Object Oriented Programming withJava - Advanced CourseProject DocumentationRobin Müller -Bady  
December 29, 2019

**Contents**

**1. Introduction**

**2. Group Members** ………..……………………………………………………….. **2  
3 Project Documentation** …………………………………………………………. **1**

**3.1. Project Introduction …**………………………………………………….. **2**

**3.2. Requirements** .…………………..……...……………………………….. **3**

**3.3. Technical description** …………………………………………………... **4**

**3.3.1. Input** ...…………………………………………………………... **4**

**3.3.2. Process** ………………………………………………………..... **3**

**3.3.3. Output** …………………………………………………………… **3**

**3.4. Implementation** …..…………………………………………………...… **4**

**3.4.1. Retrieve user command** ……..…………………………………... **4**

**3.4.2. Reading process** ……...…………………………………………. **3**

**3.4.3. Graph implementation** …………………………………….......... **3 3.4.4. Total nodes, edges, IDs** …...........………………………………. **3**

**3.4.5. Connectivity** …….……………………………………………….. **3**

**3.5. Data flow** …..…...………………………………………………………... **4**

**4. Milestones** …...…………………………………………………………………… **2**

**4.1. Milestone 1** …………………………………………………………….. **3**

**4.2. Milestone 2** …………………………………………………………….. **3**

1. **Introduction:**

This document is one developing document over the whole project and includes the first milestone report on the progress made on our "Communication Network Analysis" project in the purpose of the "Object-oriented programming with Java - Advanced Course". This project documentation refers to project introduction, technical description, designs, implementation and the first milestone report.

**2. Group Members:**

1. Ngo Minh Thong/ Ngô Minh Thông

2. Nguyen Quynh Huong/ Nguyễn Quỳnh Hương

3. Tran Huu Le Huy/ Trần Hữu Lê Huy

4. Luu Nguyen Phat/ Lưu Nguyễn Phát (representative)

**3. Project Documentation:**

**3.1. Project Introduction:**

Nowadays, communication networks grow in importance and maintenance becomes an increasingly challenging task. In order to provide an overview over the network infrastructure, this project is created to aggregate and display information appropriately. By the use of advanced object-oriented concepts in Java programming language, the aim of this project is to create a program which analyses the graph-based communication network models. The program has fully functional, meets the given requirements and provides a good documentation for users can install, run and see the expected results.

The intended audience of this document is the course instructor, who will use it as the basis for a determination of a portion of our grade. The communication network model is based on basic graph definitions, people who are interested in graph also can use this as a tool to refer some specific properties of graph.

**3.2. Requirements:**

The project has to be satisfied following requirements:

- The program was developed using an adequate object orientation.

- Information will be stored in suitable Java Collections or appropriate alternative data structures.

- The program contains adequate error handling.

- The program works with streams and ﬁles. At least one reading and writing ﬁle access has to be made.

- The program contains at least two threads.

- The program is developed using the clean code standard(s) as presented in the lecture.

- Appropriate logging is provided using java.util.Logging or comparable.

**3.3. Technical description:**

In this project, a developed program is possible to basically analyze a graph-based communication network model, from the **input** of a model over the appropriate **processing** of the model data to the **output** of the information. In details:

**3.3.1. Input:**

Via command line interface (CLI): users run the program, specify the input file and put arguments corresponding to properties which they want. The input ﬁle is a XML-based format file (*.graphml*).

The commands follow this format:

|  |  |  |  |
| --- | --- | --- | --- |
| **Run program** | **Input file** | **Properties** | **Output file** |
| java comNetAnalyze.java | input.graphml | -a | output.graphml |

Run program: required

Input file name: required, specifies the input file.

* + - * Properties: variety of properties as follows:

|  |  |
| --- | --- |
| -s x y | shortest path between two nodes: x and y |
| -b x | betweenness centrality measurement of node x |
| -a | all properties |

* + Output file: specifies the output file when choose -a property.

After the input file is opened, the program continues to read then store the graph data in an object Graph – which is designed and implemented by basic data structures in Java Collections. The arguments passed by users will be transfer to the next step – Process – to "calculate" properties.

**3.3.2. Process:**

This "Process" receives arguments from "Input" step, then performs following tasks:

Calculates number of nodes, edges and their identities (IDs)

Determines whether the model is connected or not and calculate its diameter

Finds shortest path between two vertices according to the Dijkstra algorithm (if specified)

Calculates the betweenness centrality measure for a selected node (if specified)

Finally, it passes result of performed tasks to "Output".

**3.3.3. Output:**

This step receives the result of previous step then output to the CLI and/or a file–specified in "Input".

