Data Structures PA1 Report

B07202020

Hao-Chien Wang

December 7, 2020

1 Usage

Use `./bin/pagerank <DIFF> <d>` to run the program. The input files are placed under `./input/` and the output files are generated under the main folder (`./`). The scripts are placed under `./src/` and can be compiled using `make`. You can also run the script `./runall.sh` to run for all input combinations. Use `make clean` to clean output files, executables and `*.o` files.

2 Data Structures and Algorithms

I wrote my program in c++. I defined 2 classes: `Page` and `FindPageRank`. The foremer stores the index of the page (for example, the index of `page499` is 499, saved as `int`), the number of outbranching links (`int`), pages it links to (`vector<int>`, storing indices of the pages) and its pagerank (`double`). The latter reads the input, compute pageranks and output the requested files. It stores the following information:

1. Both input values `DIFF` and `d` (as `double`)
2. A wordlist (`map<string, set<int>*>`) with the string being all the words mentioned in the pages and the set (that the pointer points to) saving the indices of the pages that mentioned the word.
3. An array of pages, saving all the `Pages` objects.

All the above are defined in `src/pagerank.h` and `src/pagerank.cpp`. The output is generated in the following steps:

1. Read input, create `Page` objects, generate wordlist and `reverseindex.txt` (done by the constructor of `FindPageRank`).
2. Iterate until the terminating conditions are met (done by `FindPageRank.iterate()`).
3. Output `pr_xx_yyy.txt` (done by `FindPageRank.printPR()`).
4. Read `list.txt`, search in wordlist according to it and output `result_xx_yyy.txt` (done by `FindPageRank.search()`).

3 Complexity Analysis

The space complexity is $O(n^2)$ since we need to save all the pages, and the size of each pages is $O(n)$ (assuming sorting does not use memory exceeding $O(n^2)$). To analyze time complexity, we first examine the complexity of each steps. Note that for simplicity we only consider the worst case scenario.

1. Reading input
   Reading the outbranching links takes $O(n^2)$ since we need to read all the input files and save them in our data structures. Reading the words mentioned in each pages is obviously $O(nw)$, where w is the words mentioned by each pages. Writing `reverseindex.txt` also takes $O(nw)$. Therefore the time complexity of this step is $O(n^2 + nw)$.

2. Iteration
   This step is a iteration of linear transformations, which we compute the expression:

   $$M(M(M\cdots(M\vec{v}_0+\vec{v}_1)+\vec{v}_1)+\vec{v}_1)\cdots+\vec{v}_1 = M^n\vec{v}_0 + \left(M^{n-1}+M^{n-2}+\cdots+M+I\right)\vec{v}_1$$

   where $\vec{v}_0$ is the initial page rank vector and M and $\vec{v}_1$ are the parameters related to the links. This expression converges exponentially fast (note that the second term is a geometric series). Therefore the time complexity of this step is $O(-\log \mathrm{DIFF})$. (I have omitted the dependency on d since it is rather complex, but it shouldn't affect much on the time complexity.)

3. Print pagerank outputs.
   The time complexity of this step is obviously $O(n)$.

4. Searching
   Sorting the results takes $O(n\log n)$ for both single-word and multiple-word. Since `map` in STL is implemented with an RB tree, it also takes $O(\log n)$ time to find the word in the wordlist. For a m-multiple-word search, Finding the union takes $O(mn)$, and finding the intersection takes $O(mn\log n)$. Therefore the overall time complexity of this step is

   $$\begin{aligned} & O\left(\underbrace{s_1 n\log n}_{\text{single-word search}} + \underbrace{\sum_m s_m mn\log n}_{m\text{-multiple-words search}}\right) \\ =& O\left(\sum_m s_m mn\log n\right) \end{aligned}$$

   where s_m is the number of m-multiple-word searches.

From the analysis above, we can see that the total time complexity is:

$$O\left(n^2 + nw - \log d - \log \mathrm{DIFF} + \sum_m s_m mn\log n\right)$$