Assessment

Leveres innen 30 apr innen 23:59 **Poeng** Ingen **Tilgjengelig** etter 22 mar i 12:00

Overview

In this assignment you will implement an index ADT that supports indexing of text documents and evaluation of queries for the occurrence of specific words in a collection of documents. A sample application uses this ADT to build a basic search engine, with a simple web interface that evaluates queries on a set of local files, and returns links to the matching files.

Query language

The query language of the index ADT must support queries for specific words, combined with the boolean operators AND, OR, and ANDNOT.

The BNF grammar for the query language is as follows:

```
query   ::= andterm

        | andterm "ANDNOT" query

andterm ::= orterm

        | orterm "AND" andterm

orterm  ::= term

        | term "OR" orterm

term    ::= "(" query ")"

        | <word>
```

You will need to implement a parser for this grammar. We recommend implementing a recursive descent parser (use web resources to learn about recursive descent parsing). Such parsers are implemented using a single recursive function for each rule in the grammar. The parsed expression is typically represented using an abstract syntax tree, where the root of the tree represents the operator (e.g. AND) and each subtree represents an operand (e.g. a word).

For example, an abstract syntax tree representation of the query '(graph AND theory) ANDNOT (dot OR math)' will look as follows:

```
     ANDNOT

      /      \

     /         \
      AND        OR

     /    \          /  \

    /       \       /      \

  graph theory dot  math
```

Queries are evaluated by traversing the abstract syntax tree. The result of evaluating a leaf node (a word) is the set of documents containing the word. Internal nodes (boolean operators) are evaluated using their corresponding set operations. For example, an OR node evaluates to the set union of its child nodes, and an AND node evaluates to the set intersection of its child nodes.

Index structure

The index is a data structure that allows queries to be evaluated efficiently. Given the query language defined above, a suitable index data structure is the *inverted index*. Such an index maps each word to the set of documents containing the word. The set and map ADTs from the previous assignments may be used to implement this: use a map where the keys are words and the values are sets of documents containing the words.

Applications

Included in the assignment code is an application called *indexer*. This application takes one command-line argument, which is the name of a directory that contains text files to be indexed. The application creates a new instance of the index ADT, adds all of the files to the index using the index_addpath() function, and proceeds to start a simple web server on port 8080. If you point your browser to http://localhost:8080/ you will be presented with a simple web interface where you can enter queries. Each query is handled by the indexer application using the index_query() function, and the returned list of documents (paths) is displayed in a result page. All you need to do to complete the indexer application is to implement the index ADT. You are not required to change anything in the web interface, although you can if you really want to.

Query and result ranking

A query entered in the web browser is sent to the *indexer* application in the form of a text string. Before calling the index_query() function, the *indexer* will tokenize the query string and place each token in a linked list. For example, if the query string is 'graph AND theory', the *indexer* will create a linked list with nodes containing the text strings 'graph', 'AND', and 'theory'. The index_query() function is presented with the tokenized query list as an argument.

The return value from index_query() must be a linked list containing (the paths of) documents that match the query. Each list entry must be a data structure of the type *query_result_t* (see index.h). It is expected that the ordering of nodes in the result list conveys the relevance of documents to the query. The earlier a document is in the list, the more relevant it should be. We suggest that you use the **tf-idf (http://en.wikipedia.org/wiki/Tf-idf)** algorithm to decide result list ordering.

Code

Your starting point is the following set of files:

- index.h - Specifies the interface of the index ADT.
- map.h - Specifies the interface of the map ADT.
- set.h - Specifies the interface of the set ADT.
- list.h - Specifies the interface of the list ADT.
- hashmap.c - An implementation of the map ADT based on hash tables.
- aatreeset.c - An implementation of the set ADT based on AA trees.
- linkedlist.c - An implementation of the list ADT based on doubly-linked lists.
- common.h - Defines utility functions for your convenience.
- common.c - Implements the functions defined in common.h.
- indexer.c - The indexer program.
- httpd.h - Interface for a simple web server, used by the indexer program to serve the web interface.
- httpd.c - Implements the web server.
- assert_index.c - Performs a series of queries to ascertain that all terms in a document is present in the index.
- template.html/style.css - Used by the http server.
- Makefile - A Makefile for compiling the code.
- cacm - A directory containing text files that you can use to test your system.

As in the first assignment you need to add a new source file that implements your index ADT, and edit the name of this source file into the Makefile, as the value of the INDEX_SRC variable.

We've bundled all of the source code in a zip file. The zip file is located in the same folder as this document (h2-pre.zip).

Deliverables

Keep in mind that this is an exam. You must develop your own understanding of the problem and discover a path to its solution. If you have any general questions, then you may ask someone else for help. But never copy another student's code or report.

There are two deliverables for this assignment: A report and the source code.

Hand in the two deliverables via WiseFlow, where you found this document.

The report must be in the pdf format, and does not need to contain any candidate numbers or names. The report has a page limit of 8 pages. The source code must be placed in a compressed file, you can choose between the zip, rar, tar.gz or tar.bz2 formats.