Project: Search Engine

Group Z
Alice Liddell Bob Ross Cruella de Vil David Attenborough
{alid,bros,cdev,datt}@itu.dk

November 18, 2019

Introduction

This document reports on the search engine project that we developed during the Introductory Programming course at the IT University of Copenhagen. In Sections 1–3 we will report on our solutions to the mandatory tasks posed in the project description. We also solved some of the challenges posed for these mandatory tasks. Their solution is described in the respective section along with the solution to the mandatory task. Furthermore, we developed the following extensions which are described in detail in the following sections:

This is just an example template; you do not need to do these extensions.

- Section 4 describes our changes to the client GUI.
- Section 5 describes how we built a webcrawler.
- Section 6 describes how we supported the "fuzzy search" feature.

The description for each solution is roughly split up into the following parts:

- **Task:** A short review on the task that we had to solve.
- **Approach:** An informal, high-level description of how we solved the task.
- **Solution:** A detailed technical description of our solution to the task.
- **Test:** Description of considerations for testing the correctness of our solution.

Other parts (e.g. design, reflection, ...) will be included where deemed appropriate. For instance, Task 1 includes a part on the benchmarking results.

The source code of our project is handed in as a single zip file called `<filename>.zip`. `<insert directory>` contains the source code that solves the mandatory tasks. Our code is also available on ITU's Github: `https://github.itu.dk/...`.

Statement of Contribution

All authors contributed equally to all parts of the solution of the mandatory tasks. Bob Ross made the graphic design for the client GUI. Bob Ross and Cruella de Vil conceived the idea of the web crawler, Cruella de Vil implemented it, and Bob Ross documented it. Alice Liddell is solely responsible for the "fuzzy search" feature.

1 Inverted Indices

Task This task involves reducing the amount of time the search engine takes to respond to a query, by means of an inverted index.

An inverted index provides a fast way to access the webpages that contain a certain word by storing a mapping from words to the list of webpages that contain the word. Figure 1 provides an example for such a data structure.

We are to implement two variants of an inverted index: one based on a `TreeMap`, and the other on a `HashMap`. We are then to evaluate the performance of these implementations, and to pick the best one to use in our search engine.

Approach As per the code handout, we generalized the notion of an index into what we call an `Index`, that has the following two methods:

- `build`: add description from your javadoc.
- `lookup`: add description from your javadoc.

We then implemented the inverted index by using Java's Map data structure. Here, Java provides different implementations and we tested the following variants in our code: ..., ...

Solution We build the index data structures as described in Figure 1. We allowed to switch the implementation of the Map data structure by ... . Building the inverted index was accomplished by the following code:

```
1 for each webpage page in list of webpages {
2     for each word p on page {
3         ...
4     }
5 }
```

Figure 1: UML Diagram for the Software Architecture of our webserver.

Index	File	Avg. lookup time (ms)
SimpleIndex	enwiki-tiny.txt	...
SimpleIndex	enwiki-small.txt	...
SimpleIndex	enwiki-medium.txt	...
InvertedIndex w/ HashMap	enwiki-tiny.txt	...
...	...	...
InvertedIndex w/ TreeMap	enwiki-tiny.txt	...
...	...	...

Table 1: Running times for different index implementations.

Test We verified the correctness of the `build` and `lookup` method with unit tests. These tests can be found in the file .... Here, the `build` method of the inverted index was tested by .... (May add Java code here.)

The `lookup` method of all three indexes was checked by carefully creating a list of test webpages and checking whether the lookup methods returns the expected results by looking at the following properties of the returned list: (i) its size and (ii) ... ? (Add Java code?)

Benchmark We evaluated which implementation provides the fastest running time for lookup operation by ... (describe your benchmarking setup / approach).

We choose the following query words for the benchmark: “itu, is, the, best”. The running times of different index implementations can be found in Table 1 (or a plot).

These benchmark results indicate that ... . Based on these results, we decided that we use ... as the index in our data structure.

2 Refined Queries

3 Ranking Algorithms

4 Extension: Improving the Client GUI

5 Extension: Webcrawler

6 Extension: Fuzzy-Search