Project4 Huffman Codes

(Bonus)

Cao Yijia

Date: 2020-05-07

Chapter 1: Introduction

In this project, we are given some characters and the frequencies. Moreover, we are given some sets of codes. Our task is to judge whether each set of code is optimal or not. The optimal solution is not necessarily generated by Huffman algorithm. Any prefix code with code length being optimal is considered correct.

So, in this project I check whether the codes satisfying the two:

1. Length of weighted path is equal to that of Huffman code.

2. Prefix code.

If both are satisfied, the code is considered to be “correct” and output “Yes”, otherwise output “No”.

Input Specification:

```
N   //(2≤N≤63) number of distinct characters chosen from {'0'-'9','a'-'z','A'-'Z','_'}
c[1] f[1] c[2] f[2] ... c[N] f[N]   //c[i]:character, f[i](≤1000):frequency of c[i]
M   //(≤1000) number of student submissions(each consists of N lines)
c[i] code[i]
//c[i]:ith character, code[i]:an non-empty string of no more than 63 '0's and '1's
……
```

Output Specification:

For each test case, print in each line either "Yes" if the student's submission is correct, or "No" if not.

Chapter 2: Data Structure and Algorithm Specification

2.1 Main Data Structure

We mainly use the Huffman Tree and Min Heap to check the codes.

```
#define MAXN 64

int N; //number of distinct characters
int M; //number of student submissions
char ch[MAXN]; //character chosen from {'0'-'9','a'-'z','A'-'Z','_'}
int w[MAXN]; //the frequency of c[i]
int length; //length of weighted path of the optimal code

typedef struct HuffmanNode {
    char ch; //the character
```

```
    int weight; //the frequency of the character
    HuffmanNode *Left;
    HuffmanNode *Right;
} *HuffmanTree;

typedef struct Heap {
    HuffmanTree *data; //elements in the heap
    int size = 0; //size of the heap
    int capacity; //max size of the heap
} *MinHeap;
```

2.2 Algorithm Specification

2.2.1 int main( )

The pseudo code of the main function is showed below. And through this, we can also clearly see the main steps to solve the problem.

```
Input N
Build MinHeap H whose capacity is 2*N //Huffman Tree with N leaf nodes has 2 N-1 nodes
for (int i = 0; i < N; i++)
{
    Input ch[i] and w[i]  //character and its frequency
    Create a new HuffmanTree node T whose ch = ch[i] and weight = w[i]
    Insert T into the MinHeap H
}
Build Haffman Tree with these characters and frequencies
length := WPL(HT, 0)  //length of weighted path of the optimal code
Input M
for (int i = 0; i < M; i++)
{
    if (Check the student submission and it's correct)
    //prefix code with WPL equal to length
        Output "Yes"
    else
        Output "No"
}
return 0;
```

Then I will show some important functions used.

2.2.2 HuffmanTree Huffman (MinHeap H)

This function is to build a Huffman Tree with the given characters and their frequencies. The pseudo code is shown below:

```
HuffmanTree T = NULL;
num := H->size;
for (i = 1; i < num; i++) {
    Create a new node T
    //greedy
    Delete root from MinHeap and attach it to left_child of node
    Delete root from MinHeap H and attach it to right_child of node
    Weight of node is the sum of weights of its children
    Insert node T into MinHeap H
}
return T;
```

2.2.3 int WPL(HuffmanTree T, int depth)

This function is to calculate the weighted path length of the Huffman Tree. The pseudo code is shown below:

```
if (T is not NULL)
{
    if (T is a leaf node)
        return depth * T->weight;
    else
        return WPL(T->Left, depth + 1) + WPL(T->Right, depth + 1);
}
else  return 0;
```

2.2.4 int Check( )

