# Template of Team DFA

HPU : Codancer & Dicer

2019 年 9 月 26 日

# 目录

# 1 杂项

## 1.1 Head & 快速读入

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N = 1e6+100;
const int mod = 1e9+7;
typedef long long ll;
const int INF = 0x3f3f3f3f;
const ll llINF = 0x3f3f3f3f3f3f3f3f;
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define fep(i,a,b) for(int i=(a);i>=(b);i--)
inline bool read(ll &num) {
    char in;bool IsN=false;
    in=getchar();
    if(in==EOF) return false;
    while(in!='-'&&(in<'0'||in>'9')) in=getchar();
    if(in=='-'){ IsN=true;num=0;}
    else num=in-'0';
    while(in=getchar(),in>='0'&&in<='9'){
           num*=10,num+=in-'0';
    }
    if(IsN) num=-num;
    return true;
}
```

## 1.2 ___int128 输入输出

```cpp
void scan(__int128 &x)//输入{
    x = 0;
    int f = 1;
    char ch;
    if((ch = getchar()) == '-') f = -f;
    else x = x*10 + ch-'0';
    while((ch = getchar()) >= '0' && ch <= '9')
        x = x*10 + ch-'0';
    x *= f;
}
void print(__int128 x)//输出
{
    if(x < 0)
    {
        x = -x;
        putchar('-');
    }
     if(x > 9) print(x/10);
    putchar(x%10 + '0');
}
```

## 1.3 O3 优化

```cpp
#pragma GCC optimize(3,"Ofast","inline")
```

## 1.4 单调栈

求第 i 个数作为最大值的区间 [1,n]

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

using namespace std;
const int N = 1e6+100;
int a[N];
int L[N],R[N];
int main(){
    int n;
    cin>>n;
    stack<int> sta;
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    for(int i=1;i<=n;i++){
        while(sta.size()&&a[sta.top()]<=a[i]) sta.pop
            ();
        if(sta.empty()) L[i]=1;
        else L[i]=sta.top()+1;
        sta.push(i);
    }
    while(sta.size()) sta.pop();
    for(int i=n;i>=1;i--){
        while(sta.size()&&a[sta.top()]<=a[i]) sta.pop
            ();
        if(sta.empty()) R[i]=n;
        else R[i]=sta.top()-1;
        sta.push(i);
    }
```

## 1.5 打印 LIS

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

using namespace std;
const int N = 60000;
const int INF = 0X3f3f3f3f;
int dp[N],fa[N],a[N],b[N],order[N],n,pos[N];
bool vis[N],ok[N];
int solve(){//求b的LIS
    int cnt=0;
    for(int i=1;i<=n;i++){
        if(!vis[i]) b[cnt++]=a[i];//b数组
    }
    memset(dp,INF,sizeof(dp));
    memset(ok,0,sizeof(ok));
    int lpos;
    pos[0]=-1;
    for(int i=0;i<cnt;i++){
        dp[lpos=(lower_bound(dp,dp+cnt,b[i])-dp)]=b[i
            ];
        pos[lpos]=i;
        fa[i]=(lpos?pos[lpos-1]:-1);
    }
    cnt=lower_bound(dp,dp+cnt,INF)-dp;
    int i;
    for(i=pos[cnt-1];~fa[i];i=fa[i]){
         ok[b[i]]=1;//说明b[i]在LIS内
    }
    ok[b[i]]=1;
    return cnt;
}
```

# 2 图论

## 2.1 Dinic 最大流

```cpp
#include<iostream>
```

```cpp
#include<algorithm>
#include<string.h>
#include<cstdio>
#include<queue>
using namespace std;
const int INF = 0x3f3f3f3f;
typedef long long ll;



//Dinic最大流, 节点编号从0开始
struct MaxFlow{
    const static ll MAX_V = 1005;
    ll V;
    //终点、容量、反向边
    struct edge{
        ll to, cap, rev;
    };
    vector<edge> G[MAX_V];
    ll level[MAX_V];//顶点到源点的距离标号
    ll iter[MAX_V];// 当前弧, 在其之前的边已经没有用了

    void add_edge(ll from, ll to, ll cap){
        G[from].push_back((edge){to, cap, (ll)G[to].
            size()});
        G[to].push_back((edge){from, 0, (ll)G[from].
            size()-1});
    }

    // 通过BFS计算从源点出发的距离标号
    void bfs(ll s){
        fill(level, level + V, -1);
        queue<ll> que;
        level[s] = 0;
        que.push(s);
        while (!que.empty()){
            ll v = que.front();
            que.pop();
            for (ll i=0; i< G[v].size(); i++){
                edge& e = G[v][i];
                if (e.cap > 0 && level[e.to] < 0){
                    level[e.to] = level[v] + 1;
                    que.push(e.to);
                }
            }
        }
    }

    //通过DFS寻找增广路
    ll dfs(ll v, ll t, ll f){
        if (v == t)
            return f;
        for (ll &i = iter[v]; i < G[v].size(); i++){
            edge& e = G[v][i];
            if (e.cap > 0 && level[v] < level[e.to]){
                ll d = dfs(e.to, t, min(f, e.cap));
                if (d > 0){
                    e.cap -= d;
                    G[e.to][e.rev].cap += d;
                    return d;
                }
            }
        }
        return 0;
    }



    //求解从s到t的最大流
    ll max_flow(ll s, ll t){
        ll flow = 0;
        for (;;){
            bfs(s);
            if(level[t] < 0)
                return flow;
            fill(iter, iter + V, 0);
            ll f;
            while ((f = dfs(s, t, INF)) > 0){
                flow += f;
            }
        }
    }



    void init(ll n = 0){
        for (ll i = 0; i < V; i++){
            G[i].clear();
        }
        V = n;
    }
}mf;
```

## 2.2 倍增求解 LCA

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

using namespace std;
const int N = 1e6+100;
vector<int> G[N];
long long bit[30];
int f[N][30];
int depth[N];
void init(){
    bit[0]=1;
    for(int i=1;i<=29;i++) bit[i]=(bit[i-1]<<1);
}
void dfs(int u,int par){
    depth[u]=depth[par]+1;
    f[u][0]=par;
    for(int i=1;bit[i]<=depth[u];i++) f[u][i]=f[f[u][i
        -1]][i-1];
    for(int i=0;i<(int)G[u].size();i++){
        int v=G[u][i];
        if(v!=par) dfs(v,u);
    }
}
int lca(int x,int y){
    if(depth[x]<depth[y]) swap(x,y);
    for(int i=29;i>=0;i--){
        if(depth[x]-depth[y]>=bit[i]){
            x=f[x][i];
        }
    }
    if(x==y) return x;
    for(int i=29;i>=0;i--){
        if(depth[x]>=(1<<i)&&f[x][i]!=f[y][i]){
            x=f[x][i];
```

```
            y=f[y][i];
        }
    }
    return f[x][0];
}
```

## 2.3 有向图最小环

```
rep(k,1,n){
    rep(i,1,k-1){
        rep(j,1,i-1){
            ans=min(ans,dis[i][j]+val[i][k]+val[k][j]);
                //val代表边权
        }
    }
    rep(i,1,n){
        rep(j,1,n){
            dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j
                ]);
        }
    }
}
```

## 2.4 Tarjan

### 2.4.1 缩点求 SCC

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

using namespace std;
const int N = 1e5+100;
typedef long long ll;
const int INF = 0x3f3f3f3f;
int n,m,scc,index;
vector<int> G[N];
ll w[N],low[N],dfn[N],minn[N],color[N],id[N];
bool is_instack[N];stack<int> sta;
//不要忘记初始化!!!!
void init(){
    scc=index=0;
    memset(low,0,sizeof(low));
    memset(dfn,0,sizeof(dfn));
    memset(color,0,sizeof(color));
    memset(minn,INF,sizeof(minn));
    memset(is_instack,0,sizeof(is_instack));
    for(int i=1;i<=n;i++) G[i].clear();
    while(!sta.empty()) sta.pop();
}
void Tarjan(int u){
    low[u]=dfn[u]=++index;
    sta.push(u);is_instack[u]=1;
    for(auto v:G[u]){
        if(!dfn[v]){
            Tarjan(v);
            low[u]=min(low[u],low[v]);
        }
        else if(is_instack[v]){
            low[u]=min(low[u],dfn[v]);
        }
    }
    if(low[u]==dfn[u]){
        ++scc;
        while(1){
            int temp=sta.top();
            color[temp]=scc;
            minn[scc]=min(minn[scc],w[temp]);
            is_instack[temp]=0;
            sta.pop();
            if(temp==u) break;
        }
    }
}

