CISC 235 Assignment #4

Instructions

This assignment is due at 4:30 p.m. on **Monday, April 4th (EXTENDED)**. Please email your submission to the Teaching Assistant assigned to your lab section.

Lab A (Monday): 8jjl1@queensu.ca

Lab B (Friday): zi@acm.org

The labs provide an opportunity for you to work on assignments with the supervision of our Teaching Assistants.

Part 1 (4 Marks): Create a Graph

Write a method to randomly generate an undirected, unweighted, connected graph with 1,000 vertices **(note: 1,000 will take a very long time to run – much longer than you'll want to wait – so test it with, for example, 20 vertices!)**. Your method must generate vertices that are arbitrarily connected. Specifically, it must be possible for simple paths (i.e. paths with no repeated vertices) to contain cycles. You can store your graph in an adjacency list or adjacency matrix.

Part 2 (8 Marks): Break the Longest Cycle

Write a method to detect the longest cycle within the graph. Once detected, the method should remove one of the edges so that the cycle no longer exists. *Note: after removing this edge, your graph will still contain only one connected component.*

Part 3 (2 Marks): Break all the Cycles

Write a method to repeatedly break the longest cycle, until no cycles remain. This method should use your method from Part 2.

Write a comment at the top of this method answering the following question: why might you want to remove edges in this order? Creative solutions are encouraged.

Part 4 (6 Marks): Calculate and Display Graph Information

Write a method to calculate and display the following information about your graph:

1) The number of connected vertices *(Note: it should still be 1,000!)*
2) The number of edges
3) The number of edges on the longest simple path
4) An edge that, if removed, would separate the graph into two trees with the most equal number of vertices in each *(Note: to display this edge, display the adjacency list or matrix indices for the two vertices that this edge connects)*

Note: you may want to test your code on specific "hard-coded" graphs to ensure that it works in all cases!