Project: Search Engine

Group: ip18groupR

IT UNIVERSITY OF COPENHAGEN

Group name: Busca.*
Introductory Programming
Master of Science in Software Development
IT University of Copenhagen
December 14, 2018

b

Contents

Chapter 1

Introduction

1.1 Introduction

The goal of htis project is to implement a large piece of software and develop web-based search engine. Several software development tools and tehniques have been used: version control(Git), testing (JUnit), debugging, documentation (JAvadoc), benchmarking, build tools (Gradle), and code review. The fallowing chapters describe the project in detail. Project is broken down into three main parts, Task 1: Fester Queries using an Inverted Index; Task 2: Refined Queries; Task 3: Ranking Algorithms.
Result
This projecct result in...

Chapter 2

Chapter 2: Faster Queries using an Inverted Index

2.1 Introduction

In this section we are evaluating three different approaches... lists, hash map and tree map
Inverted hash map and tree map...
Ran tests to compare the results, which

2.2 Set Up

As part of the set up of this task, the FileHelper class – specifically the parseFile(String filename) method – was updated such that from the database file, only websites that have a url, title, and at least one word of webpage content are read-in and stored in the server. This was accomplished by an IF statements to check the assignments of the url and title fields prior to adding a new Website object to the ArrayList<Website>. However, the meat of the changes made to this method were to how the method recognised the content of each line scanned in in order to know how to treat it. Previously, this was accomplished by making use of the knowledge of the very specific file format, String methods, and boolean field variables. This was all replaced by two regular expressions:

```
Pattern website = Pattern.compile("(https?://[A-Za-z0-9./_]+)");
Pattern webTitle = Pattern.compile("[A-Z][a-z]+[A-Za-z0-9\s]+?");
```

and the methods of the Matcher class. Though it doesn't look to be that big of a change, doing so means that the two field variables are no longer needed, which means less has to be juggled when reading and making further changes to the code.

2.3 Indexes and Data Structures

2.3.1 SimpleIndex

bla bla bla

2.3.2 InvertedHashMap

bla bla bla [Ora18]

2.3.3 InvertedTreeMap

bla bla bla

2.4 Analysis

2.4.1 Search Using Lists

2.4.2 Search Usnig InvertedHashMap

2.4.3 Search Usnig InvertedTreeMap

2.5 Testing

After the above changes were implemented, development tests were written in order determine the viability of the code and whether the changes satisfied the requirements of the task. To that end, JUnit tests were devised for each class that was updated.

2.5.1 JUnit tests

2.5.1.1 FileHelper Class

White-box tests were developed around the branching statements in the updated method, and a coverage table was produced.

From the coverage table an expectancy table was produced.

where data/test-file1.txt is an empty file, and the rest contained the following data:

As you can see from the Actual Output column of **??**, the updated code failed test B3, highlighting a weakness in the code, and subsequently had to be debugged. Including another IF statement after the while loop resolved the issue, and following that all tests were passed.

Table 2.1: Coverage table of the parseFile(String filename) method

Choice	**Input property**	**Input data set**
1 catch	incorrect file name	A
1 try	file name	B
2 while: zero times	empty file	B1
2 while: once	file has one line	B2
2 while: more than once	file has two lines	B3
2 while: more than once	file has at least three lines	B4
3 true	the line contains a web url	B3, B4
3 false	the line does not contain a web url	B1, B2
4 true	either the listOfWords field or the title field is null	B3, B4
4 false	both the listOfWords and the title fields are not null	B4
5 true	the url field is not null	B4
5 false	the url field is null	B3, B4
6 true	the line contains a website title	B3, B4
6 false	the line doesn't contain a website title	B2
7 true	listOfWords is null	B2, B4
7 false	listOfWords is not null	B4

2.5.1.2 InvertedIndexHashMap Class

2.5.1.3 InvertedIndexTreeMap Class

2.5.2 Comparison Benchmarking Results

We compared the results of ... We made sure that that the environmetn when runnin the different test are as much as possible similar, e.g. no other programms running on the machine during the testing, that could affect the test performance results.
In table 2.3 the result of benchmark can be seen.
JMH/ avg/ ns/op link to it

