Data Structures and Computational Time Complexity
CS 4750 – Natural Language Processing

Ryan Martin

In Natural Language Processing (NLP), most, if not all practical applications require the use of a lexicon, defined as the vocabulary of a language, to carry out operations such as speech generation and recognition. Lexicons, because they hold every word in a language, are often massive in size and are therefore cumbersome to work with. However, lexicons are fundamental to NLP, and there are efforts to not only deal with these tremendous amounts of data, but to augment them using various data structures so that operations are efficient. In this paper, I will discuss several distinct data structures which will represent a lexicon as well as algorithms and other techniques developers use to work with these structures at the software level. I will also compare commonly used data structures along with their time and space complexities of differing operations and suggest the best structure for a lexicon. Finally, I will explain the importance of working with an efficient lexicon in relation to an NLP application to portray the improvements of a time- and space-competent structure. Please note that the code portion of the project is included with the submission.

When implementing a lexicon, the software developer has many options available to him/her with respect to data structures and algorithms used in representing their lexicon, but their options are filtered when they take into account the constraints their current system is under. Such limitations are determined by the amount of processing power and overhead space the executing machine has, both of which NLP operations oftentimes heavily rely upon.

Regardless of the operation(s) or the reason(s) that are needed for a lexicon, those working in the field of NLP are all too familiar with the time/space complexity paradigm, listed as another limitation whereby the time and space complexities are inversely proportional to each other. For instance, if we want to make our current data structure perform actions more quickly (lower time taken to complete an operation), then the amount of required overhead space increases, and vice versa.

As previously mentioned, a developer has endless possibilities when it comes to selecting a data structure to serve as a lexicon, assuming we have reasonable amounts of processing power and overhead space. Given this, three traditional data structures used include an array, a graph, or a 2-3 tree. While all three structures store a collection of elements, they all do so in a varying fashion which arise differing operation time and space complexities. A standard one-dimensional array stores elements in consecutive positions, e.g., [..., timber, chair, bottle, ...] and because of this, an array's space complexity is O(n), where n is the number of elements in the array. The time complexity for its operations are as follows: search O(n), and index O(1) (Wikipedia: Array Data Structure). With respect to the graph structure using an adjacency list, its operations and times are: space O(|V| + |E|), add vertex O(1), add edge O(1), remove vertex O(|E|), and remove edge O(|E|) where the vertical brackets denote the number of items (Wikipedia: Graph Abstract Data Type). Finally, a 2-3 tree requres O(n) space and has the functions: search O(log n), insert O(log n) and delete O(log n) (Wikipedia: 2-3 Tree). From this, we can see why a 2-3 tree is the best structure to use in the general case.

The most frequent method of implementing a lexicon at the software level is by using a single data structure; nonetheless, any software developer is given the freedom to use multiple structures for use with a single lexicon. This is known as utilizing a fusion of data structures by which several structures are used with operations from either structure to result in overall optimal time (while less often space) complexities. In this situation, the developer will need to write additional code to connect each data structure and ensure that specific operation(s) from each one are used, and potentially revise how the client code interacts with the data structure fusion.

Thus far, the NLP applications and concepts discussed have been related to the English language; when we look at other languages and their lexicons as well as the varying semantics and syntax, we realize that the sentence structure certain languages alter greatly from English, the language we are familiar with. For instance, in Chinese and Japanese, words are arranged continuously in a phrase whereas words of a sentence in English are separated by whitespace. Difficulty entails these languages when they are mixed with NLP operations, as word segmentation must be done foremost in order to analyze the phrase and complete other functions such as semantic derivation (Ge, Ma and Chen). Segmentation requires the traversal of the sentence and the splitting up of words in the phrase by matching strings in the Chinese lexicon (Ge et al.); therefore, we need to search the lexicon multiple times to match any current string of the phrase, thus adding time to complete the operation, but also stressing the importance of utilizing a data structure with time-efficient methods (Ge et al.).

