KMP 与 EX · KMP

Ⅰ. KMP

一. 时间复杂度：O(n)

证明[1]：复杂度唯一有疑问的地方就只有构造时的那个 while 循环。不难发现，原串的每一个字符最多只能让 j[2] += 1，而原串只有 n 个字符，所以对于整个算法来讲，j 最多减少 n 次。所以总的时间复杂度仍然是 O(n)的

二. 代码参考：http://paste.ubuntu.com/17905489/[3]

三. 相关应用：

①循环节问题：

这个是 KMP 比较常考的性质，紫书也有讲，但值得注意的是：循环节不一定得整出字符串长度。比如 BZOJ_1355。以其样例数据(cabcabca)推导如下：

c a b | c a b | c a b |
c a b | c a b | c a b

图一

不难发现，即使循环节不能整除，但我们一定能够通过增添小于循环节长度的个数的字符将其补全（如红色部分）

②原串前缀的前后缀问题

仔细观察 KMP 的性质不难发现，next[i]其实是原串 S 的前缀 $S_{1\sim i}$ 的最大相同前后缀[4]。不好理解的话，还请参看题

---

[1] 匹配部分复杂度十分显然，这里仅提供构造失配函数的复杂度证明
[2] 这里所提到的变量均与下文提到的参考代码对应
[3] 仅构造适配函数
[4] 顺着 fail[]一直跳下去，即可从大到小遍历所有相同的前后缀

目:BZOJ_ 3670

③倍增相关问题

我们知道倍增算法常用于树上问题。也就是对于无法做到随机访问的数据结构，我们也可以进行倍增。更进一步：KMP 的 nxt[]数组，实际上就是树状结构。所以理论上倍增也是可以用于 KMP 的。[5]

Ⅱ. EX・KMP[6]

一. 用途：求原串 S 的每一个后缀与模板串 T 的前缀的最长公共部分

二. 学习推荐：

算法学习：http://wenku.baidu.com/view

题目练习：http://www.cnblogs.com/Rlemon/p/3157898.html

三. 代码参考：http://blog.csdn.net/acdreamers[7]

[5] 暂时没有找到相关题目，在这里仅提供一道可以在 SAM 的 fail 链上倍增的题目：http://www.lydsy.com

[6] 因为其所做的工作，SA+ST 表可以替代，所以这里仅略谈一二

[7] 非本人代码