实验三 二叉树的应用:Huffman 压缩

PB17081504
廖洲洲
2018.12.4

1. 实验要求

用 Huffman 压缩技术实现对任意文件的压缩和解压缩处理。要求对所有的文件类型进行压缩，压缩之后的文件后缀名为 huff。同时，可以对所有后缀名为 huff 的压缩文件进行解压缩。

2. 实验原理

哈夫曼编码是一种可变字长编码方式。这种方法依据字符出现的概率来构造异字头的平均长度最短的码字，有时称之为最佳编码，一般就叫哈夫曼编码。也就是说其通过使用较短的码字来给出现概率较高的信源符号编码，从而使平均码字最短。

3. 实验内容

（一）压缩部分：

1. 构造哈夫曼树，对其进行前缀编码

（1）扫描待压缩文件，得出各字符出现频率。

（2）根据给定的 n 个权值 {W1,W2,...Wn} 构成 n 棵二叉树的集合 F={T1，T2，…Tn}，每棵二叉树 Ti 中只有一个带权为 Wi 的根节点，其左右子树均空。

（3）在 F 中选取两棵根节点的权值最小的树作为左右子树构造一棵新的二叉树， 且新的二叉树的根节点的权值为其左右子树上根节点的权值之

和。

（4）在 F 中删除这两棵树。同时将新得到的二叉树加入 F 中。 重复（2）和（3），直到 F 只含一棵树为止。这棵树便是哈夫曼树。

2. 由 Huffman 树得到各字符前缀编码。

3. 根据前缀编码，对文件中各个字符进行编码， 并按每八位一次写入压缩文件。

4. 处理剩余不到八位部分，写入压缩文件。

5. 将前缀编码及相关信息写入压缩文件。

6. 关闭指针，完成压缩。

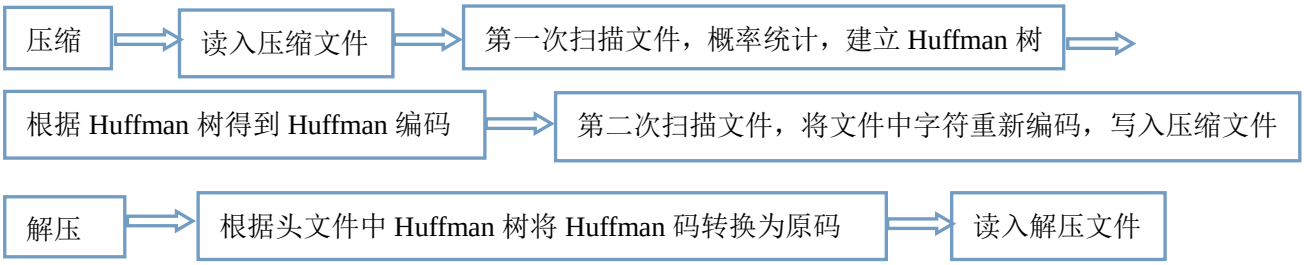
（二）解压部分

1. 读入压缩文件，读取文件头中存储的 Huffman 树，补零个数等信息。

2. 对文件中各字符与压缩相反的进行解码，并写入解压文件。

3.关闭指针，完成解压

（三）具体流程



4. 实验关键代码讲述

界面入口部分

```
void CMFCHuffmanDlg::OnBnClickedButton1()
{
    // TODO: 在此添加控件通知处理程序代码
    CFileDialog OpenFile(TRUE, NULL, NULL, 6UL, _T("All Files(*.*)|*.*"), this);// TODO: 在此添加控件通知处理程序代
    CString pathname;
    char *p;
    p = new char[200];
    if (OpenFile.DoModal() == IDOK) {
        pathname = OpenFile.GetPathName();
        USES_CONVERSION;
        p = T2A(pathname);
    }
    HuffmanCoding(p);
}

void CMFCHuffmanDlg::OnBnClickedButton2()
{
    // TODO: 在此添加控件通知处理程序代码
    CFileDialog OpenFile(TRUE, _T("huff"), NULL, 6UL, _T("Huffman Files(*.huff)|*.huff"), this);// TODO: 在此添加控
    CString pathname;
    char *p;
    p = new char[200];
    if (OpenFile.DoModal() == IDOK) {
        pathname = OpenFile.GetPathName();
        USES_CONVERSION;
        p = T2A(pathname);
    }
    HuffmanDecoding(p);
}
```

