28. RMQ

RMQ解决静态区间最值查询的问题,常数小,代码短,但不支持修改.

28.1 一维RMQ

28.1.1 天才的记忆

题意

给定一个长度为$n\ \ (1 \leq n \leq 2e5)$的整数序列$a = [a_1, \cdots, a_n]$. 现有$m\ \ (1 \leq m \leq 1e4)$个询问, 每次询问给定一个区间$[l, r]\ \ (1 \leq l \leq r \leq n)$, 求$[l, r]$中的最大值.

思路

$dp[i][j]$表示以i为左端点长度为2^j的区间中的最大值, 显然$dp[i][j] = \max\{dp[i][j-1], dp[i+2^{j-1}][j-1]\}$. 状态$O(n \log n)$个, 转移$O(1)$, 故预处理$dp[i][j]$的时间复杂度为$O(n \log n)$. 其中$j$的最大值为$\lfloor \log_2 2e5 \rfloor = 17$.

对每个询问$[l, r]$,设 区间长度为len. 对 $s.t.\ 2^k \leq len, 2^{k+1} > len$的$k$, 则$[l, r]$中的最大值即$\max\{dp[l][k], dp[k - 2^{k-1} - 1][k]\}$, 查询$O(1)$. 其中的$k$可预处理.

代码

```cpp
const int MAXN = 2e5 + 5, MAXM = 18;
int n;
int a[MAXN];
int dp[MAXN][MAXM];  // dp[i][j]表示以i左端点长度为2^j的区间中的最大值

void init() {  // 预处理dp[][]
  for (int j = 0; j < MAXM; j++) {
    for (int i = 1; i + (1 << j) - 1 <= n; i++) {
      if (!j) dp[i][j] = a[i];  // 区间内只有一个数
      else dp[i][j] = max(dp[i][j - 1], dp[i + (1 << j - 1)][j - 1]);
    }
  }
}

int query(int l, int r) {
  int len = r - l + 1;
  int k = log(len) / log(2);
  return max(dp[l][k], dp[r - (1 << k) + 1][k]);
}

int main() {
  cin >> n;
  for (int i = 1; i <= n; i++) cin >> a[i];

  init();

  CaseT{
    int l, r; cin >> l >> r;
    cout << query(l, r) << endl;
```

```
  }
}
```

28.1.2 Boris and His Amazing Haircut

原题指路:https://codeforces.com/contest/1779/problem/D

题意 (2 s)

给定一个初始序列$a_1,\cdots,a_n$和一个目标序列$b_1,\cdots,b_n$.问能否通过至多m个操作将初始序列变为目标序列,每个操作用一个整数x描述,表示选择一个区间$[l,r]$ $(1\le l\le r\le n)$,令$a_i=\min\{a_i,x\}$ $(l\le i\le r)$.

有t $(1\le t\le 2e5)$组测试数据.每组测试数据第一行输入一个整数n $(1\le n\le 2e5)$.第二行输入n个整数$a_1,\cdots,a_n$ $(1\le a_i\le 1e9)$.第三行输入n个整数$b_1,\cdots,b_n$ $(1\le b_i\le 1e9)$.第四行输入一个整数m $(1\le m\le 2e5)$.第五行输入m个整数$x_1,\cdots,x_m$ $(1\le x_i\le 1e9)$,分别描述每个操作.数据保证所有测试数据的n之和不超过2e5.

思路

因操作后a_i不增,则存在下标$i\ s.t.\ a_i<b_i$时无解.

若操作改为单点修改,则对每个 $s.t.\ a_i>b_i$的下标i都需进行一次操作.考虑可将哪些单点修改合并为一次区间修改,显然能合并的前提是两个单调修改的x值相等.考虑在下标l,r $(1\le l<r\le n)$处的两个x值相等的单点修改,它们能合并为一个$[l,r]$的区间修改的充要条件是:$\max\limits_{l+1\le i\le r-1} b_i\le x$.用ST表支持查询区间最大值即可做到$O(n\log n)$.

