**字符串匹配相关**

**KMP**

char P[maxw];

char str[maxn];

int lenP,lenS;

int pi[maxw];

void pre()

{

int p,k;

k=-1;

pi[0]=-1;

for(p=1;p<=lenP;p++)

{

while(k!=-1&&P[k+1]!=P[p])

k=pi[k];

if(P[k+1]==P[p])

k=k+1;

pi[p]=k;

}

}

//最长覆盖子串：len-pi[len-1]-1;

int work()

{

int res=0;

int k=-1;

for(int i=0;i<lenS;i++)

{

while(k!=-1&&P[k+1]!=str[i])

k=pi[k];

if(P[k+1]==str[i])

k++;

if(k==lenP-1)

{

res++;

k=pi[k];

}

}

return res;

}

AC自动机：

(动态)

struct Node

{

Node \*fail;

Node \*next[30];

int end;

Node()

{

fail=NULL;

for(int i=0; i<26; i++)

next[i]=NULL;

end=0;

}

void reset()

{

fail=NULL;

for(int i=0; i<26; i++)

if(next[i]!=NULL)

{

next[i]->reset();

delete next[i];

next[i]=NULL;

}

end=0;

}

};

Node Tree;

queue<Node\*> que;

void insert(const char \*s)

{

int len=strlen(s);

Node \*now=&Tree;

for(int i=0; i<len; i++)

{

if(now->next[s[i]-'a']==NULL)

now->next[s[i]-'a']=new Node();

now=now->next[s[i]-'a'];

}

now->end++;

}

void BuildAC()

{

while(!que.empty()) que.pop();

que.push(&Tree);

Node \*now;

Node \*p;

while(!que.empty())

{

now=que.front();

que.pop();

for(int i=0; i<26; i++)

{

if(now->next[i]!=NULL)

{

que.push(now->next[i]);

p=now->fail;

while(p!=NULL)

{

if(p->next[i]!=NULL)

{

//如果是不能包含，则加上

//if(tmp->next[i]->end)

//now->next[i]->end=1;

now->next[i]->fail=p->next[i];

break;

}

p=p->fail;

}

if(p==NULL) now->next[i]->fail=&Tree;//特别注意

}

}

}

}

int match()

{

int len=strlen(str);

int res=0;

Node \*now=&Tree;

Node \*tmp;

for(int i=0;i<len;i++)

{

while(now->next[str[i]-'a']==NULL&&now->fail!=NULL)

now=now->fail;

if(now->next[str[i]-'a']!=NULL)

{

now=now->next[str[i]-'a'];

tmp=now;

while(tmp->end!=-1&&tmp->fail!=NULL)

{

res+=tmp->end;

tmp->end=-1;

tmp=tmp->fail;

}

}

}

return res;

}

静态：

struct Node

{

int fail;

int next[4];

bool end;

void reset()

{

fail=-1;

for(int i=0; i<4; i++)

next[i]=-1;

end=0;

}

};

Node nds[maxn];

int code;

void insert(const char \*s)

{

int i=0;

int now=0;

while(s[i])

{

if(nds[now].next[mp(s[i])]==-1)

nds[now].next[mp(s[i])]=code++;

now=nds[now].next[mp(s[i])];

i++;

}

nds[now].end=1;

}

void BuildAC()

{

int now,tmp;

queue<int> que;

que.push(0);

while(!que.empty())

{

now=que.front();

que.pop();

for(int i=0; i<4; i++)

{

if(nds[now].next[i]!=-1)

{

que.push(nds[now].next[i]);

tmp=nds[now].fail;

while(tmp!=-1&&nds[tmp].next[i]==-1) //╬сх╩п╢Ёиакnds[now]

tmp=nds[tmp].fail;

if(tmp==-1)

nds[nds[now].next[i]].fail=0;

else if(nds[tmp].next[i]!=-1)

{

if(nds[nds[tmp].next[i]].end)

nds[nds[now].next[i]].end=1;

nds[nds[now].next[i]].fail=nds[tmp].next[i];

}

}

}

}

}

数据：（不能取字典），长度为10

4 10

ATAA

TAA

AA

A

59049

另一个数据（DNA repair）

2

ATGCAT

GC

ATGCAC

后缀数组：

后缀数组SA是一个一维数组，它保存1..n的某个排列SA[1]，SA[2]，……，SA[n]，并且保证 Suffix(SA[i])<Suffix(SA[i+1])，1≤i<n。也就是将S的n个后缀从小到大进行排序之后把排好序的后缀的开头位置顺次放入SA中。

字符串r的从第i个字符开始的后缀表示为Suffix(i)，也就是Suffix(i)=r[i..len(r)]

待排序的字符串放在r数组中，从r[0]到r[n-1]，长度为n，且最大值小于m

int wa[maxn],wb[maxn],wv[maxn],ws1[maxn];

int sa[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)

{

r[n++]=0;

int i,j,p,\*x=wa,\*y=wb,\*t;

for(i=0; i<m; i++) ws1[i]=0;

for(i=0; i<n; i++) ws1[x[i]=r[i]]++;

for(i=1; i<m; i++) ws1[i]+=ws1[i-1];

for(i=n-1; i>=0; i--) sa[--ws1[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++) ws1[i]=0;

for(i=0; i<n; i++) ws1[wv[i]]++;

for(i=1; i<m; i++) ws1[i]+=ws1[i-1];

for(i=n-1;i>=0;i--) sa[--ws1[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;

}

height数组：

height[i]=suffix(sa[i-1])和suffix(sa[i])的最长公共前缀

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;   
    }

最长公共前缀：

suffix(j)和suffix(k)的最长公共前缀为height[rank[j]+1]，height[rank[j]+2]，height[rank[j]+3]，……，height[rank[k]]中的最小值。用RMQ

可重叠最长重复子串

　　只需要求height数组里的最大值即可。首先求最长重复子串，等价于求两个后缀的最长公共前缀的最大值。因为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，那么这个值一定不大于height数组里的最大值。所以最长重复子串的长度就是height数组里的最大值。这个做法的时间复杂度为O(n)。

**最长公共子串:**

先将第二个字符串写在第一个字符串后面，中间用一个没有出现过的字符隔开，再求这个新的字符串的后缀数组, 只有当suffix(sa[i-1])和suffix(sa[i])不是同一个字符串中的两个后缀时，height[i]才是满足条件的。而这其中的最大值就是答案

不可重叠最长重复子串:

二分答案，

Check（k）：是否有不重叠的长度为k的子串（用sa数组判断）

数位统计：

其实就是分类讨论计数，核心是利用has,now,last三个数，要注意0不能开头就行,

Int p10[] //10^i

void getNum(int n,int res[])

{

int len=divi(n);

int now=0,last=0,has=n;

for(int i=1;i<=len;i++)

{

last=n%p10[i-1];

now=has%10;

has/=10;

for(int j=0;j<10;j++)

{

res[j]+=(has-1)\*p10[i-1];// j在中间的

if(j)

res[j]+=p10[i-1]; // j开头的

if(j<now) //多出一次j在中间的

res[j]+=p10[i-1];

else if(j==now)

res[j]+=last+1;

}

}

}