

Huffman Codes

Group ***

Member : ***

Date: yy-mm-dd

Chapter 1: Introduction

Problem description:

Because the Huffman codes are not unique and some different set of code is not right. We need to judge the test cases from standard input whether is correct Huffman codes.

Purpose of this report:

We have three purposes as follows :

First, we will show what we had do by using charts and text analysis.

Second, it will be report to everyone that some ideas for improving our algorithm.

Third, we sincerely hope that you can correct our program in some places.

Background of the data structures and the algorithms:

Nothing. We overcame this problem with some knowledge which we had studied.

Chapter 2: Algorithm Specification

Data Structure:

```
struct node{
    int element;
    struct node* next;
    struct node* left;
    struct node* right;
};
Node_c=(char*)malloc(Node_number*(sizeof(char)));
Node_f=(int*)malloc(Node_number*(sizeof(int)));
```

```
char** code=(char**)malloc(s_number*Node_number*(sizeof(char*)));
```

Algorithm Specification:

```
Int main()                                /*Main Function*/  
{  
    If(!Pre(code,length)&&Wpl(code,length,Node_f))  
        Printf("Yes\n");  
    Else  
        Printf("No\n");  
}
```

```
Compare(char* a,char*b)  
{  
    Int length=min(strlen(a),strlen(b));  
    char* p=malloc(length*sizeof(char));  
    Char*q=malloc(length*sizeof(char));  
    stringcopy(p,a);  
    stringcopy(q,b);  
    Return strcmp(p,q);  
}
```

```
Int Pre(char** code, int length)  
{  
    For(i=0;i<length-1;i++)  
    For(j=i+1;j<length;j++)  
    If(Compare(code[i],code[j])==0)  
    Return 1  
  
    Return 0;  
}
```

```
Gre createtree(int* fnode,int length)  
{  
    Gre head;  
    Gre p;  
    Head=p;  
    For(i=0;i<length;i++)  
    {  
        P->element=fnode[i];  
        P=p->next;  
    }  
    While(head->next->next!=NULL)  
    {  
        Gre q=findmin(head,head->next->next);
```

```

If(q==head)
{
Gre new=malloc(sizeof(struct node));
new->left=head;
New->right=head->next;
New->next=head->next->next;
New->element=head->element+head->next->element;
Head=new;
}
Else{
Gre new=malloc(sizeof(struct node));
new->left=head->next;
New->right=head->next->next;
New->next=head->next->next->next;
New->element=head->next->element+head->next->next->element;
Head->next=new;
}
}
Return head;
}

```

Int minwpl(Gre head,int height)

```

{
If(head!=NULL)
If(head->left==NULL)
Return height*head->element
Else
Return minwpl(head->left,height+1)+minwpl(head->right,height+1);
}

```

Int WPL(code,length,Node_f)

```

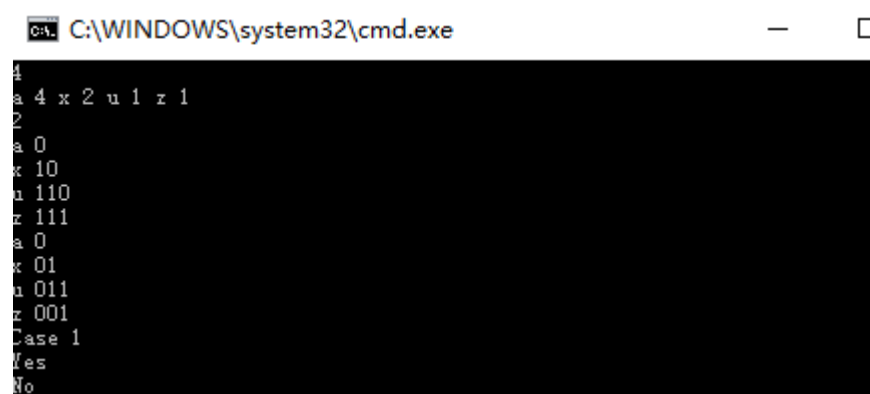
{
Wp1=0;
Wp2=0;
Create(fnode,Node_f);//create new array fnode to copy Node_f
For(i=0;i<length;i++)
Wp1=Wp1+fnode[i]*strlen(code[i]);
Height=0;
AsOrder(fnode);
Gre head=Createtree(fnode,length);
Wp2=minwpl(head,height);
If(wp1==wp2)
Return 1;
Else return 0;
}

