数据结构与算法（Python版）

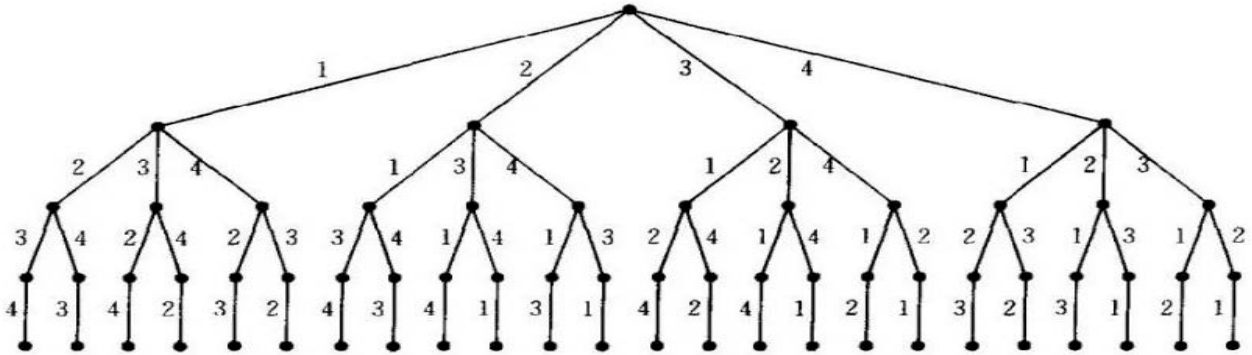
“变位词”判断问题（下）

陈斌 北京大学 gischen@pku.edu.cn

解法3：暴力法

- 暴力法解题思路为：穷尽所有可能组合
- 将s1中出现的字符进行全排列，再查看s2是否出现在全排列列表中
- 这里最大困难是产生s1所有字符的全排列

  根据组合数学的结论，如果n个字符进行全排列，其所有可能的字符串个数为n!



解法3：暴力法

- 我们已知 n! 的增长速度甚至超过2^n

  例如，对于20个字符长的词来说，将产生

  20!=2,432,902,008,176,640,000个候选词

  如果每微秒处理1个候选词的话，需要近8万年时间来做完所有的匹配。

- 结论：暴力法恐怕不能算是个好算法

解法4：计数比较

- 解题思路：对比两个词中每个字母出现的次数，如果26个字母出现的次数都相同的话，这两个字符串就一定是变位词
- 具体做法：为每个词设置一个26位的计数器，先检查每个词，在计数器中设定好每个字母出现的次数
- 计数完成后，进入比较阶段，看两个字符串的计数器是否相同，如果相同则输出是变位词的结论

解法4：计数比较-程序代码


```
1   def anagramSolution4(s1, s2):
2       c1 = [0] * 26
3       c2 = [0] * 26
4       for i in range(len(s1)):
5           pos = ord(s1[i]) - ord('a')
6           c1[pos] = c1[pos] + 1
7       for i in range(len(s2)):
8           pos = ord(s2[i]) - ord('a')
9           c2[pos] = c2[pos] + 1
10      j = 0
11      stillOK = True
12      while j < 26 and stillOK:
13          if c1[j] == c2[j]:
14              j = j + 1
15          else:
16              stillOK = False
17      return stillOK
18
19
20  print(anagramSolution4('apple', 'pleap'))
```

分别都计数

计数器比较

解法4：计数比较-算法分析

- 计数比较算法中有3个循环迭代，但不象解法1那样存在嵌套循坏
  - 前两个循环用于对字符串进行计数，操作次数等于字符串长度n
  - 第3个循环用于计数器比较，操作次数总是26次
- 所以总操作次数T(n)=2n+26，其数量级为O(n)
  - 这是一个线性数量级的算法，是4个变位词判断算法中性能最优的

解法4：计数比较-算法分析

- 值得注意的是，本算法依赖于两个长度为26的计数器列表，来保存字符计数，这相比前3个算法需要更多的存储空间

  如果考虑由大字符集构成的词（如中文具有上万不同字符），还会需要更多存储空间。

- 牺牲存储空间来换取运行时间，或者相反，这种在**时间空间之间的取舍**和权衡，在选择问题解法的过程中经常会出现。

  “不可随处小便”，“小处不可随便”