数据结构和算法
(Python描述)

郭 炜

微信公众号

微博：http://weibo.com/guoweiofpku

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄

KMP
字符串匹配算法

法国勃朗峰

字符串匹配

给定一个模式串（子串）和一个母串，求模式串在母串中出现的位置。母串中找不到模式串位置就算-1。

暴力字符串匹配算法

引入"母串匹配起点"(MS)概念。母串a,子串b，若a[i]和b[0]比较，则称"母串匹配起点"为a[i]。

设置一个母串指针，指向母串中待比较的字符；设置一个模式串指针，指向子串中待比较的字符。比较母串指针和子串指针指向的字符，如果相等，则两个指针都加1。

如果不相等......

暴力字符串匹配算法

暴力算法母串指针需要回溯。假设子串前n-1个字符已经被匹配，第n个失配：

MS

母串： $a_0a_1a_2\ldots a_{n-1}a_n\ldots$

子串： $b_0b_1b_2\ldots b_{n-1}b_n\ldots$

母串匹配起点+1，母串指针回溯到母串匹配起点：

MS

母串： $a_0a_1a_2\ldots a_{n-1}a_n\ldots$

子串： $b_0b_1\ldots b_{n-2}b_{n-1}\ldots$

KMP字符串匹配算法

不希望母串指针回溯!

母串指针如果不回溯，直接用a_n和b_0比，可能就会忽略母串匹配起点为a_3时得到的下面的情况：

母串： $a_0a_1a_2a_3a_4\cdots a_{n-1}a_n\cdots$

子串： $b_0b_1\cdots b_{k-1}b_k\cdots b_n\cdots$

$a_3a_4....a_{n-1}$ 和 $b_0b_1 ...b_{k-1}$ 可以匹配上，值得再做下去

关键：要在母串指针不回溯的情况下避免忽略上述情况。

KMP字符串匹配算法

➢ 概念：字符串的前缀和后缀

$b_0b_1 \ldots b_{n-1}\, b_n$的前缀是 $b_0b_1 \ldots b_k$ (k=0,1,2...n)

$b_0b_1 \ldots b_{n-1}\, b_n$的真前缀是 $b_0b_1 \ldots b_k$ (k=0,1,2...n-1)

$b_0b_1 \ldots b_{n-1}\, b_n$的后缀是 $b_k \ldots b_{n-1}b_n$ (k=0,1,2...n)

$b_0b_1 \ldots b_{n-1}\, b_n$的真后缀是 $b_k \ldots b_{n-1}b_n$ (k=1,2...n)

KMP字符串匹配算法

若发生了下述情况：

母串： $a_0a_1a_2a_3a_4 \cdots a_{n-1}a_n \cdots$

子串： $b_0b_1 \cdots b_{k-1}b_k \cdots b_n \cdots$

$a_3a_4 \ldots a_{n-1}a_n$ 和 $b_0b_1 \ldots b_{k-1}\, b_k$可以匹配上

则 $b_0b_1 \ldots b_{k-1}$ 是子串的前缀，也是$a_0a_1 \ldots a_{n-1}$的后缀

由于$a_0a_1 \ldots a_{n-1} = b_0b_1 \ldots b_{n-1}$ ，因此$b_0b_1 \ldots b_{k-1}$也是$b_0b_1 \ldots b_{n-1}$的后缀

若有$b_0b_1 \ldots b_{k-1}$是$b_0b_1 \ldots b_{n-1}$的**真**后缀，则称$b_0b_1 \ldots b_{k-1}$是字的符b_n的**前后缀**。

KMP字符串匹配算法

发生了下述情况时：

母串： $a_0a_1a_2a_3a_4\cdots a_{n-1}a_n\cdots$

子串： $b_0b_1\cdots b_{k-1}b_k\cdots b_n\cdots$

$a_3a_4....a_{n-1}a_n$ 和 $b_0b_1 ...b_{k-1}\ b_k$可以匹配上

字符b_n的前后缀可能不止一个，要考查最长的

即a_n和b_n失配时，直接比较a_n和b_n最长前后缀的后一个字符b_k，即母串指针不回溯，子串指针移到k。若还失配，则比较a_n和b_k的最长前后缀（即b_n的次长前后缀）后一个字符b_p；若再失配，则比较a_n和b_p最长前后缀（即b_n的第三长前后缀）的后一个字符......直到a_n和b_0比较，若还失配，则母串指针+1，子串指针回0。