代码

```cpp
template<typename T>
struct RMQ1D {  // 下标从1开始
  int n;
  vector<vector<T>> st;  // ST表
  const function<T(T, T)> cmp;  // 比较规则

  RMQ1D(int _n, const vector<T>& a, function<T(T, T)> _cmp) :n(_n), cmp(_cmp) {
    assert(a.size() > n);

    const int LG = log2(n);
    st.assign(LG + 1, vector<T>(n + 1));

    st[0] = a;
    for (int j = 1; j <= LG; j++) {
      for (int i = 1; i <= n - (1 << j) + 1; i++)
        st[j][i] = cmp(st[j - 1][i], st[j - 1][i + (1 << j - 1)]);
    }
  }

  T query(int l, int r) {  // 查询区间[l,r]中的最值
    int k = log2(r - l + 1);
    return cmp(st[k][l], st[k][r - (1 << k) + 1]);
  }
};

void solve() {
  int n; cin >> n;
  vector<int> a(n + 1), b(n + 1);
```

```
  for (int i = 1; i <= n; i++) cin >> a[i];
  for (int i = 1; i <= n; i++) cin >> b[i];
  map<int, int> cnt;
  CaseT{
    int x; cin >> x;
    cnt[x]++;
  }

  for (int i = 1; i <= n; i++) {
    if (a[i] < b[i]) {
      cout << "NO" << endl;
      return;
    }
  }

  auto rmq = RMQ1D<int>(n, b, [&](int a, int b) {
    return max(a, b);
  });

  map<int, vector<int>> poses;
  for (int i = 1; i <= n; i++) poses[b[i]].push_back(i);

  for (auto& [x, pos] : poses) {
    vector<int> cut;  // 需要操作的下标
    for (auto i : pos)
      if (a[i] != b[i]) cut.push_back(i);

    int need = cut.size();
    for (int i = 1; i < cut.size(); i++) {
      int l = cut[i - 1], r = cut[i];
      if (rmq.query(l, r) <= x) need--;
    }

    if (need > cnt[x]) {
      cout << "NO" << endl;
      return;
    }
  }
  cout << "YES" << endl;
}

int main() {
  CaseT
    solve();
}
```

28.1.3 Array Stabilization (GCD version)

原题指路:https://codeforces.com/contest/1547/problem/F

题意 (4 s)

给定一个长度为n $(n \geq 2)$的序列$a_0, \cdots, a_{n-1}$.现有操作:构造一个序列$b_0, \cdots, b_{n-1}$使得$b_i = \gcd(a_i, a_{(i+1) \bmod n})$,将$a[]$替换为$b[]$.求使得$a[]$中的所有元素都相同所需的最小操作次数.

有t $(1 \leq t \leq 1e4)$组测试数据.每组测试数据第一行输入一个整数n $(2 \leq n \leq 2e5)$.第二行输入n个整数$a_0, \cdots, a_{n-1}$ $(1 \leq a_i \leq 1e6)$.数据保证所有测试数据的n之和不超过$2e5$.

思路I

操作可终止,即有限步内能使得$a[]$中的所有元素都相同.具体地,至多n次操作能使得$a[]$中的所有元素都相同.

[证] 下面$a[]$的下标从1开始.破环成链,先将序列倍增为一个长度为$2n$的序列,即使得$a_{i+n} = a_i$ $(1 \leq i \leq n)$.

设第一次操作后$c_i = \gcd(a_i, a_{i+1})$,则第二次操作后$b_i = \gcd(c_i, c_{i+1}) = \gcd(a_i, a_{i+1}, a_{i+2})$.

数归易证:k次操作后,每个a_i会变为$\gcd(a_i, a_{i+1}, \cdots, a_{i+k})$.

若操作次数为k时,至少存在一个$b_i > \gcd(a_1, \cdots, a_n)$.

问题转化为:求一个最小的k,使得对所有的$i \in [1, n]$,都有$\gcd(a_i, a_{i+1}, \cdots, a_{i+k-1}) = \gcd(a_1, \cdots, a_n)$,则$ans = k - 1$.

显然$k \geq n$时一定有$\gcd(a_i, a_{i+1}, \cdots, a_{i+k-1}) = \gcd(a_1, \cdots, a_n)$,故至多$n$次操作能使得$a[]$中的所有元素都相同.

