Analysis and Design of Algorithms Lab Project Report

Data Structures:

-vector<vector<string>> SiteDB = WebKeyword. 2D string vector carrying the websites names and keywords.

Space Complexity:

Number of websites n x number of keywords m = n x m. O(n2)

-vector<vector<string>> WebG = WebGraph

Space Complexity:

Number of websites pointing at other website so O(n).

-map<string, vector<int>> M1 = Map for storing website name, index, number of incoming edges, number of outgoing edges (Fixed to 2 as said in the powerpoint).

Space Complexity:

The number of websites = n. 3xn = O(n).

-map<int,vector<int>> M3 = Map for storing index + index of vertices pointing at it.

Space Complexity:

Number of websites = n. O(n).

-map<int, float> PR = map for storing page rank.

Space Complexty:

Number of websites =n. O(n).

-vector<vector<string>> AM = 2d vector for storing the impressions and clickthrough rate.

Space Complexity:

Number of websites=n. 2n. O(2n)=O(n).

-map<string, vector<int>> AM2 = map containing the name of the website and a vector containing the impressions and clickthrough rate.

Space Complexity :

Number of websites =n. O(2n)=O(n).

-map<string, float> R = map the stores the name of the website and its score.

Space Complexity:

Number of websites =n. O(n).

-map<float, int> F= map used for sorting the website indecies according to their score.

Space Complexity:

Number of websites =n. O(n).

-map<string, vector<int>> Klist= map used for storing the keywords and index of the website that has them.

Space Complexity:

The number of keywords=n. Number of websites that share the same keyword=m. mn. O(mn).

-vector<vector<int>> IList=2d int vector that contains indecies for website that have the keyword we searched for.

Space Complexity:

Number of websites that have the solution that we are looking for = n. O(n).

-vector<vector<string>> SRes= 2d string vector the includes the result in string form for displaying the results.

Space Complexty:

Number of results for keyword 1=n. Number of results for keyword 2=m. O(nm).

Functions:

-void LoadFiles(vector<vector<string>>& M): loads from CSV to 2D vector.

Pseudocode:

Opens the WebKeyword.csv file, loops over each line in the file. In side the loop, we loop over each letter of the line while adding it to a string until we find a comma, we then push the string into a 1D string vector. Once the line is complete, we push the 1D string vector into a 2D string vector in order to store the keywords of multiple websites at the same time.

Time Complexity:

The number of lines in the csv files = n. The number of letters in a line = m. The number of filled coloumns in the csv file = p. n(m+p). O(n2)

-void PrintS(vector<vector<string>>& M): prints 2D string vector

Pseudocode:

Iterates over every row and every column, prints out the content of the row/ column being pointed at.

Time Complexity: O(n2)

-void LoadWebG(vector<vector<string>>& M): loads from CSV to 2D vector

Identical to void LoadFiles(). Contains an additional if condition because it was not loading the last column correctly.

-void LoadWebM(vector<vector<string>>& S, map<string, vector<int>>& K): initializes M1 using WebG

PseudoCode:

Initializes a string, vector of integer map and a vector of integers m. We loop 3 times to fill the vector of integers with zeros. We then iterate over the size of the 2D string vector and change the first column to the index number, 3rd column to 2, the number of outgoing edges, we then emplace the name of the website and the integer vector we made in the map.

Time Complexity:

Number of websites =n. 3+n(log(n))= O(nlog(n)).

-void PrintWebM(map<string, vector<int>>& K): prints M1

Same as PrintS() but takes a different data type.

-void CalcWebM(vector<vector<string>>& S, map<string, vector<int>>& K, map<int, vector<int>>& J): calculates the number of incoming edges and adds it to M1. Uses WebG, M1,M3.

Psuedocode:

We initialize a integet vector. The function Iterates over the WebG 2D string vector, uses the map find function of M1 to find the websites that each website in the webgraph is pointing at and updates their number of incoming edges. It also pushes the index of the webpage that we are currently at inside the integer vector. We then use the find function on map M3 to see if the index exists or not. If it exists, we push the current index to the vector of indices inside the map M3, else we push the vector we initialized and the index into the map before clearing the vector for future use.