**3.4. Implementations:**

**3.4.1. Retrieve user command:**

Users use command line interface (CLI) to enter their commands. If user type the correct format command, the graph properties that the user required will display on the interface. There are some errors that users can make and interface will pop-up some warnings:

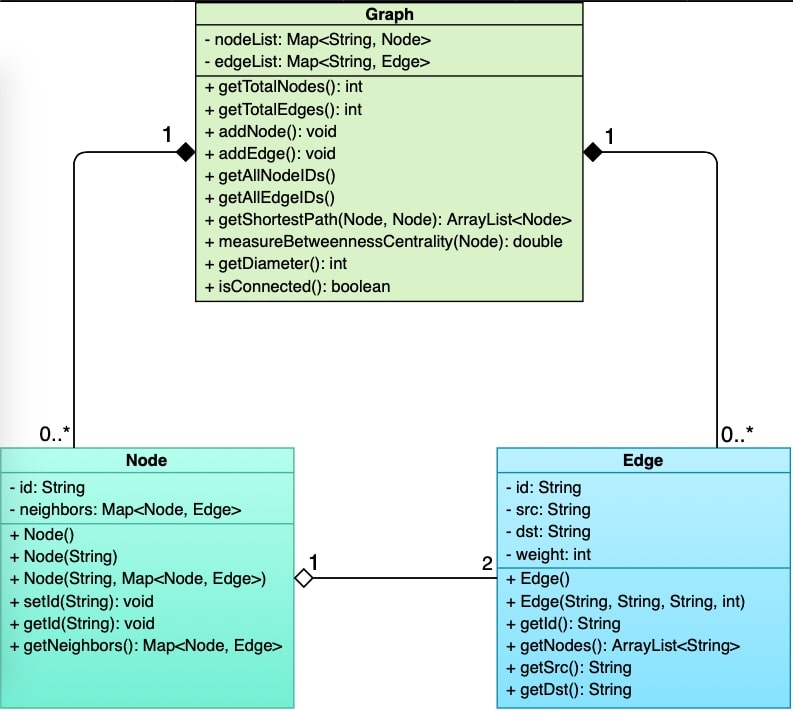
* Wrong input file name or directory: “File does not exist!”
* Forget to type arguments: “There were no command line arguments passed!”
* Wrong properties commands format: “You have entered wrong command format! Please check it again!”

**3.4.2. Reading process:**

The file .graphml is a character-based file. Using java.io.BufferedReader to open it and read it line by line using readLine(). For every read line, this process check whether if it contain node or edge information (by using regular expression) or not. If any, the process will create corresponding object and store those information to the object. Below are all regular expressions:

|  |  |
| --- | --- |
| Graph information | Its regular expression |
| Node ID | (<node id=\")(\\p{Alnum}+)(\">) |
| Nodes of edge | (<edge source=\")(\\p{Alnum}+)(\" target=\")(\\p{Alnum}+)(\") |
| Edge ID | (<data key=\"e\_id\">)(\\p{Alnum}+)(</data>) |
| Edge weight | (<data key=\"e\_weight\">)(\\p{Alnum}+)(</data>) |

**3.4.3. Graph implementation:**



Three objects have been created:

• Node: this object has 2 attributes: id and neighbors. id is its identity, make it unique to other nodes. neighbors is a list of pairs of its neighbor node and the edge connects it with that neighbor. It has 3 constructors: Node(), Node(String) and Node(String, Map<Node, Edge>) which does: creates a "null" node, a node with only its identity and a node has its identity, a list of pairs of its neighbor and a corresponding edge. Method getId returns its identity and method getNeighbors returns value of neighbors attribute.

• Edge: this object has 4 attributes: id, src, dst and weight. They are identity, the source node, the destination node and weight of the edge respectively. It has 2 constructors: Edge(), Edge(String, String, String, int) which does: create a "null" edge, create an edge with its identity, source node, destination node and weight respectively. Method getId() returns its identity, getNodes() returns a pairs of node of that edge, getSrc() returns its source node, getDst() returns its destination node.

• Graph contains nodes and edges, two list of nodes and edges which are nodeList and edgeList respectively. getTotalNodes() returns total number of its nodes; getTotalEdges() returns total number of its edges; addNode() adds a new node, addEdge() adds a new Edge to graph; getAllNodeIDs() returns a list of IDs of all nodes, getAllEdgeIDs() returns a list of IDs of all edges; getShortestPath(Node, Node) returns a ordered list of nodes which is a shortest path to go from a given node to another given node; measureBetweennessCentrality (Node) measures betweeness centrality of a node; getDiameter() return this graph diameter and isConnected() returns whether this graph is connected or not.

