# 后缀自动机



## 全家桶

```
#include<bits/stdc++.h>
typedef long long ll;
using namespace std;
const int N=1e6+7;

char s[N];
int n,t,k,vis[N];
ll dp[N];
string anss="";

namespace SAM{
    const int MAXLEN=2e6+2;
    int SZ=1,lst=1;  //当前自动机大小和整个字符串最后状态

    struct edge{int v,nxt;}e[MAXLEN<<1];
    int head[MAXLEN],cnt=0;
    void add(int u,int v){e[++cnt]=(edge){v,head[u]};head[u]=cnt;}

    struct state{   //SAM上的信息结点
        int len,fa,sz;   //长度和后缀链接
        //map<int,int>ch;
        int ch[26];  //转移列表
```

```
}st[MAXLEN];  //最多2*n-1个状态

void clr(int size){
    SZ=lst=1;cnt=0;
    for(int i=0;i<=size;++i){
        head[i]=0;
        st[i].len=st[i].fa=st[i].sz=0;
        memset(st[i].ch,0,sizeof(st[i].ch));
    }
}

void ins(int c){ //添加字符
    int p=lst,np=lst=++SZ;   //创建新的状态结点
    st[np].sz=1;
    st[np].len=st[p].len+1;
    for(;p&&!st[p].ch[c];p=st[p].fa) st[p].ch[c]=np;
    if(!p) st[np].fa=1;
    else{
        int q=st[p].ch[c];
        if(st[q].len==st[p].len+1) st[np].fa=q;
        else{
            int nq=++SZ;
            st[nq]=st[q];st[nq].len=st[p].len+1;
            st[q].fa=st[np].fa=nq;
            st[nq].sz=0;  //这里卡了半天!!!
            for(;p&&st[p].ch[c]==q;p=st[p].fa) st[p].ch[c]=nq;
        }
    }
    //ans+=st[np].len-st[st[np].fa].len;
    //对不同子串的贡献为len(np)-len(fa(np))(生成魔咒)
}

/*    下面是一些模板题
// 【模板】后缀自动机 (SAM)
void get_ans(char *s,int len){   //max(出现次数不为1的子串出现次数*子串长度)
    ll ans=0;
    for(int i=0;i<len;++i) ins(s[i]-'a');
    for(int i=1;i<=SZ;++i) add(st[i].fa,i);  //连parent树
    std::function<int(int)>dfs;
    dfs=[&](int x){  //树形dp求每个子串的答案
        for(int i=head[x];i;i=e[i].nxt) st[x].sz+=dfs(e[i].v);
        if(st[x].sz>1) ans=max(ans,(ll)st[x].len*st[x].sz);
        return st[x].sz;
    };
    dfs(1);
    printf("%lld\n",ans);
}

bool find(char *s){  //查询字符串是否出现
    int len=strlen(s),p=1;
    for(int i=0;i<len;++i){
        if(st[p].ch[s[i]-'a']) p=st[p].ch[s[i]-'a'];
        else return false;
    }
    return true;
}
// 工艺
int a[N];
```

```
void minepx(int n){ //最小循环位移
    //转化为在S+S串对应的后缀自动机上寻找最少长度为len的路径
    for(int i=1;i<=n;++i) scanf("%d",&a[i]),a[n+i]=a[i];
    for(int i=1;i<=2*n;++i) ins(a[i]);
    std::function<void(int,int)>dfs;
    dfs=[&](int x,int dep){
        int mi=(st[x].ch.begin()->first);
        printf("%d ",mi);
        if(dep==n) return;
        dfs(st[x].ch[mi],dep+1);
    };
    dfs(1,1);
}
//LCS - Longest Common Substring
char s[N],t[N];
void lcs(){  //求两个字符串的最长公共子串长度
    scanf("%s%s",s,t);
    int slen=strlen(s),tlen=strlen(t),ans=0;
    for(int i=0;i<slen;++i) ins(s[i]-'a'); //对一个串建SAM
    int p=1,len=0;
    for(int i=0;i<tlen;++i){  //将第二串放上去跑
        int c=t[i]-'a';
        if(st[p].ch[c]) ++len,p=st[p].ch[c];
        else{
            for(;p&&!st[p].ch[c];p=st[p].fa);
            if(p) len=st[p].len+1,p=st[p].ch[c];
            else len=0,p=1;
        }
        ans=max(ans,len);
    }
    printf("%d\n",ans);
}
*/
//[TJOI2015]弦论
int dfs1(int x){    //统计每个结点表示的子串数
    for(int i=head[x];i;i=e[i].nxt) st[x].sz+=dfs1(e[i].v);
    return st[x].sz;
}
void dfs2(int x){ //dp[i]:从i出发的串个数
    if(vis[x]) return;
    vis[x]=1;
    dp[x]=st[x].sz;
    for(int i=0;i<26;++i){
        if(!st[x].ch[i]) continue;
        int to=st[x].ch[i];
        dfs2(to);
        dp[x]+=dp[to];
    }
}
void dfs3(int x,int k){  //寻找第k大路径(子串)
    if(vis[x]) return;
    vis[x]=1;
    if(k<=0) return;
    for(int i=0;i<26;++i){
        if(!st[x].ch[i]) continue;
        int to=st[x].ch[i];
        if(k>dp[to]) k-=dp[to];
        else{
```

```
                k-=st[to].sz;
                anss+=(char)(i+'a');
                dfs3(to,k);
                break;
            }
        }
    }
};

using namespace SAM;

signed main(){
    scanf("%s",s);
    int n=strlen(s);
    for(int i=0;i<n;++i) ins(s[i]-'a');
    for(int i=1;i<=SZ;++i) add(st[i].fa,i);
    scanf("%d%d",&t,&k);
    if(t==0) for(int i=1;i<=SZ;++i) st[i].sz=1;
    else dfs1(1),st[1].sz=0;
    dfs2(1);
    if(dp[1]<k) cout<<"-1\n";
    else{
        memset(vis,0,sizeof(vis));
        dfs3(1,k);
        cout<<anss<<"\n";
    }
    return 0;
}
```

