

赫夫曼编码

一、基本介绍

- 1) 赫夫曼编码也翻译为 哈夫曼编码(Huffman Coding)，又称霍夫曼编码，是一种编码方式，属于一种程序算法
- 2) 赫夫曼编码是赫哈夫曼树在电讯通信中的经典的应用之一。
- 3) 赫夫曼编码广泛地用于数据文件压缩。其压缩率通常在 20%~90%之间
- 4) 赫夫曼码是可变字长编码(VLC)的一种。Huffman于 1952年提出一种编码方法，称之为最佳编码

二、原理剖析

通信领域中信息的三种处理方式

1、定长编码

- i like like like java do you like a java // 共40个字符(包括空格)
- 105 32 108 105 107 101 32 108 105 107 101 32 108 105 107 101 32 106 97 118 97 32 100 111 32 121 111 117 32 108 105 101 32 97 32 106 97 118 97 //对应Ascii码
- 01101001 00100000 01101100 01101001 01101011 01100101 00100000 01101100 01101001 01101011 01100101 00100000 01101010 01100001 01101100 01100001 00100000 01100100 01101111 00100000 01111001 01101111 01101010 00100000 01101100 01101001 01101011 01100101 00100000 01100001 00100000 01101010 01100001 01101100 01100001 // 对应的二进制
- 按照二进制来传递信息，总的长度是 359 (包括空格)
- 在线转码 工具：<https://www.mokuge.com/tool/asciito16/>

2、变长编码

- i like like like java do you like a java // 共40个字符(包括空格)
- d:1 y:1 u:1 j:2 v:2 o:2 l:4 k:4 e:4 i:5 a:5 :9 // 各个字符对应的个数
- 0= , 1=a, 10=i, 11=e, 100=k, 101=l, 110=o, 111=v, 1000=j, 1001=u, 1010=y, 1011=d
说明: 按照各个字符出现的次数进行编码, 原则是出现次数越多的, 则编码越小, 比如空格出现了9次, 编码为0, 其它依次类推.
- 按照上面给各个字符规定的编码, 则我们在传输 "i like like like java do you like a java" 数据时, 编码就是
10010110100...
- 字符的编码都不能是其他字符编码的前缀, 符合此要求的编码叫做前缀编码, 即不能匹配到重复的编码(这个在赫夫曼编码中, 我们还要进行举例说明, 不捉急)

3、赫夫曼编码

步骤如下;

- 1) 传输的 字符串: i like like like java do you like a java
- 2) 计算各个字符对应的个数 d:1 y:1 u:1 j:2 v:2 o:2 l:4 k:4 e:4 i:5 a:5 :9
- 3) 按照上面字符出现的次数构建一颗赫夫曼树, 次数作为权值
- 4) 根据赫夫曼树, 给各个字符, 规定编码 (前缀编码), 向左的路径为 0 向右的路径为 1
o: 1000 u: 10010 d: 100110 y: 100111 i: 101
a : 110 k: 1110 e:
1111 j: 0000 v: 0001
l: 001 : 01

5) 按照上面的赫夫曼编码, 我们的 "i like like like java do you like a java" 字符串对应的编码为 (注意这里我们使用的无损压缩)

10101001101111011110100110111101111010011011110111101000011000011100110011110000110
01111000100100100110111101111011100100001100001110

通过赫夫曼编码处理 长度为 133

PS: 此编码满足前缀编码, 即字符的编码都不能是其他字符编码的前缀。不会造成匹配的多义性

赫夫曼编码是无损处理方案

6) 注意事项

注意, 这个赫夫曼树根据排序方法不同, 也可能不太一样, 这样对应的赫夫曼编码也不完全一样, 但是 wpl 是一样的, 都是最小的, 最后生成的赫夫曼编码的长

度是一样，比如：如果我们让每次生成的新的二叉树总是排在权值相同的二叉树的最后一个