**ENGINEERING METHOD**

# **PROBLEM:**

In the year 2050, a global epidemic of highly contagious diseases has forced the government of Valle del Cauca to implement an evacuation strategy between quarantine centers to save the affected population. Cali's medical response team, known as "Phoenix," is charged with finding the safest and most efficient route to transport people from one quarantine center to another.

To accomplish this mission, Phoenix has identified more than 50 possible evacuation routes, each with different amounts of supplies needed. It is important to keep in mind that between two locations you can go in any direction and there is only a single road between them.

*The needs of the Phoenix team are as follows:*

* Update the evacuation route based on the current situation: as the epidemic spreads or subsides in certain areas, it is necessary to adjust the supplies needed for the route to ensure the safety of the evacuees.
* Calculate the number of intermediate locations between two given locations.
* Find the evacuation route that requires the least number of medical supplies.
* Plan the logistics of patient transfers between quarantine centers. Phoenix must be able to know the optimal configuration of all routes between quarantine centers based on the resources needed to design efficient patient transfer strategies.

The government has engaged its software team to develop an IT solution that meets all of Phoenix's needs, providing analytical capabilities and enabling efficient and safe evacuations.

# **PHASE 1: PROBLEM IDENTIFICATION.**

## *Problem context:*

In the year 2050, a worldwide epidemic of highly contagious diseases has created the need to assist the affected population to quarantine centers. The "Phoenix" medical response team must find the safest and most efficient route to transport people between quarantine centers. To accomplish this, they must update the route based on the current situation, find the route with the least use of medical supplies, and help plan the logistics of the transfers.

## *Definition of the problem:*

Phoenix requires a system that can find the safest and most efficient evacuation routes, ensuring optimal utilization of medical supplies.

## *Identification of needs and symptoms:*

* Update the evacuation route based on the current situation of the epidemic, ensuring the safety of the evacuees.
* Calculate the number of intermediate locations between one location and another.
* Find the evacuation route that requires the least possible number of medical supplies.
* Determine the main route that passes through all locations.

*Problem specification:*

***Client:*** Valle del Cauca government

***Users:*** Phoenix members.

***Functional requirements:***

* The system must allow:

FR1. Update each evacuation route.

FR2. Calculate the number of intermediate locations between one location and another.

FR3. Find an evacuation route between two locations.

FR4. Determine the main route that passes through all locations.

***Non-functional requirements:***

* The system must:

*NFR1.* Be efficient in the search for routes.

*Specification of requirements:*

**FUNCTIONAL REQUIREMENTS ANALYSIS TABLE (FR1)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name or identifier** | **FR1. Update each evacuation route.** | | |
| **Summary** | The system should allow the user to update the information of the routes between quarantine centers based on the current situation of the epidemic and based on the number of medical supplies required to move people from one to another. | | |
| **Inputs** | **Input name** | **datatype** | **Selection or repetition condition** |
| initialLocation | String |  |
| finalLocation | String | Must have a direct connection between the initial location |
| requiredSupplies | int |  |
| **General activities necessary to obtain the results** | 1. Read the initial location. 2. Read the target location. 3. Verify that the locations are adjacent. 4. Update the medical supplies on the route that connects the starting quarantine center to the destination quarantine center. | | |
| **Result or post-condition** | The evacuation route is updated. | | |
| **Outputs** | **Output name** | **datatype** | **Selection or repetition condition** |
| none | none | none |

**FUNCTIONAL REQUIREMENTS ANALYSIS TABLE (FR2)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name or identifier** | **FR2. Calculate the number of intermediate locations between one quarantine center and another.** | | |
| **Summary** | The system should make it possible to determine the number of quarantine centers between one quarantine center and another. | | |
| **Inputs** | **Input name** | **datatype** | **Selection or repetition condition** |
| initialLocation | String |  |
| targetLocation | String |  |
| **General activities necessary to obtain the results** | 1. Read the initial location. 2. Read the target location. 3. Find the number of intermediate locations to the target quarantine center. 4. Deploy the distance between the starting quarantine center and the target. | | |
| **Result or post-condition** | The distance or number of minimum locations separating one quarantine center from the other is deployed. | | |
| **Outputs** | **Output name** | **datatype** | **Selection or repetition condition** |
| numberOfLocations | int |  |

