哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：数据结构与算法

课程类型：必修

实验项目：树形结构及其应用

实验题目：哈夫曼编码与译码方法

实验日期：2020 年 11 月 3 日

班级：

学号：

姓名： Youngsc

设计成绩	报告成绩	指导老师
		张岩

一、 实验目的

1. 掌握树的链式存储方式及其操作实现（创建、遍历、查找等）。
2. 掌握二叉树用不同方法表示所对应的不同输入形式。
3. 掌握二叉树中各种重要性质在解决实际问题中的应用。
4. 掌握哈夫曼树的构造方法及其编码方法。
5. 掌握二叉排序树的特性及其构造方法。

二、实验要求及实验环境

（一）实验要求

1. 从文件中读入任意一篇英文文本文件，分别统计英文文本文件中各字符（包括标点符号和空格）的使用频率；
2. 根据已统计的字符使用频率构造哈夫曼编码树，并给出每个字符的哈夫曼编码（字符集的哈夫曼编码表）；能够计算一元多项式的 k 阶导函数。
3. 将文本文件利用哈夫曼树进行编码，存储成压缩文件（哈夫曼编码文件）；
4. 计算哈夫曼编码文件的压缩率；
5. 将哈夫曼编码文件译码为文本文件，并与原文件进行比较。
6. 能否利用堆结构，优化的哈夫曼编码算法。
7. 上述 1-5 的编码和译码是基于字符的压缩，考虑基于单词的压缩，完成上述工作，讨论并比较压缩效果。
8. 上述 1-5 的编码是二进制的编码，可以采用 K 叉的哈夫曼树完成上述工作，实现“K 进制”的编码和译码，并与二进制的编码和译码进行比较。

（二）实验环境

1. 硬件环境
   a) Legion Y7000P 2019 PG0
   b) CPU: Intel(R)_Core(TM)_i7-9750H_CPU_@_2.60GHz 2.59GHz
   c) 内存(RAM): 16GB DDR4
2. 系统环境
   a) Windows10 家庭中文版
3. 开发工具
   a) Dev C++5.11
   b) gcc (GCC) 9.2.0

（三）

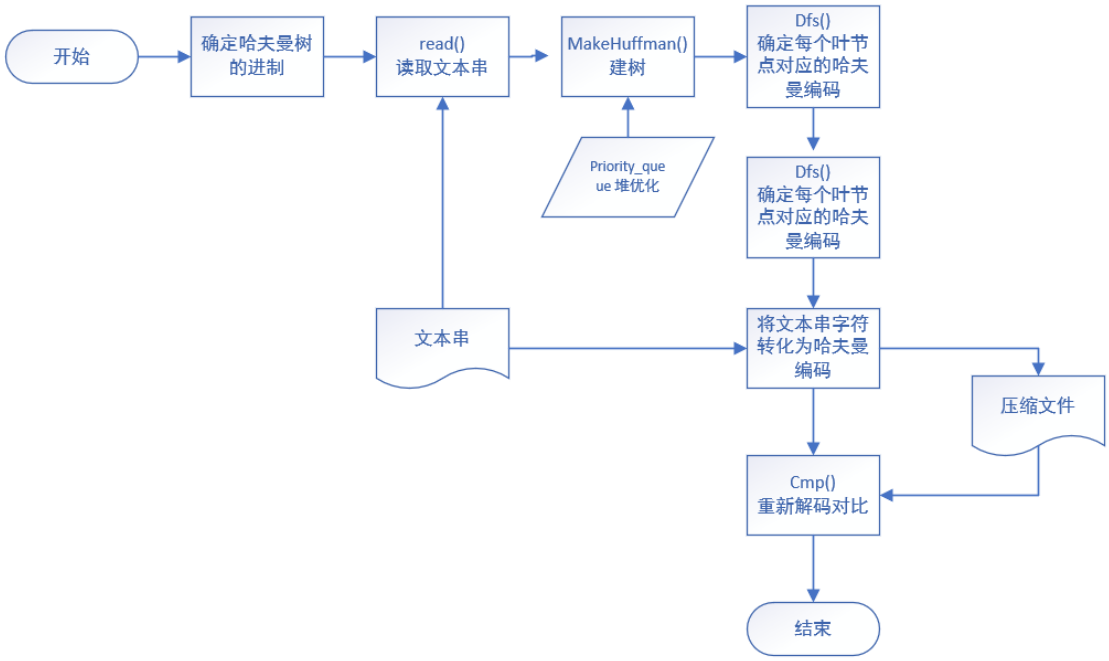
三、设计思想（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系）