//main函数Tarjan用法
main:
init();
for(int i=1;i<=n;i++){
    if(!dfn[i]) Tarjan(i);
}
```

### 2.4.2 求割点

割点去掉后各联通快大小

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

using namespace std;
const int N = 1e5+100;
typedef long long ll;
int dfn[N],low[N];
ll n,m,siz[N];
int idx;
vector<ll> G[N],GD[N];
bool jud[N];//是否为割点
void Tarjan(int u,int fa){
    dfn[u]=low[u]=++idx;
    siz[u]=1;
    int allsiz=0;
    for(int v:G[u]){
        if(!dfn[v]){//没有访问过
            Tarjan(v,u);
            siz[u]+=siz[v];
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[u]){//u为割点
                jud[u]=1;
                allsiz+=siz[v];
                GD[u].push_back(siz[v]);
            }
        }
        else if(v!=fa){
            low[u]=min(low[u],low[v]);
        }
    }
    if(jud[u]&&n-allsiz-1){
        GD[u].push_back(n-allsiz-1);
    }
}
int main(){
    idx=0;
    cin>>n>>m;
    int u,v;
    for(int i=1;i<=m;i++){
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    Tarjan(1,0);
    for(int i=1;i<=n;i++){
```

```
        if(!jud[i]){
            cout<<2*(n-1)<<endl;
        }else{
            ll ans=n*(n-1);
            ll now=0;
            for(ll v:GD[i]){
                now+=v*(v-1);
            }
            cout<<ans-now<<endl;
        }
    }
    return 0;
}
```

### 2.4.3 无向图点双连通

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

using namespace std;
const int N = 2e5+100;
int n,m;
int bcc_cnt;//bcc个数
int dfs_clock;
int pre[N];
bool is_cut[N];//判断是否是割点
int bccno[N];//第i个点是属于哪一个双连通分量
vector<int> G[N],bcc[N];

struct edge{
    int u,v;
    edge(int u,int v):u(u),v(v){}
};
stack<edge> s;
int Tarjan(int u,int fa){
    int lowu=pre[u]=++dfs_clock;
    int child=0;
    for(int v:G[u]){
        edge e=edge(u,v);
        if(!pre[v]){
            s.push(e);
            child++;
            int lowv=Tarjan(v,u);
            lowu=min(lowv,lowu);
            if(lowv>=pre[u]){//找到了割点
                is_cut[u]=1;
                bcc_cnt++;
                bcc[bcc_cnt].clear();
                while(1){
                    edge x=s.top();s.pop();
                    if(bccno[x.u]!=bcc_cnt){
                        bcc[bcc_cnt].push_back(x.u);
                        bccno[x.u]=bcc_cnt;
                    }
                    if(bccno[x.v]!=bcc_cnt){
                        bcc[bcc_cnt].push_back(x.v);
                        bccno[x.v]=bcc_cnt;
                    }
                    if(x.u==u&&x.v==v) break;
                }
            }
        }else if(pre[v]<pre[u]&&v!=fa){
            s.push(e);
            lowu=min(lowu,pre[v]);
        }
    }
    if(fa<0&&child==1) is_cut[u]=0;
    return lowu;
}
void find_bcc(int n){
    for(int i=0;i<=n;i++) pre[i]=is_cut[i]=bccno[i]=0;
    for(int i=1;i<=n;i++){
        if(!pre[i]) Tarjan(i,-1);
    }
}
int main(){
    int cnt=0;
    cin>>n>>m;
    int u,v;
    for(int i=0;i<m;i++){
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    find_bcc(n);
    cout<<bcc_cnt<<endl;
    for(int i=1;i<=bcc_cnt;i++){
        cout<<"BCC "<<i<<endl;
        for(int v:bcc[i]) cout<<v<<' ';
        cout<<endl;
    }
    return 0;
}
```

### 2.4.4 无向图边双连通

该程序是判断两点之间是否有两条不相交的路径

```
#include <bits/stdc++.h>
using namespace std;
const int N=5e4+5,M=1e5+5;
int n,m,vis[M<<1],ans;
int cnt=1,head[N],u[M],v[M];
int now,col,dfn[N],low[N],color[N],q,x,y;
stack<int> sta;
struct edge{int next,to;}e[M<<1];
inline void add(int u,int v){
    cnt++;
    e[cnt].next=head[u];
    e[cnt].to=v;
    head[u]=cnt;
    cnt++;
    e[cnt].next=head[v];
    e[cnt].to=u;
    head[v]=cnt;
}
inline void tarjan(int u)
{
    dfn[u]=low[u]=++now;
    sta.push(u);
    for (int i=head[u];i; i=e[i].next){
        if(!vis[i]){
            vis[i]=vis[i^1]=1;
            if (!dfn[e[i].to]){
                tarjan(e[i].to);
                low[u]=min(low[u],low[e[i].to]);
            }
            else low[u]=min(low[u],dfn[e[i].to]);
        }
    }
```

```
    if (low[u]==dfn[u]){
        color[u]=++col;
        while (1){
            int now=sta.top();sta.pop();
            color[now]=col;
            if(now==u) break;
        }
    }
}

int main(){
    memset(head,0,sizeof(head));
    memset(dfn,0,sizeof(head));
    scanf("%d%d",&n,&m);
    for (int i=1; i<=m; ++i){
        scanf("%d%d",&u[i],&v[i]);
        add(u[i],v[i]);
    }
    for (int i=1; i<=n; ++i){
        if (!dfn[i]) tarjan(i);
    }
    scanf("%d",&q);
    while(q--){
        scanf("%d %d",&x,&y);
        if(color[x]!=color[y]){
            puts("No");
        }else{
            puts("Yes");
        }
    }
    return 0;
}
```

## 2.5 求无向连通图的第 K 大联通子图

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

using namespace std;
bitset<105> bs[105];
long long w[105];
char maze[105][105];
struct node{
    bitset<105> cl;
    long long w;
    int last;
};
bool operator<(node a,node b){
    return a.w>b.w;
}
priority_queue<node> q;
int n,k;
long long bfs(){
    node now;
    now.cl.reset();
    now.w=0;
    now.last=0;

    q.push(now);
    while(!q.empty()){
        node rt=q.top();q.pop();
        if(--k==0) return rt.w;

        for(int i=rt.last+1;i<=n;i++){
            if(((bs[i]&rt.cl).count()==rt.cl.count())){
                node pt=rt;
                pt.cl[i]=1;
                pt.last=i;
                pt.w+=w[i];
                q.push(pt);
            }
        }
    }
    return -1;
}
int main(){
    cin>>n>>k;
    for(int i=1;i<=n;i++) cin>>w[i];
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            cin>>maze[i][j];
            if(maze[i][j]=='1'){
                bs[i][j]=1;
            }
        }
    }
    cout<<bfs()<<endl;
    return 0;
}
```

## 2.6 分层（K）图最短路

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int ,int> pii;
const int INF = 0x3f3f3f3f;
const int N = 5e6+10;
const int M = 5e6+10;

struct EDGE{
    int next;
    int to;
    int w;
}edge[M];

int n,m,k,s,e,cnt = 1;
int head[N],dis[N];
bool inq[N];

void add(int u, int v, int w){
    edge[cnt].next = head[u];
    edge[cnt].to = v;
    edge[cnt].w = w;
    head[u] = cnt++;
}

struct NODE{
    int id,dist;
}q,p;
bool operator < (NODE a, NODE b){
    return a.dist > b.dist;
}
void Dijkstra(){
    memset(dis, INF, sizeof dis);
    memset(inq, 0, sizeof inq);
    priority_queue<NODE> que;
    p.id = s; p.dist = 0;
    dis[s] = 0; que.push(p);
```

