Instituto Superior de Engenharia do Porto

03-01-2022

ESINF Report

LAPR3 INTEGRATIVE PROJECT

SPRINT 3

Group 96:

- ✓ Miguel Morais (1181032)
- ✓ Rodrigo Rodrigues (1191008)
- ✓ Tiago Ferreira (1200601)
- ✓ Mariana Lages (1200902)
- ✓ Eduardo Sousa (1200920)
- ✓ Miguel Jordão (1201487)

Index

Class Diagram Analysis

In this Sprint, we had to connect all the User Stories to the Data Base. One of the most important classes for this Sprint was the FreightNetwork, which creates a graph with cities and ports and the respective capitals, which are connected to each other. The ports of the same country are also connected to each other and the distances are calculated in kilometers. The FreightNetwork class has a connection with the DataBaseUtils class. This class interacts with our Data Base and gets all the information needed from it. The FreightNetwork class also has a direct connection with the Graph interface, which is why this class can create the graphs.

The GraphNClosestPlaces tells the user which places (cities or ports) are closest to all other places (closeness place) the number of closest places is decided by the user ("N" value). The measure of proximity is calculated as the average of the shortest path length from the local to all other locals in the network and they are calculated in kilometers.

The ColourGraph class gives the user a coloured map using as few colours as possible and avoids repetition in the neighboring countries. This class has direct connection with the Vertex interface, which sets the color of each vertex.

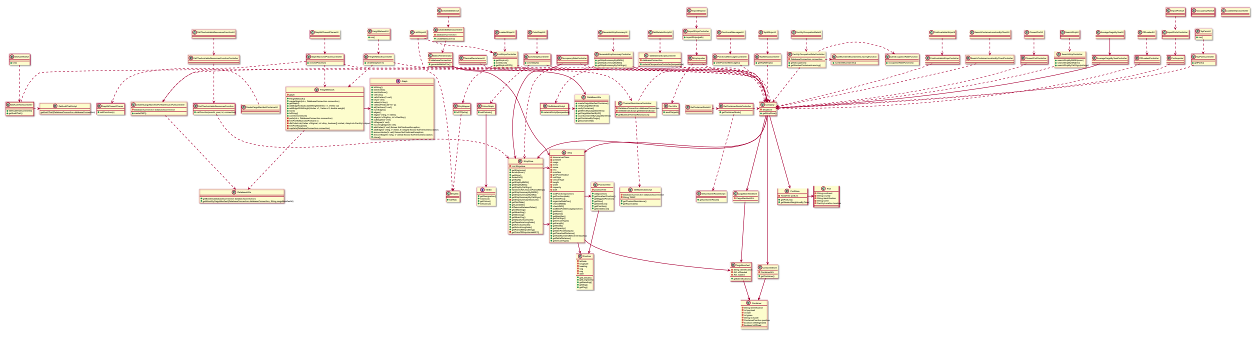

Figure 1 Global Class Diagram

US301 Complexity Analysis

The createGraph() function calls two methods: capitals() and ports(). capitals() has a complexity of $\boldsymbol{O(E+V^2)}$, being E the number of edges and V the number of vertices in the graph. This function calls the DataBaseUtils.getBorders() function that uses the database to get the borders and returns a map, being the key a city and the value all the cities that connect with it. Therefore, this method has a complexity of $O(E+V^2)$, being E the number of edges between vertices that are cities and V the number of vertices already in the graph. The function ports() calls the DataBaseUtils.getSeadists() function that uses the database to get all the distances between ports and returns a map being the key a port and the value another map which key is the port the first connects to the and the value is the distance between the two ports. Then, the method adds all this edges to the graph. To only have the desired edges, we implemented another two methods that are called inside the ports() method. They are linkPortToCapital() and linkPortToNClosestPorts(n). The linkPortToCapital() method travels all the vertices in the graph and, for each vertex that is a city, it travels all the vertexes that are ports and then links the closest port to the city. This method has a complexity of $\boldsymbol{O(V^2)}$. The method linkPortToNClosestPorts(n) starts by traveling all the vertex of the graph and checking if they are ports and, if they are, it uses a DFS (Depth First Search) algorithm to fill a list with every vertex that is not of the same country. Then, we sort each vertex by distance to the 1st vertex and we delete the corrections that are after the N top in the list. This method has a complexity of $\boldsymbol{O(VFlog(F))}$, V being the number of vertices in the graph and F the number of Ports that are not of the same country of the vertex we are analyzing in the first for cycle. Therefore, the ports() method has complexity of $\boldsymbol{O(VFlog(F))}$. Here we are assuming that the number of ports is bigger than the number of cities in the graph. If this is not true, then it is possible that the biggest complexity of the method is $O(V^2)$, leading that the complexity of ports is $\boldsymbol{O(V^2)}$. The complexity of createGraph() is $O(E + V^2) + O(VFlog(F))$, so, the complexity is $\boldsymbol{O(E + V^2)}$.

