# Project 2 – PageRank Documentation

Mark Raymond Ploski

COP3530

- The data structure I decided to go with was an adjacency list of a map<string, vector<string>> graph;. I also have maps of map<std::string, float> indexMap; to hold the old ranks and the current ranks. I did this instead of map<string, vector<pair<string, float>> graph; because it seemed simpler to me, that I could just store the ranks in a separate map.

In my project, I have over 8 different methods, most of them being getter functions.

- map<std::string, float>& getUpdatedRanks() : The time complexity of this function is O(1) in big O notation, as it simply returns a reference to the updatedRanks map.

- map<string, vector<string>>& getGraph() : The time complexity of this function is also O(1) in big O notation, as it returns a reference.

- map<std::string, float>& getOldRanks() : This is of time complexity of O(1) as it returns a reference.

- void printRanks(AdjacencyList& list) : The time complexity of this function is O(n log n), where n is the size of the map. It iterates throughout the map and visiting each element of the sorted map, with each visitation taking O(log n) time on avg. std::fixed and std::setprecision is a constant.

- float getdValue(float d) : The time complexity of this function is O(1), as it only returns a reference.

- map<std::string, float>& getIndexMap() : The time complexity of this function is O(1) since it returns a reference.

- void exchange(AdjacencyList& list) : The time complexity of this method is a constant O(1) because it only involves swapping 2 pointers.

- AdjacencyList AdjacencyList::Ranks(int power_iterations, AdjacencyList& list, vector<string>& toStrings) : The time complexity of this function is O(P * V^2), where P is the number of power iterations and V is the number of vertexes in the graph. The outer for loop iterates P times, and the inner for loop iterates over all V vertices in the graph. Within the inner for loop, we do a linear search using std::find which has a time complexity of O(V), and we access the old rank of each vertex using the [] operator, which has an average time complexity of O(1) for an unordered_map. The exchange function has a time complexity of O(1), so it does not affect the overall time complexity. Therefore, the time complexity of the entire function is O(P * V^2).

- int main() : The time complexity of the main() function is O(P * V^2) due to the main calling the Ranks() function. Main also calls printRanks, which is O(n log n). The second and third for loops in main have a time complexity of O(n), where n is the number of elements in the toStrings vector. The first for loop iterates over each element in toStrings and performs a constant amount of work for each element. The if statement inside the loop checks if the current element is already in the graph and if not, inserts it with an empty vector as its value. The second for loop iterates over each element in the list.getGraph() map and assigns an initial rank to each vertex. Since each iteration of the loop performs a constant amount of work, the time complexity of this loop is also O(n). All of these time complexities are less than O(P * V^2), so the final time complexity is O(P * V^2)

- In this assignment, I gained some newfound insight on how search engines such as Google works. I now understand the complexities of ranking websites and how to determine their importance. It also gave me first time experience ever using maps and how to iterate through more complex data structures like adjacency lists. I also gained more experience with using classes in this project, as for the first project I didn’t use any classes and it came back to bite me later on. In project 1, I didn’t use enough modularity and to organize my code enough into their own methods. This time I tried to keep that from happening again, but I think I did it a little too much. Next time I want to find the sweet spot in organizing my code in order to make it more readable.