```
    while(!que.empty()){
        q = que.top(); que.pop();
        if(inq[q.id]) continue;
        inq[q.id] = true;
        for(int i=head[q.id]; ~i; i=edge[i].next){
            int u = edge[i].to;
            if(dis[u] > q.dist + edge[i].w){
                dis[u] = q.dist + edge[i].w;
                p.id = u;
                p.dist = dis[u];
                que.push(p);
            }
        }
    }
    int ans = INF;
    for(int i=0; i<=k; ++i) ans = min(ans, dis[e + i*n
        ]);
    printf("%d\n", ans);
}
int main(int argc, char const *argv[])
{
    int u, v, w;
    memset(head, -1, sizeof head);
    scanf("%d %d %d %d %d",&n, &m, &s, &e, &k);
    for(int i=1;i<=m;++i){
        scanf("%d %d %d",&u, &v, &w);
        for(int j=0; j<=k; ++j){
            add(u + j*n, v + j*n, w);
            add(v + j*n, u + j*n, w);
            if(j != k){
                add(u + j*n, v + (j+1)*n, 0);
                add(v + j*n, u + (j+1)*n, 0);
            }
        }
    }
    Dijkstra();
    return 0;
}
```

# 3 数学

## 3.1 整除分块

计算 $\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor$

```
for(int l=1,r;l<=n;l=r+1)
{
    r=n/(n/l);
    ans+=(r-l+1)*(n/l);
}
```

## 3.2 SG 函数打表

```
int op[110],sg[11000];
int k,N;
vector<int > s;
void getSG(){
    sg[0] = 0;
    for(int i=1;i<=N;++i){
        s.clear();
        for(int j=1;i>=op[j] && j<=k;++j)
            s.push_back(sg[i - op[j]]);
        for(int j=0;;++j){
            if(count(s.begin(),s.end(),j) == 0){
                sg[i] = j;
                break;
            }
        }
    }
}
```

## 3.3 线性素数 + 莫比乌斯函数打表

```
int miu[MAXN+10], check[MAXN+10], prime[MAXN+10];
void Mobius()
{
    memset(check,false,sizeof(check));
    miu[1] = 1;
    int tot = 0;
    for(int i = 2; i <= MAXN; i++)
    {
        if( !check[i] )
        {
            prime[tot++] = i;
            miu[i] = -1;
        }
        for(int j = 0; j < tot; j++)
        {
            if(i * prime[j] > MAXN) break;
            check[i * prime[j]] = true;
            if( i % prime[j] == 0)
            {
                miu[i * prime[j]] = 0;
                break;
            }
            else
            {
                miu[i * prime[j]] = -miu[i];
            }
        }
    }
}
```

## 3.4 Lucas 定理求 C（n，m）% P

```
typedef long long LL;

LL mod;

inline LL pow(LL a, LL b)//快速幂是为了求逆元
{
    LL ans = 1;
    for(; b; b >>= 1,a = a * a % mod)
        if(b & 1)
            ans = ans * a % mod;
    return ans;
}

LL farc[1000005];

inline void prepare(LL a)
{
    farc[0]=1;
    for(LL i = 1; i <= a; ++i)
        farc[i]=farc[i-1]*i%mod;
```

```
}

inline LL Csmall(LL m, LL n) // C(m,n) = (n!)/(m!*(n-m)!)
{
    if(n < m)
        return 0;
    return farc[n] * pow(farc[m], mod-2) % mod * pow(farc[n-m], mod-2) % mod; // 费马小定理求逆元
}

inline LL C(LL m, LL n)
{
    if(n < m)
        return 0;
    if(!n)
        return 1;//Lucas的边界条件
    return C(m/mod, n/mod) % mod * Csmall(m%mod, n%mod) % mod; // 上面证明的Lucas定理
}
```

### 3.5 大数质因子分解 & 大素数检测

```
ll Abs( ll a ){ return a<0?-a:a; }
ll Min( ll a , ll b ){ return a<b?a:b; }
ll Max( ll a , ll b ){ return a>b?a:b; }
ll Gcd( ll a , ll b ){ return b==0?a:Gcd( b , a%b ); }
ll arr[5] = { 2,3,5,233,331 };
ll Qmul( ll a , ll b , ll mod )
{
    ll res = 0;
    while ( b )
    {
        if ( b&1 )
            res = ( res+a )%mod;
        a = ( a+a )%mod;
        b = b>>1;
    }
    return res;
}
ll Qpow( ll a , ll b , ll mod )
{
    ll res = 1;
    while ( b )
    {
        if ( b&1 )
            res = Qmul( res , a , mod );
        a = Qmul( a , a , mod );
        b = b>>1;
    }
    return res;
}
bool Miller_Rabin( ll n )
{
    if ( n==2 ) return true;
    if ( n <2||n%2==0 ) return false;
    ll m = n-1,k = 0;
    while ( m%2==0 ) k++,m>>=1;
    for ( int I=0 ; I<5 ; I++ )
    {
        ll a = arr[I]%(n-1)+1;
        ll x = Qpow( a , m , n );
        for ( int j=1 ; j<=k ; j++ )
        {
            ll y = Qmul( x , x , n );
            if ( y==1&&x!=1&&x!=n-1 )
                return false;
            x = y;
        }
        if ( x!=1 ) return false;
    }
    return true;
}
ll fac[110], tol = 0;
ll Pollard_rho( ll x , ll c )
{
    ll I=1,k=2;
    ll x0 = rand()%x;
    ll y0 = x0;
    while ( 1 )
    {
        I++;
        x0 = ( Qmul( x0 , x0 , x )+c )%x;
        ll d0 = Gcd( Abs( y0-x0 ) , x );
        if ( d0!=1&&d0!=x ) return d0;
        if ( y0==x0 ) return x;
        if ( I == k ) { y0=x0; k+=k; }
    }
}
void Findfac( ll n )
{
    if ( Miller_Rabin( n ) )
    {
        fac[tol++] = n;
        return;
    }
    ll p = n;
    while ( p>=n )
        p = Pollard_rho( p , rand()%(n-1)+1 );
    Findfac( p );
    Findfac( n/p );
}
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(b==0)
    {
        x=1,y=0;
        return a;
    }
    ll g=exgcd(b,a%b,x,y);
    ll tmp=x;x=y;y=tmp-a/b*y;
    return g;
}
```

### 3.6 python 通用中国剩余定理

```
"""
n 方程个数
a1 r1: x = a1 (mod r1)
flag 是否有解
"""
def egcd(a, b):
    if 0 == b:
        return 1, 0, a
    x, y, q = egcd(b, a % b)
    x, y = y, (x - a // b * y)
    return x, y, q
```

```
n = int(input().split)
flag = False
a1, r1 = map(int, input().split())
for _ in range(n-1):
    a2, r2 = map(int, input().split())
    R = r2-r1
    x, y, d = egcd(a1, a2)
    tmp = a2//d
    if R%d != 0:
        flag = True
    r1=((x*R//d)%tmp+tmp)%tmp*a1+r1
    a1=a1*(a2//d)
lcm = a1
ans = (r1%lcm+lcm)%lcm
```

## 3.7 在线求组合数

```
void init(){
    fact[0]=inv[1]=factinv[0]=inv[0]=fact[1]=factinv[1]=1;
    for(int i=2;i<=MAXN;i++){
        fact[i]=(fact[i-1]%mod*i%mod)%mod;
        inv[i]=(mod-mod/i)*inv[mod%i]%mod;
        factinv[i]=factinv[i-1]*inv[i]%mod;
    }
}
ll c(ll n,ll m){
    return fact[n]*factinv[m]%mod*factinv[n-m]%mod;
}
```

## 3.8 拉格朗日插值

### 3.8.1 连续情况

以计算 $\sum_{i=1}^{n} i^k$ 为例

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

using namespace std;
const int N = 1e6+100;
typedef long long ll;
const ll mod = 1e9+7;
ll p[N],x[N],s1[N],s2[N],ifac[N];
ll qpow(ll a,ll b){
    ll ans=1;
    while(b){
        if(b&1) ans=(ans%mod*a%mod)%mod;
        a=(a%mod*a%mod)%mod;
        b>>=1;
    }
    return (ans%mod+mod)%mod;
}

//拉格朗日插值，n项，每个点的坐标为(x_i,y_i)，求第xi项的值，保证x是连续的一段
ll lagrange(ll n, ll *x, ll *y, ll xi) {
    ll ans = 0;
    s1[0] = (xi-x[0])%mod, s2[n+1] = 1;
    for (ll i = 1; i <= n; i++) s1[i] = 1ll*s1[i-1]*(xi-x[i])%mod;
    for (ll i = n; i >= 0; i--) s2[i] = 1ll*s2[i+1]*(xi-x[i])%mod;
    ifac[0] = ifac[1] = 1;
    for (ll i = 2; i <= n; i++) ifac[i] = -1ll*mod/i*ifac[mod%i]%mod;
    for (ll i = 2; i <= n; i++) ifac[i] = 1ll*ifac[i]*ifac[i-1]%mod;
    for (ll i = 0; i <= n; i++)
        (ans += 1ll*y[i]*(i == 0 ? 1 : s1[i-1])%mod*s2[i+1]%mod
        *ifac[i]%mod*(((n-i)&1) ? -1 : 1)*ifac[n-i]%mod) %= mod;
    return (ans+mod)%mod;
}
int main(){
    ll n,k;
    cin>>n>>k;
    if(k==0){
        cout<<n<<endl;
        return 0;
    }
    p[0]=0;
    for(ll i=1;i<=k+2;i++) p[i]=(p[i-1]%mod+qpow(i,k))%mod;
    for(ll i=1;i<=k+2;i++) x[i]=i;
    if(n<=k+2){
        cout<<p[n]<<endl;
    }
    else{
        cout<<lagrange(k+2,x,p,n)<<endl;
    }
    return 0;
}
```

