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 $O(n)$，空间复杂度 $O(\sum n)$。

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

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。

题解 1

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;
}
```

题解 2

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

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。

题解 1

根据 SAM 的性质三，不难得到过程，但注意需要 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;
}
```

From:
https://wiki.cvbbacm.com/ - **CVBB ACM Team**

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E5%AD%97%E7%AC%A6%E4%B8%B2_4&rev=1599013007

Last update: **2020/09/02 10:16**