字符串匹配算法（一）

——KMP算法及拓展KMP算法

引言

- 问题的引入

  给定一字符串S，问字符串S串之中有多少子串为T

- 学习KMP之前

  BF算法（蛮力算法）

  维护两个指针$\boldsymbol{i},\boldsymbol{j}$分别指向$S$与$T$串。

  每次尝试从i起始的字符串能否与T串匹配

  若失配，返回上次起始位置，指针i向后移动一位，开始下一轮尝试。

  时间复杂度$\Theta(N \cdot M)$

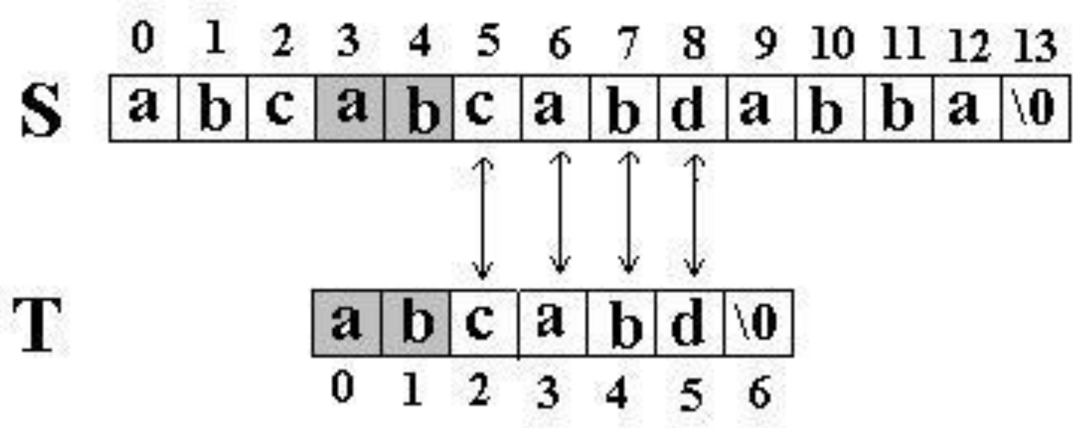
- KMP算法仅需$\Theta(N + M)$

KMP算法

- KMP的思想

  一次匹配失败时指向原串的i指针并不需要回滚，特殊处理下指向模式串的j指针即可达到效果

- 构造Next函数来指定j的回滚位置



KMP算法

- 构建NEXT函数

  模式串先与自身进行匹配,以构造nxt数组

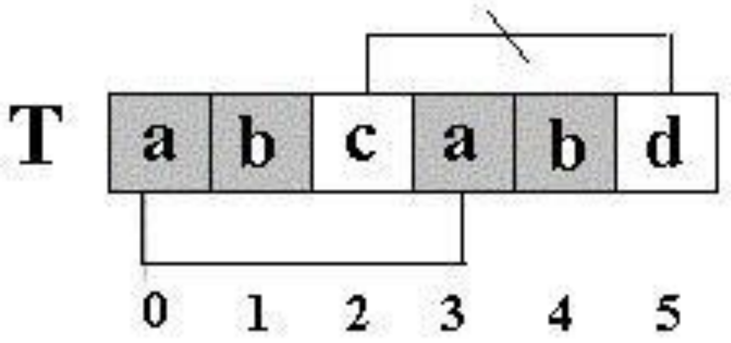
  约定：

  若$nxt[x]=y$; 则意味着在模式串下标为x的位置失配时，应将j指针移动到下标为y的位置，再进行尝试匹配。

  初始化时$nxt[0]=-1$.某时刻j指针若为-1则意味着当前位置没有进行匹配的意义，i,j可都后移一位

- NEXT函数还可求出最小循环节

- 相关代码



KMP算法

- 匹配过程

1.初始化两个指针i,j分别指向原串与匹配串的单元下标

2.若i,j所指向的字母相同或j为-1，则i,j都向后移动一位

3.否则j移动到$nxt[j]$的位置，重复步骤2直到i移动到原串的末尾

相关代码

拓展KMP

- 解决什么问题

对于一个给定的原串S与模式串T，求出对于S中每个后缀子串与模式串T的公共前缀的长度。

即最后所求得的$extend[i]=j$，表示串$S[i,len-1]$与T的公共前缀的长度为j。

由于常规的KMP算法所求得是原串S中子串为T的所有位置，实质上所求得其实是满足$extend[x]=length_T$的所有x的数目，因此拓展KMP是KMP的拓展。2333

