COMP7505 Assignment 3 Report

Anthony Carrick

Introduction

"Pre-processing the text in the document to optimise searching is an expensive process." as Richard Thomas of UQ states. [1] This report investigates some possible approaches to minimising this cost by saving the result of the pre-processing to disk. The requirements are the ability to search for a word and list its line and column positions, find the total number of occurrences of that word, find lines where one or more words occur, as well as suffixes of that word (words that it's a part of).

Possible Approaches to Minimising the cost of text pre-processing

Save memory to disk (Serialisable)

Implementing Java's Serializable interface or using Python's Pickle function the state and objects can be saved to the file system and read back in directly. This works and is easy to implement, but is not human readable and therefore not portable between programming languages.

Save data structures to disk

Save Trie to Disk

For situations with low memory, but relatively fast flash based external memory, researchers at Bangladesh University of Engineering & Technology developed DiskTrie [2] an "efficient external-memory data structure for storing strings in mobile devices using flash memory". Their solution uses path and level compression, then writes the DiskTrie in a computer such that whole blocks are read on the mobile device.

Build trie out of dictionary and save to JSON

Building a trie in memory out of a dictionary has the advantage that it can easily be saved to JSON and is human readable as in [3] and [4]. Testing this idea, an implementation to hold all the required data as noted in the Introduction proved difficult to implement.

Save a compact trie to JSON

This approach is similar to the above, but should save disk space and therefore disk loading time by not storing so many individual objects. However, it requires the trie be generated in a compact manner which is far more complex than a normal trie.

Save Results to disk and Rebuild the Trie

Create the trie and other data structures as normal, then save out the finalised results to a file. When loading it, read it back and rebuild it. This is likely slower to rebuild than the previous two, but faster than parsing the document since it's already "clean".

Chosen Approach to Cost Minimisation of text pre-processing

Based on the above summaries, serialising the state to disk seems the easiest and more reliable approach.

Implementation

Java's Serializable interface has a few requirements to use and, "under the hood" operates in a few stages.

Requirements

1. Each class must implement Serializable interface. This is just a marker interface for Java to know to allow serialisation or not. [5]
2. Each member field that can't or shouldn't be serialised should be marked as *transient*. [6]
3. *Static* fields can't be serialised.
4. An optional serialVersionUID so that the class can change intentionally and the data still be deserialised. Otherwise Java will provide one anyway, which won't cope with simple addition of a member fields or some complex compiler/platform differences. [6]

Java Under the Hood

Java's serialization technology is rather complex. Instead I will describe only the parts relevant to storing existing Tries and linked lists as follows.

1. Java writes the class's corresponding ObjectStreamClass to the stream and assigns a handle.
2. Java writes Strings as the length followed by the contents in modified UTF-8.
3. Java writes arrays (such as in a trie) as a handle, the length, then each element.
4. For other 'standard' objects, Java:
    a. Writes metadata associated with the instance (such as handle)
    b. Writes out descriptions of superclasses starting with this object, up.
    c. Writes the classes fields starting with the highest object in inheritance order down.

(Based on [7] and [8])

Analysis and Comparison to loading each time.

Loading fresh analysis

```
For each line:
    For each word on line (separated by space or hypen)
        Regex processing to remove non-alphabet characters - O(c)
        Find column start position – O(s) – using Java's String methods
        Add word to Trie
            Create and traverse the trie nodes for each letter in the word
            Add the Occurrence object to the list at the end node.
```

Which means:

Per letter: traverse nodes (string-length before this letter) - 1

Per word: O(c + s) – which is O(c*2) where c is the length plus some little constant operations. Plus everything in Per Letter.

Per line: Run the per word stuff above plus split into array O(c) where c is number of characters plus Java's string split operations.

O(W * w * L + Regex + String methods + Create LinkedList nodes)

But all the constant operations get collapsed so it’s just O(W * w * L) W = word-length, w = number of words, L = number of lines.

Deserialising analysis

Read serialised file which is larger than the in-memory objects, but still smaller than the original document (and more compact).

Create objects in memory for all the objects it reads in. This is O(n + o) where n is the total size of unique words and o is the number of occurrence objects for each distinct word.

Conclusion

Based on the analysis, it’s cheaper to serialise and deserialise the data structures rather than loading each start up.

References

[1] R. Thomas, *COMP3506/7505,* Brisbane: University of Queensland, 2018.

[2] N. M. K. Chowdhury, M. M. Akbar and M. Kaykobad, “DiskTrie: An Efficient Data Structure Using Flash Memory for Mobile Devices,” in *Workshop on Algorithms and Computation 2007 - Proceedings of First WALCOM, 12 February 2007, Dhaka, Bangladesh*, Bangladesh, 2007.

[3] Chetanrns and P. Haugh, “Converting a Trie into JSON format,” 01 08 2018. [Online]. Available: https://stackoverflow.com/questions/51624207/converting-a-trie-into-json-format. [Accessed 12 10 2018].

[4] P. Haugh, “Python - matching end of string,” 12 07 2018. [Online]. Available: https://stackoverflow.com/questions/51295864/matching-end-of-string/51296069#51296069. [Accessed 12 08 2018].

[5] M. Narang and S. Juneja, “Serialization in Java,” [Online]. Available: https://www.geeksforgeeks.org/serialization-in-java/. [Accessed 14 10 2018].

[6] T. Greanier, “Discover the secrets of the Java Serialization API,” JavaWorld, 07 2000. [Online]. Available: https://www.oracle.com/technetwork/articles/java/javaserial-1536170.html. [Accessed 14 10 2018].

[7] Oracle Java Authors, “Java Object Serialization Specification ObjectOutputStream Class,” Oracle, 2010. [Online]. Available: https://docs.oracle.com/javase/7/docs/platform/serialization/spec/output.html. [Accessed 14 10 2018].

[8] P. Banerjee, “Java Serialization Part 2,” 25 04 2017. [Online]. Available: http://prinavtech.blogspot.com/2017/04/java-serialization-part-ii.html. [Accessed 14 10 2018].