This function is to check whether the student submission is correct or not. Only if the code is prefix code and its WPL (weighted path length) is equal to the WPL of the Huffman Tree built with characters and frequencies, the submission is correct and return 1. Otherwise return 0. The pseudo code is shown below:

```
int weight; //weight of the character
int flag = 0;  //flag to record whether it is a prefix code
char c, code[MAXN];

HuffmanTree T = CreateTree();
HuffmanTree pt = NULL;
for (i = 0; i < N; i++) {
    Input chreacter c and its code
    if (strlen(code) > N - 1)
    //code word length of the HuffmanTree with N leaf nodes is at most N-1
        return 0;
    find the correcponding index of c in ch[] and get its frequency weight f
rom w[]
```

```
    pt := T;//move the pointer pt to the root
    for (j = 0; code[j] != '\0'; j++) {
        if (code[j] == '0')
        {
            if (pt has no left child)
                pt->Left := CreateTree();
            pt := pt->Left;
        }
        else if (code[j] == '1')
        {
            if (pt has no right child)
                pt->Right := CreateTree();
            pt := pt->Right;
        }
        if (pt->weight != 0)
            flag = 1; //is prefix code
        if (code[j + 1] == '\0') //should be leaf node
            pt->weight := weight;
    }
}
if (is not prefix code)
    return 0;
if (its WPL is equal to length)
    return 1;  //correct!
else return 0;
```

Chapter 3: Testing Results

3.1 PTA Test

提交时间	状态	分数	题目	编译器	耗时	用户
2020/05/13 21:51:42	答案正确	30	7-9	C++ (g++)	59 ms	3180101226

测试点	提示	结果	耗时	内存
0	sample 有并列、多分支，有长度错、长度对但是前缀错；仅英文大写字符	答案正确	4 ms	424 KB
1	小写字母，01反、且2点对换；有2点重合	答案正确	4 ms	356 KB
2	几组编码不等长，都对；等长但前缀错误；code长度超过N	答案正确	4 ms	424 KB
3	最大N&M，code长度等于63	答案正确	59 ms	4240 KB
4	最小N&M	答案正确	4 ms	352 KB
5	编码的字符是双数个，而提交采用的是等长编码。卡仅判断叶结点和度的错误算法	答案正确	4 ms	424 KB
6	非Huffman编码，但是正确；没有停在叶子上	答案正确	4 ms	424 KB

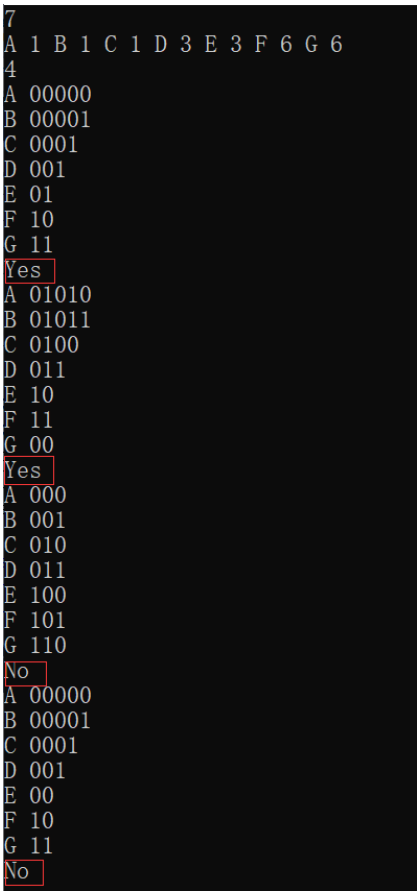
The implementation is able to pass [PTA - Huffman Codes](#).

To run successfully, change the head files as below, because stdafx.h can not be used on the website.

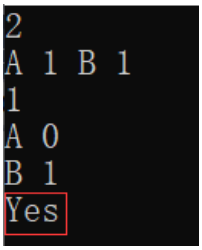
```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
```

3.2 Special Cases

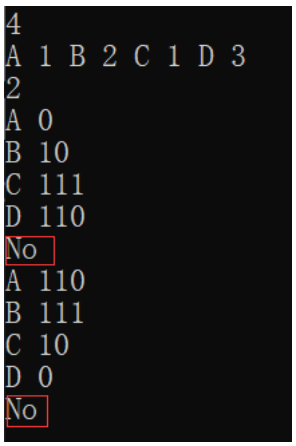
Case 1: sample (Pass!)



