

Experiment topic: Data compression algorithm based on Huffman tree

Experiment purpose :

1. Master the construction algorithm of Huffman tree.
2. Master the construction algorithm of Huffman coding.

Experiment content:

Problem Description

Input a string of strings, build a corresponding Huffman tree according to the frequency of characters in the given string, and construct a Huffman encoding table. On this basis, the compressed file can be compressed (that is, encoded), and at the same time Decompress (i.e. decode) the compressed binary encoded file.

input requirements

Multiple sets of data, each set of data is 1 row, which is a string (only 26 lowercase letters are considered). When the input string is "0", the input ends.

output requirements

Each set of data outputs $2n+3$ rows (n is the number of character categories in the input string). The first row counts the occurrence frequency of characters (only the existing characters are output, the format is: character: frequency), each two groups of characters are separated by a space, and the characters are arranged in ascending order of ASCII codes. Lines 2 to $2n$ are the final state of the storage structure of the Huffman tree (such as Table 5.2 (b) on page 139 of the main textbook, the data in a line is separated by spaces). Line $2n+1$ is the Huffman encoding of each character (only the existing characters are output, the format is: character: encoding), and each group of characters is separated by a space. The characters are arranged in ascending order of the ASCII code. The $2n+2$ row is the encoded string, and the $2n+3$ row is the decoded string (same as the input string).

CODE

HuffmanAPP.cpp

```
#include "StdAfx.h"
#include "Huffman.h"
```

```
void Count(char str[], int count[])
{
    int i;
    for(i=1;i<= 26;i++)
    {
        count[i] = 0;
    }
    for(i=0;i<strlen(str);i++)
    {
        count[str[i]-'a'+1]++;
    }
}
```

```
void Select(HuffmanTree &HT, int n, int &a, int &b)
{
    unsigned int m1, m2;
    m1 = m2 = 32767;
    for (int i = 1; i <= n; i++)
```

```

{
if (HT[i].parent == 0 && HT[i].weight < m1)
{
m2 = m1;
b = a;
m1 = HT[i].weight;
a = i;
}
else if (HT[i].parent == 0 && HT[i].weight < m2)
{
m2 = HT[i].weight;
b = i;
}
}
}

```

```

void CreateHuffmanTree(HuffmanTree &HT,int n,int count[])
{ // Construct Huffman tree HT
    int i,j,s1,s2;
    if(n<=1)
        return;
    int m=2*n-1;
    HT=new HTNode[m+1]; //0 good units are used, so m+1 units need to be allocated dynamically, HT[m]
means the root node
    for(i=1;i<=m;++i)
    {
        HT[i].parent=0; //Initialize the subscripts of the parents, left child and right child in units 1~m to 0
        HT[i].lchild=0;
        HT[i].rchild=0;
    }
    for(i=1,j=1;i<=n,j<=26;j++)
    {
        if(count[j]>0)
        {
            HT[i].weight=count[j];
            i++;
        }
    }
    //The initialization work is over, and the Huffman tree is created
    for(i=n+1;i<=m;++i)
    { //Create a Huffman tree by selecting, deleting, and merging n-1 times
        Select(HT,i-1,s1,s2); //Select two nodes in HT[k](1<=k<=i-1) whose parent fields are 0 and have the
smallest weight, and return Their sequence numbers s1 and s2 in HT
        HT[s1].parent=i;
        HT[s2].parent=i; //Get a new node i, delete s1 and s2 from the forest, change the parent fields of s1
and s2 from 0 to i
        HT[i].lchild=s1;
        HT[i].rchild=s2; //s1 and s2 are the left and right children of i respectively
        HT[i].weight=HT[s1].weight+HT[s2].weight; //The weight of i is the sum of the weights of the left and
right children
    }
}

