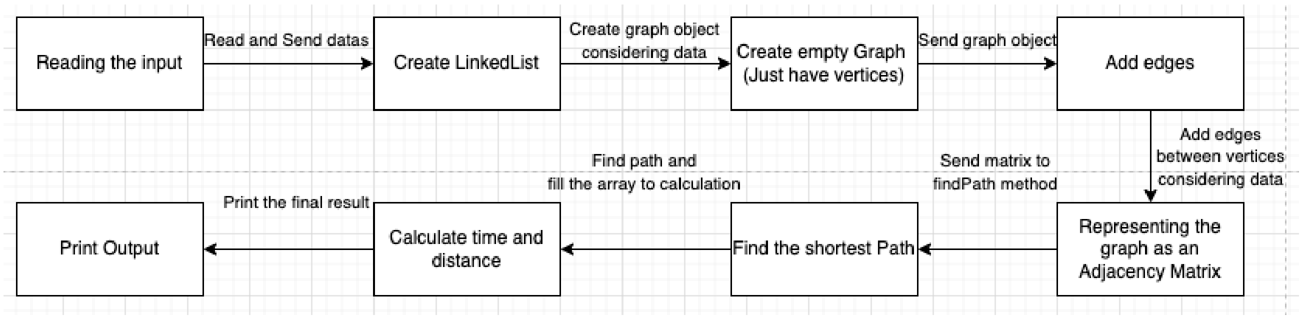
PROGRAMMING ASSIGNMENT 1

Question 1

Problem Statement and Code Design

In this question, I have properly created the algorithm of the flight system using the non-directional graph structure. I implemented the chart myself, using the bag structure. While writing the algorithm that meets the desired conditions in the question, I followed the order you see in the table below and applied it. As a result, it finds the shortest path to the desired start and end points. Also suppresses the duration and distance of the path.


Reading the input
Read and Send datas
Create LinkedList
Create graph object considering data
Create empty Graph (Just have vertices)
Send graph object
Add edges
Add edges between vertices considering data
Representing the graph as an Adjacency Matrix
Send matrix to findPath method
Find the shortest Path
Find path and fill the array to calculation
Calculate time and distance
Print the final result
Print Output

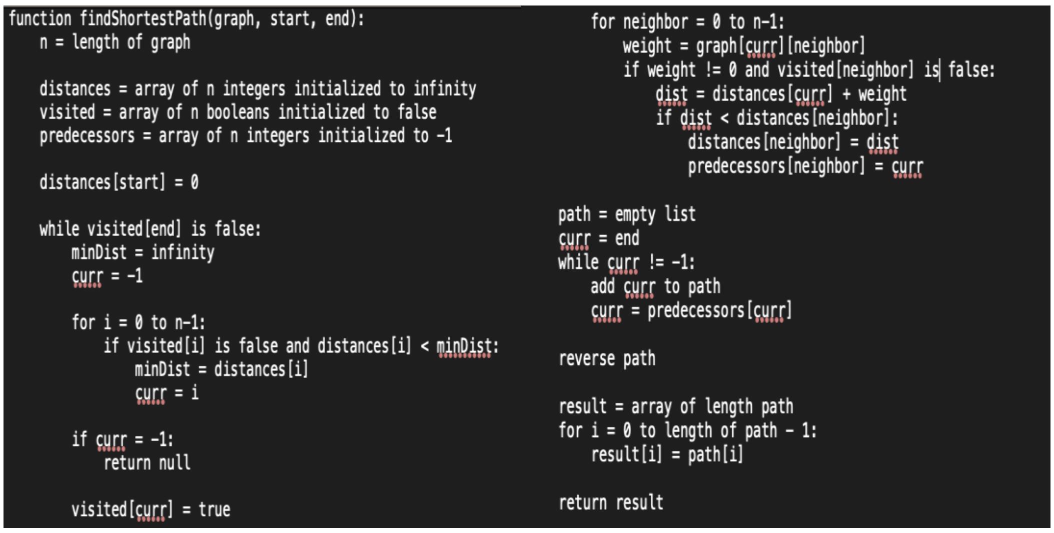
Implementation & Functionality

In my implementation, there are 3 classes; their names are Graph, Bag, and Main. Briefly, I read the input and kept the necessary information. Based on this information, I create my graph and then add the edges. By creating the adjacency matrix of this graph, I return my path through my FindShortestPath method.

