Alex Eum

Simplified\_PageRank Documentation

Data Structure used

* Two unordered\_maps, one unordered\_set, and one map.
* Vertices – unordered\_set<string>
  + Use – store all of the sites encountered. Used to combat an edge case where a site is reference from but not to and would not be an element in the incoming\_edges map. Thus, use the set of vertices instead to make sure that all sites are traversed during calculations.
  + Key – site
  + Why – set to make it simple to add every encountered site without worrying about duplicates or errors. Unordered because it doesn’t matter what order I use them in in the future, and make use of the O(1) average insert.
* Incoming edges – unordered\_map<string, unordered\_map<string, double>>
  + Use – The main data structure. Used to do the pseudo-matrix multiplication in the page rank calculation.
  + Key – to site
  + Value – unordered\_map< key-from site, value-1/outdegree of from site>
  + Why – Chose a map data structure because I found it more comfortable to use than a vector of vectors, and it is faster in the average case. The order of the components didn’t matter, thus an unordered\_map was better than a map.
* Outgoing edges – unordered\_map<string, unordered\_set<string>>
  + Use – Used to simplify the process of finding a site’s outdegree.
  + Key – from site
  + Value – unordered\_set<key-to site>
  + Why – Chose the unordered map for the same reason as incoming edges. Easy use and additions are babyproofed, so I don’t have to worry about duplicates and errors. The set is used rather than a map for the value because I don’t need to do any math with this map, only need to find the size of the set as fast as possible, which I do with the unordered-ness of the map.
* Page ranks – map<string, double>
  + Use – Stores the rank of each page through each iteration.
  + Key – site
  + Value – key’s rank
  + Why – Used the map because it allows for fast, O(log(n)), access without making anything more complicated than it needs to be. Map rather than an unordered\_map because the output of the program had to be in alphabetical order, which a map implicitly is for strings.

Time Complexities: All cases are in the same order if not explicitly mentioned. Average is equal to best in all cases.

AddEdge: O(1) Avg, O(V) Worst

* Vertices.emplace: Avg-O(1) Worst-O(V)
  + Add to unordered set is O(1) on average, O(size of container) at worst
* Incoming\_edges.find: Avg-O(1) Worst-O(V)
  + Same as emplace for set
* Edge.emplace: Avg-O(1) Worst-O(V)
* Incoming\_edges.emplace: Avg-O(1) Worst-O(V)
* Only more finds and emplaces: All Avg-O(1) Worst-O(V)

calcEdgeValues: O(E \* V \* log(V))) Avg, O(E \* V^2) Worst

* Loop through to vertices: O(V)
  + Page\_ranks.emplace: O(log(V))
  + Loop through edges connected to first vertex: O(E)
    - Calls to unordered\_map at(), find(): Avg-O(1) Worst-O(V)
    - Call to page\_ranks.emplace: O(log(V))
  + Average: O(Log(V) + (E\* log(V)))
    - Equivalent to O(E\*log(V))
  + Worst: O(Log(V) + E\*V)
    - Equivalent to O(E\*V)

pageRanks: O(P \* V^2 \* log(V)) Avg and Worst

* Loop # of iterations times: O(P)
  + Loop through all of the sites: O(V)
    - Unordered find: Avg-O(1) Worst-O(V)
    - Loop through all of the sites: O(V)
      * Ordered emplace: O(log(V))

printEdges: O(V \* E \* length of site name) – used for testing

* Loop through all of the to sites: O(V)
  + Cout: O(length of site name)
  + Loop through sites connected to the to site: O(E)
    - Cout: O(length of site name)

printRanks: O(V \* length of site name)

* Loop through all of the sites: O(V)
  + Cout: O(length of site name)

main:

Avg: O(E + (E \* V \* log(V)) + (P \* V^2 \* log(V) + (V \* length of site name))

Worst: O((E\*V) + (E\*V^2) + (P\*V^2\*log(V)) + (V\*length of site name))

As E is on the same order as V (graph is given to be sparse) –

Avg: O((P \* V^2 \* log(V)) + (V \* length of site name))

Worst: O((V^3) + (P\*V^2\*log(V)) + (V\*length of site name))

* Loop through number of edges: O(E)
  + AddEdge: O(1) Avg, O(V) Worst
* CalcEdgeValues: O(E \* V \* log(V)) Avg, O(E\*V^2) Worst
* pageRanks: O(P \* V^2 \* log(V))
* printRanks: O(V \* length of site name)

Reflection:  
If I were to redo this assignment, I would take a couple more minutes to think through the necessities of the project. As an example, I started the assignment using only one of the two unordered\_maps. I did not consider that it would be difficult to find the outdegree of sites when only having the incoming edges. When I realized this difficulty, I added the second unordered map. I think that it was the best option, in terms of time complexity, but I’m not entirely sure. It would have been smarter for me to make sure that this was the best option before doing a significant amount of work and essentially blocking myself into only having one option as a remedy after figuring out that I made a mistake. Additionally, I used the map rather than the unordered map for the page ranks, going off the idea that it was already sorted. However, it was probably smarter to take advantage of the O(1) emplace average time in the more complex calculations and take the hit when printing the ranks.