数据结构和算法实习

郭 炜

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄

后缀数组

最长公共前缀数组

贵州黄果树瀑布

名次数组Rank

- 有了sa以后，希望O(1）时间可求位置i的后缀的名次

- 构造Rank数组即可。Rank[i] 表示位置 i 的后缀的名次

```
for(int i = 0;i < n; ++i)//按名次从小到大遍历后缀
    Rank[sa[i]] = i;
```

最长公共前缀数组

- 要解决以下问题:

对给定长为n的字符串，希望经过复杂度O(nlogn) 的预处理后，任给2个后缀，都能在O(1)时间内求得这2个后缀的最长公共前缀的长度。

最长公共前缀: Longest Common Prefix，简称 LCP
最长公共前缀的长度: 简称 LCPL

最长公共前缀数组

- 一些名词:

  Suffix(i) ： 位置为 i 的后缀

  LCP(Suffix(i), Suffix(j))： 位置为 i 的后缀，和位置为j的后缀的LCP

  LCPL(Suffix(i), Suffix(j))： LCP(Suffix(i), Suffix(j)) 的长度

  LCP(i,j) ： 名次为i的后缀，和名次为j的后缀的最长公共前缀

  LCPL(i,j)： LCP(i,j) 的长度 (i < j)

最长公共前缀数组

- 预处理

  在O(nlogn)求得后缀数组sa后，可以O(n)时间求得一个最长公共前缀数组 height：

  height[0] = 0
  height[i] = LCPL(i-1,i)　　　　0 < i <= n-1

  height[i]，就是名次为i的后缀，和名次为 i-1的后缀的 LCPL

最长公共前缀数组

- 有了height后，O(1)时间求 LCPL(i,j)

  LCPL(i,j) = min { LCPL(k,k+1) | k = i..j-1 }

  height[i] = LCPL(i-1,i)   0 < i <= n-1

  LCPL(i,i+1) = height[i+1]
  LCPL(i+1,i+2) = height[i+2]
  ....
  LCPL(j-1,j) = height[j]

  LCPL(i,j) = min { height[i+1 … j] }, 这是 RMQ问题，O(1) 解决。(RMQ预处理(nlogn))

最长公共前缀数组

- 有了sa后，O(n) 求height数组

  假设已经O(n)求得了一个 H 数组:

  H[i] = LCPL(Rank[i] – 1,Rank[i])
  = LCPL(Suffix(i),Suffix(sa[Rank[i] – 1]))

  H[i]是位置i的后缀X，和名次在X前一位的后缀Y的 LCPL 。
  则:

```
for(int i = 0;i < n; ++i)  //按名次从小到大遍历后缀
        height[i] =  H[ sa[i] ] ;
```

或

```
for(int i = 0;i < n; ++i)  //按位置从小到大遍历后缀
        height[Rank[i]] =  H[i] ;
```

求 H 数组

- 关于 H 数组的定理(简称 H定理 稍后证明）：

  H[i] >= H[i-1] -1　　　　(当i > 0且 Rank[i] > 0时)

- 定理用途:
  若 Rank[i] = 0,则 H[i] = 0
  否则, 欲求H[i]，需要比较 Suffix(i)和Suffix(sa[Rank[i]-1])
  (sa就用在这里了)
  因 H[i] >= H[i-1] -1, 故这两个后缀的前 H[i-1] -1 个字符是相等的 (要求H[i-1] -1 >0) 。因此只需要从这两个后缀的第 H[i-1] 个字符开始比较即可。
  而且，比到第H[i]+1个字符，就发现不等了，因此比较的字符数不超过 H[i] – H[i-1] + 2

求 H 数组

- 求 H[0]
  硬比较 Suffix(0)和Suffix(sa[Rank[0]-1]), O(n)

- 依次求H[1],H[2] ... H[n-1]

  求 H[i]时要在两个后缀间做一些字符的比较，比较次数不超过

  H[i] - H[i-1] + 2

  则总的比较次数 不超过
  $\sum$(H[i]- H[i-1]) + 2n   i= 1...n-1

求 H 数组

- 计算 ∑(H[i]- H[i-1])   i= 1...n-1

  即所有"H值增量" 的和

  H值增量可以为负，但最小为-1(因为 H[i]>=H[i-1]-1)
  因负的增量之和最多 -(n-1) ,且 H 值最大为 n-1
  故正的增量之和不超过 2(n-1)
  故 ∑(H[i]- H[i-1]) 为 O(n)
  故求H数组为 O(n) (在sa已知的前提下)

证明H定理: H[i] >= H[i-1] -1

- LCP 引理 1

  LCPL(i, j) = min{ LCPL(k,k+1) | k = i ... j – 1}

  证明:

