数据结构实验报告

<table>
<tr><td>学号-姓名</td><td>桑龙龙-20030540015</td><td>实验时间</td><td>2020 年 11 月 14 日</td></tr>
<tr><td>诚信声明</td><td colspan="3">本实验及实验报告所写内容为本人所作 桑龙龙</td></tr>
<tr><td>实验题目</td><td colspan="3">二叉树的运算与应用
题目一、二叉树的遍历运算
题目二、哈夫曼编/译码器</td></tr>
<tr><td>实验过程中遇到的主要问题</td><td colspan="3">无</td></tr>
<tr><td>实验小结</td><td colspan="3">本次实验进行了二叉树的构建、前中后序遍历，以及 huffman 树的应用，对树的认识更加深刻。</td></tr>
<tr><td>数据结构
（自定义数据类型）</td><td colspan="3">

```
struct huffman_node {
    /* huffman 树的节点 */
    char v;
    /* 关键字 v，只有叶子节点的 v 才有意义 */
    float f;
    /* 关键字 v 在文本中出现的频率 */
    huffman_node* next[2];
    /* 左右儿子 */
    huffman_node( int _v, float _f )
    {
        next[0] = NULL;
        next[1] = NULL;
        f   = _f;
        v   = _v;
    }
    huffman_node( huffman_node* a, huffman_node* b )
    {
        f   = a->f + b->f;
        next[0] = a;
        next[1] = b;
    }
};
```

