Advanced Programming Techniques

Graph searches

Recommended programming language: C++

Disclaimer: efficient implementations of graph data structures and algorithms can be already found in various dedicated libraries. The goal is to write your own code!

1) Start implementing a Graph class. Graphs considered must be dynamic (i.e., they support edge/vertex insertion/removal). The following member functions must be implemented:
   - getters for the number of vertices and the number of edges
   - iterators on the edges, resp. on the adjacency list of any input vertex
   - adjacency testing between any two vertices

   We internally represent the graph as a vector of adjacency lists. Edges are further stored in a separate Hash-table, along with their respective positions in the adjacency lists of their end-vertices.

2) Have a look at the Stanford Large Network Dataset Collection:

   https://snap.stanford.edu/data/

   Your codes should be tested by using samples from this dataset. You may want to have a closer first look at ego-Facebook due to its smaller size than most other networks in the collection. Write a parser in order to generate this graph (as an instance of your class).

   It may be judicious to only generate a subsample of this graph. For that, select at random which vertices and edges are kept.

3) Write classical implementations of DFS and BFS. Compute the average time taken for each traversal, each time choosing a random start vertex. Plot the result, for various increasing subsamples of ego-Facebook. Use larger graphs from SNAP in order to empirically estimate the maximum size of graphs for which you can execute these searches in less than 30 min.

4) Implement the partition refinement data structure:

   https://en.wikipedia.org/wiki/Partition_refinement

   Then, write an implementation for LexBFS. Compare its execution times with the ones empirically observed for DFS and BFS.

5) Implement the A* heuristic. Compare its execution times, but also the quality of its output, with BFS and LexBFS.