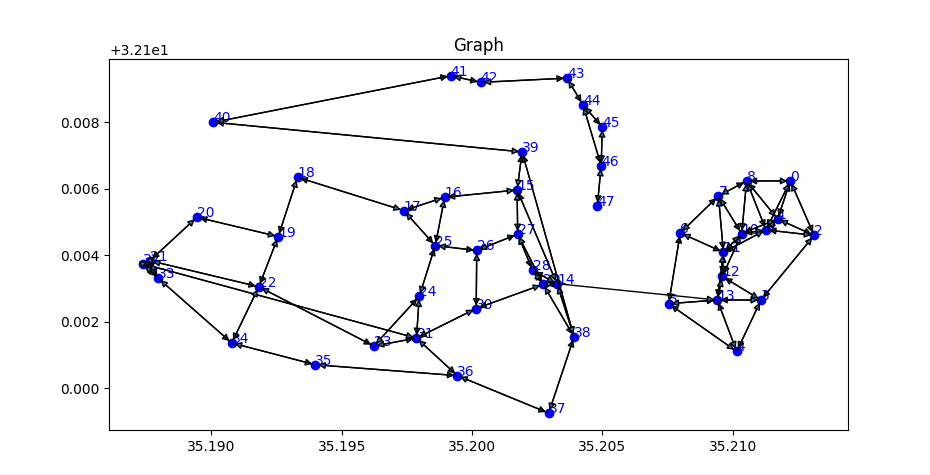
**WIKI**

This project represents a data structure for a directed-weighted graph. We used python dictionary data structure to reach O(1) complexity in the required methods.

The Algorithm class implements algorithms such as DFS, Dijkstra, SCC and more methods. Information about all methods is more specific in our READ ME file, which is added to the project.

In addition of the algorithms GraphAlgo has a method to visualize the graph data structure using the Matplotlib library.

In the project we implement the require interfaces of GraphAlgoInterFace and GraphInterFace and added NodeData classe to our implementation of the project.

We compered methods of the GraphAlgo with JAVA and NETWORKX with the running time of our project, and summarized in the tables below.

1. Shortest\_path
2. Connected\_commponent

|  |  |  |  |
| --- | --- | --- | --- |
| **NETWORX TIME** | **PYTHON TIME** | **JAVA TIME** | **JSON Graph** |
| 0.000 | 0.008 | 0.03 | G\_10\_80 |
| 0.000 | 0.0014 | 0.03 | G\_100\_800 |
| 0.001 | 0.0049 | 0.17 | G\_1000\_8000 |
| 0.3 | 0.489 | 1.25 | G\_10000\_80000 |
| 0.5 | 1.071 | 3.91 | G\_20000\_160000 |
| 0.6 | 1.47 | 10.1 | G\_30000\_240000 |
| **NETWORX TIME** | **PYTHON TIME** | **JAVA TIME** | **JSON Graph** |
| X | 0.006 | 0.09 | G\_10\_80 |
| X | 0.012 | 0.133 | G\_100\_800 |
| X | 0.129 | 0.276 | G\_1000\_8000 |
| **NETWORX TIME** | **PYTHON TIME** | **JAVA TIME** | **JSON Graph** |
| 0.012 | 0.007 | 0.9 | G\_10\_80 |
| 0.018 | 0.010 | 0.144 | G\_100\_800 |
| 0.052 | 0.102 | 0.475 | G\_1000\_8000 |