US302 Complexity Analysis

The setColours method requires a previously built graph to colour it. First, it will create an array that contains the maximum colours possible, using the size of the vertexes of the graph. After that, it will cycle all the vertexes of the graph inside the second "for" cycle, banishing the colours of the adjacent vertices (making them have a value of -1). After that, it will choose the lowest colour possible to paint the vertex, ensuring that the acceptance criteria of using as few colours as possible is achieved. In the end, it will arrange the information in a String Builder that, later, is going to be printed in the UI. Analyzing this method, it is possible to deduce that the worst-case complexity is $\boldsymbol{O(V^2)}$.

US303 Complexity Analysis

The getNClosestPlaces method requires a previously built graph and a "N" (a number introduced by the user). This number, theoretically, can't be less than 1 (since there is no negative "N" closest places), but in case that happens, the method complexity is $\boldsymbol{O(1)}$ since it returns a String warning that no values were returned. If "N" is equal or bigger than 1, the method will search through all the vertices of the introduced graph in the first "for" cycle and then, it will use the shortestPaths method to determine all shortest paths possible for that specific vertex (stored in the possible paths list) and calculate the distance of the path (stored in the distance list). After this method takes place, it is necessary to sort the distance list so we can have, in the first indexes, the lowest values possible. To achieve that, we use a bubble sort algorithm (we could use a quicksort algorithm, that we did try in first place and instead of a complexity of $O(n^2)$ we could have $O(nlog(n))$, but, since it was necessary to sort the distance list and the pathsList at the same time, it was very difficult to make this method generic and it caused several problems in the sublists created by the quick sort algorithm, more specifically, errors, so we decided to implement the bubble sort algorithm). Since there are paths that can be null, it is necessary to verify, inside the bubble sort, if the path is null or not. After the sort is completed, we now have the closest places in the first indexes of the paths list, so we just arrange the information in a String builder that contains the "N" closest places per Continent and print it in the UI. Analyzing this method, it is possible to deduce that the worst-case complexity is $\boldsymbol{O(V^3)}$.

The shortestPaths algorithm is a method that deduces all the shortest paths from a vertex to all the other vertexes of a given graph. It is necessary to have this method implemented so we can calculate as efficiently as possible all possible paths per vertex. This method registers all possible paths in a list and the weight of that in other list. Analyzing this method, it is possible to deduce that the worst-case complexity is $\boldsymbol{O}(\boldsymbol{V} * \boldsymbol{E})$.

Contributions

- US301
    - Code: Eduardo Sousa (1200920)
    - Tests: Eduardo Sousa (1200920)
    - Complexity Analysis: Eduardo Sousa (1200920)

- US302
    - Code: Mariana Lages (1200902)
    - Tests: Mariana Lages (1200902)
    - Complexity Analysis: Mariana Lages (1200902)

- US303
    - Code: Tiago Ferreira (1200601) and Miguel Jordão (1201487)
    - Tests: Tiago Ferreira (1200601) and Miguel Jordão (1201487)
    - Complexity Analysis: Tiago Ferreira (1200601) and Miguel Jordão (1201487)