8.6 哈夫曼编码

8.6 哈夫曼编码

1 引言

哈夫曼（Huffman）编码算法是基于二叉树构建编码压缩结构的，它是数据压缩中经典的一种算法。算法根据文本字符出现的频率 (或者权值)，重新对字符进行编码。为了缩短编码的长度，我们自然希望频率越高的词，编码越短，这样才能最大化压缩存储文本数据的空间。

假设现在我们要对下面这句歌词“we will we will r u”进行压缩。如果使用 ASCII 码对这句话编码则为：119 101 32 119 105 108 108 32 119 101 32 119 105 108 108 32 114 32 117（十进制表示）。我们可以看出需要 19 个字节，也就是至少需要 152 位的内存空间去存储这些数据。

很显然直接用 ASCII 码编码很浪费空间，Unicode 就更不用说了。下面我们先来统计一下这句话中每个字符出现的频率。如下表，按频率高低已排序：

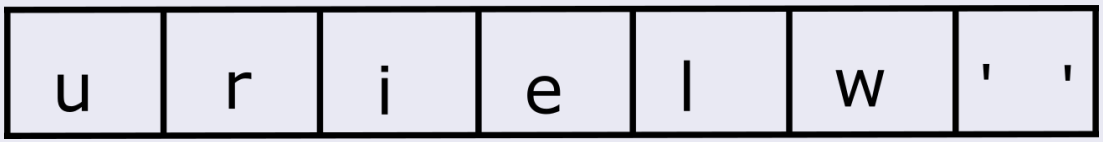
字符	空	w	l	e	i	r	u
频率	5	4	4	2	2	1	1

8.6 哈夫曼编码

2 哈夫曼二叉树构建

2.1 初始队列

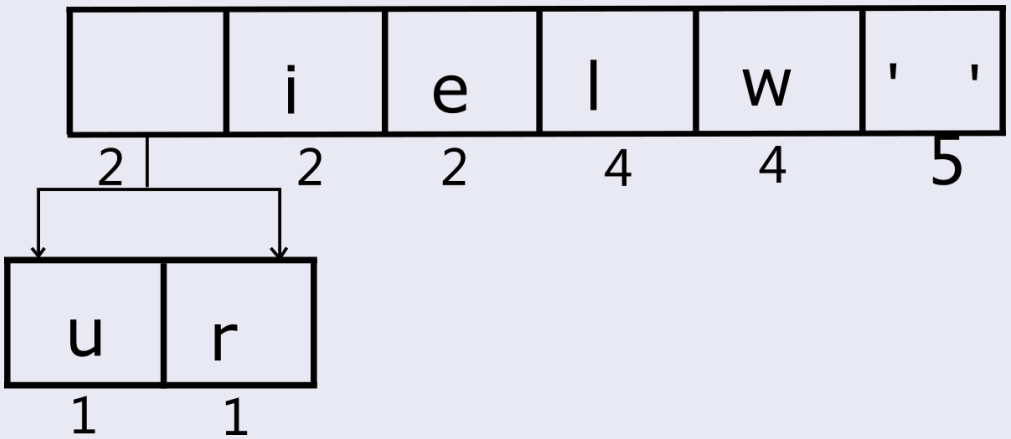
按出现频率高低将其放入一个优先级队列中，从左到右依次为频率逐渐增加。



将这个队列转换成哈夫曼树，哈夫曼树是一颗带权重的二叉树，权重由队列中每个字符出现的次数（频率）决定。哈夫曼树始终保证权重越大的字符出现在越高的地方。

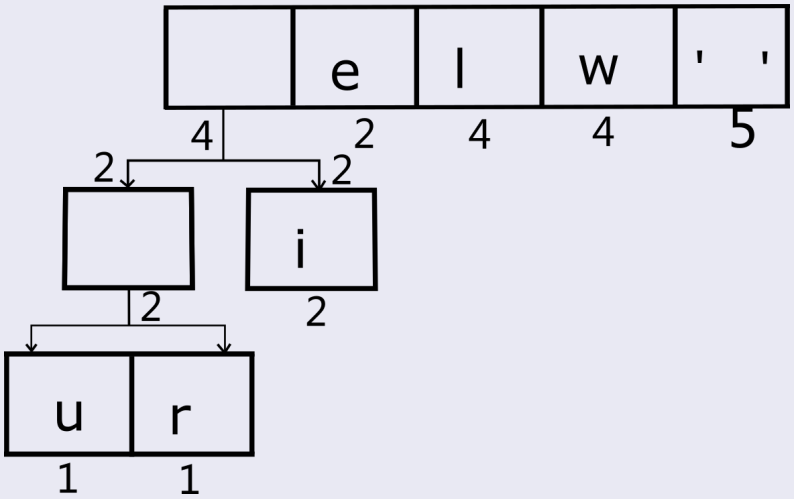
2.2 第一步合并

从左到右进行合并，依次构建二叉树。第一步取前两个字符 u 和 r 来构造初始二叉树，第一个字符作为左节点，第二个元素作为右节点，然后两个元素相加作为新元素，并且两者权重相加作为新元素的权重。



