后缀数组

【作者】

安徽省芜湖市第一中学　许智磊

【摘要】

本文介绍后缀数组的基本概念、方法以及应用。

首先介绍 O(nlogn)复杂度构造后缀数组的倍增算法，接着介绍了配合后缀数组的最长公共前缀 LCP（Longest Common Prefix）的计算方法，并给出一个线性时间内计算 height 数组（记录跨度为 1 的 LCP 值的数组）的算法。最后介绍两个应用后缀数组的例子：多模式串的模式匹配以及求最长回文子串。

【关键字】

字符串　后缀　k-前缀比较关系
后缀数组　名次数组　后缀树　倍增算法　基数排序
最长公共前缀　RMQ 问题　模式匹配　回文串　最长回文子串

【正文】

在字符串处理当中，后缀树和后缀数组都是非常有力的工具，其中后缀树大家了解得比较多，关于后缀数组则很少见于国内的资料。其实后缀数组是后缀树的一个非常精巧的替代品，它比后缀树容易编程实现，能够实现后缀树的很多功能而时间复杂度也不太逊色，并且，它比后缀树所占用的空间小很多。可以说，在信息学竞赛中后缀数组比后缀树要更为实用。因此在本文中笔者想介绍一下后缀数组的基本概念、构造方法，以及配合后缀数组的最长公共前缀数组的构造方法，最后结合一些例子谈谈后缀数组的应用。

基本概念

首先明确一些必要的定义：

字符集　一个字符集 Σ 是一个建立了全序关系的集合，也就是说，Σ 中的任意两个不同的元素 α 和 β 都可以比较大小，要么 α<β，要么 β<α（也就是 α>β）。字符集 Σ 中的元素称为字符。

字符串　一个字符串 S 是将 n 个字符顺次排列形成的数组，n 称为 S 的长度，表示为 len(S)。S 的第 i 个字符表示为 S[i]。

子串　字符串 S 的子串 S[i..j]，i≤j，表示 S 串中从 i 到 j 这一段，也就是顺次排列 S[i],S[i+1],...,S[j]形成的字符串。

后缀　后缀是指从某个位置 i 开始到整个串末尾结束的一个特殊子串。字符串 S 的从 i 开头的后缀表示为 Suffix(S,i)，也就是 Suffix(S,i)=S[i..len(S)]。

关于字符串的大小比较，是指通常所说的“字典顺序”比较，也就是对于两个字符串 u、v，令 i 从 1 开始顺次比较 u[i]和 v[i]，如果相等则令 i 加 1，否则

若 u[i]<v[i]则认为 u<v，u[i]>v[i]则认为 u>v（也就是 v<u），比较结束。如果 i>len(u)或者 i>len(v)仍比较出结果，那么若 len(u)<len(v)则认为 u<v，若 len(u)=len(v)则认为 u=v，若 len(u)>len(v)则 u>v。

从字符串的大小比较的定义来看，S 的两个开头位置不同的后缀 u 和 v 进行比较的结果不可能是相等，因为 u=v 的必要条件 len(u)=len(v)在这里不可能满足。

下面我们约定一个字符集 Σ 和一个字符串 S，设 len(S)=n，且 S[n]='$'，也就是说 S 以一个特殊字符'$'结尾，并且'$'小于 Σ 中的任何一个字符。除了 S[n]之外，S 中的其他字符都属于 Σ。对于约定的字符串 S，从位置 i 开头的后缀直接写成 Suffix(i)，省去参数 S。

后缀数组　后缀数组 SA 是一个一维数组，它保存 1..n 的某个排列 SA[1],SA[2],...SA[n]，并且保证 Suffix(SA[i])<Suffix(SA[i+1]),1≤i<n。也就是将 S 的 n 个后缀从小到大进行排序之后把排好序的后缀的开头位置顺次放入 SA 中。

名次数组　名次数组 $Rank=SA^{-1}$，也就是说若 SA[i]=j，则 Rank[j]=i，不难看出 Rank[i]保存的是 Suffix(i)在所有后缀中从小到大排列的“名次”。

