Huffman Codes

Group 15

2020/05/13

LOGO

THE MAIN CONTENTS

第 1 部分

Introduction

Problem Description

- The input format in this project is including the following things A integer n, which means the number of characters. n characters and their frequency in pairs. A integer m, which means the number of hand-in answers from students. m groups of hand-in answers from students, each contains n pairs of character and its Huffman code.

Input format

- We will check whether the hand-in answer is correct or not and output a "Yes" or "No" to show the result of our program.

Output format

第 2 部分

Data Structure and Algorithm Specification

Huffman tree

```
typedef struct node
{
        char name;
        int weight, parent;
        int lchild, rchild;
        node():name(), weight(0), parent(0), lchild(0),rchild(0){}
}HTNode,*HuffmanTree;
```

- Huffman tree is a binary prefix tree which store the characters in its leaf nodes. The Huffman code of a character is corresponding string on the Huffman tree of the data

Greedy Algorithm

- Greedy algorithm works in optimizing problems, it choose the choice that can get most benefits. But this algorithm works if and only if the local optimum is equal to the global optimum

Min Heap

```
void Huffman ( PriorityQueue  heap[ ],  int  C )
{   consider the C characters as C single node binary trees,
     and initialize them into a min heap;
     for ( i = 1; i < C; i++ ) {
        create a new node;
        /* be greedy here */
        delete root from min heap and attach it to left_child of node;
        delete root from min heap and attach it to right_child of node;
        weight of node = sum of weights of its children;
        /* weight of a tree = sum of the frequencies of its leaves */
        insert node into min heap;
   }
}
```

- Min heap is a special tree-based data structure that the parent node is less than its non-empty children, but the siblings doesn't have any special relations. It can help us find the highest priority node.

pseudo-code of the whole program

```
Algorithm:Project 4: Huffman Codes
Input: The input format in the 1.1.2
Output: m results of the checking result "Yes" or "No"

read in characters and their frequency;
build_min_heap();
min_length=calculate_weight(min_heap H);
read in m;
for(i=0;i<m;i++)
{
    total_length=0;
    for(j=0;j<n;j++)
    {
        read in character and code;
        total_length+=frequency[character]*code.length();
    }
    if(total_length==min_length&&check_prefix(code))
    {
        printf("Yes\n");
    }
    else
    {
        printf("No\n");
    }
    return 0;
}
```

The most important two functions 1

Function 1:
calculate weight

```
function calculate_weight(H)
    result:=0
    while(true)
    {
        weight:=H.top()
        H.pop()
        if(H.size()==0)
            break
        weight+=H.top()
        H.pop()
        H.push(weight)
        result+=weight
    }
    return result
```

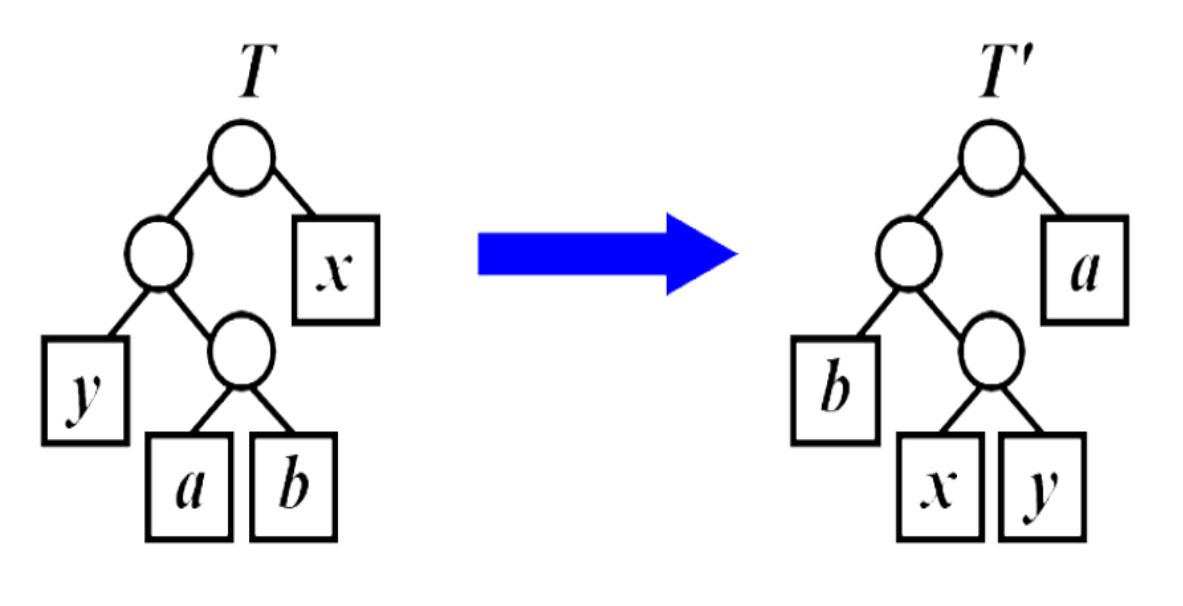
The most important two functions 2

```
function check_prefix(code)
    sort(code.begin(),code.end())
    for i=0 to code.size() do
        len:=code[i].size()
        for j=i+1 to code.size() do
            sub_prefix=code[j].substr(0,len)
            if sub_prefix==code[i]
                return false
    return true
```

