[NSD] - Practical 2 Report

Date: 05/10/2017
Authors: Andrea Ferlini,
Luca De Mori

Exercise 1 — BFS Implement an efficient BFS algorithm.
Use it to make an algorithm that outputs all connected components and their sizes (number
of nodes). Test your algorithm on some real-world graphs, what can you say?

We developed the function called *printConnectedGraph* which prints all the connected components of the graph and their size.
The algorithm has been tested on six different graphs (from 300 000 up to 378 142 420 nodes), in order to proof both its efficiency and its scalability.

The analyzed graphs present heterogeneous structures, varying from 1 to 1721 connected components.
Here now follows a table to summarize the obtained results.

Graph FileName	Size [vertices]	Volume [edges]	Connected Components
actor_collaboration	382,219	33,115,812	1721
roadNet_PA	1,088,092	30,622,564	206
orkut_groupmemberships	11,514,053	327,037,487	1
wikipedia_link_en	12,150,976	378,142,420	291
com_amazon	334,863	925,872	1
com_youtube	1,134,890	2,987,624	1

Use your BFS algorithm to make an algorithm that computes an approximation of the diameter of a graph. Test it on real-world graphs with a known diameter.

To be sure we are running the algorithm in the largest connected component, we developed a function called *extimateAccurateDiameter*. It runs an iterative probing of the “longest shortest path”, in every connected element, then it returns the maximum one.

We run the algorithm until the diameter remains stable for 5 iterations.
In the table below follow the obtained results.

Graph FileName	**Size [vertices]**	**Volume [edges]**	**Diameter [edges]**	**Required time [s]**
actor_collaboration	382,219	33,115,812	13	37
roadNet_PA	1,088,092	30,622,564	794	9
orkut_groupmemberships	11,514,053	327,037,487	8**	247
wikipedia_link_en	12,150,976	378,142,420	10	590
com_amazon	334,863	925,872	47	2
com_youtube	1,134,890	2,987,624	24	4

** the diameter is different from the one present on the website.

Exercise 3 — Dijkstra (optional) Implement the Dijkstra algorithm.

The function *executeDijkstra* we developed, works on weighted graphs in the format:

x_1 y_1 z_1
x_2 y_2 z_2
...

x = vertex(unsigned int), y = neighbour (unsigned int), w=weight (unsigned int)

In order to let it works we added the support to weighted graph in the adjacency list structure and load function. It’s required to accordingly set the flag *is_weighted* when it is called.

All the tests were run on a laptop with the following characteristics:
Memory: 8GB
Processor: Intel® Core™ i7-6700HQ CPU @ 2.60GHz × 8
OS: ubuntu 16.04 LTS 64-bit

The used graphs come from http://konect.uni-koblenz.de database.