Homework 4 - MPCS 51040

(last modified: November 28, 2014)

Issued: November 24, 2014

1 General Instructions

1.1 Compiling

Your code must compile with `gcc -std=c11 -Wall -pedantic`. There should be **no warnings or errors** when compiling with gcc-4.8 (as installed **on `linux.cs.uchicago.edu`**). You can add `-Werror` to force the compiler to treat any warning as an error, and stop compilation immeditately.

Your code will be tested on linux.cs.uchicago.edu and must compile and run there!

1.2 Handing in

To hand in your homework, you need to commit all requested files (**with correct filenames**!) to your personal subversion repository. You should have access to a repository named yourcnetid-mpcs51040-aut-14.

Make a subdirectory called 'homework/hw4' and place your files under that directory. Don't forget to commit your files! You can check on `http://phoenixforge.cs.uchicago.edu` to make sure all files were committed to the repository correctly.

If you can cannot login to `http://phoenixforge.cs.uchicago.edu` or have issues accessing your repository, please contact `techstaff@cs.uchicago.edu`. Please cc `dries@uchicago.edu`.

The deadline for this homework is 5:30pm, December 1, 2014. To grade the homework, the contents of your repository at exactly the deadline will be considered. Late policy: Changes made after the deadline will not be taken into account. No extensions are given.

1.3 Code samples

This document, and any file you might need to complete the homework can be found the subversion repository `https://wooldridge.cs.uchicago.edu/svn/mpcs51040-aut-14`.

1.4 Grading

Your code will be graded based on the following points (in order of descending importance):

- Correctness of the C code: there should be no compiler errors or warnings when compiling as described in 1.1, i.e. **on linux.cs.uchicago.edu**. There should be no memory leaks or other problems (such as those detected by valgrind).
- Correctness of the solution. Your code should implement the required functionality, as specified in this document.
- Code documentation. Properly documented code will help understand and grade your work.
- Code quality: your code should be easy to read and follow accepted good practices (avoid code duplication, use functions to structure your program, ...). This includes writing portable code (which will work on both 32 bit and 64 bit systems).

- Efficiency: your code should not use more resources (time or space) than needed.

Some items might be marked as 'optional' or 'extra credit'. When correctly completing these tasks, the points obtained will go towards mistakes made elsewhere in the homework, potentially raising your grade.

2 Assignment

2.1 Problem Description

The goal of this homework is to become familiar with tree-like data structures. In particular, we will be working with a trie.

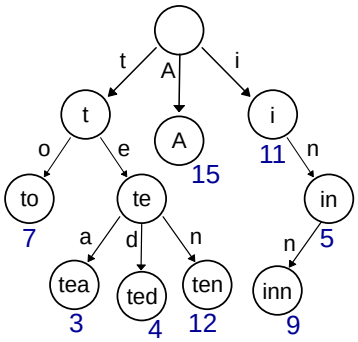

Figure 1: Example trie

A trie is a form of tree, in which each node only stores a part of the key and possibly a value. The value stored in a node corresponds to the key formed by concatenating the parts of the key found when traversing from the root of the trie down to the node holding the value.

Figure 2.1 shows an example trie. Note that each node only stores one letter, but for clarity, the letters stored in the nodes on the path from the root to the node are also included.

We will be reusing part of the code from homework 3, in particular the routines for parsing a text into a list of words.

Most of the code is already provided. There should be no need to modify `trietest.c` or `trie.h`. **If you do modify one of these files, please state so clearly in your `README.TXT` and explain why you needed to make the modification**.

Only two functions (described below) need to be completed in order to obtain a working program (called trietest).

The goal of the program is to determine how often each word of the text occurs. This is done by inserting words into the trie as they are read from the file, with the key being the word, and the value stored is an integer count. If a word already exists, its count is incremented.

2.2 Task 1 - 40 points

Complete the `trie_insert_or_find` function, inserting a word into the trie. This function returns an `trie_pos_t`, which is used by `trietest.c` to update the counter for the given word. Remember that there should be no need to modify any other file than `trie.c`. Your function has to follow the same assumption and rules the other function make, for example the `trie_new` or `trie_destroy` function. If incorrect, the program will either crash during `trie_destroy` or will not properly free all memory. Make sure to test with valgrind!

2.3 Task 2 - 40 points

Now that all the words of the document (and their frequency) are in the trie, output the words (and their frequency in the text) to the screen.

Example:

```
today% ./trietest ../small.txt
must: 2
they: 1
best: 1
worth: 1
Most: 1
part: 1
story: 2
```

```
will: 1
further: 1
stopthat: 1
...
worth: 1
writes: 2
younger: 1
Number of words in trie: 81
```

Code: Complete the `trie_analyze` function, so that it outputs

2.4 Task 3 - Extra Credit

For extra credit output the words in alphabetical order.

2.5 Task 4 - Complexity (20 points)

Create a file `analyse.txt` which clearly answers the following questions:

1. What is the space complexity of the program? Try to be as accurate as possible.
2. What is the time complexity of inserting n words?
3. What is the time complexity of `trie_analyze`?
4. What is the time complexity of the program (overall)?

For these questions, you can assume that there are n words and that the minimum word length is 1, the maximum word length is max and the average word length is p.

2.6 Task 5 (20 points)

Create a file called `README.TXT` which briefly describes how you implemented the requested functionality. Please be clear and complete; In case of accidental mistakes or bugs in your program, this file can help the person grading your work understand your intentions. **Make sure that you commit the homework to the exact directory requested in this document, using the exact filenames.**

Do not underestimate the importance of the README file!

ChangeLog

- 11/28/14: homework/hw3 to homework/hw4 directory correction.