Bjørn Are Odden
Ole Alexander Høyby

TDT4117 Information Retrieval - Autumn 2021

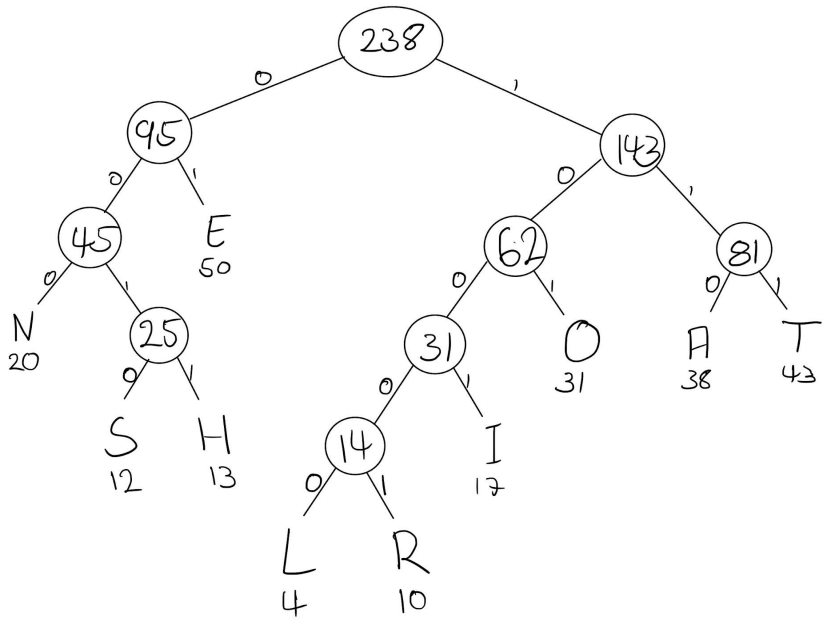
Task 1 - Text Compression

1. Given the following list of average common characters in Norwegian language with frequencies, calculate the Huffman codes by using the canonical tree.
   Show even the canonical Huffman tree.

   E=50 T=43 A=38 H=13 S=12 O=31 I=17 N=20 R=10 L=4
   What is the average length of the code? What would be the average length of the code if the frequency of the letters was equal?
   Answer:
   Letters sorted by frequency:
   E=50 T=43 A=38 O=31 N=20 I=17 H=13 S=12 R=10 L=4

   Standard Huffman Table and tree

LETTER	CODE
N	000
S	0010
H	0011
E	01
L	10000
R	10001
I	1001
O	101
A	110
T	111



Bjørn Are Odden
Ole Alexander Høyby

Step 1: Sort primarily by nr of bits and secondary by letters

Step 2: Top element gets to be 0

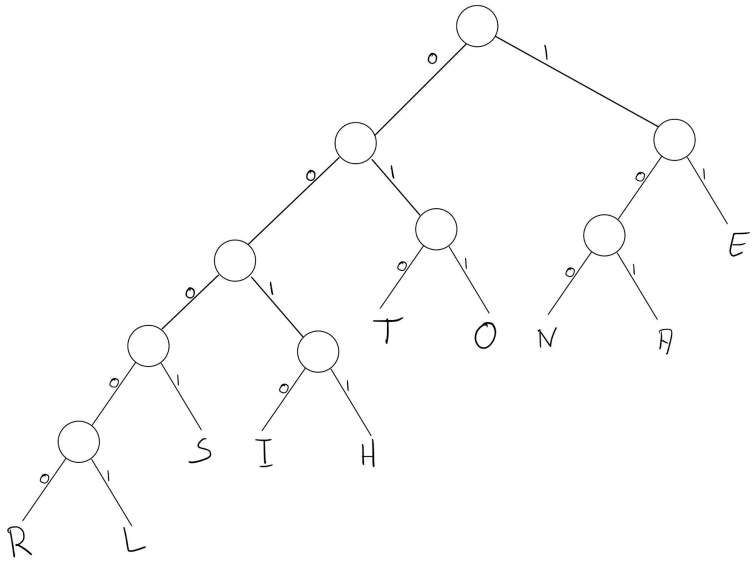
Note: the length of the final code should be the same as the original (eg. E:01 -> E:00)

Step 3: Next element: count up by one. Left shift (add 0) if the no of bits is increased.