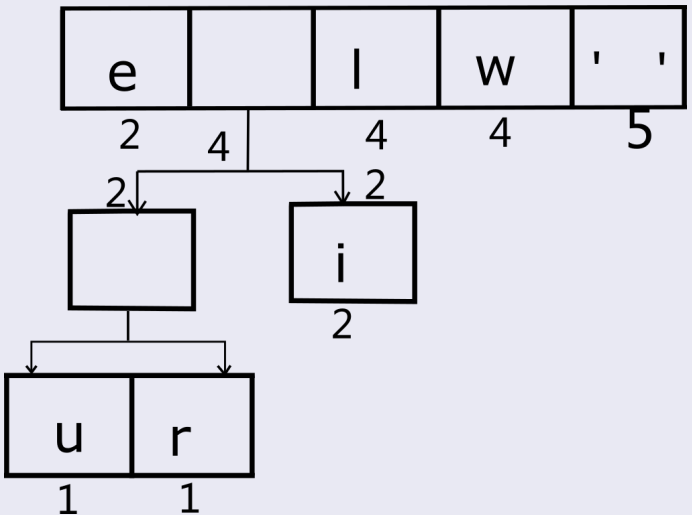
8.6 哈夫曼编码

同理，新元素可以和字符 i 再合并，如下：



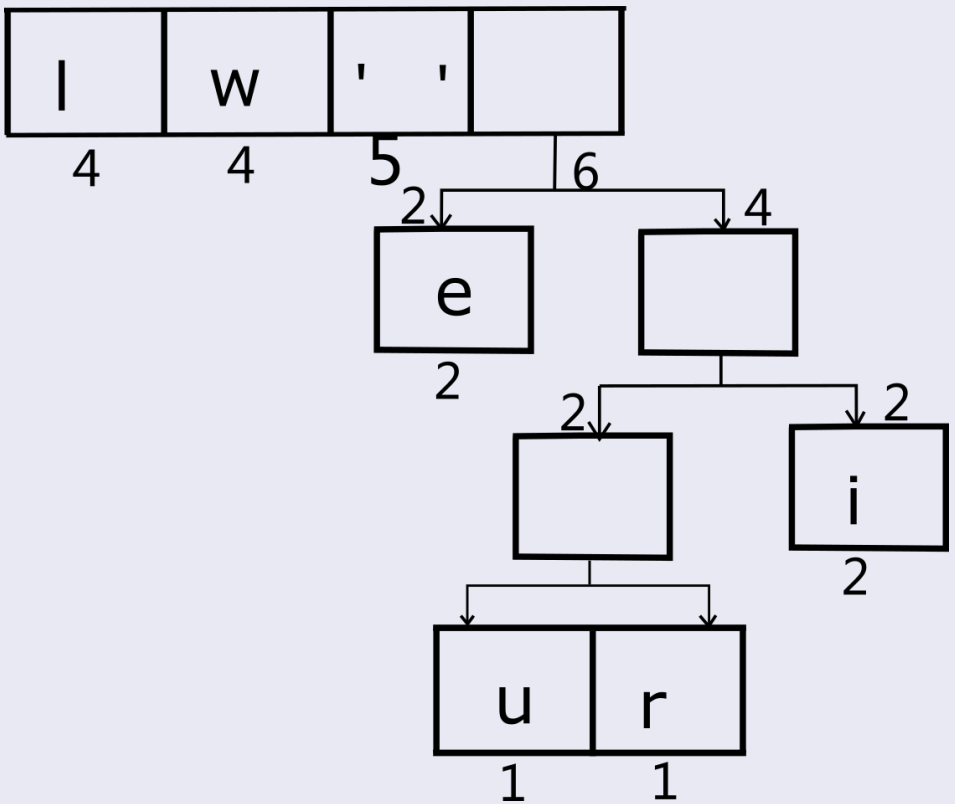
2.3 重新调整队列

上图新元素权重相加后结果变大，权重比它前面的权重大，需要对权重进行重新排序。



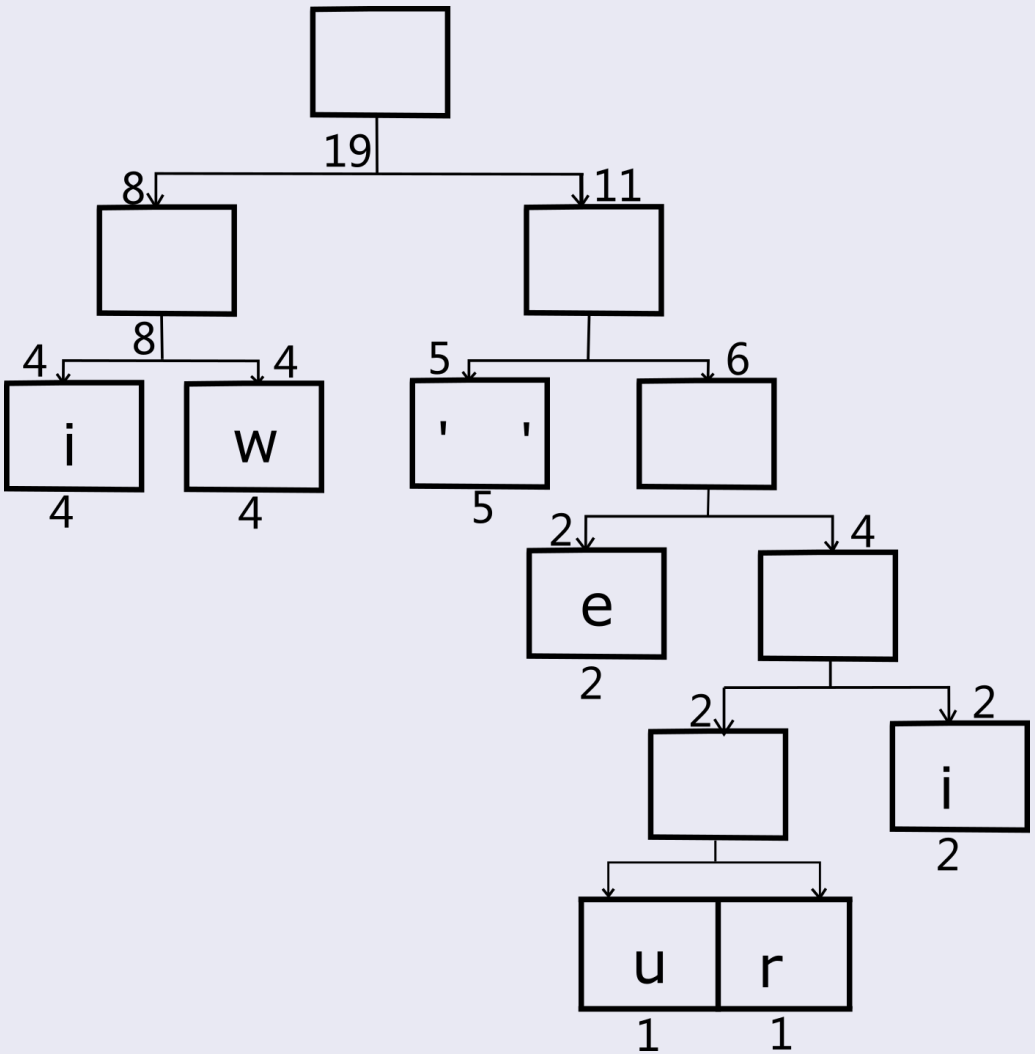
8.6 哈夫曼编码

依次从左到右合并，每合并一次则进行一次队列重新排序调整。如下：



8.6 哈夫曼编码

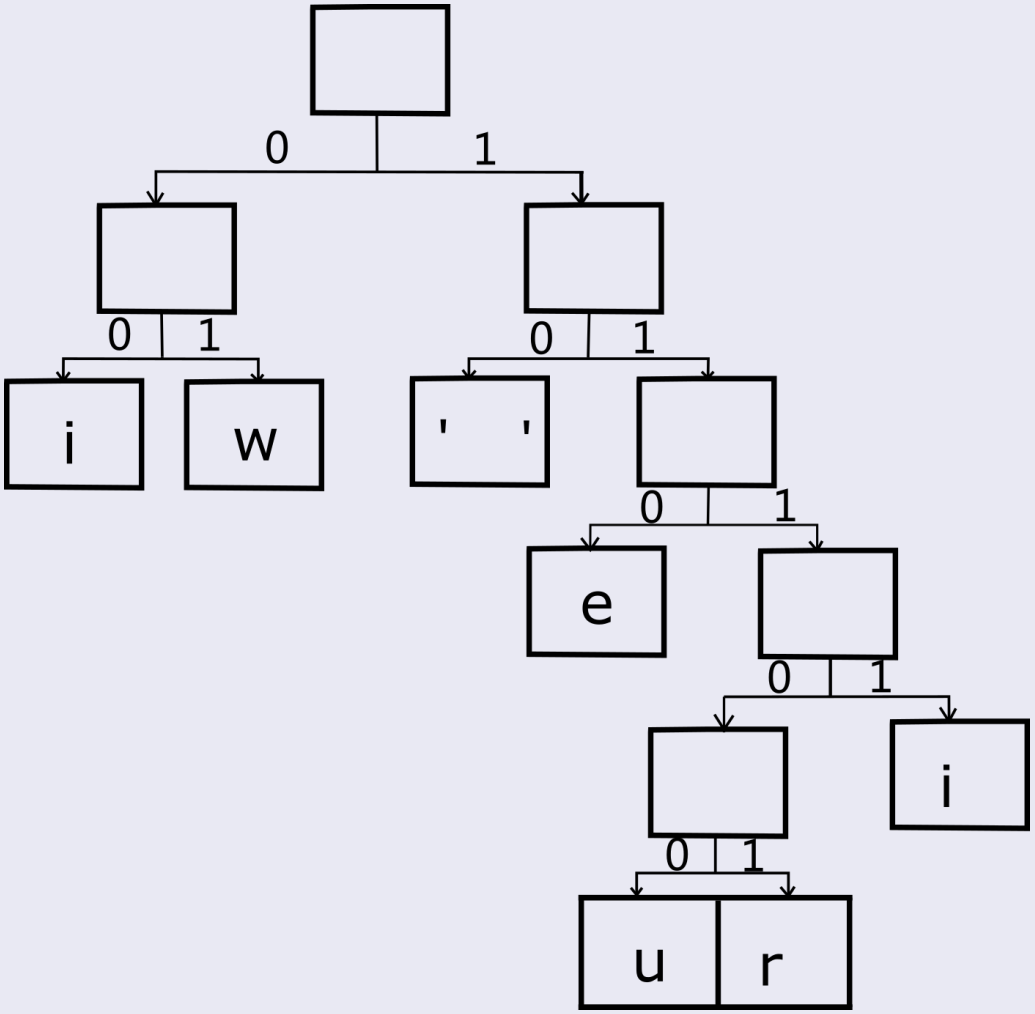
经过多步操作后，得到以下的哈夫曼树，也就是一个带有权重的二叉树：



