数据结构

珂朵莉树

```
ll rnk(){
    ll ret=seed;
    seed=(seed*7+13)%MOD;
    return ret;
}
struct Node{
  int l,r;
  mutable ll value;//
  Node(int a,int b,long long c):l(a),r(b),value(c){}
  Node(int a):l(a),r(0),value(0){}
  bool operator < (const Node& o) const{
    return l<o.l;
  }
};
set<Node>s;
set<Node>::iterator split(int pos){
    set<Node>::iterator it=s.lower_bound(Node(pos));
    if (it->l==pos && it!=s.end()) return it;//
    --it;
    if (pos > it->r) return s.end();//
    int L=it->l,R=it->r;
    ll V=it->value;
    s.erase(it);
    s.insert(Node(L,pos-1,V));
    return s.insert(Node(pos,R,V)).first;
}
void assign(int l,int r,int v){
    split(l);
    set<Node>::iterator R=split(r+1);
    set<Node>::iterator L=split(l);
    s.erase(L,R);
    s.insert(Node(l,r,v));
}
void add(int l,int r,int v){
    split(l);
    auto R=split(r+1),L=split(l);
    for (;L!=R;++L){
        L->value += v;
    }
}
ll kth(int l,int r, int k){
    split(l);
    vector< pair<ll,int> >q;
    q.clear();
    set<Node>::iterator R=split(r+1);
    set<Node>::iterator L=split(l);
    for (set<Node>::iterator it=L;it!=R;++it){
        q.push_back({ it->value , it->r - it->l + 1});
    }
    sort(q.begin(),q.end());//
    for (auto i:q)
    {
        k-=i.second;
        if (k<=0) return i.first;
    }
    return -1;
}
ll qpow(ll a,ll x,ll y){
    ll ans=1,res=a%y;
    while(x!=0){
        if ( x & 1 ) {
            ans*=res;
            ans=ans%y;
        }
        res=res*res%y;
        x>>=1;
    }
    return ans%y;
}
ll sum(int l,int r,int x,int y){
    split(l);
    ll ans=0;
    set<Node>::iterator R=split(r+1);
    set<Node>::iterator L=split(l);
    for (set<Node>::iterator it=L;it!=R;++it){
        ans+=( (ll)( it->r - it->l + 1 ) * qpow(it->value,x,y));
        ans=ans%y;
    }
    return ans;
}
```

树状数组

一组数据，要求能够区间修改和区间询问．同时要完成两种操作，只要维护两个树状数组就好了．一个差分数组，另一个是对差分数组进行求和的数组．

```
ll lowbit(ll x){    return x&(-x);}
void update(ll i,ll val){
    ll x=i;
    while(i<=N){
        ta[i]+=val;
        i+=lowbit(i);
    }
}
ll getsum(ll i){
    ll res=0,x=i;
    while(i>0){
        res+=ta[i];
        i-=lowbit(i);
    }
    return res;
}
```

ST表

```
int Min(int a,int b){
    return s[a]<=s[b]?a:b;
}
void getST(){
    len=strlen(s);
    memset(ST,0,sizeof(ST));
    for (int j=0;j<=11;j++)
        for (int i=0;i+bin[j]-1<len;i++){
            if (j==0) ST[i][j]=i;
            else ST[i][j]=Min(ST[i][j-1],ST[i+bin[j-1]][j-1]);
        }
}
int STsearch(int x,int y){
    if (x==y) return x;
    int t=Log[y-x+1];
    return Min(ST[x][t],ST[y-bin[t]+1][t]);
}
```

线段树

```
void build(int o,int l,int r){
    lazy1[o]=1;
    if (l==r) { sum[o]=num[l]%p;    return ;}
    build(lo,l,M);
    build(ro,M+1,r);
    sum[o]=(sum[lo]+sum[ro])%p;
}
void pushdown(int o,int l,int r){
    lazy1[lo]*=lazy1[o];    lazy1[lo]%=p;
    lazy1[ro]*=lazy1[o];    lazy1[ro]%=p;
    lazy2[lo]=lazy2[lo]*lazy1[o]+lazy2[o];  lazy2[lo]%=p;
    lazy2[ro]=lazy2[ro]*lazy1[o]+lazy2[o];  lazy2[ro]%=p;
    sum[lo]=((sum[lo]*lazy1[o])+lazy2[o]*len(l,M))%p;
    sum[ro]=((sum[ro]*lazy1[o])+lazy2[o]*len(M+1,r))%p;
    lazy1[o]=1;
    lazy2[o]=0;
}
ll getsum(int o,int l,int r){
    ll res=0;
    if (x<=l && r<=y) return sum[o]%p;
    pushdown(o,l,r);
    if ( M >= x) {
        res+=getsum(lo,l,M);
        res%=p;
    }
    if ( M < y) {
        res+=getsum(ro,M+1,r);
        res%=p;
    }
    return res%p;
}
void addval(int o,int l,int r){
    if (x<=l && r<=y) {
        sum[o]=(sum[o]+k*len(l,r))%p;
        lazy2[o]+=k;    lazy2[o]%=p;
        return ;
    }
    pushdown(o,l,r);
    if ( M >= x ) addval(lo,l,M);
    if ( M < y ) addval(ro,M+1,r);
    sum[o]=sum[lo]+sum[ro];
    sum[o]%=p;
}
void multival(int o,int l,int r){
    if (x<=l && r<=y) {
        sum[o]=(sum[o]*k)%p;
        lazy1[o]*=k;lazy1[o]%=p;
        lazy2[o]*=k;lazy2[o]%=p;
        return ;
    }
    pushdown(o,l,r);
    if ( M >= x) multival(lo,l,M);
    if ( M < y) multival(ro,M+1,r);
    sum[o]=sum[lo]+sum[ro];
    sum[o]%=p;
}
```