## 构造

```
int sz,last; //SAM的大小以及末状态
struct state{ //状态节点
    int len,link;
    map<char,int>nxt; //转移的列表
}st[MAXLEN*2]; //因为最多2n-1个状态

void sam_init(){ //初始化sam
    st[0].len=0;
    st[0].link=-1; //-1表示虚拟状态
    sz++;
    last=0;
}

void sam_extend(char c){ //添加字符c
    int cur=sz++; //创建新的状态
    st[cur].len=st[last].len+1;
    int p=last;
    while(p!=-1&&st[p].nxt.count(c)){ //遍历后缀链接
        st[p].nxt[c]=cur;
        p=st[p].link;
    }
    if(p==-1) st[cur].link=0; //没要找到这样的状态
    else{
        int q=st[p].nxt[c];
```

```
        if(st[p].len+1==st[q].len) st[cur].link=q;
        else{
            int clone=sz++; //复制一个状态
            st[clone].len=st[p].len+1;
            st[clone].nxt=st[q].nxt;
            st[clone].link=st[q].link;
            while(p!=-1&&st[p].nxt[c]==q){
                //后缀链接从状态p往回走
                st[p].nxt[c]=clone;
                p=st[p].link;
            }
            st[q].link=st[cur].link=clone;
        }
    }
    last=cur; //更新last为当前状态
}
```

## 检查子串是否出现

在SAM上向父亲跑边，跑完串还未跑到空状态则为原串子串。

后缀数组：跑出sa，然后从最小的后缀开始，一个个往后枚举，记录下当前匹配到的位置，如果匹配不上就下一个后缀，否则位置向后移一位。如果枚举完了后缀还没完全匹配则不是原串子串。

## 最小循环位移

字符串S+S对应的SAM中寻找最小的长度为|S|的路径。

从初始状态开始，贪心地访问最小的字符即可。

## 子串第一次出现位置

预处理每个状态第一次出现的位置$pos(i)$

只需要每次让$pos(np)=len(np)pos(nq)=pos(q)$就好了

查询答案为$pos(i)-|T|+1$

## 最短的没有出现的字符串

在SAM上做dp，设$dp[i]$表示到点i时的最短长度

如果这个点有不是T中字符的出边，则$dp[i]=1$，否则$dp[i]=1+min_{(i,j,c)\in SAM}dp[j]$

## 不同子串个数

对字符串S构造后缀自动机。每个S的子串相当于自动机中的一些路径，因此不同子串的个数等于自动机中以$t_0$为起点的不同路径条数。

考虑SAM为DAG，可用动态规划计算不同路径的条数。

令$d_v$为从状态$v$开始的路径数量, 则$d_v = 1 + \sum_{w:(v,w,c)\in DAWG} d_w$

即$d_v$可以表示为所有$v$的转移的末端之和。

如果是动态的，每次新增字符时，对答案的贡献为
$|max(p)| - |min(p)| + 1 = |max(p)| - |max(p->link)|$,新建的nq节点是没有贡献的。

## 所有不同子串的总长度

与上述做法类似，递推式$ans_v = \sum_{w:(v,w,c)\in DAWG}(d_w + ans_w)$

**luoguP3804 【模板】后缀自动机 (SAM)**

所有出现次数不为1的子串的出现次数乘以该子串长度的最大值。

## 字典序第k大子串

字典序第k大的子串对应于SAM中字典序第k大的路径，因此计算每个状态的路径数后，可以很容易的从SAM的根开始找到第k大的路径。预处理$O(|S|)$, 单词询问$O(|ans| \cdot |\Sigma|)$

