1. kmp
2. ac自动机
3. 后缀自动机
4. 后缀数组
5. 回文树
6. hash

Kmp算法：

int f[10005];

char a[10005],b[maxn];//a是模板串，b是目标串，这个会算出a在b中出现了多少次

void getfail()

{

f[1]=0;

int len=strlen(a+1);

for(int i=2;i<=len;i++)

{

int j=f[i-1];

while(j>0&&a[i]!=a[j+1])j=f[j];

f[i]=a[i]==a[j+1]?j+1:0;

}

}

void work()

{

getfail();

int len=strlen(b+1);

int n=strlen(a+1);

int j=0;

int cnt=0;

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

{

while(j>0&&a[j+1]!=b[i])j=f[j];

if(b[i]==a[j+1])j++;

if(j==n)

{

cnt++;

j=f[j];

}

}

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

}

Ac自动机：ac自动机处理的是多匹配串，然后看它在目标串中出现了几次。需要先建trie树，然后getfail()

void insert(int &now,int i,char \*s)//插入trie树

{

if(!now&&i!=0)now=++np;

if(!s[i])

{

val[now]++;//表示当前点是不是单词节点。

return;

}

int k=idx(s[i]);

insert(ch[now][k],i+1,s);

}

void getfail()//失配

{

queue<int>q;

f[0]=0;

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

{

int u=ch[0][i];

if(u){f[u]=0;q.push(u);last[u]=0;}

}

while(!q.empty())

{

int r=q.front();q.pop();

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

{

int u=ch[r][i];

if(!u){ch[r][i]=ch[f[r]][i];continue;}

q.push(u);

f[u]=ch[f[r]][i];

last[u]= val[f[u]] ? f[u]:last[f[u]];//表示该点最长的后缀中是单词节点的位置，可以加以改造使其表示以该位置结尾的单词的数量

}

}

}

void find(char \*t)

{

int m=strlen(t);

int j=0;

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

{

int c=idx(t[i]);

j=ch[j][c];

if(val[j])print(j);//print表示计算答案。

else if(last[j])print(last[j]);

}

}

后缀自动机

struct state

{

//par是par树中的边，表示right集,par可以形成一棵树

//go是真实的转移边

int par,go[26];

int val;//val是它的长度

}c[maxn];

struct SAM

{

long long vis[maxn],ww[maxn];int v[maxn],wh[maxn];

int rt,last,np;

SAM(){rt=last=++np;}

void extend(int w)//一步一步往里增加数

{

int p=last;//找到串尾,为结束位置的后缀

c[last=++np].val=c[p].val+1;//新建np,为新的结束位置的后缀

int now=np;

ww[np]=1;

while(p&&c[p].go[w]==0)//对于没有标号为x的边的祖先，加一条边

{

c[p].go[w]=np;

p=c[p].par;

}

if(p==0)//若祖先都没有去x的边，建一条根到now的边

c[now].par=rt;

else

{

int q=c[p].go[w];//找出该祖先去x的边

if(c[p].val+1==c[q].val)//若恰好长度吻合，直接接上去即可

c[now].par=q;

else

{

c[++np].val=c[p].val+1;

int nq=np;//新建一个节点令它的最长的长度为原来吻合长度

memcpy(c[nq].go,c[q].go,sizeof(c[q].go));//拷贝

c[nq].par=c[q].par;//取代q

c[q].par=nq;//令q的fa为nq

c[now].par=nq;//分成两部分

while(p&&c[p].go[w]==q)//改变所有祖先

c[p].go[w]=nq,p=c[p].par;

}

last=now;

}

}

void pre()

{

//这三步是为了把每个点按长度排序，长度相同按先后次序排

for(int i=1;i<=np;i++)v[c[i].val]++;

for(int i=1;i<=n;i++)v[i]+=v[i-1];

for(int i=np;i>=1;i--)wh[v[c[i].val]--]=i;

//ww是它的right集合的大小

ww[1]=0;

for(int i=np;i>=2;i--)

{

int t=wh[i];

if(T==1)ww[c[t].par]+=ww[t];//这是当位置不同算不同字符串时

else ww[t]=1;//这是需要本质不同才算不同字符串时

}

ww[1]=0;

//vis是当前节点的路径数以及它能正向走到的点的路径数和。

for(int i=np;i>=1;i--)

{

int t=wh[i];vis[t]=ww[t];

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

if(c[t].go[ii])vis[t]+=vis[c[t].go[ii]];

}

}

void dfs(int now,int k)//递归求答案。

{

if(k<=ww[now])return;

k-=ww[now];

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

{

if(c[now].go[i])

{

if(k<=vis[c[now].go[i]])

{

putchar(i+'a');

dfs(c[now].go[i],k);

return;

}

k-=vis[c[now].go[i]];

}

}

}

}sam;

后缀数组

构造：

int sa[maxn],rank[maxn],height[maxn];

把s的每个后缀按照字典序排序，

后缀数组sa[i]就表示排名为i的后缀的起始位置的下标

而它的映射数组rank[i]就表示起始位置的下标为i的后缀的排名

简单来说，sa表示排名为i的是啥，rank表示第i个的排名是啥

以s的第i个字符为第一个元素的后缀为suff(i)

LCP(i,j)为suff(sa[i])与suff(sa[j])的最长公共前缀

height[i]为LCP(i,i-1)，1<i<=n

关键运用height,sa,rank数组，其中height数组是关键。

void get\_height()