1．逻辑设计

首先在读入英文文本后，我们将所有的出现的字符进行计数并排序，京他们分别作为最终哈夫曼树的各个叶节点，且节点的权值为对应字符出现次数，随后采用贪心的思想，贪心得选取两个权值最小的节点合并为一个节点，该节点权值为两节点权值之和，原先两个节点作为新节点的两个子节点，用新节点来取代原先的两个节点，然后再在所有现存的节点中进行相同的操作，直到最终只剩下一个节点，该节点即为哈夫曼树的根节点。对于这棵哈夫曼树，我们将所有节点与其左儿子（如果存在）之间的连边看作 0，所有节点与其右儿子（如果存在）之间的连边看作 1，那么从根节点到每个叶节点的最短路径上经过的边所代表的 01 序列即为该叶节点所代表的字符的哈夫曼编码。在实现 K 叉哈夫曼树压缩时，我们如果直接每次取 K 个压缩，会造成最终剩下若干个不到 K 个节点使得此时情况并不是最优，为了避免这样的状况，我们首先计算出最终会省下几个节点，然后再第一次合并时先将这么多的节点合并，随后再依次取 K 个节点合并，最终会剩下一个根节点吗。这样可以保证哈夫曼编码最优。

2．物理设计

读取的过程中，我们采用 C++STL 中的 string 字符串来储存英语文章以及压缩后的编码，来增强灵活性和便捷性，由于输入的英文文章中出现的英文字符的不确定性，我们采用一个 C++的 STL 中的 map 映射来保存各个字符出现的次数，随后通过遍历该容器中的元素来对应添加叶节点。在构建哈夫曼树的过程中，我们需要每次取出两个权值最小的节点元素来进行合并，这个过程我们采用 C++STL 中的 priority_queue 优先队列（堆）来进行取权值最小节点时的优化。生成哈夫曼编码时，我们对哈夫曼树进行遍历，遍历的过程中，我们同样来使用 C++STL 中的 string 来储存当前串以及每个字符的哈夫曼编码，同时我们采用 C++STL 中的 map 映射来保存不同字符以及哈夫曼编码之间的对应关系。此外，实现 K 叉树哈夫曼压缩时，将每个节点的子节点指针增加。



四、测试结果

输入样例 1	输出样例 1
2 aaaaaaaaabbbbbbbbaaaabbaaaccccccccccccaaaaaaa	11111111100000000000000001111000011101010101010101010101011111111
输入样例 2	输出样例 2
2 zxcvbnzxcvbnzxcvbzxcvbzxcvbzxcvbzxcvb	1101110001101100110111000110110011011100011011101110001101110111000110111011100011011101110001101

