HONORS DATA STRUCTURES PROJECT – PREDICTIVE TEXT
FINAL REPORT

Overview

The goal of this project was to use an implementation of a tree to create a predictive text program that learns by "reading" files to find how commonly used a word is, as well as having the ability to dynamically learn words. The solution of this program was split into two parts: a console-based application (*TreeEdit.exe*) which allows for basic tree manipulation such as reading files and wiping and saving the tree file; and a Qt application (*PredictiveText.exe*) which predicts words using the tree created with the TreeEdit program. A third program was developed to analyze the effectiveness of the tree at predicting words by using a sample set of words.

The tree designed for this program has a node for every character of words it tracks, with ends of words signified by a null character '\0'. The root of the tree is also a null character node. Nodes contain the character they represent, as well as the usage count of the most common word this node is part of. When learning a word, the program traverses to the end of the word (creating new nodes as needed), increments the usage count of the null-character node representing the end of the word, and then returns to the root, modifying the maximum uses of nodes it visits if necessary. Predicting a word then becomes simply traversing down the most common path until the end of the word is found. Figure 1 shows a diagram of a simple tree containing the words "air", "airplane", "apart", "apple", "apples", and "apiece."

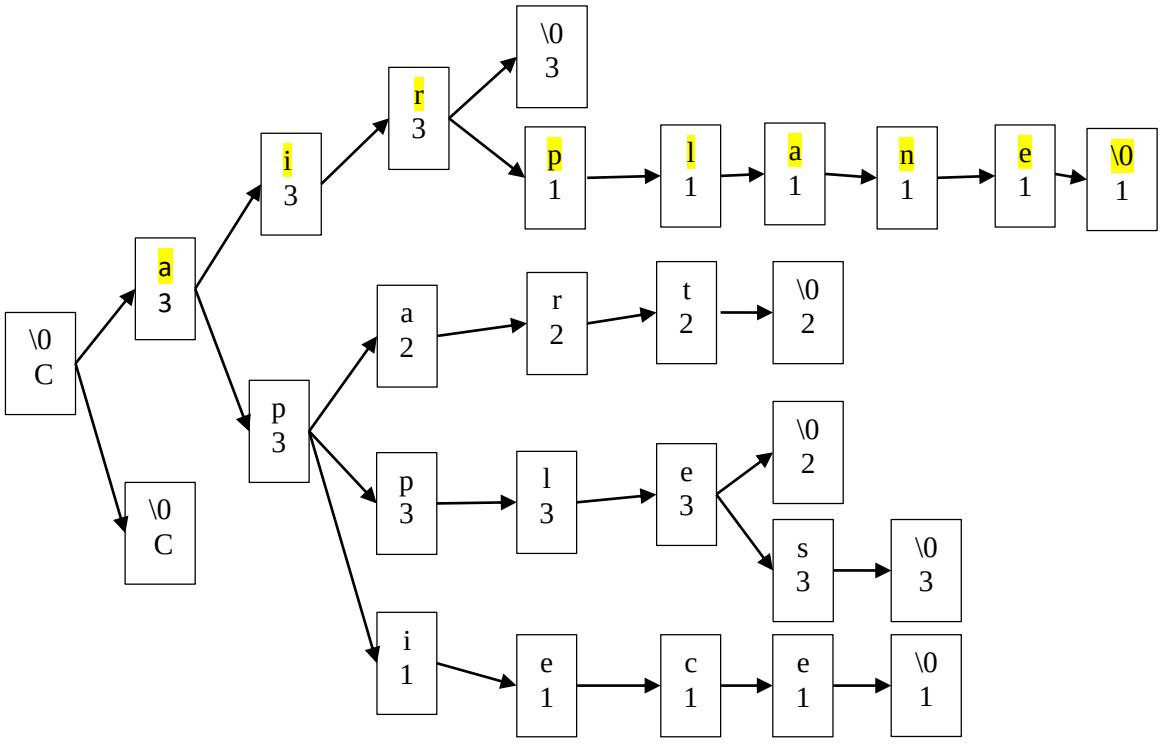

Figure 1: A simple example of the tree implemented. Note that the tracked number for each node is the number of times the most common word using this node has been processed. The trace for the word airplane has been highlighted.

