School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg
ELEN4020 — Data Intensive Computing in Data Science

Course Project — Indexing XML Hansards

1 Introduction

Search functionality on websites and in programs can easily be taken for granted. The algorithms and processes to allow search queries to be answered in a *practical* amount of time are sophisticated and resource intensive. For data-intensive applications, naive non-cached search algorithms can take hours to find all relevant results in the chosen dataset. To reduce the search-time of queries, many database engines, programs, and websites use `indexing` to enhance the performance of search algorithms. An index is some pre-processed representation of the data that is either smaller than the full dataset, in a more easily computed format, or both.

Indexing includes numerous techniques such as bitmaps, hash-tables, and complex data-structures (typically tree based). Large search queries, with complex conditions, can be resolved quickly by combining separate bitmaps using bitwise operations. Hash-tables can be used to determine whether a search term exists in a given subset of the data. Complex data-structures are used to enhance these techniques either by accelerating a specific step of the algorithm or by utilising their representations as a function of their structure. For example, a binary tree can be used to perform searches on a single variable in $\log(n)$ time.

2 Search Indexing of XML Hansards

Search indexing is a step that enhances the future performance of a program. A search program would then utilise the resulting indexes to find results in the data. Typical programs that benefit from indexing are those where the type of searches can vary significantly and can be combined with other queries such that caching results for all possible queries is not feasible. A perfect example of this is text search with cross-references.

For this project, you are tasked with indexing the transcribed records of debates in the Parliament of the Republic of South Africa (Hansards). This dataset perfectly illustrates the necessity of indexing. Every debate in a governmental institution is recorded for historical and political purposes. With the ever increasing prevalence of internet-based access to records, hard-copy records such as these are being transcribed to digital formats to be delivered online.

The Hansard dataset is given in XML (Extensible Markup Language). XML files come with a predefined structure, called a schema, that allows programs to optimise for specific structures in the data. An example of the XML Hansard data is given in Listing 1. It conforms to the Akomo Ntoso XML schema (`http://www.akomantoso.org`), as shown in line 1 in Listing 1. The full dataset covers most of the records for the 20th century. Given their high information-density, searching the full dataset without indexing would be too resource intensive to be a practical feature of any future online platform holding the data.

Listing 1: Extract from 1979 Hansard

```
<akomaNtoso xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns
    ="http://docs.oasis-open.org/legaldocml/ns/akn/3.0"
    xsi:schemaLocation="http://docs.oasis-open.org/legaldocml/ns/akn
    /3.0 akomantoso30.xsd">
<debate name="hansard">
<meta>
...
<speech by="#Gouws">
<from>
Senator
<person refersTo="hansard">W. J. GOUWS</person>
:
</from>
<p>
Senator W. J. GOUWS: May I put a question to the hon. Senator? Did the
    hon. Senator not ask me in my office which Orders of the Day would
    be taken on that day? When I discussed this with him was he not
    satisfied?
</p>
</speech>
...
```

Your task is to design, implement, and analyse a parallel program that calculates indexes for the given Hansard data. The scaling of the program must be evaluated to determine whether or not the implemented system is appropriate for indexing such types of data. The resulting indexes must be appropriate for common searches that would be made on the data. You must choose one of the following three types of searches that could be made on the data and choose an appropriate indexing scheme:

1. Finding specific speaker(s) and the associated debate(s) in the data. This includes finding the debates in which two speakers participated, or all of the speakers that participated in two specific debates.
2. Given a list of words, finding all occurrences these words and the debates in which they were spoken. This also includes the words in the list which were spoken in two specific debates.
3. Given a list of words, finding all occurrences these words and which speaker(s) used them in parliament. This also includes the words in the list which two specific people spoke in parliament.

You will be provided with a list of words where necessary. The resulting indexes must then be used to perform a search on the data to illustrate how such a program would use the new indexing structure. It is not necessary to implement a program to perform the search, the primary focus is on the generation of the indexes. However, an automated searching implementation is required to achieve an `Excellent` rating for your implementation.

3 Assessment

The assessment of this project uses the standard project rubric available on the course homepage (Sakai). This covers both the source code that implements the designed solution as well as the self-standing group report. As previously mentioned in this document, you may create a program that uses your indexes to run search queries on the data to achieve an `excellent` rating for the implementation/engineering execution component of the rubric.

3.1 Restrictions

Your implementation must conform to the "Software Guide" for the course. The entire project is to be done within your course laboratory group. All source code must be stored and hosted in your assigned group repository on the course Github organisation. You may use third-party libraries in your program; however, the core of your indexing algorithm must be bespoke and implemented by all group members. The report should be self-standing such that the source code is not necessary to understand the algorithm and implementation. The report must conform to the School's "Blue Book".

You will be given `ssh` access to a compute cluster to benchmark your program and access the full Hansard dataset. The same cluster will be used to test your implementation using the source code submitted on Github.

3.2 Deliverables

- A `git` repository with the indexing program source code.
- A single 4-6 page group report covering the design, implementation, and analysis of the chosen solution.
- An analysis of the scaling performance of the chosen solution (in the group report).