显然操作次数越多越可能满足要求,故操作次数有二段性,考虑二分.

对二分的操作次数mid,暴力检查每个长度为mid的区间是否满足的时间复杂度是$O(n^2)$,会TLE.

考虑优化,注意到gcd是可重复贡献的,故上述过程可用ST表优化至$O(n \log n \log A)$,其中$A = \max\limits_{1 \leq i \leq n} a_i$.

代码I

```cpp
template<typename T>
struct RMQ1D {  // 下标从1开始
  int n;
  vector<vector<T>> st;  // ST表
  const function<T(T, T)> cmp;  // 比较规则

  RMQ1D(int _n, const vector<T>& a, function<T(T, T)> _cmp) :n(_n), cmp(_cmp) {
    assert(a.size() > n);

    const int LG = log2(n);
    st.assign(LG + 1, vector<T>(n + 1));

    st[0] = a;
    for (int j = 1; j <= LG; j++) {
      for (int i = 1; i <= n - (1 << j) + 1; i++)
        st[j][i] = cmp(st[j - 1][i], st[j - 1][i + (1 << j - 1)]);
    }
  }

  T query(int l, int r) {  // 查询区间[l,r]中的最值
    int k = log2(r - l + 1);
    return cmp(st[k][l], st[k][r - (1 << k) + 1]);
  }
};
```

```cpp

void solve() {
  int n; cin >> n;
  vector<int> a(2 * n + 1);
  int g = 0;
  for (int i = 1; i <= n; i++) {
    cin >> a[i];
    a[i + n] = a[i];
    g = gcd(g, a[i]);
  }

  auto rmq = RMQ1D<int>(2 * n, a, [&](int a, int b) {
    return gcd(a, b);
  });

  auto check = [&](int mid)->bool {
    for (int i = 1; i + mid - 1 <= 2 * n; i++)
      if (rmq.query(i, i + mid - 1) != g) return false;
    return true;
  };

  int l = 1, r = n;
  while (l < r) {
    int mid = l + r >> 1;
    if (check(mid)) r = mid;
    else l = mid + 1;
  }
  cout << l - 1 << endl;  // 注意-1
}

int main() {
  CaseT
    solve();
}
```

思路II

注意到环上相邻元素取gcd的操作等价于将它们的相同素因子的次数取min,则最后所有元素都相等的序列中的元素是原序列的所有公共素因子的次数取min后的乘积,即$\gcd(a_1,\cdots,a_n)$.对每个公共素因子p,设其次数的最小值为$mins$,则操作次数是p的次数$>mins$的区间长度的最大值,故对每个公共素因子,扫一遍序列得到操作次数后取max即可.设$A=\max\limits_{1\le i\le n}a_i$,则上述过程的时间复杂度$O\left(n\dfrac{A}{\ln A}\right)$,会TLE.

考虑优化.

①对≤ 1000的168个素因子,用上述过程暴力求得区间长度的最大值ans.

②对>1000的素因子,注意到每个a_i至多含一个$>\sqrt{a_i}$的素因子,则对>1000的素因子p,每个a_i中的p的次数为0或1.

将每个a_i中的所有≤ 1000的素因子都剔除后,剩下的元素为1或>1000的素因子,

此时破环成链后,在链上求不包含1的区间长度的最大值$maxLength$,再与ans取max即可.

注意若$maxLength\ge n$,则所有$a[]$剔除≤ 1000的素因子后都相等,此时无需与ans取max.

总时间复杂度$O\left(n\dfrac{\sqrt{A}}{\ln\sqrt{A}}\right)=O\left(n\dfrac{\sqrt{A}}{\ln A}\right)$.