五、经验体会与不足

通过本次实验我学到了哈夫曼编码的压缩原理和机制，以及二叉树的一些基本操作与功能，学会了用堆进行过程优化，学到了二叉树的哈夫曼编码强于多叉树的哈夫曼编码。

六、附录：源代码（带注释）

```
# include <bits/stdc++.h>

# define MAXN 10


using namespace std;
```

```
int K;
map <char,int> t;
map <char,string> s;
string TXT,Hufcode;

void read(){
    getline(cin,TXT);
    int len = TXT.length();
    for (int i=0; i<len; ++i) t[TXT[i]]++;
}

namespace KthHuffman{
    struct Knode{
        char c;
        int weight;
        Knode* Child[MAXN];
        clear(){for (int i=0; i<K; ++i) Child[i]=NULL;}
        Knode(){clear(); c = 0; weight = 0;}
        Knode(char _c,int _weight){c = _c,weight = _weight,clear();}
        Knode(int _weight){weight = _weight; clear(); c = 0;}
        bool operator < (const Knode& p) const {
            return weight > p.weight;
        }
    } *root;

    priority_queue <Knode> q;

    Knode* MakeHuffman(){
```

```
        map<char,int>::iterator iter;
        printf("各字母出现频率为：\n");
        for (iter=t.begin(); iter != t.end(); iter++) q.push(*new
Knode(iter->first,iter->second)),printf("%c: %d\n",iter->first,iter->second);
        int sum = (q.size()-1)%(K-1)+1;
        if (sum > 1)
        {
            Knode* now = new Knode;
            for (int i=0; i<sum; ++i)
            {
                Knode* x = new Knode; *x = q.top(); q.pop();
                now->Child[i] = x;
                now->weight += x->weight;
            }
            q.push(*now);
        }
        while (1)
        {
            if (q.size() == 1)
            {
                Knode* ret = new Knode;
                *ret = q.top();
                return ret;
            }
            Knode* now = new Knode;
            for (int i=0; i<K; ++i)
            {
                Knode* x = new Knode; *x = q.top(); q.pop();
                now->Child[i] = x;
```

```
            now->weight += x->weight;
        }
        q.push(*now);
    }
}

void dfs(Knode* now,string S){
    if (now->c)
    {
        s[now->c] = S;
        return;
     }
    for (int i=0; now->Child[i]!=NULL; ++i) dfs(now->Child[i],S+(char)(i+'0'));
}

bool cmp(){
    freopen("textans.out","w",stdout);
    int Hufflen = Hufcode.length(),TXTlen = TXT.length();
    int t=0,i;
    for (i=0; i<TXTlen; ++i)
    {
        Knode *now = root;
        do
        {
            now = now->Child[Hufcode[t]-'0'];
            t++;
        } while (now->c == 0);
        if (now->c != TXT[i])
        {
```

```
                fclose(stdout);
                freopen("CON","w",stdout);
                return 0;
            }
            printf("%c",now->c);
        }
        fclose(stdout);
        freopen("CON","w",stdout);
        return i == TXTlen;
    }

    Main(){
        root = MakeHuffman();
        dfs(root,"");
        map<char,string>::iterator iter;
        printf("各字母哈夫曼编码为：\n");
        for (iter=s.begin(); iter != s.end(); iter++) cout << iter->first << ": " <<
iter->second << endl;
        int len = TXT.length(),sum=0;
        freopen("text.out","w",stdout);
        for (int i=0; i<len; ++i) {
            cout << s[TXT[i]];
            sum += s[TXT[i]].length();
            Hufcode += s[TXT[i]];
        }
        fclose(stdout);
        freopen("CON","w",stdout);
        printf("文件压缩完成！\n该文件的压缩率
为: %.81f%\n",((int)log2(K-1)+1)*sum*1.0/len/8*100);
```

```
        if (cmp()) printf("经比较，解压正确\n");
        else printf("经比较，解压错误\n");
    }
}

namespace Huffman{
    struct node{
        char c;
        int weight;
        node* LeftChild;
        node* RightChild;
        node(){LeftChild = RightChild = NULL; c = 0;}
        node(char _c,int _weight){c = _c,weight = _weight,LeftChild = RightChild = NULL;}
        node(int _weight,node *_Left,node* _Right){weight = _weight,LeftChild =_Left;
RightChild = _Right; c = 0;}
        bool operator < (const node& p) const {
            return weight > p.weight;
        }
    } *root;

    priority_queue <node> q; // 定义一个堆进行堆优化

    node* MakeHuffman(){
        map<char,int>::iterator iter; // 用map来存字符
        printf("各字母出现频率为：\n");
        for (iter=t.begin(); iter != t.end(); iter++) q.push(*new
node(iter->first,iter->second)),printf("%c: %d\n",iter->first,iter->second);
        while (!q.empty())
        {
```

```
        if (q.size() == 1)
        {
            node* ret = new node;
            *ret = q.top();
            return ret;
        }// 合并到只剩一个节点时候该节点为根节点
        node* x = new node; *x = q.top(); q.pop();
        node* y = new node; *y = q.top(); q.pop();
        q.push(*new node(x->weight+y->weight,x,y));
    }
}

void dfs(node* now,string S){ // 对哈夫曼树进行深搜得到每个字符对应的哈夫曼编码
    if (now->LeftChild == NULL)
    {
        s[now->c] = S;
        return;
     }
    dfs(now->LeftChild,S+"0");
    dfs(now->RightChild,S+"1");
}

bool cmp(){
    freopen("textans.out","w",stdout);
    int Hufflen = Hufcode.length(),TXTlen = TXT.length();
    int t=0,i;
    for (i=0; i<TXTlen; ++i)
    {
        node *now = root;
```

```
            do
            {
                if (Hufcode[t] == '1') now = now->RightChild;
                else now = now->LeftChild;
                t++;
            } while (now->c == 0);
            if (now->c != TXT[i])
            {
                fclose(stdout);
                freopen("CON","w",stdout);
                return 0;
            }
            printf("%c",now->c);
        }
        fclose(stdout);
        freopen("CON","w",stdout);
        return i == TXTlen;
    }

    Main(){
        root = MakeHuffman();
        dfs(root,"");
        map<char,string>::iterator iter;
        printf("各字母哈夫曼编码为：\n");
        for (iter=s.begin(); iter != s.end(); iter++) cout << iter->first << ": " <<
iter->second << endl;
        int len = TXT.length(),sum=0;
        freopen("text.out","w",stdout);
        for (int i=0; i<len; ++i) {
```

```
            cout << s[TXT[i]];
            sum += s[TXT[i]].length();
            Hufcode += s[TXT[i]];
        }
        fclose(stdout);
        freopen("CON","w",stdout);
        printf("文件压缩完成！\n 该文件的压缩率为: %.8lf%\n",sum*1.0/len/8*100);
        if (cmp()) printf("经比较，解压正确\n");
        else printf("经比较，解压错误\n");
    }
}

using namespace Huffman;
using namespace KthHuffman;

int main(){
    printf("输入哈夫曼编码进制：");
    scanf("%d",&K);
    freopen("text.in","r",stdin);
    read();//读取文本
    fclose(stdin);//关闭读入文件
    if (K==2) Huffman::Main();
    else if (K>2) KthHuffman::Main();//确定进制
    else printf("输入有误");
    return 0;
}
```