## 子串出现次数

对于每个状态$v$,预处理$cnt_v$，使之等于$endpos(v)$集合的大小。

## 两个串的最长公共子串

对母串建SAM，然后第二个字符串一个个匹配过去。

如果现态p有Si的转移，则len++并跳到下一个字符；否则在parent树上向上跳直到有这个转移为止。

## 多个字符串间的最长公共子串

还是对第一个串建SAM，然后一个一个串处理。在处理每一个串的时候记录当前节点的最大匹配长度，并且记录最大长度的最小值，就是所有串的匹配长度。

**SP1812 LCS2 - Longest Common Substring II**

```
#include<bits/stdc++.h>
#define N 2000010
#define inf 0x3f3f3f3f
using namespace std;
int n,lst,tot=1,root=1,num,ans;
```

```
int fa[N],len[N],ch[N][30],siz[N][12];
char s[N];
vector<int>G[N];

void insert(int c,int id){  //SAM构造
    int p=lst,np=lst=++tot;
    len[np]=len[p]+1,siz[np][id]++; //记录每个节点被多少字符串更新
    while(p&&!ch[p][c]) ch[p][c]=np,p=fa[p];
    if(!p) fa[np]=root;
    else{
        int q=ch[p][c];
        if(len[q]==len[p]+1) fa[np]=q;
        else{
            int nq=++tot;
            memcpy(ch[nq],ch[q],sizeof(ch[q]));
            len[nq]=len[p]+1,fa[nq]=fa[q],fa[q]=fa[np]=nq;
            while(ch[p][c]==q) ch[p][c]=nq,p=fa[p];
        }
    }
}
void dfs(int x){
    for(int i=0;i<G[x].size();i++){
        int y=G[x][i];
        dfs(y);
        for(int id=1;id<=num;++id)
            siz[x][id]+=siz[y][id]; //跑DFS处理子串信息
    }
}

bool check(int p){
    if(!p) return false;
    for(int i=1;i<=num;++i) //如果所有字符串都有子串p
        if(!siz[p][i])
            return false;
    return true;
}

void work(){
    for(int i=2;i<=tot;++i)  G[fa[i]].push_back(i);
    dfs(root);
    int p=root,l=0;
    for(int i=1;i<=n;++i){
        int c=s[i]-'a';
        if(check(ch[p][c])) l++,p=ch[p][c];
        else{
            while(p&&!check(ch[p][c])) p=fa[p];
            if(p) l=len[p]+1,p=ch[p][c];
            else l=0,p=root;
        }
        ans=max(ans,l);
    }
}
signed main(){
    while(scanf("%s",s+1)!=EOF){
        n=strlen(s+1);
        lst=root;
        num++;
        for(int i=1;i<=n;++i) insert(s[i]-'a',num); //广义SAM
```

```
    }
    work();
    printf("%d\n",ans);
    return 0;
}
```

# 广义SAM

广义sam是针对一棵trie树来说的，每次对一个串建SAM后将lst初始化即可。

**SP8093 JZPGYZ - Sevenk Love Oimaster**

给定 n个模板串，以及 m个查询串，依次查询每一个查询串是多少个模板串的子串。

```
#include<bits/stdc++.h>
typedef long long ll;
using namespace std;
const int N=1e6+7;

char s[N];
int n,m,len[N],las[N],tot;

namespace SAM{
    const int MAXLEN=2e6+2;
    int SZ=1,lst=1;  //当前自动机大小和整个字符串最后状态
    struct edge{int v,nxt;}e[MAXLEN<<1];
    int head[MAXLEN],cnt=0;
    void add(int u,int v){e[++cnt]=(edge){v,head[u]};head[u]=cnt;}

    struct state{   //SAM上的信息结点
        int len,fa,sz;   //长度和后缀链接
        //map<int,int>ch;
        int ch[26];  //转移列表
    }st[MAXLEN];  //最多2*n-1个状态

    void ins(int c){ //添加字符
        int p=lst,np=lst=++SZ;   //创建新的状态结点
        st[np].len=st[p].len+1;
        for(;p&&!st[p].ch[c];p=st[p].fa) st[p].ch[c]=np;
        if(!p) st[np].fa=1;
        else{
            int q=st[p].ch[c];
            if(st[q].len==st[p].len+1) st[np].fa=q;
            else{
                int nq=++SZ;
                st[nq]=st[q];st[nq].len=st[p].len+1;
                st[q].fa=st[np].fa=nq;
                for(;p&&st[p].ch[c]==q;p=st[p].fa) st[p].ch[c]=nq;
            }
        }
    }
};

using namespace SAM;
```

```
inline void update(int x,int y){
    for(;x&&las[x]!=y;x=st[x].fa) ++st[x].sz,las[x]=y;
}

signed main(){
    scanf("%d%d",&n,&m);getchar();
    for(int i=1;i<=n;++i){
        lst=1;
        char ch;
        while((ch=getchar())!='\n') ins(s[++tot]=ch-'a'),++len[i];
    }
    tot=0;
    for(int i=1;i<=n;++i){
        for(int j=1,x=1;j<=len[i];++j) update(x=st[x].ch[s[++tot]],i);
    }
    for(int i=1,x,flag;i<=m;++i){
        tot=0;flag=1;x=1;
        char ch;
        while((ch=getchar())!='\n'&&ch!=EOF){
            int c=ch-'a';
            if(st[x].ch[c]) x=st[x].ch[c];
            else flag=0;
        }
        printf("%d\n",flag?st[x].sz:0);
    }
    return 0;
}
```

