Huffman Report

2021201709 李俊霖

算法说明

- 输入后分别统计每个字母对应出现的次数。
- Huffman算法：初始情况为为每一个字符所在的节点创建一棵单节点的树，组成一个森林，每次对节点按照频率从小到大排序。使用贪心策略，每次选择当前具有频率最小的2棵树，且将这两棵树合并成一棵新的树。
- 具体实现方法是，以每个字母出现的次数（频率fre）为键值构建优先队列，用小项堆来维护这个优先队列。
- 在读取并输出每一个字母对应的编码时，使用中序遍历的方法。左0右1，迭代完一次回退一个字符。

```
// 霍夫曼编码
void huffman_incode()
{

    while (prior_que.size() > 1)
    {
        Tree *pr = new Tree;
        pTree pl, pr;
        pl = prior_que.top();
        prior_que.pop();
        pr = prior_que.top();
        prior_que.pop();

        pr->freq = pl->fre + pr->fre;
        pr->left = pl;
        pr->right = pr;

        prior_que.push(pr);
    }

    string str = "";
    printCode(prior_que.top(), str);
    del(prior_que.top());
}
```

复杂度分析

设需要编码的有n个字符，输入字符串长度为m。

- 数据的输入和统计处理需要O(m)时间。
- 优先队列的初始化时间复杂度为O(n)；
- 最小堆的节点删除、插入时间复杂度为O(logn)；
- 合并森林为单树的过程共有n - 1次合并，时间复杂度为O(nlogn)。
- 合并输出编码字符答案需要O(m)时间。
  该算法的总时间复杂度为O(m)+ O(nlogn)。