ACM板子合集

字符串

hash(字符串前缀哈希)

双hash（正确率高，但常数大）

```
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)

struct hash_t{
    typedef long long LL;
    int nn=0;
    const LL base1=29,mod1=1e9+7;
    const LL base2=131,mod2=1e9+9;

    // 字符串前缀哈希值与后缀逆哈希值
    vector<LL> p1,p2,h1,h2,rh1,rh2;

    hash_t(){}

    hash_t(char* s,int len):nn(len),p1(len+1),p2(len+1),
            h1(len+1),h2(len+1),rh1(len+1),rh2(len+1){
        p1[0]=p2[0]=1;
        for(int i=1;i<=nn;++i){
            p1[i]=p1[i-1]*base1%mod1;
            p2[i]=p2[i-1]*base2%mod2;
        }
        for(int i=1;i<=nn;++i){
            h1[i]=(h1[i-1]*base1+s[i])%mod1;
            h2[i]=(h2[i-1]*base2+s[i])%mod2;
            rh1[i]=(rh1[i-1]*base1+s[nn-i+1])%mod1;
            rh2[i]=(rh2[i-1]*base2+s[nn-i+1])%mod2;
        }
    }

    // 全局变量的构造
    void init(char* s,int len){
        nn=len;
        p1.resize(len+1),p2.resize(len+1);
```

```
        h1.resize(len+1),h2.resize(len+1);
        rh1.resize(len+1),rh2.resize(len+1);
        p1[0]=p2[0]=1;
        for(int i=1;i<=nn;++i){
            p1[i]=p1[i-1]*base1%mod1;
            p2[i]=p2[i-1]*base2%mod2;
        }
        for(int i=1;i<=nn;++i){
            h1[i]=(h1[i-1]*base1+s[i])%mod1;
            h2[i]=(h2[i-1]*base2+s[i])%mod2;
            rh1[i]=(rh1[i-1]*base1+s[nn-i+1])%mod1;
            rh2[i]=(rh2[i-1]*base2+s[nn-i+1])%mod2;
        }
    }

    // 不用后缀逆哈希值的构造
    void init_simple(char* s,int len){
        nn=len;
        p1.resize(len+1),p2.resize(len+1);
        h1.resize(len+1),h2.resize(len+1);
        p1[0]=p2[0]=1;
        for(int i=1;i<=nn;++i){
            p1[i]=p1[i-1]*base1%mod1;
            p2[i]=p2[i-1]*base2%mod2;
        }
        for(int i=1;i<=nn;++i){
            h1[i]=(h1[i-1]*base1+s[i])%mod1;
            h2[i]=(h2[i-1]*base2+s[i])%mod2;
        }
    }

    // 哈希值
    pair<LL,LL> get_hash(int l,int r)const{
        LL res1=((h1[r]-h1[l-1]*p1[r-l+1])%mod1+mod1)%mod1;
        LL res2=((h2[r]-h2[l-1]*p2[r-l+1])%mod2+mod2)%mod2;
        return {res1,res2};
    }
    // 逆哈希值
    pair<LL,LL> get_rhash(int l,int r)const{
        LL res1=((rh1[nn-l+1]-rh1[nn-r]*p1[r-l+1])%mod1+mod1)%mod1;
        LL res2=((rh2[nn-l+1]-rh2[nn-r]*p2[r-l+1])%mod2+mod2)%mod2;
        return {res1,res2};
    }
    bool is_palindrome(int l,int r)const{
        return get_hash(l,r)==get_rhash(l,r);
    }

    // hash +
    pair<LL,LL> add(const pair<LL,LL>& a,const pair<LL,LL>& b)const{
        LL res1=(a.first+b.first)%mod1;
        LL res2=(a.second+b.second)%mod2;
        return {res1,res2};
    }

    // *= base ^ k
    pair<LL,LL> mul(const pair<LL,LL>& a,LL k)const{
```

```
        LL res1=a.first*p1[k]%mod1;
        LL res2=a.second*p2[k]%mod2;
        return {res1,res2};
    }

    // concat
    pair<LL,LL> cat(int l1,int r1,int l2,int r2)const{
        return add(mul(get_hash(l1,r1),r2-l2+1),get_hash(l2,r2));
    }
};

const int N=2e6+10;

int n;

char s[N];

hash_t H;

vector<pair<long long,pair<long long,long long>>> ans;

int main(){
    // https://loj.ac/p/2823
    n=read();
    scanf("%s",s+1);

    if(n%2==0){
        puts("NOT POSSIBLE");
        return 0;
    }

    H.init_simple(s,n);
    rep(i,1,n/2){
        auto hs1=H.cat(1,i-1,i+1,n/2+1);
        auto hs2=H.get_hash(n/2+2,n);
        if(hs1==hs2){
            ans.push_back({i,hs1});
        }
    }
    if(H.get_hash(1,n/2)==H.get_hash(n/2+2,n)){
        ans.push_back({n/2+1,H.get_hash(1,n/2)});
    }
    rep(i,n/2+2,n){
        auto hs1=H.get_hash(1,n/2);
        auto hs2=H.cat(n/2+1,i-1,i+1,n);
        if(hs1==hs2){
            ans.push_back({i,hs1});
        }
    }

    int isok=1;
    pair<long long,long long> lst;
    for(auto&& [id,hs]:ans){
        // cout<<id<<' '<<hs.first<<' '<<hs.second<<'\n';
        if(!(lst==make_pair(0ll,0ll))&&!(hs==lst)){
            isok=0;
```

```
            break;
        }
        lst=hs;
    }

    if(ans.size()==0)puts("NOT POSSIBLE");
    else if(isok){
        string res;
        rep(i,1,n)if(i!=ans[0].first){
            res+=s[i];
            if(res.size()==n/2)break;
        }
        cout<<res<<'\n';
    }else puts("NOT UNIQUE");
    return 0;
}
```

单hash自然溢出（常数小，但可能被卡）

```
struct hash_t{
    typedef unsigned long long ull;
    int nn=0;
    const ull base=131;

    // 字符串前缀哈希值与后缀逆哈希值
    vector<ull> p,h,rh;

    hash_t(){}

    hash_t(char* s,int len):nn(len),p(len+1),
            h(len+1),rh(len+1){
        p[0]=1;
        for(int i=1;i<=nn;++i){
            p[i]=p[i-1]*base;
        }
        for(int i=1;i<=nn;++i){
            h[i]=(h[i-1]*base+s[i]);
            rh[i]=(rh[i-1]*base+s[nn-i+1]);
        }
    }

    // 全局变量的构造
    void init(char* s,int len){
        nn=len;
        p.resize(len+1);
        h.resize(len+1);
        rh.resize(len+1);
        p[0]=1;
        for(int i=1;i<=nn;++i){
            p[i]=p[i-1]*base;
        }
        for(int i=1;i<=nn;++i){
            h[i]=(h[i-1]*base+s[i]);
            rh[i]=(rh[i-1]*base+s[nn-i+1]);
        }
```

```
    }

    // 不用后缀逆哈希值的构造
    void init_simple(char* s,int len){
        nn=len;
        p.resize(len+1);
        h.resize(len+1);
        p[0]=1;
        for(int i=1;i<=nn;++i){
            p[i]=p[i-1]*base;
        }
        for(int i=1;i<=nn;++i){
            h[i]=(h[i-1]*base+s[i]);
        }
    }

    // 哈希值
    ull get_hash(int l,int r)const{
        ull res=(h[r]-h[l-1]*p[r-l+1]);
        return res;
    }
    // 逆哈希值
    ull get_rhash(int l,int r)const{
        ull res=(rh[nn-l+1]-rh[nn-r]*p[r-l+1]);
        return res;
    }
    bool is_palindrome(int l,int r)const{
        return get_hash(l,r)==get_rhash(l,r);
    }

    // concat
    ull cat(int l1,int r1,int l2,int r2)const{
        return get_hash(l1,r1)*p[r2-l2+1]+get_hash(l2,r2);
    }
};
```

KMP

```
#include <bits/stdc++.h>

using namespace std;

const int N=1e6+10;
const int INF=0x3f3f3f3f;

int a[N],b[N];
int n,m;

inline int read(){
    int res=0,sign=1;char ch=getchar();
    for(;ch>'9'||ch<'0';ch=getchar()){
        if(ch=='-')sign=-sign;
    }
    for(;'0'<=ch&&ch<='9';ch=getchar()){
        res=(res<<1)+(res<<3)+(ch^'0');
```

```
    }
    return res*sign;
}

int nxt[N];

void getnxt(){
    for(int i=2,j;i<=m;++i){
        for(j=nxt[i-1];j&&b[i]!=b[j+1];)j=nxt[j];
        if(b[i]==b[j+1])++j;
        nxt[i]=j;
    }
}

void kmp(){
    int flag=0;
    for(int i=1,j=0;i<=n;++i){
        while(j&&a[i]!=b[j+1])j=nxt[j];
        if(a[i]==b[j+1])++j;
        if(j==m){
            cout<<i-m+1<<'\n';
            flag=1;
            break;
        }
    }
    if(!flag)cout<<-1<<'\n';
}

void solve(){
    n=read(),m=read();
    for(int i=1;i<=n;++i)a[i]=read();
    for(int i=1;i<=m;++i)b[i]=read();
    getnxt();
    kmp();
}

int main(){
    int t;cin>>t;
    while(t--)solve();
    return 0;
}
```

hash 法求字符串匹配位置

每个位置比一下 hash 值就好了，$O(n)$的和 kmp 一样。

Manacher

```
#include <bits/stdc++.h>

using namespace std;

const int N=1.1e7+10;

char s[N<<1],dat[N<<1];
```

```
int len,cnt;

int p[N<<1],ans;

int main(){
    scanf("%s",s+1);len=strlen(s+1);
    dat[0]='~',dat[cnt=1]='#';
    for(int i=1;i<=len;++i){
        dat[++cnt]=s[i],dat[++cnt]='#';
    }
    for(int i=1,mid=0,r=0;i<=cnt;++i){
        if(i<=r)p[i]=min(p[(mid<<1)-i],r-i+1);
        else p[i]=1;
        while(dat[i-p[i]]==dat[i+p[i]])++p[i];
        if(i+p[i]>r)r=i+p[i]-1,mid=i;
        ans=max(ans,p[i]);
    }
    printf("%d\n",ans-1);
    return 0;
}
```

hash 法求manacher（多一个log）

```
int n;

char s[N];

hash_t H;

// 每个回文中心下的最长回文串，长度分奇偶
// 要求所有回文的话就每组最长 [l,r] 往内枚举即可
// 要求本质不同所有回文，对每个长度加一个哈希表来判断即可，
// 一旦出现枚举过的，那么内部的所有被包含的都枚举过了
// 本质不同回文个数O(n)

vector<pair<int,int>> get_manacher(){
    vector<pair<int,int>> res;
    rep(i,1,n){
        int ql=0,qr=min(i-1,n-i);
        while(ql<=qr){
            int qmid=ql+qr>>1;
            if(H.is_palindrome(i-qmid,i+qmid))ql=qmid+1;
            else qr=qmid-1;
        }
        res.push_back({i-qr,i+qr});
    }
    rep(i,1,n-1){
        int ql=0,qr=min(i-1,n-i-1);
        while(ql<=qr){
            int qmid=ql+qr>>1;
            if(H.is_palindrome(i-qmid,i+1+qmid))ql=qmid+1;
            else qr=qmid-1;
        }
        res.push_back({i-qr,i+1+qr});
```

```
    }
    return res;
}

int main(){
    scanf("%s",s+1);
    n=strlen(s+1);
    H.init(s,n);
    auto ans=get_manacher();
    // for(auto&& [l,r]:ans)printf("%d %d\n",l,r);

    // 最长的回文串的长度
    // 多一个二分的 log ，过不了 luogu
    int mx=0;
    for(auto&& [l,r]:ans){
        mx=max(mx,r-l+1);
    }
    printf("%d\n",mx);
    return 0;
}
```

Trie

```
#include <bits/stdc++.h>

using namespace std;

const int MAXN=1e4+2;

struct trie{
    int nxt[MAXN<<5][30],cnt;
    bool exist[MAXN<<5],vis[MAXN<<5];
    void insert(char s[],int len){
        int p=0;
        for(int i=0;i<len;++i){
            int c=s[i]-'a';
            if(!nxt[p][c])nxt[p][c]=++cnt;
            p=nxt[p][c];
        }
        exist[p]=1;
    }

    int find(char s[],int len){
        int p=0;
        for(int i=0;i<len;++i){
            int c=s[i]-'a';
            if(!nxt[p][c])return 0;
            p=nxt[p][c];
        }
        if(exist[p]){
            if(!vis[p])return vis[p]=true,1;
            else return 2;
        }else return 0;
    }
};
```

```
int n,m;
trie t;
string tmp;
char _tmp[60];

int main(){
    cin>>n;
    for(int i=1;i<=n;++i){
        cin>>tmp;
        for(int i=0;i<tmp.size();++i)_tmp[i]=tmp[i];
        t.insert(_tmp,tmp.size());
    }
    cin>>m;
    for(int i=1;i<=m;++i){
        cin>>tmp;
        for(int i=0;i<tmp.size();++i)_tmp[i]=tmp[i];
        int op=t.find(_tmp,tmp.size());
        if(op==0)cout<<"WRONG"<<'\n';
        else if(op==1)cout<<"OK"<<'\n';
        else if(op==2)cout<<"REPEAT"<<'\n';
    }
    return 0;
}
```

ACAM

```
#include <bits/stdc++.h>

using namespace std;

const int MAXN=1e6+2;

struct trie{
    int nxt[MAXN][30],cnt,end[MAXN],fail[MAXN];
    void insert(string s){
        int len=s.size();
        int p=0;
        for(int i=0;i<len;++i){
            int c=s[i]-'a';
            if(!nxt[p][c])nxt[p][c]=++cnt;
            p=nxt[p][c];
        }
        end[p]+=1;
    }

    void build(){
        queue<int> q;
        for(int i=0;i<26;++i){
            if(nxt[0][i])q.push(nxt[0][i]);
        }
        while(!q.empty()){
            int x=q.front();
            q.pop();
            for(int i=0;i<26;++i)if(nxt[x][i]){
```

```
                fail[nxt[x][i]]=nxt[fail[x]][i];
                q.push(nxt[x][i]);
            }else nxt[x][i]=nxt[fail[x]][i];
        }
    }

    int query(string s){
        int p=0,ret=0,len=s.size();
        for(int i=0,c;i<len;++i){
            c=s[i]-'a';
            p=nxt[p][c];
            for(int j=p;j&&end[j]!=-1;j=fail[j]){
                ret+=end[j],end[j]=-1;
            }
        }
        return ret;
    }
};

trie t;
int n;
string tmp;

int main(){
    cin>>n;
    for(int i=1;i<=n;++i){
        cin>>tmp;
        t.insert(tmp);
    }
    t.build();
    cin>>tmp;
    cout<<t.query(tmp)<<'\n';
    return 0;
}
```

ACAM + TOPO

```
#include <bits/stdc++.h>

using namespace std;

const int N=2e5+10;
const int PN=2e5+10;
const int TN=2e6+10;

struct aca_t{
    int n;
    char mods[N],txt[TN];
    int tot,vis[N],rev[N],deg[PN],ans;

    struct node_t{
        int son[27],fail,flag,ans;

        void init(){
            memset(son,0,sizeof son);
```

```
            fail=flag=0;
        }
    }trie[PN];

    void init(){
        for(int i=0;i<=tot;++i)trie[i].init();
        for(int i=1;i<=n;++i)vis[i]=0;
        tot=1;
        ans=0;
    }

    void insert(char* s,int len,int idx){
        int p=1;
        for(int i=1;i<=len;++i){
            int c=s[i]-'a';
            if(!trie[p].son[c])trie[p].son[c]=++tot;
            p=trie[p].son[c];
        }
        if(!trie[p].flag)trie[p].flag=idx;
        rev[idx]=trie[p].flag;
    }

    void getfail(){
        queue<int> q;
        for(int i=0;i<26;++i)trie[0].son[i]=1;
        q.push(1);
        trie[1].fail=0;
        while(!q.empty()){
            int u=q.front();q.pop();
            int Fail=trie[u].fail;
            for(int i=0;i<26;++i){
                int v=trie[u].son[i];
                if(v){
                    trie[v].fail=trie[Fail].son[i];
                    ++deg[trie[Fail].son[i]];
                    q.push(v);
                }else trie[u].son[i]=trie[Fail].son[i];
            }
        }
    }

    void query(char* s,int len){
        int p=1;
        for(int i=1;i<=len;++i){
            int c=s[i]-'a';
            p=trie[p].son[c];
            ++trie[p].ans;
        }
    }

    void topo(){
        queue<int> q;
        for(int i=1;i<=tot;++i){
            if(!deg[i])q.push(i);
        }
        while(!q.empty()){
```

```
            int u=q.front();q.pop();
            vis[trie[u].flag]=trie[u].ans;
            int v=trie[u].fail;
            trie[v].ans+=trie[u].ans;
            if(!(--deg[v]))q.push(v);
        }
    }

    void solve(){
        scanf("%d",&n);
        init();
        for(int i=1;i<=n;++i){
            scanf("%s",mods+1);
            int len=strlen(mods+1);
            insert(mods,len,i);
        }
        getfail();
        scanf("%s",txt+1);
        int len=strlen(txt+1);
        query(txt,len);
        topo();
        for(int i=1;i<=n;++i)printf("%d\n",vis[rev[i]]);
    }
}aca;

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

本质不同子序列个数

通过`dp`可以求到，转移方程很好想，下面代码给出子串`[l,r]`之间的本质不同子序列个数计算

```
int _dp(int l,int r){
    int len=r-l+1;
    vector<int> dp(N),lst(30);
    rep(i,1,len){
        int c=S[i+l-1]-'a';
        if(!lst[c]){
            dp[i]=(1ll*dp[i-1]*2%mod+1)%mod;
        }else{
            dp[i]=((1ll*dp[i-1]*2%mod-dp[lst[c]-1])%mod+mod)%mod;
        }
        lst[c]=i;
    }
    return dp[len];
}
```

两字符串的 lcp

可以通过 hash + 二分求，下面给出 A 的每一个后缀和 B 的 lcp 的代码（询问满足 lcp = x 的有多少）

```
const int N=2e6+10;

int n,m,q;

char s1[N];
char s2[N];

hash_t H1,H2;

int cnt[N];

int main(){
    n=read(),m=read(),q=read();
    scanf("%s",s1+1);
    scanf("%s",s2+1);
    H1.init_simple(s1,n);
    H2.init_simple(s2,m);
    rep(i,1,n){
        int l=0,r=min(n-i+1,m);
        while(l<=r){
            int mid=l+r>>1;
            if(H1.get_hash(i,i+mid-1)==H2.get_hash(1,mid))l=mid+1;
            else r=mid-1;
        }
        ++cnt[r];
    }
    rep(_,1,q){
        int x=read();
        printf("%d\n",cnt[x]);
    }
    return 0;
}
```

两字符串的大小

先求 lcp

如果 $lcp(A,B) >= min(lenA,lenB)$，则 $lenA < lenB$ 等价于 $A < B$，否则 $A[lcp(A,B)+1] < B[lcp(A,B)+1]$ 等价于 $A < B$

所以 sa 数组通过该方式来比较字符串来得到的时间复杂度是 $O(n \log n \log n)$，不过被 $O(n)$ 的方法薄纱

```
const int N=1e6+10;

char s[N];

int n;
hash_t H;
int sa[N],rk[N];
```

```
bool cmp(int i,int j){
    int l=1,r=min(n-i+1,n-j+1)+1;
    while(l<=r){
        int mid=l+r>>1;
        if(H.get_hash(i,i+mid-1)!=H.get_hash(j,j+mid-1))r=mid-1;
        else l=mid+1;
    }
    if(l>min(n-i+1,n-j+1))return i>j;
    else return s[i+l-1]<s[j+l-1];
}

int main(){
    scanf("%s",s+1);
    n=strlen(s+1);
    H.init_simple(s,n);
    for(int i=1;i<=n;++i)sa[i]=i;
    stable_sort(sa+1,sa+n+1,cmp);
    for(int i=1;i<=n;++i)printf("%d ",sa[i]);puts("");
    // for(int i=1;i<=n;++i)rk[sa[i]]=i;
    // for(int i=1;i<=n;++i)printf("%d ",rk[i]);puts("");
    return 0;
}
```

SA

记后缀位置 i 代表的是后缀串 s[i~n]，用后缀 i 简记该串

后缀数组 sa[i] 代表第 i 名次的后缀位置

rk[i] 即后缀串 s[i~n] 在所有后缀中的排名

记 $lcp(i,j)$ 为后缀 i 和后缀 j 的最长公共前缀的长度

height 数组代表 $height[i]=lcp(sa[i],sa[i-1])$

有引理 $height[rk[i]]>=height[rk[i-1]]-1$，用该引理可以在已知 sa 和 rk 的情况下 $O(n)$ 计算height

height 数组的运用：

1. 两子串的 lcp：可以转化成后缀串的 lcp 和俩子串长度求 min，而后缀串的 lcp 满足公式：$lcp(sa[i],sa[j])=min_{k=i+1}^{j}height[k]$
2. 比较两子串大小：A=s[a~b]，B=s[c~d]，如果 $lcp(a,c)>=min(lenA,lenB)$，则 $lenA<lenB$ 等价于 $A<B$，否则 $rk[a]<rk[c]$ 等价于 $A<B$
3. 不同子串的数量：$n(n+1)/2-\sum_{i=2}^{n}height[i]$

sa 的求法有 $O(n\cdot logn)$的倍增+基数排序的求法，或者 $O(n\cdot \log n\cdot \log n)$的 hash 法，下面给一个最快的 sais 方法，优点是常数小且时间复杂度 $O(n)$ 线性

```
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;
    char ch=getchar();
```

```
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

const int N=1e6+10;

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)

