MITRO 209:
Project

Mauro Sozio
sozio@telecom-paristech.fr

In this lab, we are going to implement the main algorithms for computing densest subgraphs we saw during our course. You should submit the code (in C/C++,Java or Python) as well as a short report in pdf (max 5 pages) containing the plots requested in the text of the exercise together with the answers to the questions. You will be evaluated according to how efficient is your code (C? Python? which data structures?) and how well you explained what you did (e.g. including a picture of the data structure you used might help). You should also discuss whether the results are expected or not and if not motivate why you might have obtained different results than what you had expected. Any "interesting findings" will also be appreciated. The project can be done in groups of maximum two people. Each group should send just one single email with the name of the people involved in the project. We will be more demanding with projects made by two people. The project consists of two parts as follows.

Part 1: Greedy Algorithm Implement the 2-approximation algorithm for the densest subgraph problem (slide 10 of the slides on finding densest subgraphs). The running time of your algorithm should be linear (in the size of the input graph) in the worst-case. You should explain which data structures you used and how things have been implemented. It should be clear what you did even without reading the code. Consider 5 graphs with different size from `http://konect.uni-koblenz.de/networks/`.

1. Plot the running time of your algorithm as a function of the input size. You should try to give evidence that your algorithm has indeed linear running time.

2. Report the density of the subgraphs you found as well the number of nodes in each of the subgraphs.

Part 2: Streaming Algorithm Implement the $2+\epsilon$-approximation algorithm for finding densest subgraphs in streaming (slide 4 of the slides "Densest Subgraph in Streaming"). You should implement the algorithm in the following settings. You have only sequential access to the file where the graph is stored, which can be read multiple times. While reading the file you can store some information in main memory, however, you can only use $O(n \log n)$ bits in main memory. In particular, for each node you can store in main memory only its (current) degree and not the edges incident to the node. You should explain which data structures you used and how things have been implemented (for example how do you remove the edges in the graph). It should be clear what you did even without reading the code. Consider 5 graphs with different size from `http://konect.uni-koblenz.de/networks/`.

1. Plot the running time of your algorithm as a function of the input size when $\epsilon = 0.1$. Is the running time in practice consistent with what suggested in theory? Better? Worse?

2. Then, consider just one graph and run your algorithm with ϵ in $\{0.2, 0.4, 0.6, 0.8, 0.1\}$. For each such a value, report running time and density of the subgraph found by the algorithm. How do these results compare with the theory we saw during our class?

Plagiarism: If we suspect that students copied their code or their reports from other students, all the students involved will receive 0 points for this lab and a penalty at the final mark for this course.