扫描线

```
void do_odd(ll o, ll L, ll R){
    ll all = num[R + 1] - num[L];
    tree[o].sum2 = (all - tree[o].sum2);
}
void pushup(ll o, ll L, ll R){
    if (tree[o].lazy)
        tree[o].sum1 = num[R + 1] - num[L];
    else
        tree[o].sum1 = tree[lo].sum1 + tree[ro].sum1;
}
void pushdown(ll o, ll L, ll R){
    if (tree[o].lazy2) {
        ll mid = (L + R) >> 1;
        do_odd(lo, L, mid);
        do_odd(ro, mid + 1, R);
        tree[lo].lazy2 ^= 1;
        tree[ro].lazy2 ^= 1;
        tree[o].lazy2 = 0;
    }
}
void addval(ll o, ll L, ll R){
    if (num[R + 1] <= ql || num[L] >= qr)
        return ;
    if (ql <= num[L] && num[R + 1] <= qr) {
        tree[o].lazy += k;
        do_odd(o, L, R);
        tree[o].lazy2 ^= 1;
        pushup(o, L, R);
        return ;
    }
    pushdown(o, L, R);
    ll M = (L + R) / 2;
    addval(lo, L, M);
    addval(ro, M + 1, R);
    pushup(o, L, R);
    tree[o].sum2 = tree[lo].sum2 + tree[ro].sum2;
}
```

树链剖分

```
void dfs1(int x, int f, int d){
    fa[x] = f;
    siz[x] = 1;
    dep[x] = d;
    int maxson = -1;
    for (int i = first[x]; i; i = inext[i]) {
        if (to[i] == f)
            continue;
        dfs1(to[i], x, d + 1);
        siz[x] += siz[to[i]];
        if (siz[to[i]] > maxson)
            maxson = siz[to[i]], son[x] = to[i];
    }
}
void dfs2(int x, int topf){
    id[x] = ++cur;
    top[x] = topf;
    w2[cur] = w1[x];
    if (!son[x])
        return ;
    dfs2(son[x], topf);
    for (int i = first[x]; i; i = inext[i]) {
        if (to[i] != fa[x] && to[i] != son[x])
            dfs2(to[i], to[i]);
    }
}
int lss(int beg, int end){
    int ans = 0;
    while (top[beg] != top[end]) {
        if (dep[top[beg]] < dep[top[end]])
            swap(beg, end);
        ans +=sum(1, 1, n, id[top[beg]], id[beg]);
        beg = fa[top[beg]];
    }
    if (dep[beg] > dep[end])
        swap(beg, end);
    ans += sum(1, 1, n, id[beg], id[end]);
    return ans ;
}
int imax(int beg, int end){
    int ans = -inf;
    while (top[beg] != top[end]) {
        if (dep[top[beg]] < dep[top[end]])
```

```
            swap(beg, end);
        ans = max(ans, query(1, 1, n, id[top[beg]], id[beg]));
        beg = fa[top[beg]];
    }
    if (dep[beg] > dep[end])
        swap(beg, end);
    ans = max(ans, query(1, 1, n, id[beg], id[end]));
    return ans ;
}
```

可持久化并查集

```
inline int find(int root, int x){
    node now = Tree[root];
    if (now.l == now.r)
        return now.fa;
    int mid = (now.l + now.r) >> 1;
    if (x <= mid)
        return find(now.ls,x);
    else
        return find(now.rs,x);
}
inline int getfa(int root, int x){
    int fa = find(root, x);
    if (fa == x)
        return x;
    else
        return getfa(root, fa);
}
inline int update(const int pre, int loc, int val){
    int now = tot++;
    copy(now, pre);
    if (Tree[now].l == Tree[now].r ) {
        Tree[now].fa = val;
        return now;
    }
    int mid = (Tree[now].l + Tree[now].r) >> 1;
    if (loc <= mid)
        Tree[now].ls = update(Tree[now].ls,loc,val);
    else
        Tree[now].rs = update(Tree[now].rs,loc,val);
    return now;
}
```

可持久化线段树

```
struct node {
    int val, l, r;
    int ls, rs;
} Tree[maxn<<5];
int a[maxn], rt[maxn];
int tot = 0;
int ver, op, kth, x, y;
vector<int>b;
```

```
inline int build(int l, int r){
    int now = tot++;
    Tree[now].l = l;
    Tree[now].r = r;
    if (l == r) {
        Tree[now].val = 0;
        return now;
    }
    int mid = (l + r) >> 1;
    Tree[now].ls = build(l, mid);
    Tree[now].rs = build(mid + 1, r);
    Tree[now].val = 0;
    return now;
}
inline int query(int x, int y, int k){
    node xver = Tree[x], yver = Tree[y];
    if (xver.l == xver.r)
        return xver.l;
    int mid = (xver.l + xver.r) >> 1;
    if (Tree[yver.ls].val - Tree[xver.ls].val >= k)
        return query(xver.ls, yver.ls, k);
    else
        return query(xver.rs, yver.rs, k - (Tree[yver.ls].val -
Tree[xver.ls].val));
}
inline int update(int pre){
    int now = tot++;
    Tree[now].l = Tree[pre].l;
    Tree[now].r = Tree[pre].r;
    Tree[now].ls = Tree[pre].ls;
    Tree[now].rs = Tree[pre].rs;
    Tree[now].val = Tree[pre].val + 1;
    if (Tree[now].l == Tree[now].r )
        return now;
    int mid = (Tree[now].l + Tree[now].r) >> 1;
    if (kth <= mid)
        Tree[now].ls = update(Tree[now].ls);
    else
        Tree[now].rs = update(Tree[now].rs);
    return now;
}
int getid(int x){
    return lower_bound(b.begin(), b.end(), x) - b.begin() + 1;
}
int main()
{
    int N, M, K;
    N = read();
    M = read();
    for (int i = 1; i <= N; ++i) {
        a[i] = read();
        b.push_back(a[i]);
    }
    sort(b.begin(), b.end());
    b.erase((b.begin(), b.end()),b.end());
    rt[0] = build(1, b.size());
    for (int i = 1; i <= N; ++i) {
        kth = getid(a[i]);
        rt[i]=update(rt[i - 1]);
    }
    for (int i = 1; i <= M; ++i) {
        x = read();
        y = read();
        kth = read();
        cout << b[query(rt[x - 1], rt[y], kth)-1] << endl;
    }
    return 0;
}
```

