Programming Workshop (SPWS2015)
Project Description

Tijs Slaats[*]
tslaats@itu.dk

August 12, 2015

1 Introduction

The goal of this project is to implement a simple, text-based search-engine. The project is divided in two parts:

1. A mandatory part where you create the core of the search engine.
2. A number of optional extensions to the engine. Students who have completed their mandatory tasks can pick any that they find interesting.

The project will be carried out in groups of three or four persons. Smaller groups will only be allowed under special circumstances and with prior permission from the course manager. Note that the requirements for smaller groups are the same as for regular sized groups, i.e if for some reason you end out working alone you are still expected to hand-in all mandatory tasks. Each group is required to hand in a written report at the end of the project, which will form the basis for an oral exam.

The course will start in the fifth week of the semester. There will be weekly lectures which will be used to introduce the assignments, teach some of the required programming techniques and answering questions regarding the assignments. A detailed lecture plan can be found on learnit [1]. The following sections describe the project assignments in detail, unfamiliar terms should not discourage you as they will be explained during the lectures.

[*]Based on project description by unknown original author.
[1]

2 Background

2.1 Data Files

The search engine takes a file as argument, which lists a number of web pages and the words that appear on them. Each entry starts with a *PAGE line which denotes at which address (or URL) the page can be found. This is followed by a list of words that occur on the page. Each word has its own line and the same word can occur multiple times. The following example illustrates the file format:

```
*PAGE: http://www.itu.dk/index.html
Here
is
a
word
and
one
word
more
*PAGE: http://www.itu.dk/anotherpage.html
Here
is
more
and
yet
more
```

Three data files are available for use as input data for the search engine:

1. The file itcwww-small.txt (approx. 30 KB).
2. The file itcwww-medium.txt (approx. 1.5 MB).
3. The file itcwww-big.txt (approx. 20 MB).

2.2 The program SearchCmd

You will start out with the source code for a simple search engine called SearchCmd. The program takes the name of the data file used for searching as a parameter and constructs a linked list[2] from it. The objects in the linked list contain the fields *str* and *next*. The field *str* contains a line from the data file and *next* points to the next object in the list. Figure 1 shows such a linked list build from the first three lines of the example file. After reading the data file, a simple user interface launches, enabling the user to query the search engine: Given a word, the engine informs the user whether or not the word is present in the input. At the first lecture we will go through the source code and explain it in detail.

[2]`https://en.wikipedia.org/wiki/Linked_list`

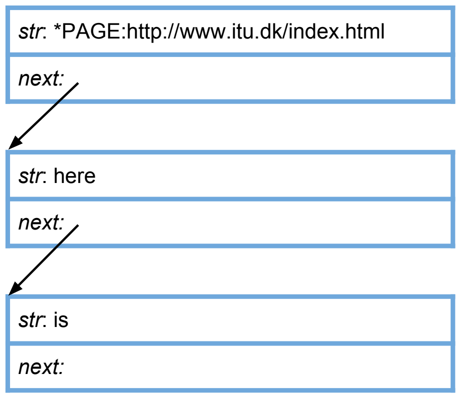

Figure 1: Example of a Linked List

3 The Mandatory Assignment

The mandatory assignment consists of solving the following five tasks, which will be explained in more detail throughout the semester. You should solve the tasks consecutively, i.e. in the second task you will expand on the work you did for the first task, etc.

For the first four tasks it is not allowed to use any packages other than java.io and java.lang. As a consequence it is not allowed to use any of the classes from the java.util package: the purpose of this project is to learn how to program various data structures yourself, as opposed to just using data structures programmed by others. As you can see, you are allowed to use different methods on String objects. It is also allowed to use the method hashCode and to allocate arrays. If you have any doubt whether some Java functionality is allowed for use or not, contact one of the teachers.

1. **Compilation**
   Compile and run the program SearchCmd

2. **Finding Pages**
   Modify SearchCmd so that when the user requests a search for a particular word, the URLs of all pages where that word occurs are shown on the screen. For this task you only need to change the search algorithm in SearchCmd. You do not need to change the data structure.