#define pus(x) (sa[cur[a[x]]--]=x)
#define pul(x) (sa[cur[a[x]]++]=x)
#define inds(lms)                                   \
    for(int i=1;i<=n;++i)sa[i]=-1;                  \
    for(int i=1;i<=n;++i)sum[i]=0;                  \
    for(int i=1;i<=n;++i)++sum[a[i]];               \
    for(int i=1;i<=n;++i)sum[i]+=sum[i-1];          \
    for(int i=1;i<=n;++i)cur[i]=sum[i];             \
    for(int i=m;i>=1;--i)pus(lms[i]);               \
    for(int i=1;i<=n;++i)cur[i]=sum[i-1]+1;         \
    for(int i=1;i<=n;++i)                           \
        if(sa[i]>1&&!tp[sa[i]-1])                   \
            pul(sa[i]-1);                           \
    for(int i=1;i<=n;++i)cur[i]=sum[i];             \
    for(int i=n;i>=1;--i)                           \
        if(sa[i]>1&&tp[sa[i]-1])                    \
            pus(sa[i]-1);

int sa[N],sum[N],cur[N],rak[N];

void sais(int* a,int n){
    vector<int> tp(n+5),p(n+5);
    tp[n]=1;
    for(int i=n-1;i>=1;--i)tp[i]=(a[i]==a[i+1])?tp[i+1]:(a[i]<a[i+1]);
    int m=0;
    for(int i=1;i<=n;++i)rak[i]=(tp[i]&&!tp[i-1])?(p[++m]=i,m):-1;
    inds(p);
    int tot=0;
    int *a1=new int[m+5];
    p[m+1]=n;
    for(int i=1,x,y;i<=n;++i)if((x=rak[sa[i]])!=-1){
            if(tot==0||p[x+1]-p[x]!=p[y+1]-p[y])
                ++tot;
            else for(int p1=p[x],p2=p[y];p2<=p[y+1];++p1,++p2)
                    if((a[p1]<<1|tp[p1])!=(a[p2]<<1|tp[p2])){++tot;break;}
            a1[y=x]=tot;
        }
    if(tot==m)
        for(int i=1;i<=m;++i)sa[a1[i]]=i;
    else
        sais(a1,m);
    for (int i=1;i<=m;++i)a1[i]=p[sa[i]];
    inds(a1);
    delete a1;
}
```

```
char str[N];

int n;

int a[N],cnt[300];

void get_sa(char* str,int len){
    // 用桶离散化字符串，字符串最后一位拼接一个绝对最小字符
    for(int i=1;i<300;++i)cnt[i]=0;
    for(int i=1;i<=len;++i)cnt[str[i]]=1;
    for(int i=1;i<300;++i)cnt[i]+=cnt[i-1];
    for(int i=1;i<=len;++i)a[i]=cnt[str[i]]+1;
    a[++len]=1;
    sais(a,len);
    for(int i=1;i<=len-1;++i)sa[i]=sa[i+1];
}

int rk[N],height[N];

void get_rk(){
    for(int i=1;i<=n;++i)rk[sa[i]]=i;
}

void get_height(){
    for(int i=1,k=0;i<=n;++i){
        if(rk[i]==0)continue;
        if(k)--k;
        while(str[i+k]==str[sa[rk[i]-1]+k])++k;
        height[rk[i]]=k;
    }
}

int main(){
    scanf("%s",str+1);
    n=strlen(str+1);
    get_sa(str,n);
    for(int i=1;i<=n;++i)printf("%d ",sa[i]);puts("");
    get_rk();
    get_height();
    for(int i=1;i<=n;++i)printf("%d ",rk[i]);puts("");
    for(int i=1;i<=n;++i)printf("%d ",height[i]);puts("");
    return 0;
}
```

$O(\log n)$求子串出现次数

设子串为 $s[x,y]$

后缀排序后后缀 $s[x,n]$的附近的后缀串只要与其的 lcp 的长度大于等于子串的长度，就视为该字串多出现了一次，所以就可以二分来得到子串出现的次数

```
// 前面的 sa rk height 已经求好
/*rmq*/

const int LOGN=21;
```

```
int f[N][LOGN],_log_2[N];

void pre(){
    _log_2[1]=0;
    _log_2[2]=1;
    for(int i=3;i<N;++i){
        _log_2[i]=_log_2[i/2]+1;
    }
    for(int i=1;i<=n;++i)f[i][0]=height[i];
    for(int k=1;k<=LOGN;++k){
        for(int i=1;i+(1<<k)-1<=n;++i){
            f[i][k]=min(f[i][k-1],f[i+(1<<(k-1))][k-1]);
        }
    }
}

int getmin(int l,int r){
    int ret;
    int s=_log_2[r-l+1];
    ret=min(f[l][s],f[r-(1<<s)+1][s]);
    return ret;
}

int lcp(int l,int r){
    if(l==r)return 0x3f3f3f3f;
    return getmin(l+1,r);
}

int query_occur_cnt(int x,int y){
    int len=y-x+1;
    int rank=rk[x];
    long long res=1;
    int ql=0,qr=n-rank;
    while(ql<=qr){
        int qmid=ql+qr>>1;
        if(lcp(rank,rank+qmid)>=len)ql=qmid+1;
        else qr=qmid-1;
    }
    res+=qr;
    ql=0,qr=rank-1;
    while(ql<=qr){
        int qmid=ql+qr>>1;
        if(lcp(rank-qmid,rank)>=len)ql=qmid+1;
        else qr=qmid-1;
    }
    res+=qr;
    return res;
}
```

SAM

对一个字符串 T 做 SAM(T)，该 SAM 接受所有 T 的后缀

记录 $endpos(s)$ 代表在 T 匹配到 s(s 是 T 子串) 的所有**结束位置**，易证 $endpos(s_1)$ 与 $endpos(s_2)$ 要么交集为空，要么为从属关系

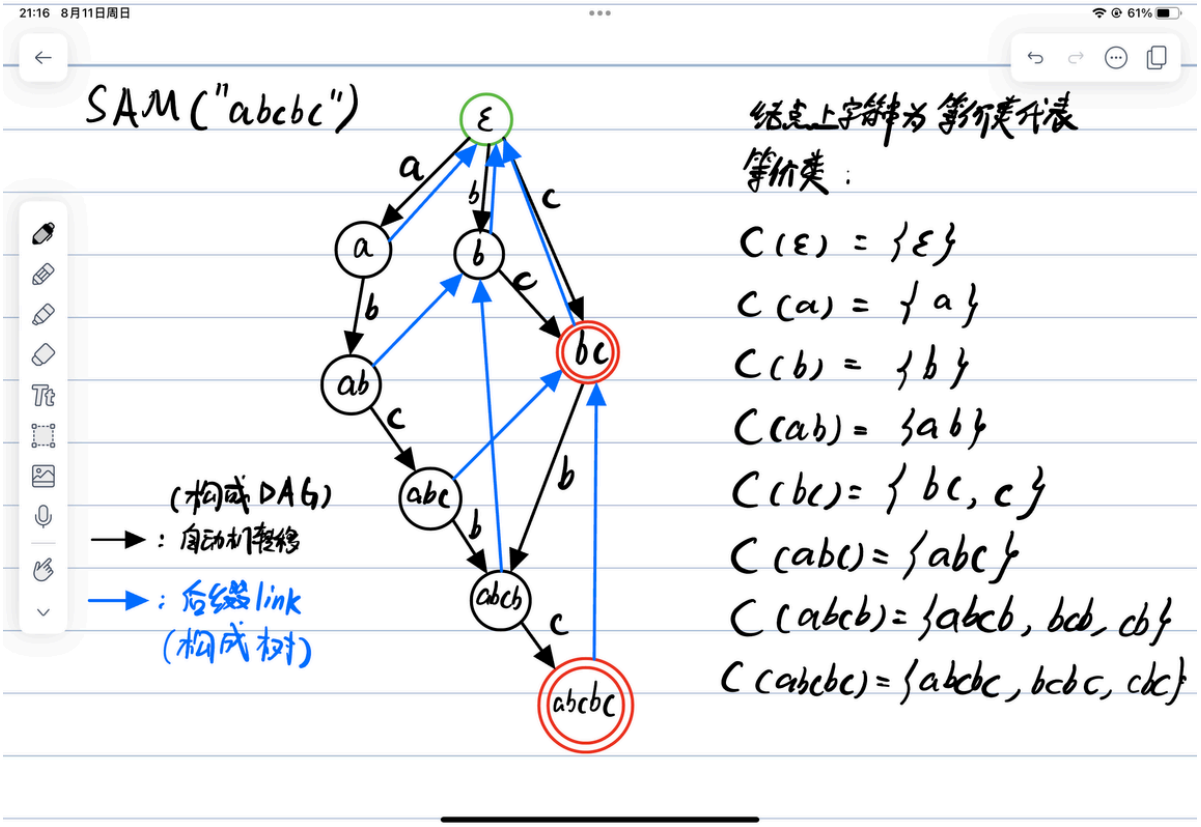
SAM 的每一个节点代表 endpos 相同的等价类，**等价类代表**不妨设为等价类中**最长的那个**，按照长度对等价类降序排序后，后面每一个都是前面的后缀

考虑某个等价类 $C(s)$，等价类的代表不妨为 s_max，记录每个等价类的 $maxlen(C(s)) = |s_{max}|$，$minlen(C(s))$，所以等价类 $C(s)$ 里的元素有
$\{s_{max}, s_{max}[2:maxlen], s_{max}[3:maxlen], \dots, s_{max}[maxlen - minlen + 1 : maxlen]\}$

而 $s' = s_{max}[maxlen - minlen + 2 : maxlen]$ 之后等等 s_max 的后缀就不会在原来的那一个等价类中，s' 应该是另一个等价类 $C(s')$ 的等价类代表，我们记录这个后缀链接 $link(C(s)) = C(s')$，易证 link 构成以 $C(\epsilon)$ 的一棵树，以及 $endpos(s) \subsetneqq endpos(s')$，
$minlen(C(s)) = maxlen(C(s')) + 1$

根据这些性质，从某个节点 $C(s)$ 一直跳 link 会回到 $C(\epsilon)$

`SAM("abcbc")` 示意图如下：



下面给出 SAM 的经典问题的解：

```cpp
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)
```

```cpp
/************************SAM_BEGIN*******************************/

struct node_t{
    // len 对应等价类代表的长度
    // link 为后缀链接
    // nxt 为自动机转移
    // siz 为 endpos 集合的大小(等价类中串出现过的次数), dfs pattern tree 后才处理好
    int len,link,siz;
    map<char,int> nxt;

    bool is_clone;
    int first_pos;
    vector<int> inv_link;
};

const int MAXLEN=1e6+10;
node_t node[MAXLEN<<1];
int sz,lst;

void init(){
    sz=0;
    // 0 为初始状态
    node[0].len=0;
    node[0].link=-1;
    ++sz;
    lst=0;
    // 若多次使用请在下面开 2 * n 个节点
    // n 为构造 sam 的字符串的长度，节点个数不超过2 * n - 1
    // for(int i=1;i<=n*2+10;++i){
    //  node[i].len=node[i].link=node[i].siz=node[i].first_pos=0;
    //  node[i].is_clone=false;
    //  node[i].nxt.clear();
    //  node[i].inv_link.clear();
    // }
}

void extend(char c){
    int cur=sz++;
    node[cur].len=node[lst].len+1;
    node[cur].siz=1;
    node[cur].is_clone=false;
    node[cur].first_pos=node[cur].len;
    int p=lst;
    while(p!=-1&&!node[p].nxt.count(c)){
        node[p].nxt[c]=cur;
        p=node[p].link;
    }
    if(p==-1){
        node[cur].link=0;
    }else{
        int q=node[p].nxt[c];
        if(node[p].len+1==node[q].len){
            node[cur].link=q;
        }else{
            int clone=sz++;
            node[clone].len=node[p].len+1;
```

```
            node[clone].link=node[q].link;
            node[clone].nxt=node[q].nxt;
            node[clone].is_clone=true;
            node[clone].first_pos=node[q].first_pos;
            while(p!=-1&&node[p].nxt[c]==q){
                node[p].nxt[c]=clone;
                p=node[p].link;
            }
            node[q].link=node[cur].link=clone;
        }
    }
    lst=cur;
}


void build_pattern_tree(){
    for(int i=1;i<=sz-1;++i){
        node[node[i].link].inv_link.push_back(i);
    }
}

// long long ans;

void dfs(int u){
    for(auto&& v:node[u].inv_link){
        dfs(v);
        node[u].siz+=node[v].siz;
    }
    // luogu template: 出现超过1次的子串，其长度和出现次数乘积最大值
    // if(node[u].siz>1)ans=max(ans,1ll*node[u].siz*node[u].len);
}

/************************SAM_END*******************************/

/*把模式串放 sam 里跑，中途转移不了就不是子串，最后跑出来的地方就是等价类那个状态*/

int query_node(char* s,int len){
    int p=0;
    for(int i=1;i<=len;++i){
        char c=s[i];
        if(node[p].nxt.count(c)){
            p=node[p].nxt[c];
        }else{
            // 不是子串
            return -1;
        }
    }
    return p;
}

/*模式串所属等价类的endpos中的第一个可以维护出来，然后打印就好*/

int query_first_pos(char* s,int len){
    int p=query_node(s,len);
    if(p!=-1)return node[p].first_pos-len+1;
    return -1;
```

```
}

/*模式串的所有匹配位置可以从等价类的状态 p 往下跑 pattern tree，
p 的子树中节点保证也是有该串的，因为 p 的子树节点表示的等价类中，
都包含模式串作为一个后缀*/

int all_pos[MAXLEN],idx;

void find_all_pos(int u,int len){
    if(!node[u].is_clone)all_pos[++idx]=node[u].first_pos-len+1;
    for(auto&& v:node[u].inv_link){
        find_all_pos(v,len);
    }
}

void query_all_pos(char* s,int len){
    idx=0;
    int p=query_node(s,len);
    if(p!=-1){
        find_all_pos(p,len);
    }
    sort(all_pos+1,all_pos+idx+1);
}

/*每个等价类中的串出现的次数通过 dfs pattern tree 后得到，
就是 endpos 集合的大小*/

int query_occur_cnt(char* s,int len){
    int p=query_node(s,len);
    if(p!=-1){
        return node[p].siz;
    }
    return 0;
}

/*本质不同子串个数可以通过所有节点 u 的 len(u) - len(link(u)) 得到，
如果要动态维护本质不同子串个数，也就是每次加一个字符后又询问，就在 extend 的时候
把新增节点 cur 的贡献 len(cur) - len(link(cur)) 记录到答案中*/

long long cnt_of_different_substr;

void dfs_for_cnt_of_different_substr(int u){
    int lk=node[u].link;
    cnt_of_different_substr+=node[u].len-((lk!=-1)?node[lk].len:0);
    for(auto&& v:node[u].inv_link){
        dfs_for_cnt_of_different_substr(v);
    }
}

void get_cnt_of_different_substr(){
    cnt_of_different_substr=0;
    dfs_for_cnt_of_different_substr(0);
}

/*询问第 k 大子串：type = 0 排名不看重的，type = 1 排名看重的
先处理出每个点 endpos 的 siz，然后通过统计 sam 的转移 dag 的
```

```
每个点下去的不同路径个数，即通过该点的不同子串个数，在 dag 上做 dp 即可，
如果 type = 1 就是每个点初始赋值成 endpos 的 siz 即可，
否则就是初始赋值为 1，由于不算空串，0 节点初始为 0
同样的，type = 0 的情况的 dp[0] 就是本质不同子串个数
luogu P3975*/

char substr_of_kth[MAXLEN];
int ans_len;
long long dp[MAXLEN<<1];
int vis[MAXLEN<<1];

long long get_dp(int u){
    if(vis[u])return dp[u];
    vis[u]=1;
    for(auto&& [c,v]:node[u].nxt){
        dp[u]+=get_dp(v);
    }
    return dp[u];
}

bool find_kth(int u,int k,int type){
    if(k>dp[u])return false;
    if(u!=0&&k<=(type==0?1:node[u].siz))return true;
    if(u!=0)k-=(type==0?1:node[u].siz);
    for(auto&& [c,v]:node[u].nxt){
        if(k>dp[v])k-=dp[v];
        else{
            substr_of_kth[++ans_len]=c;
            return find_kth(v,k,type);
        }
    }
}

void query_kth(int k,int type){
    // 使用前保证 endpos siz 处理好
    ans_len=0;
    for(int i=0;i<sz;++i)vis[i]=0;
    if(type==0){
        dp[0]=0;
        for(int i=1;i<sz;++i)dp[i]=1;
        get_dp(0);
    }else{
        dp[0]=0;
        for(int i=1;i<sz;++i)dp[i]=node[i].siz;
        get_dp(0);
    }
    // 若多次询问请把上面的给预处理处理出来，下面多次询问
    if(find_kth(0,k,type)){
        for(int i=1;i<=ans_len;++i)putchar(substr_of_kth[i]);puts("");
    }else{
        puts("-1");
    }
}

/*字典序最小的循环移位后的串：找到循环移位下的字典序最小的串
方法：可以复制原串 s 成两遍，然后在新串的 sam 上跑转移
```

```
找到长度为 |s| 的字典序最小的即可，代码略，很简单
luogu P1368*/

/*******************************************************************************/

char s[MAXLEN];

int n;

char t[MAXLEN];

int m;

int main(){
    scanf("%s",s+1);
    n=strlen(s+1);
    init();
    rep(i,1,n)extend(s[i]);
    build_pattern_tree();
    dfs(0);
    // printf("%lld\n",ans);

    scanf("%s",t+1);
    m=strlen(t+1);

    printf("first_pos: %d\n",query_first_pos(t,m));

    query_all_pos(t,m);
    puts("all_pos:");
    for(int i=1;i<=idx;++i)printf("%d ",all_pos[i]);puts("");

    printf("occur_cnt: %d\n",query_occur_cnt(t,m));

    get_cnt_of_different_substr();
    printf("cnt_of_different_substr: %lld\n",cnt_of_different_substr);

    query_kth(21,0);
    return 0;
}
```

数学

预处理 1~n 的因子 + 欧拉函数（$O(n \log n)$）

```
int phi[N];
vector<int> dv[N];

void init(){
    for(int i=1;i<N;++i)phi[i]=i;
    for(int i=1;i<N;++i){
        for(int j=i;j<N;j+=i){
            dv[j].push_back(i);
            if(j>i)phi[j]-=phi[i];
        }
```

```
    }
}
```

EXGCD

```
LL __exgcd(LL a,LL b,LL& x,LL& y){
    //矩阵法
    int x1=1,x2=0,x3=0,x4=1;
    while(b){
        LL c=a/b;
        tie(x1,x2,x3,x4,a,b)=make_tuple(x3,x4,x1-c*x3,x2-c*x4,b,a-c*b);
    }
    x=x1,y=x2;
    return a;
}
```

线性同余方程

$ax \equiv b(mod\, m)$

x的解分为无解，有有限个解的情况

记$gcd(a,m)=d$

当$d \nmid b$时，无解（因为d整除左边而不整除右边）

当$d \mid b$时，有解。此时考虑 $ax_0+mk_0=gcd(a,m)$ 的解，左右同除以d乘以b，得

$\frac{abx_0}{d}+\frac{mbk_0}{d}=b$

故最后$x=\frac{bx_0}{d}$

我们要得到所有满足的x，可以考虑先找到最小的，然后算出所有的。

$x_i=x_{min}+i*\frac{m}{d}$

故$x_{min}=(x\, mod\, \frac{m}{d}+\frac{m}{d})\, mod\, \frac{m}{d}$

代码如下

```
#include <bits/stdc++.h>

using namespace std;

#define int long long

int exgcd(int a,int b,int& x,int& y){
    int x1=1,x2=0,x3=0,x4=1;
    while(b){
        int c=a/b;
        tie(a,b,x1,x2,x3,x4)=make_tuple(b,a-c*b,x3,x4,x1-c*x3,x2-c*x4);
    }
    x=x1,y=x2;
    return a;
}

bool liEu(int a, int b, int c, int& x, int& y) {
```

```
    int d=exgcd(a,b,x,y);
    if(c%d!=0)return 0;
    int k =c/d;
    x*=k;
    y*=k;
    return 1;
}

signed main(){
    int a,b,x,y,c;
    int _x[100];
    // cin>>a>>b>>c;
    cin>>a>>b;c=1;//c=1等价求逆元
    int d=exgcd(a,b,x,y);
    if(liEu(a,b,c,x,y)){
        _x[0]=(x%(b/d)+b/d)%(b/d);
        // for(int i=1;i<d;++i){
        //     _x[i]=_x[i-1]+b/d;
        // }
    }
    // for(int i=0;i<d;++i){
    //     cout<<_x[i]<<endl;
    // }
    cout<<_x[0]<<endl;
}
```

CRT

CRT可描述如下，其中模数$n_1, n_2, n_3, \ldots, n_k$互质

$$\begin{cases} x \equiv a_1 \pmod{n_1} \\ x \equiv a_2 \pmod{n_2} \\ \quad \vdots \\ x \equiv a_k \pmod{n_k} \end{cases}$$

定义$n = \Pi_{i=1}^{k} n_i$，则在模n意义下x有唯一解

记$m_i = \frac{n}{n_i}$，m_i^{-1}为m_i在模n_i意义下的逆元

$x = \sum_{i=1}^{k} a_i \times m_i \times m_i^{-1} (mod\, n)$

证明：

> 把解带回去易得到是正确的

代码：

```
#include <bits/stdc++.h>

using namespace std;

vector<int> a,n;

int exgcd(int a,int b,int& x,int& y){
    int x1=1,x2=0,x3=0,x4=1;
    while(b){
        int q=a/b;
```

```
        tie(a,b,x1,x2,x3,x4)=make_tuple(b,a-q*b,x3,x4,x1-x3*q,x2-x4*q);
    }
    x=x1,y=x2;
    return a;
}

int CRT(vector<int> a,vector<int> n){
    int n_pro=1,ret=0;
    for(int i=0;i<a.size();++i){n_pro*=n[i];}
    for(int i=0;i<a.size();++i){
        int m=n_pro/n[i];
        int m_inv,y;
        exgcd(m,n[i],m_inv,y);
        ret=(ret+a[i]*m*m_inv%n_pro)%n_pro;
    }
    return (ret%n_pro+n_pro)%n_pro;
}

int main(){
    a={2,3,2};
    n={3,5,7};
    cout<<CRT(a,n)<<endl;
    return 0;
}
```

拓展：Garner算法，模数不互质（考虑两个方程的时候，写出显式，求exgcd，多个方程就是两两合并）

线性筛 + 求积性函数（例子为 μ，约数个数，约数和）

```
int prime[N],idx;
int vis[N];
int mu[N],smu[N];

void init(){
    mu[1]=1;
    for(int i=2;i<N;++i){
        if(!vis[i])prime[++idx]=i,mu[i]=-1;
        for(int j=1;j<=idx;++j){
            int p=prime[j];
            if(1ll*i*p>=N)break;
            vis[i*p]=1;
            if(i%p==0){
                // p一定为i的最小质因子
                mu[i*p]=0;
                break;
            }
            mu[i*p]=-mu[i];
        }
    }
    for(int i=1;i<N;++i){
        smu[i]=smu[i-1]+mu[i];
    }
}
```