Step 4: repeat step 3 until all elements have been accounted for. Then reverse all bits to make the tree reach out to the left.

Canonical Huffman Table and tree

LETTER	CODE
E	11
A	101
N	100
O	011
T	010
H	0011
I	0010
S	0001
L	00001
R	00000



Bjørn Are Odden
Ole Alexander Høyby

2. Given the following Huffman codes Letter Code

Letter	Code
A	10
T	11
H	000
U	010
N	011
L	0010
O	0011

Decode the strings 00010001000100011 and 00111011010

Answer:

000	|10	|0010	|0010	|0011		0011	|10	|11|	010
H	E	L	L	O		O	A	T	U

Bjørn Are Odden
Ole Alexander Høyby

Task 2 - Index Analysis Using Lucene

For the rest of this assignment, you will be dealing with text indexing using the Apache Lucene framework. Lucene's general goal is to provide open-source search software. The project homepage can be found at: http://lucene.apache.org/.

1. Give a short explanation of Lucene.

   Answer:

   Lucene is not a program (only the "demo" classes we use for this assignment works "out of the box"), but rather a collection of classes and methods that can be added on top of existing applications in order to provide searching capabilities, instead of implementing searching, indexing etc. yourself.

2. Index the files found in the archive DataAssignment4.tar.gz available on Blackboard. This is done by running the IndexFiles class. This class creates a Lucene index in the index directory from the directory passed as argument.
   Copy the console output after doing the indexing and include timing for the query run.

   Explain the steps involved in the indexing process according to the source code. Consider principally the main method and do not explain the first part regarding the input args reading.

Bjørn Are Odden
Ole Alexander Høyby

Answer:

```
bjoer@BAONEW MINGW64 /e/Test/assignment4lucene/smol
$ time java -cp "./*" org.apache.lucene.demo.IndexFiles -docs DataAssignment4
Indexing to directory 'index'...
adding DataAssignment4\Text1.txt
adding DataAssignment4\Text2.txt
adding DataAssignment4\Text3.txt
adding DataAssignment4\Text4.txt
adding DataAssignment4\Text5.txt
adding DataAssignment4\Text6.txt
200 total milliseconds

real    0m0.311s
user    0m0.000s
sys     0m0.000s
```

After parsing the args as well as other bookkeeping, the indexing starts properly at line 92. Using the analyzer, indexwriter and the config instantiated at 93, 94 and 98, the filetree from the “-dir”-input is traversed and each file is added to the index via the Document-class, which is then populated with three fields: “path” and “modified”, which are metadata as well as “contents”, where the contents of the individual file is entered together with a reader instance that will tokenize the content before the file is then indexed by passing it to the writer-instance at line 198/204, which then takes off into the rest of the lucene source code. After this looping/traversing is finished, the indexwriter has created the index under /index.

3. After indexing the document files, a search can be performed. This can be done by running the SearchFiles class. Do the following search:
    - God
    - Circumstances
    - claims of duty

    Copy the result from the console for each query and include timing for the query run. What query model is used here? For example, do returned documents contain all query terms together (AND), or are all documents containing any of them returned (OR)?

Bjørn Are Odden
Ole Alexander Høyby

Answer:

```
bjoer@BAONEW MINGW64 /e/Test/assignment4lucene/smol
$ time java -cp "./*" org.apache.lucene.demo.SearchFiles -query "God" -repeat 1
Searching for: god
Time: 7ms
2 total matching documents
1. DataAssignment4\Text1.txt
2. DataAssignment4\Text2.txt

real    0m0.234s
user    0m0.000s
sys     0m0.015s
```

```
bjoer@BAONEW MINGW64 /e/Test/assignment4lucene/smol
$ time java -cp "./*" org.apache.lucene.demo.SearchFiles -query "Circumstances" -repeat 1
Searching for: circumstances
Time: 7ms
1 total matching documents
1. DataAssignment4\Text6.txt

real    0m0.217s
user    0m0.000s
sys     0m0.000s
```

```
bjoer@BAONEW MINGW64 /e/Test/assignment4lucene/smol
$ time java -cp "./*" org.apache.lucene.demo.SearchFiles -query "claims of duty" -repeat 1
Searching for: claims of duty
Time: 13ms
6 total matching documents
1. DataAssignment4\Text6.txt
2. DataAssignment4\Text1.txt
3. DataAssignment4\Text4.txt
4. DataAssignment4\Text5.txt
5. DataAssignment4\Text2.txt
6. DataAssignment4\Text3.txt

real    0m0.219s
user    0m0.015s
sys     0m0.000s
```