3. **Changing the Data Structure**
   Modify the construction of the data structure to create a linked list that consist of objects containing three fields: *word*, *urls*, and *next*. The field *word* will store a word, the field *urls* will contain a pointer to a list of **all** pages that contain the word and *next* will point to the next entry in the list. Each list of URLs should be a linked list of its own, containing the fields *url* containing a single URL and *next* pointing to the next entry in the list. Figure 2 shows an example

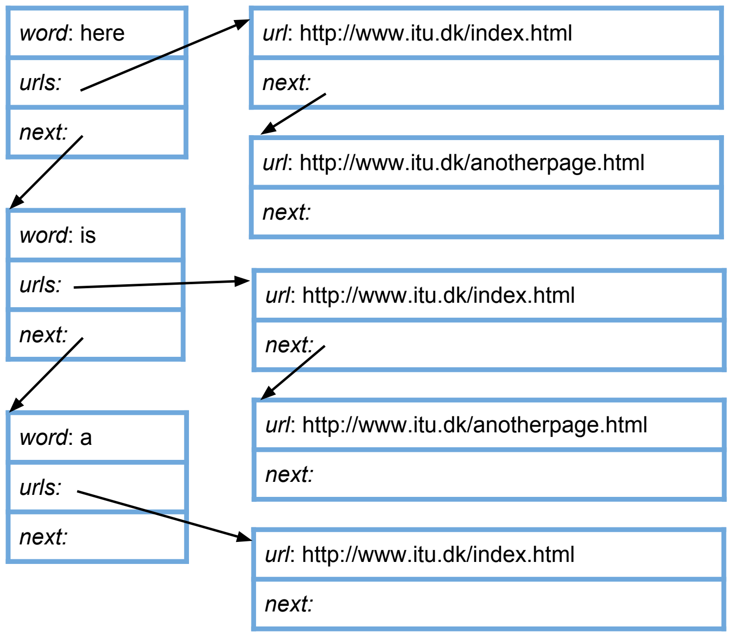
word: here
urls:
next:
word: is
urls:
next:
word: a
urls:
next:
url: http://www.itu.dk/index.html
next:
url: http://www.itu.dk/anotherpage.html
next:
url: http://www.itu.dk/index.html
next:
url: http://www.itu.dk/anotherpage.html
next:
url: http://www.itu.dk/index.html
next:

Figure 2: Example of the New Data Structure

of the new data structure, after its construction you will have to update the corresponding search procedure as well. You are also expected to investigate how the new data structure compares to the old one in terms of 1) size and 2) search time, and explain any differences.

4. Hashtables
Modify the program to use a hashtable[3] instead of a linked list of words. You can create the data structure using chained hashing[4]. The use of a hashtable will have a drastic effect on both the time it takes to initialize the search engine and the time it takes to perform individual searches.

5. Boolean Search
For the final task there are no restrictions on the Java functionality you are allowed to use. However we recommend that you do not spend too much time looking for advanced and unnecessarily complicated features in the vast collection of java libraries. The task is to add support for the boolean operators AND and OR to the search engine, so that it accepts three kinds of input strings:

1. Ordinary words, as before. Example: Hans.

[3]`https://en.wikipedia.org/wiki/Hash_table`
[4]`https://en.wikipedia.org/wiki/Hash_table#Separate_chaining_with_linked_lists`

2. A word AND another word. Example: Hans AND Grethe. This should return a list of those pages where both the words Hans and Grethe occur.
3. A word OR another word. Example: Hans OR Grethe. This should return a list of those pages where either the word Hans or Grethe (or both) occurs.

You are not required to support more complex expressions containing several occurrences of AND and OR. (If you are interested in solving that problem, see one of the suggested extensions below.)

