TDQC

Intersect

Jack Spence

5 October 2018

1. Write-Up

1.1 Requirements

Requirements were to read from at least two files and find words that appear in every file and print them in alphabetical order.

1.2 Suggested Features

Suggested feature attempted in my submission is to accept a hyphen as the first command line argument, thus reading from stdin. In addition, I have attempted sorting UTF8 encoded words, which will appear after English letters a-z. Lastly attempted was to strip leading and trailing punctuation and ignore them for comparison.

1.3 Syntax

Compile:

1. Make – Make alone will compile the default binary of intersect

2. make profile – Will produce the executable with the profile flags set to produce gmon.out

3. make debug – Produces executable with debug flags set

4. make clean – Cleans the executable as well as any .a, .o. and .outs in the current directory

Usage: ./intersect filename1 filename2 [filename3 ...]

1.4 Brief Overview of Functions

The structure used in intersect is described below

```
typedef struct Node
{
    struct Node *leftNode;
    struct Node *rightNode;

    char *word;
        int count;
} Node;
```

I created an API for dealing with the struct and sorting data, the API functions are as follows

```
/* Taking a list of arguments, opens the first file and creates
 * a BST. Subsequent files are compared to the BST and count is
 * updated if appropriate */
Node *getWords(int *files_opened, int argc, char *argv[]);
/* Takes a word and a buffer for a temp variable to use for
 * removing leading and trailing punctuation */
int ignorePuncCmp(const char *tree_word, const char *word);
/* Function that does the stripping of punctuation for ignorePuncCmp */
void stripPunct(const char *word, char *tmp, bool *symbols);
/* Generic BST Print function */
void printTree(Node *tree, int argc);
```

```
/* Generic BST Destroy function */

void destroyTree(Node *tree);
```

2. Project Design Plans

2.1 Initial Design Plans

This project was pretty similar to one we had received in the past, so I decided to implement intersect in a similar way, by using a BST. I had implemented a few BSTs in the past so I was not too concerned with this project before beginning.

2.2 What didn’t work

The longest and most frustrating portion of this project was implementing the punctuation extra feature. I could not get my algorithm for temporarily stripping the punctuation to work and took several rewrites and redesigns to get there. Implementing this feature also increased the run-time of my program significantly. I believe not relying on the BST and using a hash table would have sped the program up a bit, but the nature of the way I implemented the feature is a significant bottleneck.

2.3 What went well

Using my skeleton BST I created last week, the first stages of intersect went pretty well, and the base functionality was working within the first 6 hours of working on the project. As well as implementing two of the 3 suggested features was only a few lines of code at most.

2.4 Conclusion

I need to get around to writing skeletons for other abstract data structures to make implementing these projects a bit easier and to use the best one for any particular assignment. I also should start writing a string library for some of the more common operations we have been doing in TDQC.

3. Test Procedures

3.1 Test files

I have listed the files I used in testing under the tests directory in the project. Mainly man pages for a few linux executables, with a couple I created in there as well. I also used the dictionary at usr/share/dict/american-english, but didn’t think it was a good idea to upload it as it takes almost a minute for my program to parse it.

3.2 Testing Procedure

I had a few test cases listed in the various testfiles that I particularly wanted to test for. In addition to those in the testfiles, I used grep on the man files to double check the findings of my intersect executable. To test for the UTF8 requirements I used the “russian” file as well as the file labeled UTF8 and made sure they were sorted by what I think is the proper sorting.