CEOI2011 mat solution 中文翻译

这个问题描述了一个字符串匹配的变种问题。给定两个串，模式串 p[1..n]和文本串 t[1..m]。任务是找出所有的位置 j,1 <= j <= m - n + 1，满足模式串在位置 j 匹配文本串。而且，在这个问题中，模式串和文本串都是互异整数列。

而且，模式串 p 不是直接给出来的。给定一个数列 s 来描述 p：s1 是 p 中最小的元素，s2 是第二小的……任意一个等于输入的 p 都是等价的，所以我们假定 p 是 1..n 的一个排列。这种表示很容易由 s 在 O(n)计算出来。

在大部分模式串匹配问题中，模式串与母串在位置 j 匹配当且仅当 p=t[j..j+n-1]。这个问题给出了一种不同的串匹配定义，模式串与母串匹配，当且仅当 p 与 t[j..j+n-1]是同构的。我们称呼两个长度为 k 的串同构，当且仅当 a[i] < a[j] <==> b[i] < b[j] forall i <= i, j <= k。

为简化符号，我们把 a 与 b 同构写成 a~b。下面，我们有两个简单但是重要的结论。给定三个长为 k 的序列 a,b,c。

1. 如果 a ~ b，则 a[x..y] ~ b[x..y] for i <= x <= y <= k
2. 如果 a ~ b 且 b ~ c，则 a ~ c。

为了解决这个问题，我们拓展了 KMP 算法来满足需求。接下来，我们假定读者熟悉 KMP 算法。

失败指针：

我们定义：一个序列 a[1..k]的边界为，a[1..k]的一个长为 t 的后缀，且这个后缀与 a[1..t]相似。在 KMP 算法中，我们一开始要计算失败指针 f。对于每个 1 <= i <= n，我们想知道串 p[1..i]除自己本身外的最长边界是什么：

$f[i] = \max \{p[1..k] \sim p[i - k + 1..i]\}\ 0 \le k < i$

另外，我们把 f[0]设为 0。

我们以 i 递增的顺序来计算 f[i]。p[1..i]的最长边界包括 p[1..i - 1]的一部分和字母 p[i]。我们遍历 p[1..i - 1]的所有边界，从最长的开始，对于每个边界，我们检查添加一个字母 p[i]后能否构成 p[1..i]的一个边界。

我们用下面的引理来遍历所有边界，它的证明稍后给出。

引理 1：p[1..i]的所有边界的长度依次为 f[i], f[f[i]], f[f[f[i]]], f[f[f[f[i]]]]...

注意：由于 0 <= f[i] < i，上述序列从某些点开始就只出现 0 了。

现在仍然遗留一个问题：如何判断 p[1..i - 1]的一个边界加上一个字母 p[i]后可以构成 p[1..i]的一个边界。换句话说，给定两个串 a[1..k]、b[1..k]（前者是模式串的一个前缀，后者是模

式串的一个子串），已知 a[1..k - 1] ~ b[1..k - 1]，如何判断 a[1..k] ~ b[1..k]。注意到这就是判断是否：

a[q] < a[k] <--> b[q] < b[k] for all 1 <= q < k

这可以按照下面这个方式重新表述（注意每个序列 a、b 的元素都是不同的）：

性质 1：对于一些 1 <= r <= k，a[k]是 a[1..k]中第 r 大的元素，b[k]是 b[1..k]中第 r 大的元素。

我们现在描述一个检查是否满足上述条件的方法。

设 a[u]是 a[1..k - 1]中比 a[k]小的元素中最大的一个，a[w]是 a[1..k - 1]中比 a[k]大的元素中最小的一个。我们假定这些元素存在，其他情况类似。根据定义，a[u] < a[k] < a[w]。我们可以知道判断 b[u] < b[k] < b[w]是否成立是与判断性质 1 等价的。这是因为 a[1..k - 1] ~ b[1..k - 1]，所以 a[1..k - 1]中比 a[u]小的数的个数等于 b[1..k - 1]中比 b[u]小的数的个数。类似的，a[1..k - 1]中比 a[w]大的数的个数等于 b[1..k - 1]中比 b[w]大的

数的个数。所以，这个检测与性质 1 实质上是等价的。

现在，我们讨论如何计算 u 和 w 的下标。对于每个 1 <= i <= n，我们需要在 p[1..i]找到比 p[i]小的最大元素，把这个下标记作 g[i]。我们也需要知道比 p[i]大的最小元素，记作 h[i]（这是一个对称的问题）。

注意到 p[1..n]是一个 1..n 的排列。我们维护一个由 p[1..i]的所有元素构成的递增的双向链表。初始时他只是一个 1..n 的所有元素构成的链表。每一步都要删除一个元素。我们记录链表中的每个元素在 p[1..n]中的位置，也记录 p[1..n]的每个元素在链表中的位置。链表允许我们对于每个 i，可以在常数时间内获得与 p[i]最接近的元素——他们只是链表中，删除了 n - i 个元素之后，p[i]的前驱与后继。

这就给出了一个计算失败指针的算法。O(n)预处理出数组 g[i..n]与它的对称问题 h[i..n]之后，算法就与 KMP 算法的失败指针的计算完全一致了。整个算法的运行时间为 O(n)。

寻找匹配

KMP 匹配算法的主要过程，大概讲如下：

给定模式串的一部分，尽量拓展一个字符。如果不可行，用失败指针得到一个稍短的部分匹配，继续匹配文字。

这也是我们在这个问题中要做的事。用上面的方法，我们能够在常数时间内判断一个部分匹配能不能再拓展一个字符。正确性的证明很简单，主要思想是：如果我们跳过一些正确的匹配（即，我们用失败指针把部分匹配移动得太多超过了匹配的开始），我们马上就得到这与失败指针的定义是矛盾的。

最后，由于匹配过程只是 KMP 算法的轻微修改，所以在 O(n+m)的时间内可以出解。所以，整个算法只需要线性时间。

引理 1 的证明：

我们证明，如果 p[1..i]有一个长度为 t 的边界，那么比 t 短的最长的边界长度为 f[t]。如果 f[t] = 0，那么长度为 t 的边界是最短的一个。从边界的定义可以知道，p[1..t] ~ p[i - t + 1..i]。

我们首先证明 p[1..i]有一个长度为 f[t]的边界。首先，注意到 p[1..t] ~ p[i - t + 1..i]。由此得出，对于每个 1 <= s < t，p[1..t]有一个长度为 s 的边界。此外，任意一个 p[1..t]的边界与 p[i - t + 1..i]的边界相似。所以，一个长度为 f[t]的 p[1..t]的边界与长为 f[t]的 p[i - t + 1..i]的边界相似。这就表明 p[1..f[t]] ~ p[i - f[t] + 1..i]。

为了完成这个证明，我们得证明没有长度严格在 f[t]与 t 之间的边界存在。为了证明这个，我们证明每一个这样长度的边界一定也是 p[1..t]的边界，否则会和 f[t]的定义相矛盾。设 f[t] < u < t 且 p[1..u] ~ p[i - u + 1..i]，也就是说，p[1..i]有一个长度为 u 的边界。这意味着 p[1..t]的一个长度为 u 的后缀与 p[1..t + 1..t]的一个长度为 u 的后缀相似。我们已经知道 p[1..t]和 p[i - t + 1..i]相似，所欲 p[i - t + 1..i]的一个长为 u 的后缀与 p[1..i]的一个长为 u 的后缀相似。所以 p[1..t]的一个长为 u 的后缀与 p[1..u]相似，矛盾。