**ENGINEERING METHOD – INTEGRATIVE TASK 2**

**Problem context:**

A train company has presented problems in its travels since they don’t have a map of their stations in which they operate, so they need a system that allows them and their drivers to find the shortest way to one station to another.

**Problem identification:**

Identification of needs and symptoms:

* The flight operators and pilots need to know de shortest way between the airports.
* The airline hasn’t a program who can solve this situation.
* The solution has to show them the route so that they know which one to take

Problem definition:

The airline needs software that allows them to find and print the shortest route.

**Collection of information**

*Definitions:*

*Font:*

[*https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/*](https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/)

[*https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/*](https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/)

[*https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/*](https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/)

[*https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/*](https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/)

**BFS, Breadth-First Search,** is a vertex-based technique for finding the shortest path in the graph. It uses a Queue data structure that follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue. It is slower than DFS.

**DFS,** [**Depth First Search**](https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/), is an edge-based technique. It uses the [Stack data structure](https://www.geeksforgeeks.org/stack-data-structure/) and performs two stages, first visited vertices are pushed into the stack, and second if there are no vertices then visited vertices are popped.

# **Dijkstra’s Shortest Path Algorithm:** Given a graph and a source vertex in the graph, find the **shortest paths** from the source to all vertices in the given graph. It generates a SPT (shortest path tree) with a given source as a root. Maintain two sets, one set contains vertices included in the shortest-path tree, other set includes vertices not yet included in the shortest-path tree. At every step of the algorithm, find a vertex that is in the other set (set not yet included) and has a minimum distance from the source.

**Kruskal’s Algorithm** Kruskal’s algorithm to find the minimum cost spanning tree uses the greedy approach. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

**Prim Algorithm** Prim’s algorithm always starts with a single node, and it moves through several adjacent nodes, to explore all the connected edges along the way. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST; the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

**Idea Generation**

In a meeting where generate the next ideas.

1. Make a graph with a DFS searching and a Dijkstra Algorithm, nodes as stations and edges as connections between those existing station. Then, the shortest route is printed on the console.
2. Make a graph with a DFS searching and a Dijkstra Algorithm, nodes as stations and edges as connections between those stations and that the user can add or remove stations or routes through a graphical interface.
3. Make a graph with a BFS searching and a Prim Algorithm, nodes ad stations and edges as connections between those stations added by the user through a list of nodes and its edges printed at console as instructions to reach the objective station.
4. Make a database with all the stations and its routes to other stations and printing a list to the user with the stations and routes that it supposes to follow.
5. Make a graph with a BFS searching and a Dijkstra Algorithm, nodes as stations and edges as connections between those existing station. Then, the shortest route is printed on the console.
6. Make a graph with a DFS searching and a Kruskal Algorithm, nodes as stations and edges as connections between those stations and that the user can add or remove stations or routes through a graphical interface.

**Preliminary Designs**

At this moment, we are going to discard **idea 4** because we need a data structure that relates some objects to others, in addition to the fact that we need data between these relationships, which this structure does not do efficiently.

We can also rule out **idea 6**, since Kruskal’s algorithm only counts the edges from smallest to largest and what we need is to connect 2 nodes by their edges.

After this revision:

*Alternative 1. DFS – Dijkstra Algorithm – Console print*

* Dijkstra's algorithm ensures the shortest distances between nodes by directly calculating the distance between node and node
* The DFS traversal is quick to find connections between adjacent by assembling a subtree.
* Only the requested path is printed in the console (For example: Station(X) -> Station(Y) -> Station(Z).

*Alternative 2. DFS – Dijkstra Algorithm – Graphic Interface*

* Dijkstra's algorithm ensures the shortest distances between nodes by directly calculating the distance between node and node
* The DFS traverse is fast for finding connections between nodes.
* A graphical interface is printed showing the printed graph and nodes as stations in addition to the route indicated on the map.

*Alternative 3. BFS – Prim Algorithm – Instruction prints*

* Prim's algorithm is used to find the edge of least weight between one node and another.
* The BFS traverse ensures the search for connections between nodes.
* Printing instructions tells the user exactly what steps to follow.

*Alternative 4. BFS – Dijkstra Algorithm – Console print*

* Dijkstra's algorithm ensures the shortest distances between nodes by directly calculating the distance between node and node
* The BFS traverse ensures the search for connections between nodes.
* Only the requested path is printed in the console (For example: Station(X) -> Station(Y) -> Station(Z).

**Evaluación y selección de la mejor solución**

*Criterion:*

1. *Criterion A: Faster and better mastered:* 
   1. *[2] Faster to search for adjacency.*
   2. *[1] Slower to look for adjacency.*
2. *Criterion B: Accuracy in searching for routes:*
   1. *[2] Always find the shortest path between peer nodes.*
   2. *[1] Find a shorter path for the entire graph*
3. *Criterion C: Impression of results:*
   1. *[3] Displays results in a user-understandable way.*
   2. *[2] Print the correct route exactly.*
   3. *[1] Displays the route in a technical and unreadable way.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *A* | *B* | *C* | *Result* |
| *Alternative 1* | *2* | *2* | *2* | *6* |
| *Alternative 2* | *2* | *2* | *3* | *7* |
| *Alternative 3* | *1* | *1* | *1* | *3* |
| *Alternative 4* | *1* | *2* | *2* | *5* |