### 3.8.2 非连续情况

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

using namespace std;
const int N = 1e6+100;
typedef long long ll;
const ll mod = 998244353;
struct point{
    ll x,y;
}p[N];
int n,k;
ll qpow(ll a,ll b,ll mod){
    ll ans=1;
    while(b){
        if(b&1){
            ans=(ans%mod*a%mod)%mod;
        }
        a=(a%mod*a%mod)%mod;
        b>>=1;
    }
    return ans%mod;
}
ll Lagrange(int k){
    ll ans=0;
    for(int j=1;j<=n;j++){//
        ll base1=1;
        ll base2=1;
        for(int i=1;i<=n;i++){//lj(k)基函数
            if(j==i) continue;
            base1=(base1%mod*((k-p[i].x)%mod+mod)%mod)%mod;
```

```
            base2=(base2%mod*((p[j].x-p[i].x)%mod+mod)%mod)%mod;
        }
        ans=(ans%mod+(p[j].y%mod*base1%mod*qpow(base2,mod-2,mod)%mod)%mod)%mod;
    }
    return ans;
}
int main(){
    cin>>n>>k;
    for(int i=1;i<=n;i++) cin>>p[i].x>>p[i].y;
    cout<<Lagrange(k)<<endl;
    return 0;
}
```

## 3.9 辛普森自适应积分

```
#include<cstdio>
#include<cmath>
double a, b, c, d, L, R;
double F(double x) {
    return (c * x + d) / (a * x + b);
}
double sim(double l, double r) {
    return (F(l) + F(r) + 4 * F((l + r) / 2)) * (r - l) / 6;
}
double asr(double L, double R, double eps, double ans) {
    double mid = (L + R) / 2;
    double LL = sim(L, mid), RR = sim(mid, R);
    if(fabs(LL + RR - ans) < eps) return LL + RR;
    else return asr(L, mid, eps / 2, sim(L, mid)) + asr(mid, R, eps / 2, sim(mid, R));
}
main() {
    #ifdef WIN32
    freopen("a.in", "r", stdin);
    #endif
    scanf("%lf %lf %lf %lf %lf %lf", &a, &b, &c, &d, &L, &R);
    printf("%lf", asr(L, R, 1e-6, sim(L, R)));
```

## 3.10 欧拉函数

比 n 小的与 n 互质的数的个数

### 3.10.1 在线

```

int euler(int n)//返回euler(n)
{
    int i;
    int res = n,a = n;
    for(i = 2;i*i <= a; ++i)
    {
        if(a%i == 0)
        {
            res -= res/i; //p(n) = (p - p/p1)(1 - 1/p2).....
            while(a%i == 0) a/=i;
        }
    }
    if(a > 1) res -= res/a;//存在大于sqrt(a)的质因子
    return res;
}
```

### 3.10.2 打表

```
void SE()//select euler//类似于素数筛选法
{
    int i,j;
    euler[1] = 1;
    for(i = 2;i < Max; ++i) euler[i]=i;
    for(i = 2;i < Max; ++i)
    {
        if(euler[i] == i)//这里出现的肯定是素数
        {
          for(j = i; j < Max; j += i)//然后更新含有它的数
          {
            euler[j] = euler[j]/i*(i - 1); // n*(1 - 1/p1)....*(1 - 1/pk).先除后乘
          }
        }
    }
    //for (int i = 1; i <= 20; ++i) printf("%d ", euler[i]);
}
```

## 3.11 欧拉降幂

降幂公式:

$$a^b\%p=\begin{cases}a^{b\%\phi(p)}\%p & gcd(a,p)=1\\ a^b\%p & gcd(a,p)\neq 1, b<\phi(p)\\ a^{b\%\phi(p)+\phi(p)}\%p & gcd(a,p)\neq 1,\phi(p)\leq b\end{cases}$$

```
#include<bits/stdc++.h>
using namespace std;
const int N = 2e6+100;
const int mod = 1e9+7;
typedef long long ll;
const int INF = 0x3f3f3f3f;
const ll llINF = 0x3f3f3f3f3f3f3f3f;
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define fep(i,a,b) for(int i=(a);i>=(b);i--)
inline bool read(ll &num) {
    char in;bool IsN=false;
    in=getchar();
    if(in==EOF) return false;
    while(in!='-'&&(in<'0'||in>'9')) in=getchar();
    if(in=='-'){ IsN=true;num=0;}
    else num=in-'0';
    while(in=getchar(),in>='0'&&in<='9'){
          num*=10,num+=in-'0';
    }
    if(IsN) num=-num;
    return true;
}
ll ph[N];
void init(){
    rep(i,1,N-10){
       ph[i]=i;
    }
    rep(i,2,N-10){
```

```
        if(ph[i]==i){
            for(int j=i;j<=N-10;j+=i){
                ph[j]=ph[j]/i*(i-1);
            }
        }
    }
}
ll qpow(ll a,ll b,ll mod){
    ll ans=1;
    while(b){
        if(b&1) ans=(ans%mod*a%mod)%mod;
        a=(a%mod*a%mod)%mod;
        b>>=1;
    }
    return ans%mod;
}
bool check(ll a,ll b,ll m){
    if(b==0) return 1>=ph[m];
    if(b==1) return a>=ph[m];
    ll ans=1;
    if(ans>=ph[m]) return 1;
    rep(i,1,b-1){
        rep(j,1,a){
            ans*=a;
            if(ans>=ph[m]) return 1;
        }
    }
    return 0;
}
ll solve(ll a,ll b,ll m){
    if(m==1) return 0;
    if(b==0) return 1%m;
    if(b==1) return a%m;
    if(__gcd(a,m)==1){
        return qpow(a,solve(a,b-1,ph[m]),m);
    }
    else{
        if(check(a,b-1,m)){
            return qpow(a,solve(a,b-1,ph[m])+ph[m],m);
        }
        else return qpow(a,solve(a,b-1,m),m);
    }
}
ll T,a,b,m;
int main(){
    //freopen("1.in", "r", stdin);
    read(T);
    init();
    //cout<<ph[1000000]<<endl;
    while(T--){
        read(a);read(b);read(m);
        printf("%lld\n",solve(a,b,m)%m);
    }
    return 0;
}
```

# 4 数据结构

## 4.1 线段树

### 4.1.1 区间修改区间查询

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll maxn = 1000050;
ll ans[maxn<<1],a[maxn],mod;
ll add[maxn],mult[maxn];
inline void pushup(ll rt){
    ans[rt]=(ans[rt<<1]+ans[rt<<1|1])%mod;
}
void pushdown(ll rt,ll l,ll r){
    ll mid=(l+r)>>1;
    ans[rt<<1]=(ans[rt<<1]*mult[rt]+add[rt]*(mid-l+1))
        %mod;
    ans[rt<<1|1]=(ans[rt<<1|1]*mult[rt]+add[rt]*(r-mid
        ))%mod;

    mult[rt<<1]=(mult[rt]*mult[rt<<1])%mod;
    mult[rt<<1|1]=(mult[rt]*mult[rt<<1|1])%mod;

    add[rt<<1]=(add[rt<<1]*mult[rt]+add[rt])%mod;
    add[rt<<1|1]=(add[rt<<1|1]*mult[rt]+add[rt])%mod;

    add[rt]=0;
    mult[rt]=1;
    return ;
}
inline void buildtree(ll rt,ll l,ll r){
    mult[rt]=1;
    add[rt]=0;
    if(l==r){
        ans[rt]=a[l];
        return ;
    }
    ll mid=(l+r)>>1;
    buildtree(rt<<1,l,mid);
    buildtree(rt<<1|1,mid+1,r);
    pushup(rt);
}
inline void update1(ll nl,ll nr,ll l,ll r,ll rt,ll k)
    {
    if(nl<=l&&r<=nr){
        ans[rt]=(ans[rt]*k)%mod;
        add[rt]=(add[rt]*k)%mod;
        mult[rt]=(mult[rt]*k)%mod;
        return ;
    }
    pushdown(rt,l,r);
    ll mid=(l+r)>>1;
    if(nl<=mid){
        update1(nl,nr,l,mid,rt<<1,k);
    }
    if(nr>mid) update1(nl,nr,mid+1,r,rt<<1|1,k);
    pushup(rt);
}
inline void update2(ll nl,ll nr,ll l,ll r,ll rt,ll k)
    {
    if(nl<=l&&nr>=r){
        add[rt]=(add[rt]+k)%mod;
        ans[rt]=(ans[rt]+k*(r-l+1))%mod;
        return ;
    }
    pushdown(rt,l,r);
    ll mid=(l+r)>>1;
    if(nl<=mid){
        update2(nl,nr,l,mid,rt<<1,k);
    }
    if(nr>mid) update2(nl,nr,mid+1,r,rt<<1|1,k);
```

```
    pushup(rt);
}
ll query(ll nl,ll nr,ll l,ll r,ll rt){
    ll res=0;
    if(nl<=l&&r<=nr){
        return ans[rt]%mod;
    }
    ll mid=(l+r)>>1;
    pushdown(rt,l,r);
    if(nl<=mid) res=(res%mod+query(nl,nr,l,mid,rt<<1))
        %mod;
    if(nr>mid) res=(res%mod+query(nl,nr,mid+1,r,rt
        <<1|1))%mod;
    return res;
}
int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    ll n,m,op,x,y,k;
    cin>>n>>m>>mod;
    for(ll i=1;i<=n;i++) cin>>a[i];
    buildtree(1,1,n);
    while(m--){
        cin>>op;
        if(op==1){
            cin>>x>>y>>k;
            update1(x,y,1,n,1,k);
        }
        else if(op==2){
            cin>>x>>y>>k;
            update2(x,y,1,n,1,k);
        }
        else{
            cin>>x>>y;
            cout<<query(x,y,1,n,1)<<endl;
        }
    }
    return 0;
}
```

