中山大学计算机学院本科生实验报告

（2021 学年第 1 学期）

课程名称：Data structures and algorithms　　任课教师：张子臻

年级	20 级	专业（方向）	软件工程
学号	20337270	姓名	钟海财
电话	13397996670	Email	2940599563@qq.com
开始日期	2021/11/25	完成日期	2021/11/27

1. 实验题目

1）完全二叉树-最近公共祖先

2）Huffman coding

2.实验目的

1）完全二叉树-最近公共祖先: 给出一颗完全二叉树中的两个顶点的索引，求出其最近的公共祖先节点。

2）Huffman coding ： 给出一堆数据和这些数据对应的频数，求出其哈夫曼编码的位数和（B（T）=∑f（c）dT（c））。

3.程序设计

1）完全二叉树-最近公共祖先

设计思路

由于在目标完全二叉树里，任意节点 k 的儿子节点为 2*k 和 2*k+1，所以任意节点 t 的父亲节点为 t/2。

所以不断对两个节点 x，y 求其较大者的儿子节点，再用求得的儿子节点与较小的节点继续进行相同操作，直到 x==y，便找到了它们的最近公共祖先，所以可以得到递归函数：

```
int common(int x,int y){
    if(x>y) return common(x/2,y);  //求较大者的儿子节点与较小者的最近公共祖先
    if(x<y) return common(x,y/2);
    return x;  //x==y 时便是它们的最近公共祖先
}
```

代码

```
#include<iostream>
using namespace std;
int common(int x,int y){  //求 x 与 y 的最近公共祖先
    if(x>y) return common(x/2,y);  //求较大者的儿子节点与较小者的最近公共祖先
    if(x<y) return common(x,y/2);
    return x;  //x==y 时便是它们的最近公共祖先
}
int main(){
    int num;
     cin>>num;
      for(int i = 0 ; i < num ; ++i){
         int a,b;
         cin>>a>>b;
         cout<<common(a,b)<<endl;
      }
     return 0;
}
```

2）Huffman coding

设计思路

由哈夫曼编码的生成方式可知，哈夫曼编码的位数即为对应的叶子节点在哈夫曼树中的层数，则所求的位数和 = Σ层数*频数。

由哈夫曼树的构造方式可知，初始时每个节点的层数都为 0，每将最小的两

个节点相加生成一个大的父亲节点时（该父亲节点的值为两个儿子节点值得和），两个节点所在子树的所有节点层数都+1。

于是可将层数*频数变为层数个频数相加，可知每次生成父亲节点时，都加上这两个节点子树上所有叶子节点的和，而由哈夫曼树的特点可知，这两个节点子树上所有叶子节点代表的频数和 = 该父亲节点的值，所以要求目标位数和，可求每次生成的父亲节点值的和。

初始时 sum=0，由于每次用最小的两个未生成过父亲节点的节点生成一个父亲节点，所以我使用 vector 来记录节点的值，每次生成一个父亲节点，都将该父亲节点 push_back 进 vector 里，sum+=该父亲节点的值，再对 vector 进行局部排序（由于每次用最小两个节点合成父亲节点，所以对 vector 里后部分暂未生成过父亲节点的所有节点进行冒泡排序，将最小的两个放到前面，作为下一次生成父亲节点的两个儿子节点）。如此循环，进行 n-1 次，便求出位数和 sum 并输出。

代码

```
#include<iostream>
using namespace std;
#include<vector>
void swap(int&a,int &b){
    int t = a;
    a =b ;
    b =t;
}
void _sort(vector<int> &node,int k){
     for(int i=0; i<2 ; ++i){//只用排 2 次
/从后往前的冒泡排序,将 k+2 之后的最小两个放在 k+2 和 k+3 位置
         for(int j = node.size()-1 ; j >= k+3+i ; --j  ){
           if(node[j]< node[j-1]){
               swap(node[j], node[j-1]);
             }
        }
```

```
    }
}
int main(){
    int n;
    cin>>n;
    char c[n];//记录字符
    vector<int>a;//频数
    for(int i = 0 ; i < n ; ++i){
        cin>>c[i];//字符
        int t;
        cin>>t;
        a.push_back(t);//频树
    }
    for (int i=1;i<n;i++){//对频数进行排序
        int key = a[i];
        int j=i-1;
        for(; j>=0 && a[j]>key  ; --j ) {
            a[j+1] = a[j];
            }
        a[j+1] = key;
    }
    int sum = 0;//初始位数和为 0
    int t = 0;//记录已经遍历过的节点数，只要遍历 n-1 次
    while(t<n-1){
        int newnode = a[2*t]+a[2*t+1];
        //由于每次用最小两个节点合成父亲节点，所以 2*t 记录将要遍历的最小节点的位置
        sum+=newnode;//位数和加上合成的父亲节点
        a.push_back(newnode);//将合成的父亲节点加入序列中
        _sort(a,2*t);
        //对 a 得 2*t+2 之和的节点进行排序，只需将最小的两个节点放到 2*t+2 和 2*t+3 的位置
        ++t;
    }
    cout<<sum<<endl;//输出位数和
    return 0 ;
}
```

