acm模板

1. 编译运行、对拍

`a.sh`

```bash
#!/bin/bash
# 编译运行cpp

# AddressSanitizer (ASan): 检测内存越界、悬挂指针等问题。 启用方式: -fsanitize=address
# ThreadSanitizer (TSan): 检测多线程数据竞争。 启用方式: -fsanitize=thread
# UndefinedBehaviorSanitizer (UBSan): 检测未定义行为（如整数溢出）。 启用方式: -fsanitize=undefin
# LeakSanitizer (LSan): 检测内存泄漏。 启用方式: -fsanitize=leak
# -fno-omit-frame-pointer : 检测到内存错误时打印函数调用栈，这个参数一直都带上
# g++ -std=c++23 a.cpp -o a
g++ -fsanitize=address -g -std=c++17 a.cpp -o a

if [ $? -ne 0 ]; then
    echo "编译失败"
    exit 1
fi
/usr/bin/time -f "time: %e s\nmem: %M KB" ./a < a.in > a.out
```

对拍

```bash
# mt19937_64 rng(random_device{}());
# ll rd(ll l, ll r) {return l+rng()%(r-l+1);}
g++ -std=c++20 a.cpp -o a
g++ -std=c++20 baoli.cpp -o baoli
g++ -std=c++20 mk.cpp -o mk
t=1
maxround=10000
while (($t < $maxround))
do
    ./mk > a.in
    ./a < a.in > a.out
    ./baoli < a.in > a.ans
    if diff -Z a.out a.ans; then
        echo "AC on test $t"
    else
        echo "WA on test $t"
        break
    fi
    ((t++))
done
```

2. 树状数组

```
/*
树形数组
支持单点修改，单点查询；单点修改，区间查询；区间修改，单点查询（不能区间查询）
进行区间修改时，只需将数组差分，用树状数组维护差分数组即可；
*/
int lowbit(int x) { return x & -x; }
const int N = 500;
struct TreeArr {
    int _size;
    vector<int> tree;
    TreeArr(){
        _size = N;
        tree.resize(N + 10, 0);
    }
    TreeArr(int size) {
        _size = size;
        tree.resize(_size + 10, 0);
    }
    void add(int i, int d) {
        while(i <= _size) {
            tree[i] += d;
            i += (i & -i);
        }
    }
    int sum(int i) {
        int res = 0;
        while(i > 0) {
            res += tree[i];
            i -= (i & -i);
        }
        return res;
    }
    int query(int l, int r) {
        return sum(r) - sum(l-1);
    }
};

struct TreeArrMatrix {
    int _size;
    vector<vector<int>> tree;
    TreeArrMatrix(){
        _size = N;
        tree.resize(N + 10, vector<int>(N + 10, 0));
```

```cpp
    }
    TreeArrMatrix(int size) {
        _size = size;
        tree.resize(_size + 10, vector<int>(_size + 10, 0));
    }
    void resize(int size){
        _size = size;
        tree.resize(_size + 10, vector<int>(_size + 10, 0));
    }
    void add(int x, int y, int d) {
        for(int i = x; i <= _size; i += lowbit(i)) {
            for(int j=y; j<=_size; j += lowbit(j)) {
                tree[i][j] += d;
            }
        }
    }
    int sum(int x, int y) {
        int res = 0;
        for(int i=x; i>0; i-= lowbit(i)) {
            for(int j=y; j>0; j -= lowbit(j)) {
                res += tree[i][j];
            }
        }
        return res;
    }
    int query(int xl, int yl, int xr, int yr) {
        return sum(xr, yr) - sum(xr, yl-1) - sum(xl-1, yr) + sum(xl-1, yl-1);
    }
};
```