We can see by the last query that the documents returned contain any term in the given query (OR), as a manual check, only text6 has the phrase “claims of duty”, but also the only with all three terms. The rest of the documents got the term “of”, but not any of the other, which verifies our claim that unless specified, documents returned can contain any of the terms. Worth mentioning that text6 is returned at 1st place, maybe indicating that there is some sort of ranking as this is the one document containing all the terms, as well as the exact phrase.

4. The previous example only contains 6 small files. To demonstrate the advantage of Lucene's index, we ask you to use the Enron Email Dataset, which contains more than 500,000 e-mails. First, download and extract the following archive: https://www.cs.cmu.edu/~enron/enron_mail_20150507.tar.gz. Using the same procedure as in the previous example, index files in the folder and search for any query you find interesting. For instance, you can check if all the following terms "Norwegian University Science Technology" wear mentioned in any of the e-mails. In your report, provide the query/queries you chose and the time needed to perform the search.

   Tip: to measure program execution time, use "-repeat 1" as one of the program arguments.

   Voluntary exercise: implement a program that iterates over the dataset files and searches for a given string in them (easy implementation takes less than 10 lines in Python). Compare your program's and Lucene's speed of search.

   Answer:

```
adding maildir\zufferli-j\sent_items\98_
adding maildir\zufferli-j\sent_items\99_
adding maildir\zufferli-j\sent_items\9_
314902 total milliseconds

real    5m15.130s
user    0m0.000s
sys     0m0.031s

bjoer@DESKTOP-87J5DU8 MINGW64 ~/IdeaProjects/ass4
```

   If we want to check whether all the terms in "Norwegian University Science Technology" are present in a document, we have to modify the query, as we already know that documents without all terms will be returned in lieu of documents with all terms, and there's no distinction in the returned list whether a document has one, several or all terms. Reading up on Lucenes query parser, we've found that chaining the terms with "AND" or "&&" will give the expected result; documents containing all the terms.

Bjørn Are Odden
Ole Alexander Høyby

```
bjoer@BAONEW MINGW64 /e/Test/assignment4lucene/chonk
$ time java -cp "./*" org.apache.lucene.demo.SearchFiles -query "Norwegian AND University AND
 Science AND Technology" -repeat 1
Searching for: +norwegian +university +science +technology
Time: 33ms
7 total matching documents
1. maildir\lay-k\inbox\1126_
2. maildir\dasovich-j\notes_inbox\1641_
3. maildir\dasovich-j\all_documents\3408_
4. maildir\dasovich-j\all_documents\2437_
5. maildir\dasovich-j\notes_inbox\998_
6. maildir\dasovich-j\deleted_items\376_
7. maildir\martin-t\deleted_items\24_

real    0m0.597s
user    0m0.000s
sys     0m0.031s
```

Observe the difference when not using the boolean operators:

```
bjoer@BAONEW MINGW64 /e/Test/assignment4lucene/chonk
$ time java -cp "./*" org.apache.lucene.demo.SearchFiles -query "Norwegian University Science
 Technology" -repeat 1
Searching for: norwegian university science technology
Time: 34ms
3977 total matching documents
1. maildir\lay-k\inbox\1126_
2. maildir\lay-k\inbox\958_
3. maildir\kaminski-v\inbox\136_
4. maildir\kaminski-v\deleted_items\781_
5. maildir\skilling-j\inbox\156_
6. maildir\kaminski-v\inbox\140_
7. maildir\kaminski-v\deleted_items\776_
8. maildir\skilling-j\all_documents\1454_
9. maildir\skilling-j\discussion_threads\1144_
10. maildir\skilling-j\notes_inbox\383_

real    0m0.310s
user    0m0.000s
sys     0m0.000s
```

As far as we have understood, the use of boolean operators does not guarantee that the phrase as a whole appears, but that each of the terms do. The query parser supports phrasing, but we could not get this to work when running the program in the terminal.