Intelligent Systems

Beka Bekeri

Enrique Valverde Soriano

Raquel Del Castillo Pérez

Task 1.

First of all, we have decided to create two classes in order to represent the nodes and the edges, this classes contain all the information about each type of the graph elements.

In order to store these elements, we have chosen to use the data structure *list* having two lists, one is used to store node objects and the other is used to store edge objects.

The first method BelongNode takes the ID of an OSM node and checks by using the node list previously created if the ID is in the list of nodes meaning that the node belongs to our graph and returning true or false if the ID is not found.

The method positionNode takes the ID of an OSM node and uses an if statement in order to check if the node is in the list of nodes and if the node exists it prints the coordinates that are stored in that node object.

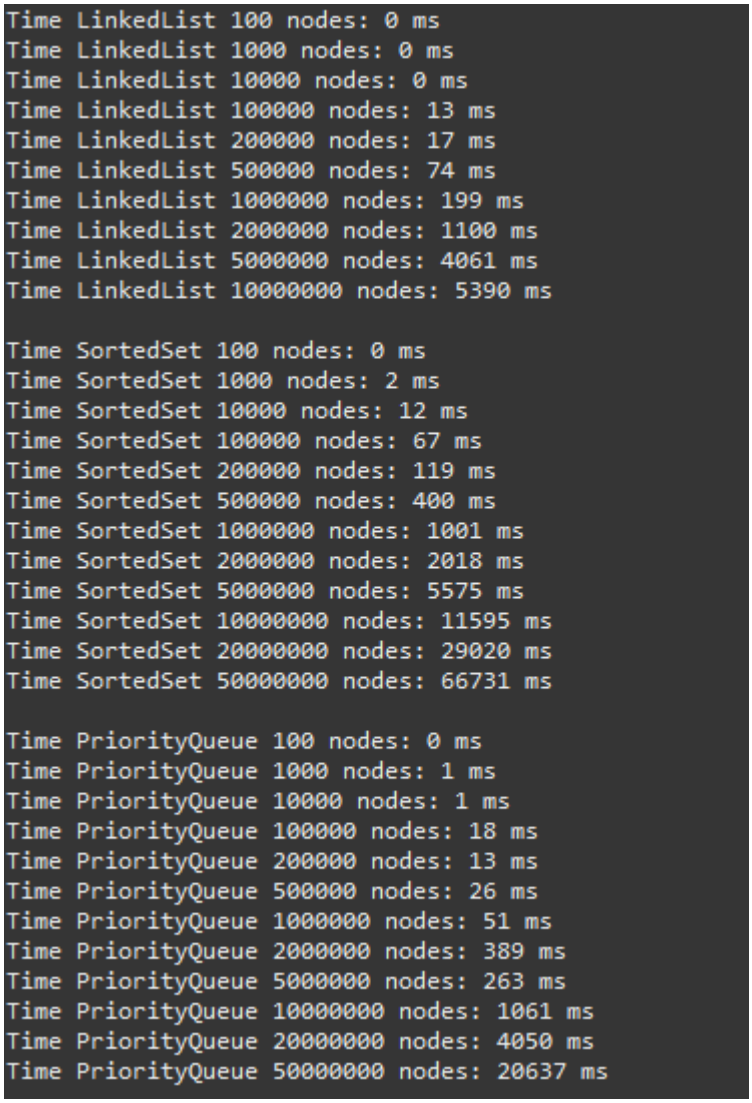
The last but not the least, adjacentNode takes the ID of an OSM node and checks if the node exists and if that happens it checks in the list of edges which of those edges has the node as the source node, if that condition is satisfied it adds the edge to the list of adjacent edges of that node and prints the results.

Task 2.

For this task we have been asked to implement the main estructure of the program, focusing on the class TreeNode in order to test different ordered data estructures that will contain objects of the class TreeNode ordered by a random variable named f, the objetive of the test is to determine which data estructure will handle better the problem, for this purpose a stress test has been implemented using different sizes for the data estructures in order to obtain the data estructure with the biggest size posible, it is important also to consider that another criteria to determine if the data estructure is adequated are the access time and if the data estructure have an implicit method for ordering.

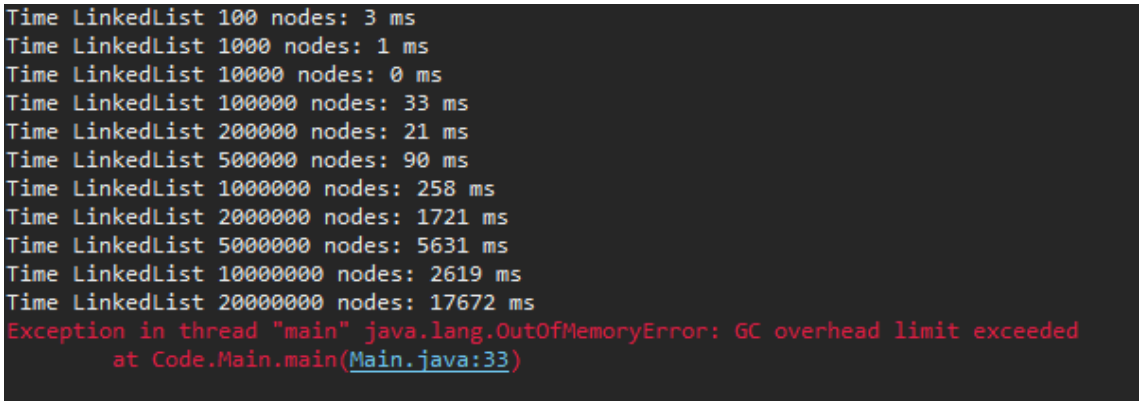
The data estructures that we are going to use in the test are the SortedSet, PriorityQueue and LinkedList, SortedSet and PriorityQueue have implicit methods for ordering (comparators) that made this two estructures an important option, for the LinkedList there is no implicit method for comparing the inserted elements so, it is needed the implementation of a method.

In the following image, the results of the three tests are shown:



As we can see, the fastest one is the LinkedList, remember that here, we are comparing the insertion times, it is important to mention that as the PriorityQueue and the SortedSet have implicit methods, the insertion needs a little more time because of the comparations that are made.

Looking at the image is easy to say that the linked list is the best option, but, there is a problem with this kind of estructure, first of all, the size of it is not so big compared with the other data estructures we are comparing and there is also a problem with the access times that is a big problem due to the fact that we will need to retrieve data from the data estructure very often.



This image shows the memory exception that comes with the use of LinkedList, it has very fast insertion times but the size problem is an important one and also the need of using a method in order to short the elements that are contained in this data estructure.

So, after all tests and a little bit of deliberation we have decided that we are going to use PriorityQueue in order to store the elements of the final program, PriorityQueue uses an implicit method for shorting, has very fast access times and a big enough size in order to store all elements in bigger problems.

The **code for this task can be found in this repository:**
https://github.com/BekaBekeri/SSInteligentes