**FUNCTIONAL REQUIREMENTS ANALYSIS TABLE (FR3)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name or identifier** | **FR3. Find an evacuation route between two locations.** | | |
| **Summary** | The system must search for the route of evacuation of patients from one quarantine center to another that demands the minimum number of medical supplies. | | |
| **Inputs** | **Input name** | **datatype** | **Selection or repetition condition** |
| initialLocation | String |  |
| finalLocation | String |  |
| **General activities necessary to obtain the results** | 1. Read the initial location. 2. Read the target location. 3. Check the evacuation route options available between the centers of quarantine. 4. Find the route that requires the least number of medical supplies to transport patients. | | |
| **Result or post-condition** | The user can display the evacuation route that demands the minimum number of supplies. | | |
| **Outputs** | **Output name** | **datatype** | **Selection or repetition condition** |
| none | none | none |

**FUNCTIONAL REQUIREMENTS ANALYSIS TABLE (FR4)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name or identifier** | **FR4. Determine the primary route through all locations.** | | |
| **Summary** | The system must allow to find an optimal configuration of the routes to go through all the quarantine centers with the least amount of the required resources. | | |
| **Inputs** | **Input name** | **datatype** | **Selection or repetition condition** |
| none | none | none |
| **General activities necessary to obtain the results** | 1. Find the route to go through all quarantine centers that require fewer medical supplies. 2. Deploy the optimal configuration of routes between quarantine centers based on the resources required. | | |
| **Result or post-condition** | The user can display the most optimal main evacuation route on the screen. | | |
| **Outputs** | **Output name** | **datatype** | **Selection or repetition condition** |
| none | none | none |

# **PHASE 2: COLLECTION OF THE NECESSARY INFORMATION.**

To solve this problem, it is necessary to know where the quarantine points will be located. Below is a map of the department of Valle del Cauca where the quarantine centers should be located based on the information provided by Phoenix to the development team.

Taken from: https://www.contraloriavalledelcauca.gov.co/publicaciones/276/informacion-para-ninos-ninas-y-adolescentes/

Also, it is necessary to use some programming concepts and tools. Below is some information that may be useful for the implementation of the system:

1. Graph Theory: Graph theory provides a set of concepts, definitions, and algorithms to analyze and solve problems related to graphs.
   1. **Graph:** graph is a data structure that stores interconnected data. In short, a graph G is composed of vertices (V) and edges (E) that connect them. Examples of graphs include social networks, computer networks, Google Maps, among others.
   2. **Vertex:** A node or point in a graph that represents a location or a specific point of interest.
   3. **Edge:** Represents a connection between two vertices, indicating a possible evacuation route.

Taken from: [Java Graph - Javatpoint](https://www.javatpoint.com/java-graph)

1. Graph Traversal Algorithms: Graph traversal algorithms are techniques used to visit all the nodes or edges of a graph. These algorithms allow for exploration and searching of information within the graph’s structure:
   1. **Breadth First Search:** The Breadth First Search algorithm is a graph traversal algorithm that scans all nodes of a graph in tiers. Start from an initial node and visit all neighboring nodes before advancing to the next level. That is, it first scans all nodes at a distance of 1, then at a distance of 2, and so on.
   2. **Depth First Search:** The Depth First Search algorithm is a graph traversal algorithm that scans as deep as possible on a path before going backwards. Start from an initial node and keep going through one of the unvisited neighbors until there are no more neighbors to visit. Then go back to the previous node and continue the process until you have visited all reachable nodes from the initial node.

Taken from: [Difference between Breadth Search (BFS) and Deep Search (DFS) (encora.com)](https://www.encora.com/es/blog/dfs-vs-bfs)

1. Shortest path algorithms: Shortest path algorithms are used to find the shortest path between two nodes or locations in a weighted graph, that is, a graph in which the edges have an associated weight or cost.
   1. **Dijkstra Algorithm:** Dijkstra's algorithm is used to find the shortest path between a source node and all other nodes in a graph weighted with non-negative weights. Its complexity is O((V + E) log V).
   2. **Floyd Warshall Algorithm:** The Floyd Warshall algorithm is used to find the shortest path between all pairs of nodes in a weighted graph with positive or negative weights. Its complexity is O(V^3).