1） 设 LCP(i,j) = "banana", 则 名次在 [i,j] 内的所有后缀，都以 "banana" 打头。因此

   LCPL(i,j) <= LCPL(k, k+1) | k = i ... j – 1
   即每个 LCPL(k,k+1)都不小于 LCPL(i,j) k = i ... j-1

证明H定理: H[i] >= H[i-1] -1

2) 不是每个 LCPL(k,k+1) 都大于 LCPL(i,j)　　k = i ... j -1

- 辅助定理: 若 LCPL(b,c) > x 且 LCPL(a,b) > x 则 LCPL(a,c) > x
- 反证法

　　LCPL(j-1,j) > LCPL(i,j) 且 LCPL(j-2,j-1) > LCPL(i,j)
=> LCPL(j-2,j) > LCPL(i,j)
　　LCPL(j-2,j) > LCPL(i,j) 且 LCPL(j-3,j-2) > LCPL(i,j)
=> LCPL(j-3,j) > LCPL(i,j) .
　　.....
　　LCPL(i,i+1) > LCPL(i,j) 且 LCPL(i+1,j) > LCPL(i,j)
=> LCPL(i,j) > LCPL(i,j)　　，矛盾。所以 2)成立

证明H定理: H[i] >= H[i-1] -1

- LCP 引理 2

对任何 $i <= k < j$, $LCPL(k,j) >= LCPL(i,j)$

i, j, k 都是后缀的名次

由 LCP 引理 1 显然推得。

证明H定理: H[i] >= H[i-1] -1

- LCP 引理 3

  如果 LCPL(Suffix(i),Suffix(j)) >= 1, 则有:

  Fact 1:

  Suffix(i) < Suffix(j) ⇔ Suffix(i+1) < Suffix(j+1)

  Fact 2:

  LCPL(Suffix(i+1),Suffix(j+1)) = LCPL(Suffix(i),Suffix(j)) - 1

证明H定理: H[i] >= H[i-1] -1

- 若 H[i-1] = 0, 则H[i] > H[i-1] - 1显然成立
- 若 H[i-1] > 0 则:

➢ 令 k = sa[Rank[i-1]-1] ,则 H[i-1] = LCPL(Suffix(k),Suffix(i-1)) > 0
LCPL(Suffix(k),Suffix(i-1)) > 0 且 Suffix(k) < Suffix(i-1)
根据LCP引理3 - fact 1:
→ Suffix(k+1) < Suffix(i)
→ Rank[k+1] < Rank[i] → Rank[k+1] <= Rank[i]-1

➢ LCPL(Suffix(k),Suffix(i-1)) > 0
根据LCP引理3 - fact 2
→ LCPL(Suffix(k+1),Suffix(i)) = LCPL(Suffix(k),Suffix(i-1)) - 1
= H[i-1] - 1
→ LCPL(Rank[k+1],Rank[i]) = H[i-1] - 1

证明H定理: $H[i] >= H[i-1] -1$

- 由于 $Rank[k+1] <= Rank[i] - 1$
  根据 LCP引理2
  → $LCPL(Rank[i]-1,Rank[i]) >= LCPL(Rank[k+1],Rank[i])$

- 由于 $H[i] = LCPL(Rank[i]-1,Rank[i])$
  $H[i-1] - 1 = LCPL(Rank[k+1],Rank[i])$

  → $H[i] >= H[i-1] - 1$

求height的实现

```
void BuildHeight(char * str,int n,int * sa,int * Rank) {
     int i, j, k;
     for(int i = 0;i < n; ++i) //i 是名次,n是字符串长度
          Rank[sa[i]] =  i;
     height[0] = 0;
     for (i = k = 0; i < n - 1; height[Rank[i++]] = k)//i是位置
          for (k ? k-- : 0, j = sa[Rank[i] - 1]; //Rank[0]>0才不越界
                    str[i+k]==str[j+k]; k++);
       //k相当于是 H[i]; height[Rank[i]] =  H[i] ;
}
```

要求一开始 Rank[0] > 0 。只有str的最后一个字符是最小的，且在前面都不出现，才可确保这一点。因此, 用此函数求 height数组，则前面调用 BuildSa,以及调用此函数时，实参n都必须是字符串实际的长度加 1 (相当于把结尾的'\0'也算成字符串的一部分) , 比如:

```
BuildSa("abcd",sa,5,130);    //实际上处理的字符串是 "abcd\0"
BuildHeight("abcd",5,sa,Rank);
```

例题1: POJ2774 Long Long Message

给定两个长度不超过 100000 的由小写字母组成的字符串，求它们的最长公共子串的长度

Sample Input

```
yeshowmuchiloveyoumydearmotherreallyicannotbelieveit
yeaphowmuchiloveyoumydearmother
```

