Kyle Zagers

Data Structures and Algorithms
Project 2 Time Complexity

Describe the data structure you used to implement the graph and why? [2.5 points]

- I used an unordered map with string keys and vectors of strings as the value. I did this because we were supposed to implement an adjacency list, and this is the most efficient way I saw to do it.
- I used a second unordered map with string keys and doubles as the value, which stored the out degree of every page. I did this for easy computing of the rank of pages with an efficient data structure.
- Lastly, I used a map with string keys and doubles as the value, which stored the pages' rank. I used a map because the ranks needed to be sorted by the pages' name.

What is the time complexity of each method in your implementation in the worst case in terms of Big O notation? [5 points]

- PageRank() and PageRank(int power_iterations)
  - Have a worst case time complexity of O(1) because all they do is set variables.
- addLink()
  - Has a worst case time complexity of O(n). This is because it is inserting 2 elements into two different unordered maps which is at worst O(2n). Then it also inserts two elements into a map which is at worst O(2log(n)). This becomes O(2n + 2log(n)) which simplifies to O(n).
  - Average case: O(1).
- initRanks()
  - Starts by calling .size() on a map, pageRank, which is O(1). Then it loops through that map and sets the value of each key equal to 1 divided by the size of pageRank which is at worst O(n).
- rank()
  - calls initRanks() which is O(n)
  - Creates a duplicate of pageRank map which is O(1)
  - Next there is a loop for the number of power iterations which contains a loop that goes through each element of pageRank and sets the previous pageRank map to the new one. Inside the pageRank loop we set the current page's rank to 0, get the adjacency list, and then loop through each string in its adjacency list. While looping through each string in the adjacency list, we set the current page's rank to the previous page rank of the element in the adjacency list divided by its out degree. The inner most loop's internals have a worst-case time complexity of O(n) because of the access of an unordered map. This makes the inner most loop's worst case time complexity O(n*k) where k is the number of pages adjacent to the current. Moving outward the internals of the next loop have time complexity O(n*k) because we are setting a variable, accessing an element of a map, but also doing the previously discussed loop which overrides both. This loop has worst case time complexity of O(k*n^2) because it loops over all pages which is the same degree as the previous n. This makes the final loop have worst case time complexity of O(p*k*n^2) where p is the number of power iterations, because it includes the previously mentioned loop and sets a variable.

    - Overall this function has a worst case time complexity of O(n+ p*k*n^2). This simplifies to O(p*k*n^2) where p is the power iterations, k is the number of adjacent pages to a given page, and n is the number of pages.
    - Average case: O(p*k*n) or O(p*e) because k*n = e which is the number of edges.
- printRank()
    - calls rank() which is O(n+ p*k*n^2)
    - There is a loop which prints the page rank of each element in the pageRank map. This loop has a worst-case time complexity of O(n).
    - Overall this function has a worst case time complexity of O(n+ p*k*n^2+n). This simplifies to O(p*k*n^2) where p is the power iterations, k is the number of adjacent pages to a given page, and n is the number of pages.
    - Average case: O(p*k*n) or O(p*e) because k*n = e which is the number of edges.

What is the time complexity of your main method in your implementation in the worst case in terms of Big O notation? [5 points]

- The first 5 lines of code are getting input and setting variables which is O(1) in the worst case.
- There is then a loop which calls addLink() j times, where j is entered by the user. This has a worst-case time complexity of O(j*n). j is a similar degree to n so this can be simplified to O(n^2).
- Next it calls printRank() which has a worst case time complexity of O(p*k*n^2).
- Overall, main() has a worst case time complexity of O(n^2 + p*k*n^2) which reduces to O(p*k*n^2) where p are the power iterations, k is the number of adjacent pages to a given page, and n is the number of pages.

What did you learn from this assignment and what would you do differently if you had to start over? [2.5 points]

Through this assignment I learned more about the use of maps and unordered maps. Because we were unable to use a matrix to represent adjacency, a map seemed to me like the next best data structure. It has fast average time complexities which I believed would help with large numbers of pages.

If I had to start over there are a few things I would change about my project. When I was thinking of my solution, I did not consider the worst-case time complexity, only the average case. Because of this, I would change the fact that I had both an adjacencyList and outDegree unordered map and try to combine them for less unordered map lookups. I could also change the fact that I had a previous page rank and page rank map and instead just have one with a pair value for less lookups as well. The first change would impact my worst-case time complexity, but the average should stay the same. The second change would impact both the average and the worst-case time complexity if I made the first change.