```

```

void CreatHuffmanCode(HuffmanTree HT,HuffmanCode &HC,int n)
{ // Find the Huffman encoding of each character from the leaf to the heel, and store it in the encoding table HC
    HC=new char*[n+1]; //Allocate encoding table space for storing n character encodings
    char* cd=new char[n]; //Allocate dynamic array space for temporary storage of each character code

```

```

cd[n-1]='\0'; //encoding terminator
for(int i=1;i<=n;++i) //find Huffman code character by character
{
    int start=n-1; //start points to the end at the beginning, that is, the position of the code end character
    int c=i;
    int f=HT[i].parent; //f points to the parent node of node c
    while(f!=0) //backtracking from the leaf node to the root node
    {
        --start; //Backtrack once and point forward to a position
        if(HT[f].lchild==c)
            cd[start]='0'; //node c is the left child of f, then generate code 0
        else
            cd[start]='1'; //node c is the right child of f, then generate code 1
        c=f;
        f=HT[f].parent; //Continue to backtrack upwards to find the code of the i-th character
    }
    HC[i]=new char[n-start]; //Allocate space for the i-th character code
    strcpy(HC[i],&cd[start]); //Copy the obtained code from the temporary space cdf to the current line of
HC
}
delete cd; // release temporary space
}

```

HuffmanAPP.cpp

```

#include "stdafx.h"
#include "Huffman.h"
#define N 10005

int main()
{
    int i,j,n,length,r;
    char str[N];
    int count[50];
    while(cin>>str)
    {
        length=strlen(str);
        if(length==1&&str[0]=='0')
            break;
        n=0;r=0;
        Count(str, count);
        for (i=97;i<123;i++)
        {
            if(count[i - 96]>0)
            {
                n++;
                printf("%c:%d ",i,count[i - 96]);
                if(r==0)
                    r=i;
            }
        }
        cout<<endl;
        HuffmanTree HT;
        CreateHuffmanTree(HT,n,count);
        for(i=1;i<=2*n-1;i++)
        {
            cout<<i<<" "<<HT[i].weight<<" "<<HT[i].parent<<" "<<HT[i].lchild<<" "<<HT[i].rchild<<endl;
        }
        HuffmanCode HC;
    }
}

```

```

        CreateHuffmanCode(HT,HC,n);
        j=0;
for(i=97;i<123;i++)
    {
        if(count[i-96]>0)
        {
            ++j;
            printf("%c:%s ",i,HC[j]);

        }
    }
cout<<endl;
for (i=0;i<length;i++)
    {
        cout<<HC[str[i]-'a'+1-(r-97)];
    }
cout<<endl;
for (i=0;i<length;i++)
    cout<<str[i];
cout<<endl;
    }
    return 0;
}

```

StdAfx.cpp

```

// stdafx.cpp : source file that includes just the standard includes
//    Huffman.pch will be the pre-compiled header
//    stdafx.obj will contain the pre-compiled type information

```

```

#include "stdafx.h"

```

```

// TODO: reference any additional headers you need in STDAFX.H
// and not in this file

```

Huffman.h

```

// Storage representation of the Huffman tree

```

```

typedef struct

```

```

{
    int weight; //The weight of the node
    int parent,lchild,rchild; //The parent of the node, the left child, the subscript of the right child
}HTNode,*HuffmanTree; //Dynamically allocate an array to store the Huffman tree

```

```

typedef char** HuffmanCode; //Dynamically allocate an array to store the Huffman code table

```

```

void Select(HuffmanTree &HT, int n, int &a, int &b);

```

```

void CreateHuffmanTree(HuffmanTree &HT,int n,int count[]); //construct Huffman tree HT

```

```

void CreatHuffmanCode(HuffmanTree HT,HuffmanCode &HC,int n); // Find the Huffman code of each character
from the leaf to the heel, and store it in the code table HC

```

```

void Count(char str[], int count[]); //Statistics of character frequency