Case 2: N and M are both minimum (Pass!)



Case 3: the WPL is larger than Huffman code (Pass!)



Chapter 4: Analysis and Comments

4.1 Time Complexity

The two initial functions CreateTree() and CreateMinHeap(Max) both cost O(1) time. The function Insert(H, item) and DeleteMin(H) both cost O (log N) time. To build the Huffman Tree of N characters, Huffman(H) cost O (N log N) time. The part of the function WPL(T, depth) for the Huffman tree is O (log N) because the function is called recursively and needs to access the leaf node. To check M student submissions by Check(), each costs O(N * len) time.

Therefore the total time complexity of the program should be:

N * (O(1) + O (log N)) + O (N log N) + O(log N) + M * O(N * len)

= **O(N log N) + O(k * M * N)**, where k is the average length of the codes

4.2 Space Complexity

The project uses two arrays: ch[MAXN] and w[MAXN], and build a Min Heap whose capacity is 2*N. And Check() uses an array: code[MAXN] for the code input.

Therefore the space complexity is **O(N)**.

Appendix: Source Code

```
1.  #include "stdafx.h"
2.  #include <iostream>
3.  #include <string>
4.  #include <algorithm>
5.  using namespace std;
6.
7.  #define MAXN 64
```

```
8.
9.  int N; //number of distinct characters
10. int M; //number of student submissions
11. char ch[MAXN]; //character chosen from {'0'-'9','a'-'z','A'-'Z','_'}
12. int w[MAXN]; //the frequency of c[i]
13. int length; //length of weighted path of the optimal code
14.
15. typedef struct HuffmanNode {
16.     char ch; //the character
17.     int weight; //the frequency of the character
18.     HuffmanNode *Left;
19.     HuffmanNode *Right;
20. } *HuffmanTree;
21.
22. typedef struct Heap {
23.     HuffmanTree *data; //elements in the heap
24.     int size = 0; //size of the heap
25.     int capacity; //max size of the heap
26. } *MinHeap;
27.
28. HuffmanTree CreateTree(); //Create a new HuffmanTree node
29. MinHeap CreateMinHeap(int Max); //Create a new MinHeap
30. int Insert(MinHeap H, HuffmanTree item); //Insert a HuffmanTree node into th
    e MinHeap
31. HuffmanTree DeleteMin(MinHeap H); //Delete the root(the minimum item) from M
    inHeap
32. HuffmanTree Huffman(MinHeap H); //Build the Huffman Tree
33. int WPL(HuffmanTree T, int depth); //calculate the weighted path length
34. int Check(); //check if the student submission is correct
35.
36. int main()
37. {
38.     cin >> N;
39.     MinHeap H = CreateMinHeap(2 * N);//Build MinHeap
40.                                      //Huffman Tree with N leaf nodes has 2N
    -1 nodes
41.     for (int i = 0; i < N; i++)
42.     {
43.         HuffmanTree T = CreateTree();
44.         cin >> ch[i] >> w[i]; //character and its frequency
45.         T->ch = ch[i];
46.         T->weight = w[i];
47.         Insert(H, T); //insert it into the min heap
48.     }
```

```
49.     HuffmanTree HT = Huffman(H); //Build Haffman Tree
50.     length = WPL(HT, 0); //length of weighted path of the optimal code
51.     cin >> M;
52.     for (int i = 0; i < M; i++) //check each submission
53.     {
54.         if (Check())//the student submission is correct
55.             cout << "Yes" << endl;
56.         else
57.             cout << "No" << endl;
58.     }
59.
60.     system("pause");
61.     return 0;
62. }
63.
64. HuffmanTree CreateTree()
65. {
66.     HuffmanTree T = (HuffmanTree)malloc(sizeof(struct HuffmanNode));
67.     T->ch = '\0';  //empty
68.     T->weight = 0;
69.     T->Left = T->Right = NULL;
70.     return T;
71. }
72. MinHeap CreateMinHeap(int Max)
73. {
74.     MinHeap H = (MinHeap)malloc(sizeof(struct Heap));
75.     H->data = (HuffmanTree *)malloc((Max + 1) * sizeof(struct HuffmanNode));
