后缀数组是处理字符串的有力工具。

实现方法主要是两种：倍增法$O(nlogn)$和DC3法$O(n)$

本文主要介绍倍增法。

倍增法

定义第i个后缀：$s[i\dots n]$。它的后缀位置为i

定义变量：

s：原字符串。$s[i]$：原字符串的第i个字母。

n：字符串长度

m：字符集大小

$sa[i]$：**排名为i的后缀位置**

$rak[i]$：**第i个后缀的排名**

$x[i]$：**基数排序中第i个后缀的第一关键字的排名**

$y[i]$：**基数排序中第二关键字排名为i的后缀位置**(即第$y[i]$个后缀的第二关键词排名为i)

$c[i]$：基数排序的桶

sa和rak可以互推：$rak[sa[i]]=i$，$sa[rak[i]]=i$

接下来对后缀进行排序。

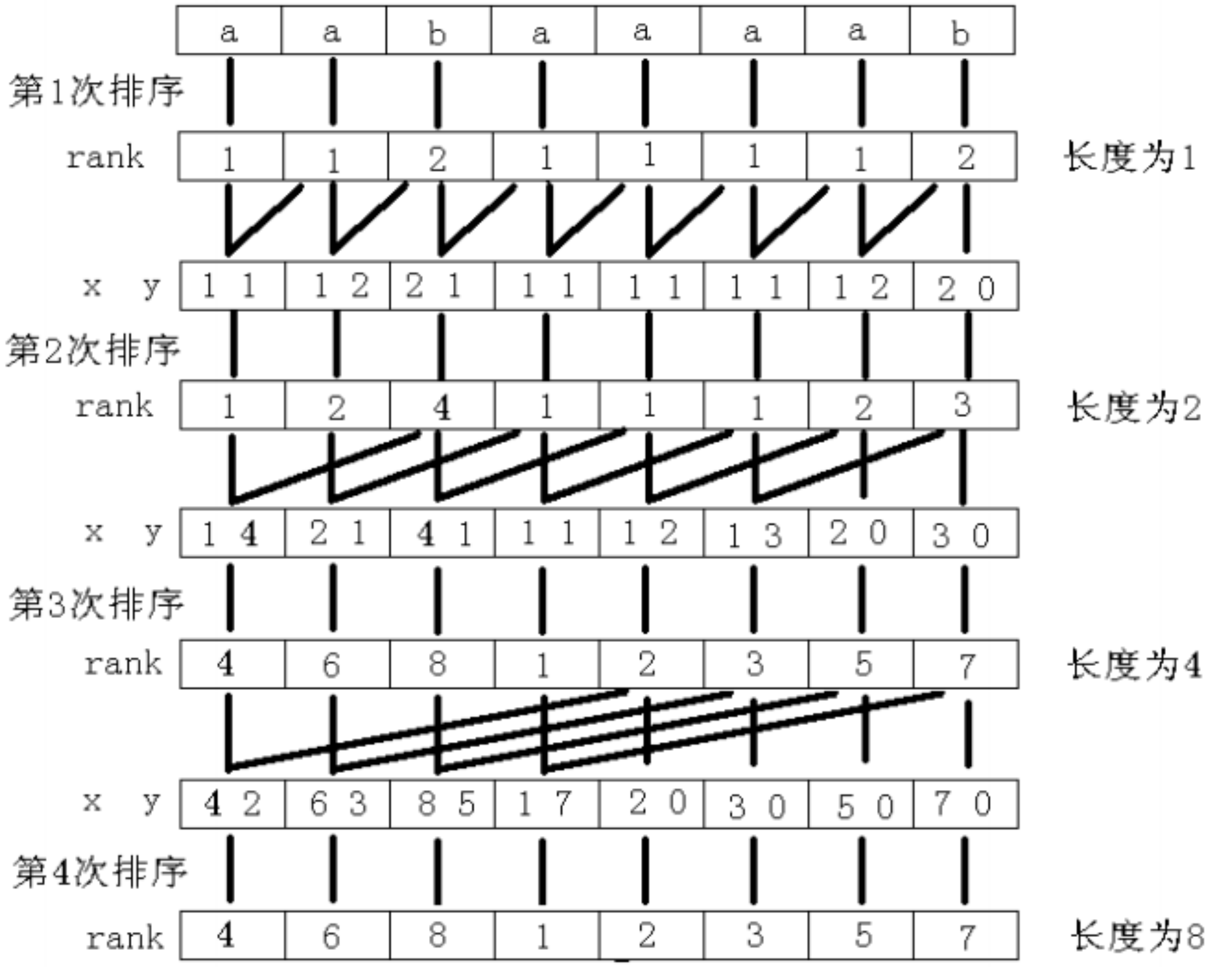
首先对所有后缀的第一个字母大小排名是能确定的，就是它的ASCII值。把第i个字母看成$(s[i],i)$的二元组，对它进行基数排序就得到了后缀第一个字母的大小排名$x[i]$。