```
int prime[N],idx,vis[N],g[N],f[N];
// g约数个数，f约数和

void init(){
    g[1]=1,f[1]=1;
    for(int i=2;i<N;++i){
        if(!vis[i])prime[++idx]=i,g[i]=2,f[i]=i+1;
        for(int j=1;j<=idx;++j){
            int p=prime[j];
            if(1ll*i*p>=N)break;
            vis[i*p]=1;
            if(i%p==0){
                g[i*p]=2*g[i]-g[i/p];
                f[i*p]=(p+1)*f[i]-p*f[i/p];
                break;
            }
            g[i*p]=g[i]*g[p];
            f[i*p]=f[i]*f[p];
        }
    }
}
```

推导(对于约数个数的部分):

> 当$i \mod p == 0, i = a * p^c$,其中$gcd(a,p) = 1$(互质), $g(i*p) = g(a) * g(p^{c+1})$, $g(i) = g(a) * g(p^c), g(i/p) = g(a) * g(p^{c-1})$, 解得$g(i*p) = 2 * g(i) - g(i/p)$
>
> 当$i \mod p \neq 0, g(i*p) = g(i) * g(p)$
>
> 总体思路就是按照积性函数的性质去求解递推式

值域内预处理后的快速gcd

如果在值域内频繁使用 gcd 可以先预处理一下，预处理时间复杂度$O(值域)$，最后查询$O(1)$

```
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)

const int N=5e3+10;
const int M=1e6+10;
const int sM=1e3+10;
const int dom=1e6;
const int len=1e3;
```

```
const int mod=998244353;

int fac[M][3],pre[sM][sM];
int prime[M],tot,vis[M];

// inline int swap(int& a,int& b){
//  return a^=b^=a^=b;
// }

#define swp(a,b) (a)^=(b)^=(a)^=(b)

void init(){
    fac[1][0]=fac[1][1]=fac[1][2]=1;
    for(int i=2;i<=dom;++i){
        if(!vis[i])prime[++tot]=i,fac[i][0]=fac[i][1]=1,fac[i][2]=i;
        for(int j=1;j<=tot;++j){
            int p=prime[j];
            if(1ll*i*p>dom)break;
            vis[i*p]=1;
            fac[i*p][0]=fac[i][0]*p;
            fac[i*p][1]=fac[i][1];
            fac[i*p][2]=fac[i][2];
            if(fac[i*p][0]>fac[i*p][1])swp(fac[i*p][0],fac[i*p][1]);
            if(fac[i*p][1]>fac[i*p][2])swp(fac[i*p][1],fac[i*p][2]);
            if(i%p==0)break;
        }
    }
    rep(i,0,len)pre[i][0]=pre[0][i]=i;
    rep(i,1,len){
        rep(j,1,i){
            pre[i][j]=pre[j][i]=pre[j][i%j];
        }
    }
}

int gcd(int a,int b){
    int res=1;
    rep(i,0,2){
        int _gcd=1;
        if(fac[a][i]>len){
            if(b%fac[a][i]==0){
                _gcd=fac[a][i];
            }
        }else{
            _gcd=pre[fac[a][i]][b%fac[a][i]];
        }
        b/=_gcd;
        res*=_gcd;
    }
    return res;
}

int a[N],b[N],n;

int main(){
    init();
```

```
    n=read();
    rep(i,1,n)a[i]=read();
    rep(i,1,n)b[i]=read();
    rep(i,1,n){
        int tmp=0;
        for(int j=1,base=i;j<=n;++j,base=1ll*base*i%mod){
            tmp=(1ll*tmp+1ll*base*gcd(a[i],b[j])%mod)%mod;
        }
        printf("%d\n",tmp);
    }
    return 0;
}
```

数论分块

举个例子

$ans=\sum_{i=1}^{n}\sum_{j=1}^{m}a[i]\times\lfloor n/i\rfloor\times\lfloor m/i\rfloor$

sum数组是a的前缀和

j代表的是n/i和m/i的值不发生改变的区间右界，i则是左界

```
long long getans(int n,int m){
    long long res=0;
    for(int i=1,j;i<=min(n,m);i=j+1){
        j=min((n/(n/i)),(m/(m/i)));
        res+=1ll*(smu[j]-smu[i-1])*(n/i)*(m/i);
    }
    return res;
}
```

莫比乌斯

积性函数之积是积性函数，卷积也是积性函数

常见积性函数$1,\varphi,\mu,id$

$\epsilon=\mu*1$

$id=\varphi*1$

$\varphi=\mu*id$

$if\ f=g*1,then\ g=f*\mu$

gcd为k的数对个数

$\sum_{i=1}^{n}\sum_{j=1}^{m}[gcd(i,j)=k]$

i, j同时除以 k 化简后为

$\sum_{d=1}^{min(\lfloor n/k\rfloor,\lfloor m/k\rfloor)}\mu(d)\lfloor n/kd\rfloor\lfloor m/kd\rfloor$

然后可以数论分块算

```
long long getans(int n,int m){
    n/=k,m/=k;
    long long res=0;
    for(int i=1,j;i<=min(n,m);i=j+1){
        j=min((n/(n/i)),(m/(m/i)));
        res+=1ll*(smu[j]-smu[i-1])*(n/i)*(m/i);
    }
    return res;
}
```

sum of lcm

$\sum_{j=1}^{n} lcm(j,n) = \sum_{j=1}^{n} \frac{j*n}{gcd(j,n)} = \sum_{d|n\&d|j\&(n/d,j/d)=1\&j<=n} \frac{n*j}{d}$

$= \sum_{d|n} \sum_{k\in Z_{n/d}^*} n*k$

而模 m 缩系中的数之和由于缩系中任意一个数 x 有 m - x 也在缩系中，所以有数论函数 f 表示缩系和如下

$f(m) = \sum_{k\in Z_m^*} k = \frac{\varphi(m)*m+[m==1]}{2}$

故有原式子

$= n*\sum_{d|n} f(n/d) = n*\sum_{d|n} f(d) = n*\frac{\sum_{d|n} d*\varphi(d)+1}{2} = n*(id\cdot\varphi\circ 1(n)+1)/2$

$id\cdot\varphi\circ 1$这个数论函数是积性函数，所以可以直接筛出来

类似地，sum of gcd $\sum_{j=1}^{n} gcd(j,n) = \sum_{d|n} d\cdot\varphi(n/d) = id\circ\varphi(n)$

```
int n;
int prime[N],idx,vis[N];
long long g[N];

void init(){
    g[1]=1;
    for(int i=2;i<N;++i){
        if(!vis[i])prime[++idx]=i,g[i]=1ll*i*(i-1)+1;
        for(int j=1;j<=idx;++j){
            int p=prime[j];
            if(1ll*i*p>=N)break;
            vis[i*p]=1;
            if(i%p==0){
                g[i*p]=g[i]+(g[i]-g[i/p])*p*p;
                break;
            }
            g[i*p]=g[i]*g[p];
        }
    }
}
```

sum of lcm 两层循环

内外循环次数不等

$\sum_{i=1}^{n}\sum_{j=1}^{m}\operatorname{lcm}(i,j) \quad (n,m\leqslant 10^7)$

$$=\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{i\cdot j}{\gcd(i,j)}=\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{\substack{d|i,d|j,\\ \gcd\left(\frac{i}{d},\frac{j}{d}\right)=1}}\frac{i\cdot j}{d}$$

$$=\sum_{d=1}^{n}d\cdot\sum_{i=1}^{\lfloor\frac{n}{d}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{d}\rfloor}[\gcd(i,j)=1]\cdot i\cdot j$$

记$\operatorname{sum}(n,m)=\sum_{i=1}^{n}\sum_{j=1}^{m}[\gcd(i,j)=1]\cdot i\cdot j=\sum_{d=1}^{n}\mu(d)\cdot d^2\cdot\left(\sum_{i=1}^{\lfloor\frac{n}{d}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{d}\rfloor}i\cdot j\right)$

原式$=\sum_{d=1}^{n}d\cdot sum(\lfloor\frac{n}{d}\rfloor,\lfloor\frac{m}{d}\rfloor)$

```
int prime[N],vis[N],idx,mu[N],g[N];

int n,m;

void init(){
    mu[1]=1;
    for(int i=2;i<N;++i){
        if(!vis[i])prime[++idx]=i,mu[i]=-1;
        for(int j=1;j<=idx;++j){
            int p=prime[j];
            if(1ll*i*p>=N)break;
            vis[i*p]=1;
            if(i%p==0){
                mu[i*p]=0;
                break;
            }
            mu[i*p]=-mu[i];
        }
    }
    for(int i=1;i<N;++i){
        int x=1ll*(mu[i]+MOD)*i%MOD*i%MOD;
        g[i]=((1ll*g[i-1]+x)%MOD+MOD)%MOD;
    }
}

int sum(int a,int b){
    int res=0;
    for(int i=1,j;i<=min(a,b);i=j+1){
        j=min(a/(a/i),b/(b/i));
        int tmp1=1ll*(1+a/i)*(a/i)/2%MOD;
        int tmp2=1ll*(1+b/i)*(b/i)/2%MOD;
        res=(1ll*res+1ll*((g[j]-g[i-1])%MOD+MOD)%MOD*tmp1%MOD*tmp2%MOD)%MOD;
    }
    return res;
}

int getans(){
    int res=0;
    for(int i=1,j;i<=min(n,m);i=j+1){
        j=min(n/(n/i),m/(m/i));
```

```
        res=(1ll*res+1ll*(j-i+1)*(i+j)/2%MOD*sum(n/i,m/i)%MOD)%MOD;
    }
    return res;
}
```

内外循环次数相等

$\sum_{i=1}^{N}\sum_{j=1}^{N}\operatorname{lcm}(i,j) \quad (N\leqslant 10^{10})$

（两层循环 gcd 之和类似化简，也是筛积性函数前缀和）

相似化简得到$\sum_{i=1}^{N} i\cdot\sum_{d|i} d\cdot\varphi(d)$

也就是求积性函数 $id\cdot((id\cdot\varphi)\circ 1)$ 的前缀和，记录这个积性函数为 f

有：

$f(p)=p^3-p^2+p$

$f(p^k)=\frac{p^{(3k+1)}+p^k}{p+1}$

那么直接 min_25 筛出 f 的前缀和即可（min_25板子具体见对应部分）

```
// https://www.51nod.com/Html/Challenge/Problem.html#problemId=1238
int f_p(int _p,int p_k){
    // f(p^k)=(p^(3k+1)+p^k)/(p+1)
    int a=1ll*qpow(p_k,3,mod)*_p%mod;
    int b=1ll*p_k%mod;
    int c=qpow(_p+1,mod-2,mod);
    return ((1ll*a+b)%mod+mod)%mod*c%mod;
}

void init(){
    prime[0]=1;// 特殊处理 p_0=1
    for(int i=2;i<N;++i){
        if(!vis[i]){
            prime[++idx]=i;
            sp0[idx]=(1ll*sp0[idx-1]+1)%mod;
            sp1[idx]=(1ll*sp1[idx-1]+i)%mod;
            sp2[idx]=(1ll*sp2[idx-1]+1ll*i*i%mod)%mod;
            sp3[idx]=(1ll*sp3[idx-1]+1ll*i*i%mod*i%mod)%mod;
        }
        for(int j=1;j<=idx;++j){
            int p=prime[j];
            if(1ll*i*p>=N)break;
            vis[i*p]=1;
            if(i%p==0)break;
        }
    }
}

void get_g(){
    for(long long l=1,r;l<=n;l=r+1){
        r=(n/(n/l));
        w[++tot]=n/l;
        int x=w[tot]%mod;
```

```
        g0[tot]=1ll*x%mod;
        g1[tot]=1ll*x*(x+1)/2%mod;
        g2[tot]=1ll*x*(x+1)%mod*(2*x+1)%mod*qpow(6,mod-2,mod)%mod;
        g3[tot]=1ll*x*x%mod*(x+1)%mod*(x+1)%mod*qpow(4,mod-2,mod)%mod;

        g0[tot]=((g0[tot]-1)%mod+mod)%mod;
        g1[tot]=((g1[tot]-1)%mod+mod)%mod;
        g2[tot]=((g2[tot]-1)%mod+mod)%mod;
        g3[tot]=((g3[tot]-1)%mod+mod)%mod;

        if(w[tot]<=sn)id1[w[tot]]=tot;
        else id2[n/w[tot]]=tot;
    }

    for(int i=1;i<=idx;++i){
        int p=prime[i];
        for(int j=1;j<=tot;++j){
            int x=w[j];
            if(1ll*p*p>x)break;

            int dp_from=x/p;
            int dp_id=dp_from<=sn?id1[dp_from]:id2[n/dp_from];

            g0[j]=((1ll*g0[j]-((1ll*g0[dp_id]-sp0[i-
1])%mod+mod)%mod*1)%mod+mod)%mod;
            g1[j]=((1ll*g1[j]-((1ll*g1[dp_id]-sp1[i-
1])%mod+mod)%mod*p%mod)%mod+mod)%mod;
            g2[j]=((1ll*g2[j]-((1ll*g2[dp_id]-sp2[i-
1])%mod+mod)%mod*p%mod*p%mod)%mod+mod)%mod;
            g3[j]=((1ll*g3[j]-((1ll*g3[dp_id]-sp3[i-
1])%mod+mod)%mod*p%mod*p%mod*p%mod)%mod+mod)%mod;
            // ...
        }
    }
}

int S(long long x,int y){
    int p=prime[y];
    if(p>=x)return 0;
    int id=x<=sn?id1[x]:id2[n/x];
    int ans0=((1ll*g0[id]-sp0[y])%mod+mod)%mod;
    int ans1=((1ll*g1[id]-sp1[y])%mod+mod)%mod;
    int ans2=((1ll*g2[id]-sp2[y])%mod+mod)%mod;
    int ans3=((1ll*g3[id]-sp3[y])%mod+mod)%mod;

    int ans=((((1ll*ans3-ans2)%mod+mod)%mod+ans1)%mod+mod)%mod;// f(p)=p^3-p^2+p
    for(int i=y+1;i<=idx;++i){
        int _p=prime[i];
        if(1ll*_p*_p>x)break;
        long long p_k=_p;
        for(int e=1;p_k<=x;++e,p_k*=_p){
            ans=(1ll*ans+1ll*f_p(_p%mod,p_k%mod)*(S(x/p_k,i)+(e!=1))%mod)%mod;
        }
    }
    return ans;
}
```

```
signed main(){
    init();
    n=read();
    sn=sqrt(n);
    get_g();
    printf("%d\n",(S(n,0)+1)%mod);
}
```

逆元

预处理

```
inv[1]=1;
for(int i=2;i<=n;++i){
    inv[i]=((-inv[p%i]*(p/i))%p+p)%p;
}
```

原理

> $p=i\times a+b$
>
> $i^{-1}=p-a\times b^{-1}(\mod p)$

直接算

p是质数才能按下面这种方法算，费马小定理

```
inv[n]=qpow(n,p-2,p);
```

一般数模意义下的组合数($O(n^2)$)

```
void init(){
    c[0][0]=1;
    for(int i=1;i<N;++i){
        c[i][0]=1;
        for(int j=1;j<=i;++j){
            c[i][j]=(c[i-1][j-1]+c[i-1][j])%mod;
        }
    }
}
```

素数模意义下的组合数

```
const int MAXN=1e6+2;
const int mod=1e9+7;

int n,m,fac[MAXN],inv[MAXN];

int qpow(int a,int b){
    int ret=1;
    for(;b;b>>=1,a=1ll*a*a%mod)if(b&1){
        ret=1ll*ret*a%mod;
```

```
    }
    return ret%mod;
}

void pre(){
    fac[0]=1;
    for(int i=1;i<=MAXN-1;++i)fac[i]=1ll*fac[i-1]*i%mod;
    inv[MAXN-1]=qpow(fac[MAXN-1],mod-2);
    for(int i=MAXN-2;i>=0;--i)inv[i]=1ll*inv[i+1]*(i+1)%mod;
}

int binom(int n,int k){
    if(k<0||k>n)return 0;
    return 1ll*fac[n]*inv[n-k]%mod*inv[k]%mod;
}
```

二次剩余

$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod p$，为1则是a是p的二次剩余，-1则不是

勒让德符号是完全积性函数，其中分子在同剩余类中的勒让德符号值相等

$\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2}$

$\left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8}$

模素数的最简二次同余式解的解个数必定为 0 或 2，手算方法见教材，模素数幂的（或模合数）二次同余式解的分析见教材

Cipolla 算法

用来处理模素数的最简二次同余式的解

```
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)

int n,p,w2;

struct com_t{
    int x,y;
    friend com_t operator*(const com_t& a,const com_t& b){
        int resx=(1ll*a.x*b.x%p+1ll*a.y*b.y%p*w2%p)%p;
        int resy=(1ll*a.x*b.y%p+1ll*b.x*a.y%p)%p;
```

```
        return {resx,resy};
    }

    friend com_t operator%(const com_t& a,int b){
        int resx=a.x%p;
        int resy=a.y%p;
        return {resx,resy};
    }
};

int qpow(int a,int b){
    int res=1,base=a%p;
    for(;b;b>>=1,base=1ll*base*base%p)if(b&1){
        res=(1ll*res*base)%p;
    }
    return res%p;
}

int iqpow(com_t a,int b){
    com_t res={1,0},base=a%p;
    for(;b;b>>=1,base=base*base)if(b&1){
        res=res*base;
    }
    return res.x;
}

int cipolla(){
    n%=p;
    if(n==0)return 0;
    if(qpow(n,(p-1)/2)==p-1)return -1;
    int a;
    do{
        a=rand()%p;
        w2=((1ll*a*a-n)%p+p)%p;
    }while(qpow(w2,(p-1)/2)!=p-1);
    return iqpow({a,1},(p+1)/2);
}

void solve(){
    n=read(),p=read();
    int ans=cipolla();
    if(ans==-1){
        puts("Hola!");
    }else{
        int _ans=p-ans;
        if(ans>_ans)swap(ans,_ans);
        if(ans)printf("%d %d\n",ans,_ans);
        else puts("0");
    }
}

int main(){
    srand((unsigned)time(nullptr));
    int t=read();
    while(t--)solve();
    return 0;
```

```
}
```

原根

若模数为 m，且原根只讨论 m 缩系下的数，有以下定义，定理或性质：

阶为$\varphi(m)$的数为 m 的原根

$a, a^2, a^3, \ldots, a^{\delta(a)}$两两互不同余

若$a^n \equiv 1 \pmod m$成立，则$\delta(a)|n$

若$(a,m)=1,(b,m)=1$，则$\delta(ab)=\delta(a)\delta(b) \Leftrightarrow (\delta(a),\delta(b))=1$

$\delta(a^k)=\frac{\delta(a)}{(\delta(a),k)}$，由该式子很容易得到原根个数的判定，见后

原根判定定理：若$m>=3,(g,m)=1$，则 g 为 m 的原根等价于$\forall p|m, g^{\frac{\varphi(m)}{p}} \not\equiv 1 \pmod m$，2 的原根就是1

原根个数定理：一个数 m 有原根则原根个数为 $\varphi(\varphi(m))$

原根存在定理：m 有原根当且仅当 m=2,4,p^α,$2p^\alpha$，p 为奇素数

求原根代码如下：

```
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)

const int N=1e6+10;
const int sN=1e3+10;
const int dom=1e6;
const int sdom=1e3;

int n,d;

int prime[N],vis[N],idx,phi[N],is_have_root[N];

// 快速gcd
int fc[N][3],pre[sN][sN];

void init(){
    fc[1][0]=fc[1][1]=fc[1][2]=1;
    for(int i=2;i<=dom;++i){
        if(!vis[i])prime[++idx]=i,phi[i]=i-1,fc[i][0]=fc[i][1]=1,fc[i][2]=i;
        for(int j=1;j<=idx;++j){
```

```
            int p=prime[j];
            if(1ll*p*i>dom)break;
            vis[i*p]=1;

            fc[i*p][0]=fc[i][0]*p;
            fc[i*p][1]=fc[i][1];
            fc[i*p][2]=fc[i][2];
            if(fc[i*p][0]>fc[i*p][1])swap(fc[i*p][0],fc[i*p][1]);
            if(fc[i*p][1]>fc[i*p][2])swap(fc[i*p][1],fc[i*p][2]);

            if(i%p==0){
                phi[i*p]=phi[i]*p;
                break;
            }
            phi[i*p]=phi[i]*phi[p];
        }
    }

    for(int i=0;i<=sdom;++i)pre[i][0]=pre[0][i]=i;
    for(int i=1;i<=sdom;++i){
        for(int j=1;j<=i;++j){
            pre[i][j]=pre[j][i]=pre[j][i%j];
        }
    }

    // 2,4,p^k,2*p^k有原根(p为奇素数)
    is_have_root[2]=is_have_root[4]=1;
    for(int i=2;i<=idx;++i){
        int p=prime[i];
        for(long long j=p;j<N;j*=p)is_have_root[j]=1;
        for(long long j=2*p;j<N;j*=p)is_have_root[j]=1;
    }
}

// int gcd(int a,int b){
//  if(!b)return a;
//  return gcd(b,a%b);
// }

// 值域内频繁使用 gcd, 快速 gcd 更快
int gcd(int a,int b){
    int res=1;
    for(int i=0;i<=2;++i){
        int _gcd=1;
        if(fc[a][i]>sdom){
            if(b%fc[a][i]==0){
                _gcd=fc[a][i];
            }
        }else{
            _gcd=pre[fc[a][i]][b%fc[a][i]];
        }
        b/=_gcd;
        res*=_gcd;
    }
    return res;
}
```

```cpp
vector<pair<int,int>> bk(int x){
    vector<pair<int,int>> res;
    for(int i=1;i<=idx;++i){
        int p=prime[i];
        if(1ll*p*p>x)break;
        if(x%p==0){
            int cnt=0;
            while(x%p==0)x/=p,++cnt;
            res.push_back({p,cnt});
        }
    }
    if(x>1)res.push_back({x,1});
    return res;
}

int qpow(int a,int b,int mod){
    int res=1,base=a%mod;
    for(;b;b>>=1,base=1ll*base*base%mod)if(b&1){
        res=1ll*res*base%mod;
    }
    return res;
}

bool is_root(int g,int x,const vector<pair<int,int>>& fac){
    // 保证 g 是 x 缩系中的数再用
    // if(gcd(g,x)!=1)return false;
    for(auto&& [p,cnt]:fac){
        if(qpow(g,phi[x]/p,x)==1)return false;
    }
    return true;
}

int get_min_root(int x,const vector<pair<int,int>>& fac){
    for(int i=1;i<x;++i)if(gcd(i,x)==1){
        if(is_root(i,x,fac))return i;
    }
    // 正常判定是否有原根后使用不会走下面return
    return 0;
}

vector<int> get_root(int x,const vector<pair<int,int>>& fac){
    vector<int> res,tag(x+1);
    int min_g=get_min_root(x,fac);
    for(int i=1,g=min_g;i<=phi[x];++i,g=1ll*g*min_g%x)if(gcd(i,phi[x])==1){
        // res.push_back(g);
        tag[g]=1;
    }
    // 用桶排序快点
    // sort(res.begin(),res.end());
    for(int i=1;i<x;++i)if(tag[i]){
        res.push_back(i);
    }
    return res;
}
```