1. **Reading the input:** Read all input with the scanner, create a txt file with these data, then buffered reader again reads input and sends it to LinkedList.
2. **Create graph:** It creates an empty graph with N vertices by using the graph(int v) constructor.
3. **Add edges:** Using the addEdge(int v, int w) method, it adds the edges between the vertices in our graph object, considering input.
4. **Creating Adj Matrix by graph:** The convert (Bag<Integer>adj[],int v) method provides to represent as a matrix by our graph. It returns the 2d array. The method logic is checking the adj array for every vertex, and it is marked as 1 if there is a connection.
5. **Find Shortest Path:** This method finds the shortest path between two vertices in a weighted graph using Dijkstra's algorithm. The method parameters are adj matrix, start-end point (int [][] graph, int start, int end). It initializes arrays to store distances, visited status, and predecessors. The algorithm finds the unvisited vertex with the smallest tentative distance, marks it as visited, and updates the distances and predecessors of its neighbors if a shorter path is found. It repeats until all vertices have been visited or the end vertex is reached. The method reconstructs the shortest path using the predecessor array and returns an array of the vertices on the path.
6. **Calculate time and find Number of City (distance):** I didn't create a method for calculating that. After found path the size of array gives the number of city. To find

time, g.time-(g.cTime - g.time)) * (result.length-2) + g.cTime*(result.length - 1) I applied this calculation.

7. **Print output**: Lastly the code prints respectively number of city, shortest path, total time.



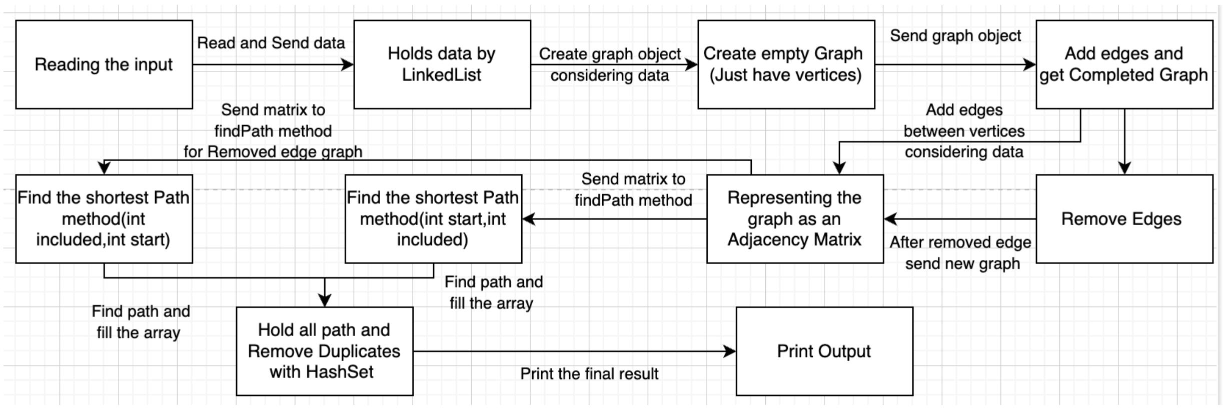
```
function findShortestPath(graph, start, end):
    n = length of graph

    distances = array of n integers initialized to infinity
    visited = array of n booleans initialized to false
    predecessors = array of n integers initialized to -1

    distances[start] = 0

    while visited[end] is false:
        minDist = infinity
        curr = -1

        for i = 0 to n-1:
            if visited[i] is false and distances[i] < minDist:
                minDist = distances[i]
                curr = i

        if curr = -1:
            return null

        visited[curr] = true

        for neighbor = 0 to n-1:
            weight = graph[curr][neighbor]
            if weight != 0 and visited[neighbor] is false:
                dist = distances[curr] + weight
                if dist < distances[neighbor]:
                    distances[neighbor] = dist
                    predecessors[neighbor] = curr

    path = empty list
    curr = end
    while curr != -1:
        add curr to path
        curr = predecessors[curr]

    reverse path

    result = array of length path
    for i = 0 to length of path - 1:
        result[i] = path[i]

    return result
```