3. 线段树

```
class segTree{
    //下标从1开始

    int N;//size of original array
    vector<ll> a;
    vector<ll> tree;//N<<2
    vector<ll> tag;//用于记录区间的修改量
public:
    segTree(int _N = 1e5){
        N = _N + 10;
        a = vector<ll>(N, 0);
        tree = vector<ll>(N*4, 0);
        tag = vector<ll>(N*4, 0);
    }
    segTree(vector<ll> _a) {
        a = _a;
        N = _a.size()+10;
        tree = vector<ll>(N*4, 0);
        tag = vector<ll>(N*4, 0);
        build_tree(1, 1, N);
    }
    void build_tree(int f, int l, int r)//f结点表示[l,r]的线段
    {
        if(l==r){tag[f]=0;tree[f]=a[l];return;}//叶子节点
        int mid=(l+r)/2;
        int lc=f*2, rc=f*2+1;
        build_tree(lc, l, mid);
        build_tree(rc, mid+1, r);
        tree[f]=tree[lc] + tree[rc];//push_up
    }

    void push_up(int f)
    {
        int lc=2*f, rc=2*f+1;
        tree[f]=tree[lc]+tree[rc];
    }

    void push_down(int f, int l, int r)
    {
        //f代表的[l,r]区间都加上k
        //将父亲结点的信息tag[]传给孩子节点，并更新孩子节点tree[]
        int mid=(l+r)/2;
        int lc=2*f, rc=2*f+1;
        tag[lc]+=tag[f];
        //tree[lc]+=(mid-l+1)*tag[lc];由于之前传递来的tag[]已经增加过了，现在只需增加新增的量tag[f
        tree[lc]+=(mid-l+1)*tag[f];
        tag[rc]+=tag[f];
        //tree[rc]+=(r-mid)*tag[rc];
        tree[rc]+=(r-mid)*tag[f];
        tag[f]=0;//增量已经传递过了
    }


    void update(int _l, int _r, ll k, int f, int l, int r)//[_l, _r]为修改区间
    {
        if(l>_r || r<_l)return;
        if(l>=_l && r<=_r){
            tree[f]+=k*(r-l+1);
            tag[f]+=k;
            return;//整个区间直接修改，不需要向下传递
        }
        push_down(f, l, r);

        int mid=(l+r)/2;
        int lc=f*2, rc=f*2+1;
        update(_l, _r, k, lc, l, mid);
        update(_l, _r, k, rc, mid+1, r);

        push_up(f);
    }

    ll quiry(int _l, int _r, int f, int l, int r)//查询[_l, _r]
    {
        ll ans=0;
        if(l>=_l && r<=_r)return tree[f];
        if(l>_r || r<_l)return 0;
        int mid=(l+r)/2;
        int lc=f*2, rc=f*2+1;
        push_down(f, l, r);//将当前结点的信息tag[]传给孩子
        ans+=quiry(_l, _r, lc, l, mid);
        ans+=quiry(_l, _r, rc, mid+1, r);
        return ans;
    }
public:
    void add(int _l, int _r, ll k) {
        update(_l, _r, k, 1, 1, N);
    }
    ll sum(int _l, int _r){
        return quiry(_l, _r, 1, 1, N);
    }
};
```

4. 并查集

```
struct DSU{
    vector<int> f,siz;
    DSU(){};
    DSU(int n){
        init(n);
    }
    void init(int n){
        f.resize(n);
        iota(f.begin(),f.end(),0);
        siz.assign(n,1);
    }
    int find(int x){
        while(x!=f[x]){
            x=f[x]=f[f[x]];
        }
        return x;
    }
    bool same(int x,int y){
        return find(x)==find(y);
    }
    bool merge(int x, int y){
        x=find(x);
        y=find(y);
        if(x==y){
            return false;
        }
        siz[x]+=siz[y];
        f[y]=x;
        return true;
    }
    int size(int x){
        return siz[find(x)];
    }
};
```

带权并查集

```cpp
const int N = 10 + 2e5;
ll n, m;
ll d[N];
ll s[N];
ll ans = 0;
ll find(ll x) {
    if(s[x] != x){
        int t = s[x];
        s[x] = find(s[x]);
        d[x] += d[t];
    }
    return s[x];
}
void merge(ll x, ll y, ll _d) {// x <= y       x->y
    ll fx = find(x), fy = find(y);
    if(fx != fy) {
        s[fx] = fy;
        d[fx] = d[y] - d[x] + _d;
    }
    else {
        if(d[x] - d[y] != _d) ++ans;
    }
}
int main() {
    std :: cin >> n >> m;
    //初始化?
    for(int i=0; i<=n; ++i) {s[i] = i; d[i] = 0;}
    while(m--) {
        ll a, b, v;
        cin >> a >> b >> v;//a <= b
        a--;
        merge(a, b, v);
    }
    cout << ans;
}
```

5. 拉格朗日插值

```cpp
constexpr int MOD = 998244353;
using LL = long long;
int inv(int k) {
  int res = 1;
  for (int e = MOD - 2; e; e /= 2) {
    if (e & 1) res = (LL)res * k % MOD;
    k = (LL)k * k % MOD;
  }
  return res;
}
// 返回 f 满足 f(x_i) = y_i
// 不考虑乘法逆元的时间，显然为 O(n^2)
std::vector<int> lagrange_interpolation(const std::vector<int> &x,
                                        const std::vector<int> &y) {
  const int n = x.size();
  std::vector<int> M(n + 1), px(n, 1), f(n);
  M[0] = 1;
  // 求出 M(x) = prod_(i=0..n-1)(x - x_i)
  for (int i = 0; i < n; ++i) {
    for (int j = i; j >= 0; --j) {
      M[j + 1] = (M[j] + M[j + 1]) % MOD;
      M[j] = (LL)M[j] * (MOD - x[i]) % MOD;
    }
  }
  // 求出 px_i = prod_(j=0..n-1, j!=i) (x_i - x_j)
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j)
      if (i != j) {
        px[i] = (LL)px[i] * (x[i] - x[j] + MOD) % MOD;
      }
  }
  // 组合出 f(x) = sum_(i=0..n-1)(y_i / px_i)(M(x) / (x - x_i))
  for (int i = 0; i < n; ++i) {
    LL t = (LL)y[i] * inv(px[i]) % MOD, k = M[n];
    for (int j = n - 1; j >= 0; --j) {
      f[j] = (f[j] + k * t) % MOD;
      k = (M[j] + k * x[i]) % MOD;
    }
  }
  return f;
}

int main() {
  std::ios::sync_with_stdio(false);
  std::cin.tie(nullptr);
  int n, k;
  std::cin >> n >> k;
  std::vector<int> x(n), y(n);
  for (int i = 0; i < n; ++i) std::cin >> x[i] >> y[i];
  const auto f = lagrange_interpolation(x, y);
  int v = 0;
  for (int i = n - 1; i >= 0; --i) v = ((LL)v * k + f[i]) % MOD;
  std::cout << v << '\n';
  return 0;
}
```