```

```
}
```

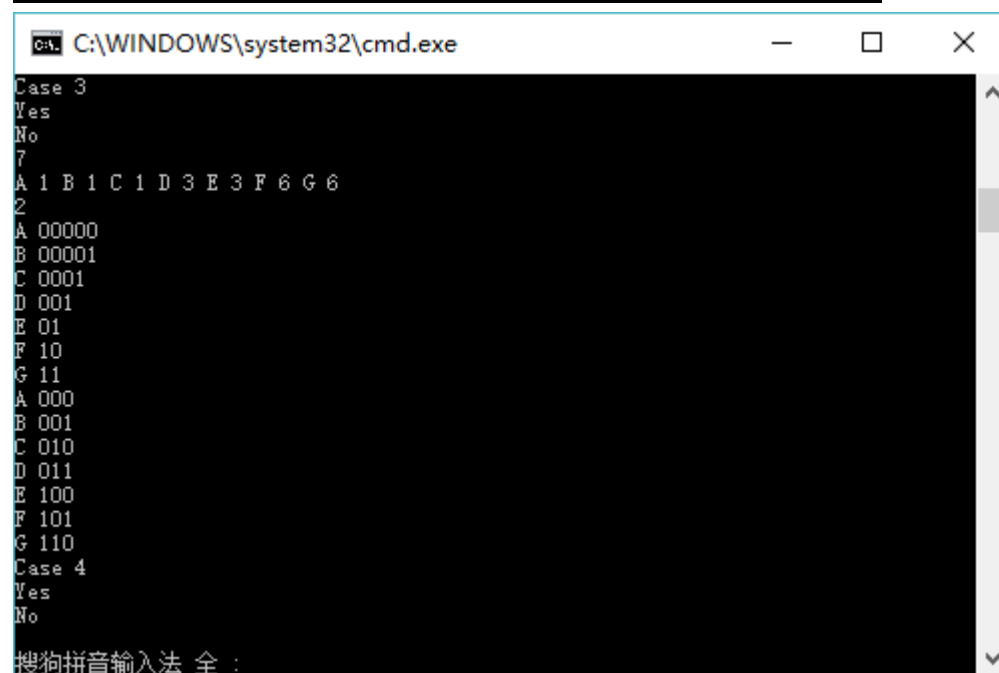
Chapter 3: Testing Results

We got the right answer to the question (As figure 1) , and after increase the number of inputs is also correct (As figure 2) , However, in the process of running the PTA test, it is found that the error of the result is correct except the different requirements of the output format PTA and the experimental requirements. But our program didn't get a full mark, After thinking that we use too many arrays, and the size is too large resulting in memory overrun, This means that the need to improve. The algorithm needs to be optimized.



```
C:\WINDOWS\system32\cmd.exe
4
a 4 x 2 u 1 z 1
2
a 0
x 10
u 110
z 111
a 0
x 01
u 011
z 001
Case 1
Yes
No
```

Figure 1



```
C:\WINDOWS\system32\cmd.exe
Case 3
Yes
No
7
A 1 B 1 C 1 D 3 E 3 F 6 G 6
2
A 00000
B 00001
C 0001
D 001
E 01
F 10
G 11
A 000
B 001
C 010
D 011
E 100
F 101
G 110
Case 4
Yes
No
搜狗拼音输入法 全 :
```

Figure 2

Chapter 4: Analysis and Comments

Analysis : We mainly use Greedy algorithm and build a structure to build the linked list and Huffman code tree. Improve the reuse of the structure. And the use of a number of sub functions to solve the problem greatly improve the readability of the program. For space complexity, the main overhead lies in the recursive algorithm and data storage are $O(n)$ and $O(n)$ so the total complexity of $O(n)$. For this program, the time complexity is mainly greedy algorithm ($O(n)$) and recursive algorithm ($O(n \cdot \log n)$). Therefore, the total time complexity is about $O(n)$

Comments : At the beginning of the experiment, we do not use the greedy algorithm and directly to build a simple Huffman tree that leads to an erroneous result. After the teacher's guidance, the greedy algorithm is used to build the Huffman tree and the recursive algorithm is used to calculate the WPL value of each probability. Simple and intuitive and effective correction of results. In addition, in comparison to whether the prefix, choose a shorter one as a basis for comparison can greatly reduce the workload

Appendix: Source Code

This part is packed with our engineering document, please view yourself.

References:

we just use the knowledge from book , so we are basically independent of our own, without reference to other people's code.

Declaration:

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

Signatures

*** **