Project

Huffman Codes

Group 17

Date:2022-05-08

Chapter 1: Introduction

```
In 1953, David A. Huffman published his paper "A Method for the
Construction of Minimum-Redundancy Codes", and hence printed his
name in the history of computer science. Huffman coding is a coding
method, and Huffman coding is a kind of variable word length coding
(VLC). This method constructs codewords with the shortest average
length of different prefixes based on the probability of occurrence
of characters. It is sometimes called the best coding. However,
there is a big problem that the Huffman codes are not unique. For
example, given a string "aaaxuaxz", we can observe that the
frequencies of the characters 'a', 'x', 'u' and 'z' are 4, 2, 1 and
1, respectively. We may either encode the symbols as {'a'=0, 'x'=10,
'u'=110, 'z'=111}, or in another way as {'a'=1, 'x'=01, 'u'=001,
'z'=000}, both compress the string into 14 bits. Another set of code
can be given as {'a'=0, 'x'=11, 'u'=100, 'z'=101}, but {'a'=0,
'x'=01, 'u'=011, 'z'=001} is NOT correct since "aaaxuaxz" and
"aazuaxax" can both be decoded from the code 00001011001001. In this
project, we will make a computer program to help determine which
ones are correct and which ones are not.

This project is about verifying whether one coding method can be
equal to Huffman code in efficiency, or it is already another form
of Huffman code.
```

Chapter 2: Algorithm Specification

Main Program

The main program uses function preinput and readsol to read in data, huffman to process the data and, verify to get the answer(Yes/No).

```
signed main(signed, char**, char**) {
    preinput();
    huffman();
    For i = 0 To m - 1 Step 1
        readsol();
        Print verify();
}
```

preinput

This part is to read in the number of characters, characters, their frequencies and, finally, the number of student submissions.

```
void preinput() {
    Input n;
    For i = 0 To n - 1 Step 1
        Input idor[i];
        id[idor[i]] = i;    // Map characters into integer [0,n)
        Input freq[i];        // Frequency
    Input m;
}
```

huffman

This part is a simplified huffman-code process, only to calculate the coded length of the original string under Huffman code.

```
void huffman() {
    priority_queue<int, vector<int>, greater<int>> heap;    //Using a min-hea
    For i = 0 To n - 1 Step 1
        heap.push(freq[i]);
    while (heap.size() != 1) {
        int a = heap.top();
        heap.pop();
        int b = heap.top();
        heap.pop();
        len = len + a + b;
        heap.push(a + b);
    }
}
```

readsol

This part is to read in student submissions.

```
void readsol() {
    For i = 0 To n - 1 Step 1
        Input cdi[i];
        Input code[id[cdi[i]]];
}
```

prefix

This part is to determine prefix to verify the correctness of a huffman-code.

```
bool prefix(int a, int b) {
    if (code[a].length() > code[b].length()) then
        return 0;                // Length exceeded, apparently not a prefix
    For i = 0 To code[a].length() - 1 Step 1
        if (code[a][i] != code[b][i]) then
            return 0;            // If different, then not a prefix
    return 1;
}
```

verify

This part is to verify the correctness of a huffman-code using function prefix and to return Yes/No.

If the coded length is unequal to the Huffman-coded length, then it is surely not a Huffman coding; Another prespect, if the code contain one or more prefix, then it's not a prefix code, and surely not a Huffman code.

```
string verify() {
    int slen = 0;
    For i = 0 To n - 1 Step 1
        slen = slen + freq[i] * code[i].length();
    if (slen != len) then
        return "No";         // The coded length is different
    sort(code, code + n);
    For i = 1 To n - 1 Step 1
        if (prefix(i - 1, i)) then
            return "No";    // Contain a prefix
    return "Yes";            // If both not detected, then it's correct.
}
```

Chapter 3: Testing Results

input	output	note
in1.txt	out1.txt	same as sample
in2.txt	out2.txt	0-1 reversed; Two groups of data overlapping. Only lowercase letters
in3.txt	out3.txt	Check whether Huffman code has the same length and prefix error
in4.txt	out4.txt	Maximum N & M
in5.txt	out5.txt	Minimum N & M

Besides these cases, you can also test the correctness in PTA *Data Structures and Algorithms (English)* 7-9.

Chapter 4: Analysis and Comments

N: number of distinct characters

M: number of student submissions

Space Complexity

- The program use O(N) space to store the data and results.

Time Complexity

- preinput and readsol: both loop N times to input, so it's $O(N)$.
- huffman: $O(N)$, for each character shall get a code.
- prefix: loop code[a].length times, so the average time complexity is $O(N)$.

- verify: two loop each loop n times, the second loop contains one prefix, with a quick-sort on N objects. So the time complexity is $O(N^2 + NlogN) = O(N^2)$.
- main program: preinput + huffman + M * (readsol + verify), so the time complexity is $O(MN^2)$.

Appendix(readme.txt)

```
We suppose read the source file to get more comments.

We have generated some data (and their coresponding output) in data
folder.
You could use these data to check the program.

We recommend using TDM-GCC9.2.0 to compile,
the C++ standard shall be C++11 or above.
```