序列自动机

算法思想

序列自动机是接受且仅接受一个字符串的子序列的自动机。字符串设为 s 。

若 s 包含 n 个字符，那么序列自动机包括 $n+1$ 个状态。

令 t 是 s 的一个子序列，则 $d(start,t)$ 是 t 在 s 第一次出现时末端的位置。

也就是说，一个状态 i 表示前缀 $s[1...i]$ 的子序列与前缀 $s[1...i-1]$ 的子序列的差集。

序列自动机上的所有状态都是接受状态。

$d(u,c)=min\{i|i>u,s[i]=c\}$ 也就是字符 c 下一次出现的位置。因为若 $i<j$ 后缀 $s[i...|s|]$ 的子序列是后缀 $s[j...|s|]$ 的子序列的子集，一定要选尽量靠前的，才能最优。

构建复杂度是 $O(n|\sum|)$ 。

算法实现

构造

```
struct SQAM {//构建复杂度是n方的
    int ch[MAXL][MAXM],las[MAXM],pre[MAXL];
    int root,tot;
    void init() {
        root=tot=1;
        for(int i=0; i<MAXM; i++) las[i]=1; //上一个是根节点
    }
    void insert(int c) {
        int p=las[c],np=++tot;//p是上一个字符c出现的结点 np是现在结点
        pre[np]=p;//现在结点的前驱是上一个出现的结点
        for(int i=0; i<MAXM; i++) { //对于每个字符对于字符c都要更新下一次出现的位置
            for(int j=las[i]; j&&!ch[j][c]; j=pre[j]) { //这个位置没有接过c这次接上 然后一直跳pre都更新
                ch[j][c]=np;
            }
        }
        las[c]=np;//别忘了把最新的位置更新
    }
} s1,s2;
```

代码练习

问题导入：

给定一个字符串 S ，q 次询问，每次给定另一个字符串 T ，询问 T 是否是 S 的子序列，$len(S)≤10^{5},q≤10^{5},\sum len(T)≤10^{6}$

最暴力的做法：建一个 t_{0} 指向所有点，然后每个点的后继是后面的点，最后每个点打结束标记，这样光建边就要 $O(len^{2})$ 再加上查询是 $O(\sum |T|)$ 显然爆炸。显然对于相同的字符贪心取前面的就好了，记录每种字符在哪些位置出现过。这个复杂度是 $O(len)$ 的，然后对于查找串的当前字符 $upper_bound(pos[c].begin(),pos[c].end(),now_pos);$ 如果查找到 end 说明不存在，如果一直找到最后说明存在。这个的复杂度是 $O(|S|+\sum |T|*log(|S|))$ 的。

```
const int MAXN=1001000;
const int MAXM=26;
int q;
char s[MAXN],t[MAXN];
vector<int> pos[MAXM];
bool work() {
    scanf("%s",t);
    int len=strlen(t);
    int nowpos=-1;
    for(int i=0;i<len;i++) {
        vector<int>:: iterator y=upper_bound(pos[t[i]-
'a'].begin(),pos[t[i]-'a'].end(),nowpos);
        if(y==pos[t[i]-'a'].end()) return false;
        nowpos=*y;
    }
    return true;
}

void solve() {
    scanf("%s",s);
    scanf("%d",&q);
    int len=strlen(s);
    for(int i=0;i<len;i++) {
        pos[s[i]-'a'].push_back(i);
    }
    while(q--) {
        if(work()) puts("Yes");
        else puts("No");
    }
}

int main() {
    solve();
    return 0;
}
```

如果要支持带修，把 $vector$ 换成平衡树即可（修改变成 $O(logn)$ 的）。

```
#include<iostream>
#include<cstdlib>
using namespace std;
const int N=1e5+10;
struct fhq_treap{
    int lson,rson;
    int val,key;
    int size;
};
fhq_treap fhq[N];
int root[N],a[N];
int tot;
inline int newnode(int val){
    tot++;
    fhq[tot].key=rand();
    fhq[tot].val=val;
    fhq[tot].size=1;
    return tot;
}
inline void pushup(int pos){
    fhq[pos].size=fhq[fhq[pos].lson].size+fhq[fhq[pos].rson].size+1;
}
void split(int pos,int val,int &x,int &y){
    if(!pos){
        x=y=0;
        return ;
    }
    if(fhq[pos].val<=val){
        x=pos;
        split(fhq[pos].rson,val,fhq[x].rson,y);
    }
    else{
        y=pos;
        split(fhq[pos].lson,val,x,fhq[y].lson);
    }
    pushup(pos);
}
int merge(int x,int y){
    if(!x||!y)
        return x+y;
    if(fhq[x].key<fhq[y].key){
        fhq[x].rson=merge(fhq[x].rson,y);
        pushup(x);
        return x;
    }
    else{
        fhq[y].lson=merge(x,fhq[y].lson);
        pushup(y);
        return y;
    }
```

```
}
void ins(int &pos,int val){
    int x,y;
    split(pos,val,x,y);
    pos=merge(merge(x,newnode(val)),y);
}
void del(int &pos,int val){
    int x,y,z;
    split(pos,val-1,x,y);
    split(y,val,y,z);
    y=merge(fhq[y].lson,fhq[y].rson);
    pos=merge(merge(x,y),z);
}
int get_suc(int &pos,int val){
    int x,y;
    split(pos,val,x,y);
    int ans=y;
    while(fhq[ans].lson)
        ans=fhq[ans].lson;
    pos=merge(x,y);
    return fhq[ans].val;
}
int main(){
    int t,n,m,k;
    cin>>t>>n>>m>>k;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        root[a[i]]=merge(root[a[i]],newnode(i));
    }
    int opt;
    int x,y;
    int pos=0;
    bool flag;
    for(int i=1;i<=m;i++){
        cin>>opt;
        if(opt){
            cin>>x;
            pos=0;
            flag=true;
            for(int j=1;j<=x;j++){
                cin>>y;
                pos=get_suc(root[y],pos);
                if(!pos)
                    flag=false;
            }
            if(flag)
                cout<<"Yes\n";
            else
                cout<<"No\n";
        }
```

```
        else{
            cin>>x>>y;
            del(root[a[x]],x);//删除
            a[x]=y;//修改
            ins(root[a[x]],x);//插入
        }
    }
}
```