SBT

```
void rotate(int &rt, int k){
    int nrt = son[rt][!k]; //new root
    son[rt][!k] = son[nrt][k];
    son[nrt][k] = rt;
    siz[nrt] = siz[rt];
    siz[rt] = siz[son[rt][0]] + siz[son[rt][1]] + 1;
    rt = nrt;
}
void maintain(int &rt, bool k){
    if (siz[son[son[rt][k]][k]] > siz[son[rt][!k]])
        rotate(son[rt][k], !k);
    else if (siz[son[son[rt][k]][!k]] > siz[son[rt][!k]])
        rotate(son[rt][k], k), rotate(rt, !k);
    else
        return ;
    maintain(son[rt][0], 0);
    maintain(son[rt][1], 1);
    maintain(rt, 0);
    maintain(rt, 1);
}
void insert(int &rt, int val){
    if (rt == 0) {
        rt = ++tails;
        datas[rt] = val;
        son[rt][0] = son[rt][1] = 0;
        siz[rt] = 1;
        return ;
    }
    ++siz[rt];
    if (val <= datas[rt])
        insert(son[rt][0], val);
    else
        insert(son[rt][1], val);
    maintain(rt, datas[rt] > val);
}
void del(int &rt, int val){
    if (datas[rt] != val) {
        del(son[rt][datas[rt] < val], val);
        siz[rt] = siz[son[rt][0]] + siz[son[rt][1]] + 1;
        return ;
    }
    --siz[rt];
    if (son[rt][0] == 0)
        rt = son[rt][1];
    else if (son[rt][1] == 0)
        rt = son[rt][0];
    else {
        int p = son[rt][1];
        while (son[p][0] != 0)
            p = son[p][0];
        datas[rt] = datas[p];
        del(son[rt][1], datas[p]);
    }
}
int pre (int val){
    int cur = sbt, ans = 0;
    while (cur != 0) {
        if (datas[cur] < val)
            ans = cur, cur = son[cur][1];
        else
            cur = son[cur][0];
    }
    return datas[ans];
}
int nxt (int val){
    int cur = sbt, ans = 0;
    while (cur != 0) {
        if (datas[cur] > val)
            ans = cur, cur = son[cur][0];
        else
            cur = son[cur][1];
    }
    return datas[ans];
}
int xth(int x){
    int cur = sbt;
    while (siz[son[cur][0]] + 1 != x) {
        if (siz[son[cur][0]] + 1 < x)
            x -= (siz[son[cur][0]] + 1), cur = son[cur][1];
        else
            cur = son[cur][0];
    }
    return datas[cur];
}
int ranks(int val){
    int cur = sbt, ans = 0;
    while (cur != 0) {
        if (datas[cur] < val)
            ans += (siz[son[cur][0]] + 1), cur = son[cur][1];
        else
            cur = son[cur][0];
    }
    return ans + 1;
}
```

Splay

```
int sz[maxn << 2], son[maxn << 2][2], fa[maxn << 2], cnt[maxn << 2];
val_t val[maxn << 2];
int rt = 0, tot = 0;
struct Splay {
public:
    void maintain(int x){
        sz[x] = sz[son[x][0]] + sz[son[x][1]] + cnt[x];
    }
    bool getson(int x){
        return son[fa[x]][1] == x;
    }
    void clear(int x){
        son[x][0] = son[x][1] = fa[x] = val[x] = sz[x] = cnt[x] = 0;
    }
    void splay(int x){
        for (int f = fa[x]; f = fa[x], f; rotate(x)) {
            if (fa[f])
                rotate(getson(x) == getson(f) ? f : x);
        }
        rt = x;
    }
    void rotate(int x){
        int flag = getson(x), y = fa[x], z = fa[y];
        son[y][flag] = son[x][flag ^ 1];
        if (son[x][flag ^ 1])
            fa[son[x][flag ^ 1]] = y;
        son[x][flag ^ 1] = y;
        fa[y] = x;
        fa[x] = z;
        if (z)
            son[z][y == son[z][1]] = x;
        maintain(x);
        maintain(y);
    }
    void insert(val_t 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 = son[cur][val[cur] < k];
            if (!cur) {
                tot++;
                fa[tot] = f;
                val[tot] = k;
                ++cnt[tot];
                son[f][val[f] < k] = tot;
                maintain(tot);
                maintain(f);
                splay(tot);
                break;
            }
        }
    }
    val_t find_kth(int k){
        int cur = rt;
        while (1) {
            if (son[cur][0] && k <= sz[son[cur][0]])
                cur = son[cur][0];
            else if ( k <= cnt[cur] + sz[son[cur][0]]) {
                splay(cur);
                return val[cur];
            }
            else
                k -= (cnt[cur] + sz[son[cur][0]]), cur = son[cur][1];
        }
    }
    int find_rk(val_t k){
        int cur = rt, rk = 0;
        while (1) {
            if (val[cur] > k)
                cur = son[cur][0];
            else {
                rk += sz[son[cur][0]];
                if (k == val[cur]) {
                    splay(cur);
                    return rk + 1;
                }
                rk += cnt[cur];
                cur = son[cur][1];
            }
        }
    }
    int pre(){
        int cur = son[rt][0];
        while (son[cur][1])
            cur = son[cur][1];
        splay(cur);
        return cur;
    }
    int nxt(){
        int cur = son[rt][1];
        while (son[cur][0])
            cur = son[cur][0];
        splay(cur);
        return cur;
    }
    void del(int k){
        find_rk(k);
        if (cnt[rt] > 1) {
            --cnt[rt];
            maintain(rt);
            return ;
        }
        if (!son[rt][0] && !son[rt][1]) {
            clear(rt);
            rt = 0;
            return ;
        }
        if (!son[rt][0]) {
            int cur = rt;
            rt = son[rt][1];
            fa[rt] = 0;
            clear(cur);
            return ;
        }
        if (!son[rt][1]) {
            int cur = rt;
            rt = son[rt][0];
            fa[rt] = 0;
            clear(cur);
            return ;
        }
        int cur = rt, x = pre();
        fa[son[cur][1]] = x;
        son[x][1] = son[cur][1];
        clear(cur);
        maintain(rt);
        return ;
    }
} T;
```

