PA3 Check point report
Cheng Qian A53209561

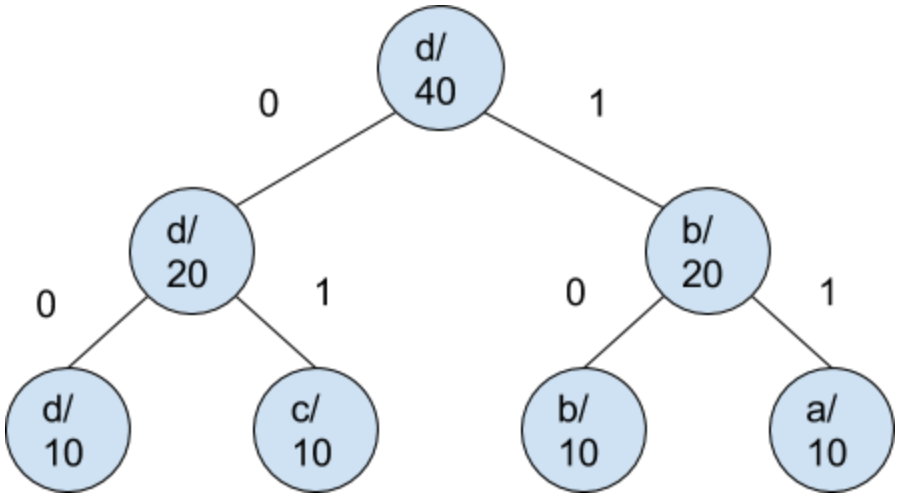
encode1.txt
11100100111001001110010011100100111001001110010011100100111001001110010011100100111001001110010011100100

encode2.txt
00000000100100100101010101010101011111111111111111111111111111111110101010101010101001001001001000000

To build the tree, I first make a statistical table of all the frequency of the256 ascii characters. In the checkpoint1.txt, the charcter-frequency table is as a followed:

character	frequency
a	10
b	10
c	10
d	10

Thus, all the symbols have same frequency in this text. We make the rule to break the tie by their alphabetical order (the ‘large’ character will have higher priority). So the sequence in the priority queue is a<b<c<d. We first pop the highest two priority nodes out, which are c and d. Let d to be node c0 and c to be node c1. Merge them into a new tree with symbol of c0 and count of the sum of two child nodes’ sum. And iteratively repeat those steps. Finally, we can get a tree as belowed.



After building the tree, we can encode the symbol by backtracking the leaf node's parent pointer. For example, if we encounter the symbol 'a', all we need to do is first go to leaves[] vector and find the address of the node 'a', and then backtrack through parent pointers, recording the number 1 or 0 in the path in a stack, and finally pop the number out from the top of the stack. Only use a stack can the output be the right order. So the encoded message for symbol 'a' is '11'. And other symbol's result is as followed:

symbol	codeword
a	11
b	10
c	01
d	00

Actually at first my output is wrong, because the last byte of the original message will appear twice in the decoded message. After debugging and checking on the c++ stl, I found that the problem is in the function "ifstream.eof()".

The C++ STL shows, as I quote,

std::ios::eof

bool eof() const;
Check whether eofbit is set
Returns true if the eofbit *error state flag* is set for the stream.

This flag is set by all standard input operations when the End-of-File is reached in the sequence associated with the stream. Thus even it is a empty file, there is an end of file bit in the file. If I use this function to judge whether the program should jump out of the file loop:

```
while(!in.eof()){
        in>>letter_byte;// at the last byte, the compiler will set the point to stop at the last byte,
```
thus the last byte will be read one more time, which is the reason why my decoded message has a repeated ending

There are two ways to solve it:

1. Set the condition of while loop always be true and use break to break out the loop

```
        while(true){
                in>>letter_byte;
                if(in.eof()) break;
                int idx = (int)letter_byte;
                freqs[idx] =  freqs[idx] + 1;
        }
```

2. Read first and enter the loop

```
        in>>letter_byte;
    while(!in.eof()){
         tree.encode(letter_byte, out);
         in>>letter_byte;
    }
```

After fixed this bug, I got the right decoded output as same as the original input sequence of symbols.