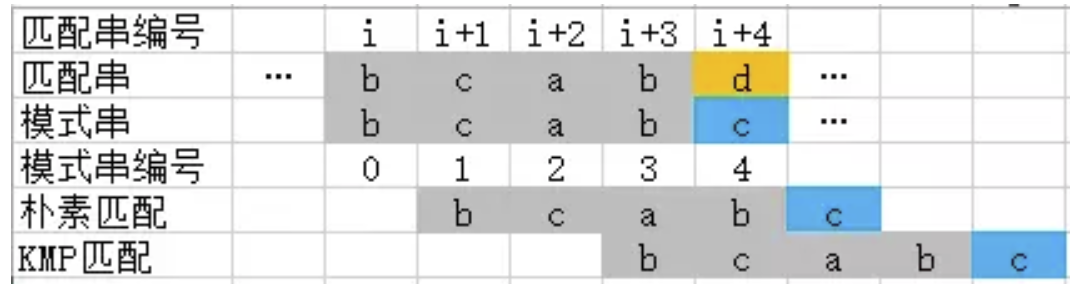
KMP算法及求解next/nextval方法简要推导

next数组含义

  
如上图所示，朴素匹配算法在匹配失败时，模式串向右移动1位。而KMP匹配则可能向右移动多位，因为灰色部分bcab中cab和ab都是以c和a开头的，不可能与b相等，KMP匹配做了个预处理（即求解next数组），使得能在此时知道移动多少位。  
下文中用s表示匹配串，p表示模式串，a[i..j]表示数组a[]的一个闭区间子序列a[i],a[i+1],...,a[j]  
当前状态：s[i-k..i-1]=p[0..k-1]，而s[i]!=p[k]。  
则j=next[k]<k代表下次将s[i]和p[j]进行比较。  
既然如此，p[j]的前缀就和s[i]的前缀必须相同，即s[i-j..i-1]=p[k-j..k-1]  
由于j<k，结合当前状态，有s[i-j..j-1]=p[0..j-1]，因为等号两边分别为s[i-k..i-1]和p[0..k-1]的前缀。  
因此有p[0..j-1]=p[k-j..k-1]，问题可以变成求解p[0..k-1]的前缀=后缀时的最长长度（这话有点绕= =），比如对"abcab"，最长长度是2，对应此时的前缀和后缀均为"ab"。

KMP算法实现

size\_t search\_kmp(const std::string& src, const std::string& pattern, size\_t pos = 0) {

auto next = get\_next(pattern); // 关键!!!

size\_t i = pos; // 匹配串当前字符序号

size\_t j = 0; // 模式串当前字符序号

while (i < src.size() && j < pattern.size()) {

if (src[i] == pattern[j]) {

i++;

j++;

} else {

j = next[j];

// j == -1即整个模式串要与s[i+1..n]进行匹配

if (j == static\_cast<size\_t>(-1)) {

i++;

j = 0;

}

}

}

// -1代表查找失败

return (j < pattern.size()) ? -1 : (i - pattern.size());

}

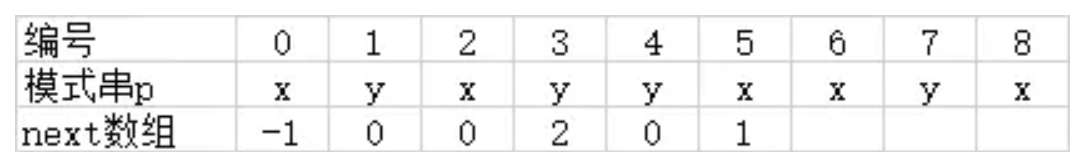
从上述代码中可以进一步看到next数组的作用，于是问题关键就在于求解next数组，这也是很多笔试题只要求算next数组的原因。

next数组求解方法

朴素的求法是找到所有等长前缀和后缀，然后一一比较。但无疑这种做法效率极其低下的。这里用数学归纳法可以推导递推式。

next[0]=-1，next[1]=0。因为如果模式串第1位p[0]就匹配失败，那么就会向右移动1位，p[0]与s[i+1]比较，等价于p[-1]与s[i]比较。而p[1]匹配失败时，会用p[0]和s[i]进行比较。

设next[k]=j，则有p[0..j]=p[k-j..k]，且不存在更大的j'使得p[0..j ']=p[k-j'..k]。现在求解j'=next[k+1]，分类讨论  
2.1 p[j+1]=p[k+1]，则有p[0..j+1]=p[k-j..k+1]，因此next[k+1]=next[k]+1。  
2.2 p[j+1]!=p[k+1]，这里就是求解next的关键部分了。此时可以把p[0..k+1]看成匹配串，p[k+1-j'..k+1]看出模式串，该模式串等于p[0..j'-1]。因此p[0..j'-2]=p[k-j'..k]，可以用同样的方法来滑动该模式串。  
比如

  
现在求解next[6]，可以发现p[2]!=p[6]，然后就可以再比较p[0]和p[6]。

next数组求解实现

inline std::vector<int> get\_next(const std::string& pattern) {

int n = pattern.size();

if (n == 0)

return {};

if (n == 1)

return { -1 };

std::vector<int> next(n);

next[0] = -1;

next[1] = 0;

int k = next[1];

for (int i = 2; i < n; i++) {

if (pattern[k] == pattern[i - 1]) {

k = next[i] = next[i - 1] + 1;

} else {

while (true) {

k = next[k];

if (k == -1 || pattern[k] == pattern[i - 1])

break;

}

next[i] = ++k;

}

}

return next;}

注意while语句部分，可以简化成像严蔚敏书上伪码一样，但是不如上面代码那么直观。  
至于考题上由于字符串下标一般从1开始，所以next数组的每个值都要加1。

nextval数组

nextval数组和next数组的关系如下

if (p[i] != p[next[i]])

nextval[i] = next[i];

else

nextval[i] = nextval[next[i]];