矩形树

```
typedef long long ll;
ll num[maxn][maxn], k, ans;
ll n, m, qx1, qx2, qy1, qy2, tot = 1, rt = 1, q;
#define o0 (tree[o].son[0])
#define o1 (tree[o].son[1])
#define o2 (tree[o].son[2])
#define o3 (tree[o].son[3])
#define len(a,b,c,d) (((b)-(a)+1)*((d)-(c)+1))
struct node {
    ll val, lazy;
    ll son[4];
} tree[maxn * maxn * 4];
void pushup(ll o){
    tree[o].val = tree[o0].val + tree[o1].val + tree[o2].val + tree[o3].val;
}
inline void build(ll o, ll x1, ll x2, ll y1, ll y2){
    tree[o].lazy = 0;
    if (x1 == x2 && y1 == y2) {
        tree[o].val = num[x1][y1];
        return ;
    }
    ll midx = (x1 + x2) / 2, midy = (y1 + y2) / 2;
    if (y1 == y2) {
        tree[o].son[0] = ++tot;
        build(o0, x1, midx, y1, y1);
        tree[o].son[1] = ++tot;
        build(o1, midx + 1, x2, y2, y2);
    }
    else if (x1 == x2) {
        tree[o].son[0] = ++tot;
        build(o0, x1, x1, y1, midy);
        tree[o].son[2] = ++tot;
        build(o2, x1, x1, midy + 1, y2);
    }
    else {
        tree[o].son[0] = ++tot;
        build(o0, x1, midx, y1, midy);
        tree[o].son[1] = ++tot;
        build(o1, midx + 1, x2, y1, midy);
        tree[o].son[2] = ++tot;
        build(o2, x1, midx, midy + 1, y2);
        tree[o].son[3] = ++tot;
        build(o3, midx + 1, x2, midy + 1, y2);
    }
    pushup(o);
}

void pushdown(ll o, ll x1, ll x2, ll y1, ll y2){
    ll midx = (x1 + x2) / 2, midy = (y1 + y2) / 2;
    tree[o0].lazy += tree[o].lazy;
    tree[o1].lazy += tree[o].lazy;
    tree[o2].lazy += tree[o].lazy;
    tree[o3].lazy += tree[o].lazy;
    tree[o0].val += len(x1, midx, y1, midy) * tree[o].lazy;
    tree[o1].val += len(midx + 1, x2, y1, midy) * tree[o].lazy;
    tree[o2].val += len(x1, midx, midy + 1, y2) * tree[o].lazy;
    tree[o3].val += len(midx + 1, x2, midy + 1, y2) * tree[o].lazy;
    tree[o].lazy = 0;
}
inline ll getsum(ll o, ll x1, ll x2, ll y1, ll y2){
    ll res = 0;
    if (qx1 <= x1 && x2 <= qx2 && qy1 <= y1 && y2 <= qy2)
        return tree[o].val;
    pushdown(o, x1, x2, y1, y2);
    ll midx = (x1 + x2) / 2, midy = (y1 + y2) / 2;
    if (midx >= qx1 && midy >= qy1)
        res += getsum(o0, x1, midx, y1, midy);
    if (midx < qx2 && midy >= qy1)
        res += getsum(o1, midx + 1, x2, y1, midy);
    if (midx >= qx1 && midy < qy2)
        res += getsum(o2, x1, midx, midy+1, y2);
    if (midx < qx2 && midy < qy2)
```

```
        res += getsum(o3, midx + 1, x2, midy + 1, y2);
    return res;
}
inline void addval(ll o, ll x1, ll x2, ll y1, ll y2){
    if (qx1 <= x1 && x2 <= qx2 && qy1 <= y1 && y2 <= qy2) {
        tree[o].val += k * len(x1, x2, y1, y2);
        tree[o].lazy += k;
        return ;
    }
    pushdown(o, x1, x2, y1, y2);
    ll midx = (x1 + x2) / 2, midy = (y1 + y2) / 2;
    if (midx >= qx1 && midy >= qy1)
        addval(o0, x1, midx, y1, midy);
    if (midx < qx2 && midy >= qy1)
        addval(o1, midx + 1, x2, y1, midy);
    if (midx >= qx1 && midy < qy2)
        addval(o2, x1, midx, midy+1, y2);
    if (midx < qx2 && midy < qy2)
        addval(o3, midx + 1, x2, midy + 1, y2);
    pushup(o);
}
int main(){
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            Read(num[i][j]);
    build(rt, 1, n, 1, m);
    for (int i = 1; i <= q; i++) {
        ll op;
        if (qx1 > qx2)
            swap(qx1, qx2);
        if (qy1 > qy2)
            swap(qy1, qy2);
        if (op == 2) {
            ans = getsum(rt, 1, n, 1, m);
            printf("%lld\n", ans);
        }
        if (op == 1) {
            Read(k);
            addval(rt, 1, n, 1, m);
        }
    }
    return 0;
}
```