Taken from: [Algoritmos Dijkstra & Floyd-Warshall – Mi Camino Master](http://micaminomaster.com.co/grafo-algoritmo/algoritmos-dijkstra-floyd-marshall/)

1. Minimum spanning trees algorithms: Minimum spanning tree algorithms are used to find the subset of edges of a connected graph that connects all nodes with the lowest possible total cost, forming a tree without cycles.
   1. **Kruskal Algorithm:**  Kruskal's algorithm is a minimal spanning tree algorithm that seeks to find the subset of edges of lower weight that connects all nodes of a connected graph. Start by sorting all edges by weight, and then select them in ascending order, as long as you don't cycle with the previously selected edges. The complexity of Kruskal's algorithm is O(E log E).
   2. **Prim Algorithm:** Prim's algorithm is another minimal spanning tree algorithm that finds the subset of edges of lesser weight that connects all nodes of a connected graph. It starts by selecting an arbitrary node and, at each iteration, chooses the smallest edge that connects a selected node to an unselected node. The process continues until all nodes have been selected. The complexity of Prim's algorithm depends on the implementation used, but in its most efficient form, it has a complexity of O(V^2).

Taken from: [Minimum weight trees: Prim and Kruskal algorithms — Discrete Mathematics for Data Science (nekomath.com)](https://madi.nekomath.com/P5/ArbolPesoMin.html)

**PHASE 3: SEARCH FOR CREATIVE SOLUTIONS.**

For the implementation of a system that meets the needs of Phoenix there are different techniques. However, it is important to consider the scalability and efficiency of the program. Based on this, we have generated a series of ideas from a brainstorming session, a technique in which a group of people propose ideas freely and without restrictions, with the objective of generating creative and innovative solutions.

User interface:

To present the information to the user and fulfill the specified needs, the development team considers two alternatives:

1. Present the information by means of a console interface.
2. Present a graphical interface with a map in which all quarantine points are modeled.

Calculate the number of intermediate locations between one location and another:

To know the number of intermediate locations between one location and another, the following algorithms can be used:

1. Breadth First Search: This algorithm can be used to calculate the number of intermediate locations between one location and another. Take a width tour of the graph, exploring all adjacent nodes before moving on to the next levels.
2. Depth First Search: Similar to the search algorithm in width but exploring connections more deeply before going back. It can also help calculate the number of intermediate locations between one location and another.

Find an evacuation route between two locations:

Different algorithms can be used to find the evacuation route that requires the lowest cost of medical supplies, but we will focus on the two most important:

1. Dijkstra Algorithm: This algorithm finds the shortest path between two locations in a weighted graph. In this case, it could be used to find the evacuation route that requires as few medical supplies as possible. You would evaluate the cost (amount of supplies) of each possible route and select the route with the lowest total cost.
2. Floyd Warshall Algorithm: It is also a shortest path finding algorithm, but in this case all possible location pairs would be evaluated to find the optimal route in terms of required medical supplies.

Determine the primary route through all locations:

To plan logistics and find a main route that allows you to pass through all locations, it is possible to use the following algorithms:

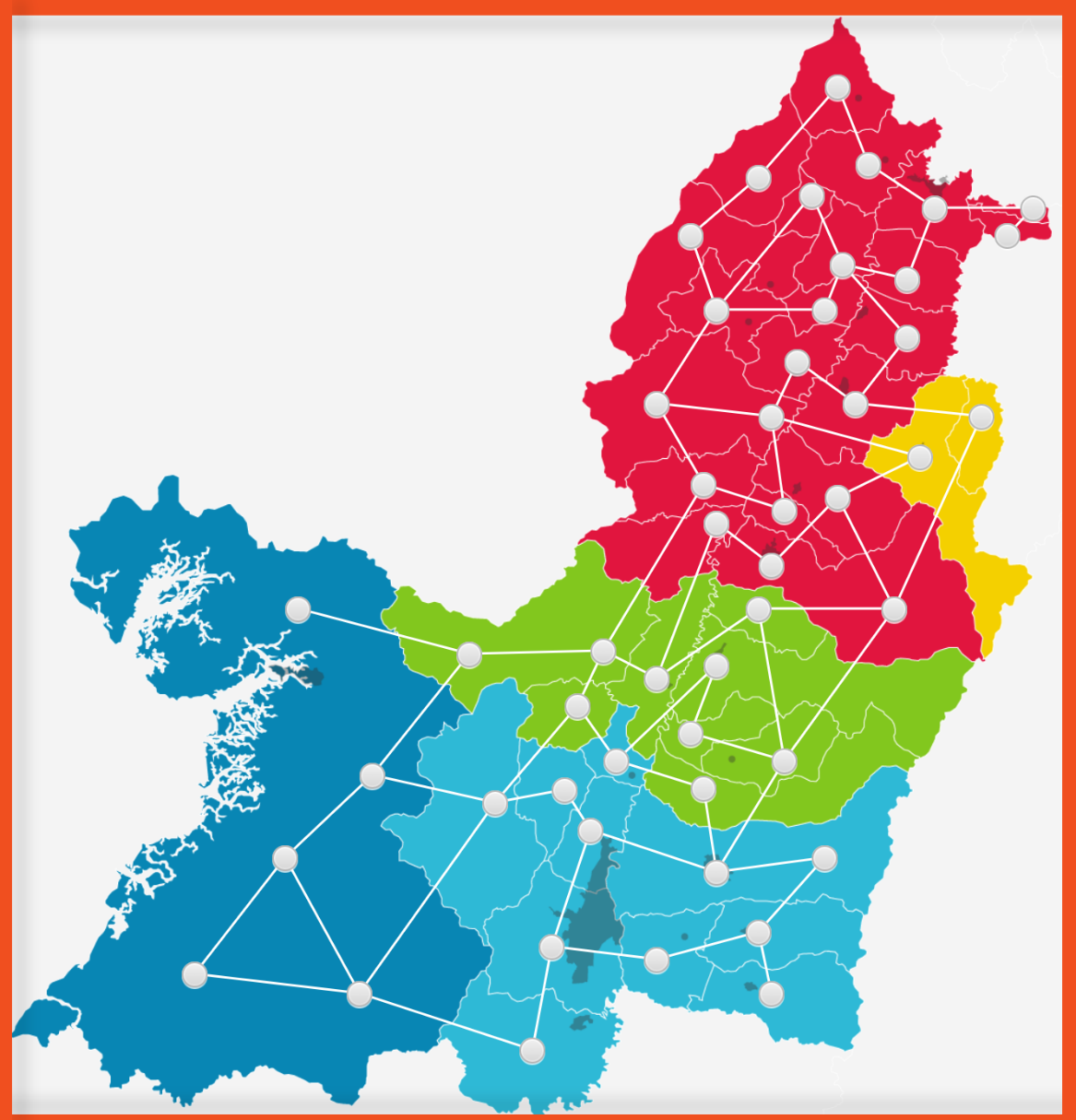
1. Kruskal Algorithm: This algorithm can be applied to find a minimum spanning tree that connects all affected locations. It will be useful in the situation where it is necessary to determine a main route that passes through all affected areas and requires as few medical supplies as possible.
2. Prim Algorithm: Like Kruskal's algorithm, Prim's algorithm can also be used to find a minimal spanning tree. In this case, it could be used to determine a primary route that covers all affected areas and requires the least number of medical supplies.

