字符串 4

后缀自动机(SAM)

简介

一种用于解决各种字符串问题的数据结构，时间复杂度 $O(n)$。

性质

性质一：SAM 是一张有向无环图，**t 为 s 子串当且仅当存在唯一从原点出发的路径使得该路径等价于 t**。

根据该性质，可以 $O(|T|)$ 判断 T 是否为 S 的子串。

性质二：对于 s 的非空子串 t，定义 endpos 表示字符串 s 中所有 t 的结束位置。

将所有 endpos 相同的子串归入同一个等价类，共用 SAM 上的一个结点。**对于两个不同等价类，要么不相交，要么属于包含关系。**

因为如果相交说明某个等价类的某个字符串一定是另一个等价类中某个字符串的后缀。

根据该性质可以构造一棵 parent 树，且树上父结点的子结点 endpos 集合不相交，且均属于父结点。

性质三：对一个等价类，取最长的字符串记为 L，最短的字符串记为 R，**则该等价类的字符串均为 L 的后缀且长度依次为 $|R|\sim |L|$**。

对于上述性质证明，可以考虑依次从 L 前端删去一个字母，显然新得到的字符串 endpos 要么与 L 相同，要么包含 $\text{endpos}(L)$。

于是可以得到第一个是 L 的后缀且不属于 $\text{endpos}(L)$ 的字符串 L'，**有 $|R|=|L'|+1$**。

根据该性质，易知每次在 S 末尾添加字符得到的新的本质不同的子串为 $|L|-|L'|$。

性质四： **parent 树中父节点一定是子节点的后缀，所有子节点的 endpos 的并集 $=$ 父节点的 endpos 删去父节点中是 S 前缀的位置。**

因为由于子节点的 $|R|=|L'|+1$，而 S 的前缀 i 属于父节点，而前缀 i 前端不可能添加字符，必然有前缀 i 是父节点中最长串 L'。

于是 $|R|=i+1$，即子节点中最短串长度已经超过 i，所以 i 不属于 endpos。

另一方面，$\text{endpos}\gt i$ 的位置的所有串必然可以表示成 $c+L'$ 的形式，于是不会丢失。

根据该性质，可以 $O(n)$ 计算出每个类 endpos 的大小，即在 S 中的出现次数。

模板

```
struct SAM{//记得开两倍内存
    int ch[MAXN][26],fa[MAXN],len[MAXN],last,cnt;
    void Insert(int u,int v){
        edge[++edge_cnt]=Edge{v,head[u]};
        head[u]=edge_cnt;
    }
    void extend(int c){
        int p=last,np=++cnt;
        last=np,len[np]=len[p]+1;
        for(;p&&!ch[p][c];p=fa[p])ch[p][c]=np;
        if(!p)fa[np]=1;
        else{
            int q=ch[p][c];
            if(len[q]==len[p]+1)fa[np]=q;
            else{
                int nq=++cnt;len[nq]=len[p]+1;
                memcpy(ch[nq],ch[q],sizeof(ch[q]));fa[nq]=fa[q];
                fa[q]=fa[np]=nq;
                for(;ch[p][c]==q;p=fa[p])ch[p][c]=nq;
            }
        }
    }
    void init(char *s,int n){
        last=cnt=1;
        mem(ch[1],0);
        _for(i,0,n)
        extend(s[i]-'a');
    }
};
```