莫队 回滚莫队

```
int ranger[maxn], rangel[maxn], block[maxn];
int cnt[maxn], cnt2[maxn];
ll ans[maxn];
int num[maxn], id[maxn], numuni[maxn];
int unitot = 0;
struct query {
    int l, r, id;
} q[maxn];
bool cmp(query a, query b){
    return (block[a.l] ^ block[b.l]) ? block[a.l] < block[b.l] : a.r < b.r;
}
int main(){
    int N, Q;
    scanf("%d", &N);
    int sqn = sqrt(N);
    int blockn = ceil((double) N / sqn);
    for (int i = 1; i <= blockn; ++i) {
        ranger[i] = sqn * i;
        rangel[i] = sqn * (i - 1) + 1;
        for (int j = rangel[i]; j <= ranger[i]; ++j)
            block[j] = i;
    }
    ranger[blockn] = N;
    for (int i = 1; i <= N; ++i) {
        scanf("%d", &num[i]);
        numuni[i] = num[i];
    }
    sort(numuni + 1, numuni + 1 + N);
    unitot = unique(numuni + 1, numuni + 1 + N) - numuni - 1;
    for (int i = 1; i <= N; ++i)
        id[i] = lower_bound(numuni + 1, numuni + 1 + unitot, num[i]) - numuni;
    scanf("%d", &Q);
    for (int i = 1; i <= Q; ++i) {
        scanf("%d %d", &q[i].l, &q[i].r);
        q[i].id = i;
    }
    sort(q + 1, q + 1 + Q, cmp);
    int i = 1;
    for (int k = 0; k <= blockn; ++k) {
        int l = ranger[k] + 1, r = ranger[k];
        memset(cnt, 0, sizeof(cnt));
        ll temp = 0, now = 0;
        for (; block[q[i].l] == k; ++i) {
            int ql = q[i].l, qr = q[i].r;
            if (block[q[i].l] == block[q[i].r]) {
                temp = 0;
                for (int j = ql; j <= qr; ++j)
                    cnt2[id[j]] = 0;
                for (int j = ql; j <= qr; ++j) {
                    if (!cnt2[id[j]])
                        cnt2[id[j]] = j;
                    else
                        temp = max(temp, 1ll * (j - cnt2[id[j]]));
                }
                ans[q[i].id] = temp;
                continue;
            }
            while (r < qr) {
                if (!cnt[id[r]])
                    cnt[id[++r]] = r;
                else
                    now = max(now, 1ll * (r - cnt[id[r]]));
            }
            temp = now;
            while (ql < l) {
                if (!cnt[id[--l]])
                    cnt[id[l]] = l;
                else
                    temp = max(temp, 1ll * abs(l - cnt[id[l]]));
            }
            ans[q[i].id] = temp;
            while (l < ranger[k] + 1)
                if (cnt[id[l]] < ranger[k] + 1)
                    cnt[id[l++]] = 0;
                else ++l;
        }
    }
    for (int i = 1; i <= Q; ++i)
        printf("%lld\n", ans[i]);
    return 0;
}
```

字符串

kmp

```
const int maxn=1e6;
char str1[maxn],str2[maxn];
int nexts[maxn],ans[maxn];
int cnt=0;
void getnextarr(char* str){
    int n=strlen(str);
    nexts[0]=-1,nexts[1]=0;
    for (int i=2;i<=n;i++){
        int k=i-1;
        while (1){
            if (nexts[k]==-1 || str[i-1]==str[nexts[k]]){
                nexts[i]=nexts[k]+1;
                break;
            }
            else k=nexts[k];
        }
    }
}
void kmp(){
    int len1=strlen(str1);
    int len2=strlen(str2);
    int i=0,j=0;
    getnextarr(str2);
    while (i<len1){
        if (str1[i]==str2[j]){
                if (j!=len2-1) i++,j++;
            else {
                ans[cnt++]=i-j+1;
                j=nexts[j];
            }
        }
        else if (nexts[j]==-1) i++;
        else j=nexts[j];
    }
    for (int t=0;t<cnt;t++) printf("%d\n",ans[t]);
    for (int t=0;t<len2;t++) {
         printf("%d ",nexts[t+1]);
    }
}
```

AC自动机

```
class Automaton
{
public:
    static const int maxn = 2e2 + 10, maxs = 1e6 + 10, maxtmp = 1e3 + 10;
    /**
     * @brief a是匹配串,s是文本串,
     */
    char a[maxn][maxtmp], s[maxs];
    int tot = 0;
    struct node {
        int f, book, next[26];
    } tree[maxs];
    /**
     * @brief 对Trie树节点初始化
     */
    void clean(int x)
    {
        memset(tree[x].next, 0, sizeof(tree[x].next));
        tree[x].f = 0;
        tree[x].book = 0;
    }
    /**
     *@brief 对每个字符串更新Trie树,随后调用getfail()
     */
    void build(int x)
    {
        int cur = 0, l = strlen(a[x]);
        for (int i = 0; i < l; ++i) {
            if (!tree[cur].next[a[x][i] - 'a']) {
                tree[cur].next[a[x][i] - 'a'] = ++tot;
                clean(tot);
            }
            cur = tree[cur].next[a[x][i] - 'a'];
        }
        tree[cur].book = x;
    }
    void getfail()
    {
        queue<int> q; tree[0].f = 0;
        for (int i = 0; i < 26; i++)
            if (tree[0].next[i] != 0) {
                tree[tree[0].next[i]].f = 0;
                q.push(tree[0].next[i]);
            }
```

```cpp
        while (!q.empty()) {
            int now = q.front(); q.pop();
            for (int i = 0; i < 26; i++)
                if (tree[now].next[i]) {
                    tree[tree[now].next[i]].f = tree[tree[now].f].next[i];
                    q.push(tree[now].next[i]);
                }
                else
                    tree[now].next[i] = tree[tree[now].f].next[i];
        }
    }
} aca;
int main(){
    int now = 0, j = -1, l = strlen(aca.s);
    for (int i = 0; i < l; ++i) {
        now = aca.tree[now].next[aca.s[i] - 'a'];
        for (int j = now; j != 0; j = aca.tree[j].f)
            ans[aca.tree[j].book].num++;
    }
    return 0;
}
```

