UCR - CS 010C

Assignment 4

**Deliverables:** Create a single PDF file that contains your answers to the questions. Then create a zip file that contains this PDF file along with all your code source files. Submit this zip file on iLearn.

**Deadline:** 12/11/2020 11:59 pm.

**Exercise 1**

Use provided C++ skeleton to insert your code.

1. Define Graph, which stores an **undirected** graph using **adjacency list**, where each node stores a CityName (string) and each edge has a double weight (distance between two cities).

Implement the following functions in Graph:

1. bool hasTripletClique(): returns true if there are three nodes in the graph that are all connected to each other. E.g., a, b, c, with edges (a, b), (b, c), and (a, c)
2. bool isConnected(): returns true if graph is connected
3. double getMinDistance(string city1,string city2): returns the shortest path distance between city1 and city 2. Hint: You may use Dijkstra Algorithm.
4. [extra credit] double getLongestSimplePath(): returns length of longest simple path (no cycle allowed)
5. What is the big-Oh complexity of your functions above if graph has nodes and edges?
6. Test your functions. Write code to create a random graph of 100 nodes, with 500 random edges with weight 1.0, 500 random edges with weight 2.0 and 500 random edges with weight 3.0. (For function in A(d) use a smaller graph if too slow.) Measure the time of each function in nanoseconds or microseconds.
   * Assume there can be at most 1 edge between 2 nodes.
   * Assume there is no self-loop (edge from one node to itself).

Assignment 4

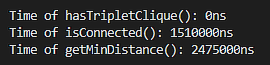
1.

B: Big-Oh complexity of functions

* 1. hasTripletClique(): O(n + m)  
       
     This function traverses through each node and every edge in the worst case.
  2. isConnected(): O(n + m)  
       
     This function combines BFS traversal with a for loop which iterates through each node to check if every node in the graph is visited. BFS traversal has a complexity of O(n + m) and the for loop iterates n times. O(n + m) + O(n) = O(2n + m) = O(n + m).
  3. getMinDistance(): O((n + m) \* logn) on average  
     O((n + m) \* logn + n) in the worst case  
       
     This function utilizes Dikstra’s algorithm to determine the shortest distance of every node from an initial node. Dikstra’s algorithm has a time complexity of O((n + m) \* log(n)) when implemented with a min heap. After Dikstra’s algorithm runs, the function searches a hash map for the distance associated with the destination node with an average time complexity of O(1). In the worst case however, calling .find() on an unordered map has a complexity of O(n).
  4. getLongestSimplePath(): O(n^3 + n^2 \* log(n) + (n \*m) \* log(n))  
     The way I implemented this function was incredibly inefficent, but basically the function calls Dikstra’s algorithm n times so that the algorithm can be run for every node in the graph as the initial node. Then the longest path value is updated to be the max distance found in the vector of node distances in each iteration. Since I implemented the node distances as a vector, it takes O(n) time to find the max distance. Thus the complexity is as follows:

O(n \* ((n + m) \* (log(n) + n))) = O(n \* (n^2 + n\*log(n) + m\*log(n))) = O(n^3 + n^2 \* log(n) + (n \* m) \* log(n))

C: Function runtimes



My runtime for hasTripletClique() was consistently 0 nanoseconds even when I tried changing  
std::chrono::high\_resolution\_clock to std::chrono::steady\_clock.

For getLongestSimplePath(), I reduced the number of nodes to 25 and edges to 125 since my graph.exe program was taking too long to execute. This function was extremely long, especially for a graph of this size when compared to the original graph size, but this makes sense since finding the longest path in an undirected graph is an NP-hard problem (according to wikipedia) and cannot be solved in polynomial time.