编程实验：

1. 实验目的与要求

在学习和理解二叉树的原理、构造及遍历方法的基础上，应用所学知识来解决实际问题。

本实验将通过一个实际应用问题的解决过程掌握 Huffman 树的构造、Huffman 编码的生成及基于
涉及的知识点包括树的构造、遍历及 C 语言位运算和二进制文件。

2. 实验内容

Huffman 编码文件压缩

【问题描述】

编写一程序采用 Huffman 编码对一个正文文件进行压缩。具体压缩方法如下：

1. 对正文文件中字符(换行字符'\n'除外，不统计)按出现次数（即频率）进行统计。
2. 依据字符频率生成相应的 Huffman 树（未出现的字符不生成）。
3. 依据 Huffman 树生成相应字符的 Huffman 编码。
4. 依据字符 Huffman 编码压缩文件（即将源文件字符按照其 Huffman 编码输出）。

说明：

1. 只对文件中出现的字符生成 Huffman 树，注意：一定不要处理\n，即不要为其生成 Huffman 编
2. 采用 ASCII 码值为 0 的字符作为压缩文件的结束符（即可将其出现次数设为 1 来参与编码）。
3. 在生成 Huffman 树前，初始在对字符频率权重进行（由小至大）排序时，频率相同的字符 AS
到有序权重序列中时，若出现相同权重，则将新生成的权重节点插入到原有相同权重节点之后（
4. 在生成 Huffman 树时，权重节点在前的作为左孩子节点，权重节点在后的作为右孩子节点。
5. 遍历 Huffman 树生成字符 Huffman 码时，左边为 0 右边为 1。
6. 源文件是文本文件，字符采用 ASCII 编码，每个字符占 8 个二进制位；而采用 Huffman 编码
此最后输出时需要使用 C 语言中的位运算将字符的 Huffman 码依次输出到每个字节中。

【输入形式】

对当前目录下文件 input.txt 进行压缩。

【输出形式】

将压缩后结果输出到文件 output.txt 中，同时将压缩结果用十六进制形式（printf("%x",...)）输出到

3. 实验准备

1．文件下载

从教学平台（judge.buaa.edu.cn）课程下载区下载文件 lab_tree2.rar，该文件中包括了本实验中用

l huffman2student.c：该文件给出本实验程序的框架，框架中部分内容未完成（见下面相关实验步
程序运行后得到相应要求的运行结果；

l input.txt：为本实验的测试数据。

2. huffman2student.c 文件中相关数据结构说明

结构类型说明：

```
struct tnode {     //Huffman 树结构节点类型
    char c;
    int weight;
    struct tnode *left;
    struct tnode *right;
     } ;
```

结构类型 struct tnode 用来定义 Huffman 树的节点，其中；

1）对于树的叶节点，成员 c 和 weight 用来存放字符及其出现次数；对于非叶节点来说，c 值可不
点生成条件，若 p 为当前 Huffman 树节点指针，则有：

p->weight = p->left->weight + p->right->weigth；
2）成员 left 和 right 分别为 Huffman 树节点左右子树节点指针。
全局变量说明：
```
 int Ccount[128]={0};
       struct tnode *Root=NULL;
char HCode[128][MAXSIZE]={0};
      int Step=0;
FILE *Src, *Obj;
```
整型数组 Ccount 存放每个字符的出现次数，如 Ccount[‘a’]表示字符 a 的出现次数。
变量 Root 为所生成的 Huffman 树的根节点指针。
数组 HCode 用于存储字符的 Huffman 编码，如 HCode['a']为字符 a 的 Huffman 编码，本实验中为
变量 Step 为实验步骤状态变量，其取值为 1、2、3、4，分别对应实验步骤 1、2、3、4。
变量 Src、Obj 为输入输出的文件指针，分别用于打开输入文件“input.txt”和输出文件“output.tx
4. 实验步骤
【步骤 1】
1)　实验要求
在程序文件 huffman2student.c 中“//【实验步骤 1】开始”和“ //【实验步骤 1】结束”间编写相应
input.txt 中字符出现频率。
```
//【实验步骤 1】开始
void statCount()
{



    }
    //【实验步骤 1】结束
```
2)　实验说明
函数 statCount 用来统计输入文件（文件指针为全局变量 Src）中字符的出现次数（频率），并将字
Ccount[‘a’]存放字符 a 的出现次数。
注意：**在该函数中 Ccount[0]一定要置为 1，即 Ccount[0]=1**。编码值为 0（’\0’）的字符用来作为
3)　实验结果
函数 print1()用来打印输出步骤 1 的结果，即输出数组 Ccount 中字符出现次数多于 0 的字符及次数
骤 1】编码后，本地编译并运行该程序，并在标准输入中输入 1，程序运行正确时在屏幕上将输出