6. 整体二分

```cpp
//RangeKth
#include<bits/stdc++.h>
using namespace std;

struct node {
    int pos, num;
    bool operator <(const node &x)const{
        if(num != x.num)
            return num < x.num;
        return pos < x.pos;
    }
};
const int N = 10 + 2e5;
int n, m;
node arr[N] = {0};
//查询
int qid[N];
int l[N];
int r[N];
int k[N];

int ans[N];

int tree[N];//树状数组
void add(int i, int v) {
    while (i <= n) {
        tree[i] += v;
        i += (i & (-i));
    }
}
int sum(int i) {
    int res = 0;
    while (i > 0) {
        res += tree[i];
        i -= (i & (-i));
    }
    return res;
}
int query(int l, int r) {
    return sum(r) - sum(l-1);
}

void compute(int ql, int qr, int vl, int vr) {
    if(ql > qr) return;
    if(vl == vr) {
        for(int i=ql; i<=qr; ++i) {
            ans[qid[i]] = arr[vl].num;
        }
    }
    else {
        int mid = (vl + vr) / 2;
        for(int i=vl; i<=mid; ++i) {
            add(arr[i].pos, 1);
        }
        vector<int> lset, rset;
        for(int i=ql; i<=qr; ++i) {
            int id = qid[i];
            int cnt = query(l[id], r[id]);

            if(cnt >= k[id]) {
                lset.push_back(id);
            }
            else if(cnt < k[id]) {
                rset.push_back(id);
                k[id] -= cnt;
            }
        }
        for(int i=0; i<lset.size(); ++i) qid[ql + i] = lset[i];
        for(int i=0; i<rset.size(); ++i) qid[ql + lset.size() + i] = rset[i];

        for(int i=vl; i<=mid; ++i) {
            add(arr[i].pos, -1);
        }

        compute(ql, ql + lset.size() - 1, vl, mid);
        compute(ql + lset.size(), qr, mid + 1, vr);
    }
}
int main() {
    cin >> n >> m;
    for(int i=1; i<=n; ++i) {
        arr[i].pos = i;
        cin >> arr[i].num;
    }
    for(int i=1; i<=m; ++i) {
        cin >> l[i] >> r[i] >> k[i];
        qid[i] = i;
    }
    sort(arr+1, arr+1+n, less<node>());
    compute(1, m, 1, n);
    for(int i=1; i<=m; ++i)cout << ans[i] << endl;
    return 0;
}
```

7. 凸包

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Point {
    double x, y;
    bool operator < (const Point &a) const {if (x != a.x) return x < a.x;return y < a.y;}
    bool operator == (const Point &a) const {return x == a.x && y == a.y;}
    double mod() { return sqrt(x * x + y * y); }
    Point operator - (const Point &a) const {return {x - a.x, y - a.y};}
    double cross(const Point &a) const {return x * a.y - y * a.x;}
};

void convexHull() {
    int n;
    cin >> n;
    vector<Point> a(n);
    for (int i = 0; i < n; ++i) cin >> a[i].x >> a[i].y;

    if (n == 1) {
        cout << "0.00\n";
        return;
    }

    // 排序并去重
    sort(a.begin(), a.end());
    a.erase(unique(a.begin(), a.end()), a.end());
    n = a.size();

    vector<Point> hull;
    // 构建下凸包
    for (int i = 0; i < n; ++i) {
        while (hull.size() >= 2 &&
               (hull.back() - hull[hull.size()-2]).cross(a[i] - hull.back()) <= 0) {
            hull.pop_back();
        }
        hull.push_back(a[i]);
    }
    // 构建上凸包
    for (int i = n - 2, t = hull.size(); i >= 0; --i) {
        while (hull.size() > t &&
               (hull.back() - hull[hull.size()-2]).cross(a[i] - hull.back()) <= 0) {
            hull.pop_back();
        }
        hull.push_back(a[i]);
    }

    // 计算周长（凸包是闭合的，hull[0] 和 hull.back() 是同一个点）
    double ans = 0;
    for (int i = 1; i < hull.size(); ++i) {
        ans += (hull[i] - hull[i-1]).mod();
    }
    cout << fixed << setprecision(2) << ans << '\n';
}