构造方法

如何构造后缀数组呢？最直接最简单的方法当然是把 S 的后缀都看作一些普通的字符串，按照一般字符串排序的方法对它们从小到大进行排序。

不难看出，这种做法是很笨拙的，因为它没有利用到各个后缀之间的有机联系，所以它的效率不可能很高。即使采用字符串排序中比较高效的 Multi-key

Quick Sort，最坏情况的时间复杂度仍然是 $O(n^2)$的，不能满足我们的需要。

下面介绍倍增算法(Doubling Algorithm)，它正是充分利用了各个后缀之间的联系，将构造后缀数组的最坏时间复杂度成功降至 O(nlogn)。

对一个字符串 u，我们定义 u 的 k-前缀

$$u^k = \begin{cases} u[1..k] & ,len(u) \geq k \\ u & ,len(u) < k \end{cases}$$

定义 k-前缀比较关系$<_k$、$=_k$和$\leqslant_k$：

设两个字符串 u 和 v，

$u<_k v$　当且仅当　$u^k<v^k$

$u=_k v$　当且仅当　$u^k=v^k$

$u\leq_k v$　当且仅当　$u^k\leq v^k$

直观地看这些加了一个下标 k 的比较符号的意义就是对两个字符串的前 k 个字符进行字典序比较，特别的一点就是在作大于和小于的比较时如果某个字符串的长度不到 k 也没有关系，只要能够在 k 个字符比较结束之前得到第一个字符串大于或者小于第二个字符串就可以了。

根据前缀比较符的性质我们可以得到以下的非常重要的性质：

性质 1.1　对 k≥n，$Suffix(i)<_k Suffix(j)$　等价于　$Suffix(i)<Suffix(j)$。

性质 1.2　$Suffix(i)=_{2k}Suffix(j)$等价于

$Suffix(i)=_k Suffix(j)$ 且 $Suffix(i+k)=_k Suffix(j+k)$。

性质 1.3　$Suffix(i)<_{2k}Suffix(j)$ 等价于

$Suffix(i)<_k S(j)$ 或 $(Suffix(i)=_k Suffix(j)$ 且 $Suffix(i+k)<_k Suffix(j+k))$。

这里有一个问题，当 i+k>n 或者 j+k>n 的时候 Suffix(i+k)或 Suffix(j+k)是无

明确定义的表达式，但实际上不需要考虑这个问题，因为此时 Suffix(i)或者 Suffix(j)的长度不超过 k，也就是说它们的 k-前缀以'$'结尾，于是 k-前缀比较的结果不可能相等，也就是说前 k 个字符已经能够比出大小，后面的表达式自然可以忽略，这也就看出我们规定 S 以'$'结尾的特殊用处了。

定义 **k-后缀数组** SA_k 保存 1..n 的某个排列 $SA_k[1],SA_k[2],\ldots SA_k[n]$ 使得 $Suffix(SA_k[i]) \leq_k Suffix(SA_k[i+1]),1\leq i<n$。也就是说对所有的后缀在 k-前缀比较关系下从小到大排序，并且把排序后的后缀的开头位置顺次放入数组 SA_k 中。

定义 **k-名次数组** $Rank_k$，$Rank_k[i]$代表 Suffix(i)在 k-前缀关系下从小到大的“名次”，也就是 1 加上满足 $Suffix(j)<_k Suffix(i)$的 j 的个数。通过 SA_k 很容易在 O(n)的时间内求出 $Rank_k$。

假设我们已经求出了 SA_k 和 $Rank_k$，那么我们可以很方便地求出 SA_{2k} 和 $Rank_{2k}$，因为根据**性质 1.2** 和 **1.3**，2k-前缀比较关系可以由常数个 k-前缀比较关系组合起来等价地表达，而 $Rank_k$ 数组实际上给出了在常数时间内进行 $<_k$ 和 $=_k$ 比较的方法，即：

