Lab 6 Finding connected components in graphs in linear time

C S 372 Data Structures and Algorithms

October 15, 2019

You will implement algorithms to find connected components in undirected graphs and strongly connected components in directed graphs. In this lab, you can re-use the DFS code you did in Lab 5.

1 Requirement for the algorithms

1.1 Connected components in undirected graphs

Implement the first algorithm to find connected components in an undirected graph with the following function prototype:

```
vector<size_t> find_connected_components(Graph & G)
{
}
```

The input must be treated as an undirected graph G. The output is a vector containing connected component ids for each node in the graph. The id of node i must be contained in the i-th entry of the vector. If the graph has K connected components, the ids must range from 0 to $K-1$.

The algorithm must run in $O(|V|+|E|)$ time.

1.2 Strongly connected components in directed graphs

Implement the second algorithm to find *strongly* connected components in a directed graph with the following function prototype:

```
vector<size_t> find_strongly_connected_components(Graph & G)
{
}
```

The input must be treated as an directed graph G. The output is a vector containing connected component ids for each node in the graph. The id of node i must be contained in the i-th entry of the vector. If the graph has K connected components, the ids must range from 0 to $K-1$.

As the standard solution to strongly connected components relies on previsit and postvisit time stamps for each node using DFS on the reverse graph of G, will the recursive and iterative DFS algorithms (from Lab 5) generate different time stamps? If so, which one you will use for strongly connected components? Please discuss this in your lab report.

2 Test the classes

Generate five example undirected graphs and five example directed 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.

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.

3 Graph 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 scale with the size of graph for the two methods. Use the random graph function you developed in Lab 4 to generate random graphs of increasing sizes.

You will produce four curves in R:

1. `find_connected_components` runtime as a function of number of nodes in the graph, given the number of edges
2. `find_strongly_connected_components` runtime as a function of number of nodes in the graph, given the number of edges
3. `find_connected_components` runtime as a function of number of edges in the graph, given the number of nodes
4. `find_strongly_connected_components` runtime as a function of number of edges in the graph, given the number of nodes

You will design sizes of the graphs so that your runtime will change substantially among the sizes.

4 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 pseudocode solutions),
3. Results
   (a) visualize the ten test graphs using R package `igraph`
   (b) show the four curves for empirical runtime on large graphs. Discuss if they appear to be linear. If not, explain the possible reasons.
4. Discussion (general implications and issues). Answer the question regarding the use of DFS iterative versus recursive solutions.
5. Conclusions (summarize the lab and point to a future direction).

Submit the source code files and your lab report online.