Table 2.2: Expectancy table of the JUnit tests

Input data set	Input data	Expected output	Actual output
A	"wrongfilename.txt"	Exception	FileNotFoundException
B1	"data/test-file1.txt"	returns an ArrayList<website>, size() == 0	returns an ArrayList<website>, size() == 0
B2	"data/test-file2.txt"	returns an ArrayList<website>, size() == 0	returns an ArrayList<website>, size() == 0
B3	"data/test-file3.txt"	returns an ArrayList<website>, size() == 0	returns an ArrayList<website>, size() == 1
B4	"data/test-file-errors.txt"	returns an ArrayList<website>, size() == 2	returns an ArrayList<website>, size() == 2
B4	"data/test-file4.txt"	returns an ArrayList<website>, size() == 2	returns an ArrayList<website>, size() == 2

data/test-file2.txt	data/test-file3.txt	data/test-file4.txt	data/test-file-erro
word3	http://example.com Title1	*PAGE:http://page1.com Title1 word1 word2 *PAGE:http://page2.com Title2 word1 word3	word1 word2 *PAGE:http://page1 Title1 word1 word2 *PAGE:http://wrong Title1 *PAGE:http://wrong *PAGE:http://wrong Titleword1 Titleword *PAGE:http://page2 Title2 word1 word3

Table 2.3: Benchmark results in nanoseconds for three type of indexes and test files

Test Files	**SimpleIndext**	**Inv.IndexHashMap**	**Inv.IndexTreemap**
	avgt Score ns/op	**avgt Score ns/op**	**avgt Score ns/op**
EnWiki Tiny	18944.884	1052.067	1591.311
EnWiki Small	8819338.592	1883.776	3622.582
EnWiki Medium	233498546.571	27451.020	30176.993

Chapter 3

Chapter 3: Refines Queries

3.1 Section 3.1

Text

3.2 Section 3.2

Text

3.2.1 Subsection 3.2.1

Text

Chapter 4

Chapter 4: Ranking Algorithms

4.1 Section 4.1

Text

4.2 Section 4.2

Text

4.2.1 Subsection 4.2.1

Text

CHAPTER 5

Chapter 5 Extensions

5.1 Section 5.1

Text

5.2 Section 5.2

Text

5.2.1 Subsection 5.2.1

Text

Chapter 6

Chapter 6 Conclusion

6.1 Section 6

Text

Appendix A

Test Figure reference

This is a test of the appendix and how to reference to something in it. Below is shown an image which is used for test[1]testimage.

[1]this is just for testing...`www.test.dk`

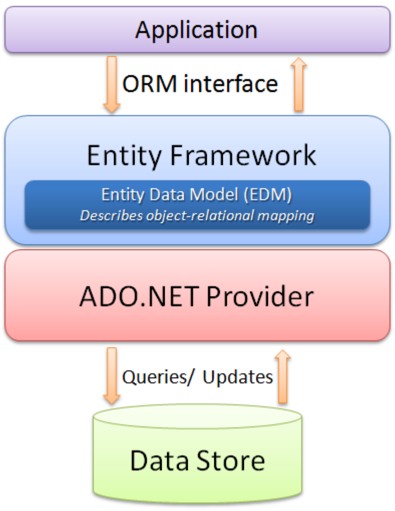

Figure A.1: Microsoft Entity Framework

APPENDIX B

Test tabel reference

This appendix is a test of creating and referencing a table in latex. In table **??** a example from Peter can be seen. can be seen.

Table B.1: Oversigt over testdeltagerne

Deltager	**Navn**	**Stilling**	**Rolle**
1	Ole Nørrekær Mortensen	Projektleder	Kundeadministrator
2	Allan Booker	Driftsleder	Inspektør
3	Ronni Bing Simonsen	Ingeniør	Kunde

Table B.2: Test af tabel

Colunm1	**Colunm2**
Celle 1	Celle 2
Celle 3	Celle 4

Tabellen har nummer B.2.

En lidt mere avanceret tabel:

I tabel B.3 kan du se hvordan teksten er justeret: l=left, c=centreret og r=right.

Table B.3: Test af tabel2

Celle 1.........	Celle 2....................	Celle 3.........
Celle 4	Celle 5	Celle 6

Bibliography

[Ora18] Oracle. Class treemap. https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html, 2018.