代码II

```cpp
const int MAXN = 1005;
namespace LinearSieve {
  int primes[MAXN], cnt;
  bool state[MAXN];

  void init() {
    for (int i = 2; i < MAXN; i++) {
      if (!state[i]) primes[cnt++] = i;
      for (int j = 0; (ll)primes[j] * i < MAXN; j++) {
        state[primes[j] * i] = true;
        if (i % primes[j] == 0) break;
      }
    }
  }
}
using namespace LinearSieve;

void solve() {
  int n; cin >> n;
  vector<int> a(2 * n + 1);
  for (int i = 1; i <= n; i++) cin >> a[i];

  int ans = 0;
  // 求得每个元素包含的1000以内的素因子的次数,并剔除这些素因子
  for (int i = 0; i < cnt; i++) {  // 枚举素因子
    vector<int> s(2 * n + 1);  // s[j]表示a[j]包含素因子primes[i]的次数
    int mins = INF;  // 公共素因子的最小次数
    for (int j = 1; j <= n; j++) {
      while (a[j] % primes[i] == 0) {
        a[j] /= primes[i];
        s[j]++;
      }
      mins = min(mins, s[j]);
    }

    for (int j = 1; j <= n; j++) s[j + n] = s[j];

    int length = 0;  // 当前区间长度
    int maxLength = 0;  // 次数>mins的区间长度的最大值
    for (int j = 1; j <= 2 * n; j++, maxLength = max(maxLength, length)) {
      if (s[j] > mins) length++;
      else length = 0;
    }
    ans = max(ans, maxLength);
  }

  for (int i = 1; i <= n; i++) a[i + n] = a[i];

  // 对>1000的素因子,求一遍答案
  int length = 0;  // 当前区间长度
  int maxLength = 0;  // 不包含1的区间长度的最大值
  for (int i = 1; i <= 2 * n; i++, maxLength = max(maxLength, length)) {
    if (a[i] == 1) length = 0;
    else if (a[i] != a[i - 1]) length = 1;
    else length++;
```

```
  }
  if (maxLength < n)  // a[]剔除所有≤1000的素因子后不都相等才需更新ans
    ans = max(ans, maxLength);

  cout << ans << endl;
}

int main() {
  init();
  CaseT
    solve();
}
```

思路III

设$g = \gcd\limits_{1\le i\le n} a_i$.对每个$a_i$,将$\frac{a_i}{g}$后素因数分解,对每个素因子,双指针$l$、$r$分别从下标$i$的前一位、下标$i$开始向左、右移动,找到一段有该公共素因子的区间$[l, r]$,答案与该区间的长度取max.统计完素因子$p$的答案后将其从$a[l\cdots r]$中剔除即可.为方便查询每个$a_i$是否包含素因子$p$,可用set存每个$a_i$素因数分解的结果,时间复杂度$O(n\cdot maxcnt\cdot \log maxcnt)$,其中$maxcnt$为$a_i$包含的不同的素因子的个数的最大值,因$a_i \le 1e6$,则$maxcnt = 8$.

代码III

```
const int MAXN = 1e6 + 5;
namespace LinearSieve {
  int primes[MAXN], cnt;
  int minp[MAXN];  // 最小素因子

  void init() {
    for (int i = 2; i < MAXN; i++) {
      if (!minp[i]) primes[cnt++] = i, minp[i] = i;
      for (int j = 0; (ll)primes[j] * i < MAXN; j++) {
        minp[primes[j] * i] = primes[j];
        if (i % primes[j] == 0) break;
      }
    }
  }
}
using namespace LinearSieve;

void solve() {
  int n; cin >> n;
  vector<int> a(n);
  int g = 0;
  for (auto& ai : a) {
    cin >> ai;
    g = gcd(g, ai);
  }

  vector<set<int>> factors(n);  // factors[i]存a[i]的不同的素因子
  for (int i = 0; i < n; i++) {
    int tmp = a[i] / g;
    while (tmp > 1) {
      factors[i].insert(minp[tmp]);
      tmp /= minp[tmp];
    }
```

```cpp
  }

  int ans = 0;
  for (int i = 0; i < n; factors[i++].clear()) {  // 注意清空
    for (auto p : factors[i]) {
      int l = (i + n - 1) % n, r = i;  // 指针l从下标i的上一位开始
      int cnt = 0;  // 包含公共素因子p的元素个数
      while (factors[l].count(p)) {  // l向左走
        factors[l].erase(p);  // 注意剔除该素因子
        l = (l - 1 + n) % n;
        cnt++;
      }
      while (factors[r].count(p)) {  // r向右走
        if (r != i) factors[r].erase(p);  // 注意剔除该素因子
        r = (r + 1) % n;
        cnt++;
      }

      ans = max(ans, cnt);
    }
  }
  cout << ans << endl;
}

int main() {
  init();
  CaseT
    solve();
}
```

