[**后缀数组解析以及应用(附详解代码)**](http://blog.csdn.net/yxuanwkeith/article/details/50636898)

后缀数组是一个比较强大的处理**字符串**的算法，是有关字符串的**基础算法**，所以**必须**掌握。   
学会**后缀自动机(SAM)**就不用学**后缀数组(SA)**了？**不**，虽然SAM看起来更为强大和全面，但是有些SAM解决不了的问题能被SA解决，只掌握SAM是远远不够的。   
……

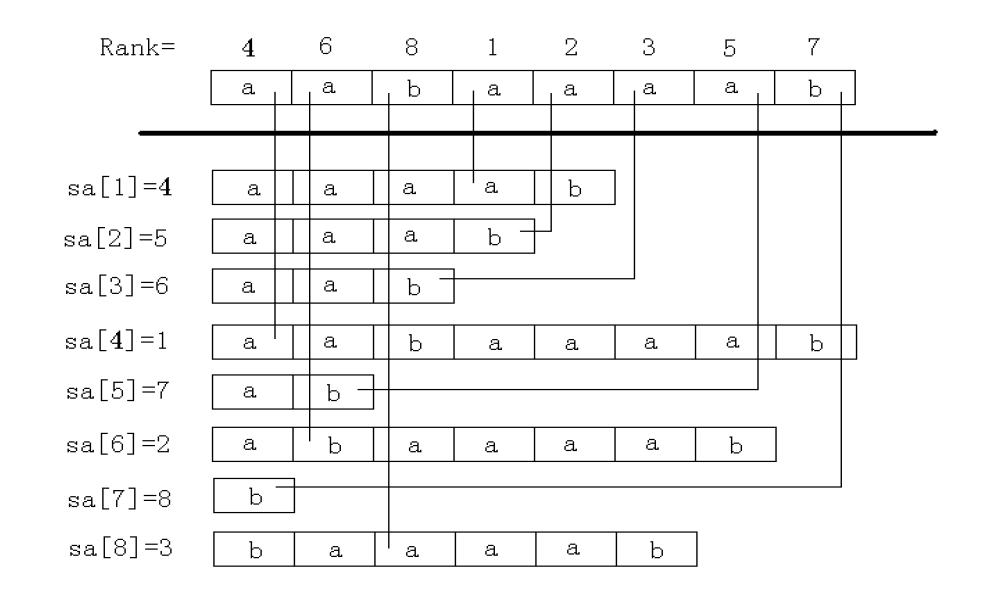
有什么SAM做不了的例子？   
比如果求一个串后缀的lcp方面的应用，这是SA可以很方便的用rmq来维护，但是SAM还要求lca，比较麻烦，还有就是字符集比较大的时候SA也有优势。

现在这里放道题，看完这个blog可能就会做了！：   
你可想想这道题：你有一个01串S，然后定义一个前缀最右边的位置就是这个前缀的结束位置。现在有q多个询问，每个询问结束位置在l~r中不同前缀的最长公共后缀是多长？   
|S|,q≤100000   
时限4s

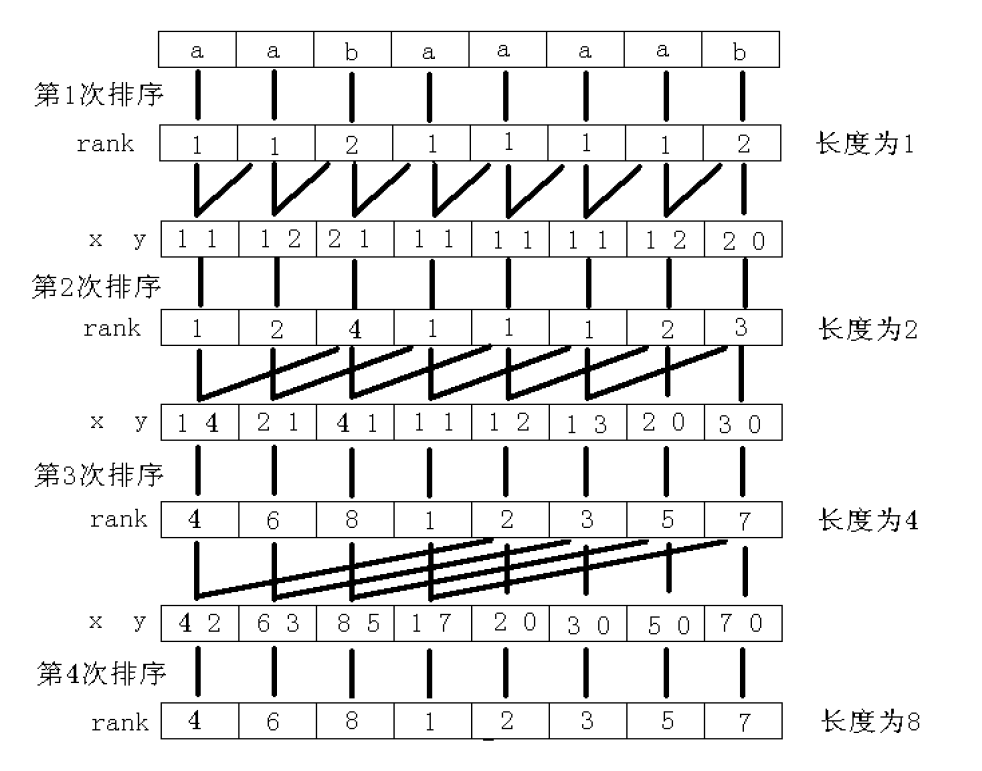
而下面是我对后缀数组的一些理解

**构造后缀数组——SA**

先定义一些变量的含义

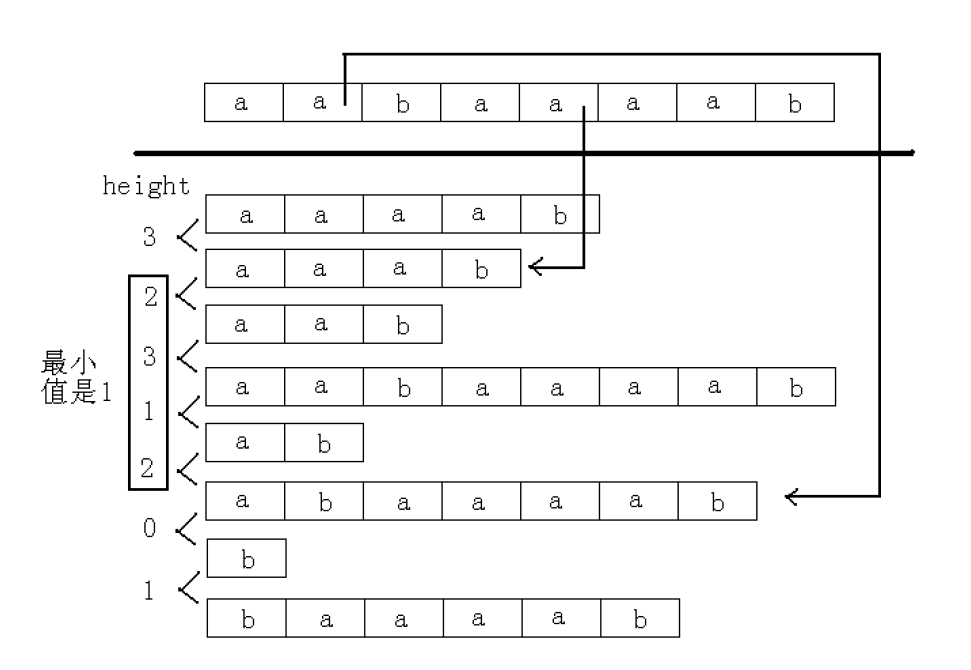
**Str ：需要处理的字符串(长度为Len)**   
**Suffix[i] ：Str下标为i ~ Len的连续子串(即后缀)**   
**Rank[i] : Suffix[i]在所有后缀中的排名**   
**SA[i] : 满足Suffix[SA[1]] < Suffix[SA[2]] …… < Suffix[SA[Len]],即排名为i的后缀为Suffix[SA[i]] (与Rank是互逆运算)**   
好，来形象的理解一下   
   
**后缀数组**指的就是这个**SA[i]**,有了它，我们就可以实现一些很强大的功能(如不相同子串个数、连续重复子串等)。如何快速的到它，便成为了这个算法的**关键**。而**SA**和**Rank**是互逆的，只要求出任意一个，另一个就可以O(Len)得到。   
现在比较主流的算法有两种，**倍增**和**DC3**，在这里，就主要讲一下稍微慢一些，但比较好实现以及理解的倍增算法(虽说慢，但也是**O(Len logLen)**)的。

进入正题——倍增算法

倍增算法的**主要思想** ：对于一个后缀**Suffix[i]**,如果想直接得到**Rank**比较困难，但是我们可以对每个字符开始的长度为2k的字符串求出排名，k从0开始每次递增1(每递增1就成为**一轮**)，当2k大于Len时，**所得到的序列就是Rank，而SA也就知道了**。**O(logLen)枚举k**  
这样做有什么**好处**呢？   
设每一轮得到的序列为**rank**(注意**r**是**小写**，最终后缀排名**Rank**是**大写**)。有一个**很美妙的性质**就出现了！**第k轮的rank**可由**第k - 1轮的rank**快速得来!   
为什么呢？为了方便描述，**设SubStr(i, len)为从第i个字符开始，长度为len的字符串**我们可以把第k轮**SubStr(i,**2k**)**看成是一个由**SubStr(i,**2k−1**)**和**SubStr(i +**2k−1**,**2k−1**)**拼起来的东西。类似**rmq算法**，这两个长度而2k−1的字符串是**上一轮遇到过的！当然上一轮的rank也知道！**那么吧每个这一轮的字符串都转化为这种形式，并且大家都知道字符串的比较是从左往右，**左边和右边的大小我们可以用上一轮的rank表示，那么……这不就是一些两位数(也可以视为第一关键字和第二关键字)比较大小吗!再把这些两位数重新排名就是这一轮的rank**。   
我们用下面这张经典的图理解一下：   
   
相信只要理解字符串的比较法则(跟实数差不多)，理解起来并不难。#还有一个细节就是怎么把这些两位数排序？这种位数少的数进行排序毫无疑问的要用一个**复杂度为长度\*排序数的个数**的优美算法——**基数排序**(对于**两位数**的数复杂度就是**O(Len)**的)。   
**基数排序原理** ： 把数字依次按照由**低位到高位**依次排序，排序时只看当前位。对于每一位排序时，因为上一位已经是有序的，所以这一位相等或符合大小条件时就不用交换位置，如果不符合大小条件就交换，实现可以用”桶”来做。(叙说起来比较奇怪，看完下面的代码应该更好理解，也可以上网查有关资料)   
好了**SA**和**Rank(大写R)**到此为止就处理好了。(下面有**详解代码！**)。但我们发现，只有这两样东西好像没什么用，为了处理重复子串之类的问题，我们就要引入一个表示**最长公共前缀**的新助手**Height数组！**

**构造最长公共前缀——Height**

同样先是定义一些变量

**Heigth[i] : 表示Suffix[SA[i]]和Suffix[SA[i - 1]]的最长公共前缀，也就是排名相邻的两个后缀的最长公共前缀**   
**H[i] : 等于Height[Rank[i]]，也就是后缀Suffix[i]和它前一名的后缀的最长公共前缀**   
而两个**排名不相邻**的最长公共前缀定义为排名在它们之间的**Height的最小值**。   
跟上面一样，先形像的理解一下：   


高效地得到Height数组

如果一个一个数按**SA中的顺序**比较的话复杂度是**O(**N2**)**级别的，想要快速的得到Height就需要用到一个关于**H数组**的性质。   
**H[i] ≥ H[i - 1] - 1!**   
如果上面这个性质是对的，那我们可以按照**H[1]、H[2]……H[Len]**的顺序进行计算，那么复杂度就降为**O(N)**了！   
**让我们尝试一下证明这个性质**: 设**Suffix[k]**是排在**Suffix[i - 1]**前一名的后缀，则它们的最长公共前缀是**H[i - 1]**。都**去掉第一个字符**，就变成**Suffix[k + 1]**和**Suffix[i]**。**如果H[i - 1] = 0或1**,那么**H[i] ≥ 0**显然成立。**否则**，**H[i] ≥ H[i - 1] - 1(去掉了原来的第一个,其他前缀一样相等)**，所以**Suffix[i]**和在它前一名的后缀的最长公共前缀**至少是H[i - 1] - 1。**   
仔细想想还是比较好理解的。H求出来，那Height就相应的求出来了，这样**结合SA，Rank和Height我们就可以做很多关于字符串的题了！**

**代码——Code**

建议复制到自己的编程软件上看

/\*

Problem: JZOJ1598(询问一个字符串中有多少至少出现两次的子串)

Content: SA's Code and Explanation

Author : YxuanwKeith

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int MAXN = 100005;

char ch[MAXN], All[MAXN];

int SA[MAXN], rk[MAXN], Height[MAXN], sum[MAXN], trk[MAXN], a[MAXN], n, m;

char str[MAXN];

//rk[i] 第i个后缀的排名; SA[i] 排名为i的后缀位置; Height[i] 排名为i的后缀与排名为(i-1)的后缀的LCP

//sum[i] 计数排序辅助数组; trk[i] rk的辅助数组(计数排序中的第二关键字),与SA意义一样。

//a为原串

void RSort() {

//rk第一关键字,trk第二关键字。

for (int i = 0; i <= m; i ++) sum[i] = 0;

for (int i = 1; i <= n; i ++) sum[rk[trk[i]]] ++;

for (int i = 1; i <= m; i ++) sum[i] += sum[i-1];

for (int i = n; i >= 1; i --) SA[sum[rk[trk[i]]] --] = trk[i]; //确保满足第一关键字的同时，再满足第二关键字的要求

} //计数排序,把新的二元组排序。

int cmp(int \*f, int x, int y, int w) { return f[x] == f[y] && f[x + w] == f[y + w]; }

//通过二元组两个下标的比较，确定两个子串是否相同

void Suffix() {

//SA

for (int i = 1; i <= n; i ++) rk[i] = a[i], trk[i] = i;

m = 127 ,RSort(); //一开始是以单个字符为单位，所以(m = 127)

for (int w = 1, p = 1, i; p < n; w += w, m = p) { //把子串长度翻倍,更新rk

//w 当前一个子串的长度; m 当前离散后的排名种类数

//当前的trk(第二关键字)可直接由上一次的SA的得到

for (p = 0, i = n - w + 1; i <= n; i ++) trk[++ p] = i; //长度越界,第二关键字为0

for (i = 1; i <= n; i ++) if (SA[i] > w) trk[++ p] = SA[i] - w;

//更新SA值,并用trk暂时存下上一轮的rk(用于cmp比较)

RSort(), swap(rk, trk), rk[SA[1]] = p = 1;

//用已经完成的SA来更新与它互逆的rk,并离散rk

for (i = 2; i <= n; i ++) rk[SA[i]] = cmp(trk, SA[i], SA[i - 1], w) ? p : ++ p;

}

//离散：把相等的字符串的rk设为相同。

//LCP

int j, k = 0;

for(int i = 1; i <= n; Height[rk[i ++]] = k)

for( k = k ? k - 1 : k, j = SA[rk[i] - 1]; a[i + k] == a[j + k]; ++ k);

//这个知道原理后就比较好理解程序

}

void Init() {

scanf("%s", str);

n = strlen(str);

for (int i = 0; i < n; i ++) a[i + 1] = str[i];

}

int main() {

Init();

Suffix();

int ans = Height[2];

for (int i = 3; i <= n; i ++) ans += max(Height[i] - Height[i - 1], 0);

printf("%d\n", ans);

}

**4个比较基础的应用**

Q1：一个串中两个串的最大公共前缀是多少？   
A1：**这不就是Height吗？**用rmq预处理，再O(1)查询。   
   
Q2：一个串中可重叠的重复最长子串是多长？   
A2：就是求任意两个后缀的最长公共前缀，而任意两个后缀的最长公共前缀都是**Height 数组里某一段的最小值**，那最长的就是**Height中的最大值**。   
   
Q3：一个串种不可重叠的重复最长子串是多长？   
A3：先**二分答案**，**转化成判别式的问题**比较好处理。假设当前需要判别长度为k是否符合要求，只需把排序后的后缀**分成若干组**，其中**每组的后缀之间的Height 值都不小于k**，再判断其中有没有不重复的后缀，具体就是看**最大的SA值和最小的SA值相差超不超过k**，有一组超过的话k就是合法答案。   
   
A4：一个字符串不相等的子串的个数是多少？   
Q4：**每个子串一定是某个后缀的前缀，那么原问题等价于求所有后缀之间的不相同的前缀的个数**。而且可以发现每一个后缀**Suffix[SA[i]]**的贡献是**Len - SA[i] + 1**,但是有子串算重复，重复的就是**Heigh[i]个与前面相同的前缀**，那么减去就可以了。最后，一个后缀**Suffix[SA[i]]**的贡献就是**Len - SA[k] + 1 - Height[k]**。   
对于后缀数组更多的应用这里就不详细阐述，经过思考后每个人都会发现它的一些不同的用途，它的功能也许比你想象中的更强大！

**最开始的那道题**

先搬下来。。。

你可想想这道题：你有一个01串S，然后定义一个前缀最右边的位置就是这个前缀的结束位置。现在有很多个询问，每q个询问结束位置在l~r中不同前缀的最长公共后缀是多长？   
|S|,q≤100000   
时限4s

**简单**思路：首先可以把字符串反过来就是求后缀的最长公共前缀了，可以用SA求出height数组，然后用rmq预处理之后就是求两个位置间的最小值。然后对于一个区间，显然只有在SA数组中相邻的两个串可以贡献答案。   
对于区间询问的问题可以用莫队处理，然后考虑加入一个后缀应该怎么处理，我们可以维护一个按SA数组排序的链表。假设我们先把所有位置的SA全部加入，然后按顺序删除，重新按顺序加入时就可以O(1)完成修改。那么按照这个思路我们可以用固定左端点的并查集，做到只加入，不删除，然后用O(nn√+nlogn)的复杂度完成这道题。

\*可能后面的处理方式比较麻烦，如果直接用splay维护区间中的后缀的话可以做到O(nn√logn)，这个方法就比较直观，而SAM在个问题上还是有点无力的。这题只是为了说明SA相比于SAM还是有他的独到之处，特别是在处理后缀的lcp之类的问题上。

**结束**

以上就是我对后缀数组的理解 ——**YxuanwKeith**