// int main() {
//     ios::sync_with_stdio(0);
//     cin.tie(0), cout.tie(0);
//     convexHull();
//     return 0;
// }
```

8. 笛卡尔树

```
// stk 维护笛卡尔树中节点对应到序列中的下标
for (int i = 1; i <= n; i++) {
  int k = top;  // top 表示操作前的栈顶，k 表示当前栈顶
  while (k > 0 && w[stk[k]] > w[i]) k--;  // 维护右链上的节点
  if (k) rs[stk[k]] = i;  // 栈顶元素.右儿子 := 当前元素
  if (k < top) ls[i] = stk[k + 1];  // 当前元素.左儿子 := 上一个被弹出的元素
  stk[++k] = i;                     // 当前元素入栈
  top = k;
}
```

9. 离散化

```
void fun(){
    int n;
    vector<int> a(n);
    for(auto &x : a) cin >> x;
    vector<int> unic = a;
    sort(unic.begin(), unic.end());
    unic.resize(unique(unic.begin(), unic.end()) - unic.begin());
    for(int i=0; i<n; ++i) {
        a[i] = lower_bound(unic.begin(), unic.end(), a[i]) - unic.begin();
    }
}
```

10. manacher回文匹配

```
// manacher
// 扩展字符串
// 考虑当前位置i与回文右边界r的位置；考虑i关于回文中心c的对称点j的回文区间对i的回文区间的贡献
// 对每一个位置求最长回文半径p[i]，原字符串回文长度len = p[i] - 1;
// 扩展串回文子串的末位置 l2 对应在原字符串的末位置 l1 : l1 = l2/2 - 1
const int N = 1e7 + 1e6 + 10;
void sol() {
    string s;
    cin >> s;
    int n = s.length();
    vector<char> ss(2*n + 1);
    vector<int> p(2*n + 1);
    for(int i=0; i<2*n+1; ++i) {
        p[i] = 1;
        ss[i] = (i & 1 ? s[i/2] : '#');
    }
    //for(auto x : ss) cout << x;
    //cout << endl;
    n = 2*n + 1;
    int r = 0, c = -1;
    for(int i=0; i<n; ++i) {
        int j = 2*c - i;
        int len = (i < r? min(p[j], r-i) : 1);
        while(i + len < n && i-len >=0 && ss[i+len] == ss[i-len]) ++len;
        p[i] = len;
        if( i + len > r) {
            r = i + len;
            c = i;
        }
    }
    int ans = 0;
    for(int i=0; i<n; ++i) ans= max(ans, p[i]);
    --ans;
    cout << ans;
}
signed main() {
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int T = 1;
    //cin >> T;
    while(T--) sol();
    return 0;
}
```

```cpp
std::vector<int> manacher(std::string s) {
    std::string t = "#";
    for (auto it : s) {
        t += it;
        t += '#';
    }
    int n = t.size();
    std::vector<int> r(n);
    for (int i = 0, j = 0; i < n; i++) {
        if (2 * j - i >= 0 && j + r[j] > i) {
            r[i] = std::min(r[2 * j - i], j + r[j] - i);
        }
        while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
            r[i] += 1;
        }
        if (i + r[i] > j + r[j]) {
            j = i;
        }
    }
    return r;
}
```

11. 莫队

11.1 普通莫队

```cpp
// 普通莫队入门题，C++版
// 给定一个长度为n的数组arr，一共有q条查询，格式如下
// 查询 l r : 打印arr[l..r]范围上有几种不同的数字
// 1 <= n <= 3 * 10^4
// 1 <= arr[i] <= 10^6
// 1 <= q <= 2 * 10^5
#include <bits/stdc++.h>
using namespace std;
struct Query {
    int l, r, id;
};

const int MAXV = 1000001;
int n, q;
vector<int> arr;
vector<Query> query;

vector<int> bi;
vector<int> cnt;
int kind = 0;

vector<int> ans;

bool QueryCmp1(const Query &a, const Query &b) {
    if (bi[a.l] != bi[b.l]) {
        return bi[a.l] < bi[b.l];
    }
    return a.r < b.r;
}

bool QueryCmp2(const Query &a, const Query &b) {
    if (bi[a.l] != bi[b.l]) {
        return bi[a.l] < bi[b.l];
    }
    if ((bi[a.l] & 1) == 1) {
        return a.r < b.r;
    } else {
        return a.r > b.r;
    }
}
```

