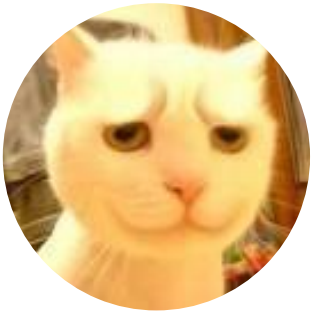


HuangWei

That depends a good deal on where you want to get to

主页

所有文章

好文收藏

超酷算法：字谜树

2015-1-1

string tree trie

algorithm

这篇文章是在没有搭建这个Blog之前帮jobbole翻译的，现在只是复制回来自己做个存档，jobbole链接在这。

我毫不犹豫的把这个算法称为“超酷”，虽然我自己发明了它，但我还是觉得它相当的酷，而且它很适合我算法系列的主题，所以无论如何要把它写下来。

当涉及到寻找单词字谜时，用的比较频繁的方法是字谜字典，简单得说，对单词的字母进行排序，以提供一个单词与所有字谜共同点的唯一索引。另外一种方法是为单词里的每个字母生成字母频率直方图。（这两种方法实际上或多或少相同。）这些方法查找确切的单字字谜

HuangWei

That depends a good deal on where you want to get to

字符串非常高效 - 如果使用哈希表，复杂度为O(1)。

然而，如果问题是查找字谜的子集（包含一个字符串里字母的一个子集的单词），仍然是相当低效的，通常需要在O(n)时间内暴力搜索整个字典，或者查找每个有序字符串的子串，复杂度与输入字符串的字母长度有关，为O(2^l)。查找字谜子集显然更有趣，因为它能查找多字字谜，可以应用在拼字游戏上。

不管怎样，我们先生成能唯一表示一组字母的直方图，再努力观察，我们可以生成一个树结构来更有效得查找字谜子集。为了构建这样的树，我们按照如下几个简单的步骤：

假设我们有如下信息：

- 一个词典或单词字典来填充树
- 词典中单词的字母表
- 一个正在构建的树
- 当前节点

词典里的每个单词：

1.为该单词生成字母频率直方图。

2.设当前节点为树的根节点。

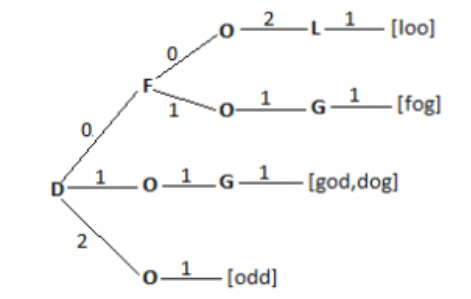
3.每个字母表里的字符：

获取当前字符在当前单词里的频率，记为f

设置当前节点为当前节点的第f个子节点，如果它不存在的话就创建

4.将当前单词添加到当前（叶）节点上的单词列表

以下是这个简单过程的结果，它是一棵固定高度的树，27个节点深，所有单词都在叶节点中，并且树的每个层级对应字母表里的字符。下面是个简略的例子（译注：原博客图片遗失，从WIKI上找了张替图）：



HuangWei

That depends a good deal on where you want to get to

一旦树创建好后，我们可以如下方式查找输入字符串的字谜集合：

假设我们有如下信息：

由上述流程所构建的树

上面使用过的字母表

一个边界集合，初始化为空

1.初始化时边界集合只包含树的根节点

2.生成输入字符串的字母频率直方图

3.对字母表中的每个字符：

1.获取当前字符在输入字符串里的频率，记为f

2.对边界集合里的每个节点，添加标号为0到f的子节点到新的边界集合中

4.当前边界集合中包含的叶节点，包含所有输入字符串的字谜子集

至少对我来说，对该算法进行运行期分析比较困难。直观的看，它在实践中比任何一种蛮力算法要快很多，但我无法量化为大O表示法。

作为一个上限，它不可能比O(n)的效率低，最坏也比蛮力算法少一个常数因子。作为下限值，边界集合中只有一个节点，那查找时间就与字母表长度成正比，为O(1)。平均情况下，依赖输入字符串所选择的字典的子集有多大。以输出的大小来量化的话，需要O(m)的操作。如果有人知道如何确定运行时更准确的范围的话，请在评论中让我知晓。

这个算法有个缺点就是，需要大量的内存开销。我用python来实现，并导入/usr/share/dict/words，在本机上这大约是2MB的大小，但需要占用内存300MB。使用Pickle模块序列化到磁盘，输出文件的大小超过30MB，使用gzip压缩后下降到大约7MB。我怀疑内存大的部分原因是python字典的最小尺寸。我将使用列表来实现，如果我能够做到更高效，届时我会更新这篇文章。

HuangWei

That depends a good deal on where you want to get to

这里是上述所生成树的数据，可能你会感兴趣：

总单词数：234,936

叶节点：215,366

内部节点：1,874,748

由此我们可以看出，内部节点的平均基数是非常低的，不会大于1。下面数据有助于澄清：

Tier	Number of nodes
0	1
1	7
2	25
3	85
4	203
5	707
6	1145
7	1886
8	3479
9	8156
10	8853
11	10835

12	19632
13	28470
14	47635
15	73424
16	92618
17	94770
18	125018
19	156406
20	182305
21	195484
22	200031
23	203923
24	205649
25	214001

靠近树的顶部节点的基数非常高，但树很快变平，最后四层树只占总结点的一半。这暗示了一个可能的空间优化：删除树的最后几层，将它们的叶子节点连在一起。当进行查找时，检查所选的节点，保证它们是输入字符串的字谜集合。

- 我可能只是重新发现了计算机科学领域30年前就被提及的论文。但惊喜的是，通过搜索尚未找到谁正在使用该算法，或者有其它方法比蛮力算法更有效。

修订：最初的实现代码在这。

修订：使用列表来重新实现我的python代码，几乎节约了一半内存。有机会我会贴出pickled后的树和源码。

修订：更多更新在这。

（译注：字谜问题可简化为字符串编码和索引问题，如Tea编码为A1E1T1，编码哈希后，同编码单词有Ate，Eat等。文章写于2007年，文中算法不是最优解，只是提供了一种使用多路查找树的思路，类似数据结构有Trie，DAG，Suffix Tree等等。）

< 创业小结（一）

HuangWei

That depends a good deal on where you want to get to