```
void solve(){
    n=read(),d=read();
    if(is_have_root[n]){
        auto fac=bk(phi[n]);
        auto root=get_root(n,fac);
        printf("%d\n",root.size());
        for(int i=d;i<=root.size();i+=d){
            printf("%d ",root[i-1]);
        }puts("");
    }else{
        puts("0");
        puts("");
    }
}

int main(){
    init();
    int t=read();
    while(t--)solve();
    return 0;
}
```

质因子分解

Pollard Rho

```
#include <bits/stdc++.h>

using namespace std;

#define LL long long

const int MAXN=100;

LL gcd(LL a,LL b){
    if(!b)return a;
    return gcd(b,a%b);
}

LL qpow(LL a,LL b,LL mod){
    LL res=1,base=a%mod;
    for(;b;b>>=1,base=(__int128_t)base*base%mod)if(b&1){
        res=(__int128_t)res*base%mod;
    }
    return res;
}

LL MR(LL p){
    if(p<2)return 0;
    if(p==2)return 1;
    if(p==3)return 1;
    LL d=p-1,r=0;
    while(!(d&1))d>>=1,++r;
    for(LL k=1;k<=10;++k){
        LL a=rand()%(p-2)+2;
```

```
        LL x=qpow(a,d,p);
        if(x==1||x==p-1)continue;
        for(int i=1;i<=r-1;++i){
            x=(__int128_t)x*x%p;
            if(x==p-1)break;
        }
        if(x!=p-1)return 0;
    }
    return 1;
}

LL PR(LL x){
    LL s=0,t=0;
    LL c=1ll*rand()%(x-1)+1;
    int step=0,goal=1;
    LL val=1;
    for(goal=1;;goal<<=1,s=t,val=1){
        for(step=1;step<=goal;++step){
            t=((__int128_t)t*t+c)%x;
            val=(__int128_t)val*abs(t-s)%x;
            if((step%127)==0){
                LL d=gcd(val,x);
                if(d>1)return d;
            }
        }
        LL d=gcd(val,x);
        if(d>1)return d;
    }
}

// P4718

// int t;
// LL maxfac,n;

// void fac(LL x){
//     if(x<=maxfac||x<2)return;
//     if(MR(x)){
//         maxfac=max(maxfac,x);
//         return;
//     }
//     LL p=x;
//     while(p>=x)p=PR(x);
//     while((x%p)==0)x/=p;
//     fac(x),fac(p);
// }

// int main(){
//     cin>>t;
//     for(int e=1;e<=t;++e){
//         srand((unsigned)time(nullptr));
//         maxfac=0;
//         cin>>n;
//         fac(n);
//         if(maxfac==n)cout<<"Prime"<<'\n';
//         else cout<<maxfac<<'\n';
```

```
//     }
//     return 0;
// }

LL fac[MAXN],cnt[MAXN],tot,len,ufac[MAXN];

void divide(LL n){
    if(n<2)return;
    if(MR(n)){
        fac[++tot]=n;
        return;
    }
    LL d=n;
    while(d>=n)d=PR(n);
    divide(d);
    divide(n/d);
}

void brk(int n){
    memset(cnt,0,sizeof cnt);
    tot=0;
    len=0;
    divide(n);
    sort(fac+1,fac+tot+1);
    for(int i=1;i<=tot;++i){
        if(fac[i]!=fac[i-1])ufac[++len]=fac[i];
        ++cnt[len];
    }
}

int main(){
    srand((unsigned)time(nullptr));
    brk(100000000);
    for(int i=1;i<=len;++i){
        cout<<ufac[i]<<' '<<cnt[i]<<'\n';
    }
    return 0;
}
```

min_25 筛

如果要求积性函数的前缀和$\sum_{i=1}^{n} f(i)$，n为1e10范围（min_25 筛时间复杂度 $O(n^{(3/4)}/log(n))$）

其中给出 $f(p^k)$为某多项式，比如说欧拉函数$f(p^k)=p^k-p^{k-1}$

那么对应的 $f(p)$ 也是一个p的多项式，我们需要这个来修改我们的板子，具体细节如下：

（如果$f(p)$不能很好表示成 p 的多项式，且 $f(p^k)$，请不要使用 min_25，则去考虑杜教筛，但是大多数情况都是满足的）

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

// https://www.51nod.com/Html/Challenge/Problem.html#problemId=1239
```

```cpp
long long read(){
    long long res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)

// 比如求欧拉函数的前缀和
// f(p)=p-1,我们需要阶为1和0的g数组做dp
// f(p^k)=p^k-p^(k-1)

const int mod=1e9+7;
const int N=1e5+10;

int qpow(int a,int b,int mod){
    int res=1,base=a%mod;
    for(;b;b>>=1,base=1ll*base*base%mod)if(b&1){
        res=1ll*res*base%mod;
    }
    return res;
}

// g 是 min_25 的 dp 数组
// sp 是 前 i 的素数幂前缀和
// g 和 sp 根据数论函数取素数的式子 f_p 的多项式的度数取
// 对不同阶素数幂分别求贡献最后累计在一起
// 比如欧拉函数的 f_p 就是 p-1 就是最高阶为1的多项式
long long n;
int sn,g0[N*3],g1[N*3],sp0[N],sp1[N];
// int g2[N*3],sp2[N];
// ......

// 数论分块
// dp 的 过程需要数论分块来知道转移，我需要知道 n/p 所在块的索引
// dp 数组 g 都是存在数论分块完成后的索引 id 的位置上的
// 每个索引对应的块的值都是 w[id]
long long w[N*3];
int tot;

int prime[N],vis[N],idx,id1[N*3],id2[N*3];

// int f_p(int p,int k){
//  return ((1ll*qpow(p,k,mod)-qpow(p,k-1,mod))%mod+mod)%mod;
// }

int f_p(int p_k_1,int p_k){
    return ((1ll*p_k-1ll*p_k_1)%mod+mod)%mod;
}

void init(){
    prime[0]=1;// 特殊处理 p_0=1
    for(int i=2;i<N;++i){
```

```
        if(!vis[i]){
            prime[++idx]=i;
            sp0[idx]=(1ll*sp0[idx-1]+1)%mod;
            sp1[idx]=(1ll*sp1[idx-1]+i)%mod;
            // sp2[idx]=(1ll*sp2[idx-1]+1ll*i*i%mod)%mod;
            // ...
        }
        for(int j=1;j<=idx;++j){
            int p=prime[j];
            if(1ll*i*p>=N)break;
            vis[i*p]=1;
            if(i%p==0)break;
        }
    }
}

void get_g(){
    for(long long l=1,r;l<=n;l=r+1){
        r=(n/(n/l));
        w[++tot]=n/l;
        int x=w[tot]%mod;
        //初始化为1~x中所有>p_0即>1的数之幂和
        g0[tot]=1ll*x%mod;
        g1[tot]=1ll*x*(x+1)/2%mod;
        // g2[tot]=1ll*x*(x+1)%mod*(2*x+1)%mod*qpow(6,mod-2,mod)%mod;
        // ...

        g0[tot]=((g0[tot]-1)%mod+mod)%mod;
        g1[tot]=((g1[tot]-1)%mod+mod)%mod;
        // g2[tot]=((g2[tot]-1)%mod+mod)%mod;
        // ...

        // 存放数论分块的索引
        // 值为x<=sn的在第id1[x]块
        // 值为x>sn的在第id2[n/x]块
        if(w[tot]<=sn)id1[w[tot]]=tot;
        else id2[n/w[tot]]=tot;
    }

    for(int i=1;i<=idx;++i){
        int p=prime[i];
        for(int j=1;j<=tot;++j){
            int x=w[j];
            if(1ll*p*p>x)break;

            int dp_from=x/p;
            int dp_id=dp_from<=sn?id1[dp_from]:id2[n/dp_from];

            g0[j]=((1ll*g0[j]-((1ll*g0[dp_id]-sp0[i-
1])%mod+mod)%mod*1)%mod+mod)%mod;
            g1[j]=((1ll*g1[j]-((1ll*g1[dp_id]-sp1[i-
1])%mod+mod)%mod*p%mod)%mod+mod)%mod;
            // g2[j]=((1ll*g2[j]-((1ll*g2[dp_id]-sp2[i-
1])%mod+mod)%mod*p%mod*p%mod)%mod+mod)%mod;
            // ...
        }
```

```
    }
}

int S(long long x,int y){
    int p=prime[y];
    if(p>=x)return 0;
    int id=x<=sn?id1[x]:id2[n/x];
    int ans0=((1ll*g0[id]-sp0[y])%mod+mod)%mod;
    int ans1=((1ll*g1[id]-sp1[y])%mod+mod)%mod;
    // int ans2=((1ll*g2[id]-sp2[y])%mod+mod)%mod;
    // ...

    // 最后按照 f_p 的多项式系数把贡献拼起来
    // 这里 f_p=p-1
    int ans=((1ll*ans1-ans0)%mod+mod)%mod;
    for(int i=y+1;i<=idx;++i){
        int _p=prime[i];
        if(1ll*_p*_p>x)break;
        long long p_k=_p;
        long long p_k_1=1;
        for(int e=1;p_k<=x;++e,p_k*=_p,p_k_1*=_p){
            ans=(1ll*ans+1ll*f_p(p_k_1%mod,p_k%mod)*(S(x/p_k,i)+(e!=1))%mod)%mod;
        }
    }
    return ans;
}

signed main(){
    init();
    n=read();
    sn=sqrt(n);
    get_g();
    // 假设 phi[1]=1 的话，得多记录1的答案
    printf("%d\n",(S(n,0)+1)%mod);
}
```

多项式乘法

FFT

```
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)
```

```
const int N=(1e6+4)*4;
const double PI=acos(-1);
typedef complex<double> COMP;

COMP a[N],b[N],ans[N];

int rev[N];

void init(int lim){
    rep(i,0,lim-1)rev[i]=(i&1)*(lim)+rev[i>>1]>>1;
}

// void change(COMP x[],int len){
//     for(int i=0;i<len;++i){
//         rev[i]=rev[i>>1]>>1;
//         if(i&1){
//             rev[i]|=len>>1;
//         }
//     }
//     for(int i=0;i<len;++i){
//         if(i<rev[i]){
//             swap(x[i],x[rev[i]]);
//         }
//     }
// }

void fft(COMP x[],int len,int on){
    // change(x,len);
    rep(i,0,len-1)if(i<rev[i])swap(x[i],x[rev[i]]);
    for(int h=2;h<=len;h<<=1){
        COMP wn(cos(2*PI/h),sin(2*PI*on/h));
        for(int i=0;i<len;i+=h){
            COMP w(1,0);
            for(int k=i;k<i+h/2;++k){
                COMP u=x[k];
                COMP t=w*x[k+h/2];
                x[k]=u+t;
                x[k+h/2]=u-t;
                w=w*wn;
            }
        }
    }
    if(on==-1){
        for(int i=0;i<len;++i){
            x[i]/=len;
        }
    }
}

int n,m,len;

int _ans[N];

int main(){
    n=read(),m=read();
    rep(i,0,n){
```

```
        a[i]={read(),0};
    }
    rep(i,0,m){
        b[i]={read(),0};
    }
    int mx=max(n,m);
    len=1;
    while(len<(mx+1)*2)len<<=1;
    init(len);

    fft(a,len,1);
    fft(b,len,1);
    rep(i,0,len-1)ans[i]=a[i]*b[i];
    fft(ans,len,-1);
    rep(i,0,len)_ans[i]=(int)(ans[i].real()+0.5);
    rep(i,0,n+m){
        printf("%d ",_ans[i]);
    }
    return 0;
}
```

NTT

```
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)

const int N=(1e6+10)*4;

const int P=998244353;

int inv3;

int qpow(int a,int b,int mod){
    int res=1,base=a%mod;
    for(;b;b>>=1,base=1ll*base*base%mod)if(b&1){
        res=1ll*res*base%mod;
    }
    return res;
}

int rev[N];

void init(int lim){
```

```
    rep(i,0,lim-1)rev[i]=(i&1)*(lim>>1)+(rev[i>>1]>>1);
}

void ntt(int x[],int len,int on){
    rep(i,0,len-1)if(rev[i]<i)swap(x[i],x[rev[i]]);
    for(int h=2;h<=len;h<<=1){
        int gn=qpow(on==1?3:inv3,(P-1)/h,P);
        for(int i=0;i<len;i+=h){
            int g=1;
            for(int k=i;k<i+h/2;++k){
                int u=x[k]%P;
                int t=1ll*g*x[k+h/2]%P;
                x[k]=(1ll*u+t)%P;
                x[k+h/2]=((1ll*u-t)%P+P)%P;
                g=1ll*g*gn%P;
            }
        }
    }
    if(on==-1){
        int inv=qpow(len,P-2,P);
        rep(i,0,len-1)x[i]=1ll*x[i]*inv%P;
    }
}

int n,m,len;

int a[N],b[N],ans[N];

int main(){
    inv3=qpow(3,P-2,P);
    n=read(),m=read();
    rep(i,0,n)a[i]=read();
    rep(i,0,m)b[i]=read();
    int mx=max(n,m);
    len=1;
    while(len<2*(mx+1))len<<=1;
    init(len);

    ntt(a,len,1);
    ntt(b,len,1);
    rep(i,0,len-1)ans[i]=1ll*a[i]*b[i]%P;
    ntt(ans,len,-1);
    rep(i,0,n+m){
        printf("%d ",ans[i]);
    }
    return 0;
}
```

快速求模(Barrett Reduction)

仅限于模数固定的情况，全局只有一个模数的情况，才会优化很多时间

```
struct Mod{
    typedef long long LL;
    LL m,p;
    void init(int pp){m=((__int128)1<<64)/pp;p=pp;}
    LL operator ()(LL x){
        return x-((__int128(x)*m)>>64)*p;
    }
};

Mod mod;
```

数据结构

Splay

1. 插入 x 数
2. 删除 x 数(若有多个相同的数，应只删除一个)
3. 查询 x 数的排名(排名定义为比当前数小的数的个数 $+1$)
4. 查询排名为 x 的数
5. 求 x 的前驱(前驱定义为小于 x，且最大的数)
6. 求 x 的后继(后继定义为大于 x，且最小的数)

```
#include <iostream>

using namespace std;
const int MAXN=1e5+10;

int read(){
    int res=0,sign=1;char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-')sign=-sign;
    for(;ch>='0'&&ch<='9';ch=getchar())res=(res<<3)+(res<<1)+(ch^'0');
    return res*sign;
}

int rt,tot,fa[MAXN],ch[MAXN][2],val[MAXN],cnt[MAXN],sz[MAXN];

void maintain(int x){sz[x]=sz[ch[x][0]]+sz[ch[x][1]]+cnt[x];}

bool get(int x){return x==ch[fa[x]][1];}

void clear(int x){fa[x]=ch[x][0]=ch[x][1]=val[x]=cnt[x]=sz[x]=0;}

void rotate(int x){
    int y=fa[x],z=fa[y],chk=get(x);

    ch[y][chk]=ch[x][chk^1];
    if(ch[x][chk^1])fa[ch[x][chk^1]]=y;

    ch[x][chk^1]=y;
    fa[y]=x;

    if(z)ch[z][y==ch[z][1]]=x;
```

```
    fa[x]=z;

    maintain(y);
    maintain(x);
}

void splay(int x){
    for(int f=fa[x];f=fa[x],f;rotate(x)){
        if(fa[f])rotate(get(x)==get(f)?f:x);
    }
    rt=x;
}

void ins(int k){
    if(!rt){
        val[++tot]=k;
        cnt[tot]++;
        rt=tot;
        maintain(rt);
        return;
    }
    int cur=rt,f=0;
    while(1){
        if(val[cur]==k){
            cnt[cur]++;
            maintain(cur);
            maintain(f);
            splay(cur);
            break;
        }
        f=cur;
        cur=ch[cur][val[cur]<k];
        if(!cur){
            val[++tot]=k;
            cnt[tot]++;
            fa[tot]=f;
            ch[f][val[f]<k]=tot;
            maintain(tot);
            maintain(f);
            splay(tot);
            break;
        }
    }
}

int rk(int k){
    int res=0,cur=rt;
    while(1){
        if(val[cur]>k){
            cur=ch[cur][0];
        }else{
            res+=sz[ch[cur][0]];
            if(!cur)return res+1;
            if(val[cur]==k){
                splay(cur);
                return res+1;
```

```
            }
            res+=cnt[cur];
            cur=ch[cur][1];
        }
    }
}

int kth(int k){
    int cur=rt;
    while(1){
        if(ch[cur][0]&&k<=sz[ch[cur][0]]){
            cur=ch[cur][0];
        }else{
            k-=sz[ch[cur][0]]+cnt[cur];
            if(k<=0){
                splay(cur);
                return val[cur];
            }
            cur=ch[cur][1];
        }
    }
}

int pre(){
    int cur=ch[rt][0];
    if(!cur)return cur;
    while(ch[cur][1])cur=ch[cur][1];
    splay(cur);
    return cur;
}

int nxt(){
    int cur=ch[rt][1];
    if(!cur)return cur;
    while(ch[cur][0])cur=ch[cur][0];
    splay(cur);
    return cur;
}

void del(int k){
    rk(k);
    if(cnt[rt]>1){
        cnt[rt]--;
        maintain(rt);
        return;
    }
    if(!ch[rt][0]&&!ch[rt][1]){
        clear(rt);
        rt=0;
        return;
    }
    if(!ch[rt][0]){
        int cur=rt;
        rt=ch[rt][1];
        fa[rt]=0;
        clear(cur);
```

```
        return;
    }
    if(!ch[rt][1]){
        int cur=rt;
        rt=ch[rt][0];
        fa[rt]=0;
        clear(cur);
        return;
    }
    int cur=rt,x=pre();
    fa[ch[cur][1]]=x;
    ch[x][1]=ch[cur][1];
    clear(cur);
    maintain(rt);
}

int getpre(int x){
    ins(x);
    int res=pre();
    del(x);
    return val[res];
}

int getnxt(int x){
    ins(x);
    int res=nxt();
    del(x);
    return val[res];
}

int n;

int main(){
    n=read();
    for(int i=1;i<=n;++i){
        int op=read();
        if(op==1){
            int x=read();
            ins(x);
        }else if(op==2){
            int x=read();
            del(x);
        }else if(op==3){
            int x=read();
            printf("%d\n",rk(x));
        }else if(op==4){
            int x=read();
            printf("%d\n",kth(x));
        }else if(op==5){
            int x=read();
            printf("%d\n",getpre(x));
        }else if(op==6){
            int x=read();
            printf("%d\n",getnxt(x));
        }
    }
```

```
    return 0;
}
```

再给出一个 splay 大小为变长的模板

```
#include <bits/stdc++.h>

using namespace std;
const int MAXN=1e5+10;

int read(){
    int res=0,sign=1;char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-')sign=-sign;
    for(;ch>='0'&&ch<='9';ch=getchar())res=(res<<3)+(res<<1)+(ch^'0');
    return res*sign;
}

struct splay_t{
    int rt,tot;
    vector<int> fa,val,cnt,sz,ch[2];

    splay_t(){
        fa.push_back(0);
        val.push_back(0);
        cnt.push_back(0);
        sz.push_back(0);
        ch[0].push_back(0);
        ch[1].push_back(0);
    }

    void maintain(int x){sz[x]=sz[ch[0][x]]+sz[ch[1][x]]+cnt[x];}

    bool get(int x){return x==ch[1][fa[x]];}

    void clear(int x){fa[x]=ch[0][x]=ch[1][x]=val[x]=cnt[x]=sz[x]=0;}

    void rotate(int x){
        int y=fa[x],z=fa[y],chk=get(x);

        ch[chk][y]=ch[chk^1][x];
        if(ch[chk^1][x])fa[ch[chk^1][x]]=y;

        ch[chk^1][x]=y;
        fa[y]=x;

        if(z)ch[y==ch[1][z]][z]=x;
        fa[x]=z;

        maintain(y);
        maintain(x);
    }

    void splay(int x){
        for(int f=fa[x];f=fa[x],f;rotate(x)){
            if(fa[f])rotate(get(x)==get(f)?f:x);
```

```
        }
        rt=x;
    }

    void ins(int k){
        if(!rt){
            ++tot;
            fa.push_back(0);
            val.push_back(k);
            cnt.push_back(1);
            sz.push_back(0);
            ch[0].push_back(0);
            ch[1].push_back(0);
            rt=tot;
            maintain(rt);
            return;
        }
        int cur=rt,f=0;
        while(1){
            if(val[cur]==k){
                cnt[cur]++;
                maintain(cur);
                maintain(f);
                splay(cur);
                break;
            }
            f=cur;
            cur=ch[val[cur]<k][cur];
            if(!cur){
                ++tot;
                fa.push_back(f);
                val.push_back(k);
                cnt.push_back(1);
                sz.push_back(0);
                ch[0].push_back(0);
                ch[1].push_back(0);
                ch[val[f]<k][f]=tot;
                maintain(tot);
                maintain(f);
                splay(tot);
                break;
            }
        }
    }

    int rk(int k){
        int res=0,cur=rt;
        while(1){
            if(val[cur]>k){
                cur=ch[0][cur];
            }else{
                res+=sz[ch[0][cur]];
                if(!cur)return res+1;
                if(val[cur]==k){
                    splay(cur);
                    return res+1;
```

```
            }
            res+=cnt[cur];
            cur=ch[1][cur];
        }
    }
}

int kth(int k){
    int cur=rt;
    while(1){
        if(ch[0][cur]&&k<=sz[ch[0][cur]]){
            cur=ch[0][cur];
        }else{
            k-=sz[ch[0][cur]]+cnt[cur];
            if(k<=0){
                splay(cur);
                return val[cur];
            }
            cur=ch[1][cur];
        }
    }
}

int pre(){
    int cur=ch[0][rt];
    if(!cur)return cur;
    while(ch[1][cur])cur=ch[1][cur];
    splay(cur);
    return cur;
}

int nxt(){
    int cur=ch[1][rt];
    if(!cur)return cur;
    while(ch[0][cur])cur=ch[0][cur];
    splay(cur);
    return cur;
}

void del(int k){
    rk(k);
    if(cnt[rt]>1){
        cnt[rt]--;
        maintain(rt);
        return;
    }
    if(!ch[0][rt]&&!ch[1][rt]){
        clear(rt);
        rt=0;
        return;
    }
    if(!ch[0][rt]){
        int cur=rt;
        rt=ch[1][rt];
        fa[rt]=0;
        clear(cur);
```

```
            return;
        }
        if(!ch[1][rt]){
            int cur=rt;
            rt=ch[0][rt];
            fa[rt]=0;
            clear(cur);
            return;
        }
        int cur=rt,x=pre();
        fa[ch[1][cur]]=x;
        ch[1][x]=ch[1][cur];
        clear(cur);
        maintain(rt);
    }

    int getpre(int x){
        ins(x);
        int res=pre();
        del(x);
        return val[res];
    }

    int getnxt(int x){
        ins(x);
        int res=nxt();
        del(x);
        return val[res];
    }
}tree;

int n;

int main(){
    n=read();
    for(int i=1;i<=n;++i){
        int op=read();
        if(op==1){
            int x=read();
            tree.ins(x);
        }else if(op==2){
            int x=read();
            tree.del(x);
        }else if(op==3){
            int x=read();
            printf("%d\n",tree.rk(x));
        }else if(op==4){
            int x=read();
            printf("%d\n",tree.kth(x));
        }else if(op==5){
            int x=read();
            printf("%d\n",tree.getpre(x));
        }else if(op==6){
            int x=read();
            printf("%d\n",tree.getnxt(x));
        }
```

```
    }
    return 0;
}
```