当然拓展KMP与KMP一样，也需要实现构建$nxt[]$辅助数组。

拓展KMP

- NXT数组意义

若当前已知$extend[0]=4$即$S[0..3]$与$T[0..3]$相同，求$extend[1]$时是否可以避免从头开始匹配?

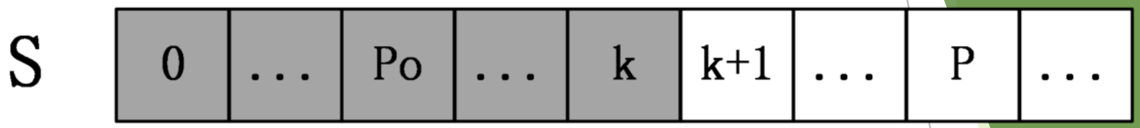
自然是可以的。

定义$nxt[i]=j$，表示$T[i..m-1]$与T的公共前缀长度为j。

回到刚刚的问题上来，若我们已知$nxt[1]=4$,即$T[1..4]$与$T[0..3]$相同，进一步得到$T[1..3]=T[0..2]$则下次前三个字符已无必要再进行匹配，直接进行$T[3]$与$S[4]$的匹配即可

nxt数组的定义避免了原串指针上无必要的回滚。

拓展KMP



拓展KMP流程

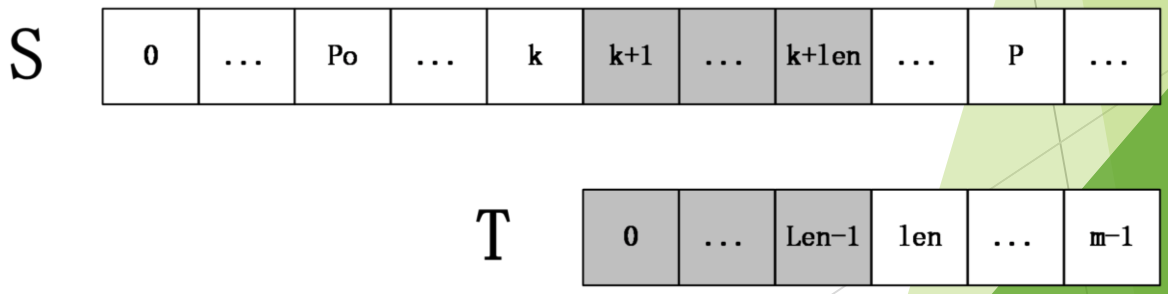
假设当前匹配向右所能到达的最远位置为P，并且取得该最大值得下标为P_o,且nxt数组已经事先构造出。当前已经计算出$extend[0..k]$的数值，现需计算$extend[k+1]$。

根据P_o与P可知$S[p_o \dots P] = T[0 \dots P-p_o]$

$$S[k+1 \dots P] = T[k+1-p_o \dots P-p_o]$$

令$len = nxt[k+1-p_o]$

若$len + k < P$, $extend[k+1] = len$



拓展KMP

- 若$len + k \ge P$,由于$S[P]$之后的字母是未知且未被处理过

S										
0	…	Po	…	k	k+1	…	P	P+1	…	…

T						
0	…	…	P−k	…	len−1	…

从$S[P+1]$,$T[P-k]$开始向下一一进行匹配，并更新最终的P与p_o的值

构造nxt数组的过程与上述相似，两个模式串T进行匹配，构建数组

相关代码