## 其他例题

**P5341 [TJOI2019]甲苯先生和大中锋的字符串**

计算出现k次的子串中出现次数的最多的长度。

设树上一个结点的出现次数是$sz_i$，那么扫描所有$sz_i == k$的点，该节点表示的所有子串都是可行的，并且它们长度是连续的一段区间$(len_{fa_i}, len_i]$，因此在parent树上做一个差分即可。

```
#include<bits/stdc++.h>
typedef long long ll;
using namespace std;
const int N=1e6+7;

namespace SAM{
    const int MAXLEN=2e6+2;
    int SZ=1,lst=1;  //当前自动机大小和整个字符串最后状态

    struct edge{int v,nxt;}e[MAXLEN<<1];
    int head[MAXLEN],cnt=0;
    void add(int u,int v){e[++cnt]=(edge){v,head[u]};head[u]=cnt;}

    struct state{   //SAM上的信息结点
        int len,fa,sz;   //长度和后缀链接
        //map<int,int>ch;
        int ch[26];  //转移列表
```

```
    }st[MAXLEN];  //最多2*n-1个状态

    void clr(int size){
        SZ=lst=1;cnt=0;
        for(int i=0;i<=size;++i){
            head[i]=0;
            st[i].len=st[i].fa=st[i].sz=0;
            memset(st[i].ch,0,sizeof(st[i].ch));
        }
    }

    void ins(int c){ //添加字符
        int p=lst,np=lst=++SZ;   //创建新的状态结点
        st[np].sz=1;
        st[np].len=st[p].len+1;
        for(;p&&!st[p].ch[c];p=st[p].fa) st[p].ch[c]=np;
        if(!p) st[np].fa=1;
        else{
            int q=st[p].ch[c];
            if(st[q].len==st[p].len+1) st[np].fa=q;
            else{
                int nq=++SZ;
                st[nq]=st[q];st[nq].len=st[p].len+1;
                st[q].fa=st[np].fa=nq;
                st[nq].sz=0;
                for(;p&&st[p].ch[c]==q;p=st[p].fa) st[p].ch[c]=nq;
            }
        }
    }
};

using namespace SAM;

char s[N];
int n,t,k,mxl,cntt[N];
int dfs(int x,int f,int k){
    for(int i=head[x];i;i=e[i].nxt) st[x].sz+=dfs(e[i].v,x,k);
    if(x!=1&&st[x].sz==k) ++cntt[st[x].len],--cntt[st[f].len];
    mxl=max(mxl,st[x].len);
    return st[x].sz;
}

signed main(){
    scanf("%d",&t);
    while(t--){
        scanf("%s",s);
        scanf("%d",&k);
        n=strlen(s);
        for(int i=0;i<n;++i) ins(s[i]-'a');
        for(int i=1;i<=SZ;++i) add(st[i].fa,i);
        dfs(1,0,k);
        int ans=-1,p=0,ans2=0;
        for(int i=mxl;i>=1;--i){
            p+=cntt[i];cntt[i]=0;
            if(p>ans2) ans=i,ans2=p;
        }
        printf("%d\n",ans);
        clr(SZ);
```

```
    }
    return 0;
}
```

## 参考资料


https://www.luogu.com.cn/blog/Kesdiael3/hou-zhui-zi-dong-ji-yang-xie

https://www.bilibili.com/video/BV1wa4y1a7Ag

https://www.bilibili.com/video/BV1ED4y1U7aU

https://www.cnblogs.com/maomao9173/p/10447821.html

https://www.cnblogs.com/mikufun-hzoi-cpp/p/12098738.html

https://www.luogu.com.cn/blog/lhm126/solution-sp1812