图论

最大流

```cpp
/**
 * @param e 边的总集合
 * @param f 连接每个节点的边的集合
 * @param cur 用于弧优化，记录当前点增广到哪一条边，之前的边不需要重复增广
 * @param vis BFS使用
 * @param d 节点层数
 */
class dinic
{
public:
    int s, t, n, m;
    static const int maxn = 100010;
    static const int dmaxn = 10010;
    // static const int INF = 0x7fffffff;
    static const int inf = 0x7fffffffffffffff;
    vector<int>f[dmaxn];
    bool vis[dmaxn];
    int cur[dmaxn], d[dmaxn];
    struct edge {
        int to, cap, flow;
    };
    vector<edge>e;
    /**
     * @brief 读入图的边
     * @param from 边的起点
     * @param to 边的终点
     * @param cap 边的最大流量
     */
    void addedge(int from, int to, int cap)
    {
        int m = e.size();
        e.push_back((edge) { to, cap, 0 });
        f[from].push_back(m);
        e.push_back((edge) { from, 0, 0 });
        f[to].push_back(m + 1);
    }
    /**
     * @brief 算出当前图上节点的层数
     * @return 返回汇点是否可达
     */
    bool bfs()
    {
        queue<int>q;
        q.push(s);
        for (int i = 0; i <= n; ++i)
            vis[i] = 0;
        d[s] = 0;
        vis[s] = 1;
        while (!q.empty()) {
            int x = q.front(); q.pop();
            int len = f[x].size();
            for (int i = 0; i < len; i++) {
                edge &o = e[f[x][i]];
                if (!vis[o.to] && o.cap > o.flow) {
                    d[o.to] = d[x] + 1;
                    vis[o.to] = 1;
                    q.push(o.to);
                }
            }
        }
        return vis[t];
    }
    /**
     * @brief dfs寻找增广路，可以多次寻找。使用了当前弧优化
     * @param x 当前所在的节点
     * @param a 当前的流量
     * @return 本次增广路得到的流量
     */
    int dfs(int x, int a){
        if (x == t || a == 0)
            return a;
        int flow = 0; int r, len = f[x].size();
        // 请勿去除这个引用，这是使用了弧优化
        for (int &i = cur[x]; i < len; i++) {
            edge &o = e[f[x][i]];
            if (d[x] + 1 == d[o.to] && (r = dfs(o.to, min(a, o.cap - o.flow))) >
0) {
                o.flow += r;
                e[f[x][i] ^ 1].flow -= r;
                flow += r; a -= r;
                if (a == 0) break;
            }
        }
        return flow;
    }
    /**
     * @brief 求最大流，要求事先完成读入和加边
     * @return 返回最大流
     */
    int max_flow() {
        int flow = 0;
        while (bfs()) {
            memset(cur, 0, sizeof(cur));
            flow += dfs(s, inf);
        }
        return flow;
    }
} DI;
/**
 * @param E 边的总集合
 * @param e 连接每个节点的边的集合
 * @param a 每次增广路过程中到达每个节点的最大流量
 * @param p 每次增广路节每个点选择增广的边
 */
class EK
{
public:
    struct edgeNode {
        int from, to, cap, flow;
        edgeNode(int _from, int _to, int _cap, int _flow): from(_from), to(_to),
cap(_cap), flow(_flow) {}
    };
    int m, n, s, t;
    static const int maxn = 100010;
    static const int INF = 0x7fffffffffffffff;
    vector<edgeNode>E;
    vector<int>e[maxn];
    int a[maxn], p[maxn];
    void addedge(int from, int to, int cap)
    {
        E.push_back(edgeNode{from, to, cap, 0});
        E.push_back(edgeNode{to, from, 0, 0});
        e[from].push_back(E.size() - 2);
        e[to].push_back(E.size() - 1);
    }
    int max_flow()
    {
        int flow = 0;
        while (1) {
            queue<int>q;
            q.push(s);
            for (int i = 0; i <= n; ++i)
                a[i] = 0;
            a[s] = INF;
            while (!q.empty()) {
                int rt = q.front(); q.pop();
                for (int i = 0; i < e[rt].size(); ++i) {
                    auto o = E[e[rt][i]];
                    if (!a[o.to] && o.cap > o.flow) {
                        a[o.to] = min(a[rt], o.cap - o.flow);
                        p[o.to] = e[rt][i];
                        q.push(o.to);
                    }
                }
                if (a[t])
                    break;
            }
            if (!a[t])
                break;
            for (int i = t; i != s; i = E[p[i]].from) {
                E[p[i]].flow += a[t];
                E[p[i] ^ 1].flow -= a[t];
            }
            flow += a[t];
        }
        return flow;
    }
};
signed main()
{
    for (int i = 1; i <= DI.m; i++)
        DI.addedge(a, b, c);
    printf("%lld\n", DI.max_flow());
    return 0;
}
```