Sample Output

```
27
```

例题1: POJ2774 Long Long Message

1）把两个字符串 str1和str2 拼接起来，中间用非小写字母(比如 '-' 隔开)，得到str 。于是，str的任意两个后缀的LCP都不会包含 '-' 。即对任一LCP, 其所对应的两个字符串 s1和s2(s1=s2但s1位置比s2更左), 只有以下三种情况:

1. s1,s2都完全属于str1
2. s1,s2都完全属于str2
3. s1 完全属于str1, s2完全属于str2

2）str1和str2的最大公共子串，就是str的某两个后缀的LCP

假设是 LCP(i,j) i,j是名次， i < j ，

则 sa[i] < len1, sa[j] > len1 (len1是str1长度)

(或 sa[j] < len1, sa[i] > len1)

例题1: POJ2774 Long Long Message

3) sa[i] 和 sa[j] 在分隔符的两侧

故总能找到一个k ∈ [i, j] ，使得 sa[k] 和 sa[k+1] 在分隔符两侧

根据 LCP 引理1，LCP(i,j) <= LCP(k,k+1)

由于 LCP(i,j)已经是str1和str2最长公共子串，因此:

LCP(i,j) = LCP(k,k+1) = height[k+1], 找出这个 k+1即可

4) 求出str的sa和height，找一个满足 sa[i-1] 和 sa[i] 在分隔字符 '-' 的两侧的最大的height[i]，该 height[i] 就是问题答案。

复杂度 O(nlogn)

5) 如果用上述模板，注意要在拼接后的str后面再加一个'\0'，并且将'\0'当作str的一部分。测试："yka"和"ykd"。OJ数据弱，没此情况

例题2:　最长回文子串

给定一个字符串，求最长回文子串的长度

例题2:　最长回文子串

给定一个字符串，求最长回文子串的长度

等价于求一个字符串和它的倒序的最长公共子串的长度

例题3: POJ3450 Corporate Identity

给定n(n <= 4000)个长度不超过200的字符串，求最长公共子串

```
Sample Input
3
aabbaabb
abbababb
bbbbbabb
2
xyz
abc
0
Sample Output
abb
IDENTITY LOST       //无公共字串
```

例题3: POJ3450 Corporate Identity

- 二分答案。对假设的答案L进行验证。

- 如果用 KMP算法验证(用第一个字符串的所有长度为L的子串去和所有其它字符串匹配)，复杂度为:

  log(200)*200*n*400

例题3:　POJ3450 Corporate Identity

- 可以用 sa和height数组验证答案 L
- 把原字符串拼接起来，用不同的不会出现在这些字符串里面的字符(整数)分隔。要做到这一点，需要把原字符串变成一个 int 数组,最终拼成的也是int数组，称为 all
- 所有all的后缀的LCP都不会跨两个原字符串

例题3:   POJ3450 Corporate Identity

- n个原字符串的公共子串"XXXX"一定会是all中n个后缀的 LCP ， 假设有k (k>= n) 个后缀的 LCPL >= L, 且这些后缀都以 "XXXX" 打头，则这k个后缀的名次必然是连续的。

- L 可行 <=> 可以找到区间 [i,j] , 对任何 k 属于 [i+1,j]，height[k] >= L, 且任一原字符串，都被名次位于[i,j]内的某个后缀的位置(在all中的起始下标) 覆盖(即这些后缀的来源涵盖了所有原串）。

- 假设原字符串x在all中，就是all[i,j]这一段，则对于任何整数 k 属于[i,j] ， 称 k 覆盖了x

例题3: POJ3450 Corporate Identity

- 验证L是否可行的方法就是从 下标 1 开始遍历 height数组，看是否能找到一个区间 [i,j], 对任何 k 属于 [i,j], height[k] >= L ,且名次位于该区间内的所有后缀的位置，覆盖了所有的原字符串(即这些后缀的来源涵盖了所有原串)

- 每个原字符串需要一个标记位，表示是否被覆盖。遍历height数组，每当元素由 >= L变为 < L 时，要清0所有标记位。

- 复杂度: (n*200)*log(n*200) + log(200)* (n * 200 + X)

- X 不好估计，是清0所有标记位的总时间

例题4:　POJ1743　Musical Theme

- 求一个字符串的不重叠的最长重复子串的长度

例题4: POJ1743 Musical Theme

- 求一个字符串的不重叠的最长重复子串的长度

➢ 二分答案，对假设的长度L，验证是否成立

➢ 验证方法：遍历 height数组，看能否找到一个区间 [i,j]， 对任何 k 属于 [i+1,j], height[k] >= L , 且该区间[i,j]内存在两个数m,n, sa[m]-sa[n] >= L