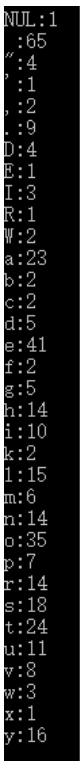


图 1 步骤 1 运行结果

在本地运行正确的情况下，将你所编写的程序文件中//【实验步骤 1】开始”和“ //【实验步骤 1】
代码【实验步骤 1】下的框中，然后点击提交按钮，若得到如下运行结果（**测试数据 1** 评判结果

测试数据	评判结果
测试数据1	完全正确
测试数据2	输出错误
测试数据3	输出错误
测试数据4	输出错误

表明**实验步骤 1：通过**，否则：**不通过。**

【步骤 2】

1)　　实验要求

在程序文件 huffman2student.c 中的 “//【实验步骤 2】开始”和“ //【实验步骤 2】结束”间编写
一个根结点指针为 Root 的 Huffman 树。

```
    //【实验步骤 2】开始
void createHTree( )
{



    }
    //【实验步骤 2】结束
```

2)　　实验说明

在程序文件 huffman2student.c 中函数 createHTree 将根据每个字符的出现次数（字符出现次数存

码值为 i 的字符出现次数），按照 Huffman 树生成规则，生成一棵 Huffman 树。

算法提示：

1．依据数组 Ccout 中出现次数不为 0 的（ 即 Ccount[i]>0）项，构造出树林 F={T0, T1, ¼, Tm}，
且根结点(叶结点)的权值为相应字符的出现次数的二叉树（每棵树结点的类型为 struct tnode，其

```
for(i=0; i<128; i++)
if(Ccount[i]>0){
p = (struct tnode *)malloc(sizeof(struct tnode));
p->c = i; p->weight = Ccount[i];
p->left = p->right = NULL;
add p into F;
}
```

2．对树林 F 中每棵树按其根结点的权值由小至大进行排序（排序时，当权值 weight 相同时，字

3．while 树个数>1 in F

a) 将 F 中 T0 和 T1 作为左、右子树合并成为一棵新的二叉树 T’，并令 T’->weight= T0->weigh

b) 删除 T0 和 T1 from F，同时将 T’加入 F。要求加入 T’后 F 仍然有序。若 F 中有树根结点权

4．Root = T0 （Root 为 Huffman 树的根结点指针。循环结束时，F 中只有一个 T0）

注：在实现函数 createHTree 时，在框中还可根据需要定义其它函数，例如：

```
void myfun()
{
  …
}
void createHTree()
{
  …
 myfun();
  …
}
```

3)　　实验结果

函数 print2()用来打印输出步骤 2 的结果，即按前序遍历方式遍历步骤 2 所生成（由全局变量 Roo
时编码值为 0 的字符用 NUL 表示、空格符用 SP 表示、制表符用 TAB 表示、回车符用 CR 表示。
序，并在标准输入中输入 2，程序运行正确时在屏幕上将输出如下结果：

```
" D W b c f s o . k NUL ' d i g E R I e a u m w x , t h n r l p v y SP
```

图 2 步骤 2 运行结果

在本地运行正确的情况下，将你在本地所编写的程序文件中//【实验步骤 2】开始”和“ //【实验
后所附代码【实验步骤 2】下的框中，然后点击提交按钮，若得到如下运行结果（**测试数据 2** 评

测试数据	评判结果
测试数据1	完全正确
测试数据2	完全正确
测试数据3	输出错误
测试数据4	输出错误

表明**实验步骤 2：通过**，否则：**不通过。**

【步骤 3】

1)　实验要求

在程序文件 huffman2student.c 中的 “//【实验步骤 3】开始”和“ //【实验步骤 3】结束”间编写相
【实验步骤 3】中所产生的 Huffman 树为文本中出现的每个字符生成对应的 Huffman 编码。遍历
右边为 1。

```
    //【实验步骤 3】开始
void makeHCode( )
{



      }
    //【实验步骤 3】结束
```

2)　实验说明

【步骤 3】依据【步骤 2】所生成的根结点为 Root 的 Huffman 树生为文本中出现的每个字符生成
如下：

```
char HCode[128][MAXSIZE];
```