8.6 哈夫曼编码

2.4 哈夫曼编码

根据上面带权重的哈夫曼树，就可以进行编码。设置二叉树分支中左边的支路编码为 0，右边分支表示为 1，如下图



8.6 哈夫曼编码

经过这个编码设置之后可以发现，出现频率越高的字符越会在上层，这样它的编码越短；出现频率越低的字符越会在下层，编码较长。经过这样的设计，最终整个文本存储空间才会最大化地缩减。
最终可得到下面的编码表：

字符	i	w	' '	e	i	r	u
编码	00	01	10	110	1111	11101	11100

2.5 字符串编码
有了上面的编码表之后，”we will we will r u”这句进行哈夫曼编码就可以得到很大的压缩，哈夫曼编码表示为：01 110 10 01 1111 00 00 10 01 110 10 01 1111 00 00 10 11101 10 11100。这样最终只需 50 位内存，比 ASCII 码编码表示节约了 2/3 空间，效果很理想。
当然现实中不是简单这样表示的，还需要考虑很多问题。

8.6 哈夫曼编码

3 补充
哈夫曼树，它是带权路径最小的二叉树，也叫最优二叉树。
它不一定是完全二叉树，也不一定是平衡二叉树，它们描述的完全不是一件事情，完全没有概念上的重叠关系。

3.1 哈夫曼编码习题
假设某符号集 X 中包含 7 个符号：(s1，s2，s3，s4，s5，s6，s7)，它们各自出现的概率分别为：(0.31，0.22，0.18，0.14，0.1，0.04，0.01)。试求集合中各符号的哈夫曼编码?