```cpp
void del(int num) {
    if (--cnt[num] == 0) {
        kind--;
    }
}

void add(int num) {
    if (++cnt[num] == 1) {
        kind++;
    }
}

void prepare() {
    int blen = (int)sqrt(n);
    bi.resize(n + 1);
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    sort(query.begin() + 1, query.end(), QueryCmp2);
}

void compute() {
    int winl = 1, winr = 0;
    for (int i = 1; i <= q; i++) {
        int jobl = query[i].l;
        int jobr = query[i].r;
        while (winl > jobl) {
            add(arr[--winl]);
        }
        while (winr < jobr) {
            add(arr[++winr]);
        }
        while (winl < jobl) {
            del(arr[winl++]);
        }
        while (winr > jobr) {
            del(arr[winr--]);
        }
        ans[query[i].id] = kind;
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin >> n;
    arr.resize(n + 1);
    for (int i = 1; i <= n; i++) {
        cin >> arr[i];
    }

    cin >> q;
    query.resize(q + 1);
    ans.resize(q + 1);
    for (int i = 1; i <= q; i++) {
        cin >> query[i].l >> query[i].r;
        query[i].id = i;
    }

    cnt.resize(MAXV, 0);
    prepare();
    compute();

    for (int i = 1; i <= q; i++) {
        cout << ans[i] << '\n';
    }
    return 0;
}
```

11.2 带修莫队

```cpp
// 带修莫队入门题，C++版
// 给定一个长度为n的数组arr，一共有m条操作，操作格式如下
// 操作 Q l r     : 打印arr[l..r]范围上有几种不同的数
// 操作 R pos val : 把arr[pos]的值设置成val
// 1 <= n、m <= 2 * 10^5
// 1 <= arr[i]、val <= 10^6
#include <bits/stdc++.h>
using namespace std;
struct Query {
    int l, r, t, id;
};
struct Update {
    int pos, val;
};

const int MAXN = 200001;
const int MAXV = 1000001;
int n, m;
int arr[MAXN];
int bi[MAXN];

Query query[MAXN];
Update update[MAXN];
int cntq, cntu;

int cnt[MAXV];
int kind;

int ans[MAXN];

bool QueryCmp(Query &a, Query &b) {
    if (bi[a.l] != bi[b.l]) {
        return bi[a.l] < bi[b.l];
    }
    if (bi[a.r] != bi[b.r]) {
        return bi[a.r] < bi[b.r];
    }
    return a.t < b.t;
}

void del(int num) {
    if (--cnt[num] == 0) {
        kind--;
    }
```

```
}

void add(int num) {
    if (++cnt[num] == 1) {
        kind++;
    }
}

void moveTime(int jobl, int jobr, int tim) {
    int pos = update[tim].pos;
    int val = update[tim].val;
    if (jobl <= pos && pos <= jobr) {
        del(arr[pos]);
        add(val);
    }
    int tmp = arr[pos];
    arr[pos] = val;
    update[tim].val = tmp;
}

void compute() {
    int winl = 1, winr = 0, wint = 0;
    for (int i = 1; i <= cntq; i++) {
        int jobl = query[i].l;
        int jobr = query[i].r;
        int jobt = query[i].t;
        int id = query[i].id;

        while (winl > jobl) {
            add(arr[--winl]);
        }
        while (winr < jobr) {
            add(arr[++winr]);
        }
        while (winl < jobl) {
            del(arr[winl++]);
        }
        while (winr > jobr) {
            del(arr[winr--]);
        }
        while (wint < jobt) {
            moveTime(jobl, jobr, ++wint);
        }
        while (wint > jobt) {
            moveTime(jobl, jobr, wint--);
        }
```

```cpp
        ans[id] = kind;
    }
}

void prepare() {
    int blen = max(1, (int)pow(n, 2.0 / 3));
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    sort(query + 1, query + cntq + 1, QueryCmp);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        cin >> arr[i];
    }

    char op;
    int l, r, pos, val;
    for (int i = 1; i <= m; i++) {
        cin >> op;
        if (op == 'Q') {
            cin >> l >> r;
            cntq++;
            query[cntq].l = l;
            query[cntq].r = r;
            query[cntq].t = cntu;
            query[cntq].id = cntq;
        } else {
            cin >> pos >> val;
            cntu++;
            update[cntu].pos = pos;
            update[cntu].val = val;
        }
    }

    prepare();
    compute();

    for (int i = 1; i <= cntq; i++) {
        cout << ans[i] << '\n';
    }
    return 0;
}
```

11.3 只删回滚莫队

```cpp
// 只删回滚莫队入门题，C++版
// 本题最优解为主席树，讲解158，题目2，已经讲述
// 给定一个长度为n的数组arr，一共有m条查询，格式如下
// 查询 l r : 打印arr[l..r]内没有出现过的最小自然数，注意0是自然数
// 0 <= n、m、arr[i] <= 2 * 10^5
struct Query {
    int l, r, id;
};

const int MAXN = 200001;
const int MAXB = 501;
int n, m;
vector<int> arr;
vector<Query> query;

int blen, bnum;
vector<int> bi;
vector<int> bl;

vector<int> cnt;
int mex;
vector<int> ans;

bool QueryCmp(const Query &a, const Query &b) {
    if (bi[a.l] != bi[b.l]) {
        return bi[a.l] < bi[b.l];
    }
    return b.r < a.r;
}

void del(int num) {
    if (--cnt[num] == 0) {
        mex = min(mex, num);
    }
}

void add(int num) {
    cnt[num]++;
}

void compute() {
    // 初始化计数数组
    for (int i = 1; i <= n; i++) {
        cnt[arr[i]]++;
    }

