Handling a large graph

Maximilien Danisch

1 To get things started

Exercise 1 — ***Preparation*** Download the following graphs:

- `http://snap.stanford.edu/data/email-Eu-core.html`
- `http://snap.stanford.edu/data/com-Amazon.html`
- `http://snap.stanford.edu/data/com-LiveJournal.html`
- `http://snap.stanford.edu/data/com-Orkut.html`
- `http://snap.stanford.edu/data/com-Friendster.html`

All these graphs will enable you to check the results of your programs.

You can, in addition, create (manually) a few small graphs and store them in files where each line is of the form:
u v
which indicates that a link exists between nodes u and v.

Exercise 2 — ***Size of a graph*** Make a program that counts the number of nodes and edges in a graph (without storing all edges in memory) and writes this value on the standard output.

Exercise 3 — ***Cleaning data*** We assume the graphs to be simple and undirected. Make a program that deletes self-loops and duplicated edges (e.g. for a bidirected edge (a line "u v" and then a line "v u" in the file), just keep one of them).
Note that you can use unix commands to do that and write the result in a new ".txt" file.

Exercise 4 — ***Node degree*** Make a program which computes the degree (*i.e.* the number of neighbors) of each node of the input graph (without storing all edges in memory) and writes this value in an output text file.

Exercise 5 — ***A special quantity*** Make a program that, given a graph $G = (V, E)$, computes the following quantity:

$$Q_G = \sum_{(u,v)\in E} d_u \times d_v,$$

where d_u is the degree of node u. Your program should be as fast as possible
Report the value of the quantity Q_G as well as the running time of your algorithm for each of the 5 downloaded graphs.

Exercise 6 — ***Degree distribution*** Make a program that computes the degree distribution of a graph and writes it in a file in the following format: "*d n*" on each line, where n is the number of nodes in the graph having degree d.
Plot the obtained degree distributions (you can use gnuplot or matplotlib).

2 Load a graph in memory

Exercise 7 — ***Three graph datastructures*** Make three programs reading a graph and storing it in memory:

1. as a list of edges,
2. as an adjacency matrix,
3. as an adjacency array.

Note that these three programs are important as they will be used in the future practicals. Make sure to have them working fine.
Use them on the 5 downloaded graphs and conclude on the scalability of the three programs.

3 Breadth-first search and diameter

Exercise 8 — ***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 the 5 downloaded graphs and, for each one of them, report the fraction of nodes in the largest connected component.
Use your BFS algorithm to make an algorithm that computes a good lower bound to the diameter of a graph.
Test your algorithm on the 5 downloaded graphs and report your lower bound.

4 Listing triangles

Exercise 9 — ***Triangles*** Implement an efficient algorithm for counting triangles.
Test your algorithm on the 5 downloaded graphs and report the number of triangles as well as the running time of your algorithm.
Generalize your algorithm to compute the transitivity ratio of the graph.
Generalize your algorithm to count the number of triangles each node belongs to and to compute the clustering coefficient of each node in the graph and the clustering coefficient of the graph.