An everyday obstacle when dealing with lexicons is their substantial size and required space; moreover, this is expected considering that a lexicon holds all words from a natural language. Overhead space is a concern when our machine has limited available space, or when we need to transfer lexical data over a network, but there are certain data compression algorithms we can call to reduce space complexity of a given lexicon. One such technique is tree compression for lexicons represented as a tree in which a tree is saved in a similar data structure in compressed form (Gupta, Hon, Shah and Vitter). Another solution is to compress the tree structure by decreasing the height so that space is reduced, but only by a constant amount (Gupta et al.). We can now understand the advantages of using a lexicon with a space-friendly approach when dealing with such smaller machines as a mobile device, although this is becoming less of an issue as these handheld devices' hardware specifications are rapidly improving.

To look at a practical situation where a lexicon and its space and operational time complexities are vital, we look at a mobile device and the constraints that system is under. For most mobile programs which impose on NLP to function correctly such as video games or Google's “okay, Google” feature, there are several data compression approaches we can take to reduce the size of dictionary data structures for mobile devices. As Bentevis, Kerkinos and Kalogeraki discuss in their research, a lexicon can be represented as either a cache-based dictionary or by employing a string compression technique in order to decrease the amount of

required space. A cache-based dictionary stores the most frequently used words of a language, but this approach also weakens the functionality of a program (Bentevis, Kerkinos and Kalogeraki). In their work, string compression is used in conjuction with the cache-based dictionary data structure to further lessen the high memory requirement by integrating an encoding method that works to decrease the memory amount necessary for each string (Bentevis et al.). A second incredibly important effort is the advancement of mobile devices at the hardware level, more specifically the betterment of a device's processing power from the Central Processing Unit (CPU) as well as increasing the amount of Random Access Memory (RAM), and also the amount of secondary memory (also known as internal storage). By using our current means of reducing space complexities (including the use of time and space efficient data structures) along with hardware improvements for machines, we will quickly see the convergence of NLP and its current success in contrast to its past, as well as the awareness and importance of NLP study and research.

To sum up, most applications in Natural Language Processing rely on the use of a lexicon for a broad range of operations. As a lexicon holds all words in a natural language, they are often huge in space, making certain operations, like search, on the data set difficult and time costly. On a positive note, we have many data structures available to us in serving as a lexicon and if chosen carefully, we can greatly reduce time and space complexities of required operations. Upon deciding on a suitable data structure to substitute a lexicon, we reduce the time complexity of various NLP functions, sometimes greatly. Even with an appropriate data structure, a developer may wish to use a fushion of structures and write his/her own

algorithm(s) to use with a single lexicon, or incorporate a data or tree compression method to reduce the amount of space necessary for the structure.

References

1. “A BST-based approach to dictionary structure for Chinese word segmentation.“ Ge, Chang; Ma, Ningjing; Chen, Xudong. Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on (Volume:1). Pages 355-357. 2011. **http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=5953238#citedby-section**.

2. “Compressed data structures: dictionaries and data-aware measures.” Data Compression Conference, 2006. DCC 2006. Pages 213-222. 2006. Gupta, A.; Hon, W.; Shah, R.; Vitter, J.S. **http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=1607256**.

3. “Dictionary Data Structures for Smartphone Devices.” PETRA '12 Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments. Article 46. 2012. Bentevis, Alexandros; Kerkinos, Ioannis; Kalogeraki, Vana. **http://delivery.acm.org/10.1145/2420000/2413155/a46-bentevis.pdf?ip=134.153.23.138&id=2413155&acc=ACTIVE%20SERVICE&key=FD0067F557510FFB%2EE6F865C24D7DB1C8%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=452658846&CFTOKEN=98697682&__acm__=1415337974_f72dff2b7c70117bb2e7c91d3a8a19c0**.

4. 2-3 Tree. Retrieved from http://en.wikipedia.org/wiki/2%E2%80%933_tree.

5. Array Data Structure. Retrieved from http://en.wikipedia.org/wiki/Array_data_structure.

6. Graph Abstract Data Type. Retrieved from http://en.wikipedia.org/wiki/Graph_(abstract_data_type).