

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	2021/12/2

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 $(B(T) = \sum f(c) \lg T(c))$ 。

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：

```
1. int common(int x,int y){
2.     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
3.     if(x<y) return common(x,y/2);
4.     return x; //x==y 时便是它们的最近公共祖先
5. }
```

代码

```
1. #include<iostream>
2. using namespace std;
3. int common(int x,int y){ //求 x 与 y 的最近公共祖先
4.     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
5.     if(x<y) return common(x,y/2);
6.     return x; //x==y 时便是它们的最近公共祖先
7. }
8. int main(){
9.     int num;
10.    cin>>num;
11.    for(int i = 0 ; i < num ; ++i){
12.        int a,b;
13.        cin>>a>>b;
14.        cout<<common(a,b)<<endl;
15.    }
16.    return 0;
17. }
```

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 $= \sum \text{层数} * \text{频数}$ 。

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

```
1. #include<iostream>
2. using namespace std;
3. #include<vector>
4. void swap(int&a,int &b){
5.     int t = a;
6.     a =b ;
7.     b =t;
8. }
9. void _sort(vector<int> &node,int k){
10.     for(int i=0; i<2 ; ++i){
11.         //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
12.         for(int j = node.size()-1-i ; j >= k+3+i ; --j ){
13.             if(node[j]< node[j-1]){
14.                 swap(node[j], node[j-1]);
15.             }
16.         }
```

```

17.     }
18. }
19. int main(){
20.     int n;
21.     cin>>n;
22.     char c[n];//记录字符
23.     vector<int>a;//频数
24.     for(int i = 0 ; i < n ; ++i){
25.         cin>>c[i];//字符
26.         int t;
27.         cin>>t;
28.         a.push_back(t);//频数
29.     }
30.     _sort(a,-2);//将最小的两个节点放到 a[0]和 a[1]的位置
31.     int sum = 0;//初始位数和为0
32.     int t = 0;//记录已经遍历过的节点数，只要遍历 n-1 次
33.     while(t<n-1){
34.         int newnode = a[2*t]+a[2*t+1];
35.         //由于每次用最小两个节点合成父亲节点，所以用 2*t 记录将要遍历的最小节点的位置
36.         sum+=newnode;//位数和加上合成的父亲节点
37.         a.push_back(newnode);//将合成的父亲节点加入序列中
38.         _sort(a,2*t);
39.         //对 a 的 2*t+2 之后的节点进行排序，只需将最小的两个节点放到 2*t+2 和 2*t+3 的位置
40.         ++t;
41.     }
42.     cout<<sum<<endl;//输出位数和
43.     return 0 ;
44. }

```

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

```

1. 2
2. 10 4
3. 7 13

```

测试输出 1

```
1. 2
2. 3
3.
```

测试输入 2

```
1. 10
2. 3 34
3. 42 45
4. 23 78
5. 31 126
6. 22 133
7. 785 23
8. 12 242
9. 123 5858
10. 26 35
11. 712 72
```

测试输出 2

```
1. 1
2. 5
3. 2
4. 31
5. 2
6. 1
7. 3
8. 1
9. 1
10. 2
11.
```

测试输入 3

```
1. 10
2. 501 1
3. 501 2
4. 501 3
5. 501 4
6. 501 5
7. 501 6
8. 501 7
9. 501 8
10. 501 9
11. 501 10
```

测试输出 3

```
1. 1
2. 1
3. 3
4. 1
5. 1
6. 3
7. 7
8. 1
9. 1
10. 1
11.
```

2) Huffman coding

测试输入 1

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

测试输出 1

45

测试输入 2

```
1. 20
2. 1 1
3. 2 1
4. 3 1
5. 4 1
6. 5 1
7. 6 1
8. 7 1
9. 8 1
10. 9 1
11. 0 1
```

```
12. ! 1
13. @ 1
14. # 1
15. $ 1
16. % 1
17. ^ 1
18. & 1
19. * 1
20. ( 1
21. ) 1
```

测试输出 2

88

测试输入 3

```
1. 26
2. a 24
3. b 4265
4. c 3546
5. d 535
6. e 567
7. f 2111
8. g 4358
9. h 855
10. i 9237
11. j 2424
12. k 2314
13. l 4856
14. m 875
15. n 1058
16. o 872
17. p 4235
18. q 21
19. r 549
20. s 753
21. t 8576
22. u 2451
23. v 435
24. w 1590
25. x 3875
26. y 999
27. 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，不用全排序。所以有如下代码：

```
1. #include<iostream>
2. using namespace std;
3. #include<vector>
4. struct Node{
5.     int d;
6.     char data;
7.     int nums;
8.     Node* lf;
9.     Node* rf;
10.    Node(){
11.        d = 0;
12.        nums = 0;
13.        data = ' ';
14.        lf = NULL;
15.        rf = NULL;
16.    }
17. };
18. void add_d(Node *node){ //增加节点的层数
19.     if(node == NULL) return ;
```



```

20.     if(node->l!=NULL && node->r!=NULL) node->d += 1;
21.     add_d(node->l);
22.     add_d(node->r);
23. }
24. void add(vector<Node*>&node,int k){
25.     Node * p = new Node();
26.     p->nums = node[k]->nums+node[k+1]->nums;
27.     p->l = node[k];
28.     p->r = node[k+1];
29.     add_d(p);
30.     node.push_back(p);
31. }
32. void _swap(Node*&a,Node*&b){
33.     Node* t = a;
34.     a =b ;
35.     b =t;
36. }
37. 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
38.     for(int i=0; i<node.size()-1 ; ++i){ //冒泡排序
39.         for(int j = 0 ; j < node.size()-1-i ; ++j ){
40.             if(node[j]->nums > node[j+1]->nums){
41.                 _swap(node[j], node[j+1]);
42.             }
43.         }
44.     }
45. }
46. void _sort(vector<Node*> &node,int k){
47.     for(int i=0; i<2 ; ++i){
48. //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
49.         for(int j = node.size()-1-i ; j >= k+3+i ; --j ){
50.             if(node[j]->nums < node[j-1]->nums){
51.                 _swap(node[j], node[j-1]);
52.             }
53.         }
54.     }
55. }
56.
57. int sum(Node*node){ //返回所有叶子节点的 层数 d*频数 nums 之和
58.     if(node ==NULL ) return 0;
59.     if(node->l!=NULL && node->r!=NULL) return (node->d)*(node->nums);
60.     return sum(node->l)+sum(node->r);
61. }
62.
63. int main(){
64.     int n;
65.     cin>>n;

```

```

66.     char c[n];
67.     vector<Node*> node;
68.     for(int i = 0 ; i < n ; ++i){
69.         Node * p = new Node();
70.         cin>>c[i]; //字符值
71.         int t;
72.         cin>>t; //个数, 权重
73.         p->data = c[i];
74.         p->nums = t;
75.         node.push_back(p);
76.     }
77.     _sort(node, -2);
78.     int k = 0 ; //记录将要遍历的最小节点的位置
79.     int t = 0; //记录已经遍历过的节点数, 只要遍历 n-1 次
80.     while(t<n-1){
81.         add(node, k);
82.         _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;
83.         //sort(node); //如果全部进行排序则要具备稳定性
84.         k+=2;
85.         ++t;
86.     }
87.     int Sum = sum(node[node.size()-1]); //最后这个节点为哈夫曼树的根节点
88.     cout<<Sum<<endl;
89.     return 0 ;
90. }

```

附录、提交文件清单