C语言大作业——Huffman

如何实现压缩——Huffman编码

- 1.等长编码ASCII码表

例如：

’a’ = 97D = 61H = 0110 0001 B

’A’ = 65D = 41H = 0100 0001 B

’ ’ = 32D = 20H = 0010 0000 B

Huffman编码是什么?

- 2.一种更高效的编码

高效？？

Huffman编码是什么?

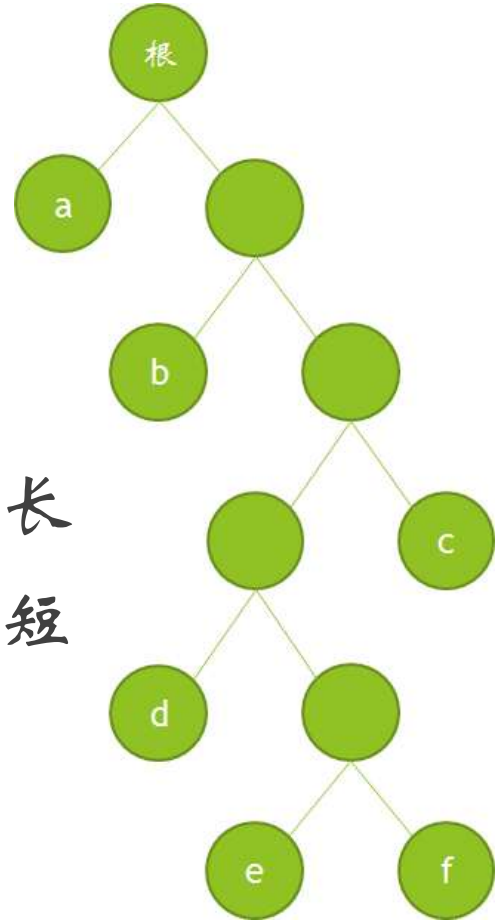
- 2.一种更高效的编码

——如何最优

按照频率分配编码长度

出现次数少的，频率低的字母，编码长

出现次数多的，频率高的字母，编码短



Huffman编码是什么?