最小费用最大流

```
struct node {
    int v, cap, cost, back;
} o[100010];
int cnt = 1;
vector<int>e[100000];
int mcost = 0, mflow = 0;
int inq[100000], pre[100000], xb[100000], dis[100000], flow[100000];
queue<int>q;
void addnode(int from, int to, int val, int w){
    o[cnt].v = to; o[cnt].cap = val; o[cnt].cost = w;
    o[cnt].back = cnt + 1; e[from].push_back(cnt); ++cnt;
    o[cnt].v = from; o[cnt].cap = 0; o[cnt].cost = -w;
    o[cnt].back = cnt - 1; e[to].push_back(cnt); ++cnt;
}
int spfa(int s, int t){
    memset(dis, 127, sizeof(dis));
    memset(inq, 0, sizeof(inq));
    memset(pre, -1, sizeof(pre));
    while (!q.empty())
        q.pop();
    int inf = dis[0]; inq[s] = 1; q.push(s); dis[s] = 0; pre[s] = 0; flow[s] =
0x7fffffff;
    while (!q.empty()) {
        int tmp = q.front(); q.pop();
        inq[tmp] = 0;
        int len = e[tmp].size();
        for (int i = 0; i < len; i++) {
            int rea = e[tmp][i];
            if (o[rea].cap > 0 && dis[o[rea].v] > dis[tmp] + o[rea].cost) {
                dis[o[rea].v] = dis[tmp] + o[rea].cost;
                pre[o[rea].v] = tmp;
                xb[o[rea].v] = rea;
                flow[o[rea].v] = min(flow[tmp], o[rea].cap);
                if (!inq[o[rea].v]) {
                    inq[o[rea].v] = 1; q.push(o[rea].v);
                }
            }
        }
    }
    if (dis[t] >= inf)
        return 0;
    return 1;
}
void max_flow(int s, int t){
    while (spfa(s, t)) {
        int k = t;
        while (k != s) {
            o[xb[k]].cap -= flow[t];
            o[o[xb[k]].back].cap += flow[t];
            k = pre[k];
        }
        mflow += flow[t];
        mcost += flow[t] * dis[t];
    }
}
```

缩点

```
stack<int>q;
struct node {
    int u,v;
}e[1000000];
int node_cnt=0,n,m,inst[1000000],beg[1000000];
int dfn[1000000],low[1000000],Index=0;
int belong[1000000],bcnt=0,ly[1000000],gxw[1000000];
int first[1000000],next[1000000],val[1000000];
void add(int a,int b){
//  e[++node_cnt].u=a;e[node_cnt].v=b;
    next[++node_cnt]=first[a];first[a]=node_cnt;
}
void tarjan(int rt){
    low[rt]=dfn[rt]=++Index;
    q.push(rt);
    inst[rt]=true;
    for (int i=first[rt];i;i=next[i]){
        int v=e[i].v;
        if (!dfn[v]) {
            tarjan(v);
            low[rt]=min(low[rt],low[v]);
        }
        else if(inst[v]) low[rt]=min(low[rt],dfn[v]);
    }
    if (dfn[rt]==low[rt]) {
        int j;
        bcnt++;
        do
        {
            j=q.top();
            q.pop();
            inst[j]=false;
            gxw[bcnt]+=val[j];
            belong[j]=bcnt;
        } while(j!=rt);
    }
}
void dfs(int rt){
    if (ly[rt]) return ;
    ly[rt]=gxw[rt];
    int maxsum=0;
    for (int i=first[rt];i;i=next[i]){
        if (!ly[e[i].v]) dfs(e[i].v);
        maxsum=max(maxsum,ly[e[i].v]);
    }
    ly[rt]+=maxsum;
}
int main()
{
    int ans=0;
    memset(ly,0,sizeof(ly));
    memset(gxw,0,sizeof(gxw));
    memset(first,0,sizeof(first));
    memset(next,0,sizeof(next));
    memset(beg,0,sizeof(beg));
    scanf ("%d %d",&n,&m);
    for (int i=1;i<=n;i++){
        scanf ("%d",&val[i]);
    }
    for (int i=1;i<=m;i++){
        scanf ("%d%d",&e[i].u,&e[i].v);
        add(e[i].u,e[i].v);
    }
    for (int i=1;i<=n;i++){
        if (!dfn[i]) tarjan(i);
    }
/*  for (int i=1;i<=n;i++)
        gxw[belong[i]]+=val[i];*/
    memset(first,0,sizeof(first));
    memset(next,0,sizeof(next));
    node_cnt=0;
    for (int i=1;i<=m;i++){
        int y=e[i].u,z=e[i].v;
        if (belong[y]!=belong[z]){
            e[node_cnt+1].u=belong[y];e[node_cnt+1].v=belong[z];
            add(belong[y],belong[z]);
        }
    }
    for (int i=1;i<=bcnt;i++){
        if (!ly[i]) {
            dfs(i);
            ans=max(ans,ly[i]);
        }
    }
    printf("%d",ans);
    return 0;
}
```

二分图最大匹配

```
int check[10000],matches[10000],matching[10000];
int v[1000010],fir1[1000010],nex1[1000010],cnt=0;
bool dfs(int rt){
    for (int i=fir1[rt];i;i=nex1[i]){
        int val=v[i];
        if (!check[val]) {
            check[val]=true;
            if (!matching[val] || dfs(matching[val])){
                matches[rt]=val;
                matching[val]=rt;
                return true;
            }
        }
    }
    return false;
}
int main()
{
//  freopen("P3386 [模板]二分图匹配.in","r",stdin);
    int m,n,e,ans=0;
    scanf ("%d%d%d",&n,&m,&e);
    for (int i=1;i<=e;i++){
        int a,b;
        scanf ("%d%d",&a,&b);
        if (b>m || a>n) continue;
        v[++cnt]=b,nex1[cnt]=fir1[a],fir1[a]=cnt;
    }
    for (int i=1;i<=n;i++){
        if (!matches[i]){
            memset(check,0,sizeof(check));
            if (dfs(i))
                ans++;
        }
    }
    printf("%d\n",ans);
    return 0;
}
```