直接在 Visual Studio 中创建 MFC 程序，在自动生成的图形界面中修改。

压缩部分

- 第一次扫描，统计字符频率

```
//开始第一遍扫描文件，创建Huffman树
for (int i = 0; i < 520; i++) {
    HT[i].weight = 0;
    HT[i].lchild = 0;
    HT[i].parent = 0;
    HT[i].rchild = 0;
}
unsigned long length = 0;
unsigned char c;
while (fread(&c, 1, 1, infp)) {
    HT[c + 1].weight++;
    length++;

}  //统计完成
short int charcount = 0;  //统计字符种类数
for (int i = 1; i <= 256; i++) {
    if (HT[i].weight) {
        HT[i].date = (unsigned char)(i - 1);
        charcount++;
    }
}//给统计到的字符存储在节点中
//下面将在文件中存在的字符节点放在数组前面，便于建树
//采用冒泡排序
BubbleSort(HT + 1, 257); //冒泡排序完成
```

- 建树、求字符编码

```
int m = 2 * charcount - 1;
int s1 = 1, s2 = 1;
for (int i = charcount + 1; i <= m; ++i) {
    //在HT【1...i】选择perent为0且weight最小的两个结点，取序号分别为s1和s2
    Select(i - 1, &s1, &s2);
    HT[s1].parent = i; HT[s2].parent = i;
    HT[i].lchild = s1; HT[i].rchild = s2;
    HT[i].weight = HT[s1].weight + HT[s2].weight;
}
//从叶子到根逆向求每个字符的Huffman编码
char *cd;
int start;
int j, f;
HC = (char * *)malloc((charcount + 1) * sizeof(char*));
cd = (char *)malloc(charcount * sizeof(char)); //存储每次读出的编码
cd[charcount - 1] = '\0';

for (int i = 1; i <= charcount; i++) {
    start = charcount - 1;//编码结束符位置
    for (j = i, f = HT[i].parent; f != 0; j = f, f = HT[f].parent) {//从叶子到根逆向求编码
        if (HT[f].lchild == j)
            cd[--start] = '0';
        else
            cd[--start] = '1';
    }
    HC[i] = (char*)malloc((charcount - start) * sizeof(char));
    strcpy(HC[i], cd + start);
    printf("%s\n", HC[i]);
}
```

- 将 Huffman 树、叶节点个数号写入压缩文件头，同时预留空间用来存储补零数

```
fwrite(&charcount, sizeof(short int), 1, outfp);
fwrite(ht, sizeof(htnode), 2 * charcount, outfp);//将Huffman树写入压缩文件头
short int num0 = 0;
//  fseek(outfp,sizeof(int),SEEK_CUR);//将读写指针向尾移动存储int的空间，用来存储加入的0个数
fwrite(&num0, sizeof(short int), 1, outfp);
fseek(infp, 0, SEEK_SET);//将读写指针移到文件开头
char code[300];//记录每次翻译的编码，最长的编码不会超过256位
code[0] = '\0';
```

- 第二次扫描文件，将读到的字符用相应的编码代替写入压缩文件

```
while (fread(&c, 1, 1, infp)) {
    for (i = 1; i <= charcount; i++) {
        if (c == HT[i].date)
            break;
    }
    //      int len=strlen(HC[i]);
    //      printf("%d\n",len);
    strcat(code, HC[i]);
    codelength = strlen(code);
    c = 0;
    while (codelength >= 8) {
        for (int i = 0; i < 8; i++) {
            if (code[i] == '1')     //c用来暂时保存新的编码，经过8位移位，其二进制码变为新编码
                c = (c << 1) | 1;      //向左移一位，最后一位通过或加1
            else
                c = c << 1;        //直接左移，自动补0
        }
        fwrite(&c, sizeof(char), 1, outfp); //写入压缩文件
        strcpy(code, code + 8);
        codelength = strlen(code);
    }
}
```