28.1.4 Rorororobot

原题指路: https://codeforces.com/problemset/problem/1709/D

题意 (2 s)

在$n \times m$ $(1 \le n \le 1e9, 1 \le m \le 2e5)$的网格中, 第$i$ $(1 \le i \le m)$列的底部有a_i $(0 \le a_i \le m)$个格子为障碍物, 其余$(n - a_i)$个格子无障碍物. 一个机器人可在网格中上、下、左、右移动, 但它每次会朝一个方向移动k个格子. 若机器人移动到有障碍的格子或网格之外, 它会爆炸. 现有q $(1 \le q \le 2e5)$个询问, 每个询问给定机器人的起点坐标(x_s, y_s) $(a[y_s] < x_s \le n, 1 \le y_s \le m)$和终点坐标$(x_f, y_f)$ $(a[y_f] < x_f \le n, 1 \le y_f \le m)$和整数$k$ $(1 \le k \le 1e9)$, 问能否从起点出发, 经若干步移动恰停在终点.

思路

显然若有解, 则起点到终点的路径可逆, 不妨设$y_s \le y_f$, 即机器人不往左走.

有解的充要条件是: $k \mid |x_s - x_f|, k \mid |y_s - y_f|$, 且$\min\limits_{y_s \le i \le y_f} a_i < lim$, 其中$lim$为不超过$(n - x_s)$的最大的$k$的倍数, 即$lim = \left\lfloor \dfrac{n - x_s}{k} \right\rfloor \cdot k$.

因$m \le 2e5$, 可用ST表查询区间min. 时间复杂度$O(m \log m + q \log m) = O((m + q) \log m)$.

代码

```cpp
template<typename T>
struct RMQ1D {  // 下标从1开始
    int n;
    vector<vector<T>> st;  // ST表
    const function<T(T, T)> cmp;  // 比较规则

    RMQ1D(int _n, const vector<T>& a, function<T(T, T)> _cmp) :n(_n), cmp(_cmp) {
        assert(a.size() > n);

        const int LG = log2(n);
        st.assign(LG + 1, vector<T>(n + 1));

        st[0] = a;
        for (int j = 1; j <= LG; j++) {
        for (int i = 1; i <= n - (1 << j) + 1; i++)
            st[j][i] = cmp(st[j - 1][i], st[j - 1][i + (1 << j - 1)]);
        }
    }

    T query(int l, int r) {  // 查询区间[l,r]中的最值
        int k = log2(r - l + 1);
        return cmp(st[k][l], st[k][r - (1 << k) + 1]);
    }
};

void solve() {
    int n, m; cin >> n >> m;
    vector<int> a(m + 1);
    for (int i = 1; i <= m; i++) cin >> a[i];

    auto rmq = RMQ1D<int>(m, a, [&](int a, int b) {
        return max(a, b);
    });

    CaseT {
        int xs, ys, xf, yf, k; cin >> xs >> ys >> xf >> yf >> k;
        if (ys > yf) swap(xs, xf), swap(ys, yf);

        if (abs(xs - xf) % k || abs(ys - yf) % k) cout << "NO" << endl;
        else {
            int lim = xs + (n - xs) / k * k;
            cout << (rmq.query(ys, yf) < lim ? "YES" : "NO") << endl;
        }
    }
}

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

28.1.5 Not Equal on a Segment

原题指路: https://codeforces.com/problemset/problem/622/C

题意

给定一个长度为$n\ \ (1 \le n \le 2e5)$的序列$a = [a_1, \cdots, a_n]\ \ (1 \le a_i \le 1e6)$. 现有$m\ \ (1 \le m \le 2e5)$个询问, 每个询问输入三个整数$l, r, x\ \ (1 \le l \le r \le n, 1 \le x \le 1e6)$, 求下标$pos \in [l, r]\ s.t.\ a_{pos} \ne x$. 若有解, 输出任一解; 否则输出$-1$.

思路I

$last[i]$表示前缀$a[1 \cdots i]$中最靠右的与$a[i]$不同的元素的下标, 则递推公式$last[i] = \begin{cases} i-1, a[i] \ne a[i-1] \\ last[i-1], a[i] = a[i-1] \end{cases}$.

对每个询问, 有如下三种情况:

①若$a[r] \ne x$, 则r是一个解.

②若$last[r] \ge l$, 则$last[r]$是一个解.

③否则无解.

时间复杂度$O(n+m)$.

代码I

