Homework 8 - MPCS 51040

(last modified: May 24, 2017)

May 22, 2017

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 (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 git repository.

Make a subdirectory called 'homework/hw8' and place your files under that directory. Don't forget to commit and push your files! You can check on `http://mit.cs.uchicago.edu` to make sure all files were committed to the repository correctly.

The deadline for this homework is 5:30pm, May 30, 2016. To grade the homework, the contents of your repository at exactly the deadline will be considered.

1.3 Code samples

This document, and any file you might need to complete the homework can be found in the git repository `https://mit.cs.uchicago.edu/`

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 Trie

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.

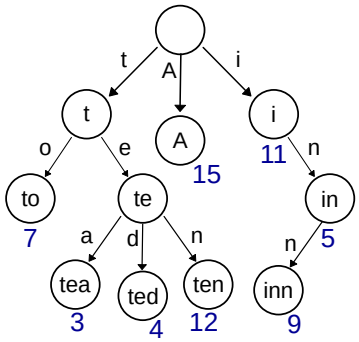

Figure 1: Example trie

A trie is a form of tree, in which each node only stores a part of the key and an optional 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 in a trie, each node only stores one letter, but for visualization purposes, in the image, the letters stored in the nodes on the path from the root to the node are also included. Note that leaf nodes as well as internal nodes might have a value. For example, the node marked 'in' has a value since the word 'in' was added to the tree. The node marked 'te' does not have a value, since no word that was added to the tree terminates at that node.

A unit test for the trie (`trie_test.c`) and the header (`trie.h`) are already provided. There should be no need to modify `trie_test.c` or `trie.h`. **The assignment is to implement, in `trie.c` each of the functions described in `trie.h`.**

2.1.1 Task 1

There is no Makefile provided. It is up to you to provide a makefile, which needs to build the unit test (don't forget to link with the cunit library). For an example, take a look at the makefiles provided with previous homeworks.

2.1.2 Task 2

First, make the unit test compile by providing a stub implementation for each of the required functions. You will also have to provide a struct definition for `trie_data_t` and `trie_node_t`. You should now be able to run '`make test`', which should build the `trie_test` program and execute it.

All tests should fail (and depending on your stub functions, the unit test might crash). (You can always uncomment the functions you haven't implemented yet – see the bottom of `trie_test.c` for this. Don't forget to uncomment once everything is working fine!)

Now, implement `trie_create` and `trie_destroy`, and so on. Note that your trie code should not have any artificial limitations (for example on the maximum length of a word or on the maximum number of words that can be stored).

Some advice:

- Don't forget to frequently check with valgrind, in order to catch problems early on.
- When a test fails, look at the test output, which includes the exact line number of the failed assertion in the `trie_test.c` file. You should be able to deduce what the test was expecting, and hopefully fix your trie function to conform.
- The tests are not independent – for example, if your insert function is not working properly, the other tests (for example for retrieving a value or removing a word) will likely not succeed. So, if a test function fails, make sure to fix all problems before continuing to the next one.
- One possible layout is to have each node store a pointer to the first child node of that node (if any), and to link all child nodes together using a 'sibling' pointer. In other words, a node which has no children will have a `NULL` child pointer, but might have a non-`NULL` sibling pointer if the parent of the node has other children.

Some requirements:

- Your remove function should actually remove the elements from your trie. (Don't just mark them as deleted.) In other words, **the memory usage of your data structure should go down as you remove words from the trie**.

- There are many different ways of designing the trie nodes. You should **not be using more memory than needed**. For example, you can't just allocate a fixed-size array of child pointers for your trie node (since that would mean that a trie node is using more memory than is needed to store the actual number of children it has).

2.2 Handing in

Please commit `trie.c` and your `Makefile` to the repository.