Project 2 – Simplified PageRank:

SEBASTIAN PAULIS

Documentation:

1. Describe the data structure you used to implement the graph and why?
   - The data structure used to implement the graph was an adjacency list. Specifically, it is of type "unordered_map" with a key of type string and a value of "unordered_set", which has a type of string. The reasoning behind the use of a map and set was to use the string values as indexes for O(1) basic operation. This allows the adjacency list to be used in the same time complexity as a two-dimensional array without having unnecessary memory that is not being used. Conveniently, all vertices and edges can be accessed and read through their string value, rather than an arbitrary integer value. This makes readability highly possible, since there are less lines of code and easy indices to work with. Finally, the reasoning behind an "unordered_set" is the idea that these graphs are guaranteed to not have parallel edges, which means that each edge will be unique in their direction; therefore, a set would work perfectly to gather the outdegree of a vertex in O(1) and whether a vertex is connected to a neighbor in O(1).
2. What is the computational complexity of each method in your implementation in the worst case in terms of Big O notation?
   - Method 1: Constructor
     - The function "AdjacencyList" is the constructor for the "AdjacencyList" class. The purpose of the class is to promote modularity, where a page rank can be calculated multiple times for different iterations using the same calculated adjacency list. The purpose of the constructor is to build the list using a single parameter called "edges" (of type vector containing pairs of strings). Besides the singular for loop, all operations inside the function are O(1) since they either use hash tables/hash sets or are simply created a set number of variables (3 variables are created: edge, from, to) to complete the task. Therefore, the complexity is based on the singular for loop, which iterates over the entirety of a list of edges. Therefore, the worst-case time complexity is O(E), "E" being the number of edges inputted.
   - Method 2: Page Rank Function
     - The function "CalculatePageRank" is the only callable function after instantiation, providing a map of key/value of strings containing the vertex and its page rank with the proper precision and fixed rounding. This function has more components to break down, splitting into 3 main parts.

       First, the "rank_matrix" map that will be used for calculations is initialized, which loops over the "adjacency_list" map. Since the keys of the "adjacency_list" map are the vertices, the worst-case time complexity is O(V), "V" being the number of vertices in the graph.

Second, the calculations for the rank_matrix are performed (n-1) times, since the first value n=1 is the initialized matrix. During each iteration, the entirety of the "adjacency_list" map is looped through, using a double for loop. Ignoring the for loops, all other operations during this stage are O(1), either using hash tables/hash sets or initializing a set amount of new variables. Therefore, the major influencers towards the time complexity are (n-1), V being the number of vertices, and outdegree(V) being the number of edges out of each vertex. After simplification, the worst-time complexity is O(n*V*outdegree(V)).

Finally, the rank matrix is compiled into a map of key/value of strings, for outputting purposes with proper precision and fixed rounding. This uses a singular for loop based on the rank matrix size and all other operations are O(1). Therefore, the worst-case time complexity for this stage is O(V), "V" being the number of vertices in the graph.

The only thing left to do is compile the three complexities into one, joining the three stages and representing the whole function. This complexity without simplification would be O(V + V + n*V*outdegree(V)). Therefore, the worst-case time complexity for the "CalculatePageRank" function is O(n*V*outdegree(V)), "n" being the number or iterations, "V" being the number of vertices in the graph, and "outdegree(V)" being the number of edges going out of each vertex.

3. What is the computational complexity of your main method in your implementation in the worst case in terms of Big O notation?
   - The main method is comprised of three main parts: inputs, calculations, and outputs.

     The input stage creates a set number of variables and iterates a certain number of times based on the initial input, which dictates how many edges will be entered. All operations in this for loop are O(1), including the "push_back" vector function. Therefore, the worst-case time complexity for the input stage is O(E), "E" being the number of edges in the graph.

     The calculations stage is contingent on the "AdjacencyList" object created and called. Since both the constructor and page rank function complexities were already calculated, the two complexities can be added and simplified. Therefore, the worst-case time complexity for the calculation stage is O(E + n*V*outdegree(V)), "E" being the number of edges in the graph, "n" being the number or iterations, "V" being the number of vertices in the graph, and "outdegree(V)" being the number of edges going out of each vertex.

     Finally, the output stage simply iterates over the returned map object from the page rank function, and all operations inside the for loop are O(1). Therefore, the worst-case time complexity of this stage is O(V), "V" being the number of vertices in the graph.

     Compiling the three stages together, the worst-case time complexity is O(E+V+ E + n*V*outdegree(V)), which simplifies down to O(E+ n*V*outdegree(V)) ("E" being the number of edges in the graph, "n" being the number or iterations, "V" being the

number of vertices in the graph, and “outdegree(V)” being the number of edges going out of each vertex).

4. What did you learn from this assignment and what would you do differently if you had to start over?
   - I learned the basic of what is a graph and how to traverse over it. Knowing a self-made implementation helped digest the concept of vertices and edges with directed properties. What I would have done different is change my data structure used to represent the adjacency list. While I do like that its easy to read. The adjacency list uses a hash function every time it wants to be used. Instead, a vector or vectors would have allowed for O(1) operations while not needing to use the hash functions during every iteration of the page rank function, and simple maps from string to int and vice versa could be used to translate the dimensions of the two-dimensional vector.