```cpp
void solve() {
    int n, m; cin >> n >> m;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i];

    vector<int> last(n + 1);  // last[i]表示前缀a[1...i]中最靠右的与a[i]不同的元素的下标
    for (int i = 1; i <= n; i++)
        last[i] = a[i] != a[i - 1] ? i - 1 : last[i - 1];

    while (m--) {
        int l, r, x; cin >> l >> r >> x;

        if (a[r] != x) cout << r << endl;
        else if (last[r] >= l) cout << last[r] << endl;
        else cout << -1 << endl;
    }
}

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

思路II

用ST表维护区间最小值和最大值及其出现的位置.

对每个询问区间$[l, r]$, 设其中的最小值和最大值分别为$minv$和$maxv$, 有如下两种情况:

①若$minv = maxv = x$, 则该区间内的数都为x, 进而无解.

②若$minv \neq x$或$maxv \neq x$, 则$minv$或$maxv$的出现位置即一个解.

时间复杂度$O(n \log n + m)$.

代码II

```cpp
template<typename T>
struct RMQ1D {  // 下标从1开始
    int n;
    vector<T> a;
    vector<vector<T>> st;  // ST表
    const function<T(T, T)> cmp;  // 比较规则

    RMQ1D(int _n, const vector<T>& _a, function<T(T, T)> _cmp) :
        n(_n), a(_a), cmp(_cmp) {
        assert(a.size() > n);

        const int LG = log2(n);
        st.assign(LG + 1, vector<T>(n + 1));

        for (int i = 1; i <= n; i++) st[0][i] = i;

        // st[0] = a;
        for (int j = 1; j <= LG; j++) {
        for (int i = 1; i <= n - (1 << j) + 1; i++)
            st[j][i] = cmp(st[j - 1][i], st[j - 1][i + (1 << j - 1)]);
        }
    }

    T query(int l, int r) {  // 查询区间[l,r]中的最值
        int k = log2(r - l + 1);
        return cmp(st[k][l], st[k][r - (1 << k) + 1]);
    }
};

void solve() {
    int n, m; cin >> n >> m;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i];

    auto rmqMin = RMQ1D<int>(n, a, [&](int i, int j) {
        return a[i] < a[j] ? i : j;
    });
    auto rmqMax = RMQ1D<int>(n, a, [&](int i, int j) {
        return a[i] > a[j] ? i : j;
    });

    while (m--) {
        int l, r, x; cin >> l >> r >> x;

        int minidx = rmqMin.query(l, r), maxidx = rmqMax.query(l, r);
        if (a[minidx] == a[maxidx] && a[minidx] == x) cout << -1 << endl;
        else cout << (a[minidx] != x ? minidx : maxidx) << endl;
    }
}

int main() {
    solve();
```

```
}
```

思路III

同思路II, 但改为用线段树维护区间最小值和最大值及其出现的位置.

时间复杂度$O(n \log n + m)$.