ST表

```
const int MAXN=2e6+5;
const int LOGN=21;
int f[MAXN][LOGN],_log_2[MAXN],n,m;

void pre(){
    //对数取整
    _log_2[1]=0;
    _log_2[2]=1;
    for(int i=3;i<MAXN;++i){
        _log_2[i]=_log_2[i/2]+1;
    }
}

void _pre(){
    for(int k=1;k<=LOGN;++k){
        for(int i=1;i+(1<<k)-1<=n;++i){
            f[i][k]=max(f[i][k-1],f[i+(1<<(k-1))][k-1]);
        }
    }
}

int getmax(int l,int r){
    int ret;
    int s=_log_2[r-l+1];
    ret=max(f[l][s],f[r-(1<<s)+1][s]);
    return ret;
}

int main(){
    n=read(),m=read();
    for(int i=1;i<=n;++i)f[i][0]=read();
    pre();
    _pre();
    for(int i=1;i<=m;++i){
        int l,r;l=read(),r=read();
        printf("%d\n",getmax(l,r));
    }
    return 0;
}
```

线段树染色

贴报纸问题

```
#include <iostream>
#include <cstring>

using namespace std;
```

```
const int MAXN=1e5+2;
const int INF=0x7fffffff;

struct node_t{
    int ls,rs;
    int tag;
}d[MAXN<<5];

int rt,cnt,vis[MAXN];

#define MID int m=s+((t-s)>>1)
#define lson d[p].ls
#define rson d[p].rs

inline void pushdown(int p,int s,int t){
    MID;
    if(s!=t&&d[p].tag){
        if(!lson)lson=++cnt;
        if(!rson)rson=++cnt;
        d[lson].tag=d[p].tag,d[rson].tag=d[p].tag;
        d[p].tag=0;
    }
}

void modify(int& p,int s,int t,int l,int r,int c){
    if(!p)p=++cnt;
    if(l<=s&&t<=r){
        d[p].tag=c;
        return;
    }
    MID;
    pushdown(p,s,t);
    if(l<=m)modify(lson,s,m,l,r,c);
    if(r>m)modify(rson,m+1,t,l,r,c);
}

int query(int p,int s,int t,int l,int r){
    if(!p)return 0;
    if(d[p].tag){
        if(vis[d[p].tag])return 0;
        else return vis[d[p].tag]=1;
    }
    MID;
    pushdown(p,s,t);
    int sum=0;
    if(l<=m)sum+=query(lson,s,m,l,r);
    if(r>m)sum+=query(rson,m+1,t,l,r);
    return sum;
}

int n;

void solve(){
    rt=cnt=0;
    memset(d,0,sizeof d);
```

```
    memset(vis,0,sizeof vis);
    cin>>n;
    for(int i=1,l,r;i<=n;++i){
        cin>>l>>r;
        modify(rt,1,INF,l,r,i);
    }
    cout<<query(rt,1,INF,1,INF)<<'\n';
}

int main(){
    int t;cin>>t;
    while(t--){
        solve();
    }
    return 0;
```

扫描线

面积

线段树中闭区间$[l,r]$表示的实际数轴上$[l,r+1]$的线段，数轴上每一个小区间$[x,x+1]$表示线段树的数组中下标为x的叶子节点

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

const int MAXN=1e5+2;
const int INF=0x7fffffff;

struct line_t{
    int l,r,h,mark;

    line_t(int _l,int _r,int _h,int _mark):l(_l),r(_r),h(_h),mark(_mark){}

    bool operator<(const line_t& o)const{
        return h<o.h;
    }
};

vector<line_t> lines;

int n,cnt,rt;

#define ls d[p].lson
#define rs d[p].rson
#define MID int m=s+((t-s)>>1)

struct node_t{
    int lson,rson,tag;
    int len;
}d[MAXN<<7];

void pushup(int p,int s,int t){
    if(d[p].tag){
```

```
        d[p].len=t-s+1;
    }else d[p].len=d[ls].len+d[rs].len;
}

void modify(int& p,int s,int t,int l,int r,int c){
    if(!p)p=++cnt;
    if(l<=s&&t<=r){
        d[p].tag+=c;
        pushup(p,s,t);
        return;
    }
    MID;
    if(l<=m)modify(ls,s,m,l,r,c);
    if(r>m)modify(rs,m+1,t,l,r,c);
    pushup(p,s,t);
}

int main(){
    cin>>n;
    for(int i=1,x,y,_x,_y;i<=n;++i){
        cin>>x>>y>>_x>>_y;
        lines.emplace_back(x,_x,y,1);
        lines.emplace_back(x,_x,_y,-1);
    }
    sort(lines.begin(),lines.end());

    LL ans=0;

    for(int i=0;i<lines.size()-1;++i){
        modify(rt,0,INF,lines[i].l,lines[i].r-1,lines[i].mark);
        ans+=1ll*d[1].len*(lines[i+1].h-lines[i].h);
    }cout<<ans<<'\n';
    return 0;
}
```

周长

```
#include <iostream>
#include <cstdio>
#include <vector>
#include <algorithm>

#define LL long long
using namespace std;

const int MAXN=5e3+2;
const int INF=0x7fffffff;

struct line_t{
    int l,r,h,mark;

    line_t(int _l,int _r,int _h,int _mark):l(_l),r(_r),h(_h),mark(_mark){}

    bool operator<(const line_t& o)const{
        if(h==o.h)return mark>o.mark;
```

```
        return h<o.h;
    }
};

vector<line_t> lines;

int n,cnt,rt;

#define ls d[p].lson
#define rs d[p].rson
#define MID int m=1ll*s+((1ll*t-s)>>1)

struct node_t{
    int lson,rson,tag;//左儿子，右儿子，区间是否被线段完全覆盖
    int c;// 区间线段个数
    bool lc,rc;//区间左右端点是否被覆盖
    int len;//区间内的线段总长度
}d[MAXN<<7];

void pushup(int p,int s,int t){
    if(d[p].tag){
        d[p].len=t-s+1;
        d[p].lc=d[p].rc=true;
        d[p].c=1;
    }else{
        d[p].len=d[ls].len+d[rs].len;
        d[p].lc=d[ls].lc,d[p].rc=d[rs].rc;
        d[p].c=d[ls].c+d[rs].c;
        if(d[ls].rc&&d[rs].lc){
            d[p].c-=1;
        }
    }
}

void modify(int& p,int s,int t,int l,int r,int c){
    if(!p)p=++cnt;
    if(l<=s&&t<=r){
        d[p].tag+=c;
        pushup(p,s,t);
        return;
    }
    MID;
    if(l<=m)modify(ls,s,m,l,r,c);
    if(r>m)modify(rs,m+1,t,l,r,c);
    pushup(p,s,t);
}

int main(){
    cin>>n;
    for(int i=1,x,y,_x,_y;i<=n;++i){
        cin>>x>>y>>_x>>_y;
        // lines.emplace_back(x,_x,y,1);
        // lines.emplace_back(x,_x,_y,-1);
        lines.push_back({x,_x,y,1});
        lines.push_back({x,_x,_y,-1});
    }
```

```
    sort(lines.begin(),lines.end());

    LL ans=0;
    int pre=0;

    for(int i=0;i<lines.size()-1;++i){
        modify(rt,-INF,INF,lines[i].l,lines[i].r-1,lines[i].mark);
        ans+=abs(d[1].len-pre);
        ans+=2*d[1].c*(lines[i+1].h-lines[i].h);
        pre=d[1].len;
    }

    ans+=lines.back().r-lines.back().l;

    cout<<ans<<'\n';
    return 0;
}
```

主席树

主席树，全名可持久化线段树，又名hjt tree，是一种保留了多个历史版本的权值线段树。插入节点的时候去和上一个版本的权值线段树做连接，计算前缀。询问的时候带着两个版本一起跳儿子，统计前缀的差来得到区间信息。

区间第k大

值域到了`int`，先做离散化。

这里先算出左儿子区间上的点的个数 x，如果排名 k 小于 x，那么就往做左儿子跳，否则往右儿子跳找排名为 k-x 的节点，和平衡树的查找很像。

```
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-')sign=-sign;
    for(;ch>='0'&&ch<='9';ch=getchar())res=(res<<3)+(res<<1)+(ch^'0');
    return res*sign;
}

const int N=2e5+10;

int n,m;
int tot,sum[N<<5],rt[N],ls[N<<5],rs[N<<5];

int a[N],idx[N],len;

#define MID int m=s+((t-s)>>1)

int build(int s,int t){
    int p=++tot;
    if(s==t)return p;
    MID;
```

```
    ls[p]=build(s,m);
    rs[p]=build(m+1,t);
    return p;
}

int ins(int s,int t,int x,int _p){
    int p=++tot;
    ls[p]=ls[_p],rs[p]=rs[_p],sum[p]=sum[_p]+1;
    if(s==t)return p;
    MID;
    if(x<=m)ls[p]=ins(s,m,x,ls[p]);
    else rs[p]=ins(m+1,t,x,rs[p]);
    return p;
}

int query(int s,int t,int u,int v,int k){
    int x=sum[ls[v]]-sum[ls[u]];
    if(s==t)return s;
    MID;
    if(k<=x)return query(s,m,ls[u],ls[v],k);
    else return query(m+1,t,rs[u],rs[v],k-x);
}

int getid(int x){
    return lower_bound(idx+1,idx+1+len,x)-idx;
}

void init(){
    for(int i=1;i<=n;++i)idx[i]=a[i];
    sort(idx+1,idx+1+n);
    len=unique(idx+1,idx+1+n)-idx-1;
    rt[0]=build(1,len);
    for(int i=1;i<=n;++i){
        rt[i]=ins(1,len,getid(a[i]),rt[i-1]);
    }
}

void solve(){
    for(int i=1;i<=m;++i){
        int l=read(),r=read(),k=read();
        int ans=idx[query(1,len,rt[l-1],rt[r],k)];
        printf("%d\n",ans);
    }
}

int main(){
    n=read(),m=read();
    for(int i=1;i<=n;++i){
        a[i]=read();
    }
    init();
    solve();
    return 0;
}
```

区间[l, r]中属于某一个值域[L, R]中的点的个数

一个序列 $a_i\ (i=1,2,\ldots,n)$，在 $a_l, a_{l+1}, \ldots, a_r$ 中有多少个 a_i 满足 $L <= a_i <= R$？

带着两个版本的树跳，然后统计区间中满足的点的个数就好了。

```
#include <bits/stdc++.h>

using namespace std;

const int N=1e5+10;

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

int n,q;

vector<int> G[N];
int ver[N];
int tin[N],tout[N],timer;

void dfs(int u,int fno){
    tin[u]=++timer;
    for(auto&& v:G[u])if(v!=fno){
        dfs(v,u);
    }
    tout[u]=timer;
}

int ls[N<<5],rs[N<<5],rt[N<<5],sum[N<<5],tot;

#define MID int m=s+((t-s)>>1)

int ins(int s,int t,int x,int _p){
    int p=++tot;
    ls[p]=ls[_p],rs[p]=rs[_p],sum[p]=sum[_p]+1;
    if(s==t)return p;
    MID;
    if(x<=m)ls[p]=ins(s,m,x,ls[p]);
    else rs[p]=ins(m+1,t,x,rs[p]);
    return p;
}

int query(int s,int t,int u,int v,int l,int r){
    if(l<=s&&t<=r)return sum[v]-sum[u];
    MID;
    int res=0;
    if(l<=m)res+=query(s,m,ls[u],ls[v],l,r);
    if(r>m)res+=query(m+1,t,rs[u],rs[v],l,r);
    return res;
}
```

```
void solve(){
    n=read(),q=read();
    for(int i=1;i<=n;++i)G[i].clear();
    tot=0,timer=0;
    for(int i=1,u,v;i<=n-1;++i){
        u=read(),v=read();
        G[u].push_back(v);
        G[v].push_back(u);
    }
    dfs(1,0);
    for(int i=1;i<=n;++i)ver[i]=read(),rt[i]=ins(1,n,tin[ver[i]],rt[i-1]);
    for(int _=1;_<=q;++_){
        int l=read(),r=read(),x=read();
        int L=tin[x],R=tout[x];
        int res=query(1,n,rt[l-1],rt[r],L,R);
        if(res>=1)puts("YES");
        else puts("NO");
    }
    puts("");
}

int main(){
    int t=read();
    while(t--)solve();
    return 0;
}
```

带历史版本的区间加区间查询

也就是做一个可持久化线段树但是是支持区修的，这类题型目前不常见，也不会主动用这个方法，是自己摸索的一个方法，慎用

```
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)

const int N=2e5+10;

struct node_t{
    int v,lson,rson,tag;
};

int rt[N],tot;
```

```
node_t d[N<<7];

#define ls d[p].lson
#define rs d[p].rson
#define _ls d[_p].lson
#define _rs d[_p].rson
#define MID int m=s+((t-s)>>1)

void pushup(int p,int s,int t){
    d[p].v=d[ls].v+d[rs].v+d[p].tag*(t-s+1);
}

void build(int& p,int s,int t){
    p=++tot;
    if(s==t)return;
    MID;
    build(ls,s,m);
    build(rs,m+1,t);
    pushup(p,s,t);
}

void modify(int& p,int _p,int s,int t,int l,int r,int c){
    p=++tot;
    d[p]=d[_p];
    if(l<=s&&t<=r){
        d[p].v+=c*(t-s+1),d[p].tag+=c;
        return;
    }
    MID;
    if(l<=m)modify(ls,_ls,s,m,l,r,c);
    if(r>m)modify(rs,_rs,m+1,t,l,r,c);
    pushup(p,s,t);
}

int query(int p,int s,int t,int l,int r,int tag){
    if(!p)return 0;
    if(l<=s&&t<=r){
        return d[p].v+tag*(t-s+1);
    }
    MID;
    int ans=0;
    if(l<=m)ans+=query(ls,s,m,l,r,tag+d[p].tag);
    if(r>m)ans+=query(rs,m+1,t,l,r,tag+d[p].tag);
    return ans;
}

int n,m;

int main(){
    n=read(),m=read();
    build(rt[0],1,n);
    rep(i,1,m){
        int op=read(),l=read(),r=read();
        if(op==1){
            int x=read();
```

```
            modify(rt[i],rt[i-1],1,n,l,r,x);
        }else{
            rt[i]=rt[i-1];
            int lst=read(),now=read();
            printf("%d\n",query(rt[now],1,n,l,r,0)-query(rt[lst-1],1,n,l,r,0));
        }
    }
    return 0;
}
```

树套树

线段树套平衡树

二逼平衡树（树套树）板子，代码如下：

`splay in segment tree`

```
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)

const int N=5e4+10;
const int INF=2147483647;

struct splay_t{
    int rt,tot;
    vector<int> fa,val,cnt,sz,ch[2];

    splay_t(){
        fa.push_back(0);
        val.push_back(0);
        cnt.push_back(0);
        sz.push_back(0);
        ch[0].push_back(0);
        ch[1].push_back(0);
    }

    void maintain(int x){sz[x]=sz[ch[0][x]]+sz[ch[1][x]]+cnt[x];}

    bool get(int x){return x==ch[1][fa[x]];}

    void clear(int x){fa[x]=ch[0][x]=ch[1][x]=val[x]=cnt[x]=sz[x]=0;}
```

```
void rotate(int x){
    int y=fa[x],z=fa[y],chk=get(x);

    ch[chk][y]=ch[chk^1][x];
    if(ch[chk^1][x])fa[ch[chk^1][x]]=y;

    ch[chk^1][x]=y;
    fa[y]=x;

    if(z)ch[y==ch[1][z]][z]=x;
    fa[x]=z;

    maintain(y);
    maintain(x);
}

void splay(int x){
    for(int f=fa[x];f=fa[x],f;rotate(x)){
        if(fa[f])rotate(get(x)==get(f)?f:x);
    }
    rt=x;
}

void ins(int k){
    if(!rt){
        ++tot;
        fa.push_back(0);
        val.push_back(k);
        cnt.push_back(1);
        sz.push_back(0);
        ch[0].push_back(0);
        ch[1].push_back(0);
        rt=tot;
        maintain(rt);
        return;
    }
    int cur=rt,f=0;
    while(1){
        if(val[cur]==k){
            cnt[cur]++;
            maintain(cur);
            maintain(f);
            splay(cur);
            break;
        }
        f=cur;
        cur=ch[val[cur]<k][cur];
        if(!cur){
            ++tot;
            fa.push_back(f);
            val.push_back(k);
            cnt.push_back(1);
            sz.push_back(0);
            ch[0].push_back(0);
            ch[1].push_back(0);
            ch[val[f]<k][f]=tot;
```

```
                maintain(tot);
                maintain(f);
                splay(tot);
                break;
            }
        }
    }

    int rk(int k){
        int res=0,cur=rt;
        while(1){
            if(val[cur]>k){
                cur=ch[0][cur];
            }else{
                res+=sz[ch[0][cur]];
                if(!cur)return res+1;
                if(val[cur]==k){
                    splay(cur);
                    return res+1;
                }
                res+=cnt[cur];
                cur=ch[1][cur];
            }
        }
    }

    int kth(int k){
        int cur=rt;
        while(1){
            if(ch[0][cur]&&k<=sz[ch[0][cur]]){
                cur=ch[0][cur];
            }else{
                k-=sz[ch[0][cur]]+cnt[cur];
                if(k<=0){
                    splay(cur);
                    return val[cur];
                }
                cur=ch[1][cur];
            }
        }
    }

    int pre(){
        int cur=ch[0][rt];
        if(!cur)return cur;
        while(ch[1][cur])cur=ch[1][cur];
        splay(cur);
        return cur;
    }

    int nxt(){
        int cur=ch[1][rt];
        if(!cur)return cur;
        while(ch[0][cur])cur=ch[0][cur];
        splay(cur);
        return cur;
```

```
    }

    void del(int k){
        rk(k);
        if(cnt[rt]>1){
            cnt[rt]--;
            maintain(rt);
            return;
        }
        if(!ch[0][rt]&&!ch[1][rt]){
            clear(rt);
            rt=0;
            return;
        }
        if(!ch[0][rt]){
            int cur=rt;
            rt=ch[1][rt];
            fa[rt]=0;
            clear(cur);
            return;
        }
        if(!ch[1][rt]){
            int cur=rt;
            rt=ch[0][rt];
            fa[rt]=0;
            clear(cur);
            return;
        }
        int cur=rt,x=pre();
        fa[ch[1][cur]]=x;
        ch[1][x]=ch[1][cur];
        clear(cur);
        maintain(rt);
    }

    int getpre(int x){
        ins(x);
        int res=pre();
        del(x);
        return val[res];
    }

    int getnxt(int x){
        ins(x);
        int res=nxt();
        del(x);
        return val[res];
    }
};

int n,m;

int a[N];

splay_t splay[N<<2];
```

```
#define ls (p<<1)
#define rs (p<<1|1)
#define MID int m=s+((t-s)>>1)

void build(int p,int s,int t){
    splay[p].ins(-INF);
    splay[p].ins(INF);
    if(s==t){
        return;
    }
    MID;
    build(ls,s,m);
    build(rs,m+1,t);
}

void ins(int p,int s,int t,int x,int c){
    splay[p].ins(c);
    if(s==t){
        return;
    }
    MID;
    if(x<=m)ins(ls,s,m,x,c);
    else ins(rs,m+1,t,x,c);
}

int qry_rank(int p,int s,int t,int l,int r,int x){
    if(l<=s&&t<=r){
        // -1 是为了排除-INF
        return splay[p].rk(x)-1;
    }
    MID;
    int res=0;
    if(l<=m)res+=qry_rank(ls,s,m,l,r,x);
    if(r>m)res+=qry_rank(rs,m+1,t,l,r,x);
    if(l<=m&&r>m)--res;
    return res;
}

int qry_kth(int l,int r,int k){
    int ql=0,qr=1e8;
    while(ql<=qr){
        int qmid=ql+qr>>1;
        if(k>=qry_rank(1,1,n,l,r,qmid))ql=qmid+1;
        else qr=qmid-1;
    }
    return qr;
}

void modify(int p,int s,int t,int x,int c){
    splay[p].del(a[x]);
    splay[p].ins(c);
    if(s==t){
        return;
    }
    MID;
    if(x<=m)modify(ls,s,m,x,c);
```

