**Engineering design process**

**Step 1: Problem identification**

**Problem Definition:** El conocido Metro de Londres cuenta con 289 estaciones y 11 líneas que conectan el circuito ferroviario, debido a la gran cantidad de opciones para viajar por la ciudad en este tipo de transporte, la mayoría de los usuarios del Metro han visto la necesidad de una aplicación que les ayude a encontrar la ruta más rápida para desplazarse hacia su punto de destino. Por suerte, Transport for London proporciona el tiempo de viaje aproximado entre cada estación del metro. De este modo, es de suma importancia utilizar esta información para proporcional una solución eficiente y amigable para los usuarios del metro de Londres.

**Identification of needs and symptoms:**

* El sistema debe ser eficiente debido a la gran cantidad de estaciones.
* El sistema debe contar con una interfaz amigable con el usuario debido a la gran cantidad de estaciones.
* El sistema debe permitirle al usuario ver las estaciones en el mapa del metro.
* El sistema va a ser utilizado por muchas personas.

**Step 2. Information Gathering**

**Graph:** A Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph is composed of a set of vertices( V ) and a set of edges( E ). The graph is denoted by G(E, V).

**Depth First Search:** DFS is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally, print the nodes in the path.

**Breadth First Search**: BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

**Dijkstra’s:** This algorithm makes a tree of the shortest path from the starting node, the source, to all other nodes (points) in the graph. Dijkstra's algorithm makes use of weights of the edges for finding the path that minimizes the total distance (weight) among the source node and all other nodes. This algorithm is also known as the single-source shortest path algorithm. It is important to note that Dijkstra’s algorithm is only applicable when all weights are positive because, during the execution, the weights of the edges are added to find the shortest path.

**Floyd-Warshall Algorithm** is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative).

**Spanning Tree:** A Spanning tree is a subset to a connected graph G, where all the edges are connected, i.e, one can traverse to any edge from a particular edge with or without intermediates. Also, a spanning tree must not have any cycle in it. Thus we can say that if there are N vertices in a connected graph then the no. of edges that a spanning tree may have is N-1.

**Minimum Spanning Tree:** Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

**Prim's Algorithm** is a greedy algorithm that is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized. Prim's algorithm starts with the single node and explores all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.

**Kruskal's Algorithm** is used to find the minimum spanning tree for a connected weighted graph. The main target of the algorithm is to find the subset of edges by using which we can traverse every vertex of the graph. It follows the greedy approach that finds an optimum solution at every stage instead of focusing on a global optimum.

**Bellman Ford Algorithm**: Bellman ford algorithm is a single-source shortest path algorithm. This algorithm is used to find the shortest distance from the single vertex to all the other vertices of a weighted graph. There are various other algorithms used to find the shortest path like Dijkstra algorithm, etc. If the weighted graph contains the negative weight values, then the Dijkstra algorithm does not confirm whether it produces the correct answer or not. In contrast to Dijkstra algorithm, bellman ford algorithm guarantees the correct answer even if the weighted graph contains the negative weight values.

**A\* Algorithm** is one specific pathfinding algorithm, first published in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael. It is generally considered to be the best algorithm to use when there is no opportunity to pre-compute the routes and there are no constraints on memory usage. Both memory and performance complexity can be O(b^d) in the worst case, so while it will always work out the most efficient route, it's not always the most efficient way to do so. A\* is actually a variation on Dijkstra's Algorithm, where there is additional information provided to help select the next node to use. This additional information does not need to be perfect – if we already have perfect information, then pathfinding is pointless. But the better it is, the better the end result will be.

**Johnson's algorithm** is a shortest path algorithm that deals with the all pairs shortest path problem. The all pairs shortest path problem takes in a graph with vertices and edges, and it outputs the shortest path between every pair of vertices in that graph. Johnson's algorithm is very similar to the Floyd-Warshall algorithm; however, Floyd-Warshall is most effective for dense graphs (many edges), while Johnson's algorithm is most effective for sparse graphs (few edges).

**Step 3. Search for Creative Solutions**

Para la generación de ideas creativas se realizó la técnica de lluvia de ideas donde se revisaron los conceptos investigados en la recolección de información.

**BFS:** Esta solución propone implementar un grafo dirigido y no ponderado. Las estaciones serian vértices y las vías que las conectan, aristas. Ahora, para encontrar el camino más corto se propone utilizar el algoritmo de BFS. Este algoritmo retornaría el número mínimo de estaciones que hay que recorrer para llegar a la estación de destino.

**Dijsktra´s algorithm:** Esta solución propone implementar un grafo dirigido y ponderado. Las estaciones serian vértices y las vías que las conectan, aristas. Las aristas tendrían un peso asociado que correspondería al tiempo que le toma al metro ir de una estación a otra. Ahora, para encontrar el camino más corto se propone utilizar el algoritmo de Dijkstra. Este algoritmo retornaría el tiempo más corto para viajar entre dos estaciones y el camino recorrido para encontrar el tiempo mínimo.

**Floyd – Warshall algorithm:** Esta solución propone implementar un grafo dirigido y ponderado. Las estaciones serian vértices y las vías que las conectan, aristas. Las aristas tendrían un peso asociado que correspondería al tiempo que le toma al metro ir de una estación a otra. Ahora, para encontrar el camino más corto se propone utilizar el algoritmo de Floyd – Warshall algorihm. Este algoritmo retornaría el tiempo más corto para viajar entre cualesquiera dos estaciones del subterráneo de Londres y el camino recorrido para encontrar cada tiempo mínimo.