    // 计算初始mex
    mex = 0;
    while (cnt[mex] != 0) {
        mex++;
    }

    int winl = 1, winr = n;
    for (int block = 1, qi = 1; block <= bnum && qi <= m; block++) {
        // 移动左边界到当前块的起始位置
        while (winl < bl[block]) {
            del(arr[winl++]);
        }

        int beforeJob = mex;

        // 处理当前块内的所有查询
        for (; qi <= m && bi[query[qi].l] == block; qi++) {
            int jobl = query[qi].l;
            int jobr = query[qi].r;
            int id = query[qi].id;

            // 移动右边界到查询右边界
            while (winr > jobr) {
                del(arr[winr--]);
            }

            int backup = mex;

            // 移动左边界到查询左边界
            while (winl < jobl) {
                del(arr[winl++]);
            }

            ans[id] = mex;
            mex = backup;

            // 回滚左边界到当前块的起始位置
            while (winl > bl[block]) {
                add(arr[--winl]);
            }
        }

        // 恢复右边界到数组末尾
        while (winr < n) {
            add(arr[++winr]);
        }

        mex = beforeJob;
    }
}

void prepare() {
    blen = (int)sqrt(n);
    bnum = (n + blen - 1) / blen;

    // 初始化分块数组
    bi.resize(n + 1);
    bl.resize(bnum + 1);

    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    for (int i = 1; i <= bnum; i++) {
        bl[i] = (i - 1) * blen + 1;
    }

    // 排序查询
    sort(query.begin() + 1, query.end(), QueryCmp);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> n >> m;

    // 使用vector替代数组
    arr.resize(n + 1);
    query.resize(m + 1);
    cnt.resize(MAXN, 0);
    ans.resize(m + 1);

    for (int i = 1; i <= n; i++) {
        cin >> arr[i];
    }
    for (int i = 1; i <= m; i++) {
        cin >> query[i].l >> query[i].r;
        query[i].id = i;
    }
```

```cpp
    prepare();
    compute();

    for (int i = 1; i <= m; i++) {
        cout << ans[i] << '\n';
    }
    return 0;
}
```

11.4 树上莫队

```
// 树上莫队入门题，C++版
// 一共有n个节点，每个节点给定颜色值，给定n-1条边，所有节点连成一棵树
// 一共有m条查询，格式 u v : 打印点u到点v的简单路径上，有几种不同的颜色
// 1 <= n <= 4 * 10^4
// 1 <= m <= 10^5
// 1 <= 颜色值 <= 2 * 10^9
#include <bits/stdc++.h>
using namespace std;
struct Query {
   int l, r, lca, id;
};

const int MAXN = 40001;
const int MAXM = 100001;
const int MAXP = 20;

int n, m;
int color[MAXN];
int sorted[MAXN];
int cntv;

Query query[MAXM];

// 修改：使用 vector 替代链式前向星
vector<int> graph[MAXN];

int dep[MAXN];
int seg[MAXN << 1];
int st[MAXN];
int ed[MAXN];
int stjump[MAXN][MAXP];
int cntd;

int bi[MAXN << 1];

bool vis[MAXN];
int cnt[MAXN];
int kind;

int ans[MAXM];

int kth(int num) {
    int left = 1, right = cntv, mid, ret = 0;
    while (left <= right) {
```

```cpp
        mid = (left + right) / 2;
        if (sorted[mid] <= num) {
            ret = mid;
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return ret;
}

void dfs(int u, int fa) {
    dep[u] = dep[fa] + 1;
    seg[++cntd] = u;
    st[u] = cntd;
    stjump[u][0] = fa;
    for (int p = 1; p < MAXP; p++) {
        stjump[u][p] = stjump[stjump[u][p - 1]][p - 1];
    }
    // 修改：使用 vector 遍历邻接表
    for (int v : graph[u]) {
        if (v != fa) {
            dfs(v, u);
        }
    }
    seg[++cntd] = u;
    ed[u] = cntd;
}

int lca(int a, int b) {
    if (dep[a] < dep[b]) {
        swap(a, b);
    }
    for (int p = MAXP - 1; p >= 0; p--) {
        if (dep[stjump[a][p]] >= dep[b]) {
            a = stjump[a][p];
        }
    }
    if (a == b) {
        return a;
    }
    for (int p = MAXP - 1; p >= 0; p--) {
        if (stjump[a][p] != stjump[b][p]) {
            a = stjump[a][p];
            b = stjump[b][p];
        }
    }
    return stjump[a][0];
    }

bool QueryCmp(Query &a, Query &b) {
    if (bi[a.l] != bi[b.l]) {
        return bi[a.l] < bi[b.l];
    }
    return a.r < b.r;
}

void invert(int node) {
    int val = color[node];
    if (vis[node]) {
        if (--cnt[val] == 0) {
            kind--;
        }
    } else {
        if (++cnt[val] == 1) {
            kind++;
        }
    }
    vis[node] = !vis[node];
}