findShortestPath() Pseudocode

Question 2

Problem Statement and Code Design

For the second part of the question, I used a similar solution to question 1. I used same structures graph and bag. But unlike the first problem, in this problem my findpath method needed to go from the starting point to a circle with a starting point instead of going from the starting point to the end point, and that circle had to pass through a certain point, that is, contain it.


Reading the input
Read and Send data
Holds data by LinkedList
Create graph object considering data
Create empty Graph (Just have vertices)
Send graph object
Add edges and get Completed Graph
Send matrix to findPath method for Removed edge graph
Add edges between vertices considering data
Find the shortest Path method(int included,int start)
Find the shortest Path method(int start,int included)
Send matrix to findPath method
Representing the graph as an Adjacency Matrix
Remove Edges
After removed edge send new graph
Find path and fill the array
Find path and fill the array
Hold all path and Remove Duplicates with HashSet
Print the final result
Print Output

Implementation & Functionality

Instead of creating an algorithm from scratch, I found this solution by calling the FindShortestPath() method, which I used in the first question, with different parameters and additionally by implementing the removeEdge() method and using HashSet. Thus, they will not use the same path between them again. Also, if I need to explain the logic of the solution, I created a graph, considering the data, as I did in the first question. I then first called the path method between the start point and the included point like this: FindShortestPath(int startpoint,

int includedpoint, adj matrix [][]). Then I remove the edge between this path and the other because they shouldn't be using the same path. Then call FindShortestPath (int includedpoint, int startpoint, and adj matrix [][]) again with the same parameters, but their order is changed. But to complete the circle, we used two paths and merged them.

1. **Reading the input:** Same as Question 1
2. **Create graph:** Same as Question 1
3. **Add edges:** Same as Question 1
4. **Creating Adj Matrix by graph:** Same as Question 1
5. **Find Shortest Path:** Same as Question 1
6. **Remove Edges:** This method provides for removing the edge between two vertices. Its parameters are (int v, int w), and in the implementation of this method, it uses the remove method of the bag class. This remove method provides for the removal of a node from a singly linked list that contains integers. I used this method to call the findShortestPath method twice, and it uses the first path again for going to the start point.
7. **Merged Two Path to Get Circle:** I took two path the first one is includes "start point" to "include point", the second path includes "include point" to "start point" but this path not same with first one because I used removeEdge() method for first path so it canceled to prefer to go same path. The merge part consists of resizing the array and merging it. Then, I used a hash set to remove duplicate elements and I used Array.sort() to make the array be sorted.
8. **Print output**: Lastly, the code prints the correct path or as we can say circle in orderly.

   I didn’t add again pseudocode because the method is the same as in the first question.

<u>Testing and final assessment parts are common for both questions 1 and 2.</u>

Testing

When I started doing homework, I assumed our input was in text file format, so we needed to read it. But after tested in VPL, I got it to it should take with scanner line by line. So instead of changing my code, I wrote an extra code for doing this. First, I scanned it and then created a text file; then, I read this txt file, which holds data, and held it in a linked list. After I solved it, all the test cases passed successfully.

FINAL ASSESSTMENTS

1. I learned the logic of finding the shortest path using algorithms such as depth-first search (DFS), breadth-first search (BFS), and Dijkstra’s algorithm.
2. I understood the importance of the adjacency list. Also, I learned how to create an adjacency matrix with adjacency list reference and number of vertices of graph.
3. I understood how the graph structure can be used while doing all these algorithms and implementations.