Lab 7 Breadth-first search on graphs

C S 372 Data Structures and Algorithms

October 29, 2019

In this lab, you will implement the breadth-first search (BFS) algorithm using an iterative strategy and make sure it runs in linear time to the size of the input graph.

1 Requirement for the BFS algorithm

Design a linear-time BFS algorithm with the following function prototype

```
void BFS(Graph & G, Node & s)
{
  // Input: graph G, which can be either undirected or directed

  // Output: the effect of BFS is that all nodes
  //         1. are marked with a distance label

  // perform BFS on the graph from a given node s
}
```

The effect of the BFS on the graph G are distance labels indicating the distance from source node s to every node in the graph. In C++, you can generate a `double` type positive infinity by the function

```
std::numeric_limits<double>::infinity()
```

The distances will be stored within the `Graph` data structure. You can enhance the graph class from previous labs in more than one way. You are free to design your strategy and describe it in your lab report. The distance values cannot be saved as global variables.

2 Iterative solution

You can use the `std::queue` class in C++ to hold the discovered but not yet processed nodes. The useful member functions of the class are `push()` to inject an object into the end of the queue, `pop()` to eject an element from the front of the queue, `front()`, and `empty()`.

3 Test the classes

Generate five example graphs to test your code. The graphs do not have to be very large but must represent a variety: cyclic/acyclic, directed/undirected, having one or more connected components. You test your code against the precomputed distance values as the truth.

Your C++ program must include a main() function that calls the `testall()` function. When the program is compiled by a C++ compiler it generates a binary executable file that will run when invoked from the command line.

4 Plot the runtime as a function of number of nodes and edges

After testing the correctness of the the program, you will study how the runtime scales with the size of graph for your BFS implementation. Use the random graph generation function you developed in Lab 4 to produce large graphs in increasing sizes.

You will produce by R two curves of BFS runtime as a function of

1. the number of nodes, given the number of edges
2. the number of edges, given the number of nodes

in the input graphs.

Your algorithm must be able to handle graphs with 10,000 or more nodes and 1,000,000 or more edges in the order of a few seconds.

5 Submission

Write a lab report to describe your lab work done in the following sections:

1. Introduction (define the background, motivation, and the problem),
2. Methods (provide the solutions),
3. Results
   (a) visualize your five test graphs using R package `igraph`
   (b) show the two curves for empirical runtime of BFS on large graphs. Discuss if they appear to be linear. If not, explain the possible reasons.
4. Discussion (general implications and issues),
5. Conclusions (summarize the lab and point to a future direction).

Submit the source code files and your lab report online.