算法练习

习题一

洛谷p3804

题意

给定字符串 S，求 S 中出现字数不为 1 的子串乘以子串长度的最大值。

SAM 题解

根据 SAM 的性质四，不难得到过程，时间复杂度 $O(n)$。

```
const int MAXN=2e6+5;
LL ans;
```

```
struct SAM{
    struct Edge{
        int to,next;
    }edge[MAXN];
    int ch[MAXN][26],fa[MAXN],len[MAXN],last,cnt;
    int head[MAXN],sz[MAXN],edge_cnt;
    void Insert(int u,int v){
        edge[++edge_cnt]=Edge{v,head[u]};
        head[u]=edge_cnt;
    }
    void extend(int c){
        int p=last,np=++cnt;
        last=np,len[np]=len[p]+1,sz[np]=1;
        for(;p&&!ch[p][c];p=fa[p])ch[p][c]=np;
        if(!p)fa[np]=1;
        else{
            int q=ch[p][c];
            if(len[q]==len[p]+1)fa[np]=q;
            else{
                int nq=++cnt;len[nq]=len[p]+1;
                memcpy(ch[nq],ch[q],sizeof(ch[q]));fa[nq]=fa[q];
                fa[q]=fa[np]=nq;
                for(;ch[p][c]==q;p=fa[p])ch[p][c]=nq;
            }
        }
    }
    void init(char *s,int n){
        last=cnt=1;
        mem(ch[1],0);
        _for(i,0,n)
        extend(s[i]-'a');
        _rep(i,2,cnt)Insert(fa[i],i);
    }
    void dfs(int u){
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            dfs(v);
            sz[u]+=sz[v];
        }
        if(sz[u]>1)
        ans=max(ans,1LL*sz[u]*len[u]);
    }
}solver;
char buf[MAXN];
int main()
{
    scanf("%s",buf);
    solver.init(buf,strlen(buf));
    solver.dfs(1);
    enter(ans);
    return 0;
```

```
}
```

SA 题解

考虑 SA 维护得到 height 数组，从大到小添加 height 数组代表的连边同时并查集维护集合大小即可，时间复杂度 $O(n\log n)$。

习题二

洛谷p4070

题意

给定一个空串 S，每次向 S 末尾添加一个字符串，求每次操作后 S 中本质不同的子串个数。

SAM 题解

根据 SAM 的性质三，不难得到过程，但注意需要字符集较大时需要 map 存图，时间复杂度 $O(n\log n)$。

```
const int MAXN=2e5+5;
LL ans;
struct SAM{
    map<int,int> ch[MAXN];
    int fa[MAXN],len[MAXN],last,cnt;
    void extend(int c){
        int p=last,np=++cnt;
        last=np,len[np]=len[p]+1;
        for(;p&&!ch[p].count(c);p=fa[p])ch[p][c]=np;
        if(!p)fa[np]=1;
        else{
            int q=ch[p][c];
            if(len[q]==len[p]+1)fa[np]=q;
            else{
                int nq=++cnt;len[nq]=len[p]+1;
                ch[nq]=ch[q],fa[nq]=fa[q],fa[q]=fa[np]=nq;
                for(;ch[p].count(c)&&ch[p][c]==q;p=fa[p])ch[p][c]=nq;
            }
        }
        ans+=len[np]-len[fa[np]];
    }
    void init(){
        last=cnt=1;
        ch[1].clear();
    }
}solver;
int buf[MAXN];
int main()
```

```
{
    int n=read_int();
    _for(i,0,n)buf[i]=read_int();
    solver.init();
    _for(i,0,n){
        solver.extend(buf[i]);
        enter(ans);
    }
    return 0;
}
```