后缀的第二个字母的排名可以通过后缀的第一个字母排名计算。具体的，**第i个后缀的第2个字母就是第$i+1$个后缀的第1个字母**，因此它们的排名相同。用第一个字母的排名和第二个字母的排名进行**基数排序**，得到所有长度为2的后缀的排名。

利用倍增的思想，接下来对每个后缀的前4个字母排序。上一轮得到前2个字母的排名；第i个后缀的第3、4个字母就是第$i+2$个后缀的第1、2个字母，即得到了第i个后缀后2个字母的排名。双关键字进行基数排序。

...

最后就能完成所有后缀的排序。整体排序过程：



具体代码解析

```
for (int i = 1; i <= n; i ++) c[x[i] = s[i]] ++ ;
for (int i = 2; i <= m; i ++) c[i] += c[i - 1] ;
for (int i = n; i; i --) sa[c[x[i]] --] = i ;
```

第一行：第i个后缀的第一个字母的排名$x[i]$即为字符串第i个字母：$x[i] = s[i]$

第二行：把排名(字母)对应的桶计数++，求前缀和，记录第一关键字排名为i的排名范围。

第三行：第一次对$(s[i], i)$二元组进行基数排序，从大到小枚举第二关键字i，找到它的排名对应的桶，这一二元组的排名即为该桶的大小。具体来说，**对于相同的第一关键字，第二关键字i越大，它就在相同的第一关键字中排名越后**，即在第二关键字对应的第一关键字排名的桶中越靠后，即该桶的大小。找到排名后，将该桶的大小减一。

倍增部分：

```
for (int k = 1; k <= n; k <<= 1) {
    int num = 0 ;
    for (int i = n - k + 1; i <= n; i ++) y[++ num] = i ;
```

```
    for (int i = 1; i <= n; i ++)
        if (sa[i] > k) y[++ num] = sa[i] - k ;
    memset (c, 0, sizeof c) ;
    for (int i = 1; i <= n; i ++) c[x[i]] ++ ;
    for (int i = 2; i <= m; i ++) c[i] += c[i - 1] ;
    for (int i = n; i; i --) sa[c[x[y[i]]] --] = y[i], y[i] = 0 ;
    swap (x, y) ;
    x[sa[1]] = 1; num = 1 ;
    for (int i = 2; i <= n; i ++)
        x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + k] == y[sa[i - 1] +
k]) ? num : ++ num ;
    if (num == n) return ;
    m = num ;
}
```

```
for (int k = 1; k <= n; k <<= 1)
```

倍增循环：k表示第一关键字和第二关键字长度为k

```
int num = 0 ;
for (int i = n - k + 1; i <= n; i ++) y[++ num] = i ;
```

num：第二关键字的排名计数

第二行：因为第$n-k+1$到第n个后缀**没有第二关键字**，因此它们的第二关键字排名最前（空串的排名最前），即排名为num的第二关键字的后缀位置为i。(参考y数组定义)

```
for (int i = 1; i <= n; i ++)
    if (sa[i] > k) y[++ num] = sa[i] - k ;
```