76.     H->size = 0; //empty
77.     H->capacity = Max;
78.     return H;
79. }
80. int Insert(MinHeap H, HuffmanTree item)
81. {
82.     int i;
83.     if (H->size == H->capacity) //the MinHeap is full
84.         return 0; //fail
85.
86.     i = ++H->size; //the position of the last element in the back heap
87.     for (; i > 1 && H->data[i / 2]->weight > item->weight; i /= 2)
88.         H->data[i] = H->data[i / 2];
89.     H->data[i] = item; //insert item
90.
91.     return 1; //succeed
```

```
92. }
93. HuffmanTree DeleteMin(MinHeap H)
94. {
95.     int parent, child;
96.     HuffmanTree MinItem, temp;
97.     if (H->size == 0) //MinHeap is empty
98.         return NULL;
99.
100.     MinItem = H->data[1];  //minimum item
101.
102.     temp = H->data[H->size--];
103.     for (parent = 1; parent * 2 <= H->size; parent = child) {
104.         child = parent * 2;
105.         if ((child != H->size) && (H->data[child]->weight > H->data[child +
    1]->weight))
106.             child++;
107.         if (temp->weight > H->data[child]->weight)
108.             H->data[parent] = H->data[child];
109.         else
110.             break; //find suitable positon
111.     }
112.     H->data[parent] = temp;
113.
114.     return MinItem;
115. }
116. HuffmanTree Huffman(MinHeap H)
117. {
118.     int i, num;
119.     HuffmanTree T = NULL;
120.
121.     num = H->size;
122.     for (i = 1; i < num; i++) {
123.         T = CreateTree();  //create a new node
124.         /*greedy*/
125.         T->Left = DeleteMin(H);  //delete root from min heap and attach it
    to left_child of node
126.         T->Right = DeleteMin(H);  //delete root from min heap and attach it
     to right_child of node
127.         T->weight = T->Left->weight + T->Right->weight;  //weight of node i
    s the sum of weights of its children
128.         Insert(H, T);  //insert node into min heap
129.     }
130.     return T;
131. }
```

```
int WPL(HuffmanTree T, int depth)
{
    if (T) {
        if (!T->Left && !T->Right) //leaf node
            return depth * T->weight;
        else
            return WPL(T->Left, depth + 1) + WPL(T->Right, depth + 1);
    }
    else //T is NULL
        return 0;
}
int Check()
{
    int i, j, weight;
    int flag = 0;  //flag to record whether it is a prefix code
    char c, code[MAXN];

    HuffmanTree T = CreateTree();
    HuffmanTree pt = NULL;
    for (i = 0; i < N; i++) {
        cin >> c >> code;
        if (strlen(code) > N - 1)
            return 0; //code word length of the Huffman tree composed of N leaf nodes is at most N-1
        for (j = 0; c != ch[j]; j++); //find the correcponding index of character c
        weight = w[j]; //get the frequency of character c
        pt = T;
        for (j = 0; code[j] != '\0'; j++) {
            if (code[j] == '0')
            {
                if (!pt->Left)
                    pt->Left = CreateTree();
                pt = pt->Left;
            }
            else if (code[j] == '1')
            {
                if (!pt->Right)
                    pt->Right = CreateTree();
                pt = pt->Right;
            }
            if (!pt->weight)  flag = 1; //is prefix code
            if (code[j + 1] == '\0') //should be leaf node
                pt->weight = weight;
```

```
174.         }
175.     }
176.     if (!flag) //is not prefix code
177.         return 0;
178.     if (length == WPL(T, 0)) //its WPL is equal to length
179.         return 1;  //correct!
180.     else return 0;
181. }
```

Author List

曹一佳 3180101226

Declaration

We hereby declare that all the work done in this project titled "Huffman Codes" is of our independent effort as a group.

Signatures