{

int k=0;

for(int i=1;i<=n;i++)rank[sa[i]]=i;

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

{

if(rank[i]==1){k=0;height[1]=0;continue;}

if(k)k--;

int j=sa[rank[i]-1];

while(j+k<=n&&i+k<=n&&s[i+k]==s[j+k])k++;

height[rank[i]]=k;

}

}

void prefix\_doubling(int m)

{

int i,\*x=t1,\*y=t2;

for(i=1;i<=m;i++)sum[i]=0;

for(i=1;i<=n;i++)sum[x[i]=s[i]]++;

for(i=2;i<=m;i++)sum[i]+=sum[i-1];

for(i=n;i>=1;i--)sa[sum[x[i]]--]=i;

for(int k=1;k<=n;k<<=1)

{

int p=0;

for(i=n-k+1;i<=n;i++)y[++p]=i;

for(i=1;i<=n;i++)

if(sa[i]>k)y[++p]=sa[i]-k;

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

for(i=1;i<=n;i++)sum[x[i]]++;

for(i=2;i<=m;i++)sum[i]+=sum[i-1];

for(i=n;i>=1;i--)sa[sum[x[y[i]]]--]=y[i];

swap(x,y);p=1;x[sa[1]]=1;

for(i=2;i<=n;i++)

{

/\*x[sa[i]]=y[sa[i-1]]==y[sa[i]] && y[sa[i-1]+k]==y[sa[i]+k]?p:++p;\*/

if(y[sa[i]]!=y[sa[i-1]]||(sa[i]+k>n?0:y[sa[i]+k])!=(sa[i-1]+k>n?0:y[sa[i-1]+k]))

++p;

x[sa[i]]=p;

}

if(p>=n)break;

m=p;

}

}

prefix\_doubling(130);//130表示最大字符集

get\_height();

后缀数组经典5题

//最长相同可重叠子串

int ans=0;

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

ans=max(ans,height[i]);

//计算不同的子串个数

long long aans=(n+1ll)\*n/2;

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

aans-=height[i];

//最长相同不重叠子串

height[n+1]=-1;

int l=0,r=n,mid=0;ans=0;

while(r>=l)//二分答案，然后连续计算

{

mid=(l+r)/2;

int isok=0,maxv=sa[1],minv=sa[1];

for(int i=2;i<=n;i++)

{

if(height[i]>=mid)//当一直相同时就不断更新maxv,minv。

{

maxv=max(sa[i],maxv);

minv=min(sa[i],minv);

if(maxv-minv>=mid)isok=1;

if(isok==1)break;

}

else minv=sa[i],maxv=sa[i];//不相同就重置min和max

}

if(isok)

{

ans=mid;

l=mid+1;

}

else r=mid-1;

}

//至少出现了p次的最长重复子串（子串可重叠）

l=0,r=n,mid=0;ans=0;

while(r>=l)//二分算答案

{

mid=(l+r)/2;

int tot=1,isok=0;

for(int i=2;i<=n;i++)

{

if(height[i]>=mid)//算tot

{

tot++;

if(tot>=q)isok=1;

if(isok)break;

}

else tot=1;

}

if(isok)

{

ans=mid;

l=mid+1;

}

else r=mid-1;

}

回文树：

回文树可以解决的问题：回文串的个数（本质不同的和位置不同的都可以），每个回文串的长度和等价的问题

每个回文串出现的次数，倘若要动态询问就需要lct来维护fail树，为了区间修改。

struct mytree

{

//每个点都至少代表一个回文。

int to[maxn][26],fail[maxn],val[maxn],len[maxn],s[maxn],np,n,last;

int newnode(int r)

{

memset(to[np],0,sizeof(to[np]));

//这里fail的含义是以该节点结束的回文（即fail下去的首先是它的后缀，然后是它的回文）

fail[np]=0;

val[np]=0;//val表示的是出现的次数

len[np]=r;//len可以替换为其他可以求的。

return np++;

}

mytree()

{

//首先初始化，建立两个起点，一个奇数回文起点，一个偶数回文起点

//其中为了方便奇数回文len为-1,偶数回文len为1，将s[0]设为-1，fail[0]设为1

np=0; n=0; last=0;

memset(s,0,sizeof(s));

newnode(0); newnode(-1);

s[0]=-1; fail[0]=1;

}

int getfail(int x)

{

//倘若构不成回文就持续fail

while(s[n-len[x]-1]!=s[n])x=fail[x];

return x;

}

void insert(int x)

{

s[++n]=x;

//找到当前的可以匹配的点，若一直找不到就会到奇数回文起点导致s[n]=s[n]即可。

int cur=getfail(last);

if(!to[cur][x])

{

//若没有走到该边的

int p=newnode(len[cur]+2);

//找到它失配后到达的位置

fail[p]=to[getfail(fail[cur])][x];

to[cur][x]=p;//连上。

}

last=to[cur][x];

val[last]++;//更新val值，当当前回文出现时意味着它的fail回文均出现过，每次更新的是它的最长回文

}

long long dp()

{

long long ans=0;

for(int i=np-1;i>=0;i--)val[fail[i]]+=val[i];//累加上去

for(int i=0;i<np;i++)ans=max(ans,(long long)val[i]\*len[i]);

return ans;

}

}c;