- 补零操作

```
if (codelength > 0) { //若最后未满8位，补零
    num0 = 8 - codelength;
    strcat(code, "00000000");//补0
    c = 0;
    for (i = 0; i < 8; i++) {
        if (code[i] == '1')
            c = (c << 1) | 1;
        else
            c = c << 1;
    }
    fwrite(&c, sizeof(char), 1, outfp);
}  //写入完成
```

- 压缩完成，写入补零个数，关闭文件

```
fseek(outfp, 2 * charcount * sizeof(htnode) + sizeof(short int), SEEK_SET);
fwrite(&num0, sizeof(short int), 1, outfp);

fclose(infp);
fclose(outfp);
```

解压部分

- 读取文件头信息

```
short   int charcount;
short   int num0;
int i;
fread(&charcount, sizeof(short int), 1, infp);
htnode *ht;
ht = (htnode *)malloc(2 * charcount * sizeof(htnode));
HTNode *HT;
HT = (HTNode *)malloc(2 * charcount * sizeof(HTNode));

for (i = 0; i < 2 * charcount; i++)
    fread(&ht[i], sizeof(htnode), 1, infp);
for (i = 0; i < 2 * charcount; i++) {
    HT[i].date = ht[i].date;
    HT[i].lchild = ht[i].lchild;
    HT[i].rchild = ht[i].rchild;

}
for (i = 1; i < 2 * charcount; i++) {
    HT[HT[i].lchild].parent = i;
    HT[HT[i].rchild].parent = i;
}
fread(&num0, sizeof(short int), 1, infp);
for (i = 1; i <= charcount; i++) {
    printf("%c...", HT[i].date);
}
```

- 扫描文件，根据 Huffman 树解出字符，写入解压文件

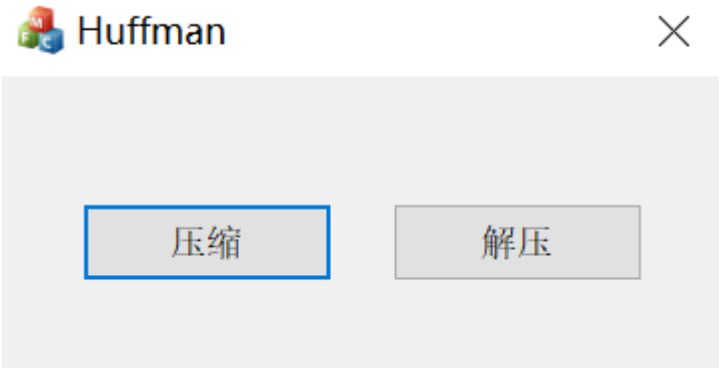
```
while (fread(&c, sizeof(char), 1, infp)) {
    j = (long)c;
    itoa(j, buffer, 2);          //将c转换位二进制,由于转换后可能后面不是8位，需要在前
    codelength = strlen(buffer);
    len0 = 8 - codelength;
    for (int i = 0; i < len0; i++)
        strcat(code, "0");
    strcat(code, buffer);
    if (feof(infp)) {
        for (int i = 0; i < 8 - num0; i++) {
            if (code[i] == '1') {
                if (flag == 1) {
                    flag = 0;
                    node = HT[rootnode.rchild];
                }
                else
                    node = HT[node.rchild];
            }
            else {
                if (flag == 1) {
                    flag = 0;
                    node = HT[rootnode.lchild];
                }
                else
                    node = HT[node.lchild];
            }
            if (!node.lchild && !node.rchild) {//读到数据点
                date = node.date;
                fwrite(&date, sizeof(char), 1, outfp);
                fwrite(&date, sizeof(char), 1, outfp);
                flag = 1;
            }
        }
    }
    else {
        for (int i = 0; i < 8; i++) {
            if (code[i] == '1') {
                if (flag == 1) {
                    flag = 0;
                    node = HT[rootnode.rchild];
                }
                else
                    node = HT[node.rchild];
            }
            else {
                if (flag == 1) {
                    flag = 0;
                    node = HT[rootnode.lchild];
                }
                else
                    node = HT[node.lchild];
            }
            if (!node.lchild && !node.rchild) {//读到数据点
                date = node.date;
                fwrite(&date, sizeof(char), 1, outfp);
                flag = 1;
            }
        }
```

