**Huffman解码在ffmpeg中的实现**

# Huffman编码

霍夫曼树是根据样本的概率模型排列出的最佳熵树，比如给出8个样本{A,B,C,D,E,F,G,H}，对应的出现概率假设为{0.1, 0.15, 0.15, 0.2, 0.05, 0.05, 0.26, 0.04}，那么按照概率出现大小ascend排序，样本有排列如下：

{H, E, F, A, B, C, D, G}

那么样本的霍夫曼树构成过程如下：

{0.04, 0.05, 0.05, 0.1, 0.15, 0.15, 0.2, 0.26}

{ 0.05, 0.09, 0.1, 0.15, 0.15, 0.2, 0.26}

{0.1, 0.14, 0.15, 0.15, 0.2, 0.26}

{ 0.15, 0.15, 0.2, 0.24,0.26}

{ 0.2, 0.24, 0.26, 0.3}

{ 0.26, 0.3, 0.44}

{0.44, 0.56}

以次

1. 将样本最小的两个元素按左小右大的顺序排列
2. 并将之和重置到数组中按升序排
3. 重复1，直到数组中只剩两数

这样得到的数如下所示：

C

B

G

D

A

F

E

H

以左向step为0，右项为1，我们能得到如下编码：

D : 00

A : 010

F : 0110

H : 01110

E : 01111

G : 10

B : 110

C : 111

这个过程就是，huffman编码

# Huffman解码

解码过程很简单，由于树节点的唯一性，以上节的二叉树为例，假如有码流如00110111就算read bit by bit，我们也能解出DBC这串码字。但是一个Bit的读法其实效率相当低，要读8次才能解码出全部码字。

想当然，三个码字最低需要3次读取，如何做到？Ffmpeg给出答案。

继续上述例子，给出二叉树：

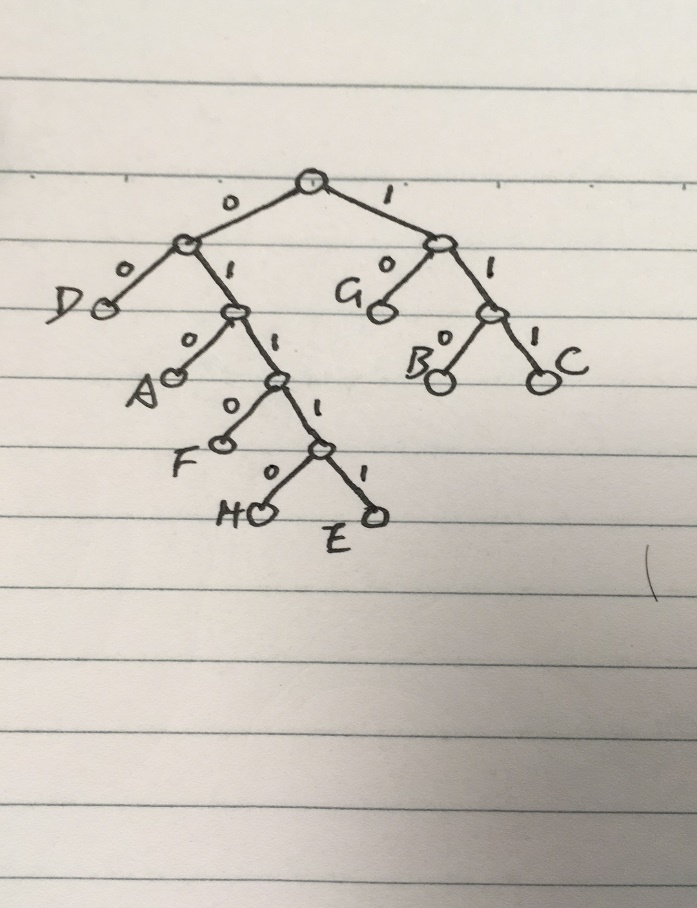


图1

Ffmpeg通过每次读取3bit来进行解码，如果3bit内有对应的叶节点，则输出码字，没有则继续读下3bit来寻找对应的节点。形象一点说，就是，将整个二叉树以3层为单位，层层递进，层层寻找叶节点。那么图1可划分为：

