POJ3461（HDOJ1686）【基础】

题目大意：
给出 T 对字符串，其中包含一个长度不超过 10000 个字符的模式串和一幕长度不超过 1000000 个字符的文本串，求模式串在文本串中出现了多少次。

输入：
第一行有一个整数 T，接下来有 T 对字符串，每一对字符串第一个是模式串，第二个是文本串。

输出：
每一对字符串输出一行，包含一个整数，即模式串在文本串中出现的次数。

题解：
KMP 的模板题。关于 KMP 的原理，推荐 Matrix67 的文章 http://www.matrix67.com/blog/archives/115，这篇文章写得很清楚。我的模板代码是从红书上改过来的，其中 res[0]保存的是模式串出现的次数，res[i]（i>=1）保存的值是模式串第 i 次出现的位置。

HDOJ1771【基础】

题目大意：
有 ab 两个数组，每个数组的元素不超过 1000000 个元素，每个元素的值在[-1000000,1000000]中。问 b 是否在 a 中出现，如果出现，第一次出现的位置是哪里。

输入：
第一行有一个整数 T，表示有 T 组测试数据。每一组测试数据第一行有两个整数 len1、len2，分别表示数组 a 和 b 的元素个数。第二行有 len1 个空格分隔的整数，为 a 的第 1 到 len1 个元素。第三行有 len2 个空格分隔的整数，是 b 对应的元素。

输出：
每一组测试数据输出一行，包含一个整数，即 b 第一次在 a 中出现的位置。如果 b 不是 a 的子串，输出-1。

题解：
KMP 的裸模板题。

POJ1226（HDOJ1238）【基础】

题目大意：
给出 N 个（1<=N<=100）长度不超过 100 个字符的字符串，求这些字符串中最长的一个子串，满足此子串或者其倒序在所有的 N 个字符串中都出现。

输入：
第一行有一个整数 T，表示有 T 组测试数据。每一组测试数据第一行有一个整数 N，接下来有 N 行，每一行一个字符串。

输出：
每一组测试数据输出一行，包含一个整数，表示最长的子串的长度。

题解：
枚举+KMP。由于这个子串在所有字符串中都出现了，所以可以直接从第一个字符串中枚举子串，然后求这个子串及其逆序是否在其他字符串中出现即可。

POJ3080【基础】

题目大意：
给出 N 个（1<=N<=10）个长度为 60 的 DNA 串，要求出最长的一个子串，满足这个子串在所有的 DNA 串中都出现，且子串长度要大于等于 3。当两个子串长度相同时，输出字典序较小的一个。

输入：
第一行有一个整数 T，表示有 T 组测试数据。每一组测试数据第一行有一个整数 N，接下来有 N 行，每一行有一个长度为 60 个字符的 DNA 串。

输出：
每一组测试数据输出一行，包含一个字符串：如果找到满足要求的子串，输出该子串，否则输出 no significant commonalities。

题解：
数据很小，直接枚举。和 POJ 1226 类似，枚举子串，和其他串进行匹配。子串长度从大到小，每找到一个符合的就和当前保存的答案进行对比，然后决定是否保存即可。

类似的还有一题 POJ 3450，只不过是字符串长度最长不超过 200 个字符，N 最大不超过 1000，以及更改了数据的输入输出格式。一样可以直接枚举，而且代码基本改动不大，不另写题解，仅附上代码。

HDOJ2203【基础】

题目大意、输入输出要求见原题的中文描述。

题解：
由于 s1 可以循环移位，所以把 s1 的前 strlen(s2)位拼接到 s1 最后面，就可以满足 s1 循环移位后可能包含 s2 的情况了。接下来就是裸的 KMP 了。