## 4.2 主席树

### 4.2.1 区间第 k 小

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

using namespace std;
const int N = 1e5+100;
struct node{
    int l,r,num;
}T[N*30];
vector<int> v;
int n,m,a[N],t,cnt,roots[N];
int getid(int x){
    return lower_bound(v.begin(),v.end(),x)-v.begin()
        +1;
}
void update(int l,int r,int &x,int y,int pos){
    T[++cnt]=T[y];T[cnt].num++;x=cnt;
    if(l==r) return ;
    int mid=(l+r)>>1;
    if(pos<=mid) update(l,mid,T[x].l,T[y].l,pos);
    else update(mid+1,r,T[x].r,T[y].r,pos);
}
int query(int l,int r,int x,int y,int k){
    if(l==r) return l;
    int sum=T[T[x].l].num-T[T[y].l].num;
    int mid=(l+r)>>1;
    if(sum>=k) return query(l,mid,T[x].l,T[y].l,k);
    else return query(mid+1,r,T[x].r,T[y].r,k-sum);
}
int main(){
    scanf("%d",&t);
    while(t--){
        v.clear();
        cnt=0;
        scanf("%d %d",&n,&m);
        for(int i=1;i<=n;i++){
            scanf("%d",&a[i]);
            v.push_back(a[i]);
        }
        sort(v.begin(),v.end());
        v.erase(unique(v.begin(),v.end()),v.end());
        for(int i=1;i<=n;i++) update(1,n,roots[i],
            roots[i-1],getid(a[i]));
        while(m--){
            int l,r,k;
            scanf("%d %d %d",&l,&r,&k);
            printf("%d\n",v[query(1,n,roots[r],roots[l
                -1],k)-1]);
        }
    }
    return 0;
}
```

### 4.2.2 区间内小于等于 x 的最大值

```
struct node
{
    ll sum,l,r;
}t[maxn*32];
int cnt;
void update(ll l,ll r,ll &x,ll y,ll pos){
    t[++cnt]=t[y];t[cnt].sum++;x=cnt;//复制节点并且更新
    if(l==r) return ;
    int mid=(l+r)>>1;
    if(mid>=pos) update(l,mid,t[x].l,t[y].l,pos);
    else update(mid+1,r,t[x].r,t[y].r,pos);
}
int query(int a,int b,int x,int l,int r)
{
    if(l==r)
    {
        if(l==x)
            return 0;
        else
            return l;
    }
    int mid=(l+r)>>1;
    int xx=t[t[b].l].sum-t[t[a].l].sum;
    int yy=t[t[b].r].sum-t[t[a].r].sum;
    int res=0;
    if(yy&&x>mid)
        res=query(t[a].r,t[b].r,x,mid+1,r);
    if(xx&&!res)
        res=query(t[a].l,t[b].l,x,l,mid);
    return res;
```

```
}

for(int i=1;i<=n;i++) update(1,n,roots[i],roots[i-1],
    a[i]);
query(roots[L-1],roots[R],x,1,n);
```

### 4.2.3 区间内距离 p 第 k 近的距离

```
struct node
{
   ll sum,l,r;
}t[maxn*32];
int cnt;
void update(ll l,ll r,ll &x,ll y,ll pos){
   t[++cnt]=t[y];t[cnt].sum++;x=cnt;//复制节点并且更新
   if(l==r) return ;
   int mid=(l+r)>>1;
   if(mid>=pos) update(l,mid,t[x].l,t[y].l,pos);
   else update(mid+1,r,t[x].r,t[y].r,pos);
}
int query(int a,int b,int x,int l,int r)
{
   if(l==r)
   {
      if(l==x)
         return 0;
      else
         return l;
   }
   int mid=(l+r)>>1;
   int xx=t[t[b].l].sum-t[t[a].l].sum;
   int yy=t[t[b].r].sum-t[t[a].r].sum;
   int res=0;
   if(yy&&x>mid)
      res=query(t[a].r,t[b].r,x,mid+1,r);
   if(xx&&!res)
      res=query(t[a].l,t[b].l,x,l,mid);
   return res;
}

for(int i=1;i<=n;i++) update(1,n,roots[i],roots[i-1],
    a[i]);
query(roots[L-1],roots[R],x,1,n);
```

### 4.2.4 区间内小于等于 x 的最大值

```
struct node
{
   ll sum,l,r;
}t[maxn*32];
int cnt;
void update(ll l,ll r,ll &x,ll y,ll pos){
   t[++cnt]=t[y];t[cnt].sum++;x=cnt;//复制节点并且更新
   if(l==r) return ;
   int mid=(l+r)>>1;
   if(mid>=pos) update(l,mid,t[x].l,t[y].l,pos);
   else update(mid+1,r,t[x].r,t[y].r,pos);
}
int query(int a,int b,int x,int l,int r)
{
   if(l==r)
   {
      if(l==x)
         return 0;
      else
         return l;
   }
   int mid=(l+r)>>1;
   int xx=t[t[b].l].sum-t[t[a].l].sum;
   int yy=t[t[b].r].sum-t[t[a].r].sum;
   int res=0;
   if(yy&&x>mid)
      res=query(t[a].r,t[b].r,x,mid+1,r);
   if(xx&&!res)
      res=query(t[a].l,t[b].l,x,l,mid);
   return res;
}

for(int i=1;i<=n;i++) update(1,n,roots[i],roots[i-1],
    a[i]);
query(roots[L-1],roots[R],x,1,n);
```

### 4.2.5 区间数的种类数

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1e6+5;
int n,q,a[N],p[N];

int rt[N*40],ls[N*40],rs[N*40],sum[N*40],cnt=0;
void up(int pre,int& o,int l,int r,int pos,int val) {
   o=++cnt;
   ls[o]=ls[pre];
   rs[o]=rs[pre];
   sum[o]=sum[pre]+val;
   if(l==r) return ;
   int m=(l+r)/2;
   if(pos<=m) up(ls[pre],ls[o],l,m,pos,val);
   else up(rs[pre],rs[o],m+1,r,pos,val);
}

int qu(int o,int l,int r,int ql,int qr) {
   if( ql<=l && qr>=r ) return sum[o];
   int ans = 0,m = (l+r)/2;
   if(ql<=m) ans += qu(ls[o],l,m,ql,qr);
   if(qr>m) ans += qu(rs[o],m+1,r,ql,qr);
   return ans;
}

int main(){
   scanf("%d",&n);
   for(int i=1;i<=n;i++) {
      scanf("%d",&a[i]);
      if(!p[a[i]]) {
         up(rt[i-1],rt[i],1,n,i,1);
      }else {
         int tp;
         up(rt[i-1],tp,1,n,p[a[i]],-1);
         up(tp,rt[i],1,n,i,1);
      }
      p[a[i]] = i;
   }
   scanf("%d",&q);
   while(q--) {
      int l,r;
      scanf("%d%d",&l,&r);
      int ans = qu(rt[r],1,n,l,r);
```