4.程序运行与测试

1）完全二叉树-最近公共祖先

测试输入 1

```
2
10 4
7 13
```

测试输出 1

```
2
3

```

测试输入 2

```
10
3 34
42 45
23 78
31 126
22 133
785 23
12 242
123 5858
26 35
712 72
```

测试输出 2

```
1
5
2
31
2
1
3
1
1
2

```

测试输入 3

```
10
501 1
501 2
501 3
501 4
501 5
501 6
501 7
501 8
501 9
501 10
```

测试输出 3

```
1
1
3
1
1
3
7
1
1
1
```

2）Huffman coding

测试输入 1

```
5
0 5
1 4
2 6
3 2
4 3
```

测试输出 1

45

测试输入 2

```
20
1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
0 1
! 1
@ 1
# 1
$ 1
% 1
^ 1
& 1
* 1
( 1
) 1
```

测试输出 2

```
88
```

测试输入 3

```
26
a 24
b 4265
c 3546
d 535
e 567
f 2111
g 4358
h 855
i 9237
j 2424
k 2314
l 4856
m 875
n 1058
```

```
o 872
p 4235
q 21
r 549
s 753
t 8576
u 2451
v 435
w 1590
x 3875
y 999
z 4442
```

测试输出 3

271660

5.实验总结与心得

本次实验在解决 2）Huffman coding，开始时我使用构建哈夫曼树的方法，每次合成一个父亲节点（将其 push_back 插入到最后）都进行排序，发现使用稳定的排序如冒泡排序和插入排序时答案正确，而使用希尔排序，选择排序等不稳定的排序则答案错误。

这时想到哈夫曼树中更小的节点层数要更高，所以在两个节点的值相同时，层数更高的那个节点在剩余序列中的位置要更靠后，这样在最后生成的树中的层数才能更高，而层数更高的节点在我的序列中在后面（是通过 push_back 插入到最后的），所以排序需要稳定的排序。进一步分析后，每次生成一个父亲节点只需把后面最小的两个节点排序到前面。所以有如下代码：

```
#include<iostream>
using namespace std;
#include<vector>
struct Node{
    int d;
    char data;
    int nums;
```

```
    Node* lf;
    Node* rf;
     Node(){
         d = 0;
         nums = 0;
         data = ' ';
         lf = NULL;
         rf = NULL;
     }
};
void add_d(Node *node){ //增加叶子节点的层数
    if(node ==NULL) return ;
    if(node->lf==NULL && node->rf==NULL) node->d += 1;
    add_d(node->lf);
    add_d(node->rf);
}
void add(vector<Node*>&node,int k){  //将 k 和 k+1 合成一个父亲节点
    Node * p = new Node();
    p->nums = node[k]->nums+node[k+1]->nums;
    p->lf = node[k];
    p->rf = node[k+1];
    add_d(p);
    node.push_back(p);
}
void _swap(Node*&a,Node*&b){
    Node* t = a;
    a =b ;
    b =t;
}
void sort(vector<Node*> &node){ //排序要具有稳定性，防止前面已经遍历过的顺序打乱
     for(int i=0; i<node.size()-1 ; ++i){ //冒泡排序 or 插入排序
         for(int j = 0 ; j < node.size()-1-i ; ++j  ){
           if(node[j]->nums > node[j+1]->nums){
               _swap(node[j], node[j+1]);
             }
         }
   }
}
void _sort(vector<Node*> &node,int k){
     for(int i=0; i<2 ; ++i){/
/从后往前的冒泡排序,k+2 之后的最小两个放在 k+2 和 k+3 位置
         for(int j = node.size()-1-i ; j >= k+3+i ; --j  ){
           if(node[j]->nums < node[j-1]->nums){
               _swap(node[j], node[j-1]);
             }
         }
```

```
    }
}

int sum(Node*node){  //求出所有叶子节点的频数*层数之和
    if(node ==NULL ) return 0;
    if(node->lf==NULL && node->rf==NULL) return (node->d)*(node->nums);
    return sum(node->lf)+sum(node->rf);
}
int main(){
    int n;
    cin>>n;
    char c[n];
    int a[n];
    for(int i = 0 ; i < n ; ++i){
        cin>>c[i];//字符值
        cin>>a[i];//个数，权重
    }
    for (int i=1;i<n;i++){
        int key = a[i];
        char t = c[i];
        int j;
        for(j = i-1; j>=0 && a[j]>key  ; --j ) {
            a[j+1] = a[j];
            c[j+1] = c[j];
            }
        a[j+1] = key;
        c[j+1] = t ;
    }

    vector<Node*> node;
    for(int i = 0 ; i < n ;++i){
        Node * p = new Node();
        p->data = c[i];
        p->nums = a[i];
        node.push_back(p);
    }
    int k = 0 ;//记录将要遍历的最小节点的位置
    int t = 0;//记录已经遍历过的节点数，只要遍历 n-1 次
    while(t<n-1){
        add(node,k);  //将 k 和 k+1 合成一个父亲节点
        _sort(node,k);//该排序只需把 k+2 之后的最小两个放在 k+2 和 k+3 位置；
        //sort(node);//如果全部进行排序则要具备稳定性
        k+=2;
        ++t;
    }
    int Sum = sum(node[node.size()-1]);
```

```
100.        cout<<Sum<<endl;
101.        return 0 ;
102.    }
```

附录、提交文件清单