

后缀数组

算法思想

后缀数组有两个数组 sa 和 rk □

$sa[i]$ 表示将所有后缀排序后第 i 小的后缀的编号 □ $rk[i]$ 表示后缀 i 的排名。

所以显然 $sa[rk[i]] = rk[sa[i]] = i$ □

显然暴力排序求这俩数组的做法是 $n^2 \log n$ 的。

有一个倍增算法，是利用上一次前和后两个段的排序名次作为第一第二关键字，进行新一轮排序，这样需要排序 $\log n$ 次，每次如果是 $sort$ 排序，单次复杂度是 $n \log n$ 次，这样是 $\log^2 n$ 的。用基数排序可以 $O(n)$ 排出结果，这样复杂度就是 $O(n \log n)$ 的。

重要定理 □ $height$ 数组表示 $lcp(sa[i], sa[i-1])$ □ 其中 lcp 数组表示最长公共前缀。

我们有 $height[rk[i]] \geq height[rk[i-1]] - 1$ □

求 $height$ 数组的复杂度是 $O(n)$ 的。

$lcp(sa[i], sa[j]) = \min\{height[i+1 \dots j]\}$

可以处理两个子串的最长公共前缀的问题，比如如果 $A = S[a \dots b], B = S[c \dots d]$ □ 现在分情况讨论：

如果 $lcp(a, c) \geq \min(|A|, |B|)$ □ 则 $A < B$ 等价于 $|A| < |B|$ □

否则 □ $A < B$ 等价于 $rk[a] < rk[c]$ □

还可以求不同子串的数目：

子串在后缀数组里，就是后缀数组的前缀，所以我们可以枚举后缀，然后计算前缀的总数，再减掉重复的。

前缀总数是子串个数就是 $\frac{n(n+1)}{2}$ □ 然后按照后缀排序的顺序枚举后缀，每次新增的子串就是除了与上一个后缀的 lcp 之外的其他的前缀一定是新增的 □ lcp 的部分在上一个后缀已经计算过了。所以答案是 $\frac{n(n+1)}{2} - \sum_{i=2}^n height[i]$ □

如果有一个让求出现了 k 次的子串，就求出 $height$ 数组，如果相邻的 $k-1$ 个位置的值的的最小值大于 0 ，就说明有字符串，如果求其中的最大长度，就求所有 $k-1$ 个相邻位置的 $height$ 值的的最小值，再在其中取最大值。

如果问是否有某字符串在文本串中至少不重叠地出现了两次，这个东西显然可以二分长度 $|s|$ □ 然后对于 $height$ 数组，求出所有连续 lcp 大于等于 s 的段，然后对于每一个段求出他们最大和最小的下标，然后做差如果大于 $|s|$ □ 则一定有字符串不重叠地出现了两次，这个可以求出满足条件字符串的长度最大值和是哪个字符串。这个方法非常实用！

算法实现

```
char s[maxn];
```

```
int y[maxn],x[maxn],c[maxn],sa[maxn],rk[maxn],height[maxn];
int n,m;
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>=1; --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;
        for (int i=1; i<=m; ++i) c[i]=0;
        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>=1; --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) break;
        m=num;
    }
    /*
    for (int i=1; i<=n; ++i) {
        printf("%d ",sa[i]);
    }
    putchar(10);
    */
}
void get_height() {
    int k=0;
    for (int i=1; i<=n; ++i) rk[sa[i]]=i;
    for (int i=1; i<=n; ++i) {
        if (rk[i]==1) continue;
        if (k) --k;
        int j=sa[rk[i]-1];
        while (j+k<=n && i+k<=n && s[i+k]==s[j+k]) ++k;
        height[rk[i]]=k;
    }
    /*
    putchar(10);
    for (int i=1; i<=n; ++i) {
        printf("%d ",height[i]);
    }
    */
}
```

代码练习

1. <https://www.luogu.com.cn/problem/P4051>

题目大意

给一个字符串，长度不超过 10^5 。将这个字符串循环放置，让求将这些循环出来的字符串按照字典序从小到大排序，并取最后一个字符接成一个新的字符串，让求这个新的字符串是什么。

题目解析

设原来的长度为 len 。既然是循环放置，我们自然考虑倍增。那么比较这些字符串的大小，只需要取出长度大于原长度的所有后缀，分别取他们的前 len 项，但是注意到，我们比较他们的大小，相当于比较这些后缀的大小。比如现在有两个题目给的串 s_1, s_2 。不妨设 $s_1 > s_2$ 。然后加上后缀，我们必然有 $s_11 > s_22$ 。因为这两个字符串在 len 长度的时候就已经比出来了 $s_1 > s_2$ 。反之一样，如果 $s_1 = s_2$ 。那么我们无所谓哪个放在前面，因为最后一位是一样的，所以就是倍增之后求后缀数组，存一下位置之后按 rk 排序，之后取走每个位置的最后一个字符就可以了。

```
#include <bits/stdc++.h>
#define maxn 200050
using namespace std;
char s[maxn];
int y[maxn], x[maxn], c[maxn], sa[maxn], rk[maxn], height[maxn];
int n, m;
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>=1; --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;
        for (int i=1; i<=m; ++i) c[i]=0;
        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>=1; --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) break;
        m=num;
    }
    /*
    for (int i=1; i<=n; ++i) {
```

```
        printf("%d ",sa[i]);
    }
    putchar(10);
    */
}
void get_height() {
    int k=0;
    for (int i=1; i<=n; ++i) rk[sa[i]]=i;
    for (int i=1; i<=n; ++i) {
        if (rk[i]==1) continue;
        if (k) --k;
        int j=sa[rk[i]-1];
        while (j+k<=n && i+k<=n && s[i+k]==s[j+k]) ++k;
        height[rk[i]]=k;
    }
    /*
    putchar(10);
    for (int i=1; i<=n; ++i) {
        printf("%d ",height[i]);
    }
    */
}

struct Node {
    int id,val;
}node[maxn];

int cmp(Node a,Node b) {
    return a.val<b.val;
}

int main() {
    scanf("%s",s+1);
    n=strlen(s+1);
    for(int i=1;i<=n;i++) {
        s[i+n]=s[i];
    }
    n<<=1;
    m=257;
    get_SA();
    for(int i=1;i<=n;i++) {
        rk[sa[i]]=i;
    }
    for(int i=1;i<=n;i++) {
        node[i].id=i;
        node[i].val=rk[i];
    }
    sort(node+1,node+n/2+1,cmp);
    for(int i=1;i<=n/2;i++) {
        printf("%c",s[node[i].id+n/2-1]);
    }
}
```

```

    }
    return 0;
}

```

其实最小移动位置也是可以这么求的，只不过复杂度变成了 $O(n \log n)$ 的。

2.如何在线在字符串中找出模式串

设主串是 T 模式串是 S 我们先构造出主串的后缀数组，注意到如果模式串出现在主串中，一定是作为这个主串某些后缀的前缀，这样我们可以二分 S 比较子串 S 和当前后缀的时间复杂度是 $O(|S|)$ 的，于是这个复杂度就是 $|S| \log |T|$ 的。如果要求出现的次数，可以在后面补一个很大的字符，然后两次二分做差就可以了，输出位置取两次二分出来的排名，然后取左闭右开区间这些排名的 sa 数组位置，输出就好了。因为后缀平衡树是可以实现后缀数组的操作的，所以这个东西后缀平衡树也可以，已经在那篇里写过了。

3.<https://www.luogu.com.cn/problem/P2870>

题目大意

给定一个长度为 n 的字符串，要求构造一个新的字符串，构造规则是：每次取出原字符串的首或尾巴，排到新字符串的尾巴，然后要新的字符串字典序最小

题目解析

通过大眼观察法就可以知道，其实我们每次判断的依据无非是剩下字符串是从头到尾字典序小还是从尾到头字典序小，就这么贪心地取，因为这样从字典序小的那边取一定不会比从字典序大的那边取差，所以是没问题的。直接硬比是 $O(n^2)$ 的。这时就有一种想法，顺着求后缀数组并且逆着求后缀数组，但是这样怎么比较哪个小呢，这么比依然是 $O(n^2)$ 的。哦，那就逆着接到后面一起求后缀数组就好了啊，是我太蠢了...这时和上面的类似，我们每次比较的是想同长度的串，所以后缀之间的比较就和那两个串的比较结果是一样的（相同的串取哪个都无所谓）。最后就设两个队头，直接比较他们的 rk 数组值就好了，小的放在队里面，最后把队输出就行了。

```

#include <bits/stdc++.h>
#define maxn 1000050
using namespace std;
char s[maxn];
int y[maxn],x[maxn],c[maxn],sa[maxn],rk[maxn],height[maxn];
int n,m;
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>=1; --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;
        for (int i=1; i<=m; ++i) c[i]=0;
        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>=1; --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) break;
    m=num;
}
/*
for (int i=1; i<=n; ++i) {
    printf("%d ",sa[i]);
}
putchar(10);
*/
}

void get_height() {
    int k=0;
    for (int i=1; i<=n; ++i) rk[sa[i]]=i;
    for (int i=1; i<=n; ++i) {
        if (rk[i]==1) continue;
        if (k) --k;
        int j=sa[rk[i]-1];
        while (j+k<=n && i+k<=n && s[i+k]==s[j+k]) ++k;
        height[rk[i]]=k;
    }
    /*
    putchar(10);
    for (int i=1; i<=n; ++i) {
        printf("%d ",height[i]);
    }
    */
}

struct Node {
    int id,val;
}node[maxn];
int bj[maxn];
int cmp(Node a,Node b) {
    return a.val<b.val;
}
char anss[maxn];
int main() {
    scanf("%d",&n);
    for(int i=1;i<=n;i++) {
        cin>>s[i];
    }
    for(int i=1;i<=n;i++) {
        s[i+n]=s[n-i+1];
    }
}
```

```
    }
    n<<=1;
    //printf("%s\n",s+1);
    m=257;
    get_SA();
    for(int i=1;i<=n;i++) {
        rk[sa[i]]=i;
    }
    int h1=1,h2=n/2+1;
    int cnt=0;
    while(cnt<n/2) {
        if(rk[h1]<rk[h2]) {
            anss[++cnt]=s[h1++];
        }else{
            anss[++cnt]=s[h2++];
        }
    }
    int L=strlen(anss+1);
    int cc=0;
    for(int i=1;i<=L;i++) {
        putchar(anss[i]);
        cc++;
        if(cc>=80) {
            cc=0;
            putchar('\n');
        }
    }
    return 0;
}
```

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link: https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:%E7%8E%8B%E6%99%BA%E5%BD%AA:%E5%90%8E%E7%BC%80%E6%95%B0%E7%BB%84&rev=1627296791

Last update: 2021/07/26 18:53