void compute() {
    int winl = 1, winr = 0;
    for (int i = 1; i <= m; i++) {
        int jobl = query[i].l;
        int jobr = query[i].r;
        int lca = query[i].lca;
        int id = query[i].id;
        while (winl > jobl) {
            invert(seg[--winl]);
        }
        while (winr < jobr) {
            invert(seg[++winr]);
        }
        while (winl < jobl) {
            invert(seg[winl++]);
        }
        while (winr > jobr) {
            invert(seg[winr--]);
        }
        if (lca > 0) {
            invert(lca);
        }
        ans[id] = kind;
        if (lca > 0) {
            invert(lca);
        }
    }
}

void prepare() {
    for (int i = 1; i <= n; i++) {
        sorted[i] = color[i];
    }
    sort(sorted + 1, sorted + n + 1);
    cntv = 1;
    for (int i = 2; i <= n; i++) {
        if (sorted[cntv] != sorted[i]) {
            sorted[++cntv] = sorted[i];
        }
    }
    for (int i = 1; i <= n; i++) {
        color[i] = kth(color[i]);
    }
    int blen = (int) sqrt(cntd);
    for (int i = 1; i <= cntd; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    sort(query + 1, query + m + 1, QueryCmp);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        cin >> color[i];
    }
    for (int i = 1, u, v; i < n; i++) {
        cin >> u >> v;
        // 修改：使用 vector 的 push_back 替代 addEdge
        graph[u].push_back(v);
        graph[v].push_back(u);
    }
    dfs(1, 0);
    for (int i = 1, u, v, uvlca; i <= m; i++) {
        cin >> u >> v;
        if (st[v] < st[u]) {
            swap(u, v);
        }
        uvlca = lca(u, v);
        if (u == uvlca) {
            query[i].l = st[u];
            query[i].r = st[v];
            query[i].lca = 0;
        } else {
            query[i].l = ed[u];
            query[i].r = st[v];
            query[i].lca = uvlca;
        }
        query[i].id = i;
    }
    prepare();
    compute();
    for (int i = 1; i <= m; i++) {
        cout << ans[i] << '\n';
    }
    return 0;
}
```

13. 常用类型模板

modnum

```
using ll = long long;
const ll mod = 998244353;
template<ll mod> // template was not stolen from https://codeforces.com/profile/SharpEdged
struct modnum {
    static constexpr bool is_big_mod = mod > numeric_limits<int>::max();

    using S = conditional_t<is_big_mod, ll, int>;
    using L = conditional_t<is_big_mod, __int128, ll>;

    S x;

    modnum() : x(0) {}
    modnum(ll _x) {
        _x %= static_cast<ll>(mod);
        if (_x < 0) { _x += mod; }
        x = _x;
    }

    modnum pow(ll n) const {
        modnum res = 1;
        modnum cur = *this;
        while (n > 0) {
            if (n & 1) res *= cur;
            cur *= cur;
            n /= 2;
        }
        return res;
    }
    modnum inv() const { return (*this).pow(mod-2); }

    modnum& operator+=(const modnum& a){
        x += a.x;
        if (x >= mod) x -= mod;
        return *this;
    }
    modnum& operator-=(const modnum& a){
        if (x < a.x) x += mod;
        x -= a.x;
        return *this;
    }
    modnum& operator*=(const modnum& a){
        x = static_cast<L>(x) * a.x % mod;
        return *this;
    }
    modnum& operator/=(const modnum& a){ return *this *= a.inv(); }

    friend modnum operator+(const modnum& a, const modnum& b){ return modnum(a) += b; }
    friend modnum operator-(const modnum& a, const modnum& b){ return modnum(a) -= b; }
    friend modnum operator*(const modnum& a, const modnum& b){ return modnum(a) *= b; }
    friend modnum operator/(const modnum& a, const modnum& b){ return modnum(a) /= b; }

    friend bool operator==(const modnum& a, const modnum& b){ return a.x == b.x; }
    friend bool operator!=(const modnum& a, const modnum& b){ return a.x != b.x; }
    friend bool operator<(const modnum& a, const modnum& b){ return a.x < b.x; }

    friend ostream& operator<<(ostream& os, const modnum& a){ os << a.x; return os; }
    friend istream& operator>>(istream& is, modnum& a) { ll x; is >> x; a = modnum(x); return
}
```

combination

```
using mint = modnum<mod>;
struct Combi{
    vector<mint> _fac, _ifac;
    int n;

    Combi() {
        n = 1;
        _fac.assign(n + 1, 1);
        _ifac.assign(n + 1, 1);
    }

    void check_size(int m){
        int need = n;
        while (need < m) need *= 2;
        m = need;
        if (m <= n) return;

        _fac.resize(m + 1);
        _ifac.resize(m + 1);
        for (int i = n + 1; i <= m; i++) _fac[i] = i * _fac[i - 1];

