**Page Rank Documentation**

Data Structure: For this project I utilized an adjacency list class. This class was functionally a wrapper class that encompassed an unordered\_map, where the key was a string representing the from page, and the value was a list containing all the to pages that the from page linked to. An unordered\_map was used as the key, value construct was required to associate the from page to the pages it linked to. Additionally, this structure did not have a requirement to be ordered, so the unordered version was chosen over the standard map. A list was used to store the to pages as it was the lightest weight container that allowed storage, iteration, and stored a size variable.

Time complexity, methods:

The adjacency list implements two public methods:

**Insert**: inserts the from page and to page of an edge into the underlying map->list data structure. Insertion into a list is constant time, so this runs in linear time with respect to the length of the string representing the from page, as the unordered\_map is wrapper over a hash table.

**Size**: retrieves the size variable from the underlying unordered\_map, runs in constant time.

A **begin** and **end** method are also implemented that return the beginning and end of the underlying unordered\_map to allow iteration. An iteration through the keys in the adjacency list runs in O(|V|).

The PageRank class implements three methods:

**ParseInput**: gets the input from the user and stores the input edges into an adjacency list. E number of inserts which execute in constant time (discussed above). Runs in O(|E|).

**IteratePageRank**: Runs the actual algorithm. Iterates through all edges in the adjacency list p times, where p is the power iteration input by the user. Results in O(p \* |E|).

**PrintResults**: Outputs the results of the page rank algorithm. Iterates over a separate map structure that associates all vertices with a page rank value. Runs in O(|V|).

The time complexity of the main Page Rank algorithm is O(p \* |E|) as discussed above. If we consider p to be a constant, this reduces down to a time complexity of O(|E|).

Lessons learned: RTFM. During project one, I spent a significant chunk of time attempting to figure out how to programmatically test the project with IO. Never figured it out and missed two IO related test cases because of it. Spent more time during this project trying to figure it out and got nowhere. Randomly clicked in the readme file included in the Catch code for project one and discovered that the answer that I need was readily available to me the entire time. So, RTFM, and make sure to go through all the given materials at the beginning.