COMPSCI 2XC3 — Lab 7

DUE : Sunday, March 13, 11:59 PM

Submit the lab on Avenue. Note, Avenue must receive your lab by the due date. **Do not leave your submission to the last minute! Late submissions, even by seconds, will not be graded.** You have been warned.

This lab is worth (59/7)% of your final grade in the course. Read this document carefully and completely. **Whenever I ask you to discuss something, I implicitly mean to include that discussion in your lab report. Furthermore, when I ask you to run an experiment or evaluate the performance of a function/method include scatter plots where appropriate. Do not import external libraries, especially those containing data structures you are meant to implement!** Include all timing experiments in your `code.py` file.

Purpose

The goals of this lab are as follows:

1. Become familiar with implementations of graphs.
2. Modify implementations of basic graph algorithms.
3. Implement basic graph algorithms.

Note, treat this lab as an introduction to graph algorithms and implementation. The majority of the remainder of labs will be graph related.

Submission

You will submit your lab via Avenue. You will submit your lab as three separate files:

- code.py
- graphs.py
- report.pdf (or docx, etc.)

Only one member of your lab group will submit to Avenue – see the section below for more details on that. Your report .pdf will contain all information regarding your group members, i.e. name, students number, McMaster email, and enrolled lab section. This will be given on the title page of the report. For the remainder of this document, read carefully to gauge what other material is required in the report. Your report should be professional and free from egregious grammar, spelling, and formatting errors. Moreover, all graphs/figures in your report should

be professional and clear. You may lose grades if this is not the case. Organize the report in a such a way to make things easy for the TA to grade. You are not doing yourself any favors by making your report difficult to mark!

Basic Graph Algorithms [70%]

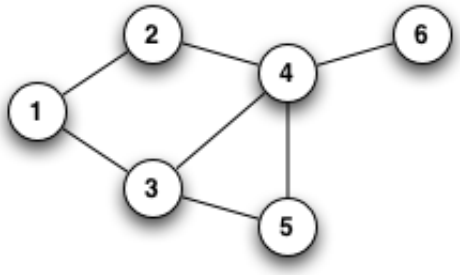
In `lab7.py` you will find an implementation of an undirected (non-weighted) graph alongside some simplistic implementations of Bread First Search (BFS) and Depth First Search (DFS). If everything goes according to plan, we will also be going over these elements in lecture as well. Right now, BFS and DFS are not the most useful implementations. They simply return True iff a path exists from `node1` to `node2` (using the Breadth and Depth approaches).

Implement variations on these which return the path from `node1` to `node2` as a list of nodes, starting with `node1` and ending with `node2`. For example, `BFS2(G, 3, 5)` would return:

```
[3, 10, 4, 1, 5]
```

if via BFS a path from 3 to 5 was found by first going to 10, then 4, then 1, then finally 5. If no path is found the function should return an empty list. Name these functions `BFS2` and `DFS2` and include them in your graphs.py file.

Now implement variations on BFS and DFS which do not take in a second node. Instead, the function finds a path to every node (which there is a path to) and returns these paths in the form of a "predecessor dictionary". Name these functions `BFS3` and `DFS3` and include them in your graphs.py file. This predecessor dictionary encodes all the path information. For example, for the graph below:



the BFS3(1) predecessor dictionary would be:

```
{2 : 1, 3 : 1, 4 : 2, 5 : 3, 6 : 4}
```

Note, from the above one could build the path from 1 to 6 by working backwards from 6 looking at its predecessors until they reached 1. If there is not path to a node k from the input node, then k should not appear in the predecessor dictionary.

For the following two functions, reuse your code above where possible.

- Implement a `has_cycle(G)` function in your graph.py file. This function should return True iff there is a cycle in G.
- Implement a `is_connected(G)` function in your graph.py file. This function should return True iff there is a path between any two nodes in G.

Cycles and Connected Probability

Consider how to randomly construct graphs with a general number of nodes k. How would you add edges to your graph? Start by focusing on graphs with $k = 100$, let the number of edges added to the graph be c. Imagine, $c = 30$. Some of the randomly created graphs will have cycles, and others will not. But what portion will and will not? Graph The portion of graphs which have a cycle vs c.

Now do the same but for portions of graphs which are connected vs c. For each experiment, what approximately is the value of c which roughly half the graphs to contain a cycle/be connected. Does it make sense why one of these values would be less than the other?