- 2.一种更高效的编码

  它可能是这个样子的（左0右1）

  ’a’ = 0 B

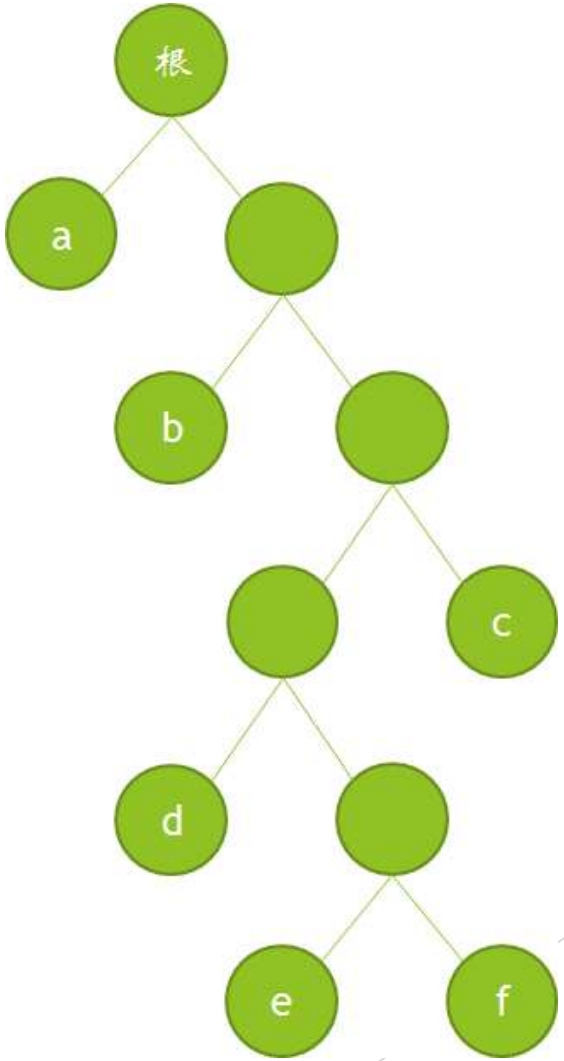
  ’b’ = 10 B

  ’c’ = 111 B

  ’d’ = 1100 B

  ’e’ = 11010 B

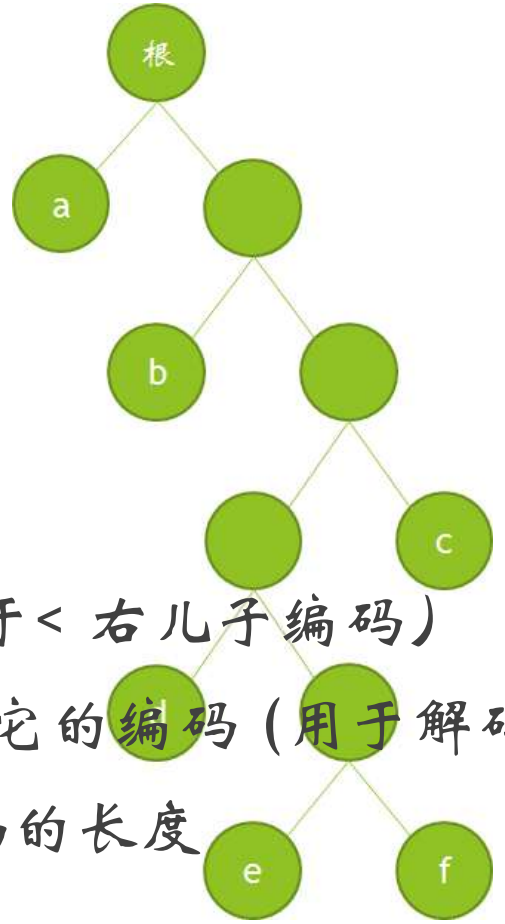
  ’f’ = 11011 B



Huffman编码是什么?

2.一种更高效的编码

——分析结构



1. 树的每一个节点，最多有两个分支
2. 只有叶子结点可以为字符编码；
3. 左儿子为0，右儿子为1（左儿子编码 永远小于< 右儿子编码）
4. 从根结点root走到叶子leaf结点的路径对应着它的编码（用于解码）
5. 某个叶子节点的高度即为该叶节点的哈曼编码的长度

如何进行Huffman编码——构建huffman树

- 1.统计每个字符出现的次数
- 2.把字符和对应的次数“绑”在一起
- 3.把每一捆都扔进一个“池子” 里（程序里用priority queue优先队列——pq模拟，当然可以用数组等其他数据结构模拟）
- 4.从池子里捞出次数最小的两捆
- 5.将频次最小的两捆扎成一捆，新捆的次数是两捆的频次相加的和，新捆就是两个儿子的根结点。把这个结点再次扔进池子。注意：新捆的字符没有意义！
- 6.如果池子里的捆超过1个，那么就回到4，继续打捆.
- 7.剩下的这一个就是总的根

如何进行Huffman编码——构建huffman树build tree

iteration	tree
1	{('a',0.01),('b',0.04),('c',0.05),('d',0.11),('e',0.19),('f',0.20),('g',0.4)}
2	{('ab',0.05),('c',0.05),('d',0.11),('e',0.19),('f',0.20),('g',0.4)}
3	{('abc',0.1),('d',0.11),('e',0.19),('f',0.20),('g',0.4)}
4	{('e',0.19),('f',0.20),('abcd',0.21),('g',0.4)}
5	{('abcd',0.21),('ef',0.39),('g',0.4)}
6	{('g',0.4),('abcdef',0.6)}
7	{('abcdefg',1.00)}

- 1.统计每个字符出现的次数
- 2.把字符和对应的次数“绑”在一起
- 3.把每一捆都扔进一个“池子” 里（程序里用priority queue优先队列——pq模拟，当然可以用数组等其他数据结构模拟）
- 4.从池子里捞出次数最小的两捆
- 5.将频次最小的两捆扎成一捆，新捆的次数是两捆的频次相加的和，新捆就是两个儿子的根结点。把这个结点再次扔进池子。注意：新捆的字符没有意义！
- 6.如果池子里的捆超过1个，那么就回到4，继续打捆.
- 7.剩下的这一个就是总的根

如何进行Huffman编码？

- 捆怎么存？——数据结构

```
struct kun
{
    kun *left,*right;//左右两个儿子结点
    char ch;//存字符（仅当结点是叶子结点时有效）
    float p;//频次（这里用频率，你也可以用int记录出现的具体次数）
    ...//(其他可以简化你操作的，你认为有用的东西，也许是int isleaf，也许是
        kun*parent，也许是...)
};
```

如何进行Huffman编码?