1.https://www.luogu.com.cn/problem/P4112#submit

题意

给定两个长度不超过 2000 的字符串，分别设为 s_{1},s_{2}

分为四个问题：

- 求 s_{1} 的最短的子串且不是 s_{2} 的子串 - 求 s_{1} 的最短的子串且不是 s_{2} 的子序列 - 求 s_{1} 的最短的子序列且不是 s_{2} 的子串 - 求 s_{1} 的最短的子序列且不是 s_{2} 的子序列

如果没找到输出 -1。

题解

显然要分别对两个串建立后缀自动机和序列自动机

因为要找最短的，所以 BFS 开始搜，不管是什么自动机，从根节点往下开始搜，搜到 s_{1} 有结点且 s_{2} 没有的结点就可以了，如果到最后都没找到输出 -1 即可。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;

const int MAXL=2021;
const int MAXM=46;

const int MAXN=2021;
struct SAM {
    int las,tot;
    struct NODE {
        int ch[MAXM];
        int len,fa;
        NODE() {
            memset(ch,0,sizeof(ch));
            len=0;
            fa=0;
        }
```

```
    } dian[MAXN<<1];
    void init() {
        las=tot=1;
    }
    void add(int c) {
        int p=las;
        int np=las=++tot;
        dian[np].len=dian[p].len+1;
        for(; p&&!dian[p].ch[c]; p=dian[p].fa)dian[p].ch[c]=np;
        if(!p)dian[np].fa=1;//以上为case 1
        else {
            int q=dian[p].ch[c];
            if(dian[q].len==dian[p].len+1)dian[np].fa=q;//以上为case 2
            else {
                int nq=++tot;
                dian[nq]=dian[q];
                dian[nq].len=dian[p].len+1;
                dian[q].fa=dian[np].fa=nq;
                for(; p&&dian[p].ch[c]==q; p=dian[p].fa)dian[p].ch[c]=nq; //
以上为case 3
            }
        }
    }
}S1,S2;

char A[MAXL],B[MAXL];
int la,lb;

struct SQAM {//构建复杂度是n方的
    int ch[MAXL][MAXM],las[MAXM],pre[MAXL];
    int root,tot;
    void init() {
        root=tot=1;
        for(int i=0; i<MAXM; i++) las[i]=1; //上一个是根节点
    }
    void insert(int c) {
        int p=las[c],np=++tot;//p是上一个字符c出现的结点 np是现在结点
        pre[np]=p;//现在结点的前驱是上一个出现的结点
        for(int i=0; i<MAXM; i++) { //对于每个字符对于字符c都要更新下一次出现的位置
            for(int j=las[i]; j&&!ch[j][c]; j=pre[j]) { //这个位置没有接过c□这
次接上 然后一直跳pre都更新
                ch[j][c]=np;
            }
        }
        las[c]=np;//别忘了把最新的位置更新
    }
} s1,s2;

struct Node {
    int a,b,s;
```

```
};
int vis[MAXL<<1][MAXL<<1];

int BFS(int v){
    queue<Node>q;
    vis[1][1]=v;
    Node ntmp;
    ntmp.a=1,ntmp.b=1,ntmp.s=0;
    q.push(ntmp);
    while(!q.empty()) {
        ntmp=q.front();
        int a=ntmp.a,b=ntmp.b,s=ntmp.s;
        //printf("%d %d %d %d\n",a,b,s,v);
        q.pop();
        for(int i=0,da,db;i<MAXM;i++) {
            if(v==1) {
                da=S1.dian[a].ch[i],db=S2.dian[b].ch[i];
            }else if(v==2) {
                da=S1.dian[a].ch[i],db=s2.ch[b][i];
            }else if(v==3) {
                da=s1.ch[a][i],db=S2.dian[b].ch[i];
            }else if(v==4) {
                da=s1.ch[a][i],db=s2.ch[b][i];
            }
            //printf("%d %d %d\n",da,db,v);
            if(vis[da][db]==v) continue;
            if(da&&!db) return s+1;
            if(da&&db) {
                Node nntmp;
                nntmp.a=da,nntmp.b=db,nntmp.s=s+1;
                q.push(nntmp);
                vis[da][db]=v;
            }
        }
    }
    return -1;
}

int main() {
    scanf("%s%s",A,B);
    la=strlen(A); lb=strlen(B);
    s1.init();s2.init();S1.init();S2.init();
    for(int i=0; i<la; i++) {
        S1.add(A[i]-'a');
        s1.insert(A[i]-'a');
    }
    for(int i=0; i<lb; i++) {
        S2.add(B[i]-'a');
        s2.insert(B[i]-'a');
    }
    int a1=BFS(1),a2=BFS(2),a3=BFS(3),a4=BFS(4);
```

```
    printf("%d\n%d\n%d\n%d\n",a1,a2,a3,a4);
    return 0;
}
```