```
    else modify(rs,m+1,t,x,c);
}

int getpre(int p,int s,int t,int l,int r,int x){
    if(l<=s&&t<=r){
        return splay[p].getpre(x);
    }
    MID;
    int res=-INF;
    if(l<=m)res=max(res,getpre(ls,s,m,l,r,x));
    if(r>m)res=max(res,getpre(rs,m+1,t,l,r,x));
    return res;
}

int getnxt(int p,int s,int t,int l,int r,int x){
    if(l<=s&&t<=r){
        return splay[p].getnxt(x);
    }
    MID;
    int res=INF;
    if(l<=m)res=min(res,getnxt(ls,s,m,l,r,x));
    if(r>m)res=min(res,getnxt(rs,m+1,t,l,r,x));
    return res;
}

int main(){
    n=read(),m=read();
    build(1,1,n);
    rep(i,1,n)a[i]=read(),ins(1,1,n,i,a[i]);
    rep(i,1,m){
        int op=read();
        if(op==1){
            int l=read(),r=read(),x=read();
            printf("%d\n",qry_rank(1,1,n,l,r,x));
        }else if(op==2){
            int l=read(),r=read(),k=read();
            printf("%d\n",qry_kth(l,r,k));
        }else if(op==3){
            int x=read(),c=read();
            modify(1,1,n,x,c);
            a[x]=c;
        }else if(op==4){
            int l=read(),r=read(),x=read();
            printf("%d\n",getpre(1,1,n,l,r,x));
        }else if(op==5){
            int l=read(),r=read(),x=read();
            printf("%d\n",getnxt(1,1,n,l,r,x));
        }
    }
    return 0;
}
```

`rbtree in segment tree`

```
#include <bits/stdc++.h>
```

```cpp
using namespace std;

#include<ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
typedef
tree<double,null_type,less<double>,rb_tree_tag,tree_order_statistics_node_update>
rbt_t;
/*
insert(), erase(), lower_bound(), order_of_key(), find_by_order(), prev(), next()
*/

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)

const int N=5e4+10;
const int INF=2147483647;

int n,m;

int a[N];

struct tree_t{
    rbt_t rbt;

    void ins(int x,int tim){
        rbt.insert(x+tim*1e-6);
    }

    void del(int x){
        rbt.erase(rbt.lower_bound(x));
    }

    int rk(int x){
        return (int)rbt.order_of_key(x)+1;
    }

    int kth(int k){
        return (int)*rbt.find_by_order(k-1);
    }

    int pre(int x){
        return (int)round(*(--rbt.lower_bound(x)));
    }

    int nxt(int x){
        return (int)round(*rbt.lower_bound(x + 1));
    }
```

```
};

tree_t T[N<<2];

#define ls (p<<1)
#define rs (p<<1|1)
#define MID int m=s+((t-s)>>1)

void build(int p,int s,int t){
    T[p].ins(-INF,0);
    T[p].ins(INF,0);
    if(s==t){
        return;
    }
    MID;
    build(ls,s,m);
    build(rs,m+1,t);
}

void ins(int p,int s,int t,int x,int c,int tim){
    T[p].ins(c,tim);
    if(s==t){
        return;
    }
    MID;
    if(x<=m)ins(ls,s,m,x,c,tim);
    else ins(rs,m+1,t,x,c,tim);
}

int qry_rank(int p,int s,int t,int l,int r,int x){
    if(l<=s&&t<=r){
        // -1 是为了排除-INF
        return T[p].rk(x)-1;
    }
    MID;
    int res=0;
    if(l<=m)res+=qry_rank(ls,s,m,l,r,x);
    if(r>m)res+=qry_rank(rs,m+1,t,l,r,x);
    if(l<=m&&r>m)--res;
    return res;
}

int qry_kth(int l,int r,int k){
    int ql=0,qr=1e8;
    while(ql<=qr){
        int qmid=ql+qr>>1;
        if(k>=qry_rank(1,1,n,l,r,qmid))ql=qmid+1;
        else qr=qmid-1;
    }
    return qr;
}

void modify(int p,int s,int t,int x,int c,int tim){
    T[p].del(a[x]);
    T[p].ins(c,tim);
    if(s==t){
```

```
        return;
    }
    MID;
    if(x<=m)modify(ls,s,m,x,c,tim);
    else modify(rs,m+1,t,x,c,tim);
}

int getpre(int p,int s,int t,int l,int r,int x){
    if(l<=s&&t<=r){
        return T[p].pre(x);
    }
    MID;
    int res=-INF;
    if(l<=m)res=max(res,getpre(ls,s,m,l,r,x));
    if(r>m)res=max(res,getpre(rs,m+1,t,l,r,x));
    return res;
}

int getnxt(int p,int s,int t,int l,int r,int x){
    if(l<=s&&t<=r){
        return T[p].nxt(x);
    }
    MID;
    int res=INF;
    if(l<=m)res=min(res,getnxt(ls,s,m,l,r,x));
    if(r>m)res=min(res,getnxt(rs,m+1,t,l,r,x));
    return res;
}

int main(){
    n=read(),m=read();
    build(1,1,n);
    rep(i,1,n)a[i]=read(),ins(1,1,n,i,a[i],i);
    rep(i,1,m){
        int op=read();
        if(op==1){
            int l=read(),r=read(),x=read();
            printf("%d\n",qry_rank(1,1,n,l,r,x));
        }else if(op==2){
            int l=read(),r=read(),k=read();
            printf("%d\n",qry_kth(l,r,k));
        }else if(op==3){
            int x=read(),c=read();
            modify(1,1,n,x,c,i+n);
            a[x]=c;
        }else if(op==4){
            int l=read(),r=read(),x=read();
            printf("%d\n",getpre(1,1,n,l,r,x));
        }else if(op==5){
            int l=read(),r=read(),x=read();
            printf("%d\n",getnxt(1,1,n,l,r,x));
        }
    }
    return 0;
}
```

分块

```
#include <bits/stdc++.h>

using namespace std;

#define LL long long

int read(){
    int res=0,sign=1;char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-')sign=-sign;
    for(;ch>='0'&&ch<='9';ch=getchar())res=(res<<3)+(res<<1)+(ch^'0');
    return res*sign;
}

const int N=5e4+10;

int n;

int id[N],len;
LL a[N],b[N],s[N];

void add(int l,int r,int c){
    int op=id[l],ed=id[r];
    if(op==ed){
        for(int i=l;i<=r;++i)a[i]+=c,s[op]+=c;
        return;
    }
    for(int i=l;id[i]==op;++i)a[i]+=c,s[op]+=c;
    for(int i=op+1;i<ed;++i)b[i]+=c,s[i]+=1ll*c*len;
    for(int i=r;id[i]==ed;--i)a[i]+=c,s[ed]+=c;
}

int query(int l,int r,int mod){
    int op=id[l],ed=id[r],res=0;
    if(op==ed){
        for(int i=l;i<=r;++i)res=(1ll*res+a[i]+b[op]+mod)%mod;
        return res;
    }
    for(int i=l;id[i]==op;++i)res=(1ll*res+a[i]+b[op]+mod)%mod;
    for(int i=op+1;i<ed;++i)res=(1ll*res+s[i]+mod)%mod;
    for(int i=r;id[i]==ed;--i)res=(1ll*res+a[i]+b[ed]+mod)%mod;
    return res;
}

int main(){
    n=read();len=sqrt(n);
    for(int i=1;i<=n;++i){
        a[i]=read();
        id[i]=(i-1)/len+1;
        s[id[i]]+=a[i];
    }
    for(int i=1,op,l,r,c;i<=n;++i){
        op=read(),l=read(),r=read(),c=read();
        if(op==0){
```

```
            add(l,r,c);
        }else{
            printf("%d\n",query(l,r,c+1));
        }
    }
    return 0;
}
```

图论

判断是否是一个点子树

通过$tin[v] >= tin[u] \&\& tin[v] <= tout[u]$判断

LCA

倍增版本

```
void dfs(int x,int fno){
    fa[x][0]=fno;
    dep[x]=dep[fno]+1;
    for(int i=1;i<=30;++i){
        fa[x][i]=fa[fa[x][i-1]][i-1];
    }
    for(auto&& v:p[x])if(v!=fno){
        dfs(v,x);
    }
}

int lca(int x,int y){
    if(dep[x]>dep[y])swap(x,y);
    for(int i=30;i>=0;--i)if(dep[fa[y][i]]>=dep[x]){
        y=fa[y][i];
    }
    if(x==y)return x;
    for(int i=30;i>=0;--i)if(fa[x][i]!=fa[y][i]){
        x=fa[x][i],y=fa[y][i];
    }
    return fa[x][0];
}
```

树剖版本

```
void dfs(int u,int fno){
    fa[u]=fno;
    siz[u]=1;
    dep[u]=dep[fno]+1;
    son[u]=-1;
    for(auto&& v:G[u])if(v!=fno){
        dfs(v,u);
        siz[u]+=siz[v];
        if(son[u]==-1||siz[son[u]]<siz[v])son[u]=v;
    }
}
```

```
void _dfs(int u,int ufno){
    top[u]=ufno;
    if(son[u]==-1)return;
    _dfs(son[u],ufno);
    for(auto&& v:G[u])if(v!=fa[u]&&v!=son[u]){
        _dfs(v,v);
    }
}

int lca(int u,int v){
    while(top[u]!=top[v]){
        if(dep[top[u]]<dep[top[v]])v=fa[top[v]];
        else u=fa[top[u]];
    }
    if(dep[u]<dep[v])return u;
    else return v;
}
```

最短路

SPFA

```
bool spfa(int n,int s){
    fill_n(dis+1,n,INF);
    dis[s]=0,isv[s]=1;
    q.push(s);
    while(!q.empty()){
        int u=q.front();q.pop();
        isv[u]=0;
        for(auto& [v,w]:p[u]){
            if(dis[v]>dis[u]+w){
                dis[v]=dis[u]+w;
                cnt[v]=cnt[u]+1;
                if(cnt[v]>=n)return false;
                if(!isv[v])q.push(v),isv[v]=1;
            }
        }
    }
    return true;
}
```

Dijstra

```
void dijkstra(int n,int s){
    fill_n(dis+1,n,INF);
    dis[s]=0;
    q.push({s,0});
    while(!q.empty()){
        int u=q.top().u;
        q.pop();
        if(vis[u])continue;
        vis[u]=1;
        for(auto& [v,w]:p[u]){
```

```
            if(dis[v]>dis[u]+w){
                dis[v]=dis[u]+w;
                q.push({v,dis[v]});
            }
        }
    }
}
```

差分约束

给出一组包含 m 个不等式，有 n 个未知数的形如：

$$\begin{cases} x_{c_1} - x_{c_1'} \leq y_1 \\ x_{c_2} - x_{c_2'} \leq y_2 \\ \cdots \\ x_{c_m} - x_{c_m'} \leq y_m \end{cases}$$

的不等式组，求任意一组满足这个不等式组的解。

这个题的意思就是将c'向c建边，权值为y，看这个图有没有负环。用spfa算法处理即可。

MST

Kruskal贪心加边即可

次小生成树

方法是先找出最小生成树，建立最小生成树。然后倍增维护最小生成树的祖先，路径上的最大权值，次大权值。

最后在看所有的非最小生成树的边的两点在最小生成树中的路径上的最大值是多少，并且与未使用的该边替换，更新答案。如果最大值等于未使用的该边的权值，那么直接找次大值替换。

最后的答案等于所有替换方案的最小值。

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
const int MAXN=1e5+2;
const int MAXM=3e5+2;
const int INF=0x7fffffffffffffff;

struct Tr{
private:
    struct edge{
        int v,w;
    };
    vector<edge> p[MAXN];
    int fa[MAXN][31],dep[MAXN],m[MAXN][31],mm[MAXN][31];

public:
    void add(int u,int v,int w){
        p[u].push_back({v,w});
        p[v].push_back({u,w});
    }
```

```
    void dfs(int x,int fno){
        if(fno)dep[x]=dep[fno]+1,fa[x][0]=fno,mm[x][0]=-INF;

        for(int i=1;i<=30;++i){
            fa[x][i]=fa[fa[x][i-1]][i-1];
            int tmp[4]={m[x][i-1],m[fa[x][i-1]][i-1],
                        mm[x][i-1],mm[fa[x][i-1]][i-1]};
            sort(tmp,tmp+4);
            m[x][i]=tmp[3];
            int ptr=2;
            for(;ptr>=0&&tmp[ptr]==tmp[3];){--ptr;}
            mm[x][i]=ptr==-1?-INF:tmp[ptr];
        }

        for(auto& edge:p[x]){
            int v=edge.v,w=edge.w;
            if(v==fno)continue;
            m[v][0]=w;
            dfs(v,x);
        }
    }

    int lca(int x,int y){
        if(dep[x]>dep[y])swap(x,y);
        for(int i=30;i>=0;--i){
            if(dep[fa[y][i]]>=dep[x])y=fa[y][i];
        }
        if(x==y)return x;
        for(int i=30;i>=0;--i){
            if(fa[x][i]!=fa[y][i]){
                x=fa[x][i],y=fa[y][i];
            }
        }
        return fa[x][0];
    }

    int query(int x,int y,int w){
        int ret=-INF;
        for(int i=30;i>=0;--i){
            if(dep[fa[x][i]]>=dep[y]){
                if(w!=m[x][i])
                    ret=max(ret,m[x][i]);
                else
                    ret=max(ret,mm[x][i]);
                x=fa[x][i];
            }
        }
        return ret;
    }
};

Tr tr;

struct _edge_{
    int u,v,w;
    bool operator<(const _edge_& other)const{return w<other.w;}
```

```
};

vector<_edge_> edges;
int fa[MAXN],n,m,used[MAXM];

int find(int x){return fa[x]==x?x:fa[x]=find(fa[x]);}
void unite(int u,int v){int fu=find(u),fv=find(v);if(fu!=fv)fa[fu]=fv;}

int kruskal(){
    int ans=0,cnt=0;
    for(int i=1;i<=n;++i)fa[i]=i;
    sort(edges.begin(),edges.end());
    for(int i=0;i<edges.size();++i){
        _edge_ e=edges[i];
        int u=e.u,v=e.v,w=e.w;
        if(find(u)!=find(v)){
            unite(u,v);
            ans+=w,++cnt;
            used[i]=1;
            tr.add(u,v,w);
        }
        if(cnt==n-1)break;
    }
    return ans;
}

signed main(){
    cin>>n>>m;
    for(int i=1;i<=m;++i){
        int u,v,w;cin>>u>>v>>w;
        edges.push_back({u,v,w});
    }
    int ans_1=kruskal(),ans_2=INF;
    tr.dfs(1,0);
    for(int i=0;i<edges.size();++i){
        if(!used[i]){
            int u=edges[i].u,v=edges[i].v,w=edges[i].w;
            int _lca=tr.lca(u,v);
            int tmp1=tr.query(u,_lca,w);
            int tmp2=tr.query(v,_lca,w);
            int _m=max(tmp1,tmp2);
            if(_m>-INF)
            ans_2=min(ans_2,ans_1-_m+w);
        }
    }
    if(ans_2!=INF)cout<<ans_2<<endl;
    else cout<<-1<<endl;
}
```

内向基环树森林

对于所有的点都只有一条出边，那么最后的图要么为裸的简单环，要么为简单环上，每一个点向外出去的反边构成一棵树，所有的这样联通的分量构成内向基环树森林。

如何找到所有的环和预处理所有环外树的信息？

可以先对于所有入度为 0 的点 dfs 找到环，如果被找过就不管，没找过就记录该环。然后森林中只剩下裸的简单环，用另一种找简单环的 dfs 找到即可。

枚举所有环上的点，向外构建树的信息即可。

例题：

给你一个长度为 N 的序列 X，其中每个元素的长度都在 1 和 N 之间，以及一个长度为 N 的序列 A。

输出在 A 上执行以下操作 K 次的结果。

- 用 B 替换 A，使得 $B_i = A_{X_i}$.

思路：也就是找到 i 被 X 作用 k 次后的结果，构建内向基环树森林 $i->x[i]$，倍增处理出所有环外树的节点的父亲，就可以实现 $O(\log depth)$ 跳点，然后如果在环上，就找到往后数 k 个的结果即可，时间复杂度为 $O(n \log n)$(当然这个题有码量更小的 $O(n \log k)$ 的数学做法，直接倍增即可)

代码如下：

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

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)

typedef long long ll;

const int N=2e5+120;

int n,k,x[N],a[N];

int deg[N];

int tot,st[N],ed[N];

int vis[N];

int isinring[N];
vector<int> tree[N];
int fa[N][32];
int dep[N];

vector<int> ring[N];
int bel[N];
int pos[N];

void getring1(int u,int f,int tag){
```

```
    if(vis[u]==tag){
        st[++tot]=u;
        ed[tot]=f;
        return;
    }else if(vis[u]!=0&&vis[u]!=tag){
        return;
    }
    vis[u]=tag;
    getring1(x[u],u,tag);
}

void getring2(int u,int f){
    if(vis[u]){
        st[++tot]=u;
        ed[tot]=f;
        return;
    }
    vis[u]=n+1;
    getring2(x[u],u);
}

void getinfo(int u,int f,int d){
    fa[u][0]=f;
    dep[u]=d;
    for(int i=1;i<=30;++i)fa[u][i]=fa[fa[u][i-1]][i-1];
    for(auto&& v:tree[u])if(!isinring[v]){
        getinfo(v,u,d+1);
    }
}

int jumpnode(int u,int k){
    for(int i=30;i>=0;--i)if((k>>i)&1)u=fa[u][i];
    return u;
}

int ans[N];

signed main(){
    n=read(),k=read();
    rep(i,1,n)x[i]=read(),++deg[x[i]],tree[x[i]].push_back(i);
    rep(i,1,n)a[i]=read();
    rep(i,1,n)if(!vis[i]&&!deg[i])getring1(i,0,i);
    rep(i,1,n)if(!vis[i])getring2(i,0);

    rep(i,1,tot){
        for(int u=st[i];;u=x[u]){
            isinring[u]=1;
            bel[u]=i;
            ring[i].push_back(u);
            pos[u]=ring[i].size()-1;
            if(u==ed[i])break;
        }
    }

    rep(i,1,tot){
        for(auto&& u:ring[i]){
```

```
            getinfo(u,0,0);
        }
    }

    rep(i,1,n){
        if(isinring[i]){
            int p=pos[i],id=bel[i],sz=ring[id].size();
            ans[i]=ring[id][(p+k)%sz];
        }else{
            if(k<=dep[i]){
                ans[i]=jumpnode(i,k);
            }else{
                int tmp=jumpnode(i,dep[i]);
                int p=pos[tmp],id=bel[tmp],sz=ring[id].size();
                ans[i]=ring[id][(p+(k-dep[i]))%sz];
            }
        }
    }
    rep(i,1,n)printf("%lld ",a[ans[i]]);puts("");
}
```

直接倍增做法如下：

```
#define int long long
const int N=2e5+120;

int n,k,x[N],a[N];

int xx[N][70];

int ans[N];

void init(){
    rep(i,1,n){
        xx[i][0]=x[i];
    }
    rep(j,1,60)rep(i,1,n){
        xx[i][j]=xx[xx[i][j-1]][j-1];
    }
}

signed main(){
    n=read(),k=read();
    rep(i,1,n)x[i]=read(),ans[i]=i;
    rep(i,1,n)a[i]=read();
    init();
    for(int j=60;j>=0;--j)if((k>>j)&1){
        rep(i,1,n)ans[i]=xx[ans[i]][j];
    }
    rep(i,1,n)printf("%lld ",a[ans[i]]);puts("");
    return 0;
}
```

Tarjan

```cpp
#include <bits/stdc++.h>

using namespace std;

const int MAXN=1e4+2;

int n,m;
int dfn[MAXN],low[MAXN],cnt;
int bel[MAXN],ins[MAXN],siz[MAXN],idx;
int st[MAXN],top;

vector<int> p[MAXN];

void add(int u,int v){
    p[u].push_back(v);
}

void tarjan(int x){
    dfn[x]=low[x]=++cnt;
    ins[st[++top]=x]=1;
    for(auto v:p[x]){
        if(!dfn[v]){
            tarjan(v);
            low[x]=min(low[x],low[v]);
        }else if(ins[v]){
            low[x]=min(low[x],dfn[v]);
        }
    }
    if(low[x]==dfn[x]){
        int v;++idx;
        do{
            ins[v=st[top--]]=0;
            bel[v]=idx;
            ++siz[idx];
        }while(v!=x);
    }
}

int main(){
    cin>>n>>m;
    for(int i=1,u,v;i<=m;++i){
        cin>>u>>v;
        add(u,v);
    }
    for(int i=1;i<=n;++i){
        if(!dfn[i])tarjan(i);
    }
    int ans=0;
    for(int i=1;i<=idx;++i){
        //siz>1的强连通分量个数
        ans+=siz[i]>1;
    }cout<<ans<<'\n';
    return 0;
```

```
}
```

若是无向图，则上面的`else if`就直接`else`，并且注意`dfs`添加一个参数`f`表示上一个点（类似父亲），遍历`v`时添加`if(v!=f)`

无向图的割边：

```
//求割边 加在!dfn[v]分支最后
if(low[v]>dfn[x])e.push_back({min(x,v),max(x,v)});
```

2-SAT问题

若某个条件是`a`或`b`

这个题的关键是将非`a`与`b`连接，非`b`与`a`连接。我们在这个题先分配状态，1~n为$x_i=0$，n+1~2*n为$x_i=1$。这样子我们只需要跑一遍缩点（缩点后的图是有向无环图），然后看每一个$x_i=0$的状态和$x_i=1$的状态的拓扑序，我们创造解只要看拓扑序大的，也就是记录连通分量顺序小的。如果一样，说明强连通分量中出现了矛盾。

```
#include <bits/stdc++.h>

using namespace std;

const int MAXN=2e6+2;

int n,m;
int dfn[MAXN],low[MAXN],cnt;
int bel[MAXN],idx;
int st[MAXN],ins[MAXN],top;

vector<int> p[MAXN];

inline void add(int u,int v){
    p[u].push_back(v);
}

void tarjan(int x){
    dfn[x]=low[x]=++cnt;
    ins[st[++top]=x]=1;
    for(auto v:p[x]){
        if(!dfn[v]){
            tarjan(v);
            low[x]=min(low[x],low[v]);
        }else if(ins[v]){
            low[x]=min(low[x],dfn[v]);
        }
    }
    if(low[x]==dfn[x]){
        int v;++idx;
        do{
            ins[v=st[top--]]=0;
            bel[v]=idx;
        }while(v!=x);
    }
```