1. Connected\_commponents

Corectness – Shortest Path

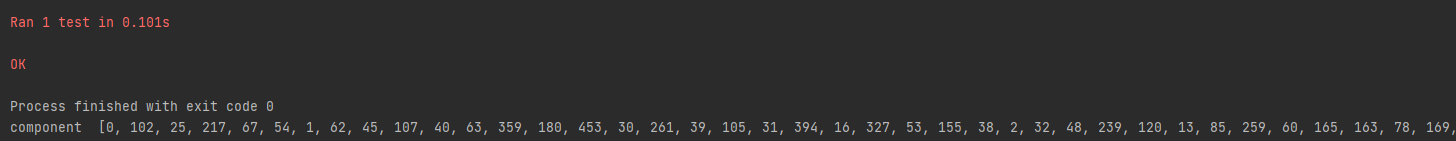
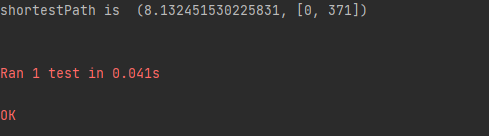
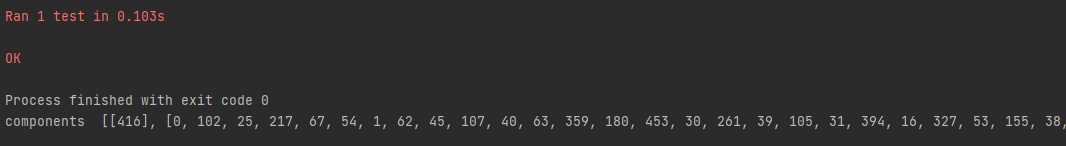
|  |  |  |  |
| --- | --- | --- | --- |
| **NETWORX** | **PYTHON** | **JAVA** | **JSON Graph** |
| (5.222720639544136 [0, 2]) | (5.222720639544136 [0, 2]) | (5.222720639544136 [0, 2]) | G\_10\_80  0-2 |
| (53.330911142826054, [0, 70, 81, 77, 2]) | (53.330911142826054, [0, 70, 81, 77, 2]) | (53.330911142826054, [0, 70, 81, 77, 2]) | G\_100\_800  0-33 |
| (8.132451530225831, [0, 371]) | (8.132451530225831, [0, 371]) | (8.132451530225831, [0, 371]) | G\_1000\_8000-371 |

Corectness – Connected\_commponent

|  |  |  |  |
| --- | --- | --- | --- |
| **NETWORX** | **PYTHON** | **JAVA** | **JSON Graph** |
|  | [0, 2, 1, 4, 3, 5, 7, 6, 8, 9] | [0, 2, 1, 4, 3, 5, 7, 6, 8, 9] | G\_10\_80  0 |
|  | [0, 31, 2, 25, 16, 4, 1, 6, 13, 43, 14, 9, 21, 47, 30, 10, 57, 8, 27, 5, 35, 32, 73, 19, 17, 50, 36, 39, 3, 56, 34, 62, 46, 20, 28, 64, 11, 41, 24, 15, 61, 26, 40.. | [0, 31, 2, 25, 16, 4, 1, 6, 13, 43, 14, 9, 21, 47, 30, 10, 57, 8, 27, 5, 35, 32, 73, 19, 17, 50, 36, 39, 3, 56, 34, 62, 46, 20, 28, 64, 11, 41, 24, 15, 61, 26, 40.. | G\_100\_800  0 |
|  | [0, 102, 25, 217, 67, 54, 1, 62, 45, 107, 40, 63, 359, 180, 453, 30, 261, 39.. | [0, 102, 25, 217, 67, 54, 1, 62, 45, 107, 40, 63, 359, 180, 453, 30, 261, 39.. | G\_1000\_8000  0 |

Corectness – Connected\_commponents

|  |  |  |  |
| --- | --- | --- | --- |
| **NETWORX** | **PYTHON** | **JAVA** | **JSON Graph** |
| [[0, 2, 1, 4, 3, 5, 7, 6, 8, 9]] | [[0, 2, 1, 4, 3, 5, 7, 6, 8, 9]] | [[0, 2, 1, 4, 3, 5, 7, 6, 8, 9]] | G\_10\_80 |
| [[0, 31, 2, 25, 16, 4, 1, 6, 13, 43, 14, 9, 21, 47, 30, 10, 57, 8, 27, 5, 35, 32, 73, 19, 17, 50, 36, 39.. | [[0, 31, 2, 25, 16, 4, 1, 6, 13, 43, 14, 9, 21, 47, 30, 10, 57, 8, 27, 5, 35, 32, 73, 19, 17, 50, 36, 39.. | [[0, 31, 2, 25, 16, 4, 1, 6, 13, 43, 14, 9, 21, 47, 30, 10, 57, 8, 27, 5, 35, 32, 73, 19, 17, 50, 36, 39.. | G\_100\_800 |
| [[416], [0, 102, 25, 217, 67, 54, 1, 62, 45, 107, 40, 63, 359, 180, 453, 30, 261, 39.. | [[416], [0, 102, 25, 217, 67, 54, 1, 62, 45, 107, 40, 63, 359, 180, 453, 30, 261, 39.. | [[416], [0, 102, 25, 217, 67, 54, 1, 62, 45, 107, 40, 63, 359, 180, 453, 30, 261, 39.. | G\_1000\_8000 |

All results were copied from our project and then arranged in a table, example: