后缀数组

倍增算法

主要思想就是先对单个字符排序，再根据单个字符排序的结果通过移位得到下一个单个字符的排序结果，再排序就可以得到两个字符组成的子串的排序结果；再移位和合并排序得到四个字符组成的子串的排序结果...因此要确定次序的子串长度是按指数增长的，加之使用基数排序每次的复杂度为$O(n)$,总的时间复杂度为$O(nlogn)$.

```
int wa[maxn],wb[maxn],wv[maxn],ws[maxn];
int cmp(int *r,int a,int b,int l) {
    return r[a]==r[b]&&r[a+l]==r[b+l];
}
void da(int *r,int *sa,int n,int m) {
    int i,j,p,*x=wa,*y=wb,*t;
    for(i=0;i<m;i++) ws[i]=0;
    for(i=0;i<n;i++) ws[x[i]=r[i]]++;
    for(i=1;i<m;i++) ws[i]+=ws[i-1];
    for(i=n-1;i>=0;i--) sa[--ws[x[i]]]=i;
    for(j=1,p=1;p<n;j*=2,m=p)
    {
        for(p=0,i=n-j;i<n;i++) y[p++]=i;
        for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
        for(i=0;i<n;i++) wv[i]=x[y[i]];
        for(i=0;i<m;i++) ws[i]=0;
        for(i=0;i<n;i++) ws[wv[i]]++;
        for(i=1;i<m;i++) ws[i]+=ws[i-1];
        for(i=n-1;i>=0;i--) sa[--ws[wv[i]]]=y[i];
        for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
            x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
    }
    return;
}
```

1. wa, wb, wv, ws四个数组为辅助数组，在程序运行时有不同的含义，之后再具体说明
2. 第2-4行的cmp函数作用是比较关键字（*在初始比较单个字符的顺序时，关键字为字符的ASCII码值；之后同时比较*$2^k(k \geq 1)$*个字符的顺序时，关键字为*2^{k-1}*个字符的次序*）是否相同，若相同，即两个字符(串)完全相同（*在长度为*$2^k(k \geq 1)$*的字符串比较时，*l*的值为*2^{k-1}*,可以理解为前*2^{k-1}*个字符次序相同且后*2^{k-1}*个字符次序也相同，即两个字符串完全一样*），返回1；若不相同，则返回0
3. 函数的声明中，r为原字符串,sa为后缀数组，n为原字符串的长度加一，原因是在原字符串后加了一个比所有字符次序都小的数(*比如0，主要是为了方便函数实现*)，m为基数排序的比较参数，一般字符串中只有字母数字(*在ASCII码表中的值*)等，设为128即可

4. 第7-10行为基数排序，将字符串拆成长度为1的n个字符，然后进行排序，最后将排序结果存在数组sa中，第10行for循环从$n-1$开始，目的在于，当遇到相同的字符时，在字符串中位置靠前的字符次序也靠前(*作用后面说*)
5. 第11行的for循环条件中，j每次变为2倍，即每次将前一次排好序的结果作为排序基础；p在这里有两个用途，一是作为数组递增的下标；二是作为排序后最靠后的子串的次序，如果为n即全部排好序，则退出循环。$m=p$这句是一个小优化，在以上一次比较得到的次序为关键字的排序中，排序参数最大为p(*上述p的第二个用途*)
6. 第13-14行实现了对第二关键字的排序，比如原字符的长度为1，那么第二关键字就是相应原字符后一个位置的字符。由于最后j个字符的第二关键字是空(或是0)，肯定是排在最前面的，第13行做了这一步；由于之前已经有了第一关键字的排序，只需将数组sa中的各数值左移一位即为按第二关键字排序的子串的起始位置
7. 第15行是将按照第二关键字顺序得到的子串的起始位置的名次保存在数组wv中，在之后的基数排序中用以确定两个子串的先后(*只有一个子串时保存的不是名次，是相应字符对应的数值，不过用途一样*)
8. 第16-19行是对已经按第二关键字排好序的子串再对第一关键字进行排序，注意sa数组从y中取值，因为y中保存了第二关键字已有序的子串的下标，最后一个循环中i从$n-1$开始，就使得对于两个相同的第一关键字，当前位置靠后的一个在最后的顺序中仍然靠后，这也就实现了第二关键字仍然有序。
9. 第20-21行即对已经有序的子串(*顺序信息存在sa数组中*)，找到它们相应的名次，注意此时两个完全相同的子串的名次应该是一样的，最后得到的数组x即保存了各子串的名次信息
10. 若干次循环后，所有的子串的次序都不同，便跳出循环，数组sa是后缀数组，数组x是名次数组

我也是看着例子模拟了一遍才搞懂的，论文里例子的字符串是$aabaaaab$，下面是根据这个字符串模拟了一遍算法:

此例中字符串为 aabaaaab0
int r[]={97,97,98,97,97,97,97,98,0}
int n=9

循环开始之前(第7-11行)
首先对单个字符进行基数排序

```
for(i=0;i<m;i++)  ws[i]=0; //初始化
for(i=0;i<n;i++)  ws[x[i]=r[i]]++;
```
//这个循环结束后，ws中存放的是值为r[i]的字符的个数。同时将r[i]中的元素复制到x[i]中，之后用于标记次序大小。
此例中，ws[97]=6, ws[98]=2, ws[0]=1. 其余均为0.
x=[97,97,98,97,97,97,97,98,0]

```
for(i=1;i<m;i++)  ws[i]+=ws[i-1];
```
//ws保存前i个数值为r[i]的字符的个数和，可以理解为数值为r[i]的字符的最大可能次序为ws[i].
此例中，ws[0~96]=1, ws[97]=7,
ws[98]=9, ws[99~127]=9.

```
for(i=n-1;i>=0;i--)  sa[--ws[x[i]]]=i;
```
//sa中存放的是第i个位置的字符在排序之后的位置。且两个相同的字符，在原字符串中越靠前，排序后的位置也越靠前。
此例中，sa=[8,0,1,3,4,5,6,2,7]
(相对应的字符：0, a_0, a_1, a_3, a_4, a_5, a_6, b_2, b_7).

循环开始执行
(第13-14行) 对第二关键字进行排序

```
for(p=0,i=n-j;i<n;i++) y[p++]=i;
```
//此例中此时 j=1, n=9, i=8. 则 y[0]=8, p=2.
即第8个位置(r中的0)无第二关键字，一定排在最前面。

```
for(i=0;i<n;i++)
    if(sa[i]>=j)
        y[p++]=sa[i]-j;
```
//由于sa[i]=0的字符不可能做为第二关键字(此例中a_1)而且单个字符的顺序已经排过，对第二关键字排序时只需将现有的sa中每个字符的位置左移一位，就是对第二关键字排好序的字符子串(此处长度为2)的顺序。
此例中 y=[8,7,0,2,3,4,5,1,6]
(对应的字符: 0, $b_2$0, a_1a_2, b_1a_3, a_3a_4, a_4a_5, a_5a_6, a_2b_1, a_6b_2)

(第15行) 存储按第二关键字排好序的第一关键字的对应数值

```
for(i=0;i<n;i++)  wv[i]=x[y[i]];
```
// wv=[0,98,97,98,97,97,97,97,97]

(第16-19行) 对第一关键字进行基数排序.

```
for(i=0;i<m;i++)  ws[i]=0;
for(i=0;i<n;i++)  ws[wv[i]]++;
for(i=0;i<m;i++)  ws[i]+=ws[i-1];
for(i=n-1;i>=0;i--)  sa[--ws[wv[i]]=y[i];
```
//从y中取值以保证第一关键字有序的条件下第二关键字也有序.
得到 sa=[8,0,3,4,5,1,6,7,2].
(对应的字符: 0, a_1a_2, a_3a_4, a_4a_5, a_5a_6, a_2b_1, a_6b_2, $b_2$0, b_1a_3.)

(第20-21行) 确定已经排好序的这些字符子串的顺序.

```
for(t=x,x=y,y=t,p=1;x[sa[0]]=0,i=1;i<n;i++)
    x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
```
//将2个字符的子串的次序确定下来，完全相同的子串有相同的名次.
将x,y交换，现在y中存储的是原字符串中相应位置各字符的数值.
x[sa[1]] = x[0] = cmp(y,8,0,1)?0:1 = 1
x[sa[2]] = x[3] = cmp(y,0,3,1)?1:2 = 1
x[sa[3]] = x[4] = cmp(y,3,4,1)?1:2 = 1
⋮
最终有 x=[1,2,4,1,1,1,2,3,0].
表示以2个字符为一个子串排序的名次数组

循环第二次执行.
此时 j=2.
(第13-14行)
//i=9-2=7. 即 y[0]=7, y[1]=8
表示原字符串中位置为7和8的字符无第二关键字
(此次循环中，第二关键字为4个字符组成的子串中的后2个)
因为已经按2个字符的子串排好序存在sa中，此时将相应的sa中的值左移两位即可.
有 y=[7,8,6,1,2,3,4,5,0]
(对应的字符为: $b_2$0, 0, $a_6b_2$0, $a_2b_1a_3a_4$, $b_1a_3a_4a_5$,
$a_3a_4a_5a_6$, $a_4a_5a_6b_2$, $a_5a_6b_2$0, $a_1a_2b_1a_3$)

(第15行)
//x中存的是子串(2字符)的名次，
wv中即可得到按第二关键字排序的各子串的名次.
wv=[3,0,2,2,4,1,1,1,1].

(第16-19行)
//仍然是按第一关键字排序.
这时不需要各字符的具体数值，只需知道次序即可确定是否相同
sa=[8,3,4,5,0,6,1,7,2].
(对应字符为 0, $a_3a_4a_5a_6$, $a_4a_5a_6b_2$, $a_5a_6b_2$0,
$a_1a_2b_1a_3$, $a_6b_2$0, $a_2b_1a_3a_4$, $b_2$0, b_1a_3a

(第20-21行)
//交换过后，y中存储的是2字符的子串的名次.
即 y=[1,2,4,1,1,1,2,3,0]
x[sa[1]] = x[3] = cmp(y,8,3,2)?0:1 = 1
x[sa[2]] = x[4] = cmp(y,3,4,2)?1:2 = 2
x[sa[3]] = x[5] = cmp(y,4,5,2)?2:3 = 3
⋮
有 x=[4,6,8,1,2,3,5,7,0]
表示以4个字符为子串的排序后的名次数组.
循环结束后 p=9, 即已经全部分出先后，便退出循环.

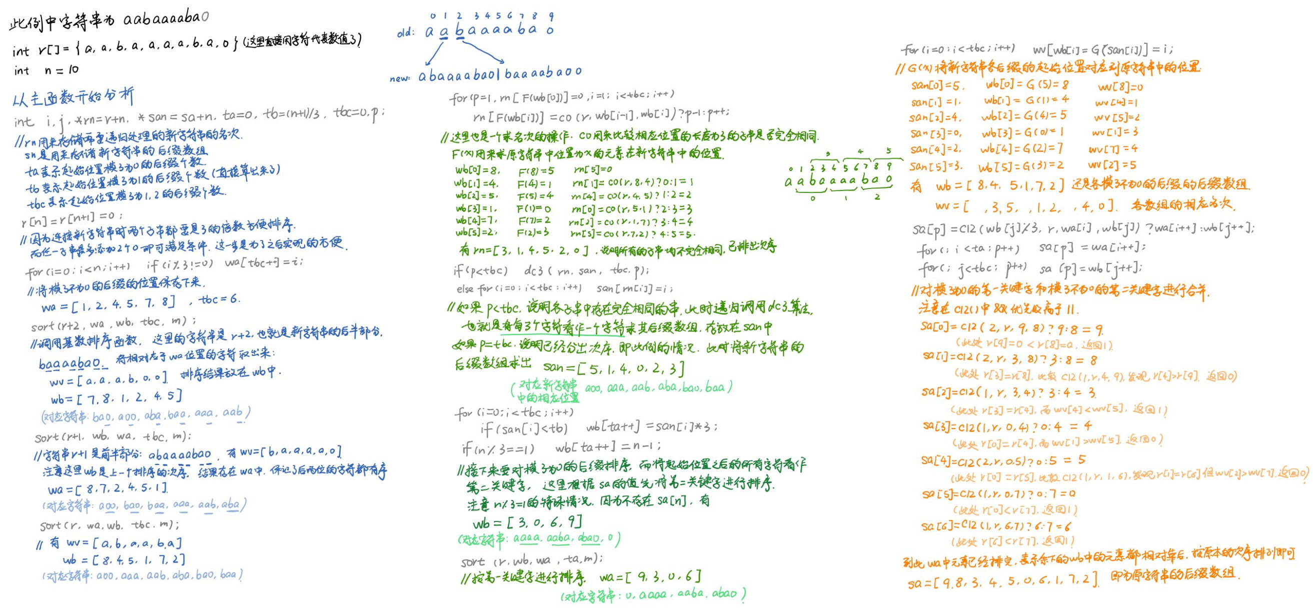
DC3算法

主要思想是将整个字符串分为两个部分，模3为0的后缀(记为A)和模3不为0的后缀(记为B)，然后对B的两部分(*模3为1和模3为2的部分*)进行连接，得到一个新字符串，然后每3个字符看作1个字符进行基数排序，得到一个次序(*如果存在完全相同的两个字符，那么递归调用这个函数，直到全部分出次序为止*)，最后将这个B部分的次序作为第二关键字与A部分排好序后的第一关键字进行合并，就可以得到最终的后缀数组

```
#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
int wa[maxn],wb[maxn],wv[maxn],ws[maxn];
int c0(int *r,int a,int b) {
    return r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];
}
int c12(int k,int *r,int a,int b) {
    if(k==2)
        return r[a]<r[b]||r[a]==r[b]&&c12(1,r,a+1,b+1);
    else
        return r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];
}
void sort(int *r,int *a,int *b,int n,int m) {
    int i;
    for(i=0;i<n;i++) wv[i]=r[a[i]];
    for(i=0;i<m;i++) ws[i]=0;
    for(i=0;i<n;i++) ws[wv[i]]++;
    for(i=1;i<m;i++) ws[i]+=ws[i-1];
    for(i=n-1;i>=0;i--) b[--ws[wv[i]]]=a[i];
    return;
}
void dc3(int *r,int *sa,int n,int m)
{
    int i,j,*rn=r+n,*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;
    r[n]=r[n+1]=0;
    for(i=0;i<n;i++) if(i%3!=0) wa[tbc++]=i;
    sort(r+2,wa,wb,tbc,m);
    sort(r+1,wb,wa,tbc,m);
    sort(r,wa,wb,tbc,m);
    for(p=1,rn[F(wb[0])]=0,i=1;i<tbc;i++)
    rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;
    if(p<tbc) dc3(rn,san,tbc,p);
    else for(i=0;i<tbc;i++) san[rn[i]]=i;
    for(i=0;i<tbc;i++) if(san[i]<tb) wb[ta++]=san[i]*3;
    if(n%3==1) wb[ta++]=n-1;
    sort(r,wb,wa,ta,m);
    for(i=0;i<tbc;i++) wv[wb[i]=G(san[i])]=i;
    for(i=0,j=0,p=0;i<ta && j<tbc;p++)
    sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];
    for(;i<ta;p++) sa[p]=wa[i++];
    for(;j<tbc;p++) sa[p]=wb[j++];
    return;
}
```

1. 第24行的参数，rn和san数组都是为了递归调用，rn保存了当前字符串的名次信息，san保存了当前字符串的后缀数组
2. 第25行是因为在把模3不为0的两个部分连接之前，要保证每个部分的长度都是3的倍数才能进行基数排序，而对任意长度，最多需要加2个0就可以保证是3的倍数，这里是为了之后可能使用的方便
3. 第26-29行，对新字符串3个字符为一组的子串进行基数排序，最后将后缀数组保存在wb中，注意中间数组wa和wb的位置由一个交换，是为了让对较高数位的排序结果同时依赖于较低数位已经排好序的结果。
4. 第30-31行，通过F函数将原字符串的后缀的起始位置对应到新字符串的起始位置(**注意新字符串是把3个字符看作1个字符的**)，求出在新字符串中各字符的名次数组(rn)
5. 第32-33行，如果此次排序，所有的字符都不完全相同，即已经全部出先后次序，那么直接求出其后缀数组保存在san中；如果存在完全相同的字符，那么递归调用该函数，结果仍然存在数组san中(*由于每次递归之前，新字符串的长度不大于原字符串长度的$2/3$,所以递归一定会停止*)
6. 第34-36行，将模3不为0的部分作为第二关键字，直接从当前的san中获取到次序信息(*注意$n\%3=1$的特殊情况，因为不存在$san[n]$，所以需要单独赋值*)，再用基数排序与模3为0的部分(第一关键字)进行合并，得到以模3为0部分进行排序的后缀数组，存放在wa中
7. 第37行，再将新字符串各字符的位置通过G函数对应到原字符串中，得到原字符串模3不为0的各后缀的后缀数组wb和名次数组wv
8. 第38-41行，将模3为0和模3不为0的两部分进行合并，注意当两个后缀数组wa和wb中有一个数组的元素全被取出来之后，说明另外一个后缀数组的次序都比较靠后，之前加在sa数组后面即可

论文里的例子是字符串$aabaaaaba$,下面也是模拟的一遍过程：



Height数组

简要定义

　　height数组是指在排好序的后缀中，相邻后缀的最长公共前缀。比如$rank[i]$的后缀$sa[rank[1]]$和$rank[i-1]$的后缀$sa[rank[i-1]]$，它们的最长公共前缀记为$height[rank[i]]$,简记为$h[i]$,也就是在字符串中起始位置为i的后缀与字典序中它前面一位的后缀的最长公共前缀的长度

重要性质

对于任意的$i \geq 1$,有$h[i] > h[i-1]-1$

证明：假设i和j满足$rank[i]-rank[j]=1$,那么这两个后缀的最长公共前缀长度即为$h[i]$,现在考虑原字符串中起始位置在j之后和i之后的两个后缀，这里记为$j+1$和$i+1$　　第一种情况：第i和j两个后缀的首字符不一样，那么$h[i]=0$,总有$h[i+1] \geq h[i]-1$　　第二种情况：第i和j两个后缀的首字符一样，那么第$i+1$和第$j+1$这两个后缀分别是由第i和第j个后缀去掉首字符得到的，显然第$j+1$个字符排在第$i+1$个后缀之前，它们的最长公共前缀长度为$h[i]-1$;再考虑比第$i+1$个后缀排序更靠前的后缀中，一定是与它相邻的后缀和它有最长的公共前缀(*可以理解为相似度更高*),也就是说名次为$rank[i+1]$(*这里的$i+1$是指在字符串里的位置*)的后缀和名次为$rank[i+1]-1$的后缀的最长公共前缀至少是$h[i]-1$(*因为第$j+1$个后缀至少也要在第$i+1$个后缀的前面一位*),即$h[i+1] \geq h[i]-1$(*这里i和$i+1$就是指在字符串里的位置*)　　这里举个栗子，对于字符串$aabaaaab$,它的$rank$数组为[4,6,8,1,2,3,5,7],我们假设$i=6, j=5$;那么$suffix(i)=abaaaab, suffix(j)=ab$,而第$i+1$个后缀为$baaaab$,第$j+1$个后缀为$b$,第$j+1$个后缀一定在$i+1$个后缀的前面，而且第$i+1$个后缀的$rank$值为8,第$j+1$个后缀的$rank$值为7，我们说$height[rank[7]] \geq height[rank[6]]-1$,这里因为第$j+1$个后缀的排名恰为第$i+1$个后缀的排名的前一个，所以取等号

代码实现

```
int rank[maxn],height[maxn];
void calheight(int *r,int *sa,int n)
{
    int i,j,k=0;
    for(i=1;i<=n;i++) rank[sa[i]]=i;
    for(i=0;i<n;height[rank[i++]]=k)
        for(k?k--:0,j=sa[rank[i]-1];r[i+k]==r[j+k];k++);
    return;
}
```

1. 第5行，根据字符串的后缀数组求出相应的名次数组
2. 第6-7行，当k等于0时，便从第一个字符开始逐个搜寻，之后按照名次数组的顺序依次找下一个要求的h值，由性质可知，此时不需要再从头搜索，只需要从第$k-1$个字符的位置开始搜索即可，如此，就可以得到所有的$height$数组的值