HCode 变量用来存放每个字符的 Huffman 编码串，如 HCode[‘e’]存放的是字母 e 的 Huffman 编码

算法提示：

可编写一个按前序遍历方法对根节点为 Root 的树进行遍历的递归函数，并在遍历过程中用一个字
的路径（经过的边），经过左边时记录为’0’，经过右边时记录为’1’；当遍历节点为叶节点时，
即执行 strcpy(HCode[p->c],路径串)。

注：在实现函数 makeHCode 时，在框中还可根据需要定义其它函数，如调用一个有类于前序遍
Huffman 编码：

```
void visitHTree()
{
  …
}
void makeHCode()
{
  …
  visitHTree();
  …
}
```

3)　实验结果

函数 print3()用来打印输出步骤 3 的结果，即输出步骤 3 所生成的存储在全局变量 HCode 中非空字
后，本地编译并运行该程序，并在标准输入中输入 3，在屏幕上将输出 ASCII 字符与其 Huffman 编
的 Huffman 编码，其中 NUL 表示 ASCII 编码为 0 的字符，SP 表示空格字符编码值为 0 的字符用

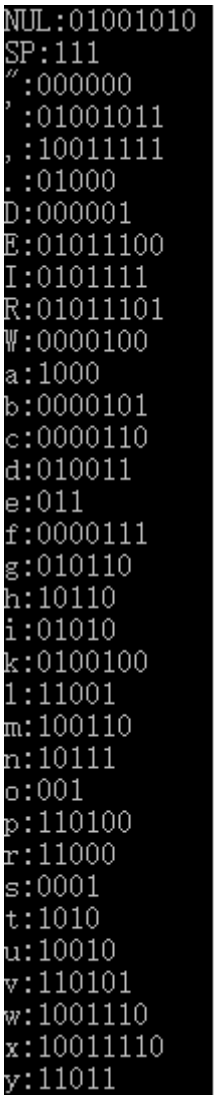


图 3 步骤 3 运行结果

在本地运行正确的情况下，将你在本地所编写的程序文件中//【实验步骤 3】开始”和“ //【实验
后所附代码【实验步骤 3】下的框中，然后点击提交按钮，若得到如下运行结果（**测试数据 3** 评

测试数据	评判结果
测试数据1	完全正确
测试数据2	完全正确
测试数据3	完全正确
测试数据4	输出错误

表明**实验步骤 3：通过**，否则：**不通过。**

【步骤 4】

1)　　实验要求

在程序文件 huffman2student.c 函数中的 “//【实验步骤 4】开始”和“ //【实验步骤 4】结束”间
据【实验步骤 3】中所生成的字符 ASCII 码与 Huffman 编码对应表（存储在全局变量 HCode 中，
编码串，在本实验中值为字符串”011”），将原文本文件（文件指针为 Src）内容（ASCII 字符）转
（文件指针为 Obj）中，以实现文件压缩。同时将输出结果用十六进制形式（printf("%x",...)）输

```
    //【实验步骤4】开始
void atoHZIP()
{


    }
    //【实验步骤4】结束
```

2)　　实验说明

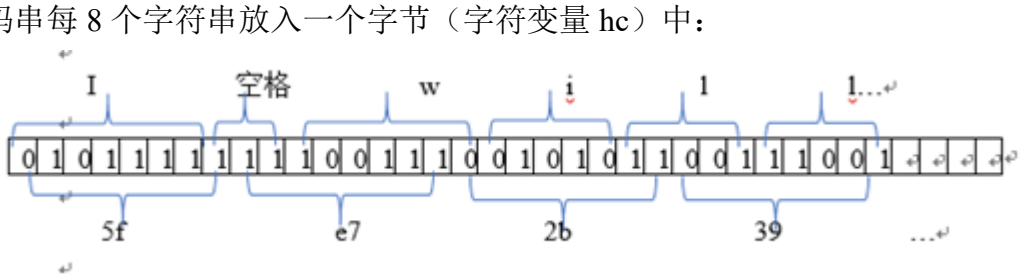
Huffman 压缩原理：在当前 Windows、Linux 操作系统下，文本文件通常以 ASCII 编码方式存储和
定占一个字节（即 8 位），如字符'e'的 ASCII 编码为十进制 101（十六进制 65，二进制为 011001
验中其依据文本中字符的频率进行编码，频率高的字符的编码长度短（小于 8 位），而频率低的字
验中，字符''（空格）的出现频率最高（出现 65 次），其 Huffman 编码为 111（占 3 位，远小于一
如字符'e'的 Huffman 编码为 011、字符'o'的 Huffman 编码为 111；字符'x'出现频率低（出现 1 次
好一个字节）（注意，在其它问题中，字符最长 Huffman 编码可能会超过 8 位）。正是由于高频字
总长度要小于 ASCII 文本文件，以实现压缩文件的目的。

