Project 3- Text Processing
CSCI 230 T Th 11:10 am
Compiler: g++
OS: Windows 10/WSL

Michael Morikawa

May 12, 2020

Notes

Status

Both HuffmanCoding and Trie portion of the project are completed with no errors.

Extra Credit

Did the improved standard trie that gives the amount of times a word occurs.

Design Decisions

For the HuffmanCoding input, my program will only work with window style line endings. If its not then it will not read a new line character. The solution for that is commented out in the code; to fix it I would just need to add a newline after each call to getline and the remove the final newline character since it will add an extra one.

For the trie I decided to ignore the numbers simply because the child array is much smaller. In order to include the numbers while still using a lookup table as the child array it would have to be much larger since numbers are not right next to the lowercase letters in the ascii table.

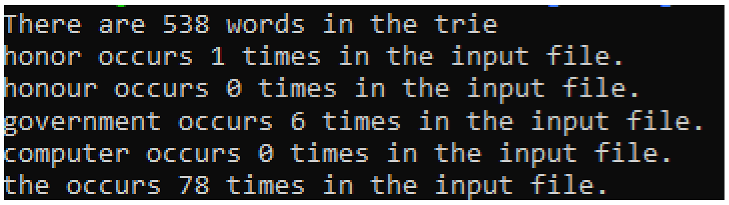
output

moneyOut.txt

```
  0000
    100
d  1011
e  11
m  001
n  011
o  010
r  0001
y  1010

____________
Number of characters: 18
Number of bits: 54
Compressed: 000000101000011110000101001111101010001111111011111011
```

Trie Output



```
There are 538 words in the trie
honor occurs 1 times in the input file.
honour occurs 0 times in the input file.
government occurs 6 times in the input file.
computer occurs 0 times in the input file.
the occurs 78 times in the input file.
```

Source Code

main.cpp

```cpp
#include <vector>
#include <fstream>
#include <algorithm>
#include <iostream>
#include "HuffmanCoding.hpp"
#include "Trie.hpp"

int main()
{
    HuffmanCoding test("docs/moneyIn.txt", "docs/moneyOut.txt");
    test.compress();
    Trie declaration("docs/usdeclarPC.txt");
    std::string searchTerms[]{"honor", "honour", "government", "computer", "the"};
    int occurences = 0;
    std::cout << "There are " << declaration.size() << " words in the trie\n";
    for (std::string s : searchTerms)
    {
        occurences = declaration.search(s);
        std::cout << s << " occurs " << occurences << " times in the input file.\n";
    }
}
```

HuffmanNode.hpp

```cpp
#pragma once

class HuffmanNode
{
public:
    HuffmanNode(char c, int f, HuffmanNode *l, HuffmanNode *r)
    {
        data.first = c;
        data.second = f;
        left = l;
        right = r;
    }
    HuffmanNode(std::pair<char, int> d, HuffmanNode *l, HuffmanNode *r)
    {
        left = l;
        right = r;
        data = d;
    }
    int frequency()
    {
        return data.second;
    }
    char getChar()
    {
        return data.first;
    }
    bool isExternal()
    {
        return left == NULL && right == NULL;
    }
    HuffmanNode *getLeft()
    {
        return left;
    }
    HuffmanNode *getRight()
    {
        return right;
    }

private:
    std::pair<char, int> data;
    HuffmanNode *left;
    HuffmanNode *right;
};
```

HuffmanCoding.hpp

```
#pragma once
#include <map>
#include "HuffmanNode.hpp"

class HuffmanCoding
{
public:
    HuffmanCoding(const char *inFile, const char *outFile);
    void compress();

protected:
    void buildFreqTable();
    HuffmanNode *buildTree();
    typedef std::pair<char, int> pair;
    void getCodes(HuffmanNode *node, std::string prefix,
                  std::map<char, std::string> &output);

private:
    std::map<char, int> freqTable;
    const char *inputFileName;
    const char *outputFilename;
    std::string text;
};
```