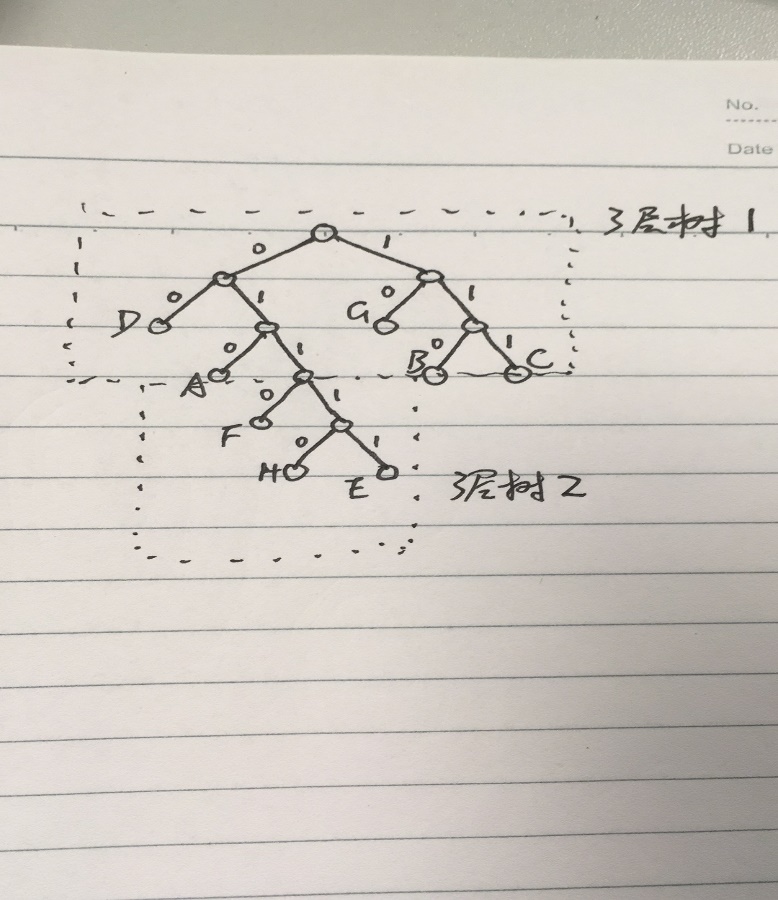


图2

图中可能会有疑惑，不过是3树1还是3树2，似乎都不是完整的树，特别是3树2，明明最多只有2层深度，这如何理解？

其实很简单，2层树也能引申层3层树，更何况是3树1那种缺胳膊少腿的情况？我们适当补全之后，得：

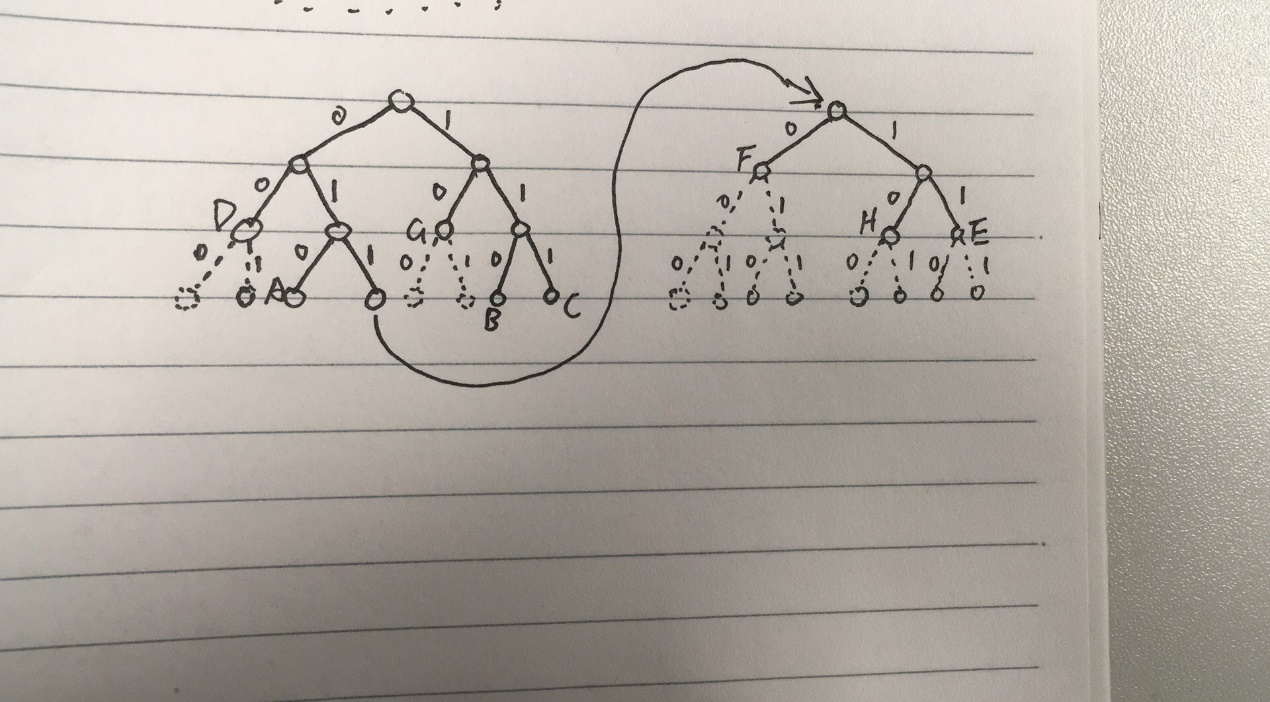


图3

所以，我们分解出来了2个3层树。接下来，如何通过读到的3bit数据来解析码字？

Ffmpeg按照如下方式，在每个3层树处做同样处理：

1. 读完3bit，先确定该节点是否为叶节点。
2. 如果是叶节点，则标明实际所占bit数，并且将其所有子孙节点所表示值都归为叶节点所有。

如D点，只占了2个bit，但是在整个3层树种，经过D的值有000, 001。按照这法则1，如果我们读3bit的时候读到了000，001，那么都表示我们碰到了D节点，但是实际值用了2bit，所以我们要还给bit reader 1个bit。

1. 如果不是叶节点，则代表这个节点下面还有子树。这时候，就再读3bit，回到0步骤，如此递归。

最后，我们用上述方法解0011001111111码流。

1. 先读3bit，得001。发现经过了D的前缀00，则先解出D，还给bit reader 1个bit，码流变为11001111111
2. 再读3bit，得110，那就是B，不用还，码流变为01111111
3. 继续读3bit，得011，发现不是叶节点，需要在子树中继续读3bit，得111，在子树中发现了E的前缀11，则解出E，记得还给bit reader 1bit，这个过程总共消耗5bit。码流变为111
4. 再读3bit，得111，这就是C。

所以，解码得DBEC