COMP2521: Assignment 1
Information Retrieval

[The specification may change. Please check the change log on this page.]

> **Change log:**
>
> - [8pm, 26 March] Instructions on "How to Submit" are now available, see under "Submission".
> - The deadline is now extended to 08:00am Thursday Week 7

Objectives

- To implement an information retrieval system using well known tf-idf measures
- To give you further practice with C and data structures (Tree ADT)

Admin

Marks	10 marks (8 marks towards total course mark)
Individual Assignment	This is an individual assignment.
Due	~~09:00am Tuesday~~ 08:00am Thursday Week 7
Late Penalty	1 marks per day off the ceiling. Last day to submit this assignment is 9am Monday of Week-08, of course with late penalty.
Submit	See the section named "How to Submit" under "Submission".

Aim

In this assignment, your task is to implement an information retrieval system using a well known term-weighting scheme called "tf-idf". You should start by reading the Wikipedia entries on these topics. The following Wikipedia page describes how to calculate `tf-idf` values. Later I will also discuss these topics in the lecture.

- tf-idf

For this assignment,

- calculate *relative term frequency* tf(t,d) adjusted for document (d) length,
  `tf(t, d) = (frequency of term t in d) / (number of words in d)`.
- calculate *inverse document frequency* `idf(t, D)` by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient:

$$\text{idf}(\textbf{this}, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

- calculate `tfidf(t, d) = tf(t,d) * idf(t,D)`.

For clarifications, see the example below.

Example of tf–idf [edit]

Suppose that we have term count tables of a corpus consisting of only two documents, as listed on the right.

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

The calculation of tf–idf for the term "this" is performed as follows:

In its raw frequency form, tf is just the frequency of the "this" for each document. In each document, the word "this" appears once; but as the document 2 has more words, its relative frequency is smaller.

$$\mathrm{tf}(\text{"this"}, d_1) = \frac{1}{5} = 0.2$$
$$\mathrm{tf}(\text{"this"}, d_2) = \frac{1}{7} \approx 0.14$$

An idf is constant per corpus, and **accounts** for the ratio of documents that include the word "this". In this case, we have a corpus of two documents and all of them include the word "this".

$$\mathrm{idf}(\text{"this"}, D) = \log\left(\frac{2}{2}\right) = 0$$

So tf–idf is zero for the word "this", which implies that the word is not very informative as it appears in all documents.

$$\mathrm{tfidf}(\text{"this"}, d_1) = 0.2 \times 0 = 0$$
$$\mathrm{tfidf}(\text{"this"}, d_2) = 0.14 \times 0 = 0$$

A slightly more interesting example arises from the word "example", which occurs three times but only in the second document:

$$\mathrm{tf}(\text{"example"}, d_1) = \frac{0}{5} = 0$$
$$\mathrm{tf}(\text{"example"}, d_2) = \frac{3}{7} \approx 0.429$$
$$\mathrm{idf}(\text{"example"}, D) = \log\left(\frac{2}{1}\right) = 0.301$$

Finally,

$$\mathrm{tfidf}(\text{"example"}, d_1) = \mathrm{tf}(\text{"example"}, d_1) \times \mathrm{idf}(\text{"example"}, D) = 0 \times 0.301 = 0$$
$$\mathrm{tfidf}(\text{"example"}, d_2) = \mathrm{tf}(\text{"example"}, d_2) \times \mathrm{idf}(\text{"example"}, D) = 0.429 \times 0.301 \approx 0.13$$

(using the base 10 logarithm).

Download

- `invertedIndex.h`
- Simple example files
  - exmp1.zip (OR download individual files from the dir exmp1)
  - exmp2.zip (OR download individual files from the dir exmp2).

Part-1: Inverted Index, using BST

You need to implement the required functions in the file `invertedIndex.c` that reads data from a given collection of files in `collection.txt` (see simple example files) and generates an "inverted index" that provides a sorted list (set) of filenames for every word in a given collection of files. You need to use binary search ADT to implement your inverted index. For more information on this, please see the following hints:

- **"How to Implement Ass1"**

We will also discuss the above hints in the lecture. Please note that each list of filenames (for a single word) in your inverted index should be alphabetically ordered, using ascending order, and importantly duplicate filenames are not allowed.

> *Hints:* You should use `fscanf` to read words from a file, you do not need to impose any restriction on a line length. You need to use a dynamic data structure(s) to handle words in a file and across files, no need to know max words beforehand. You can assume a maximum word length of 100.

Normalise words: Before inserting words in your inverted index, you need to "normalise" words by,

- removing leading and trailing spaces,
- converting all characters to lowercase,
- remove the following punctuation marks, if they appear at the end of a word.
  - '.' (dot),
  - ',' (comma),
  - ';' (semicolon),
  - ? (question mark)

Please note that if you have multiple punctuation marks at the end of a word, you need to remove only the last punctuation mark. You also don't need to remove the above punctuation marks if they appear in the middle or at the start of a word.

Importantly, you need to modify a given string, do NOT create another copy. You can use the functions `tolower` and `strlen`. You may find the following links useful:

- tolower example
- C Strings

For example,

word	normalised word
.Net	.net
smh.com.au	smh.com.au
Sydney!	sydney!
why?	why
order.	order
text;	text
abc.net.au.	abc.net.au
sydney???	sydney??

You need to implement the following functions in the file `invertedIndex.c`. The API file `invertedIndex.h` is provided, you **must** implement the required functions in your `invertedIndex.c`. We will individually test these functions for awarding marks.