**Bellman Ford algorithm:** Esta solución propone implementar un grafo dirigido y ponderado. Las estaciones serian vértices y las vías que las conectan, aristas. Las aristas tendrían un peso asociado que correspondería al tiempo que le toma al metro ir de una estación a otra. Ahora, para encontrar el camino más corto se propone utilizar el algoritmo deBellman Ford. Este algoritmo retornaría el tiempo más corto para viajar entre dos estaciones y el camino recorrido para encontrar el tiempo mínimo.

**A\* algorithm:** Esta solución propone implementar un grafo dirigido y ponderado. Las estaciones serian vértices y las vías que las conectan, aristas. Las aristas tendrían un peso asociado que correspondería al tiempo que le toma al metro ir de una estación a otra. Ahora, para encontrar el camino más corto se propone utilizar el algoritmo deA\*. Este algoritmo retornaría el tiempo más corto para viajar entre dos estaciones y el camino recorrido para encontrar el tiempo mínimo.

**Johnson's Algorithm:** Esta solución propone implementar un grafo dirigido y ponderado. Las estaciones serian vértices y las vías que las conectan, aristas. Las aristas tendrían un peso asociado que correspondería al tiempo que le toma al metro ir de una estación a otra. Ahora, para encontrar el camino más corto se propone utilizar el algoritmo de Johnson**.**  Este algoritmo retornaría el tiempo más corto para viajar entre cualesquiera dos estaciones del subterráneo de Londres y el camino recorrido para encontrar cada tiempo mínimo.

**Step 4. Transition from ideas to preliminary designs**

Primero, descartamos la idea del grafo dirigido y no ponderado con la implementación del algoritmo **BFS**, debido a que la solución solo tiene en cuenta el número de estaciones y no el tiempo entre dichas estaciones. El tiempo es de vital importancia para calcular la ruta más óptima para el usuario. Así que, esta idea no cumple con los requerimientos para solucionar esta situación problemática.

Segundo, descartamos la idea del grafo dirigido y ponderado con la implementación del algoritmo de **Bellman Ford,** debido a que el problema no requiere de pesos negativos. En este contexto, el peso de las aristas es el tiempo de viaje entre dos estaciones, por lo que no tiene sentido que sean negativos.

Tercero, descaramos la idea del grafo dirigido y ponderado con la implementación del algoritmo de **Johnson,** debido a que este algoritmo es mejor para grafo dispersos. En base a la investigación, la complejidad de este algoritmo depende del número de aristas del grafo. Es decir, si el número de arista es pequeño este algoritmo es eficiente. En el contexto del metro subterráneo, la cantidad de aristas es muy grande por lo que la implementación de este algoritmo no es conveniente para esta situación problemática.

**Careful review of the other alternatives leads us to the following:**

**Dijsktra´s algorithm:**

* El algoritmo se ejecuta cada vez que el usuario requiera conocer la ruta más optima entre dos estaciones.
* Esta solución es menos eficiente si el grafo es denso.
* Esta solución hace una búsqueda a ciegas, por lo que gasta recursos innecesarios para encontrar el camino más corto entre dos vértices.

**Floyd – Warshall algorithm:**

* El algoritmo se ejecuta solo una vez al inicio y guarda el tiempo mínimo entre un cada par de estaciones. De esta manera, cuando el usuario requiera conocer la ruta más optima, no se ejecuta el algoritmo de nuevo y se ahorran recursos.
* El algoritmo tiene una complejidad temporal cubica.
* La solución es fácil de implementar.

**A\* algorithm:**

* El algoritmo se ejecuta cada vez que el usuario requiera conocer la ruta más optima entre dos estaciones.
* Esta solución no toma en cuenta los caminos innecesarios para encontrar la ruta más optima.
* Esta solución es eficiente si la función heurística h(x) es buena.
* La implementación del algoritmo es muy complicada.

.

**Step 5. Evaluation and Selection of the Best Solution**

**Criterion A. Ease of implementation**

* [1] difficult
* [2] medium
* [3] easy

**Criterion B. Efficiency**

* [1] low
* [2] average
* [3] high

**Criterion C. Resource use**

* [1] high
* [2] average
* [3] low

**Criterion D. Completeness**

* [1] low
* [2] average
* [3] high

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Criterion A** | **Criterion B** | **Criterion C** | **Criterion D** | **Total** |
| **Dijsktra´s algorithm** | 3 | 3 | 2 | 3 | 11 |
| **Floyd – Warshall algorithm** | 3 | 2 | 2 | 3 | 10 |
| **A\* algorithm** | 1 | 3 | 3 | 3 | 10 |

De acuerdo con la evaluación anterior, tenemos que la mejor opción para resolver el problema de la ruta más optima entre dos estaciones del Metro de Londres es el grafo dirigido y ponderado con la implementación del algoritmo Dijkstra.

**Referencias**

<https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

<https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>

<https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/>

<https://www.analyticssteps.com/blogs/dijkstras-algorithm-shortest-path-algorithm>

<https://www.programiz.com/dsa/floyd-warshall-algorithm>

<https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/>

<https://www.javatpoint.com/prim-algorithm>

<https://www.javatpoint.com/kruskal-algorithm>

<https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/>

<https://www.javatpoint.com/bellman-ford-algorithm>

<https://www.baeldung.com/java-a-star-pathfinding>

<https://brilliant.org/wiki/johnsons-algorithm/>