- 模板给出的数据结构——huffman树中节点数据结构
- struct tnode {　　//tree node
- struct tnode* left; /*used when in tree*/　　//左右儿子节点
- struct tnode* right; /*used when in tree*/
- struct tnode* parent; /*used when in tree*/　//父亲节点，用于从叶子节点搜索到根节点
- struct tnode* next; /*used when in list*/ //链表相关数据结构——pq优先队列中使用
- float freq;　　//单词频率
- int isleaf;　　//是否为叶子节点的标志，1：叶子节点，0：非叶子节点
- char symbol;　//字母
- };

解码流程

```
int main() {
    const char* IN_FILE = "encoded.txt";
    const char* CODE_FILE = "code.txt";
    const char* OUT_FILE = "decoded.txt";
    FILE* fout;
    FILE* fin;
    /*allocate root*/
    root = talloc();
    fin = fopen(CODE_FILE, "r");
    /*build tree*/
    build_tree(fin);
    fclose(fin);

    /*decode*/
    fin = fopen(IN_FILE, "r");
    fout = fopen(OUT_FILE, "w");
    decode(fin, fout);
    fclose(fin);
    fclose(fout);
    /*cleanup*/
    freetree(root);
    return 0;
}
```

Build_tree 建树流程

```
/* @function build_tree 注意：code.txt格式，一行一个 symbol，因此以行为单位处理
   @desc     builds the symbol tree given the list of symbols and code.h */
void build_tree(FILE* fp) {
        char symbol;          char strcode[MAX_LEN];
        int items_read;       int i, len;
        struct tnode* curr = NULL;
        while (!feof(fp)) {                          //读入每1行，并为每个 symbol建树
        items_read = fscanf(fp, "%c %s\n", &symbol, strcode);          //格式: “symbol+ +strcode”
        if (items_read != 2) break;                  //判断成功
        curr = root;                                 //from root  从根节点开始重建
        len = strlen(strcode);     //hufman code length = depth 编码长度等于该叶子节点的深度
        for (i = 0; i < len; i++) {     //从根节点遍历 rebuild the tree according to strcode[len] 0-depth-1
                                        //根据strcode中保存huffman码的每一字符重建树
        if(strcode[i] == '0') { //strcode[i] == 0 : left 左儿子
                                 if(curr -> left == NULL)
                                curr -> left = talloc();
                                curr = curr -> left;
                        }
                      else {
                                 if(curr -> right == NULL)                  //strcode[i] != 0 :  right 右儿子
                                curr -> right = talloc();
                                curr = curr -> right;
                        }
                }
        curr->isleaf = 1;      /*到达叶子节点assign code for the leaf node */
        curr->symbol = symbol;
        printf("inserted symbol:%c\n", symbol);
    }
```

decode 解码流程

```
//解码流程
void decode(FILE* fin, FILE* fout) {
    char c;
    struct tnode* curr = root;          //start from root从根开始解码
    while ((c = getc(fin)) != EOF) {    //从压缩文件读入每一个字符
        /*TODO:
                traverse the tree
                print the symbols only if you encounter a leaf node
                遍历树，遍历到叶节点说明一个huffman码解码结束；
                恢复current指针指向root，继续读入下一个huffman码进行解码
        */
        if(c == '0') {
                    curr = curr -> left;
                    if(curr->isleaf) { //叶子节点，解码并继续if leaf decode and traverse again from the root
                    fprintf(fout, "%c", curr -> symbol);
                     curr = root;       //恢复current指针指向huffman树根节点root
                }
        }
        else
        {
            curr = curr -> right;
            if(curr->isleaf) {        //叶子节点，解码并继续if leaf decode and traverse again from the root
                fprintf(fout, "%c", curr -> symbol);
                curr = root;          //恢复current points to the root again
            }
        }
    }
}
```

释放树 freetree 流程

```
//删除这棵树 recursive function递归函数，从叶节点开始释放
//递归过程同学们通过debug跟踪
void freetree(struct tnode* root) {
    if (root == NULL)
        return;
                        //递归到base case，释放左、右、父节点
    freetree(root->right);          //释放右儿子节点
    freetree(root->left);           //释放左儿子节点
    free(root);                  //释放父节点parent node
}
```

对于本次实验的要求

- 建议大家按照所给的代码进行完善，其中注释中有“TODO”的地方是需要大家自己去完善的。
- 实在觉得有很大压力的同学，可以用数组去模拟出池子的操作~
- 觉得毫无压力游刃有余的同学，可以自学“堆”，脱离所给代码进行编写程序。

对于本次实验的要求

- Huffman编码
- 输入：一个文本“a.txt”里面有文字信息
- 输出：它对应的编码后的文件

- Huffman解码
- 输入：它对应的编码后的文件
- 输出：还原出原文件（a.txt的内容）

有关解码

- 1.还原这棵树
- 2.把得到的编码按照树的结构还原出来

注意

- 本次实验编码量和阅读量都很大，所以，大家一定要熟练运用调试功能
- 厉害的同学不要送“助攻”，否则，很惨！
- 无论做到什么程度，都不要放弃！会根据完成的情况给分数。