Alex Eum

Simplified_PageRank Documentation

Data Structures used

- One vector of unordered maps, Four unordered maps
- edges– vector<unordered_map<int, double>>
    - Use – Store all of the edges encountered, specifically tracking the incoming edges to any site along with the weight of the edge. Also used to do the pseudo-matrix multiplication in the page rank calculation.
    - Index – The integer equivalent of the referenced site.
    - Element – unordered_map (key: integer equivalent to the referencing site, value: weight of the edge ie 1/outdegree)
    - Why – The vector allowed for O(1) insertions and lookups, and the unordered_map uses less space then a vector of vectors while still providing O(1) lookups, along with a second spot (the value) for the weights to be.
- edgeOutDegrees – unordered_map<int, int>
    - Use – An unordered_map used to track the outdegree of each site.
    - Key – site's integer representation
    - Value – outdegree of key
    - Why – The map wasn't necessary, a vector could have worked just as well. Unordered to provide O(1) average lookups.
- intToSite– unordered_map<int, string>
    - Use – provides a mapping between integers and site names. Each site is mapped to an integer when they are first encountered, and this is the complement to siteToInt.
    - Key – site's integer representation
    - Value – site's name
    - Why – The necessity of this map was due to the O(n) nature of cpp's hashing function with strings. Had I used strings as the keys to all of the other unordered_maps, I would have been unable to take advantage of the O(1) lookup time due to the hash function being linear to the site name length.
- SiteToInt – map<string, int>
    - Use – provides a mapping between a site name and an integer. Used to map a site to an integer, complemented by intToSite.
    - Key – site name
    - Value – site's integer value.
    - Why – Once again used to avoid the O(n) relationship between the hash function and string length. This function is only used while adding edges (at O(length of site name), and afterwards sites are only referred to by their integer mapping.

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

AddEdge: O(length of site name) Average, O(length of site name * V) Worst

- No Loops
- Edges - at: O(1), map operations: O(1) Average, O(V) Worst
- siteToInt – all operations: O(length of site name) Average, O(length of site name * V) Worst
- intToSite – all operations: O(1) Average, O(V) Worst
- edgeOutDegrees – all operations: O(1) Average, O(V) Worst

calcEdgeValues: O(V^2) Avg, O(V^3) Worst

- Loop through all vertices: O(V)
    - Page_ranks.emplace: O(1) Average, O(V) Worst
    - Edges – at - size: O(1)
    - Loop through edges connected to first vertex: O(V)
        - edgeOutDegrees.at – O(1) Avg, O(V) Worst

pageRanks: O(P * V^2) Avg, O(P * V^3) Worst

- Loop # of iterations times: O(P)
    - Loop through all of the sites: O(V)
        - Loop through all sites connected to first vertex: O(V)
            - PageRanks – at: O(1) Avg, O(V) Worst
        - NewRanks – emplace: O(1) Avg, O(V) Worst

printEdges: O(V^2 * length of site name) – used for testing

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

printOutDegrees: O(V * length of site name) Avg, O(V^2 * length of site name) Worst– used for testing

- Loop through all of the sites: O(V)
    - intToSite – at: O(1) Avg, O(V) Worst
    - Cout: O(length of site name)

printRanks: O(V * length of site name) Average, O(V^2 * length of site name) Worst

- Loop through all sites: O(V)
    - Ordered_ranks – emplace: O(length of site name) Avg, O(length of site name * V) Worst
- Loop through all sites: O(V)
    - Cout – O(length of site name)

main:

Avg: O((E * Length of site name) + V^2 + (P*V^2) + (V * Length of site name))

Worst: O((E * Length of site name) + V^3 + (P*V^3) + (V^2 * Length of site name))

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

Avg: O((P*V^2) + (V*Length of site name))

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

- Loop through number of edges: O(E)
  - AddEdge: O(length of site name) Average, O(length of site name * V) Worst
- CalcEdgeValues: O(V^2) Avg, O(V^3) Worst
- pageRanks: O(P * V^2) Avg, O(P * V^3) Worst
- printRanks: O(V * length of site name) Average, O(V^2 * length of site name) Worst

Reflection:
If I were to redo this assignment, I would take a couple more minutes to think through the necessities of the project.

I redid this assignment after realizing that I had made a mistake that significantly increased my time complexity. Specifically, I didn't have the intToSite and siteToInt maps in my first attempt at the project. After running some stress tests, I realized that my code was too slow, mainly due to the O(n) relationship between site name length and the hash function that cpp uses. After realizing this, I almost completely redid the project, which was unfortunate. So, I should have taken time to realize that the integer representation was necessary to reduce the time complexity.

Additionally, I would have used a vector of vectors to contain all of the edge out degrees, as this would have made it possible to handle parallel edges.