SA 题解

如果向后加入字符将影响所所有后缀以及对应的 height 数组。于是考虑将字符串翻转，本质不同的串个数并不会改变。

但是向后加入字符操作转化为向前加入字符操作，而向前加入新字符操作只是增加一个后缀，并不会影响之前的 height 数组。

于是可以利用平衡树和 ST 表， $O(\log n)$ 时间复杂度计算每次新加入字符产生的本质不同的字符串个数。总时间复杂 $O(n\log n)$

```
const int MAXN=1e5+5,MAXM=20;
namespace SA{
    int sa[MAXN],rk[MAXN],height[MAXN],X[MAXN],Y[MAXN],c[MAXN];
    int d[MAXN][MAXM],lg2[MAXN];
    void get_sa(int *s,int n,int m){
        int *x=X,*y=Y;
        _rep(i,0,m)c[i]=0;
        _rep(i,1,n)c[x[i]=s[i]]++;
        _rep(i,1,m)c[i]+=c[i-1];
        for(int i=n;i;i--)sa[c[x[i]]--]=i;
        for(int k=1;k<n;k<<=1){
            int pos=0;
            _rep(i,n-k+1,n)y[++pos]=i;
            _rep(i,1,n)if(sa[i]>k)y[++pos]=sa[i]-k;
            _rep(i,0,m)c[i]=0;
            _rep(i,1,n)c[x[i]]++;
            _rep(i,1,m)c[i]+=c[i-1];
            for(int i=n;i;i--)sa[c[x[y[i]]]--]=y[i],y[i]=0;
            swap(x,y);
            pos=0,y[n+1]=0;
_rep(i,1,n)x[sa[i]]=(y[sa[i]]==y[sa[i-1]]&&y[sa[i]+k]==y[sa[i-1]+k])?pos:++
pos;
            if(pos==n)break;
            m=pos;
        }
        _rep(i,1,n)rk[sa[i]]=i;
    }
    void get_height(int *s,int n){
```

```
        for(int i=1,k=0;i<=n;i++){
            if(k)k--;
            while(s[i+k]==s[sa[rk[i]-1]+k])k++;
            height[rk[i]]=k;
        }
    }
    void build_st(int n){
        lg2[1]=0;
        _rep(i,2,n)
        lg2[i]=lg2[i>>1]+1;
        _rep(i,1,n)
        d[i][0]=height[i];
        for(int j=1;(1<<j)<=n;j++){
            _rep(i,1,n+1-(1<<j))
            d[i][j]=min(d[i][j-1],d[i+(1<<(j-1))][j-1]);
        }
    }
    int lcp(int lef,int rig){
        int len=rig-(++lef)+1;
        return min(d[lef][lg2[len]],d[rig-(1<<lg2[len])+1][lg2[len]]);
    }
}
int buf[MAXN],a[MAXN];
set<int> s;
int main()
{
    int n=read_int();
    _rep(i,1,n)buf[i]=a[i]=read_int();
    reverse(buf+1,buf+n+1);
    sort(a+1,a+n+1);
    int m=unique(a+1,a+n+1)-a;
    _rep(i,1,n)buf[i]=lower_bound(a+1,a+m,buf[i])-a;
    SA::get_sa(buf,n,m);
    SA::get_height(buf,n);
    SA::build_st(n);
    LL ans=0;
    for(int i=n;i;i--){
        s.insert(SA::rk[i]);
        set<int>::iterator it=s.find(SA::rk[i]);
        int lcp=0;
        if(it!=s.begin()){
            lcp=max(lcp,SA::lcp(*(--it),SA::rk[i]));
            it++;
        }
        if((++it)!=s.end())
        lcp=max(lcp,SA::lcp(SA::rk[i],*it));
        ans+=n+1-i-lcp;
        enter(ans);
    }
    return 0;
```

```
}
```