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

### 4.2.6 区间内未出现过的最小自然数

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int N = 2e5+5;
const int M = 1e9;

int rt[N*30],ls[N*30],rs[N*30],mn[N*30],cnt=0;
void up(int pre,int& o,int l,int r,int val,int pos) {
    o=++cnt;
    ls[o]=ls[pre];
    rs[o]=rs[pre];
    mn[o]=pos;
    if(l==r) return ;
    int m=(l+r)/2;
    if(val<=m) up(ls[pre],ls[o],l,m,val,pos);
    else up(rs[pre],rs[o],m+1,r,val,pos);
    mn[o]=min(mn[ls[o]],mn[rs[o]]);
}

int qu(int o,int l,int r,int pos) {
    if(l==r) return l;
    int m=(l+r)/2;
    if( mn[ls[o]]<pos ) return qu(ls[o],l,m,pos);
    return qu(rs[o],m+1,r,pos);
}

int a[N],n,q,l,r;
int main(){
    scanf("%d%d",&n,&q);
    for(int i=1;i<=n;i++) {
        scanf("%d",&a[i]);
        up(rt[i-1],rt[i],0,M,a[i],i);
    }
    while(q--) {
        scanf("%d%d",&l,&r);
        int ans = qu(rt[r],0,M,l);
        printf("%d\n",ans);
    }
    return 0;
}
```

## 4.3 树链剖分

```
#pragma GCC optimize(2)
#include<bits/stdc++.h>
#define rep(i, a, b) for(int i = (a); i <= (int)(b); ++i)
#define per(i, a, b) for(int i = (a); i >= (int)(b); --i)
#define debug(x) cerr << #x << ' ' << x << endl;
#define ls x<<1
#define rs x<<1|1
using namespace std;

typedef long long ll;
const int MAXN = 1e6 + 7;

int son[MAXN], fa[MAXN], dep[MAXN], siz[MAXN], top[MAXN], tid[MAXN], rnk[MAXN], w[MAXN];
vector<int> G[MAXN];
int n, m, s, cur = 0;

struct SegTree{
    struct Node{
        int l, r;
        ll lz, sum;
        int mid(){return (l+r)>>1;}
        int size(){return (r-l+1);}
    }s[MAXN<<2];
    inline void pushdown(int x){
        s[ls].lz += s[x].lz;
        s[ls].sum += s[x].lz * s[ls].size();
        s[rs].lz += s[x].lz;
        s[rs].sum += s[x].lz * s[rs].size();
        s[x].lz = 0;
    }
    inline void pushup(int x){
        s[x].sum = s[ls].sum + s[rs].sum;
    }
    inline void build(int x, int l, int r){
        s[x].l = l; s[x].r = r;
        if(l == r){
            s[x].lz = 0;
            s[x].sum = w[rnk[l]];
            return;
        }
        int mid = s[x].mid();
        build(ls, l, mid);
        build(rs, mid + 1, r);
        pushup(x);
    }
    inline ll query(int x, int l, int r){
        if(s[x].l == l && s[x].r == r) return s[x].sum;
        pushdown(x);
        int mid = s[x].mid();
        if(r <= mid) return query(ls, l, r);
        else if(l > mid) return query(rs, l, r);
        else return query(ls, l, mid) + query(rs, mid + 1, r);
    }
    inline void updata(int x, int l, int r, int v){
        if(s[x].l == l && s[x].r == r){
            s[x].lz += v;
            s[x].sum += 1LL * v * s[x].size();
            return;
        }
        pushdown(x);
        int mid = s[x].mid();
        if(r <= mid) updata(ls, l, r, v);
        else if(l > mid) updata(rs, l, r, v);
        else {
            updata(ls, l, mid, v);
            updata(rs, mid + 1, r, v);
        }
        pushup(x);
    }
}st;
void dfs1(int x, int f = 0){
    son[x] = -1;
```

```
    siz[x] = 1;
    dep[x] = dep[f] + 1;
    fa[x] = f;
    for(int u: G[x]){
        if(u == f) continue;
        dfs1(u, x);
        siz[x] += siz[u];
        if(son[x] == -1 || siz[son[x]] < siz[u]) son[x] = u;
    }
}

void dfs2(int x, int t){
    top[x] = t;
    cur++;
    tid[x] = cur;
    rnk[cur] = x;
    if(son[x] == -1) return;
    dfs2(son[x], t);
    for(int u: G[x]){
        if(u != son[x] && u != fa[x]) dfs2(u, u);
    }
}


//链上更新
inline void linkadd(int u, int v, int w){
    int fu = top[u], fv = top[v];
    while(fu != fv){
        if(dep[fu] >= dep[fv]){
            st.updata(1, tid[fu], tid[u], w);
            u = fa[fu];
        } else {
            st.updata(1, tid[fv], tid[v], w);
            v = fa[fv];
        }
        fu = top[u];
        fv = top[v];
    }
    if(tid[u] > tid[v]) swap(u, v);
    st.updata(1, tid[u], tid[v], w);
}

//链上求和查询
inline ll linkquery(int u, int v){
    int fu = top[u], fv = top[v];
    ll res = 0;
    while(fu != fv){
        if(dep[fu] >= dep[fv]){
            res += st.query(1, tid[fu], tid[u]);
            u = fa[fu];
        } else {
            res += st.query(1, tid[fv], tid[v]);
            v = fa[fv];
        }
        fu = top[u];
        fv = top[v];
    }
    if(tid[u] > tid[v]) swap(u, v);
    res += st.query(1, tid[u], tid[v]);
    return res;
}

//子树查询
inline ll subtreequery(int x){
    return st.query(1, tid[x], tid[x] + siz[x] - 1);
}

//子树更新
inline void subtreeadd(int x, int w){
    return st.updata(1, tid[x], tid[x] + siz[x] - 1, w);
}