4 Extensions

When you are done with the mandatory assignment you are encouraged to work on further improvements of the search engine. You could try to improve the basic design and make searches faster and/or decrease the memory usage, for example by using the java.util.HashSet class or other classes from the Java Collections Framework. In addition you can work on extending the basic search engine, a number of suggestions follow below (in no particular order):

1. Write a web crawler.
2. Investigate how to easily support prefix-search. For example a search for arm* should find all pages with arms, armoury, armadillo, armageddon, armistice etc. This only requires an extra table of pointers to the words, where the table is ordered by words.
3. Create a graphical user interface.
4. Construct your own hashCode method.
5. Find other data other than homepages to search through and modify your search engine to fit your situation. For example, you could be searching for grep patterns in a text file.
6. Design a tool that, given a web page x, will search for web pages similar to x.
7. Enable the user to limit the search to URLs matching a pattern such as *.dk or itu.dk/dkm/*.
8. Combine two user-imposed constraints on searching, for example limited-domain search and search for a particular prefix.
9. Support complex boolean searches, for instance ((((a AND b) OR c) AND f OR b) OR g) AND NOT h.
10. If a search produces no result, it might be because the user performed an erroneous keystroke, so that, e.g., an a became an s. In such cases you can choose to return pages which contain words almost matching the word searched for.

11. Support free text searches. That is, give the opportunity to search for (fragments of) sentences. A strategy is to implement a free-text search as an AND boolean search. This limits the search to a small amount of pages which are then examined in closer detail. Another approach is to calculate hash values for sentences instead of words. A third possibility is a more compact representation of the text. Instead of having the word repeated a certain number of times, you remember at which positions in the homepage the word occurs. Find your own favorite strategy.

12. How can the data structure be updated dynamically? You can choose to collect information on a lot of new homepages before updating. Another approach is to implement dynamic data structures such as dynamic search trees. The method chosen will be a compromise between space, search time and the time for updating the data structure.

13. Implement the main part using vectors, collections etc. and compare this (these) solution(s) to the solution from the main part where these classes were not used.

5 The report

Your group should hand-in a report covering at least the mandatory assignments. You can find out how to hand in written work in the study guide [5]. The report should start with the standard front cover[6], followed by a table of contents and a foreword. The foreword should shortly (in 1 page or less) describe what is covered in the report, i.e. what tasks and extensions did you manage to solve and implement. For each of the mandatory tasks you should:

1. Describe how the different search engines work, which data structures are used and how they are used. This description must be in a natural language and Java code may only be used to support the description.

2. Investigate and document how effective the different search engines are by running them. Be careful to run each engine several times on the same input.

3. Compare the different implementations to each other by using your benchmarking results from the preceding clause and/or by applying algorithm analysis. It is a good idea to use O-notation in this comparison, if you are familiar with it.

4. Describe how you tested your code for correctness.

The following structure of the report is advised:

[5] `http://studyguide.itu.dk/SDT/Your-Programme/courses-and-projects/submitting-written-work`

[6] `http://studyguide.itu.dk/~/media/Studyguide/StudyGuide2014/submissionofwrittenworkV2pdf.pdf?la=en`

1. Cover page
2. Foreword
3. Walk-through for Mandatory Task 2
4. Walk-through for Mandatory Task 3
5. Walk-through for Mandatory Task 4
6. Walk-through for Mandatory Task 5
7. Benchmarking results for the different search engines together with an explanation of the differences.
8. Extra assignments

For each task the description of your search engine can be presented as follows:

1. Describe the Input/Output-relation (What does the program do?) without reference to code in such a way that it could be read by anyone.
2. Describe the data structure without reference to the specific implementation.
3. Describe ideas, invariants etc. for the code.
4. Describe in general terms and without references to the code how it is structured.
5. Describe details of the code if needed.
6. Describe how much time and space the code uses.
7. Compare to previously described search engines.

In addition to the project report you should hand in the source code for all relevant versions of your search engine as a **single** zip file. It should be clear what code belongs to which task (for example by using a separate folder for each version) and clear instructions on how to run the code should be included.