        _ifac[m] = _fac[m].inv();
        for (int i = m - 1; i > n; i--) _ifac[i] = _ifac[i + 1] * (i + 1);
        n = m;
    }

    mint fac(int m){
        check_size(m);
        return _fac[m];
    }

    mint ifac(int m){
        check_size(m);
        return _ifac[m];
    }

    mint ncr(int n, int r){//comb n选r
        if (n < r || r < 0) return 0;

        return fac(n) * ifac(n - r) * ifac(r);
    }

    mint npr(int n, int r){
        if (n < r || r < 0) return 0;
        return fac(n) * ifac(n - r);
    }
} comb;
```

Fraction分数

```
#include<bits/stdc++.h>
int gcd(int a,int b){if(b==0) return a;return gcd(b,a%b);}
int lcm(int a,int b){return a/gcd(a,b)*b;}

class Fraction {
public:
    int a,b;
    int sign(int x) {return (x>0?1:-1);}
    Fraction():a(0),b(1){}
    Fraction(int x):a(x),b(1){}
    Fraction(int x,int y)
    {
        int m = gcd(abs(x),abs(y));
        a = x/m*sign(y);
        if(a==0)b=1;else b = abs(y/m);
    }
    int get_denominator() {return b;}
    int get_numerator() {return a;}
    Fraction operator+(const Fraction &f)
    {
        int m = gcd(b,f.b);
        return Fraction(f.b/m*a+b/m*f.a,b/m*f.b);
    }
    Fraction operator-(const Fraction &f)
    {
        int m = gcd(b,f.b);
        return Fraction(f.b/m*a-b/m*f.a,b/m*f.b);
    }
    bool operator<(const Fraction &f)
    {
        int m = gcd(b,f.b);
        return f.b/m*a-b/m*f.a < 0;
    }
    bool operator<=(const Fraction &f)
    {
        int m = gcd(b,f.b);
        return f.b/m*a-b/m*f.a <= 0;
    }
    bool operator>(const Fraction &f)
    {
        int m = gcd(b,f.b);
        return f.b/m*a-b/m*f.a > 0;
    }
    bool operator>=(const Fraction &f)
    {
        int m = gcd(b,f.b);
        return f.b/m*a-b/m*f.a >= 0;
    }
    Fraction operator*(const Fraction &f)
    {
        int m1 = gcd(abs(a),f.b);
        int m2 = gcd(b,abs(f.a));
        return Fraction((a/m1)*(f.a/m2),(b/m2)*(f.b/m1));
    }
    Fraction operator/(const Fraction &f)
        {return (*this)*Fraction(f.b,f.a);}
    friend ostream &operator << (ostream &out,const Fraction &f)
    {
        if(f.a==0) cout << 0;
        else if(f.b==1) cout << f.a;
        else cout << f.a << '/' << f.b;
        return out;
    }
};
```

int128

```
/*
本文件为128位整数的模板
128位整数的范围大概为 e32
c++标准库对于__int128并没有配套的函数操作，仅支持+、-、*、/ 四则运算
对于输入、输出、开方等操作需要自己实现
*/
#include <bits/stdc++.h>
using namespace std;
__int128 mod=1e30;
__int128 powmod(__int128 a,__int128 b) {__int128 res=1;a%=mod; assert(b>=0); for(;b;b>>=1){if(
__int128 gcd(__int128 a,__int128 b) { return b?gcd(b,a%b):a;}

__int128 str_i128(const string& s) {
    __int128 res=0;
    for(auto si:s){
        res = 10*res + si-'0';
    }
    return res;
}

string i128_str(__int128 x) {
    string res="";
    bool sign =(x<0? 1 : 0);
    if(sign)x *= -1;
    while(x){
        res.append(1, '0'+x%10);//?
        x/=10;
    }
    if(sign)res.append(1, '1');
    reverse(res.begin(),res.end());
    return res;
}
__int128 read128() {
    string tem;
    cin>>tem;
    return str_i128(tem);
}
void put(__int128 x) {
    stack<int> sta;
    char sign = x>=0? '+':'-';
    while(x){
        sta.push(x%10);
        x/=10;
    }
    if(sta.size()==0){cout<<'0';return;}
    if(sign=='-')cout<<sign;
    while(sta.size()){
        cout<<sta.top();
        sta.pop();
    }
}
/*
void put(__int128 x) {
    cout<<i128_str(x);
}
*/
__int128 Sqrt(const __int128& x) {
    //x>=0
    __int128 L=0, R=1e19;
    while(L + 1 < R){
        __int128 M = (L + R)/2;
        if(M * M <=x){
            L = M;
        }
        else R = M;
    }
    return L;
}
```

point

```
struct Point {
    double x, y;
    bool operator < (const Point &a) const {if (x != a.x) return x < a.x;return y < a.y;}
    bool operator == (const Point &a) const {return x == a.x && y == a.y;}
    double mod() { return sqrt(x * x + y * y); }
    Point operator - (const Point &a) const {return {x - a.x, y - a.y};}
    double cross(const Point &a) const {return x * a.y - y * a.x;}
};
```