**PHASE 4: TRANSITION FROM IDEAS FORMULATION TO PRELIMINARY DESIGNS.**

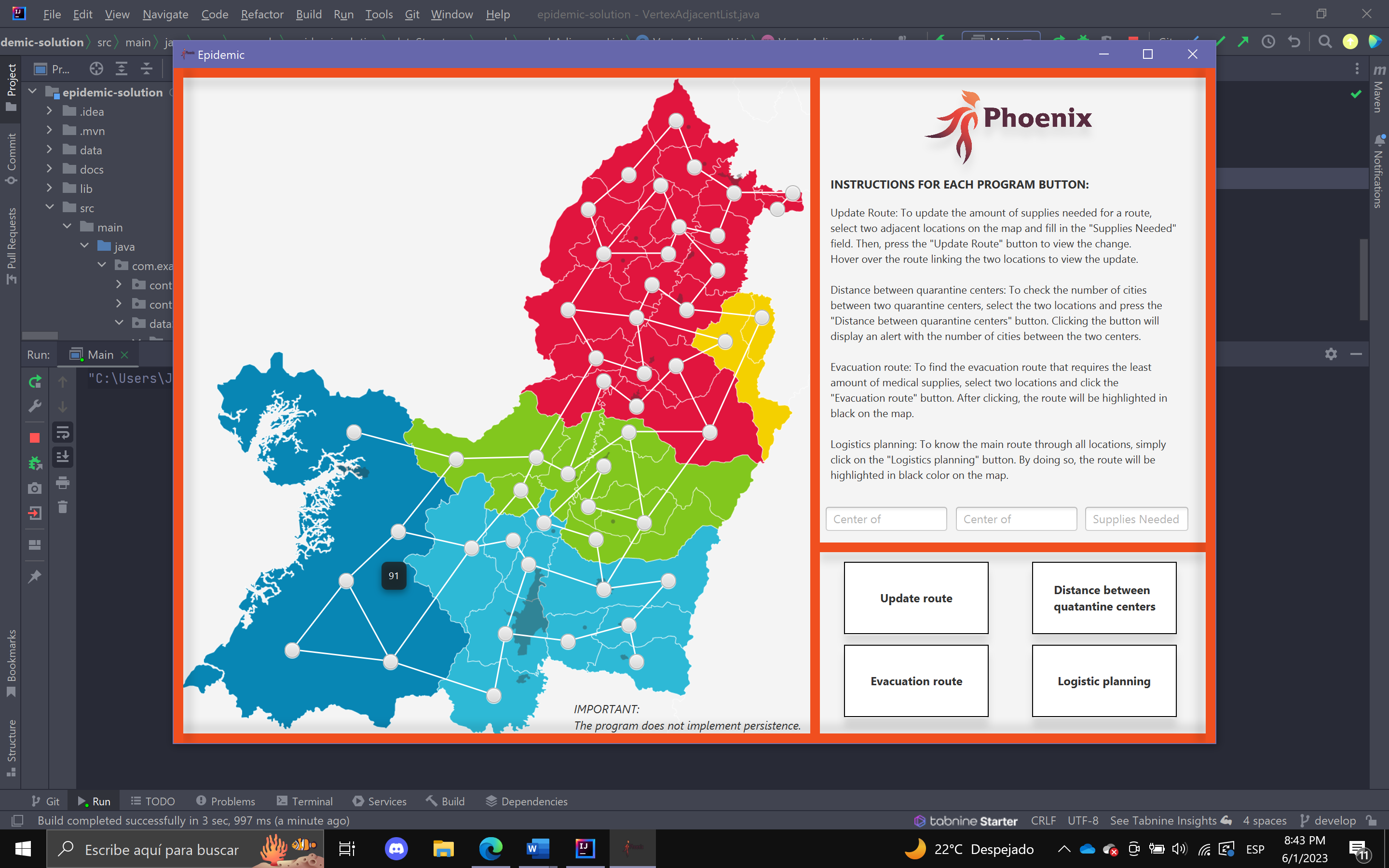
During phase 3 of the development process, several ideas were generated through a brainstorming session. Next, in phase 4, preliminary designs based on the ideas generated in the previous phase will be carried out.

User interface:

The information presented by the program must be accurate and easy to understand. In this order of ideas, displaying the resulting information through a console program can make it difficult to understand. For this reason, the development team has decided to show the information to the user through a graphical interface.