Since the tree is used for multiple programs and needs to retain its knowledge through separate runs of the program, it must save to a file. In order to maintain a compact but somewhat decipherable file, a compact text format similar to JSON is used. Figure 2 shows an example of this save format.

```
{' ,xC,[
   {'a,x3,[
      {'i,x3,[
         {'r,x3,[
            {' ,x3,[]},
            {'p,x1,[
               {'l,x1,[
                  {'a,x1,[
                     {'n,x1,[
                        {'e,x1,[
                           {' ,x1,[]}]}]}]}]}]}]}]}]}]}]},
   {'p,x3,[
      {'a,x2,[
         {'r,x2,[
            {'t,x2,[
               {' ,x2,[]}]}]}]},
      {'p,x3,[
         {'l,x3,[
            {'e,x3,[
               {' ,x2,[]},
               {'s,x3,[
                  {' ,x3,[]}]}]}]}]},
      {'i,x1,[
         {'e,x1,[
            {'c,x1,[
               {'e,x1,[
                  {' ,x1,[]}]}]}]}]}]},
   {' ,xC,[]}]}
```

Figure 2: An example of the save file generated from the tree displayed in Figure 1 (see previous page), in which the trace for the word “airplane” has been highlighted. Nodes have the form `{'c,x#,[…]}`, where `c` is the character of the node, `#` is a hex representation of the number of uses of the maximum word using this node, and the data inside the square brackets (`[…]`) is the children of this node. In this example, a space is used to denote the null character `'\0'`. Whitespace was added to this example to enhance readability. Note that the save tree files must be opened with UTF-8 encoding.

Analysis

The program was tested using three different sets of files to learn from: classic books, Wikipedia articles, and Reddit posts. Since novels tend to use a wider variety of words than Wikipedia articles or commenters on Reddit posts, it is logical that the tree would know more words when trained on this. The contents of the Reddit pages use informal language and thus a smaller amount of words; however, since words are reused more often, the tree arguably becomes smarter at predicting those words which it does know when trained using Reddit. Wikipedia is an intermediary between these two, since it uses mostly simple words mixed with jargon in its articles.

The program was tested with two files – a list of random words from the dictionary and a list of common verbs – to see how quickly it correctly predicted words based off of each of the three separate training sets. Sources for these word lists can be found at the end of the document. Table 1 (see next page) displays the results of this testing. In the table, average letters predicted refers to the average number of letters remaining in a word when the program correctly predicts it. Words not predicted refers to the number of words where the testing program reached the end of the word without predicting any letters, caused by either predicting a different word or by not having that character in the database. In terms of both letters the

program predicted and number of words predicted out of the set of 218 words, the tree trained with novels performed best, followed by Reddit and Wikipedia.

	Wikipedia	Reddit	Novels
Average letters predicted	0.896774	1.205263	1.335165
Words not predicted	155	109	104

Table 1: Performance with Separate Training Sets.

Compiling and Executing Instructions

The TreeEdit program was developed using Visual Studio 2015. All files for this program are located in the *TreeEdit* folder included in this ZIP. To view the project in Visual Studio, choose *File->Open->Project/Solution* and select the *PredictiveText.pro* file located in the *TreeEdit* folder included in this ZIP.

The PredictiveText program was developed using Qt Creator, which can be downloaded here. Specifically, this program was implemented using Qt 5.5. Compiling with Qt Creator requires Visual Studio 2013 to be installed as well, which can be downloaded here. To view the project in Qt Creator, choose *File->Open File or Project* and select the *PredictiveText.pro* file located in the *PredictiveText* folder included in this ZIP.

Additionally, precompiled executables of these programs are included in the ZIP. *TreeEdit.exe* is the console application and requires the TreeEdit_help.txt file to function properly. Documentation for this program can be found either by typing entering the command “help” or by viewing the *TreeEdit_doc.rtf* file included in the ZIP. *PredictiveText.exe*, the Qt application, requires the *Qt5Core*, *Qt5Gui*, and *Qt5Widgets* dll files in order to run. Documentation for this program can be found in the *PredictiveText_doc.rtf* file included in the ZIP.

Sources

All books were obtained and used legally through Project Gutenberg. Other training text was obtained from various Wikipedia articles and Reddit posts. Links to specific pages can be found in these files.

The common verbs used for testing robustness of the tree were obtained through world-english.org. The random words were obtained through Text Fixer.