//查询LAC
inline int lca(int u, int v){
    int fu = top[u], fv = top[v];
    while(fu != fv){
        if(dep[fu] >= dep[fv]) u = fa[fu];
        else v = fa[fv];
        fu = top[u];
        fv = top[v];
    }
    if(dep[u] > dep[v]) swap(u, v);
    return u;
}
int main() {
    scanf("%d %d %d", &n, &m, &s);
    rep(i, 1, n) scanf("%d", &w[i]);
    int u, v, w, op;
    rep(i, 1, n-1){
        scanf("%d %d", &u, &v);
        G[u].push_back(v);
        G[v].push_back(u);
    }
    dfs1(s);
    dfs2(s, s);
    st.build(1, 1, n);
    while(m--){
        scanf("%d %d", &op, &u);
        if(op == 1) {
            scanf("%d %d", &v, &w);
            linkadd(u, v, w);
        } else if(op == 2) {
            printf("%lld\n", linkquery(u, u));
        } else {
            printf("%lld\n", subtreequery(u));
        }
    }
    return 0;
}
```

## 4.4 树状数组

### 4.4.1 二位偏序求矩形内点的个数

离线算法，以南京网赛 A 为例

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1e6+100;
const int mod = 1e9+7;
typedef long long ll;
const int INF = 0x3f3f3f3f;
const ll llINF = 0x3f3f3f3f3f3f3f3f;
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define fep(i,a,b) for(int i=(a);i>=(b);i--)
inline bool read(ll &num) {
    char in;bool IsN=false;
    in=getchar();
```

```
    if(in==EOF) return false;
    while(in!='-'&&(in<'0'||in>'9')) in=getchar();
    if(in=='-'){ IsN=true;num=0;}
    else num=in-'0';
    while(in=getchar(),in>='0'&&in<='9'){
           num*=10,num+=in-'0';
    }
    if(IsN) num=-num;
    return true;
}
ll T,n,p,m,c[N];
int lowbit(ll x){
    return (x&(-x));
}
void add(ll x,ll v){
    for(;x<N;x+=lowbit(x)){
        c[x]+=v;
        //cout<<x<<' '<<v<<endl;
    }
}
ll query(ll x){
    ll ans=0;
    for(;x;x-=lowbit(x)){
        ans+=c[x];
    }
    return ans;
}
//BIT
struct point{
    ll x,y;
    int flag;
}pp[600000];
bool cmp(point a,point b){
    if(a.x==b.x){
        if(a.y==b.y){
          return a.flag<b.flag;
        }
        return a.y<b.y;
    }
    return a.x<b.x;
}

ll dig(ll x){
    ll ans=0;
    while(x){
        ans+=x%10;
        x/=10;
    }
    return ans;
}
ll cal(ll x,ll y){//计算(x,y)处的值
    x=x-n/2-1;
    y=y-n/2-1;
    ll t=max(abs(x),abs(y));
    if(x>=y) return n*n-4*t*t-2*t-x-y;
    else return n*n-4*t*t+2*t+x+y;
}
map<pair<ll,ll>,ll> mmp;
ll yy[N],id[N];
ll x__1[100001],y__1[100001],x__2[100001],y__2
    [100001];
int main(){
    read(T);
    while(T--){
        mmp.clear();
        read(n);read(m);read(p);
        memset(c,0,sizeof(c));
        ll x,y,x_1,y_1,x_2,y_2;
        rep(i,1,m){
            read(x);read(y);
            pp[i]={x,y,0};
        }
        rep(i,1,p){
            read(x_1);read(y_1);read(x_2);read(y_2);
            x__1[i]=x_1;y__1[i]=y_1;x__2[i]=x_2;y__2[i
                ]=y_2;
            pp[++m]={x_1-1,y_1-1,1};
            pp[++m]={x_2,y_2,1};
            pp[++m]={x_1-1,y_2,1};
            pp[++m]={x_2,y_1-1,1};
        }
        rep(i,1,m) yy[i]=pp[i].y;
        sort(yy+1,yy+m+1);
        int siz=unique(yy+1,yy+m+1)-yy-1;
        sort(pp+1,pp+m+1,cmp);
        rep(i,1,m){
            id[i]=lower_bound(yy+1,yy+siz+1,pp[i].y)-yy
                ;
        }
        //离散化
        rep(i,1,m){
            if(pp[i].flag==0){
                add(id[i],dig(cal(pp[i].x,pp[i].y)));
                //cout<<pp[i].x<<' '<<pp[i].y<<' '<<id[i
                    ]<<' '<<dig(cal(pp[i].x,pp[i].y))<<
                    endl;
            }
            else{
                mmp[{pp[i].x,pp[i].y}]=query(id[i]);
                //cout<<pp[i].x<<' '<<pp[i].y<<' '<<
                    query(id[i])<<endl;
            }
        }
        rep(i,1,p){
            //cout<<x__2[i]<<' '<<y__2[i]<<endl;
            printf("%lld\n",mmp[{x__2[i],y__2[i]}]-mmp
                [{x__1[i]-1,y__2[i]}]-mmp[{x__2[i],y__1
                [i]-1}]+mmp[{x__1[i]-1,y__1[i]-1}]);
        }
    }
    return 0;
}
```

### 4.4.2 树状数组求区间最值

```
struct BIT{
    ll e[MAXN];
    int lowbit(int x){
        return x & -x;
    }
    void upd(int x){
        int lx;
        while(x <= n){
            e[x] = a[x];
            lx = lowbit(x);
            for(int I = 1; I < lx; I <<= 1) e[x] = max(
                e[x], e[x-I]);
            x += lowbit(x);
        }
```

```
    }
    ll query(int l, int r){
        ll ans = 0;
        while(r >= l){
            ans = max(a[r], ans);
            r--;
            while(r >= l + lowbit(r)){
                ans = max(e[r], ans);
                r -= lowbit(r);
            }
        }
        return ans;
    }
}bit;
```

#### 4.4.3 二维树状数组

查询二维前缀和

```
#include<iostream>
#include<string.h>
#include<algorithm>
#include<stdio.h>

using namespace std;
const int N = 1e3+100;
int c[N][N];
int n;
int lowbit(int x){
    return x&(-x);
}
void update(int x,int y,int k){
    for(int i=x;i<=n;i+=lowbit(i)){
        for(int j=y;j<=n;j+=lowbit(j)){
            c[i][j]+=k;
        }
    }
}
int query(int x,int y){
    int ans=0;
    for(int i=x;i;i-=lowbit(i)){
        for(int j=y;j;j-=lowbit(j)){
            ans+=c[i][j];
        }
    }
    return ans;
}
```

# 5 字符串

## 5.1 序列自动机

```
int nxt[N][27];//距离i往后最近的字母j的位置
void init(char *s){
    int l=strlen(s);
    for(int i=0;i<26;i++) nxt[l][i]=INF;
    for(int i=l-1;i>=0;i--){
        for(int j=0;j<26;j++){
            nxt[i][j]=nxt[i+1][j];
        }
        nxt[i][s[i]-'a']=i;
    }
}
```

## 5.2 KMP 计算 next 函数

### 5.2.1 vector 版

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

using namespace std;
const int N = 2e6+100;
int n,m;int a[N],b[N];
vector<int> res;
vector<int> cal(vector<int> a){
    int n=(int)a.size();
    vector<int> nxt(n);
    for(int i=1;i<n;i++){
        int j=nxt[i-1];
        while(j>0&&a[i]!=a[j]) j=nxt[j-1];
        if(a[i]==a[j]) j++;
        nxt[i]=j;
    }
    return nxt;
}
```

### 5.2.2 KMP 匹配过程

```
int kmp(){
    int i,j;
    i=j=0;
    while(i<n&&j<m){
        if(s[i]==t[j]){
            i++;j++;
        }
        else if(!j){
            i++;
        }
        else{
            j=nxt[j-1];
        }
    }
    if(j==m) return i-m+1;
    else return -1;
}
```

## 5.3 Z-function / Exkmp

```
vector<int> z_function(string s) {
  int n = (int)s.length();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r) z[i] = min(r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z
        [i];
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

## 5.4 Manacher

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

using namespace std;
```

```
const int N = 1e6+100;
int p[N];
string s;
int manacher(string s){
    string t="";
    t+='*';
    for(int i=0;i<(int)s.size();i++){
        t+=s[i];
        t+='*';
    }
    int ans=0;
    int pos=0;int maxxright=0;
    for(int i=0;i<(int)t.length();i++){
        p[i]=maxxright>i?min(p[2*pos-i],maxxright-i)
            :1;//关键
        while(i-p[i]>=0&&i+p[i]<(int)t.length()&&t[i-p
            [i]]==t[i+p[i]]) p[i]++;
        if(i+p[i]-1>maxxright){
            maxxright=i+p[i]-1;
            pos=i;
        }
        ans=max(ans,p[i]);
    }
    return ans-1;
}
int main(){
    cin>>s;
    cout<<manacher(s)<<endl;
    return 0;
}
```

## 5.5 后缀数组

从 1 到 n 输出 sa[i] sa[i] 代表排名为 i 的下标 ran[i] 代表下标为 i 的排名

