Ali Ashraf

CSCE220301 - Anlys and Design of Algorithms Lab (2021 Fall)

Dr Mohamed Alhalaby

Sunday, 21 November 2021

Search Engine Report

Table of Contents

**No table of contents entries found.**

# Data Structures

The following is a description of the major data structures that were used throughout the implementation of the search engine.

**Acknowledgement**: while the implementations of classes as data containers might be memory-inefficient for high scale operations, this effect rests negligible given the small-scale nature of such an assignment.

## WebPage class

Was constructed to define and associate attributes and, inherently, all the accommodating operations such including updating and retrieving from a centralized entity - that is, the WebPage object. Besides that, implementing the Object-Oriented Programming allowed for easy debugging and easy maintainability of the code.

## Graph class

Was also defined to minimize code deficiencies, including code duplications. The Graph class receives a set of edges – comprised of source and destination nodes and populates an adjacency list - technically a vector of vectors - accordingly such that PageRank can, subsequently, function suitably.

## Trie class

The Trie class was constructed, likewise, to minimize code deficiencies. The Trie data structure was chosen as a storage and retrieval medium of keywords, which is arguably the most efficient and apt data structure for said function. Specifically, they are used in the implementation of text-correcting software and dictionaries. Trie is a digital tree the stores strings as character-nodes where every node represents a character whose children are the possible character combinations (the maximum size of which is the alphabet size). Therefore, insertion and search operations depend strictly on the length, ***L***, of the search query, resulting in a time complexity of ***O(L)***. Each node was stored as a struct of

Bool isLeaf indicates whether the current node is a leaf node, which corresponds to a complete string or word. If so, the insert function append to the empty vector the webpage URL.

### Positive implications

* Insertion and search are done in a constant time of Θ(L) which is rather fast and more efficient than mere arrays and the likes of self-balancing trees such as binary-search Trees and AVLs.
* By contrast to self-balancing trees, it does not require hashing and, consequently, evades the tiresome, time-consuming collision-handling process; therefore, it is faster.

### Drawbacks

* Seeing as every node in Tries contains a huge number of other node pointers – equal to the 24, the size of the alphabet – one can conclude that Tries are not memory-efficient and consume a lot of unnecessary space as a string-storing data structure.

## Unordered Map of WebPages

Storing the WebPage instantiations demanded a data structure that allows for a nearly constant access time. Hence, an unordered map was the ideal choice seeing as they are associative containers – that is, it stores key-mapped values. Additionally, this was optimal since I was able to associate WebPage URLs as unique identifiers that distinguishes a page from another and accordingly access, manipulate, or retrieve data from said WebPages conveniently by their names and in an average time complexity of ***O(1)***. It was declared as a global variable such that it remains accessible for other classes, namely the Graph class.