Time Complexity:

Number of websites =n. Number of websites each website is pointing at = m. n(m(log(n)+m+m)) = O(n2log(n)).

-void PRinit(map<string, vector<int>>& K, map<int, float>& M): initializes the PR map to 1/30.

Pseudocode:

Iteratres through the map M1 and emplaces the index of each website and sets its page rank to 1/30.

Time Complexity:

Number of websites =n. O(n).

-void PrintPR(map<int, float>& M): prints the PR map.

Pseudocode:

Iterates over the PR array and couts the contents.

Time Complexity:

Number of websites = n. O(n).

-void PrintMi(map<int, vector<int>>& M): prints map M3.

Pseudocode:

Iteratres over the indecies stored in the map and then iterates over the contents of the key vector.

Time Complextiy:

Number of websites =n. Number of websites pointing at a certain website =m. n(m). O(n2).

-void CalcPR(vector<vector<string>>& S, map<string, vector<int>>& K, map<int, vector<int>>& M,map<int, float>& J): takes WebG (Webgraph as 2D vector), M1(Map with website names as keys, index, number of incoming edges, and number of outgoing edges), M3 (Map with website indecies as keys), and PR (Map with website indecies as keys, page rank as values) and calculates the page rank. Does 2 iterations of the calculation.

Pseudocode:

Does the following twice: iteratres through the map M1. Initializes the sum to zero. Searches for the website from M1 in M3 using the find functions and the index. If a result is found, it loops through the key vector of M3 in order to see the websites pointing at M1. We then search for the page rank of these websites in the PR map using the find function. We then divide the page rank by 2, the number of outgoing edges for all websites, and we add it to the sum. We then update the page rank of the current page.

Time Complexity:

The number of websites =n.The number of websites pointing at the current website =m. 2(n(log(n)+m(log(n)+3)))) = 2nlog(n)+2mnlog(n) = O(mnlog(n)).

-void LoadWebI(vector<vector<string>>& M): loads the WebIMP csv file to AM.

Identical to LoadWebG.

-void LoadWebIM(vector<vector<string>>& S, map<string,vector <int>>& K): initializes AM2 using AM

Pseudocode:

Initializes an integer vector, loops over every website, uses stoi() to turn the impressions and clickthrough rate to integers before inserting them into the map AM2.

Time Complexity:

Number of websites =n. n(n)= O(n2).

-void PrintWebM(map<string, vector<int>>& K) prints AM2

Identical to print Mi.

-void SCORE(map<string, vector <int>>& K, map<int, float>& M, map<string, vector <int>>& J, map<string, float>& F): takes M1(website name, index, numbers of incoming edges, outgoing edges), PR(index and page rank), AM2(name and impressions and clickthrough rate), and R (name of the website and score)and initializes R.

Pseudocode:

Loops over M1, initializes the score to zero, uses the algorithm in the powerpoint in order to calculate the score. Places the score in the score map for sorting it.

Time Complexity:

Number of websties =n. O(n).

-void PrintScore(map<string, float>& F): prints R.

Identical to PrintPR().

-void SCSF(map<string, vector <int>>& K, map<string, float>& F, map<float,int>& Q): takes M1, R, and F in order to initialize F.

Pseudocode:

Loops over map M1, searches for the website in R in order to get the score. Inserts the score and index into F.

Time Complexity:

Number of websites=n. O(nlog(n)).

-void PrintSC(map<float, int>&Q): prints F.

Identical to PrintPR().

-void FinalRank(map<float, int>& Q, vector<vector<string>>&M) takes the score and index of the website stored in F and the website keywords vector SiteDB and sorts it according to the score.

Pseudocode:

Initializes a 2D string vector. Iterates from the end of the score map and inserts the correct website into the map according to their scores.

Time Complexity:

Number of websites=n. O(n).

-void LoadMap(map<string, vector<int>>& K, vector<vector<string>>& M): initializes Klist from SiteDB.