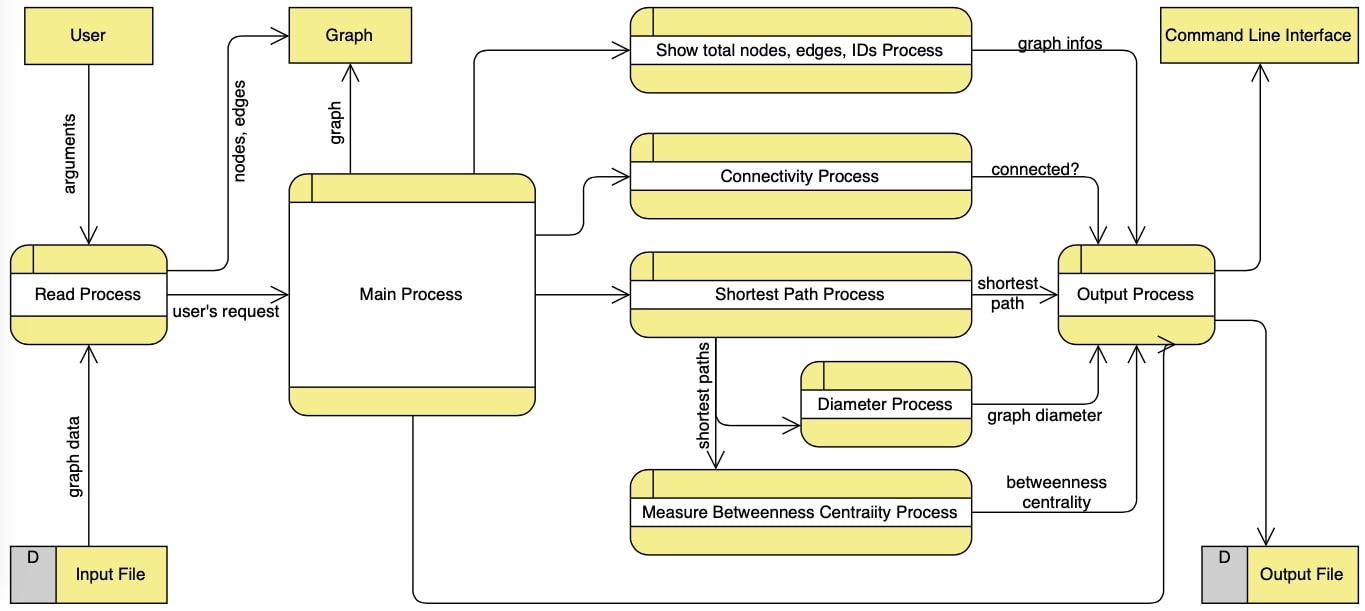
**3.4.4. Total nodes, edges, IDs:**

Graph contains nodeList and edgeList, both use Map data structure, by calling nodeList.size(), edgeList.size() we will get the total nodes, edges respectively. By looping through nodeList, for each item we return its ID and finally we will got all IDs of nodes. So, we can return IDs of all edges in the same way

**3.4.5. Connectivity:**

To check the connectivity of a graph, a common way is trying to traverse the graph. In this program we use Depth- First Search Algorithm to implement that. When then method isConnected() is called, a new graph name DFSTree will be created through the algorithm from a random node of the undirected graph. If the tree has all the nodes of the given graph or has the same total nodes then the graph is connected.

**3.5. Data flow:**



**4. Milestones:**

**4.1. Milestone 1:**

**Previous Background:**

As this is our first progress report, we have little to report as previous background. Prior to this reporting period, we:

* Discussed and analyzed project's requirements.
* Designed the user interface and objects of the project.
* Created tasks, milestone and assigned tasks to team members.
* Used GitHub to control versions of source code.

**Work Completed:**

During this reporting period we have accomplished the following:

|  |  |  |
| --- | --- | --- |
| Task ID | Task description | Accomplished by |
| 1 | Reading input from users and parse arguments. | Hương |
| 2 | Load network model from user-specified file using regular expression | Phát |
| 3 | Store the given graph model by using HashMap in Java Collections | Phát + Thông |
| 4 | Implemented the algorithm determine connectivity of the graph by using Depth First Search. | Thông |
| 5 | Output number of nodes, edges and their identities (IDs). | Huy |
| 6 | Started writing project documents and designing related diagrams. | Team |

**Work Scheduled:**

During the next reporting period, we plan to:

* Summarize reports from members and combine their works to project.
* Present the progress of the project have done by team

During the subsequent weeks, we plan to:

* Implement the Dijkstra algorithm to find shortest path between two given nodes
* Implement the function to find diameter of the graph
* Code the function to measure "betweenness centrality" of the graph
* Continue writing document about completed works.
* Improve some implementations of algorithm.
* Organize team meeting to discuss about the progress of working project.

**Problems Encountered:**

The items below have been resolved:

* Error appears when checking connectivity of big graph
* Some members have trouble with cloning, committing and pushing repository from GitHub to Eclipse IDE.

**Changes in Requirements**

There have been no changes in the initial requirements.

**Overall Assessment of the Project**

The project is going well and assigned tasks have been done before the first milestone deadline. Project document is on progress of writing. All problems have been solved and some improvements on the code were made.