HDOJ2087【基础】

题目大意、输入输出要求见原题的中文描述。

题解：
KMP 的一个小改动。由于这一次匹配后被匹配的内容要从文本串中除去，所以直接让模式串在匹配成功后不要回溯而直接重新指向开头（指向 0），就可以了。

POJ2406【中等】

题目大意：
有一个 N 个字符（1<=N<=1000000）组成的字符串，问这个串最多是由多少个相同的串重复而来的。

输入：
有若干组测试数据。每一组测试数据有一行，包含一个要求解的字符串。如果字符串只有一个字符'.'，表示测试数据结束。

输出：
每一组测试数据输出一行，包含一个整数，即最多是多少个相同的串重复而来。

题解：

这题是 KMP 里面的 Next 函数的应用，简直神奇。回顾一下 next[]数组的性质，next[i]=k 表示当模式串匹配到第 i 位无法继续的时候，需要回退到第 k 位，也就是 pattern[1]～pattern[k]=pattern[i-k]～pattern[i]。现在我们已经把目标串全部求得了 Next[]，设目标串的长度为 len，设 Next[len]的值为 k，则由上面的性质，pattern[1]～pattern[k]=pattern[len-k]～pattern[len](*)，设 len-k=x 那么 pattern[k-x]～pattern[k]=pattern[len-x]～pattern[len]（(*)等号两边的两个串的末尾 x 位相等）。若 x 是 len 的因子，则由此可以继续推得前面所有的内容都是重复的，就得到了一个循环。并且，由于 Next 的性质，容易知道我们得到的循环节长度一定是最小的。所以直接得到答案 len/(len-next[len])。
与这题几乎完全一样的还有一题 POJ 1961，只不过是要求一个字符串的每个前缀最多是由多少个相同的串重复而来，也就是对 len 从 2 到原字符串长度跑一边判断一边即可。输入输出格式有变化，不另写题解，仅附上代码。
与 POJ 1961 类似的还有一题 HDOJ 1358，在 POJ 1961 的基础上要求前缀串的循环节数>=2。附上代码，题解不另写。

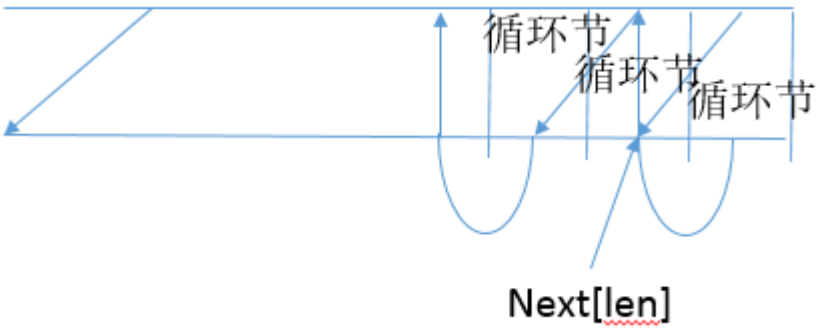
HDOJ3746【中等】

题目大意：
给出一个长度不超过 100000 个字符的字符串，问这个字符串最少添加多少个字符才能构成一个循环，循环次数大于等于 2。

输入：
第一行有一个整数 T，表示有 T 组测试数据。每一组测试数据有一行，包含一个字符串。

输出：
每一组测试数据输出一行，包含一个整数，表示最少需要添加的字符个数。

题解：
由 POJ 2406 可知，如果 sectlen=len-next[len]可以满足 len%sectlen==0 那么就不需要再添加其他字符了。如果不能整除，那么最小循环节长度必定为 sectlen。道理和 POJ 2406 题解里写的一样，只不过这一次不需要完全覆盖，但是还是能满足循环的性质。



这个时候只要添加 sectlen-next[len]%sectlen 个字符就可以了。

POJ2185【中等】

题目大意：
给出一个 r 行 c 列（1<=r<=10000，1<=c<=80）的字符矩阵，求一个最小覆盖子矩阵，使得这个子矩阵的无限复制扩张之后的矩阵，能包含原来的矩阵。