For this purpose, a map has been designed to show the location of the quarantine centers in Valle del Cauca. According to the data provided by Phoenix, each one of the 42 municipalities has its own assistance center. However, due to the size of some municipalities, it has been determined that several of them will have more than one quarantine center. A preliminary design of the graphic interface is shown below:

Where the routes and medical supplies to transport those affected can be modeled as follows:



Calculate the number of intermediate locations between one location and another:

1. Breadth First Search (BFS): This algorithm can be used to calculate the number of intermediate locations between two points. It starts from an initial location and gradually explores all its neighbors. During this process, we can keep track of the number of steps or levels required to reach the final location.
2. Depth First Search (DFS): Like BFS, this algorithm allows us to traverse the entire graph. However, since we need to explore from a specific location, DFS is not a viable option for implementation as it doesn't require an initial location to start its execution.

Find an evacuation route between two locations:

1. Dijkstra Algorithm: As described in the definitions section, Dijkstra's algorithm finds the shortest path between two locations. Its complexity is O(n^2), where "n" is the number of vertices in the graph. In this project, since we need to perform individual queries, this algorithm can be feasible.
2. Floyd Warshall Algorithm: This algorithm also finds the shortest path in a graph, but unlike Dijkstra's algorithm, it performs this procedure for every pair of nodes in the graph. This approach is not feasible as we are trying to create a program where the user makes individual queries to the program.

Determine the primary route through all locations:

1. Kruskal Algorithm: As described in the definitions section, Kruskal's algorithm returns a forest with minimum spanning trees of that graph by initially taking the shortest edge in a network and continuing step by step by adding edges to the tree. Therefore, it is useful for determining the main route that passes through all locations.
2. Prim Algorithm: It is very similar to Kruskal's algorithm in the sense that both return a minimum spanning tree. However, the difference is that Prim's algorithm returns a single minimum spanning tree generated from a root vertex. Given the above, this algorithm is not viable as the problem does not consider a starting point to determine the route.

**PHASE 5: EVALUATION AND SELECTION OF THE BEST SOLUTION**

Depending on the solutions proposed to the problem, some criteria are presented that may be relevant to evaluate the ideas generated in the brainstorming phase for the implementation of the system. It should be noted that the following criteria will only be used to evaluate the algorithms that will provide a solution to the needs of Phoenix, not the final design of the interface:

**Criterion A: Efficiency -** Refers to the ability of the system to process and deliver the required data and services quickly and without interruption. Efficiency is measured in terms of speed of response or by the time complexity of operations on a scale of 1 to 5, with 1 being very low efficiency and 5 being very high efficiency.

**Criterion B: Scalability -** Refers to the ability of the system to grow and evolve without restrictions, and to meet future business or user needs without compromising its quality or stability. It can be evaluated on a scale of 1 to 5, with 1 being very low scalability and 5 being very high scalability.

**Criterion C: Development time -** Refers to the time to complete the development of a specific system or functionality. This criterion is affected by factors such as the complexity of the system and the urgency of the airline to have the system in place. It can be evaluated on a scale of 1 to 5, with 1 being too much time and 5 being too little time.

Evaluation of ideas, where the highest score is the best alternative:

Calculate the number of intermediate locations between one location and another:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Alternative | Criterion A | Criterion B | Criterion C | Total |
| Breadth First Search | 5 | 4 | 5 | 14 |
| Depth First Search | 4 | 4 | 3 | 11 |

Find an evacuation route between two locations:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Alternative | Criterion A | Criterion B | Criterion C | Total |
| Dijkstra Algorithm | 5 | 4 | 4 | 13 |
| Floyd Warshall Algorithm | 3 | 3 | 3 | 9 |

Determine the primary route through all locations:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Alternative | Criterion A | Criterion B | Criterion C | Total |
| Kruskal Algorithm | 5 | 3 | 3 | 11 |
| Prim Algorithm | 3 | 3 | 3 | 9 |

Based on the above evaluation, the best alternatives are:

Calculate the number of intermediate locations between one location and another: Breadth First Search.

Find an evacuation route between two locations: Dijkstra Algorithm.

Determine the main path that passes through all locations: Kruskal Algorithm.