```

StdAfx.h

```

// stdafx.h : include file for standard system include files,

```

```

// or project specific include files that are used frequently, but

```

```

// are changed infrequently

```

```

//#if !defined(AFX_STDAFX_H__5F113B9E_6030_49A6_A08D_2F75E1050285__INCLUDED_)

```

```

#define AFX_STDAFX_H__5F113B9E_6030_49A6_A08D_2F75E1050285__INCLUDED_

```

```

#if _MSC_VER > 1000

```

```

#pragma once

```

```

#endif // _MSC_VER > 1000

```

```

#define WIN32_LEAN_AND_MEAN // Exclude rarely-used stuff from Windows headers

```

```

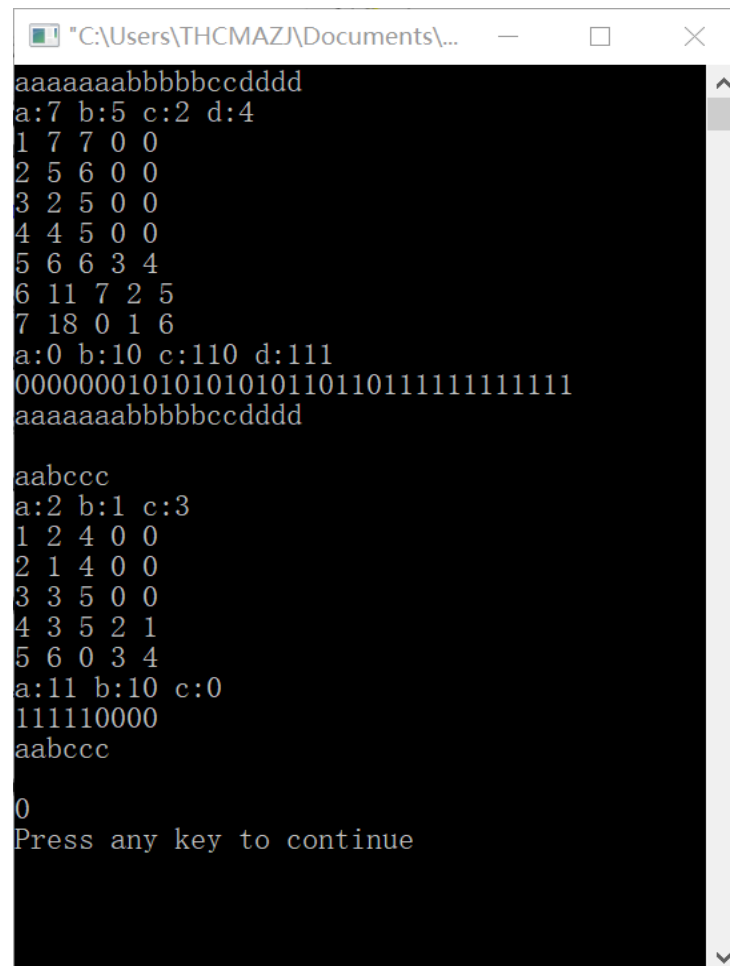
#include <stdio.h>
#include <string.h>
#include <iostream>
using namespace std;

// TODO: reference additional headers your program requires here
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__5F113B9E_6030_49A6_A08D_2F75E1050285__INCLUDED_)

```

Test Results:



The screenshot shows a Windows command prompt window with the title bar "C:\Users\THCMAZJ\Documents\...". The output of the program is as follows:

```

aaaaaaabbbbbccdddd
a:7 b:5 c:2 d:4
1 7 7 0 0
2 5 6 0 0
3 2 5 0 0
4 4 5 0 0
5 6 6 3 4
6 11 7 2 5
7 18 0 1 6
a:0 b:10 c:110 d:111
00000001010101011011011111111111
aaaaaaabbbbbccdddd

aabccc
a:2 b:1 c:3
1 2 4 0 0
2 1 4 0 0
3 3 5 0 0
4 3 5 2 1
5 6 0 3 4
a:11 b:10 c:0
111110000
aabccc

0
Press any key to continue

```