- `char *normaliseWord(char *str);`
  Follow the instructions provided earlier in the specs to normalise a given string. You need to modify a given string, do NOT create another copy.
- `InvertedIndexBST generateInvertedIndex(char *collectionFilename);`
  The function needs to read a given file with collection of file names, read each of these files, generate inverted index as discussed earlier in the specs and return the inverted index. Do not modify `invertedIndex.h` file.
- `void printInvertedIndex(InvertedIndexBST tree);`
  Your program should output a give inverted index tree to a file named `invertedIndex.txt`. One line per word, words should be alphabetically ordered, using ascending order. Each list of filenames (for a single word) should be alphabetically ordered, using ascending order.

For example, `invertedIndex.txt` may look like the following. Please note that the following example is not related to any sample files provided. The example below offers formatting information.

```
design file11.txt file21.txt
mars nasa.txt news1.txt
weather info31.txt nasa.txt news1.txt
```

Part-2: Information Retrieval Engine

In this part, you need to implement an information retrieval function that finds files (documents) with one or more query terms, and uses the summation of `tf-idf` values of all matching query terms (words) for ranking such files (documents). You need to calculate the `tf-idf` value for each matching query term in a file (document), and rank files (documents) based on the summation of `tf-idf` values for all matching query terms present in that file. Use the "inverted index" you created in Part 1 to locate files with one or more query terms and calculate the required `tf-idf` values for such files.

Implement the following information retrieval function `retrieve` in the file `invertedIndex.c` that given search terms (words), returns an ordered list of type `TfIdfList`, where each node contains a filename and the corresponding summation of `tf-idf` values for the given `searchWords`. The list must be in **descending order** of summation of `tf-idf` values. See `invertedIndex.h` for the type definition of `TfIdfList`. You also need to implement another function `calculateTfIdf`.

You need to implement the following two functions in the file `invertedIndex.c`. Total number of documents `D` is provided as an argument in both the functions.

- `TfIdfList calculateTfIdf(InvertedIndexBST tree, char *searchWord, int D);`
  This function returns an ordered list where each node contains a filename and the corresponding `tf-idf` value for a given `searchWord`. You only need to include documents (files) that contain the given searchWord. The list must be in *descending order* of `tf-idf` value. If there are multple files with same `tf-idf` value, order them by their filename in *ascending order*.
- `TfIdfList retrieve(InvertedIndexBST tree, char* searchWords[], int D);`
  This function returns an ordered list where each node contains a filename and the summation of `tf-idf` values of all the matching `searchWords` for that file. You only need to include documents (files) that contain one or more of the given `searchWords`. The list must be in *descending order* of summation of `tf-idf` values (`tfIdfSum`). If there are multiple files with the same `tf-idf` sum, order them by their filename in *ascending order*.

  The `searchWords` array will be terminated with a NULL pointer. Here's an example:

```
char *words[] = { "nasa", "mars", "earth", NULL };
TfIdfList list = retrieve(index, words, 7);
```

Testing

Read instructions provided at the top of the file **`test_Ass1.c`** to run simple tests. The file `test_Ass1.c` is part of the provided sample zip files (see under the section "Download").

- Go to Download

Marking Scheme

This assignment will be marked out of 10 marks. There will be 8 marks for the correctness. We will test each function separately for its correctness, and also all of them together.

There will be 2 marks for "Style and Complexity of code". Regarding time complexity, considering the specs is already asking you to use specific ADTs (BST and ordered lists), the corresponding time complexity will apply, O(n) in both the cases. However, even if your time complexity is O(n), if your code is too complex, your tutor may provide you some feedback and deduct marks.

For example, you should consider **separate ADTs** (*.c and *.h files) for the structures like `InvertedIndexBST`, `FileList`, `TfIdfList`, etc. Each of these ADTs need to support operations required for this assignment (only).

For "Style", we will consider your program layout, meaningful variable names, suitable comments, etc.

Submission

Additional files: You can submit additional supporting files, `*.c` and `*.h`, for this assignment.

IMPORTANT: Make sure that your additional files (*.c) DO NOT have "main" function.

For example, you may implement your binary search tree ADT in files `myBST.c` and `myBST.h` and submit these two files along with other required file `invertedIndex.c`. However, make sure that files you submit do not have "main" function.

I explain below how we will test your submission, hopefully this will answer most of your questions.

You need to submit the following file, along with your supporting files (`*.c` and `*.h`):

- `invertedIndex.c`

Now say we want to mark your functions from your submitted file `invertedIndex.c`. The auto marking program will take all your supporting files (other *.h and *.c) files, along with `invertedIndex.c` and execute the following command to generate executable file (say called `invertedIndex`).

```
% gcc -Wall -Werror -lm -std=c11  *.c  -o invertedIndex
```

So we will **not use your Makefile** (if any). The above command will generate object files from your supporting files and the file to be tested `invertedIndex.c`, links these object files and generates executable file (`invertedIndex` in the above example). Again, please make sure that you **DO NOT have main function in your supporting files** (other *.c files you submit).

How to Submit

Go to the following page, select the tab "Make Submission", select "Browse" to select all the files you want to submit and submit ising "Submit" button. The submission system will try to compile each required file, and report the outcome (ok or error). Please see the output, and correct any error. If you do not submit a file(s) for a task(s), it will report it as an error(s).

You **can now submit this assignment**, click on "Make Submission" tab, and follow the instructions.

Plagiarism

This is an individual assignment. You are not allowed to use code developed by persons other than you. In particular, it is not permitted to exchange code or pseudocode between students. You are allowed to use code from the course material (for example, available as part of the labs, lectures and tutorials). If you use code from the course material, please **clearly acknowledge** it by including a comment(s) in your file. If you have questions about the assignment, ask your tutor.

Before submitting any work you should read and understand the sub section named ***Plagiarism*** in the section "Student Conduct" in the course outline. We regard unacknowledged copying of material, in whole or part, as an extremely serious offence. For further information, see the course outline.

-- end --