代码III

```
struct SegmentTree {
    int n;
    vector<int> a;
    struct Node {
        int minval, maxval;
        int minidx, maxidx;
    };
    vector<Node> SegT;

    SegmentTree(int _n, const vector<int>& _a) :n(_n), a(_a) {
        SegT.resize(n + 1 << 2);
        build(1, 1 ,n);
    };

    friend Node operator+(const Node ls, const Node rs) {
        Node u;
        if (ls.minval < rs.minval) u.minval = ls.minval, u.minidx = ls.minidx;
        else u.minval = rs.minval, u.minidx = rs.minidx;

        if (ls.maxval > rs.maxval) u.maxval = ls.maxval, u.maxidx = ls.maxidx;
        else u.maxval = rs.maxval, u.maxidx = rs.maxidx;
        return u;
    }

    void pushUp(int u) {
        SegT[u] = SegT[u << 1] + SegT[u << 1 | 1];
    }

    void build(int u, int l, int r) {
        if (l == r) {
            SegT[u].minval = SegT[u].maxval = a[l];
            SegT[u].minidx = SegT[u].maxidx = l;
            return;
        }

        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushUp(u);
    }

    Node query(int u, int l, int r, int L, int R) {  // 查询区间[L, R]中的minidx和maxidx
        if (L <= l && r <= R) return SegT[u];

        int mid = l + r >> 1;
        if (R <= mid) return query(u << 1, l, mid, L, R);
        else if (L > mid) return query(u << 1 | 1, mid + 1, r, L, R);
        else return query(u << 1, l, mid, L, R) + query(u << 1 | 1, mid + 1, r, L, R);
```

```
    }
};

void solve() {
    int n, m; cin >> n >> m;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i];

    SegmentTree st(n, a);

    while (m--) {
        int l, r, x; cin >> l >> r >> x;

        auto [minval, maxval, minidx, maxidx] = st.query(1, 1, n, l, r);
        if (minval == maxval && minval == x) cout << -1 << endl;
        else cout << (minval !=x ? minidx : maxidx) << endl;
    }
}

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

28.2 二维RMQ

28.2.1 Valiant's New Map

原题指路:https://codeforces.com/contest/1731/problem/D

题意

有t $(1 \leq t \leq 1000)$组测试数据.每组测试数据给定一个$n \times m$的整型矩阵$A_{n\times m} = (a_{ij})_{n\times m}$.求最大的$l$ $s.t.$ 矩阵中存在一个$l \times l$的正方形,使得其中所有元素$\geq l$.数据保证所有测试数据的$nm \leq 1e6$.

思路

二分l后,将原矩阵按照元素是否$\geq l$转化为$0/1$矩阵,用二维前缀和求全为1的最大正方形来check,或用二维RMQ检查每个正方形内的最小值是否为1.

事实上二维前缀和check比用二维RMQcheck要快.

代码I:二维前缀和check

```
void solve() {
  int n, m; cin >> n >> m;
  vector<vector<int>> a(n + 1, vector<int>(m + 1));
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= m; j++) cin >> a[i][j];

  auto check = [&](int mid)->bool {
    vector<vector<int>> b(n + 1, vector<int>(m + 1));
    for (int i = 1; i <= n; i++)
      for (int j = 1; j <= m; j++) b[i][j] = a[i][j] >= mid;

    vector<vector<int>> pre(n + 1, vector<int>(m + 1));  // 二维前缀和
```

```
    int res = 0;
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= m; j++) {
        if (b[i][j])
          pre[i][j] = min({ pre[i][j - 1],pre[i - 1][j],pre[i - 1][j - 1] }) + 1;
        res = max(res, pre[i][j]);
      }
    }
    return res >= mid;
  };

  int l = 1, r = min(n, m);
  while (l < r) {
    int mid = l + r + 1 >> 1;
    if (check(mid)) l = mid;
    else r = mid - 1;
  }
  cout << l << endl;
}

int main() {
  CaseT
  solve();
}
```

代码II:二维RMQcheck

```
template<typename T>
struct RMQ2D {
  int n, m;
  vector<vector<vector<T>>> st;  // ST表
  const function<T(T, T, T, T)> cmp;  // 比较规则