$Suffix(i)<_k Suffix(j)$ 当且仅当 $Rank_k[i]<Rank_k[j]$

$Suffix(i)=_k Suffix(j)$ 当且仅当 $Rank_k[i]=Rank_k[j]$

因此，比较 Suffix(i)和 Suffix(j)在 k-前缀比较关系下的大小可以在常数时间内完成，于是对所有的后缀在 $\leqslant_k$ 关系下进行排序也就和一般的排序没有什么区别了，它实际上就相当于每个 Suffix(i)有一个主关键字 $Rank_k[i]$和一个次关键字 $Rank_k[i+k]$。如果采用快速排序之类 O(nlogn)的排序，那么从 SA_k 和 $Rank_k$ 构造出 SA_{2k} 的复杂度就是 O(nlogn)。更聪明的方法是采用基数排序，复杂度为 O(n)。

求出 SA_{2k} 之后就可以在 O(n)的时间内根据 SA_{2k} 构造出 $Rank_{2k}$。因此，从 SA_k

和 $Rank_k$ 推出 SA_{2k} 和 $Rank_{2k}$ 可以在 O(n)时间内完成。

下面只有一个问题需要解决：如何构造出 SA_1 和 $Rank_1$。这个问题非常简单：因为 $<_1$，$=_1$ 和 $\leqslant_1$ 这些运算符实际上就是对字符串的第一个字符进行比较，所以只要把每个后缀按照它的第一个字符进行排序就可以求出 SA_1，不妨就采用快速排序，复杂度为 O(nlogn)。

于是，可以在 O(nlogn)的时间内求出 SA_1 和 $Rank_1$。

求出了 SA_1 和 $Rank_1$，我们可以在 O(n)的时间内求出 SA_2 和 $Rank_2$，同样，我们可以再用 O(n)的时间求出 SA_4 和 $Rank_4$，这样，我们依次求出：

SA_2 和 $Rank_2$，SA_4 和 $Rank_4$，SA_8 和 $Rank_8$，……直到 SA_m 和 $Rank_m$，其中 $m=2^k$ 且 $m\geq n$。而根据**性质 1.1**，SA_m 和 SA 是等价的。这样一共需要进行 logn 次 O(n)的过程，因此

可以在 O(n1ogn)的时间内计算出后缀数组 SA 和名次数组 Rank。

最长公共前缀

现在一个字符串 S 的后缀数组 SA 可以在 O(nlogn)的时间内计算出来。利用 SA 我们已经可以做很多事情，比如在 O(mlogn)的时间内进行模式匹配，其中 m,n 分别为模式串和待匹配串的长度。但是要想更充分地发挥后缀数组的威力，我们还需要计算一个辅助的工具——**最长公共前缀**（Longest Common Prefix）。

对两个字符串 u,v 定义函数 $lcp(u,v)=max\{i|u=_i v\}$，也就是从头开始顺次比较 u 和 v 的对应字符，对应字符持续相等的最大位置，称为这两个字符串的**最长**

公共前缀。

对正整数 i,j 定义 LCP(i,j)=lcp(Suffix(SA[i]),Suffix(SA[j])，其中 i,j 均为 1 至 n 的整数。LCP(i,j)也就是后缀数组中第 i 个和第 j 个后缀的最长公共前缀的长度。

关于 LCP 有两个显而易见的性质：

性质 2.1　LCP(i,j)=LCP(j,i)

性质 2.2　LCP(i,i)=len(Suffix(SA[i]))=n-SA[i]+1

这两个性质的用处在于，我们计算 LCP(i,j)时只需要考虑 i<j 的情况，因为 i>j 时可交换 i,j，i=j 时可以直接输出结果 n-SA[i]+1。

直接根据定义，用顺次比较对应字符的方法来计算 LCP(i,j)显然是很低效的，时间复杂度为 O(n)，所以我们必须进行适当的预处理以降低每次计算 LCP 的复杂度。

经过仔细分析，我们发现 LCP 函数有一个非常好的性质：

设 i<j，则 LCP(i,j)=min{LCP(k-1,k)|i+1≤k≤j}　（**LCP Theorem**）

要证明 **LCP Theorem**，首先证明 **LCP Lemma**:

对任意 1≤i<j<k≤n，LCP(i,k)=min{LCP(i,j),LCP(j,k)}

证明：设 p=min{LCP(i,j),LCP(j,k)}，则有 LCP(i,j)≥p,LCP(j,k)≥p。

设 Suffix(SA[i])=u,Suffix(SA[j])=v,Suffix(SA[k])=w。

由 $u=_{LCP(i,j)}v$ 得 $u=_p v$；同理 $v=_p w$。

于是 $Suffix(SA[i])=_p Suffix(SA[k])$，即 LCP(i,k)≥p。　(1)

又设 LCP(i,k)=q>p，则

u[1]=w[1],u[2]=w[2],...u[q]=w[q]。

而 min{LCP(i,j),LCP(j,k)}=p 说明 u[p+1]≠v[p+1]或 v[p+1]≠w[q+1]，

设 u[p+1]=x,v[p+1]=y,w[p+1]=z，显然有 x≤y≤z，又由 p<q 得 p+1≤q，应该有 x=z，也就是 x=y=z，这与 u[p+1]≠v[p+1] 或 v[p+1]≠w[q+1]矛盾。

于是，q>p 不成立，即 LCP(i,k)≤p。 (2)

综合(1),(2)知 LCP(i,k)=p=min{LCP(i,j),LCP(j,k)}，**LCP Lemma** 得证。

于是 **LCP Theorem** 可以证明如下：

当 j-i=1 和 j-i=2 时，显然成立。

设 j-i=m 时 **LCP Theorem** 成立，当 j-i=m+1 时，

由 **LCP Lemma** 知 LCP(i,j)=min{LCP(i,i+1),LCP(i+1,j)}，

因 j-(i+1)≤m，LCP(i+1,j)=min{LCP(k-1,k)|i+2≤k≤j}，故当 j-i=m+1 时，仍有 LCP(i,j)=min{LCP(i,i+1),min{LCP(k-1,k)|i+2≤k≤j}}=min{LCP(k-1,k}|i+1≤k≤j)

根据数学归纳法，**LCP Theorem** 成立。

根据 **LCP Theorem** 得出必然的一个推论：

LCP Corollary 对 i≤j<k，LCP(j,k)≥LCP(i,k)。

定义一维数组 height，令 height[i]=LCP(i-1,i)，1<i≤n，并设 height[1]=0。

由 **LCP Theorem**，LCP(i,j)=min{height[k]|i+1≤k≤j}，也就是说，计算 LCP(i,j)等同于询问一维数组 height 中下标在 i+1 到 j 范围内的所有元素的最小值。如果 height 数组是固定的，这就是非常经典的 RMQ（Range Minimum Query）问题。

RMQ 问题可以用线段树或静态排序树在 O(nlogn)时间内进行预处理，之后

每次询问花费时间 O(logn)，更好的方法是 RMQ 标准算法，可以在 O(n)时间内进行预处理，每次询问可以在常数时间内完成。

对于一个固定的字符串 S，其 height 数组显然是固定的，只要我们能高效地求出 height 数组，那么运用 RMQ 方法进行预处理之后，每次计算 LCP(i,j)的时间复杂度就是常数级了。于是只有一个问题——如何尽量高效地算出 height 数组。