```
string s;
int ran[N],tmp[N],sa[N];
int n,k;
bool cmp(int i,int j){
    if(ran[i]!=ran[j]) return ran[i]<ran[j];
    int ri=i+k<=n?ran[i+k]:-1;
    int rj=j+k<=n?ran[j+k]:-1;
    return ri<rj;
}
void construct_sa(string s,int *sa){
    n=(int)s.length();
    for(int i=0;i<=n;i++){
        sa[i]=i;
        ran[i]=i<n?s[i]:-1;
    }
    for(k=1;k<=n;k*=2){
        sort(sa,sa+n+1,cmp);
        tmp[sa[0]]=0;
        for(int i=1;i<=n;i++){
            tmp[sa[i]]=tmp[sa[i-1]]+(cmp(sa[i-1],sa[i])
                ?1:0);
        }
        for(int i=0;i<=n;i++) ran[i]=tmp[i];
    }
}
```

## 5.6 双哈希

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

using namespace std;
typedef long long ull;
typedef long long ll;
const int N = 1e6+100;
const int base1 = 233;
const int base2 = 2333;
const int mod1 = 1e9+9;
const int mod2 = 1e9+7;
ull hashes1[N],p1[N];
ull hashes2[N],p2[N];
string s;
ull gethash1(int l,int r){
    if(l==0) return hashes1[r]%mod1;
    return (hashes1[r]-(hashes1[l-1]%mod1*p1[r-l+1]%
        mod1)%mod1+mod1)%mod1;
}
ull gethash2(int l,int r){
    if(l==0) return hashes2[r]%mod2;
    return (hashes2[r]-(hashes2[l-1]%mod2*p2[r-l+1]%
        mod2)+mod2)%mod2;
}
int main(){
    int n;cin>>n>>s;
    hashes1[0]=s[0];p1[0]=1;
    hashes2[0]=s[0];p2[0]=1;
    for(int i=1;i<n;i++){
        hashes1[i]=(hashes1[i-1]*base1%mod1+(ull)s[i]%
            mod1)%mod1;
        hashes2[i]=(hashes2[i-1]*base2%mod2+(ull)s[i]%
            mod2)%mod2;
        p1[i]=(base1*p1[i-1])%mod1;
        p2[i]=(base2*p2[i-1])%mod2;
    }
}
```

## 5.7 字典树

### 5.7.1 指针版

```
struct Tire {
    const int MAXN = 4e5 + 7;
    int tr[MAXN][26], tot = 0;
    int cnt[MAXN];

    void insert(string s ) {
        int cur = 0, sz = s.size();
        cnt[cur] ++; //插入的字符串个数
        for(int i = 0; i < sz; ++i) {
            int to = s[i] - 'a';
            if(!tr[cur][to]) tr[cur][to] = ++tot;
            cur = tr[cur][to];
        }
        cnt[cur]++; //当前节点所表示的字符串的出现次数 +
            1
    }

    int get(string s) {
        int cur = 0, sz = s.size();
        for(int i = 0; i < sz; ++i) {
            int to = s[i] - 'a';
            if(!tr[cur][to]) return 0;
            cur = tr[cur][to];
```

```
        }
        return cnt[cur];
    }
}trie;
```

### 5.7.2 数组版 01 字典树

从数组 a 中找出一个数使得 k 和它 xor 最大

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

using namespace std;
const int N = (1e5+100)*33;
int tree[N][2];int a[N];
bool vis[N];
int tot;
void insert(int x){
    int now=0;
    for(int i=31;i>=0;i--){
        int id=(x>>i)&1;
        if(!tree[now][id]) tree[now][id]=++tot;
        now=tree[now][id];
    }
}
int find(int x){
    int ans=0;
    int now=0;
    for(int i=31;i>=0;i--){
        int id=!((x>>i)&1);
        ans*=2;
        if(tree[now][id]){
            ans++;now=tree[now][id];
        }
        else now=tree[now][!id];
    }
    return ans;
}
int main(){
    int T,n,m,s,all;
    all=1;
    scanf("%d",&T);
    while(T--){
        tot=0;
        memset(tree,0,sizeof(tree));
        // memset(vis,0,sizeof(vis));
        scanf("%d %d",&n,&m);
        for(int i=1;i<=n;i++) scanf("%d",&a[i]);
        for(int i=1;i<=n;i++){
            insert(a[i]);
        }
        printf("Case #%d:\n",all++);
        while(m--){
            scanf("%d",&s);
            printf("%d\n",find(s)^s);
        }
    }
    return 0;
}
```

## 5.8 AC 自动机

```
namespace AC{
    const int MAXN = 1e6 + 7;
    //注意字符集大小
    int tr[MAXN][30], tot = 0;
    int cnt[MAXN], fail[MAXN];
    void insert(string s){
        int cur = 0;
        for(int i = 0; i < s.size(); ++i){
            int to = s[i] - 'a';
            if(tr[cur][to] == 0) tr[cur][to] = ++tot;
            cur = tr[cur][to];
        }
        cnt[cur]++;
    }
    queue<int> que;
    void build(){
        for(int i = 0; i < 26; ++i){
            if(tr[0][i]) que.push(tr[0][i]);
        }
        while(!que.empty()){
            int cur = que.front();
            que.pop();
            for(int i = 0; i < 26; ++i){
                if(tr[cur][i]){
                    fail[tr[cur][i]] = tr[fail[cur]][i];
                    que.push(tr[cur][i]);
                } else tr[cur][i] = tr[fail[cur]][i];
            }
        }
    }
    //注意每次匹配结束 cnt 数组发生改变
    //如果需要再次匹配 需要重新建树
    int query(string s){
        int cur = 0, res = 0;
        for(int i = 0; i < s.size(); ++i){
            int to = s[i] - 'a';
            cur = tr[cur][to];
            for(int j = cur; j && cnt[j] != -1; j =
                fail[j]){
                res += cnt[j];
                cnt[j] = -1;
            }
        }
        return res;
    }
}
```

## 5.9 最小表示法

```
int k = 0, i = 0, j = 1;
while (k < n && i < n && j < n) {
  if (sec[(i + k) % n] == sec[(j + k) % n]) {
    k++;
  } else {
    //最大表示法只需要将此处的 > 更改为 < 即可
    sec[(i + k) % n] > sec[(j + k) % n] ? i = i + k +
        1 : j = j + k + 1;
    if (i == j) i++;
    k = 0;
  }
}
i = min(i, j);
```

# 6 C++ pbds

## 6.1 头文件

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>//用tree
#include<ext/pb_ds/hash_policy.hpp>//用hash
#include<ext/pb_ds/trie_policy.hpp>//用trie
#include<ext/pb_ds/priority_queue.hpp>//用
    priority_queue
using namespace __gnu_pbds;
```

## 6.2 Hash

```
cc_hash_table<int,bool> h; //拉链法
gp_hash_table<int,bool> h; //探测法 稍快
```

## 6.3 Tree

```
#define pii pair<int,int>
#define mp(x,y) make_pair(x,y)
tree<pii,null_type,less<pii>,rb_tree_tag,
    tree_order_statistics_node_update> tr;
pii //存储的类型
null_type //无映射(低版本g++为null_mapped_type)
less<pii> //从小到大排序
rb_tree_tag //红黑树
tree_order_statistics_node_update //更新方式
tr.insert(mp(x,y)); //插入;
tr.erase(mp(x,y)); //删除;
tr.order_of_key(pii(x,y)); //求排名 从0开始
tr.find_by_order(x); //找k小值，返回迭代器 从0开始
tr.join(b); //将b并入tr，前提是两棵树类型一样且没有重复
    元素
tr.split(v,b); //分裂，key小于等于v的元素属于tr，其余的
    属于b
tr.lower_bound(x); //返回第一个大于等于x的元素的迭代器
tr.upper_bound(x); //返回第一个大于x的元素的迭代器
//元素不能重复
//以上所有操作的时间复杂度均为O(logn)
```

## 6.4 Trie

```
typedef trie<string,null_type,
    trie_string_access_traits<>,pat_trie_tag,
    trie_prefix_search_node_update> tr;
//第一个参数必须为字符串类型，tag也有别的tag，但pat最快，
    与tree相同，node_update支持自定义
tr.insert(s); //插入s
tr.erase(s); //删除s
tr.join(b); //将b并入tr
pair//pair的使用如下:
pair<tr::iterator,tr::iterator> range=base.
    prefix_range(x);
for(tr::iterator it=range.first;it!=range.second;it
    ++)
   cout<<*it<<' '<<endl;
//pair中第一个是起始迭代器，第二个是终止迭代器，遍历过去就
    可以找到所有字符串了。
```

## 6.5 优先队列

```
priority_queue<int,greater<int>,TAG> Q;//小根堆，大根
    堆写less<int>
/*其中的TAG为类型，有以下几种:
pairing_heap_tag
thin_heap_tag
binomial_heap_tag
rc_binomial_heap_tag
binary_heap_tag
其中pairing_help_tag最快*/
Q.push(x);
Q.pop();
Q.top();
Q.join(b);
Q.empty();
Q.size();
Q.modify(it,6);
Q.erase(it);
//以上操作我都不讲了，pbds里的优先队列还可以用迭代器遍历
```