BLG 336E – Analysis of Algorithms II, Spring 2014

Project 2 – Huffman Coding

Total Worth : 10% of your grade

Handed Out : 08.04.2014, Tuesday

Due : 22.04.2014, Tuesday

Overview

Huffman coding is an algorithm for lossless data compression that assigns a binary code to each character in a text, based on frequency rate of the character. The algorithm uses a greedy approach in constructing an optimal prefix codes.

In this project, you will implement Huffman coding algorithm to encode amino acid sequences of proteins.

Preliminary Information

Huffman coding

The main idea behind Huffman coding is to assign shorter binary codes to the characters that occur more frequently in a text document. While other encodings without compression such as ASCII uses fixed length codes for all characters. Table below displays ASCII and Huffman codes of some characters. Huffman codes in this example are built based on the frequencies of the characters in Shakespeare's Hamlet[i].

Character	ASCII value	ASCII (binary)	Huffman (binary)
' '	32	00100000	10
'a'	97	01100001	0001
'b'	98	01100010	0111010
'c'	99	01100011	001100
'e'	101	01100101	1100
'z'	122	01111010	00100011010

As you can see, rare characters such as ‘z’ have long Huffman codes while frequent ones such as space are very short.

A greedy algorithm is used in order to generate Huffman codes. First step is to count frequencies of all characters in the text. Let’s assume that your input consists of integers in [1,6]. You need to count the number of occurrence of each integer in your input. And create a frequency:value mapping and sort it by frequencies as follows.

[i] This table is taken from http://courses.cs.washington.edu/

```
5 :1    7 :2    10:3    15:4    20:5    45:6
```

Next step is to build a code generation tree from this sorted map[ii]. At each iteration you should remove the two least frequent items from your map, create a parent node with a frequency that is the sum of these items, and finally add this parent node back to the list in the correct order of frequency. More frequent child should be on the right. For our example, 1 and 2 are the least frequent ones, so we remove them and create a parent:

```
12:{1,2}
  /   \
5:1   7:2
```

Our new map is as follows:

```
[10:3   12:{1,2}    15:4        20:5    45:6]
```

New least frequent items are 3 and {1,2} so we need to combine them:

```
 22:{1,2,3}
   /    \
10:3   12:{1,2}                          [15:4   20:5    22:{1,2,3}      45:6]
        /    \
      5:1    7:2
```

Next we need to combine 4 and 5:

```
 35:{4,5}
   /    \                                [22:{1,2,3}     35:{4,5}        45:6]
15:4  20:5
```

Then, {1,2,3} and {4,5}:

```
       57:{1,2,3,4,5}
      ____/     \____
     /               \
 22:{1,2,3}         35:{4,5}            [45:6   57:{1,2,3,4,5}]
   /    \             /    \
10:3   12:{1,2}  15:4   20:5
        /    \
      5:1    7:2
```

Finally, we combine remaining two items and the tree is complete:

```
                102:{1,2,3,4,5,6}
   ____________________/     \__
  /                           \
45:6                    57:{1,2,3,4,5}
                      _____/     \_____
                     /                 \
                 22:{1,2,3}          35:{4,5}
                   /    \              /    \
                10:3  12:{1,2}     15:4   20:5
                        /    \
                      5:1    7:2
```

[ii] Tutorial on generating the Huffman tree is based on http://www.siggraph.org/

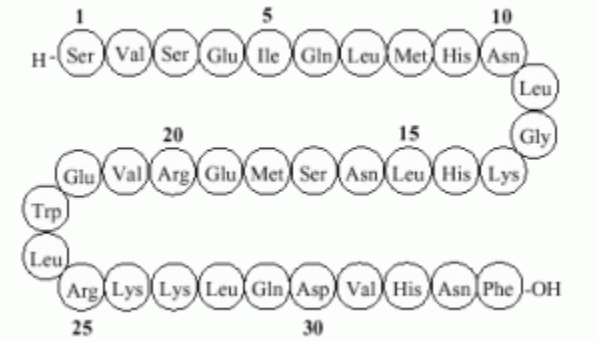
After the tree is created, you can generate codes by traversing the tree from root to the leaves and outputting a 0 every time you take a left-hand branch, and a 1 every time you take a right-hand branch. For instance, to reach item 6, only one left branch is enough. So its Huffman code is simply 0. Similarly to reach 3, you need to make a right branch then two left branches, so the code is 100. As explained before, rare items such as 1 and 2 have longest Huffman codes (1010 and 1011, respectively).