Pseudocode:

We iterate over the rows and columns of the 2D string vector SiteDB. We then use the find function on each keyword in the map. If it is not in the map, we initialize a vector and place the keyword and the index of the website that has it, else we add the index to the index vector of the keyword in the map.

Time Complexity:

Number of websites=n. Number of keywords =m. n(m(log(n)+n))=O(n3).

-void PrintM(map<string, vector<int>>& M): prints Klist.

Identical to PrintMi().

-void SearchPreProcess(string Query, map<string, vector<int>>& K, vector<vector<int>>& M): the function that does the preprocessing, searching, and post processing.

Pseudocode:

We check if the search query has double quotes, AND, or OR. There are 3 cases, we will analyze the longest one, the AND search. We use the string find function to find the location of the AND. We take the words before and after AND and make copies of them. We then use the SearchQuery2() function to search for them and place the results in the 2d integer solution vector. We then change the casing of the keywords and search for their opposite case using the CapitalizeS() function. We then call the function ValidateDup() which removes any websites that have both the lower and upper case of a keyword and merges the rest with each other. We then call a second validation function ValidateRVAND2() to check which websites have both keywords and remove the rest. Replace ValidateRVAND2() with ValidateDup2() for the OR searching.

Time Complexity:

Number of letters in the query string =n. Length of string returned by the substr function=m. Number of solutions=l. Number of duplicate solutions=z. Log(n)+m1+m2 + SearchQuery2() + SearchQuery2() + CapitalizeS() + SearchQuery2() + SearchQuery2() + ValidateDup() + ValidateRVAND2() = O(l(m)).

-void SearchQuery2(string Query, map<string, vector<int>>& K, vector<vector<int>>& M) the searching function takes a keyword and fils the 2d integer solution vector with the indecies of websites that have the searched for keywords.

Pseudocode:

Use the find function on the map Klist in order to search for matching keywords. If we find nothing we add an empty vector to our solution vector or the vector of indicies that have the right keyword.

Time Complexity:

Number of keywords=n. O(log(n)).

-void CapitalizeS(string& S, string &S2): takes the 2 keywords in and and or searching, capitalizes them to search for both capital and small versions of the keywords.

Pseudocode:

We check the first letter if it is capital or small, we then loop over the string and change from small too capital or capital to small.

Time Complexity:

Number of letters in each string=n. O(n).

-void ValidateDup(vector<vector<int>>& M): removes any duplicate websites when a website contains the lower and higher case of the same keyword.

Pseudocode:

Iterates over the 2d solution vecor IList. It compares rows 0 and 2 and rows 1 and 3 using binary search. If it finds a common index, it deletes it.

Time Complexity:

Number of solutions for key word1= n. Number of solutions for keyword2=m. n(log(n))+m(log(m)). O(nlog(n)).

-void ValidateDup2(vector<vector<int>>& M): removes any websites that have both keywords.

Pseudocode:

We initialize a 2d integer vector, we then merge row 0 and 2 from the solution index vector and 1 and 3 with each other into it. We then loop through the new merged vector and remove any duplicates between row 0 and 1 using binary search.

Time Complexity:

Number of solutions for keyword 1=n. Number of solutions for keyword 2=m. Number of duplicate indecies=l;.n+m +m+n(log(m+n)+(l)). O(m+n(l)).

-void ValidateRVAND2(vector<vector<int>>& M): validates that that both keywords are present on websites for the and search.

Pseudocode:

Identical to ValideDup2() except we do binary search twice to check if the website contains both keywords using an additional binary search.

Time Complexity:

Identical to ValideDup2()

-void PrintI(vector<vector<int>>& I): prints the solution 2d int vector with the indecies of the solution.

Identical to PrintPR() just for a different data type.

-void LoadSR(vector<vector<int>>& I, vector<vector<string>>& M, vector<vector<string>>& S, map<string,vector<int>>&J): initializes SRes from IList and SitDB and AM2.