然而，将普通 ASCII 文本文件转换为变长编码的文件不便之处在于 C 语言中输入/输出函数数据处
法直接将 Huffman（不定长）编码字符输出，在输出时需要将不定长编码序列转换为定长序列，按
符其编码要比一个字节短（如本实验中字符'e'的 Huffman 编码为 011，不够一个字节，还需要和
编码可能超过一个字节。如何将不定长编码字符序列转换成定长字符序列输出，一个简单方法是

1）根据输入字符序列将其 Huffman 编码串连接成一个（由 0、1 字符组成的）串；
2）然后依次读取该串中字符，依次放入到一个字节的相应位上；
3）若放满一个字节（即 8 位），可输出该字节；剩余的字符开始放入到下一个字节中；
4）重复步骤 2 和 3，直到串中所有字符处理完。

下面通过实例来说明：

原始文件 input.txt 中内容以"I will…"开始，依据所生成的 Huffman 码表，字母 I 对应的 Huffma
对应"1001110"，i 对应"01010"，l 对应　　"11001"。因此，将其转换后得到一个 Huffman 编码
由于在 C 中，最小输出单位是字节（共 8 位），因此，要通过 C 语言的位操作符将每 8 个 01
"01011111"中的每个 0 和 1 放入到一个字符中十六进制（即 printf（"%x",c）输出时，屏幕上将显
编码串每 8 个字符串放入一个字节（字符变量 hc）中：



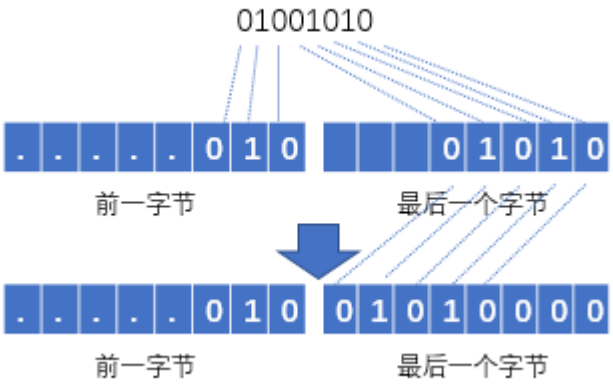

```
char hc;
...
for(i=0; s[i] != '\0'; i++) {
    hc = (hc << 1) | (s[i]-'0');
if((i+1)%8 == 0) {
fputc(hc,obj);      //输出到目标（压缩）文件中
```

```
printf("%x",hc);   //按十六进制输出到屏幕上
}
}
```
…

说明：

1．当遇到源文本文件输入结束时，应将输入结束符的 Huffman 码放到 Huffman 编码串最后，即将

2．在处理完成所有 Huffman 编码串时（如上述算法结束时），处理源文本最后一个字符（文件结束符时，可能出现如下情况：其子串”010”位于前一个字节中输出，而子串“01010”位于另（最后）左端的头，最后 3 位补 0，然后再输出最后一个字节。



注：在实现函数 atoHZIP 时，在框中还可根据需要定义其它函数或全局变量，如：

```
void myfun()
{
  …
}
void atoHZIP()
{
  …
 myfun();
  …
}
```

1)　　实验结果

函数 print4()用来打印输出步骤 4 的结果，即根据输出步骤 3 所生成的存储在全局变量 HCode 中 H 的 ASCII 字符转换为 Huffman 编码字符输出到文件 output.txt 中，同时按十六进行输出到屏幕上。程序，并在标准输入中输入 4，在屏幕上将输出：

图 4 步骤 4 运行结

说明：
从屏幕输出结果可以看出，由于采用了不定长的 Huffman 编码，且出现频率高的字符的编码长度
200 字节，文件压缩了 45.95%。

在本地运行正确的情况下，将你在本地所编写的程序文件中//【实验步骤 4】开始”和“ //【实验
实验报告后所附代码【实验步骤 4】下的框中，然后点击提交按钮，若得到如下运行结果（**测试**

测试数据	评判结果
测试数据1	完全正确
测试数据2	完全正确
测试数据3	完全正确
测试数据4	完全正确

表明**实验步骤 4：通过**，否则：**不通过。**