枚举上一轮排序后后缀的排名，**如果排名为i的后缀位置大于k，那么它可以作为别的后缀的后k个字母**。因为按照上一轮排序后后缀的排名枚举，所以该串作为别的后缀的后k个字母排名一定靠前。记录排名为num的第二关键字的后缀位置：**后k个字母的起始点为$sa[i]$，整个后缀的起始点即为$sa[i]-k$**。

```
memset (c, 0, sizeof c) ;
for (int i = 1; i <= n; i ++) c[x[i]] ++ ;
for (int i = 2; i <= m; i ++) c[i] += c[i - 1] ;
```

第一行：将上一轮排序的桶清空。

第二、三行：把第一关键字排名的桶计数++，求前缀和。(参考对第一个字母的排序过程)

```
for (int i = n; i; i --) sa[c[x[y[i]]] --] = y[i], y[i] = 0 ;
```

这行是对双关键字进行基数排序，比较难理解。

对比第一次双关键字基数排序：

```
for (int i = n; i; i --) sa[c[x[i]] --] = i ;
```

第一次双关键字排序是对$(x[i], i)$排序。第二关键字i越大，对应第一关键字中的排名越靠后。

```
for (int i = n; i; i --) sa[c[x[y[i]]] --] = y[i], y[i] = 0 ;
```

此行同理。i越大，第二关键字排名越大。$y[i]$记录的是第二关键字排名为i的后缀位置，$x[i]$记录的是后缀位置为i的第一关键字排名。$x[y[i]]$**即第二关键字排名为i对应的第一关键字排名。第二关键字排名越靠后，在对应的第一关键字排名就越靠后，排名即为该对应第一关键字排名的桶的大小，即**$c[x[y[i]]]$。$sa[i]$表示排名为i的后缀位置。当前排名为$c[x[y[i]]]$，后缀位置为$y[i]$(参考$y[i]$定义)，即$sa[c[x[y[i]]]] = y[i]$。最后将对应的第一关键字排名桶$c[x[y[i]]]$的计数减一。$sa[c[x[y[i]]] - -] = y[i]$

```
swap (x, y) ;
x[sa[1]] = 1; num = 1 ;
for (int i = 2; i <= n; i ++)
    x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + k] == y[sa[i - 1] + k])
? num : ++num;
```

此时要用之前的排序结果更新这一轮的排名$x[i]$。sa数组已在基数排序时更新完毕，利用sa来更新x。

y数组已没用，因此直接交换x和y数组。或可以写成：

```
memcpy (y, x, sizeof x) ;
```

此时的x可以看做rak数组。$x[sa[1]] = 1$。(参考sa和rak互推公式)

随后for循环更新排名。因为可能会出现排名一样的情况，即前k个字母排名相同，后k个字母排名相同。

num统计出现了几个排名。

当$y[sa[i]] == y[sa[i-1]]$并且$y[sa[i]+k] == y[sa[i-1]+k]$时，即**排名为$i$的前$k$个字母排名与排名为$i-1$的前$k$个字母排名一致，排名为$i$的后$k$个字母与排名为$i-1$的后$k$个字母排名一致，它们的总排名一致**。否则排名为$num+1$，并把$num++$。

```
if (num == n) return ;
m = num ;
```