Pseudocode:

We create a integer vector. We then merge both rows of Ilist into it. We then iterate over this new vector and use the indecies in it to push each website into the solution vector SRes from the SiteDB. We then use the find function on AM2 to change the impressions of the websites we will display in our search results.

Time Complexity:

Number of solutions =n. Number of websites =m. n+n(log(m)). O(nlog(m)).

-void PrintSR(vector<vector<string>>& S): prints SRes.

Identical to PrintMi() just uses a different data type.

void SaveN(map<string, vector <int>>& K, vector<vector<string>>&M): saves the impressions and clickthrough rate to the file.

Pseudocode:

Opens the file WebIMP.csv, iterates using a for loop through the 2d string vector (AM) that contains the website names, impressions, and clickthrough rate in order to access the map containing the website name, impressions, and clickthrough rate (AM2).

We use AM to move through AM2 because AM2 does not have the original order of the WebIMP file. We save the impressions and clickthrough rate from AM2 rather than AM because the impressions and clickthrough rate are updated in the map AM2 not the 2d vector AM.

Timecomplextiy: n=the size of the 2d vector. The map find function works in log(n) time. n x log(n).

-void Opensite(vector<vector<string>>& S, int k,map<string, vector<int>>& K): opens website to change the CTR.

Pseudocode:

Checks if the index of the website is larger than the solution vector, else it uses the find function to update the clickthrough rate of the website.

Time Complexity:

Number of soltions =n. O(log(n)).

Standard Algorithms Used:

-std::istream.clear(): Constant.

-std::istream.ignore(): Constant.

-std::map::find(): “Logarithmic in size”. (cplusplus.com)

-std::map::erase(): “Amortized Constant” in my use case. (cplusplus.com)

-std::getline(): “Generally Linear”. (cplusplus.com)

-std::vector::push\_back(): “Constant”. (cplusplus.com)

-std::vector::erase(): “Linear is size”. (cplusplus.com)

-std::map::emplace(): “Logarithmic in container size”. (cplusplus.com)

-std::stoi(): “Generaly linear in the number of characters.” (cplusplus.com)

-std::string::substr(): ”generally linear in the length of the returned object.” (cplusplus.com)

-std::string::find():”Linear up to length.” (cplusplus.com)

-cctype::isupper()/islower(): Constant.

-cctype::tolower()/toupper(): Constant.

-std::merge(): “Up to linear”. (cplusplus.com)

-std::map::begin()/end(): Constant. (cplusplus.com)

-std::find(): Linear. (cplusplus.com)

-std::binary\_search(): Logarithmic (cplusplus.com)

Tradeoffs:

-Large space complexity.

-File manipulation greatly increases the time complexity.

-Searching is logarithmic; however, the preprocessing and validation take a lot of additional time and complexity.

Works Cited

“Std::Binary\_search.” *CPlusPlus*, https://cplusplus.com/reference/algorithm/binary\_search/. Accessed 3 Dec. 2022.

“Std::Getline.” *CPlusPlus*, https://cplusplus.com/reference/string/string/getline/.Accessed 3 Dec. 2022.

“Std:: map::find().” *CPlusPlus*, https://cplusplus.com/reference/map/map/find/.Accessed 3 Dec. 2022.

“Std::string::substr().” *CPlusPlus*, https://cplusplus.com/reference/string/string/substr/.Accessed 3 Dec. 2022.

“Std::string::find().” *CPlusPlus*, https://cplusplus.com/reference/string/string/find/.Accessed 3 Dec. 2022.

“Std::npos.” *CPlusPlus*, https://cplusplus.com/reference/string/string/npos/.Accessed 3 Dec. 2022.

“Std::map::begin().” *CPlusPlus*, https://cplusplus.com/reference/map/map/begin/.Accessed 3 Dec. 2022.

“Std::map::erase().” *CPlusPlus*, https://cplusplus.com/reference/map/map/erase/.Accessed 3 Dec. 2022.

“Std::cctype::islower().” *CPlusPlus*, https://cplusplus.com/reference/cctype/islower/.Accessed 3 Dec. 2022.