```
}

vector<int> ans;

int main(){
    cin>>n>>m;
    for(int i=1,u,a,v,b;i<=m;++i){
        // ksat问题计算理论中讲过，能否使得所有k元的析取表达式之合取为真
        // 2sat: u==a||v==b
        cin>>u>>a>>v>>b;
        add(u+(1-a)*n,v+b*n);
        add(v+(1-b)*n,u+a*n);
    }
    for(int i=1;i<=2*n;++i){
        if(!dfn[i])tarjan(i);
    }
    bool flag=true;
    for(int i=1;i<=n;++i){
        if(bel[i]<bel[i+n]){
            ans.push_back(0);
        }else if(bel[i]>bel[i+n]){
            ans.push_back(1);
        }else{
            flag=false;
            break;
        }
    }
    if(flag){
        cout<<"POSSIBLE"<<'\n';
        for(auto i:ans){
            cout<<i<<" ";
        }cout<<'\n';
    }else{
        cout<<"IMPOSSIBLE"<<'\n';
    }
    return 0;
}
```

网络流

最大流

EK算法

通过`bfs()`查找增广路来更新最大流

关键思想

`lst`：在`bfs`过程中找到的增广路的某个点，它上个点的索引

`pre`：在`bfs`过程中找到的增广路的某个点，它上个点与自己连成的边在上个点的邻接表中的索引

`flow`：某个点的通过的流量

```
#include <bits/stdc++.h>
#define int long long
```

```cpp
using namespace std;

const int MAXN=2e2+2;
const int INF=0x3f3f3f3f3f3f3f3f;

struct edge_t{
    int v,w,rid;
};

int n,m,s,t;
vector<edge_t> G[MAXN];
bool vis[MAXN];
int lst[MAXN],pre[MAXN],flow[MAXN];

inline void add(int u,int v,int w){
    int id=G[u].size();
    int rid=G[v].size();
    G[u].push_back({v,w,rid});
    G[v].push_back({u,0,id});
}

bool bfs(){
    memset(vis,0,sizeof vis);
    queue<int> q;
    q.push(s);
    vis[s]=true;
    flow[s]=INF;
    while(!q.empty()){
        int u=q.front();q.pop();
        for(int i=0;i<G[u].size();++i){
            int v=G[u][i].v,w=G[u][i].w;
            if(!vis[v]&&w>0){
                vis[v]=true;
                flow[v]=min(flow[u],w);
                lst[v]=u;
                pre[v]=i;
                if(v==t)return true;
                q.push(v);
            }
        }
    }
    return false;
}

int ek(){
    int res=0;
    while(bfs()){
        res+=flow[t];
        for(int i=t;i!=s;i=lst[i]){
            int u=lst[i],v=i,rid=G[u][pre[v]].rid;
            G[u][pre[v]].w-=flow[t];
            G[v][rid].w+=flow[t];
        }
    }
    return res;
}
```

```
signed main(){
    cin>>n>>m>>s>>t;
    for(int i=1,u,v,w;i<=m;++i){
        cin>>u>>v>>w;
        add(u,v,w);
    }
    cout<<ek()<<'\n';
}
```

Dinic算法

通过`bfs()`分层后，考虑当前弧优化和剪枝的情况下做`dfs()`去更新最大流，下面`dfs()`有两种写法，推荐未注释写法。

关键思想

`dep`：`bfs`分层结果

`cur`：当前弧优化，就是将不需要搜的边排除掉所用到的

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

const int MAXN=2e2+2;
const long long INF=0x3f3f3f3f3f3f3f3f;

int n,m,s,t;
int dep[MAXN],cur[MAXN];

struct edge_t{
    int v,w,rid;
};

vector<edge_t> G[MAXN];

inline void add(int u,int v,int w){
    int id=G[u].size();
    int rid=G[v].size();
    G[u].push_back({v,w,rid});
    G[v].push_back({u,0,id});
}

bool bfs(){
    memset(dep,-1,sizeof dep);
    queue<int> q;
    q.push(s);dep[s]=0;
    while(!q.empty()){
        int u=q.front();q.pop();
        for(auto&& [v,w,rid]:G[u])if(dep[v]==-1&&w>0){
            dep[v]=dep[u]+1;
            q.push(v);
        }
    }
    return dep[t]!=-1;
```

```
}

// int dfs(int u,int flow){
//     if(u==t)return flow;
//     for(int i=cur[u];i<G[u].size();++i){
//         cur[u]=i;
//         int v=G[u][i].v,w=G[u][i].w,rid=G[u][i].rid,res;
//         if(dep[v]==dep[u]+1&&w>0){
//             if(res=dfs(v,min(w,flow))){
//                 G[u][i].w-=res;
//                 G[v][rid].w+=res;
//                 return res;
//             }else dep[v]=-1;
//         }
//     }
//     return 0;
// }

int dfs(int u,int flow){
    if(u==t)return flow;
    int ans=0,res;
    for(int i=cur[u];i<G[u].size()&&flow>0;++i){
        cur[u]=i;
        int v=G[u][i].v,w=G[u][i].w,rid=G[u][i].rid;
        if(dep[v]==dep[u]+1&&w>0){
            res=dfs(v,min(flow,w));
            if(res==0)dep[v]=-1;
            G[u][i].w-=res;
            G[v][rid].w+=res;
            ans+=res;
            flow-=res;
        }
    }
    return ans;
}

int dinic(){
    int ans=0,res;
    while(bfs()){
        memset(cur,0,sizeof cur);
        while(res=dfs(s,INF))ans+=res;
    }
    return ans;
}

signed main(){
    cin>>n>>m>>s>>t;
    for(int i=1,u,v,w;i<=m;++i){
        cin>>u>>v>>w;
        add(u,v,w);
    }
    cout<<dinic()<<'\n';
}
```

最小割

最小割等于最大流

最小费用最大流

最大流情况下的最小费用（因为最大流的分配情况不唯一），实现就是用`EK`算法的基础将`bfs`改成最短路`spfa`

关键思想（`EK`之外的）

`dis`：当前增广路的单位费用和

`vis`：`spfa`算法中的入队判断

反向建边的单位费用为相反数

```
#include <bits/stdc++.h>
//ek中bfs改成spfa
using namespace std;

const int MAXN=5e3+10;
const int INF=0x3f3f3f3f;

int n,m,s,t,minc,maxf;
int flow[MAXN],lst[MAXN],pre[MAXN];
int dis[MAXN],vis[MAXN];

struct edge_t{
    int v,w,c,rid;
};

vector<edge_t> G[MAXN];

inline void add(int u,int v,int w,int c){
    int id=G[u].size();
    int rid=G[v].size();
    G[u].push_back({v,w,c,rid});
    G[v].push_back({u,0,-c,id});
}

bool spfa(){
    queue<int> q;
    memset(dis,0x3f,sizeof dis);
    memset(vis,0,sizeof vis);
    q.push(s);dis[s]=0;vis[s]=1;flow[s]=INF;
    while(!q.empty()){
        int u=q.front();q.pop();vis[u]=0;
        for(int i=0;i<G[u].size();++i){
            int v=G[u][i].v,w=G[u][i].w,c=G[u][i].c,rid=G[u][i].rid;
            if(dis[v]>dis[u]+c&&w>0){
                dis[v]=dis[u]+c;
                flow[v]=min(flow[u],w);
                lst[v]=u;
                pre[v]=i;
                if(!vis[v])q.push(v),vis[v]=1;
            }
```

```
        }
    }
    if(dis[t]==INF)return false;
    return true;
}

void mcmf(){
    while(spfa()){
        maxf+=flow[t];
        minc+=dis[t]*flow[t];
        for(int i=t;i!=s;i=lst[i]){
            int u=lst[i],v=i,rid=G[u][pre[v]].rid;
            G[u][pre[v]].w-=flow[t];
            G[v][rid].w+=flow[t];
        }
    }
}

int main(){
    cin>>n>>m>>s>>t;
    for(int i=1,u,v,w,c;i<=m;++i){
        cin>>u>>v>>w>>c;
        add(u,v,w,c);
    }
    mcmf();
    cout<<maxf<<' '<<minc<<'\n';
    return 0;
}
```

杂项

珂朵莉树

- $1\ l\ r\ x$：将$[l,r]$区间所有数加上x
- $2\ l\ r\ x$：将$[l,r]$区间所有数改成x
- $3\ l\ r\ x$：输出将$[l,r]$区间从小到大排序后的第x个数是的多少(即区间第x小，数字大小相同算多次，保证$1 \leq x \leq r-l+1$)
- $4\ l\ r\ x\ y$：输出$[l,r]$区间每个数字的x次方的和模y的值(即$(\sum_{i=l}^{r} a_i^x) \mod y$)

```
#include <bits/stdc++.h>
#define LL long long
#define int long long
using namespace std;

const int MAXN=1e5+2;
const int mod7=1e9+7;

int n,m,seed,vmax,a[MAXN];

int rnd(){
    int ret=seed;
    seed=(1ll*seed*7+13)%mod7;
    return ret;
```

```
}

int qpow(LL a,int b,int mod){
    int base=1ll*a%mod,res=1;
    for(;b;b>>=1,base=1ll*base*base%mod)if(b&1){
        res=1ll*res*base%mod;
    }
    return res%mod;
}

namespace odt{
struct node_t{
    int l,r;
    mutable LL v;
    node_t(int _l,int _r,LL _v):l(_l),r(_r),v(_v){}
    bool operator<(const node_t& o)const{return l<o.l;}
};

set<node_t> tree;

auto split(int x){
    auto it=tree.lower_bound(node_t{x,0,0});
    if(it!=tree.end()&&it->l==x)return it;
    --it;
    int l=it->l,r=it->r;
    LL v=it->v;
    tree.erase(it);
    tree.insert(node_t(l,x-1,v));
    return tree.insert(node_t(x,r,v)).first;
}

void assign(int l,int r,LL v){
    auto ed=split(r+1),op=split(l);
    tree.erase(op,ed);
    tree.insert(node_t(l,r,v));
}

void add(int l,int r,LL v){
    auto ed=split(r+1),op=split(l);
    for(auto it=op;it!=ed;++it){
        it->v+=v;
    }
}

int kth(int l,int r,int k){
    auto ed=split(r+1),op=split(l);
    vector<pair<int,int>> vec;
    for(auto it=op;it!=ed;++it){
        vec.push_back(make_pair(it->v,it->r-it->l+1));
    }
    sort(vec.begin(),vec.end());
    for(auto&& i:vec){
        k-=i.second;
        if(k<=0)return i.first;
    }
    return -1;
```

```
}

int sum_of_pow(int l,int r,int x,int y){
    auto ed=split(r+1),op=split(l);
    int res=0;
    for(auto it=op;it!=ed;++it){
        res=1ll*(1ll*res+1ll*qpow(it->v,x,y)*(it->r-it->l+1))%y;
    }
    return res;
}
}

void start(){
    for(int i=1;i<=n;++i){
        a[i]=(rnd()%vmax)+1;
        odt::tree.insert(odt::node_t(i,i,a[i]));
    }
    odt::tree.insert(odt::node_t(n+1,n+1,0));
}

signed main(){
    cin>>n>>m>>seed>>vmax;
    start();
    for(int i=1,op,l,r,x,y;i<=m;++i){
        op=(rnd()%4)+1;
        l=(rnd()%n)+1;
        r=(rnd()%n)+1;
        if(l>r)swap(l,r);

        if(op==3){
            x=(rnd()%(r-l+1))+1;
        }else{
            x=(rnd()%vmax)+1;
        }

        if(op==4){
            y=(rnd()%vmax)+1;
        }

        if(op==1){
            odt::add(l,r,x);
        }else if(op==2){
            odt::assign(l,r,x);
        }else if(op==3){
            cout<<odt::kth(l,r,x)<<'\n';
        }else if(op==4){
            cout<<odt::sum_of_pow(l,r,x,y)<<'\n';
        }
    }
}
```

普通莫队

1到N之间的格子，每个都有一个颜色c，区间[l, r]之间取两个格子颜色相同的概率

```
#include <bits/stdc++.h>

using namespace std;

#define int long long

int read(){
    int res=0,sign=1;char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-')sign=-sign;
    for(;ch>='0'&&ch<='9';ch=getchar())res=(res<<3)+(res<<1)+(ch^'0');
    return res*sign;
}

const int N=5e4+10;

int n,m,len;
int c[N],id[N];

struct query_t{
    int l,r,idx;
    bool operator<(const query_t& o)const{
        if(id[l]!=id[o.l])return l<o.l;
        return (id[l]&1)?r<o.r:r>o.r;
    }
};

int gcd(int a,int b){
    if(!b)return a;
    return gcd(b,a%b);
}

struct ans_t{
    int p,q;

    void setans(int _p,int _q){
        int _gcd=gcd(_p,_q);
        p=_p/_gcd,q=_q/_gcd;
    }
};

query_t q[N];
ans_t ans[N];

int sum,cnt[N];

void add(int i){
    sum+=cnt[i];
    ++cnt[i];
}

void del(int i){
```

```
    --cnt[i];
    sum-=cnt[i];
}

signed main(){
    // freopen("C:\\Users\\ASUS\\Downloads\\P1494_7.in","r",stdin);
    n=read(),m=read();
    len=sqrt(n);
    for(int i=1;i<=n;++i){
        c[i]=read();
        id[i]=(i-1)/len+1;
    }
    for(int i=1;i<=m;++i){
        q[i].l=read(),q[i].r=read(),q[i].idx=i;
    }
    sort(q+1,q+1+m);

    for(int i=1,l=1,r=0;i<=m;++i){
        int idx=q[i].idx;
        if(q[i].l==q[i].r){
            ans[idx].setans(0,1);
            continue;
        }
        while(l>q[i].l)add(c[--l]);
        while(r<q[i].r)add(c[++r]);
        while(l<q[i].l)del(c[l++]);
        while(r>q[i].r)del(c[r--]);
        ans[idx].setans(sum,(r-l+1)*(r-l)/2);
    }
    for(int i=1;i<=m;++i){
        printf("%lld/%lld\n",ans[i].p,ans[i].q);
    }
}
```

整体二分（仅静态区间第k小问题模板）

对于求静态区间第k小问题，我们有一种离线做法，叫做整体二分，代码如下：

（实际上树套树解决的问题如果是离线算法可做的，可以用到CDQ分治或者整体二分）

```
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)
```

```
const int N=2e5+10;

int n,m;

int a[N],rk[N],len;

void init(){
    rep(i,1,n)rk[i]=a[i];
    sort(rk+1,rk+1+n);
    len=unique(rk+1,rk+1+n)-rk-1;
}

int getidx(int x){
    return lower_bound(rk+1,rk+1+len,x)-rk;
}

int d[N];

void ins(int x,int c){
    for(;x<N;x+=x&-x){
        d[x]+=c;
    }
}

int query(int x){
    int res=0;
    for(;x;x-=x&-x){
        res+=d[x];
    }
    return res;
}

struct query_t{
    int l,r,k,id;
};

query_t q[N<<1],q1[N<<1],q2[N<<1];

int tot;

int ans[N];

void solve(int s,int t,int ql,int qr){
    if(ql>qr)return;
    if(s==t){
        rep(i,ql,qr)if(q[i].id)ans[q[i].id]=s;
        return;
    }
    int mid=s+t>>1,tot1=0,tot2=0;
    rep(i,ql,qr)if(!q[i].id){
        if(q[i].k<=mid){
            ins(q[i].l,1);
            q1[++tot1]=q[i];
        }else{
            q2[++tot2]=q[i];
        }
```

```
    }else{
        int x=query(q[i].r)-query(q[i].l-1);
        if(q[i].k<=x){
            q1[++tot1]=q[i];
        }else{
            q[i].k-=x;
            q2[++tot2]=q[i];
        }
    }

    rep(i,1,tot1)if(!q1[i].id)ins(q1[i].l,-1);
    rep(i,1,tot1)q[ql+i-1]=q1[i];
    rep(i,1,tot2)q[ql+tot1+i-1]=q2[i];
    solve(s,mid,ql,ql+tot1-1);
    solve(mid+1,t,ql+tot1,qr);
}

int main(){
    n=read(),m=read();
    rep(i,1,n){
        a[i]=read();
    }
    init();
    rep(i,1,n){
        q[++tot]={i,0,getidx(a[i]),0};
    }
    rep(i,1,m){
        int l=read(),r=read(),k=read();
        q[++tot]={l,r,k,i};
    }
    solve(1,len,1,tot);
    rep(i,1,m)printf("%d\n",rk[ans[i]]);
    return 0;
}
```

pbds中的平衡树

由于 pbds 中平衡树中元素不能重复，为了使得达到重复的效果，用 double 的模板然后包装一下，最后只要记得插入的时候投一个询问时间戳就好了

```
#include <cstdio>

using namespace std;

#include<ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
typedef
tree<double,null_type,less<double>,rb_tree_tag,tree_order_statistics_node_update>
rbt_t;
/*
insert(), erase(), lower_bound(), order_of_key(), find_by_order(), prev(), next()
*/

int read(){
    int res=0,sign=1;
```

```
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)

struct tree_t{
    rbt_t rbt;

    void ins(int x,int tim){
        rbt.insert(x+tim*1e-6);
    }

    void del(int x){
        rbt.erase(rbt.lower_bound(x));
    }

    int rk(int x){
        return (int)rbt.order_of_key(x)+1;
    }

    int kth(int k){
        return (int)*rbt.find_by_order(k-1);
    }

    int pre(int x){
        return (int)round(*(--rbt.lower_bound(x)));
    }

    int nxt(int x){
        return (int)round(*rbt.lower_bound(x + 1));
    }
};

tree_t T;

int q;

int main(){
    q=read();
    rep(i,1,q){
        int opt=read(),x=read();
        if(opt == 1)
            T.ins(x,i);
        if(opt == 2)
            T.del(x);
        if(opt == 3)
            printf("%d\n",T.rk(x));
        if(opt == 4)
            printf("%d\n",T.kth(x));
        if(opt == 5)
            printf("%d\n",T.pre(x));
        if(opt == 6)
```

```
            printf("%d\n",T.nxt(x));
    }
    return 0;
}
```

int128 的使用

```
typedef __int128_t lll;

lll read(){
    lll res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

inline void write(lll x){
    if(x<0)putchar('-'),x=-x;
    if(x>9)write(x/10);
    putchar(x%10+'0');
}
```

pair<int,int> 作为键值的 unordered_map

```
struct pair_hash{
    template<class T1,class T2>
    std::size_t operator()(const std::pair<T1,T2>& p)const{
        auto h1=std::hash<T1>{}(p.first);
        auto h2=std::hash<T2>{}(p.second);
        return h1^h2;
    }
};

unordered_map<pair<int,int>,int,pair_hash> mp;
```