当排名个数等于n，即每个后缀排名都不同时，后缀排序结束。

将字符集大小赋值成排名个数。

Luogu P3809【模板】后缀排序

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e6 + 10 ;
char s[maxn] ;
int n, m ;
int sa[maxn], x[maxn], y[maxn], c[maxn] ;
void get_sa () {
    for (int i = 1; i <= n; i ++) c[x[i] = s[i]] ++ ;
    for (int i = 2; i <= m; i ++) c[i] += c[i - 1] ;
    for (int i = n; i; i --) sa[c[x[i]] --] = i ;
    for (int k = 1; k <= n; k <<= 1) {
        int num = 0 ;
        for (int i = n - k + 1; i <= n; i ++) y[++ num] = i ;
        for (int i = 1; i <= n; i ++)
            if (sa[i] > k) y[++ num] = sa[i] - k ;
        memset (c, 0, sizeof c) ;
        for (int i = 1; i <= n; i ++) c[x[i]] ++ ;
        for (int i = 2; i <= m; i ++) c[i] += c[i - 1] ;
        for (int i = n; i; i --) sa[c[x[y[i]]] --] = y[i], y[i] = 0 ;
        swap (x, y) ;
        x[sa[1]] = 1; num = 1 ;
        for (int i = 2; i <= n; i ++)
            x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + k] == y[sa[i - 1] + k]) ? num : ++ num ;
        if (num == n) return ;
        m = num ;
    }
}
int main() {
    scanf("%s", s + 1) ;
    n = strlen (s + 1); m = 128 ;
    get_sa () ;
    for (int i = 1; i <= n; i ++)
        printf("%d ", sa[i]) ;
    cout << endl ;
    return 0 ;
}
```

---

height数组

定义：

$lcp(x,y)$：第x个后缀和第y个后缀的最长公共前缀

$LCP(x,y)$：排名为x的后缀与排名为y的后缀的最长公共前缀

$height[i]$：$lcp(sa[i-1],sa[i])$ 排名为i的后缀和排名为$i-1$的后缀的最长公共前缀

$H[i]$：$height[rak[i]]$ 第i个后缀和它排名前一名的后缀的最长公共前缀

LCP的一些性质：

1. $LCP(i,j)=LCP(j,i)$
2. $LCP(i,i)=len(suffix(sa[i]))=n-sa[i]+1$
3. $LCP(i,j)=min_{i<k\leq j}LCP(k-1,k)=min_{i<k\leq j}height[k]$，$i<j$

H的性质：$H[i]\geq H[i-1]-1$

证明：设第$i-1$个后缀，排名在它前一位的是第k个后缀。

根据$height$的定义，第$i-1$个后缀和第k个后缀的最长公共前缀是$height[rak[i-1]]$

讨论第i个后缀和第$k+1$个后缀的关系。

1. 第$i-1$个后缀的首字母与第k个后缀的首字母不同。

   则$height[rak[i-1]]=0$，那么$height[rak[i]]\geq height[rak[i-1]]-1$恒成立。

2. 第$i-1$个后缀的首字母与第k个后缀的首字母相同。

   第i个后缀等于第$i-1$个后缀去掉首字母，第$k+1$个后缀等于第k个后缀去掉首字母。

   根据假设第k个后缀排名在第$i-1$个后缀的前一位，那么第$k+1$个后缀一定排在第i个后缀的前面，即$rak[k+1]<rak[i]$。

   因为第$i-1$个后缀和第k个后缀的最长公共前缀是$height[rak[i-1]]$，所以第i个后缀和第$k+1$个后缀的最长公共前缀就是$height[rak[i-1]]-1$

   与第i个后缀拥有最长公共前缀的后缀一定是与第i个后缀排名相邻的后缀，即为第$sa[rak[i]-1]$个后缀

   $\because rak[k+1]\leq rak[i]-1<rak[i],lcp(i,k+1)=height[rak[i-1]]-1$

   $\therefore lcp(i,sa[rak[i]-1])\geq height[rak[i-1]]-1$

   即$height[rak[i]]\geq height[rak[i]-1]-1$，$H[i]\geq H[i-1]-1$

$\therefore$ 得证

构造方法：

```
void get_height () {
    for (int i = 1; i <= n; i ++) rak[sa[i]] = i ;
    int k = 0 ;
    for (int i = 1; i <= n; i ++) {
        if (k) k -- ;
        int j = sa[rak[i] - 1] ;
        while (s[j + k] == s[i + k]) k ++ ;
        height[rak[i]] = k ;
    }
}
```

应用

1. 两个后缀的最大公共前缀

   $lcp(x, y) = LCP(rak[x], rak[y]) = min_{rak[x] < i \le rak[y]} height[i]$，设$rak[x] < rak[y]$

   用RMQ维护$height$，$O(1)$查询

2. 可重叠最长重复子串：$height$中的最大值

3. 不可重叠最长重复子串

   二分答案len，对$height$进行分组，要求每一组$height$的最小值$\ge len$。统计每一组中$sa[i]$和$sa[i-1]$的最小值和最大值，$max - min \ge len$即合法。

4. 本质不同的子串数量

   枚举后缀排名，排名为i的后缀对答案的贡献为$n - sa[i] + 1 - height[i]$