哈夫曼树
哈夫曼编码

通常的编码方法有等长编码和不等长编码两种。

等长的编码：所有字符的编码长度相等，n个不同的字符需要$\lceil \log n \rceil$位编码。

不等长编码：经常使用的字符编码较短，不常用的字符编码较长。

最优编码方案是指编码总长度最短。不等长编码方法需要解决两个关键问题：

1）编码尽可能短。

2）不能有二义性（前缀码特性）。

扫码

不等长编码的例子我们可以借助“五笔输入法”来理解
五笔输入法就是利用了这个原理，中文的常用字使用的编码少，不常用的字使用的编码多

关于二义性：即一个编码翻译出了两种答案，我们称这种情况为二义性
前缀码：是在有效字符前加的通用型代码
前缀码特性：任何一个字符的编码都不能是其他字符编码的前缀
具有前缀码特性的编码即为前缀码（名字有歧义）

将所要编码的字符作为叶子节点，将该字符在文件中的使用频率作为叶子节点的权值，以自底向上的方式，通过$n-1$次“合并”构造哈夫曼树。

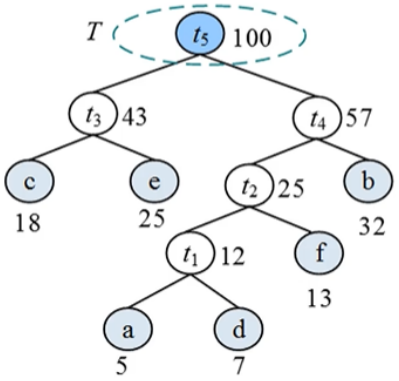
哈夫曼编码的核心思想：权值大的叶子离根近。

哈夫曼算法的贪心策略：每次从树的集合中取出没有双亲且权值最小的两棵树作为左右子树，构造一棵新树，新树根节点的权值为其左右孩子节点权值之和，并将新树插入树的集合中。

根据以下字符频率构造一棵哈夫曼树。

字符	a	b	c	d	e	f
频率	0.05	0.32	0.18	0.07	0.25	0.13

约定左分支上的编码为“0”，右分支上的编码为“1”。从根节点到叶子节点路径上的字符组成的字符串为该叶子节点的哈夫曼编码。



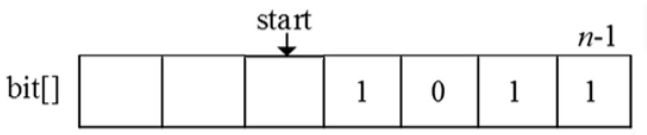
数据结构设计

```
typedef struct{ //节点结构体
    double weight; //权值
    int parent; //双亲
    int lchild; //左孩子
    int rchild; //右孩子
    char value; //字符信息
}HNodeType;
```

weight	parent	lchild	rchild	value

数据结构设计

```
typedef struct{ //编码结构体
    int bit[MAXBIT];
    int start;
}HCodeType;
```



	weight	parent	lchild	rchild	value
0	5	−1	−1	−1	a
1	32	−1	−1	−1	b
2	18	−1	−1	−1	c
3	7	−1	−1	−1	d
4	25	−1	−1	−1	e
5	13	−1	−1	−1	f
6	0	−1	−1	−1	
7	0	−1	−1	−1	
8	0	−1	−1	−1	
9	0	−1	−1	−1	
10	0	−1	−1	−1	

关于哈夫曼编码的算法复杂度分析:

*-时间复杂度 O(n^2)

*-空间复杂度 O(n*MAXBIT)

可以优化:

① 使用 Priority_Queue 容器

② 使用 Vector 容器

可变基哈夫曼编码

一般来讲，哈夫曼编码都是二进制的，即都是用 0 和 1 进行编码，也可以说是用二进制进行编码。

不等长编码方法需要解决两个关键问题：

1）编码尽可能短。

2）不能有二义性（前缀码特性）。

最优编码方案是指编码总长度最短。

哈夫曼编码的**核心思想**：权值大的叶子离根近，将所要编码的字符作为叶子节点，将该字符在文件中的使用频率作为叶子节点的权值，以自底向上的方式，通过$n-1$次“合并”构造哈夫曼树，然后进行哈夫曼编码。

扫码