计算几何

二维计算几何

```
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
```

```
#define dep(i,r,l) for(int i=r;i>=l;--i)

const double eps=1e-9;
const double inf=1e20;
const double pi=acos(-1.0);
const int maxp=1010;

int sgn(double x){
    if(fabs(x)<eps)return 0;
    if(x<0)return -1;
    else return 1;
}

inline double sqr(double x){return x*x;}

struct Point{
    double x,y;
    Point(){}
    Point(double _x,double _y){x=_x,y=_y;}
    bool operator==(const Point& b)const{return sgn(x-b.x)==0&&sgn(y-b.y)==0;}
    bool operator<(const Point& b)const{return sgn(x-b.x)==0?sgn(y-b.y)<0:x<b.x;}
    Point operator-(const Point& b)const{return Point(x-b.x,y-b.y);}
    double operator^(const Point& b)const{return x*b.y-y*b.x;}
    double operator*(const Point& b)const{return x*b.x+y*b.y;}
    double len()const{return hypot(x,y);}
    double len2()const{return x*x+y*y;}
    double distance(const Point& p)const{return hypot(x-p.x,y-p.y);}
    Point operator+(const Point& b)const{return Point(x+b.x,y+b.y);}
    Point operator*(const double& k)const{return Point(x*k,y*k);}
    Point operator/(const double& k)const{return Point(x/k,y/k);}
    // pa 与 pb 的夹角
    double rad(const Point& a,const Point& b)const{
        Point p=*this;
        return fabs(atan2(fabs((a-p)^(b-p)),(a-p)*(b-p)));
    }
    // 转化为长度为 r 的向量
    Point trunc(double r)const{
        double l=len();
        if(!sgn(l))return *this;
        r/=l;
        return Point(x*r,y*r);
    }
    // 逆时针旋转 90 °
    Point rotleft()const{
        return Point(-y,x);
    }
    // 顺时针旋转 90 °
    Point rotright()const{
        return Point(y,-x);
    }
    // 绕 p 逆时针旋转 angle
    Point rotate(const Point& p,double angle)const{
        Point v=*this-p;
        double c=cos(angle),s=sin(angle);
        return Point(p.x+v.x*c-v.y*s,p.y+v.x*s+v.y*c);
    }
```

```
};

struct Line{
    Point s,e;
    Line(){}
    Line(Point _s,Point _e){s=_s,e=_e;}
    bool operator==(const Line& v){return (s==v.s)&&(e==v.e);}
    // 点斜式 0<=angle<pi
    Line(Point p,double angle){
        s=p;
        if(sgn(angle-pi/2)==0)e=(s+Point(0,1));
        else e=s+Point(1,tan(angle));
    }
    // 标准式 ax+by+c=0
    Line(double a,double b,double c){
        if(sgn(a)==0){
            s=Point(0,-c/b);
            e=Point(1,-c/b);
        }else if(sgn(b)==0){
            s=Point(-c/a,0);
            e=Point(-c/a,1);
        }else{
            s=Point(0,-c/b);
            e=Point(1,(-c-a)/b);
        }
    }
    void adjust(){
        if(e<s)swap(s,e);
    }
    double length()const{
        return s.distance(e);
    }
    // 0<=angle<pi
    double angle()const{
        double k=atan2(e.y-s.y,e.x-s.x);
        if(sgn(k)<0)k+=pi;
        if(sgn(k-pi)==0)k-=pi;
        return k;
    }
    // 点与直线关系
    // 1：在左侧 2：在右侧 3：在直线上
    int relation(const Point& p)const{
        int c=sgn((p-s)^(e-s));
        if(c<0)return 1;
        else if(c>0)return 2;
        else return 3;
    }
    // 点在线段上的判断
    bool pointonseg(const Point& p)const{
        return sgn((p-s)^(e-s))==0&&sgn((p-s)*(p-e))<=0;
    }
    // 判断方向平行（直线平行或者重合）
    bool parallel(const Line& v)const{
        return sgn((e-s)^(v.e-v.s))==0;
    }
    // 线段相交判断
```

```
    // 2：规范相交 1：非规范相交 0：不相交
    int segcrossseg(const Line& v)const{
        int d1=sgn((e-s)^(v.s-s));
        int d2=sgn((e-s)^(v.e-s));
        int d3=sgn((v.e-v.s)^(s-v.s));
        int d4=sgn((v.e-v.s)^(e-v.s));
        if((d1^d2)==-2&&(d3^d4)==-2)return 2;
        return (d1==0&&sgn((v.s-s)*(v.s-e))<=0)||
            (d2==0&&sgn((v.e-s)*(v.e-e))<=0)||
            (d3==0&&sgn((s-v.s)*(s-v.e))<=0)||
            (d4==0&&sgn((e-v.s)*(e-v.e))<=0);
    }
    // 直线线段相交判断
    // *this：直线 v：线段
    // 2：规范相交 1：非规范相交 0：不相交
    int linecrossseg(const Line& v)const{
        int d1=sgn((e-s)^(v.s-s));
        int d2=sgn((e-s)^(v.e-s));
        if((d1^d2)==-2)return 2;
        return (d1==0||d2==0);
    }
    // 直线相交判断
    // 0：平行 1：重合 2：相交
    int linecrossline(const Line& v)const{
        if((*this).parallel(v))return v.relation(s)==3;
        return 2;
    }
    // 求两直线交点
    // 要求不平行或重合
    Point crosspoint(const Line& v)const{
        double a1=(v.e-v.s)^(s-v.s);
        double a2=(v.e-v.s)^(e-v.s);
        return Point((s.x*a2-e.x*a1)/(a2-a1),
                (s.y*a2-e.y*a1)/(a2-a1));
    }
    // 点到直线距离
    double dispointtoline(const Point& p)const{
        return fabs((p-s)^(e-s))/length();
    }
    // 点到线段距离
    double dispointtoseg(const Point& p)const{
        if(sgn((p-s)*(e-s))<0||sgn((p-e)*(s-e))<0)
            return min(p.distance(s),p.distance(e));
        return dispointtoline(p);
    }
    // 线段到线段距离
    // 线段相交的结果为 0
    double dissegtoseg(const Line& v)const{
        return min({dispointtoseg(v.s),dispointtoseg(v.e),
                    v.dispointtoseg(s),v.dispointtoseg(e)});
    }
    // 点 p 在直线上的投影
    Point lineprog(const Point& p)const{
        return s+(((e-s)*((e-s)*(p-s)))/((e-s).len2()));
    }
    // 点 p 关于直线的对称点
```

```
    Point symmetrypoint(const Point& p)const{
        Point q=lineprog(p);
        return Point(2*q.x-p.x,2*q.y-p.y);
    }
};

struct circle{
    Point p;
    double r;
    circle(){}
    circle(Point _p,double _r){p=_p,r=_r;}
    circle(double x,double y,double _r){p=Point(x,y),r=_r;}
    // 三角形外接圆：1 内切圆：2
    circle(Point a,Point b,Point c,int op){
        if(op==1){
            Line u=Line((a+b)/2,((a+b)/2+((b-a).rotleft())));
            Line v=Line((b+c)/2,((b+c)/2+((c-b).rotleft())));
            p=u.crosspoint(v);
            r=p.distance(a);
        }else if(op==2){
            Line u,v;
            double m=atan2(b.y-a.y,b.x-a.x),n=atan2(c.y-a.y,c.x-a.x);
            u.s=a;
            u.e=u.s+Point(cos((n+m)/2),sin((n+m)/2));
            v.s=b;
            m=atan2(a.y-b.y,a.x-b.x),n=atan2(c.y-b.y,c.x-b.x);
            v.e=v.s+Point(cos((n+m)/2),sin((n+m)/2));
            p=u.crosspoint(v);
            r=Line(a,b).dispointtoseg(p);
        }
    }
    bool operator==(const circle& v)const{return (p==v.p)&&sgn(r-v.r)==0;}
    bool operator<(const circle& v)const{return ((p<v.p)||((p==v.p)&&sgn(r-v.r)
<0));}
    double area()const{return pi*r*r;}
    double circumference()const{return 2*pi*r;}
    // 点与圆的关系
    // 0：圆外 1：圆上 2：圆内
    int relation(const Point& b)const{
        double dst=b.distance(p);
        if(sgn(dst-r)<0)return 2;
        else if(sgn(dst-r)==0)return 1;
        else return 0;
    }
    // 线段与圆的关系
    // 比的是圆心与线段的距离与半径的大小关系
    int relationseg(const Line& v)const{
        double dst=v.dispointtoseg(p);
        if(sgn(dst-r)<0)return 2;
        else if(sgn(dst-r)==0)return 1;
        else return 0;
    }
    // 直线与圆的关系
    // 比的是圆心与直线的距离与半径的大小关系
    int relationline(const Line& v)const{
        double dst=v.dispointtoline(p);
```

```
        if(sgn(dst-r)<0)return 2;
        else if(sgn(dst-r)==0)return 1;
        else return 0;
    }
    // 两圆的关系
    // 5：相离 4：外切 3：相交 2：内切 1：内含
    int relationcircle(const circle& v)const{
        double d=p.distance(v.p);
        if(sgn(d-r-v.r)>0)return 5;
        if(sgn(d-r-v.r)==0)return 4;
        double l=fabs(r-v.r);
        if(sgn(d-r-v.r)<0&&sgn(d-l)>0)return 3;
        if(sgn(d-l)==0)return 2;
        if(sgn(d-l)<0)return 1;
    }
    // 求两圆交点
    // 0：无交点 1：一个交点 2：两个交点
    int pointcrosscircle(const circle& v,Point& p1,Point& p2)const{
        int rel=relationcircle(v);
        if(rel==1||rel==5)return 0;
        double d=p.distance(v.p);
        double l=(d*d+r*r-v.r*v.r)/(2*d);
        double h=sqrt(r*r-l*l);
        Point tmp=p+(v.p-p).trunc(l);
        p1=tmp+((v.p-p).rotleft().trunc(h));
        p2=tmp+((v.p-p).rotright().trunc(h));
        if(rel==2||rel==4)
            return 1;
        return 2;
    }
    // 求直线与圆交点
    // 0：无交点 1：一个交点 2：两个交点
    int pointcrossline(const Line& v,Point& p1,Point& p2)const{
        if(!(*this).relationline(v))return 0;
        Point a=v.lineprog(p);
        double d=v.dispointtoline(p);
        d=sqrt(r*r-d*d);
        if(sgn(d)==0){
            p1=a;
            p2=a;
            return 1;
        }
        p1=a+(v.e-v.s).trunc(d);
        p2=a-(v.e-v.s).trunc(d);
        return 2;
    }
    // 得到过 a，b 两点，半径为 r1 的两个圆
    int gercircle(const Point& a,const Point& b,double r1,circle& c1,circle&
c2)const{
        circle x(a,r1),y(b,r1);
        int t=x.pointcrosscircle(y,c1.p,c2.p);
        if(!t)return 0;
        c1.r=c2.r=r;
        return t;
    }
    // 得到和直线 u 相切，过点 q，半径 r1 的圆
```

```
    int getcircle(const Line& u,const Point& q,double r1,circle &c1,circle
&c2)const{
        double dis=u.dispointtoline(q);
        if(sgn(dis-r1*2)>0)return 0;
        if(sgn(dis)==0){
            c1.p=q+((u.e-u.s).rotleft().trunc(r1));
            c2.p=q+((u.e-u.s).rotright().trunc(r1));
            c1.r=c2.r=r1;
            return 2;
        }
        Line u1=Line((u.s+(u.e-u.s).rotleft().trunc(r1)),(u.e+
            (u.e-u.s).rotleft().trunc(r1)));
        Line u2=Line((u.s+(u.e-u.s).rotright().trunc(r1)),(u.e
            +(u.e-u.s).rotright().trunc(r1)));
        circle cc=circle(q,r1);
        Point p1,p2;
        if(!cc.pointcrossline(u1,p1,p2))cc.pointcrossline(u2,p1,p2);
        c1=circle(p1,r1);
        if(p1==p2){
            c2=c1;
            return 1;
        }
        c2=circle(p2,r1);
        return 2;
    }
    // 同时与直线 u，v 相切，半径为 r1 的圆
    int getcircle(const Line& u,const Line& v,double r1,circle& c1,circle&
c2,circle& c3,circle& c4)const{
        if(u.parallel(v))return 0;
        Line u1=Line(u.s+(u.e-u.s).rotleft().trunc(r1),u.e+(u.e-
u.s).rotleft().trunc(r1));
        Line u2=Line(u.s+(u.e-u.s).rotright().trunc(r1),u.e+(u.e-
u.s).rotright().trunc(r1));
        Line v1=Line(v.s+(v.e-v.s).rotleft().trunc(r1),v.e+(v.e-
v.s).rotleft().trunc(r1));
        Line v2=Line(v.s+(v.e-v.s).rotright().trunc(r1),v.e+(v.e-
v.s).rotright().trunc(r1));
        c1.r=c2.r=c3.r=c4.r=r1;
        c1.p=u1.crosspoint(v1);
        c2.p=u1.crosspoint(v2);
        c3.p=u2.crosspoint(v1);
        c4.p=u2.crosspoint(v2);
        return 4;
    }
    // 同时与不相交圆 cx，cy相切，半径为 r1 的圆
    int getcircle(const circle& cx,const circle& cy,double r1,circle &c1,circle
&c2)const{
        circle x(cx.p,r1+cx.r),y(cy.p,r1+cy.r);
        int t=x.pointcrosscircle(y,c1.p,c2.p);
        if(!t)return 0;
        c1.r=c2.r=r1;
        return t;
    }
    // 过一点做圆的切线
    int tangentline(const Point& q,Line &u,Line &v)const{
        int x=relation(q);
```

```
        if(x==2)return 0;
        if(x==1){
            u=Line(q,q+(q-p).rotleft());
            v=u;
            return 1;
        }
        double d=p.distance(q);
        double l=r*r/d;
        double h=sqrt(r*r-l*l);
        u=Line(q,p+((q-p).trunc(l)+(q-p).rotleft().trunc(h)));
        v=Line(q,p+((q-p).trunc(l)+(q-p).rotright().trunc(h)));
        return 2;
    }
    // 求两圆相交的面积
    double areacircle(const circle& v)const{
        int rel=relationcircle(v);
        if(rel>=4)return 0.0;
        if(rel<=2)return min(area(),v.area());
        double d=p.distance(v.p);
        double hf=(r+v.r+d)/2.0;
        double ss=2*sqrt(hf*(hf-r)*(hf-v.r)*(hf-d));
        double a1=acos((r*r+d*d-v.r*v.r)/(2.0*r*d));
        a1=a1*r*r;
        double a2=acos((v.r*v.r+d*d-r*r)/(2.0*v.r*d));
        a2=a2*v.r*v.r;
        return a1+a2-ss;
    }
    // 求圆和三角形 pab 的相交面积（ p 是圆点）
    double areatriangle(const Point& a,const Point& b)const{
        if(sgn((p-a)^(p-b))==0)return 0.0;
        Point q[5];
        int len=0;
        q[len++]=a;
        Line l(a,b);
        Point p1,p2;
        if(pointcrossline(l,q[1],q[2])==2){
            if(sgn((a-q[1])*(b-q[1]))<0)q[len++]=q[1];
            if(sgn((a-q[2])*(b-q[2]))<0)q[len++]=q[2];
        }
        q[len++]=b;
        if(len==4&&sgn((q[0]-q[1])*(q[2]-q[1]))>0)swap(q[1],q[2]);
        double res=0;
        for(int i=0;i<len-1;++i){
            if(relation(q[i])==0||relation(q[i+1])==0){
                double arg=p.rad(q[i],q[i+1]);
                res+=r*r*arg/2.0;
            }else{
                res+=fabs((q[i]-p)^(q[i+1]-p))/2.0;
            }
        }
        return res;
    }
};

struct polygon{
    // 0 ~ n - 1
```

```
    int n;
    Point p[maxp];
    Line l[maxp];
    void add(const Point& q){
        p[n++]=q;
    }
    void getline(){
        for(int i=0;i<n;++i){
            l[i]=Line(p[i],p[(i+1)%n]);
        }
    }
    // 极角排序结构体函数
    struct cmp{
        Point p;
        cmp(const Point& p0){p=p0;}
        bool operator()(const Point& aa,const Point& bb)const{
            Point a=aa,b=bb;
            int d=sgn((a-p)^(b-p));
            if(d==0){
                return sgn(a.distance(p)-b.distance(p))<0;
            }
            return d>0;
        }
    };
    // 极角排序
    void norm(){
        Point mi=p[0];
        for(int i=1;i<n;++i)mi=min(mi,p[i]);
        sort(p,p+n,cmp(mi));
    }
    // 获得凸包
    void getconvex(polygon& convex){
        sort(p,p+n);
        convex.n=n;
        for(int i=0;i<min(n,2);++i){
            convex.p[i]=p[i];
        }
        if(convex.n==2&&(convex.p[0]==convex.p[1]))--convex.n;
        if(n<=2)return;
        int& top=convex.n;
        top=1;
        for(int i=2;i<n;++i){
            while(top&&sgn((convex.p[top]-p[i])^(convex.p[top-1]-p[i]))<=0)--top;
            convex.p[++top]=p[i];
        }
        int temp=top;
        convex.p[++top]=p[n-2];
        for(int i=n-3;i>=0;--i){
            while(top!=temp&&sgn((convex.p[top]-p[i])^(convex.p[top-1]-p[i]))
<=0)--top;
            convex.p[++top]=p[i];
        }
        if(convex.n==2&&(convex.p[0]==convex.p[1]))--convex.n;
        convex.norm();
    }
    // 求凸包的另一个方法
```

```
    void Graham(polygon& convex){
        norm();
        int& top=convex.n;
        top=0;
        if(n==1){
            top=1;
            convex.p[0]=p[0];
            return;
        }
        if(n==2){
            top==2;
            convex.p[0]=p[0];
            convex.p[1]=p[1];
            if(convex.p[0]==convex.p[1])--top;
            return;
        }
        convex.p[0]=p[0];
        convex.p[1]=p[1];
        top=2;
        for(int i=2;i<n;++i){
            while(top>1&&sgn((convex.p[top-1]-convex.p[top-2])^(p[i]-
convex.p[top-2]))<=0)--top;
            convex.p[top++]=p[i];
        }
        if(convex.n==2&&(convex.p[0]==convex.p[1]))--convex.n;
    }
    // 判断是不是凸多边形
    bool isconvex()const{
        bool s[2];
        memset(s,false,sizeof(s));
        for(int i=0;i<n;++i){
            int j=(i+1)%n;
            int k=(j+1)%n;
            s[sgn((p[j]-p[i])^(p[k]-p[i]))+1]=true;
            if(s[0]&&s[2])return false;
        }
        return true;
    }
    // 判断点和任意多边形的关系
    // 3：点上 2：边上 1：内部 0：外部
    int relationpoint(const Point& q){
        for(int i=0;i<n;++i){
            if(p[i]==q)return 3;
        }
        getline();
        for(int i=0;i<n;++i){
            if(l[i].pointonseg(q))return 2;
        }
        int cnt=0;
        for(int i=0;i<n;++i){
            int j=(i+1)%n;
            int k=sgn((q-p[j])^(p[i]-p[j]));
            int u=sgn(p[i].y-q.y);
            int v=sgn(p[j].y-q.y);
            if(k>0&&u<0&&v>=0)++cnt;
            if(k<0&&v<0&&u>=0)--cnt;
```

```
        }
        return cnt!=0;
    }
    // 直线 u 切割凸多边形左侧（取出左侧）
    void convexcut(const Line& u,polygon &po){
        int &top=po.n;
        top=0;
        for(int i=0;i<n;++i){
            int d1=sgn((u.e-u.s)^(p[i]-u.s));
            int d2=sgn((u.e-u.s)^(p[(i+1)%n]-u.s));
            if(d1>=0)po.p[top++]=p[i];
            if(d1*d2<0)po.p[top++]=u.crosspoint(Line(p[i],p[(i+1)%n]));
        }
    }
    // 周长
    double getcircumference(){
        double sum=0;
        for(int i=0;i<n;++i){
            sum+=p[i].distance(p[(i+1)%n]);
        }
        return sum;
    }
    // 面积
    double getarea(){
        double sum=0;
        for(int i=0;i<n;++i){
            sum+=(p[i]^p[(i+1)%n]);
        }
        return fabs(sum)/2;
    }
    // 旋转卡壳求凸包的直径
    // 必须是凸多边形才能用下面的旋转卡壳
    double rotatingcalipers(){
        if(n==2)return p[0].distance(p[1]);
        int cur=0;
        double ans=0;
        for(int i=0;i<n;++i){
            Line line(p[i],p[(i+1)%n]);
            while(line.dispointtoline(p[cur])<=line.dispointtoline(p[(cur+1)%n]))
{
                cur=(cur+1)%n;
            }
            ans=max(ans,max(p[cur].distance(p[i]),p[cur].distance(p[(i+1)%n])));
        }
        return ans;
    }
    // ...更多见 kuangbin ACM 模板
};

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

线段树优化动态规划

问题：给定序列$a_1, a_2, \ldots, a_n$，给定集合s（大小不超过10），划分区间$[1, n]$后满足每个子区间中，每种数的个数都不属于集合s的总方案数是多少？（模998244353）

分析：由于集合s大小很小，可以考虑对于某个特定的数x怎么做，然后集合中每个数都同样处理即可。假如$dp[i]$表示$[1, i]$的方案数，那么考虑$0 <= j <= i-1$中，哪些j是可以转移过来答案的，很明显是$[j+1, i]$中每种数的个数都不等于x的这种j满足转移，此时做转移$dp[i] += dp[j]$。具体实现是：考虑$j == 0$和$1 <= j <= i-1$两种贡献，第一种就直接维护cnt为$[1, i]$之间多少种数的个数属于集合s，只有当$cnt == 0$时，才能从$j == 0$转移，也就是$dp[i] += 1$；第二种贡献就是考虑记录$[1, i]$中每种数它的序列索引在一个$vector < int > pos$中，对于$a[i]$的那个pos数组来说，那么$j = [pos[siz - x - 1], pos[siz - x] - 1]$时，$[j+1, i]$这个区间中就有$a[i]$个数是$x$个，其余枚举的$j$的区间$[j+1, i]$中都不是$x$个，那么就在线段树上打标记（先删除上次对于$a[i]$这种数打的标记），统计所有可以转移的位置的$dp$值之和（可以转移的位置就是未被打标记的位置，这个可以用区间加区间查询最小值线段树来做），加在$dp[i]$上，最后的答案就是$dp[n]$

代码如下：

```
#include <bits/stdc++.h>

using namespace std;

int read(){
    int res=0,sign=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-'){sign=-sign;}
    for(;ch>='0'&&ch<='9';ch=getchar()){res=(res<<3)+(res<<1)+(ch^'0');}
    return res*sign;
}

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define dep(i,r,l) for(int i=r;i>=l;--i)

const int N=2e5+10;
const int mod=998244353;
const int INF=0x3f3f3f3f;

int n,m;

int a[N];

set<int> S;

struct node_t{
    int mn,tag,cnt,val;
    #define ls (p<<1)
    #define rs (p<<1|1)
    #define MID int m=s+((t-s)>>1)
};

node_t d[N<<2];

node_t merge(const node_t& a,const node_t& b){
    int mn=min(a.mn,b.mn),cnt=0,val=0;
    if(a.mn==mn)cnt+=a.cnt,val=(1ll*val+a.val)%mod;
```

```
    if(b.mn==mn)cnt+=b.cnt,val=(1ll*val+b.val)%mod;
    return node_t({mn,0,cnt,val});
}

void pushup(int p){
    d[p]=merge(d[ls],d[rs]);
}

void pushdown(int p,int s,int t){
    if(s!=t&&d[p].tag){
        d[ls].mn+=d[p].tag,d[rs].mn+=d[p].tag;
        d[ls].tag+=d[p].tag,d[rs].tag+=d[p].tag;
        d[p].tag=0;
    }
}

void build(int p,int s,int t){
    if(s==t){
        d[p].mn=0,d[p].cnt=1,d[p].val=0;
        return;
    }
    MID;
    pushdown(p,s,t);
    build(ls,s,m);
    build(rs,m+1,t);
    pushup(p);
}

void modify(int p,int s,int t,int l,int r,int c){
    if(l>r)return;
    if(l<=s&&t<=r){
        d[p].mn+=c,d[p].tag+=c;
        return;
    }
    MID;
    pushdown(p,s,t);
    if(l<=m)modify(ls,s,m,l,r,c);
    if(r>m)modify(rs,m+1,t,l,r,c);
    pushup(p);
}

void add(int p,int s,int t,int x,int c){
    if(s==t){
        d[p].val=(1ll*d[p].val+c)%mod;
        return;
    }
    MID;
    if(x<=m)add(ls,s,m,x,c);
    else add(rs,m+1,t,x,c);
    pushup(p);
}

node_t query(int p,int s,int t,int l,int r){
    if(l>r)return {INF,0,0};
    if(l<=s&&t<=r){
        return d[p];
```

```
    }
    MID;
    pushdown(p,s,t);
    if(l<=m&&r>m)return merge(query(ls,s,m,l,r),query(rs,m+1,t,l,r));
    else if(l<=m)return query(ls,s,m,l,r);
    else return query(rs,m+1,t,l,r);
}

int getans(int p,int s,int t,int x){
    if(s==t)return d[p].val;
    MID;
    if(x<=m)return getans(ls,s,m,x);
    else return getans(rs,m+1,t,x);
}


int getpos(const vector<int>& pos,int i){
    if(i<0)return 1;
    return pos[i];
}

int main(){
    n=read(),m=read();
    rep(i,1,n)a[i]=read();
    rep(i,1,m)S.insert(read());

    build(1,1,n);
    vector<vector<int>> mp(n+1);

    int cnt=0;

    if(!S.count(1)){
        add(1,1,n,1,1);
    }else ++cnt;

    mp[a[1]].push_back(1);

    rep(i,2,n){
        auto& pos=mp[a[i]];
        pos.push_back(i);
        int siz=pos.size();
        for(auto&& c:S){
            modify(1,1,n,getpos(pos,siz-c-2),getpos(pos,siz-c-1)-1,-1);
            modify(1,1,n,getpos(pos,siz-c-1),getpos(pos,siz-c)-1,1);
        }
        auto ans=query(1,1,n,1,i);

        // printf("ans: %d %d %d\n",i,ans.mn,ans.val);

        if(ans.mn==0){
            add(1,1,n,i,ans.val);
        }
        for(auto&& c:S){
            if(siz==c)++cnt;
            else if(siz==c+1)--cnt;
        }
```

```
        // printf("cnt: %d\n",cnt);

        if(cnt==0)add(1,1,n,i,1);

        // printf("dp_i: %d\n",getans(1,1,n,i));
    }
    int ans=getans(1,1,n,n);
    printf("%d\n",ans);
    return 0;
}
```