“Std::merge().” *CPlusPlus*, https://cplusplus.com/reference/algorithm/merge/.Accessed 3 Dec. 2022.

“Std::find().” *CPlusPlus*, https://cplusplus.com/reference/algorithm/find/.Accessed 3 Dec. 2022.

“Std::vector::erase().” *CPlusPlus*, https://cplusplus.com/reference/vector/vector/erase/.Accessed 3 Dec. 2022.

Andrist, Bjö, and Viktor Sehr. *C++ High Performance : Boost and Optimize the Performance of Your C++17 Code*, Packt Publishing, Limited, 2018.*ProQuest Ebook Central*, <https://ebookcentral.proquest.com/lib/aucegypt/detail.action?docID=5259448>.

“Binary Search Functions in C++ STL (Binary\_search, Lower\_bound and Upper\_bound).” *GeeksforGeeks*, 23 Apr. 2017, https://www.geeksforgeeks.org/binary-search-functions-in-c-stl-binary\_search-lower\_bound-and-upper\_bound/.

Byers, Mark. “Answer to ‘How Can I Get Double Quotes into a String Literal?’” *Stack Overflow*, 9 Sept. 2012, https://stackoverflow.com/a/12338826.

*C++ <string> Push\_back() Function - AlphaCodingSkills*. https://www.alphacodingskills.com/cpp/notes/cpp-string-push-back.php. Accessed 3 Dec. 2022.

*C++ Tolower() - C++ Standard Library*. https://www.programiz.com/cpp-programming/library-function/cctype/tolower. Accessed 3 Dec. 2022.

*C++ Toupper() - C++ Standard Library*. https://www.programiz.com/cpp-programming/library-function/cctype/toupper. Accessed 3 Dec. 2022.

*Google’s PageRank Algorithm: Explained and Tested*. https://www.link-assistant.com/news/google-pagerank-algorithm.html. Accessed 3 Dec. 2022.

“Isupper() and Islower() and Their Application in C++.” *GeeksforGeeks*, 30 Aug. 2017, https://www.geeksforgeeks.org/isupper-islower-application-c/.

“Merge() in C++ STL.” *GeeksforGeeks*, 26 Aug. 2017, https://www.geeksforgeeks.org/merge-in-cpp-stl/.

Moscow, Vlad from. “Answer to ‘C++ 2D Vector(Matrix) How to Delete the Nth Row?’” *Stack Overflow*, 30 Dec. 2021, https://stackoverflow.com/a/70536154.

none. “Finding an Element in a 2d Vector and Then Removing It.” *Stack Overflow*, 20 Oct. 2012, https://stackoverflow.com/q/12993312.

“Page Rank Algorithm and Implementation.” *GeeksforGeeks*, 30 Aug. 2017, https://www.geeksforgeeks.org/page-rank-algorithm-implementation/.

Pandey, Avinesh. “Answer to ‘How Can I Resize a 2D Vector of Objects given the Width and Height?’” *Stack Overflow*, 22 Dec. 2020, https://stackoverflow.com/a/65407963.

programing\_is\_hard. “C++ Reading Csv File.” *Stack Overflow*, 12 Nov. 2013, https://stackoverflow.com/q/19936483.

Prowl, Andy. “Answer to ‘Merits of Std::Find.’” *Stack Overflow*, 22 June 2013, https://stackoverflow.com/a/17256035.

Sharma, Ayush. “Set in C++.” *Scaler Topics*, 1 July 2022, https://www.scaler.com/topics/cpp/set-in-cpp/.

Singh, Prabhdeep. “Map in C++.” *Scaler Topics*, 31 May 2022, https://www.scaler.com/topics/cpp/map-in-cpp/.

“Unordered Sets in C++ Standard Template Library.” *GeeksforGeeks*, 24 Mar. 2016, <https://www.geeksforgeeks.org/unordered_set-in-cpp-stl/>.

https://cs.brown.edu/courses/cs016/static/files/assignments/projects/GraphHelpSession.pdf