这里需要判别编码是否从根开始走，故用 flag 标志，一旦读到数

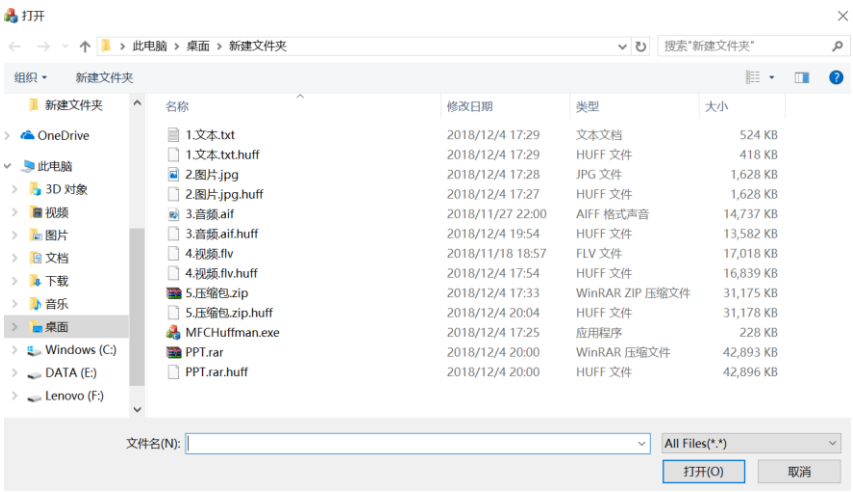
据点，说明要从根结点开始走。同时若时文件最后一个字符，则还需把补的 0 去掉。

5. 实验结果及分析（结合相关数据截图分析）

本软件启动界面如下：



点击相应按钮进入相应功能，首先进入文件选择界面，然后将进行解压缩：



测试集解压缩展示：

1.文本.txt	2018/12/4 17:29	文本文档	524 KB
1.文本.txt.huff	2018/12/4 17:29	HUFF 文件	418 KB
2.图片.jpg	2018/12/4 17:28	JPG 文件	1,628 KB
2.图片.jpg.huff	2018/12/4 17:27	HUFF 文件	1,628 KB
3.音频.aif	2018/11/27 22:00	AIFF 格式声音	14,737 KB
3.音频.aif.huff	2018/12/4 19:54	HUFF 文件	13,582 KB
4.视频.flv	2018/11/18 18:57	FLV 文件	17,018 KB
4.视频.flv.huff	2018/12/4 17:54	HUFF 文件	16,839 KB
5.压缩包.zip	2018/12/4 17:33	WinRAR ZIP 压缩文件	31,175 KB
5.压缩包.zip.huff	2018/12/4 20:04	HUFF 文件	31,178 KB
MFCHuffman.exe	2018/12/4 17:25	应用程序	228 KB
PPT.rar	2018/12/4 20:00	WinRAR 压缩文件	42,893 KB
PPT.rar.huff	2018/12/4 20:00	HUFF 文件	42,896 KB

可以发现：使用 Huffman 编码对于文本、音频、视频的压缩较为有效，但对于图片、压缩包来说，由于图片使用的各个字符频率相差不大，故压缩率较低，而压缩包是由更为成熟的压缩软件压缩的，故可供压缩的空间几乎没有，同时由于要保存树的信息，故压缩后文件反而更大了。

6. 实验小结

- **通过本实验我对 Huffman 编码技术有了更深的了解，更熟悉树的存储结构，同时加深的对树的创建、遍历等操作的了解。**
- **本次实验学习了图形化界面的生成，第一次利用 Visual Studio 的 MFC 写出了图形化程序。让我有了很大的提升。**