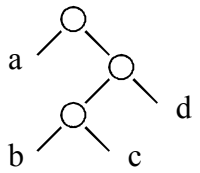
DAAI Lab, Thu Dau Mot University
Data Struture and Algorithm in Python Course
Summer 2018
Assignment 2
Due 09 June, 2018, 12:00 pm

Introduction

One of the most important lossless data compression algorithms is called Huffman coding. A Huffman code is defined by a tree, whose leaves are the symbols in the alphabet. For example, the tree



If we indicate going left by 0 and going right by 1, then the Huffman code for the above tree is:

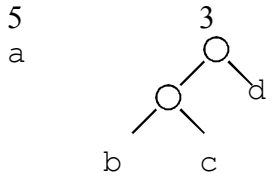
```
a  0
b  100
c  101
d  11
```

A string of symbols aadbaaca would be encoded as 0011100001010. This would yield compression considering the original requires 8 bits per symbol. So 64 bits have been reduced to 13 bits. The reason we get compression is that the symbol "a" occurs quite frequently in the original and the Huffman code uses just one bit to encode it. There is a simple process to decoding a Huffman code. Start at the root of the tree. If you are at a leaf output the symbol. Otherwise read a bit and go left if is 0 and go right if is 1 and continue in that manner until reaching a leaf. An *optimal Huffman* code is one that produces the shortest code given frequencies for the symbols.

It turns out there is an elegant algorithm for generating an optimal Huffman code. The algorithm uses a priority queue. First you need to calculate the frequency of each symbol in the input. Make a leaf node for each symbol and store its frequency in the node. Repeatedly do the following, find the two trees with the smallest frequencies. Make them the left and right children of a new node whose frequency the sum of the two frequencies. When one tree remains we are done. In the example above the frequencies are a:5, b:1, c:1, d:1.

```
5    1    1    1
a    b    c    d
```

```
5          2          1
a                     d
          b   c
```



In one more step we are done. The average bit rate of the code can be computed as

$ABR = (F_1L_1 + F_2L_2 + \ldots + F_mL_m)/N$

where F_i is the frequency of the i'th symbol, L_i is the length of the code for the i'th symbol and N is the length of the file. Without actually compressing the file the compression ratio for a text file can be computed as 8/ABR. This is because the uncompressed tex file is stored with 8 bits per symbol.

The goal in this project is to use predefined priority queues to build an optimal Huffman tree. Your priority queue will maintain the current set of trees ordered by their frequencies. One challenge is to efficiently traverse the optimal Huffman tree to generate the code to be printed out.

The User Interface

Again we are not too concerned with the user interface because we are working with the data structures. The program requests a text file name, then computes the optimal Huffman code and prints it. The compression ratio is then printed.

```
> Please enter a text file name:
foo.txt
> The optimal Huffman code is:
> a  0
> b  100
> c  101
> d  11
> The compression ratio is:
> 4.92
```

Your program should check for invalid inputs.

Good Luck!