根据计算后缀数组的经验，我们不应该把 n 个后缀看作互不相关的普通字符串，而应该尽量利用它们之间的联系，下面证明一个非常有用的性质：

为了描述方便，设 h[i]=height[Rank[i]]，即 height[i]=h[SA[i]]。h 数组满足一个性质：

性质 3　对于 i>1 且 Rank[i]>1，一定有 h[i]≥h[i-1]-1。

为了证明**性质 3**，我们有必要明确两个事实：

设 i<n,j<n，Suffix(i)和 Suffix(j)满足 lcp(Suffix(i),Suffix(j)>1，则成立以下两点：

Fact 1　Suffix(i)<Suffix(j)　等价于　Suffix(i+1)<Suffix(j+1)。

Fact 2　一定有 lcp(Suffix(i+1),Suffix(j+1))=lcp(Suffix(i),Suffix(j))-1。

看起来很神奇，但其实很自然：lcp(Suffix(i),Suffix(j))>1 说明 Suffix(i)和 Suffix(j)的第一个字符是相同的，设它为 α，则 Suffix(i)相当于 α 后连接 Suffix(i+1)，Suffix(j)相当于 α 后连接 Suffix(j+1)。比较 Suffix(i)和 Suffix(j)时，第一个字符 α 是一定相等的，于是后面就等价于比较 Suffix(i)和 Suffix(j)，因此 **Fact 1** 成立。**Fact 2** 可类似证明。

于是可以证明**性质 3**：

当 h[i-1]≤1 时，结论显然成立，因 h[i]≥0≥h[i-1]-1。

当 h[i-1]>1 时，也即 height[Rank[i-1]]>1，可见 Rank[i-1]>1，因 height[1]=0。

令 j=i-1,k=SA[Rank[j]-1]。显然有 Suffix(k)<Suffix(j)。

根据 h[i-1]=lcp(Suffix(k),Suffix(j))>1 和 Suffix(k)<Suffix(j)：

由 **Fact 2** 知 lcp(Suffix(k+1),Suffix(i))=h[i-1]-1。

由 **Fact 1** 知 Rank[k+1]<Rank[i]，也就是 Rank[k+1]≤Rank[i]-1。

于是根据 **LCP Corollary**，有

$$\begin{aligned}LCP(Rank[i]-1,Rank[i]) &\geq LCP(Rank[k+1],Rank[i])\\ &= lcp(Suffix(k+1),Suffix(i))\\ &= h[i-1]-1\end{aligned}$$

由于 h[i]=height[Rank[i]]=LCP(Rank[i]-1,Rank[i])，最终得到 h[i]≥h[i-1]-1。

根据**性质 3**，可以令 i 从 1 循环到 n 按照如下方法依次算出 h[i]：

若 Rank[i]=1，则 h[i]=0。字符比较次数为 0。

若 i=1 或者 h[i-1]≤1，则直接将 Suffix(i)和 Suffix(Rank[i]-1)从第一个字符开始依次比较直到有字符不相同，由此计算出 h[i]。字符比较次数为 h[i]+1，不超过 h[i]-h[i-1]+2。

否则，说明 i>1，Rank[i]>1，h[i-1]>1，根据**性质 3**，Suffix(i) 和 Suffix(Rank[i]-1)至少有前 h[i-1]-1 个字符是相同的，于是字符比较可以从 h[i-1] 开始，直到某个字符不相同，由此计算出 h[i]。字符比较次数为 h[i]-h[i-1]+2。

设 SA[1]=p，那么不难看出总的字符比较次数不超过

$$\begin{aligned}&h[1]+1+\sum_{i=2}^{p-1}(h[i]-h[i-1]+2)+h[p+1]+\sum_{i=p+1}^{n}(h[i]-h[i-1]+2)\\ &\leq 1+h[p-1]+2(p-2)+h[n]+2+2(n-p)\leq 4n\end{aligned}$$

也就是说，整个算法的复杂度为 O(n)。

求出了 h 数组，根据关系式 height[i]=h[SA[i]]可以在 O(n)时间内求出 height 数组，于是

可以在O(n)时间内求出height数组。

结合RMQ方法，在O(n)时间和空间进行预处理之后就能做到在常数时间内计算出对任意(i,j)计算出LCP(i,j)。

因为lcp(Suffix(i),Suffix(j))=LCP(Rank[i],Rank[j])，所以我们也就可以在常数时间内求出S的任何两个后缀之间的最长公共前缀。这正是后缀数组能强有力地处理很多字符串问题的重要原因之一。

后缀数组的应用

下面结合两个例子谈谈如何运用后缀数组。

例一　多模式串的模式匹配问题

后缀数组与后缀树的比较