习题三

SP1811

题意

给定两个串，求两个串的最长公共子串(LCS)。

SAM 题解

考虑将一个串压入 SAM 另一个串在 SAM 上匹配。

如果存在 $ch[pos][c]$ 则直接匹配，且匹配长度 $+1$。否则考虑尽可能地保留后缀，于是不断跳 parent 树。

由于当前串的后缀与当前等价类地某个串匹配，而该等价类的最短串长度大于其父节点的等价类中的最长串，于是当前串一定能与父节点最长串匹配。

于是跳 parent 树过程中如果遇到 $ch[pos][c]$ 存在的情况，匹配长度变为 len_pos+1。

每次失配向上跳使得匹配长度变短，而匹配长度最多增加 $|T|$ 次，于是失配的均摊复杂度为 $O(1)$ 总时间复杂度 $O(n)$。

```
const int MAXN=5e5+5;
struct SAM{
    int ch[MAXN][26],fa[MAXN],len[MAXN],last,cnt;
    void extend(int c){
        int p=last,np=++cnt;
        last=np,len[np]=len[p]+1;
        for(;p&&!ch[p][c];p=fa[p])ch[p][c]=np;
        if(!p)fa[np]=1;
        else{
            int q=ch[p][c];
            if(len[q]==len[p]+1)fa[np]=q;
            else{
                int nq=++cnt;len[nq]=len[p]+1;
                memcpy(ch[nq],ch[q],sizeof(ch[q]));fa[nq]=fa[q];
                fa[q]=fa[np]=nq;
                for(;ch[p][c]==q;p=fa[p])ch[p][c]=nq;
            }
        }
    }
    void init(char *s,int n){
        last=cnt=1;
        mem(ch[1],0);
        _for(i,0,n)
        extend(s[i]-'a');
    }
```

```
    int lcs(char *s,int n){
        int pos=1,cur=0,ans=0;
        _for(i,0,n){
            int c=s[i]-'a';
            if(ch[pos][c])cur++,pos=ch[pos][c];
            else{
                while(pos&&!ch[pos][c])pos=fa[pos];
                if(!pos)cur=0,pos=1;
                else
                cur=len[pos]+1,pos=ch[pos][c];
            }
            ans=max(ans,cur);
        }
        return ans;
    }
}solver;
char s[MAXN],t[MAXN];
int main()
{
    scanf("%s%s",s,t);
    solver.init(s,strlen(s));
    enter(solver.lcs(t,strlen(t)));
    return 0;
}
```

SA 题解

将两个串拼接后中间用 # 字符分隔，然后直接枚举相邻 sa 的 LCP 即可，注意只有相邻 sa 属于不同串才能产生贡献。

提前对 sa 进行染色即可，时间复杂度 $O(n\log n)$。

习题四

SP1812

题意

给定 n 个串，求 n 个串的 LCS。

SAM 题解

考虑将一个串压入 SAM，其他所有串在 SAM 上匹配。

对每个串，记录每个结点匹配的答案，然后每个结点取该结点所有匹配答案的最小值，然后所有结点的最大值即为所求答案。

注意 parent 树的子节点可以将答案贡献给父节点，可以每次匹配后根据拓扑序转移答案。时间复杂度 $O(\sum |S|)$。

```
const int MAXN=2e5+5;
struct SAM{
    int ch[MAXN][26],fa[MAXN],len[MAXN],last,cnt;
    int c[MAXN],topu[MAXN],maxv[MAXN],minv[MAXN];
    void extend(int c){
        int p=last,np=++cnt;
        last=np,len[np]=len[p]+1;
        for(;p&&!ch[p][c];p=fa[p])ch[p][c]=np;
        if(!p)fa[np]=1;
        else{
            int q=ch[p][c];
            if(len[q]==len[p]+1)fa[np]=q;
            else{
                int nq=++cnt;len[nq]=len[p]+1;
                memcpy(ch[nq],ch[q],sizeof(ch[q]));fa[nq]=fa[q];
                fa[q]=fa[np]=nq;
                for(;ch[p][c]==q;p=fa[p])ch[p][c]=nq;
            }
        }
    }
    void init(char *s,int n){
        last=cnt=1;
        mem(ch[1],0);
        _for(i,0,n)
        extend(s[i]-'a');
        _rep(i,1,cnt)c[len[i]]++,minv[i]=n;
        _rep(i,1,cnt)c[i]+=c[i-1];
        _rep(i,1,cnt)topu[c[len[i]]--]=i;
    }
    void lcs(char *s,int n){
        int pos=1,cur=0,ans=0;
        _for(i,0,n){
            int c=s[i]-'a';
            if(ch[pos][c])cur++,pos=ch[pos][c];
            else{
                while(pos&&!ch[pos][c])pos=fa[pos];
                if(!pos)cur=0,pos=1;
                else
                cur=len[pos]+1,pos=ch[pos][c];
            }
            maxv[pos]=max(maxv[pos],cur);
        }
        for(int i=cnt;i;i--){
            int u=topu[i];
            if(maxv[u])maxv[fa[u]]=len[fa[u]];
            minv[u]=min(minv[u],maxv[u]),maxv[u]=0;
        }
    }
}solver;
char s[MAXN];
```

```
int main()
{
    scanf("%s",s);
    solver.init(s,strlen(s));
    while(~scanf("%s",s))
    solver.lcs(s,strlen(s));
    int ans=0;
    _rep(i,1,solver.cnt)
    ans=max(ans,solver.minv[i]);
    enter(ans);
    return 0;
}
```