算法设计与分析实验报告

倪玮昊

2020211346

一、 实验题目

1.1 题目描述

对给出的字符设计 Huffman 编码，计算期望：

$$W = \sum_{1}^{n} P_i * L_i$$

其中 P_i、L_i 代表第 i 个字符的出现概率与编码长度。

1.2 输入格式

输入文件名为 huffman.in。输入共两行。
第一行一个正整数 n，代表字符个数。
第二行包含 n 个三位小数 P_i，代表第 i 个字符的出现概率，两数之间用空格隔开。

1.3 输出格式

输出文件名为 huffman.out。输出共一行。
第一行包含一个三位小数 W，为最后的期望。

1.4 输入输出样例

huffman.in	huffman.out
4 0.100 0.100 0.200 0.600	1.600

1.5 数据范围

$0 < n \leq 10^6$,
$0 < Pi \leq 1$。

1.6 说明/提示

假设四个字符的编码分别是 000、001、01、1，则 $0.1\times3+0.1\times3+0.2\times2+0.6\times1=1.6$。

二、实验过程

算法思路

哈夫曼编码方法
哈夫曼编码首先会使用字符的频率创建一棵树，然后通过这个树的结构为每个字符生成一个特定的编码，出现频率高的字符使用较短的编码，出现频率低的则使用较长的编码，这样就会使编码之后的字符串平均长度降低，从而达到数据无损压缩的目的。

1. 计算字符串中每个字符的频率：

2. 按照字符出现的频率进行排序，组成一个队列 Q
   出现频率最低的在前面，出现频率高的在后面。

3. 把这些字符作为叶子节点开始构建一颗哈夫曼树
   哈夫曼树又称为最优二叉树，是一种带权路径长度最短的二叉树。

关键函数及代码段的描述

快速排序

```
void Quick_Sort(double a[], int begin, int end){
    if(begin >= end)//只有一个元素,递归完成
        return;
    double sign = a[end];
    int i = begin;
    int j = end;
    while(i != j){//对右边大和左边小的值进行调换
        while(a[i] <= sign && j > i)
            i++;
        while(a[j] >= sign && j > i)
            j--;
        if(j > i){
            double t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    a[end] = a[j];//与末尾调换
    a[j] = sign;
    Quick_Sort(a, begin, i-1);//重复过程
    Quick_Sort(a, i+1, end);
}
```

建立树,计算期望

```
for( i=2;i<n;i++){
    for(int i1=0;i1<i;i1++){
        p[i1].expect++;
    }
}
for( i=0;i<n;i++){
   expecting=expecting+p[i].expect*p[i].value;
}
```

算法时间及空间复杂性分析

采用快速排序，编码的时间复杂度是O(n^2logn)；

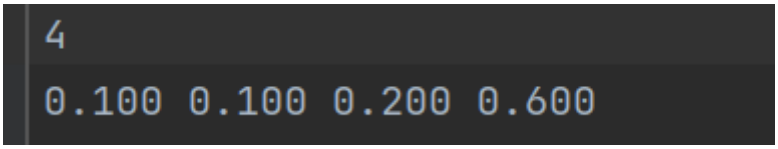
三、实验结果

程序执行环境及运行方式

win11,CLion

程序执行示例

输入:



输出:

四、 实验总结

以前学习过哈夫曼编码,大作业也用哈夫曼编码实现过一个小的压缩文件和解压文件,故不难

五、 算法源代码

```
#include <deque>
#include <algorithm>
#include <iostream>
#include <fstream>
using namespace std;

#define rep(i, a, b) for (int i = a; i <= b; ++i)

class point{

public:
    int expect=1;
    double value;
};
void Quick_Sort(double a[], int begin, int end){
    if(begin >= end)//只有一个元素,递归完成
        return;
    double sign = a[end];
    int i = begin;
    int j = end;
    while(i != j){//对右边大和左边小的值进行调换
        while(a[i] <= sign && j > i)
            i++;
        while(a[j] >= sign && j > i)
            j--;
        if(j > i){
            double t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    a[end] = a[j];//与末尾调换
```

```
    a[j] = sign;
    Quick_Sort(a, begin, i-1);//重复过程
    Quick_Sort(a, i+1, end);
}

int main()
{
    int i;
    fstream fin("huffman.in", ios::in), fout("huffman.out", ios::out);
    int n;
    double  tmp;
    fin >> n;
    double a[n];
    double expecting=0;
    rep(i, 0, n-1) {
        fin >> tmp;
        a[i]=tmp;
    }
    Quick_Sort(a,0,n-1);
    point p[n];
    rep(i, 0, n-1) {
        fin >> tmp;
        p[i].value=a[i];
    }
    for( i=2;i<n;i++){
        for(int i1=0;i1<i;i1++){
            p[i1].expect++;
        }
    }
    for( i=0;i<n;i++){
       expecting=expecting+p[i].expect*p[i].value;
    }
    fout<<expecting;

    return 0;

}
```