Function 2:
check prefix

第 3 部分

Analysis and Comments



The Greedy choice property

- 【Lemma】Let C be an alphabet in which each character c in C has the frequency c.freq. Let x and y be two characters in C having the lowest frequencies. Then there must exist an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit.

The optimal substructure property

- Let C be a given alphabet with frequency c.freq defined for each character c in C. Let x and y be 2 characters in C with minimum frequency. . Let C' be the alphabet C with a new character z replacing x and y, and z.freq = x.freq + y.freq. Let T' be any tree representing an optimal prefix code for the alphabet C'. Then the tree T, obtained from T' by replacing the leaf node for z with an internal node having x and y as children, represents an optimal prefix code for the alphabet C.

Analysis of Min Heap

- The time complexity of the min-heap operation cost most in out program, we have serveral operations in the program

  Build min-heap: It is O(N) obviously to build a min heap because each insert cost O(logN)

  Calculate the total weight: In the while loop, each time we pop 2 elements from the top of the heap(delete min) and insert 1 element into the heap. Delete min from heap and insert into heap both need O(logN) each time, so each time we need O(3logN+c) time. And we need do the loop N times to calculate the total weight of the Huffman tree, so the time complexity is O(NlogN) .

- The space complexity of the min-heap is obviously O(N)

Analysis of algorithm complexity

- Time complexity: The whole program contains several parts:

  Read in the data:

  It needs O(2N+MN)=O(MN) time complexity

  Heap Operations:

  It needs O(NlogN) according to 4.1

  Check each group of answer:

  It has m rounds and each round need $O(N^2)$ because if the prefix checking according to 4.2, so it needs $O(MN^2)$ in total

  All in all : the time complexity is $O(MN^2)$

- Space complexity: The whole program contains several parts:

  Build the heap: O(N)

  Store the map from character to Huffman code: O(N)

  Store the result of Yes and No to output them together finally: O(M)

  So the time complexity is : O(M+N)

第 4 部分

Testing Results

04 Testing Results

```
input:
7
A 1 B 1 C 1 D 3 E 3 F 6 G 6
4
A 00000
B 00001
C 0001
D 001
E 01
F 10
G 11
A 01010
B 01011
C 0100
D 011
E 10
F 11
G 00
A 000
B 001
C 010
D 011
E 100
F 101
G 110
A 00000
B 00001
C 0001
D 001
E 00
F 10
G 11
```

Test 1: Sample Data

```
Output:
Yes
Yes
No
No
```

04 Testing Results

```
input:
7
9 1 B 1 C 1 D 3 e 3 f 6 _ 6
4
9 00000
B 00001
C 0001
D 001
e 01
f 10
_ 11
9 01010
B 01011
C 0100
D 011
e 10
f 11
_ 00
9 000
B 001
C 010
D 011
e 100
f 101
_ 110
9 00000
B 00001
C 0001
D 001
e 00
f 10
_ 11
```

Test 2: Have characters of digit, '_' and letters

```
Output:
Yes
Yes
No
No
```

04 Testing Results

Test 3: Have some not optimum condition

```
Input:
4
a 4 c 1 b 2 d 1
2
a 0
b 10
c 110
d 111
a 0
c 11000
d 10001
b 11111
Output:
Yes
No
```

Test 4: Have some code be prefix

```
Input:
7
A 1 B 1 C 1 D 3 E 3 F 6 G 6
1
A 00000
B 00001
C 0001
D 011
E 01 ---here is a prefix
F 10
G 11
Output:
No
```

04 Testing Results

Test 5: A smallest size of test data.

```
Input
2
a 1 b 2
2
a 0
b 1
a 1
b 10
Output:
Yes
No
```

Test 6: A largest size of test data

```
Input
63 characters with 50 hand-in answers
The test data is too big, you can find the data in the testdata.txt
Output
Yes
Yes
No
No
No
Yes
Yes
Yes
No
No
Yes
Yes
No
No
No
Yes
Yes
Yes
No
```

.....

谢谢您的观看与聆听

Group 15

2020/05/13

LOGO