KMP字符串匹配算法

发生了下述情况时：

母串： $a_0a_1a_2a_3a_4\cdots a_{n-1}a_n\cdots$

子串： $b_0b_1\cdots b_{k-1}b_k\cdots b_n\cdots$

$a_3a_4....a_{n-1}a_n$ 和 $b_0b_1 ...b_{k-1}$ b_k可以匹配上

母串匹配位置从a0跳到了a3 ，忽略了母串匹配位置在a1和a2的情况，即忽略了母串分别从a1和a2开始与子串进行比较的情况。

如果能证明母串分别从a_1和a_2开始与子串进行比较，都不会使得$a_1...a_{n-1}$被匹配成功，则忽略是合理的。

KMP字符串匹配算法

反证法：

母串：　$a_0a_1a_2a_3a_4\ldots a_{n-1}a_n\ldots$

子串：　$b_0b_1b_2b_3\ldots b_{m-1}b_m\ldots$

$a_3a_4\ldots\ldots a_{n-1}a_n$ 和 $b_0b_1\ldots b_{k-1}\ b_k$可以匹配上

假设母串分别从a_1开始与子串进行比较，$a_1\ldots a_{n-1}$和$b_0\ldots b_{m-1}$匹配成功，则 $b_0b_1b_2b_3\ldots b_{m-1}$是$a_0a_1a_2a_3a_4\ldots a_{n-1}$的后缀，即是也是子串$b_0b_1b_2b_3\ldots b_{n-1}$ 的后缀（因$a_0a_1a_2a_3a_4\ldots a_{n-1} = b_0b_1b_2b_3\ldots b_{n-1}$），即为字符$b_n$的前后缀。len($b_0b_1b_2b_3\ldots b_{m-1}$) > len($b_0b_1\ldots b_{k-1}$)，这和$b_0b_1\ldots b_{k-1}$是字符$b_n$的最长前后缀矛盾。

因此母串匹配位置为a_1时不可能成功。

KMP字符串匹配算法

用next[i]表示字符b_i的最长前后缀的长度，则字符b_i的次长前后缀的长度为next[next[i]]，第三长前后缀的长度为next[next[next[i]]].....

设b_i最长前后缀长度为k，则$b_0b_1...b_{k-1}$ 是最长前后缀，和

$b_{i-k}...b_{i-1}$相同。b_i次长前后缀就是b_k的最长前后缀

故Len(b_i次长前后缀) = next[k] = next[next[i]]

KMP字符串匹配算法

➢ 算法核心思想：

设立列表next。next[i]就是b[i]的最长前后缀的长度。

next[i]表示匹配到子串字符b[i]时，若发生了失配，则母串指针不动，子串指针应该变为next[i]，然后继续匹配，再失配，则子串指针变为next[next[i]].....

显然，next只和子串有关，和母串无关，且有 next[1] = 0
记next[0] = -1，则next[i] >= 0 (i = 1,2....)

若即b[0]失配，则母串指针+1，子串指针置为0

KMP字符串匹配算法

KMP算法是如何避免母串指针回溯的?

↓

母串:　　acabacakg......

子串:　　acabacaef...

KMP字符串匹配算法

KMP算法是如何避免母串指针回溯的?

↓

母串： acabacakg……

子串： acabacaef…

KMP字符串匹配算法

KMP算法是如何避免母串指针回溯的?

母串: acabacakg......
子串: acabacaef...

KMP字符串匹配算法

KMP算法是如何避免母串指针回溯的?

↓

母串： acabacakg……

子串： acabacaef…

KMP字符串匹配算法

KMP算法是如何避免母串指针回溯的?

母串：　aabcdaakg……

子串：　　　　　　　　acabacaef…

KMP字符串匹配算法

字符串b: acabacaef

next: [-1, 0, 0, 1, 0, 1, 2, 3, 0]

next[i]: 字符b[i]的最长前后缀长度

KMP字符串匹配算法

➢ 关键：求子串b的next列表

递推：已知next[0],next[1]....next[i],如何求next[i+1]

设next[i] = k, 则若 b[i] = b[k] 则 next[i+1] = k + 1

$b_0b_1b_2...b_{k-1}b_k...b_{i-1}b_ib_{i+1}$

若next[i] = k，说明$b_0b_1b_2...b_{k-1} = b_{i-k}...b_{i-1}$
此时若b[i]=b[k]，则$b_0b_1b_2...b_{k-1}b_k = b_{i-k}...b_{i-1}b_i$

即next[i+1] = k+1

KMP字符串匹配算法

设next[i] = k, 则若 b[i] != b[k] 则?

b[i+1]的最长前后缀，必然是以b[i]的某个前后缀加上b[i]构成
如果b[i]的最长前后缀(长度k)，加上b[i]，不能构成b[i+1]的最长前后缀
则要考虑b[i]的次长前后缀（长度为next[k])，能否加上b[i]，构成b[i+1]的最长前后缀，次长的不行，则考虑次次长前后缀(长度为next[next[k]].......