这个悲剧的东西在经历了半个月的研究后，终于研究明白了……

学会了之后突然发现，这个东西实在是太 nb 了~~

首先 还是先说一说这个东西的精髓~所谓后缀数组就是把一个整串的所有后缀的信息以一种比较高效的方式提取出来。而这种工具就是后缀数组。为什么我们偏偏选择了 后缀呢？因为我们发现了了一个非常不错的现象——整个串的所有子串实际上就是所有后缀的一个个前缀。这样，几乎所有类型的字符串问题都可以变成后缀数组 了。

下面给出后缀数组的定义：

定义数组 SA[i]表示将所有一个串的所有的后缀排序之后(排序的规则：字典序，如果前面的东西都一样，那就看字符串的长度)，排在第 i 位上的后缀的首字符在字符串中的位置、

同样定义一个 rank 数组，rank[i]表示 i 在后缀数组中的位置，因此我们知道 sa 和 rank 的关系是**一一对应**的。

求这个东西看起来很简单，但是实际上当我们仔细想想看，我们发现如果直接快排求 sa 会很悲剧。原因在于在比较两个字符串的大小时我们会遇到麻烦。我们无法保证时间复杂度是 O(nlogn)的。所以我们需要一种更好的算法来求后缀数组 SA。

比较常见的两种方法一种是倍增算法，一种是 DC3 算法。前者时间复杂度为 O(nlogn)，后者为 O(n)。但是在实测当中，两者相差不多，并且后者的编程复杂度十分高，所以我们选择前者。

倍增算法当然就是用倍增的思想来构造后缀数组。思想大概是这样的：**每一次取上一次排序长度的二倍的字符串进行处理，这样我们可以通过将上轮排序中的结果倍增，通过两个关键字的比较而得到新的顺序。**直到所有的后缀的 SA 值都已求出，在这过程中 rank 数组同样也被求出来了。

P.s.想法很简单，但是在实际写出来的时候并没有想象中的那么简单。建议读者自己尝试着写一写。

贴下求 sa 和 rank 的代码：

[delphi]view plaincopyprint?

```
1. procedure  suffix_init;
2. var
3.     i,p,m,j:longint;
4. begin
5.     fillchar(sum,sizeof(sum),0);
```

```
6. for  i:=1to  ndo
7. begin
8.             rank[i]:=ord(s[i])-ord('a')+1;
9.             inc(sum[rank[i]]);
10.end;
11.for  i:=1to26do  inc(sum[i],sum[i-1]);
12.for  i:=ndownto1do
13.begin
14.            sa[sum[rank[i]]]:=i;
15.            dec(sum[rank[i]]);
16.end;
17.   m:=0;
18.for  i:=1to  ndo
19.begin
20.if  rank[sa[i]]<>rank[sa[i-1]]then  inc(m);
21.            tmprank[sa[i]]:=m;
22.end;
23.   rank:=tmprank; j:=1;
24.while  m<ndo
25.begin
26.            fillchar(sum,sizeof(sum),0);
27.            p:=0;
28.for  i:=n-j+1to  ndobegin  inc(p); tmp[p]:=i;end;
29.for  i:=1to  ndo
30.if  sa[i]>jthenbegin  inc(p); tmp[p]:=sa[i]-j;end;
31.for  i:=1to  ndo
32.begin
33.                    tmprank[i]:=rank[tmp[i]];
34.                    inc(sum[tmprank[i]]);
35.end;
36.for  i:=1to  mdo  inc(sum[i],sum[i-1]);
37.for  i:=ndownto1do
38.begin
39.                    sa[sum[tmprank[i]]]:=tmp[i];
40.                    dec(sum[tmprank[i]]);
41.end;
42.            m:=0;
43.for  i:=1to  ndo
44.begin
45.if  (rank[sa[i]]<>rank[sa[i-1]])or  (rank[sa[i]+j]<>rank[sa[i-1
   ]+j])then  inc(m);
46.                    tmprank[sa[i]]:=m;
47.end;
48.            rank:=tmprank;
```

```
49.             j:=jshl1;
50. end;
51. end;
```

写到这里并不是结束，因为很多情况下我们的操作是和前缀有关的，所以经常我们会维护公共前缀的长度。这个要怎么做呢？

我们定义这样一个数组 height[i]，表示 suffix(sa[i])和 suffix(sa[i-1])的最长公共前缀。

这个 height 的用法就是，比如你要求 suffix(i)和 suffix(j)的最长公共前缀。这个问题就变成了 rmq(height,rank[i],rank[j])。为什么请读者思考~

求 法当然不能一个个来，有一种相当高效的方法。想法是从一个前缀过度到后一个前缀的过程实际上是去掉了第一个字符。所以维护一个 j，然后按照字符串的原始顺 序进行扫描，每次比较 rank[i]和 rank[i]-1 比较公共部分然后 inc(j)。当然这样 height[i]就是 j 了，当然不能这样结束，每次结 束的时候都 dec(j)，这样下一次只需要从 j 开始比较了，因为公共前缀的长度至少为 j。

代码：

[delphi]view plaincopyprint?

```
1. procedure  calc_height;
2. var
3.    i,j,k:longint;
4. begin
5.    j:=0;
6. for  i:=1to  ndo
7. begin
8. if  rank[i]=1then  continue;
9.            k:=sa[rank[i]-1];
10.while  s[i+j]=s[k+j]do  inc(j);
11.           height[rank[i]]:=j;
12.if  j>0then  dec(j);
13.end;
14.end;
```