</td></tr>
</table>

主要算法
（或算法说明）

```
/* 二叉树的遍历运算 */
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <algorithm>
using namespace std;
struct node {
    node    * left;
    node    * right;
    char    val;
    node( int _v ) : left( NULL ), right( NULL ), val( _v ){}
};

void create( node* &root )
{
    /* 创建二叉树 */
    char v = getchar();
    if ( v == '.' )
        root = NULL;
    else{
        root = new node( v );
        create( root->left );
        create( root->right );
    }
}


void preorder( node* root )
{
    /* 前序遍历 */
    if ( !root )
        return;
    putchar( root->val );
    preorder( root->left );
    preorder( root->right );
}

void inorder( node* root )
{
    /* 中序遍历 */
    if ( !root )
        return;
    inorder( root->left );
    putchar( root->val );
```

```
46.     inorder( root->right );
47. }
48. 
49. 
50. void postorder( node* root )
51. {
52.     /* 后序遍历 */
53.     if ( !root )
54.         return;
55.     postorder( root->left );
56.     postorder( root->right );
57.     putchar( root->val );
58. }
59. 
60. 
61. int main()
62. {
63.     node* root = NULL;
64.     create( root );
65.     printf( "pre order: " );
66.     preorder( root );
67.     putchar( '\n' );
68. 
69.     printf( "in order: " );
70.     inorder( root );
71.     putchar( '\n' );
72. 
73.     printf( "post order: " );
74. 
75.     postorder( root );
76.     putchar( '\n' );
77.     return(0);
78. }
```

```
1.  /* 哈夫曼编/译码器 */
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.  #include <iostream>
5.  #include <algorithm>
6.  #include <queue>
7.  #include <vector>
8.  #include <string.h>
9.  /* https://paste.ubuntu.com/p/BY2Rv3Rwv3/ */
10. using namespace std;
11. /*
12.  * 输入：
13.  * 第一行一个 n，表示关键字个数
14.  * 后序 n 行，每行一个字符 c 和一个频率
15.  * 之后一行文本，需要我们来解析
16.  * 输出：
17.  * n 行一个字符，以及其对应的 huffman 码
18.  * 之后会输出文本的经过 Huffman 编码后的文本
19.  * 之后一行会输出经过 decode 后的原文本
20.  */
21. /*
22.  * 样例 1：
23.  * 26
24.  * a 0.830000
25.  * b 0.860000
26.  * c 0.770000
27.  * d 0.150000
28.  * e 0.930000
29.  * f 0.350000
30.  * g 0.860000
31.  * h 0.920000
32.  * i 0.490000
33.  * j 0.210000
34.  * k 0.620000
35.  * l 0.270000
36.  * m 0.900000
37.  * n 0.590000
38.  * o 0.630000
39.  * p 0.260000
40.  * q 0.400000
41.  * r 0.260000
42.  * s 0.720000
43.  * t 0.360000
44.  * u 0.110000
45.  * v 0.680000
```

```
 * w 0.670000
 * x 0.290000
 * y 0.820000
 * z 0.300000
 * abcdefghigkmnopqrstuvwxyzaaaaaaa
 */
using p = pair<char, float>;
p keyword[128];
/* 用来记录输入关键字及其频率 */
char code[128][9], path[9], after_encode[10000];
/*
 * code 用来存储 ascii 码对应的的 huffman 码
 * after_encode 用来存储经过编码后的文本的 huffman 码
 */
int n, pos;
struct huffman_node {
    /* huffman 树的节点 */
    char v;
    /* 关键字 v，只有叶子节点的 v 才有意义 */
    float f;
    /* 关键字 v 在文本中出现的频率 */
    huffman_node* next[2];
    /* 左右儿子 */
    huffman_node( int _v, float _f )
    {
        next[0] = NULL;
        next[1] = NULL;
        f   = _f;
        v   = _v;
    }


    huffman_node( huffman_node* a, huffman_node* b )
    {
        f   = a->f + b->f;
        next[0] = a;
        next[1] = b;
    }
}
;
void getcode( huffman_node* root, int step )
{
    /*
     * 解析每一个关键字的 huffman 码
     * 将其存储起来，方便 encode
```

```
91.      */
92.     if ( !root->next[0] && !root->next[1] )
93.     {
94.         path[step] = 0;
95.         memcpy( code[root->v], path, step );
96.         return;
97.     }
98.     path[step] = '0';
99.     getcode( root->next[0], step + 1 );
100.    path[step] = '1';
101.    getcode( root->next[1], step + 1 );
102. }
103.
104.
105. huffman_node* build_huffman()
106. {
107.    /* 构建 huffman 树 */
108.    huffman_node    * a;
109.    huffman_node    * b;
110.    huffman_node    * c;
111.    float       f;
112.    char        v;
113.    auto        cmp = [] (const huffman_node * a, const huffman_node * b) {
114.        return(a->f > b->f);
115.    }
116.    ;
117.    priority_queue<huffman_node*, vector<huffman_node*>, decltype( cmp )> Q( cmp
    );
118.    /* 利用堆来加快 huffman 树的构造 */
119.    for ( int i = 0; i < n; i++ )
120.    {
121.        scanf( "%c %f\n", &keyword[i].first, &keyword[i].second );
122.        a = new huffman_node( keyword[i].first, keyword[i].second );
123.        Q.push( a );
124.    }
125.    while ( Q.size() > 1 )
126.    {
127.        a   = Q.top(), Q.pop();
128.        b   = Q.top(), Q.pop();
129.        Q.push( new huffman_node( a, b ) );
130.    }
131.    getcode( Q.top(), 0 );
132.    return(Q.top() );
133. }
134.
```

```
135.
136. void encode( huffman_node* root )
137. {
138.     /* encode 输入的文本文件 */
139.     char v;
140.     while ( (v = getchar() ) != EOF )
141.     {
142.         strcat( after_encode, code[v] );
143.     }
144. }
145.
146.
147. void decode( huffman_node* root )
148. {
149.     /* 对 huffman 码进行解析，恢复为原来的句子 */
150.     if ( !root->next[0] && !root->next[1] )
151.     {
152.         putchar( root->v );
153.         return;
154.     }
155.     decode( root->next[after_encode[pos++] - '0'] );
156. }
157.
158.
159. int main()
160. {
161.     scanf( "%d\n", &n );
162.     if ( n <= 1 )
163.     {
164.         /* 如果关键字只有一个就没有必要编码 */
165.         printf( "this is no need to encoding\n" );
166.         return(0);
167.     }
168.     huffman_node*root = build_huffman();
169.     for ( int i = 0; i < n; i++ )
170.     {
171.         /* 输出每个关键字对应的 huffman 码 */
172.         printf( "%c 's huffman code is: %s\n", keyword[i].first, code[keyword[i]
    .first] );
173.     }
174.     encode( root );
175.     /* 编码 */
176.     printf( "after_encode %s\n", after_encode );
177.     printf( "now will decode:\n" );
178.     while ( after_encode[pos] != 0 )
```

```
        decode( root );
    /* 解码 */
    return(0);
}
```