输入：
有若干行测试数据，处理到文件结尾。每一组测试数据第一行有两个整数 r 和 c，接下来有 r 行，每一行有一个长度为 c 的字符串。

输出：
每一组测试数据输出一行，包含一个整数，即最小覆盖子矩阵的面积。

题解：
一开始我的写法是，先对所有的行求一次循环节长度，然后取它们的最小公倍数，如果大于原来矩阵的宽就取原来矩阵的宽，作为最小子矩阵的宽，对列也是如此处理。后来看到网上有人指出，这是一个错误的贪心思路。举一个典型的例子：
2 8
ABCDEFAB
AAAABAAA
子矩阵的宽=lcm(5,6)=30>8，取 8，但是实际上只需要取 6 就可以了。所以，实际上的做法应该是先把每一行（列）视作一个元素，然后对列（行）做一次 Next 函数求值，得到最终的子矩阵高（宽）。

HDOJ1867【中等】

题目大意：

给出两个长度不超过 100000 个字符的字符串 s1、s2，规定它们可以首位相接，如果相接的部分是重复出现的（也就是 s1 的后几位是 s2 的前几位，或者反过来）。求长度最小的相接串，如果长度相同则取字典序小的一个。

输入：
有若干组测试数据，处理到文件末尾。每一组测试数据占一行，有两个字符串，用空格分隔。

输出：
每一组测试数据输出一行，包含长度最小的相接串。

题解：
这题是扩展 KMP 算法的模板题。关于扩展 KMP，可以参考 http://blog.sina.com.cn/s/blog_4ec51c420100gvzt.html 这里。很显然地，这一题就是求 s1 对 s2 后缀的公共前缀，以及求 s2 对 s1 后缀的公共前缀。当某个后缀与另一个串的公共前缀的长度等于这个后缀的长度，那么就是可以相接的。剩下的就是判断长度和字典序了，这个并不困难。

POJ2752【中等偏上】

题目大意：
给出一个长度不超过 40 万个字符的字符串，如果这个串存在一个长度为 n 的前缀串，和长度为 n 的后缀串，并且这两个串相等，则输出他们的长度 n。求出所有的长度 n。

输入：
有若干行测试数据，处理到文件末尾。每一行是一组测试数据，有一个字符串。

输出：
每一组测试数据输出一行，包含若干个空格分隔的整数，表示满足要求的前/后缀串的长度，按升序输出。

题解：

这题是对 Next 函数的又一个应用。由上面的 POJ 2406 我们知道，Next[len]=k 表示字符串前 k 位和后 k 位是相等的。那么这就是一个符合要求的前/后缀串。这个时候，如果 Next[k]=k'，意味着 1～k'和 k-k'+1～k 是相等的，而 k-k'+1～k 和 len-k'+1～len 又是相等的，所以就又可以得到多一个前/后缀串。因此这样不断迭代下去，直到 Next[k]=0 的时候才结束。注意，全串本身也是一个满足要求的前/后缀串，不能漏了！

HDOJ2594【中等偏上】

题目大意：
有两个长度不超过 50 万个字符的字符串 s1 和 s2，求一个 s1 的前缀串，同时也是 s2 的后缀串，要求这个串的长度最长。

输入：
有若干组测试数据，处理到文件结尾。每一组测试数据有两行，分别是字符串 s1 和 s2。字符串的内容是英文字母。

输出：
每一组测试数据输出一行，如果存在满足的前/后缀串，输出这个串，以及它的长度，用空格分隔。否则输出一行 0。

题解：
和 POJ 2752 基本一样，只需要把 s2 接在 s1 后面求 Next 数组即可。但是需要注意，这个串的长度不应该超过 s1 的长度和 s2 的长度。因此得到 i=next[len]以后需要一直取 i=next[i]，直到 i 第一次小于原来两个串的长度的较小者。这个时候停下得到的就是最长的前/后缀串，否则就得到 POJ 2752 里所需要的其他串了。

HDOJ3336【中等偏上】

题目大意：
给定一个不超过 20 万个字符的字符串 s，对于 s，有 len 个前缀串（包括其本身），pre[i]在 s 中的出现次数为 num[i]，求 sigma(num[i])。