If we use fixed length codes for these 6 items, we need to denote each one of them with 3 bits. As a result, we need 102 x 3 = 306 bits to represent the input. However, Huffman coding uses only 45 x 1 + 20 x 3 + 15 x 3 + 10 x 3 + 7 x 4 + 5 x 4 = 228 bits. That is a 25% compression rate.

When you decode an input, you need to traverse the tree top-down according to the bits in the input left to right. You should take a left branch at each 0 and a right branch at each 1. When you reach a leaf, you should print it and restart from the root until you reach the end of the input. For instance, if the input is 1000110, then the output is 364. (100 → leaf (3) → Go to the root → 0 → leaf (6) → Go to the root →110 → leaf (4) → End of input)

Protein Sequencing

Amino acid sequence is the order of amino acids that are connected by peptide bounds in a protein. It represents the unique primary structure of the protein. Each protein may have a different number of amino acids. A sample sequence is given below.



There are 20 common amino acids and they are usually abbreviated to 3 letters.

Alanine ala	Arginine alg	Asparagine asn	Aspartic acid asp	Cysteine cys
Leucine leu	Lysine lys	Methionine met	Phenylalanine phe	Proline pro
Glutamic acid glu	Glutamine gln	Glycine gly	Histidine his	Isoleucine ile
Serine ser	Threonine thr	Tryptophan thp	Tyrosine tyr	Valine val

Provided input file (proteins.txt) contains the sequences for 16 real world proteins. Each line in the file represents a protein and the amino acids in the same line are separated by spaces.

Project Definition

Code (60 pts.)

Different from the example above, you will encode 3-letter strings representing amino acids, instead of individual characters using Huffman coding. Your program should:

1. Read in the entire input file, and calculate the frequencies of all amino acids. **[10 pts.]**
2. Build a Huffman tree for all amino acids that appear in the input file.
3. Generate codes for all amino acids and store the codes in a lookup table. **[25 pts.]**
4. Print amino acid names, frequencies and their codes to the standard output (console).
5. Use lookup table to encode sequences of all 16 proteins in the input file and print the result to the output file (encodedProteins.txt). **[25 pts.]**
   Each line in the output file should represent a protein and there should be no space between the amino acid codes.
6. Receive a binary input from the standard input (console) and decode it by traversing the Huffman tree. You may assume that the input would be well-formed. **[Bonus 20 pts.]**

Your program should compile and run using the following commands:

```
g++ yourStudentID.cpp –o Huffman
./Huffman protein.txt
```

Your program will be evaluated with a different input file, with different proteins.

Report (40 pts.)

1. Explain the purposes of all your classes and methods. **[5 pts.]**
2. Explain the data structures you have implemented. **[5 pts.]**
3. Show complexity of your algorithm on pseudo-code using Θ notation. **[5 pts.]**
4. Draw your final Huffman tree. **[10 pts.]**
5. Calculate how many bits in total would a fixed length encoding require and how many bits your Huffman encoding used to represent the input. Give the compression rate. Explain in what kind of text input you expect to have better compression? **[15 pts.]**

Submission

You should be aware that the Ninova system clock may not be synchronized with your computer, watch, or cell phone. Do not e-mail the teaching assistant or the instructors your submission after the Ninova site submission has closed. If you have submitted to Ninova once and want to make any changes to your report, you should do it before the Ninova submission system closes. Your changes will not be accepted by e-mail. Connectivity problems to the Internet or to Ninova in the last few minutes are not valid excuses for being unable to submit. You should not risk leaving your submission to the last few minutes. After uploading to Ninova, check to make sure that your project appears there.

Policy: You may discuss the problem addressed by the project at an abstract level with your classmates, but you should not share or copy code from your classmates or from the Internet. You should submit your own, individual project. Plagiarism and any other forms of cheating will have serious consequences, including failing the course.

Submission Instructions: Please submit your project files through Ninova. Please upload all your code files in a *.zip or *.rar archive. In the archived file, you must include all your program and header files. A second *.pdf file that contains your report must also be uploaded.

All your code must be written in C++, and we must be able to compile and run on it on ITU's Linux Server (you can access it through SSH) using g++. You should supply one yourStudentID.cpp file that calls necessary routines for all questions (Multiple files are acceptable, as long as you state the compilation instructions in your report).

When you write your code, try to follow an object-oriented methodology with well-chosen variable, method, and class names and comments where necessary. Your code must compile without any errors; otherwise, you may get a grade of zero on the assignment.

If anything in this document is not clear, please let the teaching assistant know by email (aralat@itu.edu.tr).