中文多模匹配三种方式比较

一.普通字典树多模匹配

1. 设计思路:因为每个字节共有8位，所以不同字节表示的状态最多为256个，所以将我们的中文模式串 按0到255的每一个字节建成字典树。

   (1). 在文本里读入200行模式串，读入一个较大的文本串进行匹配。

   (2). 当一个模式串插入完毕时，标记此时的节点为结束位，并将中文模式串的信息存储到此时节点的字符指针位置。

   (3). 查找时遍历文本串的每一位，如果匹配成功则输出当前找到的模式串。

2. 预计效率:假设模式串中256种字节出现的频率大概一样，那么效率公式应该为：

   设模式串总长度为n，文本串总长度为m，字典树叉个数为x，每一个节点的空间为P，log为预计树深.

   $$x = 256$$

   则：查找计算公式为

   $$T = n/\log_x m * m$$

   则：空间计算公式为

   $$S = n/256 * P$$

3. 实现过程及代码：见附录。

4. 运行结果及实际效率：见附录。

二叉字典树多模匹配

1. 设计思路:将每一个0到255的字节转化成对应的8位01编码，进行普通的字典树插入。

   (1).进行文件读取，插入模式串。

   (2). 当一个模式串按01编码的方式插入完毕时，标记此时的节点为结束位，并将模式串的字符串信息存储到此时节点的字符指针位置。

   (3). 将文本串的每一个字节转换成01编码，并去每一位01匹配，查找时遍历文本串的01形式的每一位，如果匹配成功则输出当前找到的模式串。

2. 预计效率：二叉字典树需要比256的节点多8倍，因为需要转换成01编码假设模式串中256种字节出现的频率大概一样，那么效率公式应该为：

   设模式串总长度为n，文本串总长度为m，字典树叉个数为x，每一个节点的空间为P，log为预计树深.

   $$x = 2$$

   则:查找效率为：

   $$T = n/\log_x n * m * 8$$

   则:空间效率为：

   $$S = n/2 * P * 8$$

3. 实际过程及代码:见附录。
4. 运行结果及实际效率:见附录。

三.哈弗曼加二叉字典树

1. 设计思路:建立一个50M的中文语料，统计0到255字节的词频，依据词频建立哈弗曼树，得到每一个字节对应的哈弗曼01编码，这样频次高的字节对应的01编码会尽可能的少，然后进行字典树操作。

   (1).进行文件读取，插入模式串。

   (2). 当一个模式串按对应字节的哈弗曼编码的方式插入完毕时，标记此时的节点为结束位，并将模式串的字符串信息存储到此时节点的字符指针位置。

   (3). 将文本串的每一个字节按哈弗曼编码转换成01编码，并去每一位01匹配，查找时遍历文本串的01形式的每一位，如果匹配成功则输出当前找到的模式串。

2. 预计效率: 计算01编码长度时按哈弗曼编码计算，并按平均码长

   (1):哈弗曼平均码长计算:平均码长=码长*出现频率。

   设模式串总长度为n，文本串总长度为m，字典树叉个数为x，每一个节点的空间为P，log为预计树深，平均码长为l.

   $$x = 2$$

   查找效率:

   $$T = n/\log_x n * m * l$$

   空间效率:

   $$S = n/2 * P * l$$

3. 实际过程及代码:见附录。
4. 运行结果及实际效率:见附录。

四.效率对比:

1. 预期效率对比:

   (1).空间效率：哈弗曼加二叉树 > 二叉树 > 普通字典树

   (2).查找效率：普通字典树 > 哈弗曼加二叉树 > 二叉树

   2. 实际效率对比:

```
(1).空间效率：和预期一样
```

```
(2).查找效率：和预期一样
```

五.问题分析:

测了3,4组，虽然哈弗曼的查找效率比二叉树查找效率快，但是快的并不尽人意，分析一下可能是选用的语料库不太友好，哈弗曼表建立之后平均码长应该是大于一个字节所应有的长度了，选用的模式串可能是频率比较低的字节。但是普通字典树的查找很快，可是缺点是空间利用率太低，可以用双数组改进，普通字典树的查找效率约是二叉树的50倍，是哈弗曼的30倍，空间利用率哈弗曼大概是普通二叉树的2倍，是普通字典树的15倍。