HuffmanCoding.cpp

```
#include <fstream>
#include <string>
#include <queue>
#include "HuffmanCoding.hpp"
#include "HuffmanNode.hpp"

class Greater
{
public:
    bool operator()(HuffmanNode *a, HuffmanNode *b) const
    {
        return a->frequency() > b->frequency();
    }
};

HuffmanCoding::HuffmanCoding(const char *inFile, const char *outFile)
    : inputFileName(inFile), outputFilename(outFile)
{
}
```

```
void HuffmanCoding::buildFreqTable()
{
    std::ifstream inFile(inputFileName);
    std::string temp;
    while (std::getline(inFile, temp))
    {
        text += temp;
        //Have to add a new line since getline doesn't include it
        //!Only on LF line endings?
        //   text.push_back('\n');
    }
    //removes extra newline
    //text.pop_back();
    for (char c : text)
    {
        if (freqTable.count(c))
        {
            freqTable[c]++;
        }
        else
        {
            freqTable[c] = 1;
        }
    }
}

HuffmanNode *HuffmanCoding::buildTree()
{
    buildFreqTable();
    std::priority_queue<HuffmanNode *, std::vector<HuffmanNode *>, Greater> nodes;
    for (auto p : freqTable)
    {
        nodes.push(new HuffmanNode(p, NULL, NULL));
    }
    HuffmanNode *left;
    HuffmanNode *right;
    int freqSum;
    while (nodes.size() > 1)
    {
        left = nodes.top();
        nodes.pop();
        right = nodes.top();
        nodes.pop();
        freqSum = left->frequency() + right->frequency();

        nodes.push(new HuffmanNode('0', freqSum, left, right));
    }
    HuffmanNode *root = nodes.top();
    nodes.pop();
    return root;
}

void HuffmanCoding::getCodes(HuffmanNode *node, std::string prefix,
                             std::map<char, std::string> &output)
{
    if (node->isExternal())
    {
        output[node->getChar()] = prefix;
    }
    else
    {
        getCodes(node->getLeft(), prefix + "0", output);
        getCodes(node->getRight(), prefix + "1", output);
    }
}

void HuffmanCoding::compress()
{
    HuffmanNode *root = buildTree();
    std::map<char, std::string> codes;
    getCodes(root, "", codes);
    std::string result;
    std::ofstream outfile(outputFilename);
    for (char c : text)
    {
        result += codes[c];
    }
    for (auto p : codes)
    {
        outfile << p.first << ' ' << p.second << '\n'; //Output the table containing the character codes
    }
    outfile << "\n--------\nNumber of characters: " << root->frequency()
            << "\nNumber of bits: " << result.length() << "\nCompressed: "
            << result;
}
```

Trie.hpp

```
#pragma once
#include <string>

const int ALPHABET_SIZE = 26;

class TrieNode
{
public:
    TrieNode() : isEndOfWord(false), count(0)
    {
        for (int i = 0; i < ALPHABET_SIZE; i++)
        {
            children[i] = NULL;
        }
    }

    TrieNode *children[26];
    bool isEndOfWord;
    int count;
};

class Trie
{
public:
    Trie(std::string file);
    void insert(const std::string &word);
    int search(const std::string &word); //Returns the number of times the word occurs in the text
    int size() const;                    //Returns the number of unique words stored in a tree
private:
    int n;
    TrieNode *root;
};
```

Trie.cpp

```
#include "Trie.hpp"
#include <algorithm>
#include <fstream>
#include <iostream>

Trie::Trie(std::string file) : n(0), root(new TrieNode())
{
    std::ifstream infile(file);
    std::string word;
    while (infile >> word)
    {
        //Removes anything that is not a letter from the word
        word.erase(std::remove_if(word.begin(), word.end(), [](char c) { return !isalpha(c); }), word.end());
        //If the word was just a number don't add the empty string to the trie
        if (word.length() == 0)
        {
            continue;
        }
        for (int i = 0; i < word.length(); i++)
        {
            word[i] = std::tolower(word[i]);
        }
        insert(word);
    }
}

int Trie::size() const
{
    return n;
}

void Trie::insert(const std::string &word)
{
    TrieNode *node = root;
    int index = 0;
    for (char c : word)
    {
        index = c - 'a';
        if (node->children[index] == NULL)
        {
            node->children[index] = new TrieNode();
        }
        node = node->children[index];
    }

    if (!(node->isEndOfWord))
    {
        node->isEndOfWord = true;
        n++;
    }
    node->count++;
}

int Trie::search(const std::string &word)
{
    TrieNode *node = root;
    int index = 0;
    for (char c : word)
    {
        index = c - 'a';
        if (node->children[index] == NULL)
        {
            return 0;
        }
        node = node->children[index];
    }
    return node->count;
}
```