输入：
第一行有一个整数 T，表示有 T 组测试数据。每一组测试数据第一行有一个整数，表示给出字符串的长度。第二行是给出的字符串。

输出：
每一组测试数据输出一行，包含一个整数，即 simga(num[i]) mod 10007。

题解：
对 Next[]数组的理解和一点小小的状态转移思想。求得 Next 数组后，设 dp[i]为以 i 为结尾的前缀在 s 中的出现次数（即 num[i]），考虑到 i 在匹配失败后将回溯到 next[i]，所以以 next[i]为结尾的前缀串在 i 为结尾的前缀串中出现了，因此可以得到 dp[i]=1+dp[next[i]]。

HDOJ4300【中等偏上】

题目大意：
给出一个小写英文字母的加密置换表（第 i 位上的字母是什么就把第 i 个字母置换为什么）。现在知道完整的密文和接在密文后面的明文前面一小段。现在需要把明文补全。密文的长度不会超过 50000 个字符。

输入：
第一行有一个整数 T，表示有 T 组测试数据。每一组测试数据第一行是加密置换表，第二行是密文和一小段明文。

输出：
每一组测试数据输出一行，先输出一次密文，然后输出一次明文，中间没有空格。

题解：
这题需要用到扩展 KMP 算法。首先把原来的密文和明文串 s1 解密，得到完整的明文和后面某一截不知为何物的字符串 s2。由于扩展 KMP 算法的作用是求一个字符串 a 对于另一个字符串 b 的所有后缀的公共前缀，那么这里 a=s2，b=s1，就是求明文串对于后缀某一部分是明文串的原串的最长前缀。因此做了一次扩展 KMP 以后，扫描一次数组 B（在部分网上的模板里是数组 extend，或者数组 B，在我的代码里是 BRes），当某一位 i 的 BRes[i]+i=原串长度的时候，就找到了密文和明文的分割位置了。

POJ2541【难】

这一题是一题比较麻烦的题目，因为原文的题意说得有点不好理解。我搜的时候这题被划入到了 KMP 的题目分类中，但是我按照网上题解的说法写了一个代码没有通过（虽然和另外一个网上 AC 的代码对拍过了）。后来看到这篇题解

http://www.cnblogs.com/swm8023/archive/2012/08/03/2621288.html，不清楚到底应该是状态压缩 DP 还是 KMP。

HDOJ4333【难】

题目大意：
给出一个长度不超过 100000 个字符的字符串，执行一种操作：每一次把最后一位抽出来放到第一位，得到一个新串。如此重复下去，直到将原串的所有字符都循环了一次。问所有得到的新串中，有多少种新串比原来的串要大、要小以及和原来的串相等。

输入：
第一行有一个整数 T，表示有 T 组测试数据。每一组测试数据占一行，包含一个由数字组成的字符串。

输出：
每一组测试数据输出一行，格式为 Case %测试数据组编号%： %小于原串种数% %等于原串种数% %大于原串种数%。其中测试数据组编号从 1 开始递增。

题解：
这题网上的很多题解都说得不清不楚，我自己摸索出道理以后决定写一个清晰一点的。
首先，我们需要判断，原来的字符串是否有循环节。因为如果有循环节，那么位移生成的新串也是循环出现的，我们只统计一个循环节中生成的新串就可以了。判断循环节直接用 KMP 算法里的 Next 函数就够了，这个在上述题解中有提及到。
然后，我们把给出的字符串重复一次接在原串后面。为什么可以这么做呢？假设现在按原来的要求操作，但是不删去末尾的字符，只是通过两个指针（虚拟的，脑中有这个想象就好）表示实际串的位置：
ABCD->DABCD->CDABCD->BCDABCD->ABCDABCD，和直接把原串重复一次得到的新串是一样的。这个时候，就可以使用扩展 KMP 算法了，把原串作为模式串，加倍后的串作为文本串，求出文本串的第 i 位开始的后缀（实际上就是原串位移了 len-i 次后得到的串+一个无关紧要的后续重复）与原串的公共前缀。如果公共前缀的长度等于原串，那么位移串和原串就是相等的。否则，可以直接对比匹配位置后一位的大小，得到原串和位移串的大小。
我在代码里的注释把位移 len-i 次直接写成了位移 i 次，实际上是错误的，但是不影响理解，这里写清楚，以免误解。