割点

```
int v[500005],first[100005],next[500005];
int num[100005],low[100005],root=1,index=0,ans=0;
bool book[100005];
int dfs(int cur,int fa){
    ++index;
    int child=0;
    num[cur]=index;low[cur]=index;
    for (int i=first[cur];i;i=next[i]){
        if (num[v[i]]==0){
            child++;
            dfs(v[i],cur);
            low[cur]=min(low[cur],low[v[i]]);
            if (cur!=root && low[v[i]]>=num[cur] && !book[cur])
book[cur]=true,ans++;
            if(cur==root && child==2 && !book[cur]) book[cur]=true,ans++;
        }
        else if (v[i]!=fa) low[cur]=min(low[cur],num[v[i]]);
    }
}
int main()
{
//  freopen("[模版] 割点.in","r",stdin);
    int n,m,tot=1;
    cin>>n>>m;
    for (int i=1;i<=m;i++){
        int a,b,c;
        cin>>a>>b;
        v[tot]=b;next[tot]=first[a];first[a]=tot;tot++;
        v[tot]=a;next[tot]=first[b];first[b]=tot;tot++;
    }
    for (root=1;first[root]==0;root++);
    dfs(root,root);
    printf("%d\n",ans);
    for (int i=1;i<=n;i++){
        if (book[i]) printf("%d ",i);
    }
    return 0;
}
```

LCA

```
#include <cstdio>
#include <iostream>
#include <cstring>
using namespace std;
int next[1000010],first[1000010],v[1000010],fa[1000010];
int searn[1000010],sear[1000010],nn[1000010],ans[1000010];
bool vis[500010];

int fifa(int x){
    if (fa[x]!=x)   fa[x]=fifa(fa[x]);
    return fa[x];
```

```cpp
}

int he(int x,int y){
    int root=fifa(x);
    fa[fifa(y)]=root;
    fifa(y);
    return 0;
}
void dfs(int root,int last){
    for (int i=first[root];i>0;i=next[i]){
        if (!vis[v[i]] && v[i]!=last) {
            dfs(v[i],root);
            he(root,v[i]);
            vis[v[i]]=1;
        }
    }
    for (int i=sear[root];i>0;i=searn[i]){
        if (vis[nn[i]]){
            ans[(i+1)/2]=fifa(nn[i]);
        }
    }
    return ;
}
void p(int m,int n){
    cout<<'v'<<endl;
    for (int i=1;i<=2*n-2;i++)  printf("%d ",v[i]);
    putchar(10);
    cout<<'f'<<' '<<'n'<<endl;
    for (int i=1;i<=2*n-2;i++)  printf("%d %d\n",first[i],next[i]);
    putchar(10);
}
int main()
{
    int n,m,s;
    scanf ("%d %d %d",&n,&m,&s);
    memset(sear,-1,sizeof(sear));
    memset(searn,-1,sizeof(searn));
    memset(next,-1,sizeof(next));
    memset(first,-1,sizeof(first));
    memset(vis,0,sizeof(vis));
    for (int i=1;i<=n;i++){
        fa[i]=i;
    }
    for (int i=1;i<n;i++){
        int u,t;
        scanf ("%d %d",&u,&t);
        next[2*i-1]=first[u];first[u]=2*i-1;v[2*i-1]=t;
        next[2*i]=first[t];first[t]=2*i;v[2*i]=u;
    }
    for (int i=1;i<=m;i++){
        int x,y;
        scanf ("%d %d",&x,&y);
        searn[2*i-1]=sear[x];sear[x]=2*i-1;nn[2*i-1]=y;
        searn[2*i]=sear[y];sear[y]=2*i;nn[2*i]=x;
    }
```

```
//   p(m,n);
    dfs(s,0);
    for (int i=1;i<=m;i++){
        printf("%d\n",ans[i]);
    }
    return 0;
}
```

最小生成树

```
struct bian{
    int beg,end,w;
}en[200001];
int d[5001];
bool cmp(bian a,bian b){
    return a.w<b.w;
}
int findfa(int x)
{
    if (d[x]!=x) d[x]=findfa(d[x]);
    return d[x];
}
void he(int x,int y)
{
    int root=findfa(x);
    d[findfa(y)]=root;
    findfa(y);
    return ;
}
int main()
{
    int n,m,imin=0;
    bool flag=true;
    scanf("%d%d",&n,&m);
    for (int i=1;i<=n;i++) d[i]=i;
    for (int i=1;i<=m;i++) {
        scanf ("%d%d%d",&en[i].beg,&en[i].end,&en[i].w);
    }
    sort(en+1,en+m+1,cmp);
    for (int i=1;i<=m;i++){
        if (findfa(en[i].beg)!=findfa(en[i].end)){
            he(en[i].beg,en[i].end);
            imin+=en[i].w;
        }
    }
    if (!flag) printf ("orz\n");
    else    printf("%d\n",imin);
    return 0;
}
```

杂七杂八

快速读入

```
inline char nc(){
    static char buf[100000], *p1 = buf, *p2 = buf;
    return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2)
? EOF : *p1++;
}
inline int read() {
  int x = 0, w = 1;
  char ch = 0;
  while (ch < '0' || ch > '9') {
     if (ch == '-') w = -1;
     ch = getchar();
  }
  while (ch >= '0' && ch <= '9') {
      x = (x<<3) + (x<<1) + (ch - '0');
      ch = getchar();
  }
  return x * w;
}
```

对拍

```
int main() {
  while (true) {
    system("gen > test.in");  // 数据生成器将生成数据写入输入文件
    system("test1.exe < test.in > a.out");  // 获取程序1输出
    system("test2.exe < test.in > b.out");  // 获取程序2输出
    if (system("fc a.out b.out")) {
      system("pause");  // 方便查看不同处
      return 0;
    }
  }
}
```