  RMQ2D(int _n, int _m, const vector<vector<T>>& a, function<T(T, T, T, T)> _cmp)
    :n(_n), m(_m), cmp(_cmp) {
    assert(a.size() > n && a[0].size() > m);

    const int LG = log2(min(n, m));
    st.assign(LG + 1, vector(n + 1, vector<T>(m + 1)));

    st[0] = a;
    for (int k = 1; k <= LG; k++) {
      for (int i = 1; i <= n - (1 << k) + 1; i++) {
        for (int j = 1; j <= m - (1 << k) + 1; j++) {
          st[k][i][j] = cmp(st[k - 1][i][j], st[k - 1][i + (1 << k - 1)][j],
            st[k - 1][i][j + (1 << k - 1)], st[k - 1][i + (1 << k - 1)][j + (1 << k -
1)]);
        }
      }
    }
  }

  T query(int x1, int y1, int x2, int y2) {  // 查询以(x1,y1)为左上角、以(x2,y2)为右下角的正方形
中的最值
    assert(x2 - x1 == y2 - y1);

```

```
    int k = log2(x2 - x1 + 1);
    return cmp(st[k][x1][y1], st[k][x2 - (1 << k) + 1][y1],
      st[k][x1][y2 - (1 << k) + 1], st[k][x2 - (1 << k) + 1][y2 - (1 << k) + 1]);
  }
};

void solve() {
  int n, m; cin >> n >> m;
  vector<vector<int>> a(n + 1, vector<int>(m + 1));
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= m; j++) cin >> a[i][j];

  auto check = [&](int mid)->bool {
    auto b = a;
    for (int i = 1; i <= n; i++)
      for (int j = 1; j <= m; j++) b[i][j] = a[i][j] >= mid;

    auto rmq = RMQ2D<int>(n, m, b, [&](int a, int b, int c, int d) {
      return min({ a,b,c,d });
      });

    bool ok = false;
    for (int i = 1; i + mid - 1 <= n; i++) {
      for (int j = 1; j + mid - 1 <= m; j++)
        ok |= rmq.query(i, j, i + mid - 1, j + mid - 1);
    }
    return ok;
  };

  int l = 1, r = min(n, m);
  while (l < r) {
    int mid = l + r + 1 >> 1;
    if (check(mid)) l = mid;
    else r = mid - 1;
  }
  cout << l << endl;
}

int main() {
  CaseT
  solve();
}
```

28.2.2 理想的正方形

原题指路:https://www.luogu.com.cn/problem/P2216

题意

给定一个$a \times b$ $(2 \leq a, b \leq 1000)$的元素范围为$[1, 1e9]$整型矩阵,求一个$n \times n$ $(1 \leq n \leq \min\{a, b, 100\})$的正方形,使得其中元素的最大值余最小值之差最小.

思路

数据范围较小,用二维RMQ暴力统计即可.

代码

```
template<typename T>
struct RMQ2D {
  int n, m;
  vector<vector<vector<T>>> st;  // ST表
  const function<T(T, T, T, T)> cmp;  // 比较规则

  RMQ2D(int _n, int _m, const vector<vector<T>>& a, function<T(T, T, T, T)> _cmp)
    :n(_n), m(_m), cmp(_cmp) {
    assert(a.size() > n && a[0].size() > m);

    const int LG = log2(min(n, m));
    st.assign(LG + 1, vector(n + 1, vector<T>(m + 1)));

    st[0] = a;
    for (int k = 1; k <= LG; k++) {
      for (int i = 1; i <= n - (1 << k) + 1; i++) {
        for (int j = 1; j <= m - (1 << k) + 1; j++) {
          st[k][i][j] = cmp(st[k - 1][i][j], st[k - 1][i + (1 << k - 1)][j],
            st[k - 1][i][j + (1 << k - 1)], st[k - 1][i + (1 << k - 1)][j + (1 << k -
1)]);
        }
      }
    }
  }

  T query(int x1, int y1, int x2, int y2) {  // 查询以(x1,y1)为左上角、以(x2,y2)为右下角的正方形
中的最值
    assert(x2 - x1 == y2 - y1);

    int k = log2(x2 - x1 + 1);
    return cmp(st[k][x1][y1], st[k][x2 - (1 << k) + 1][y1],
      st[k][x1][y2 - (1 << k) + 1], st[k][x2 - (1 << k) + 1][y2 - (1 << k) + 1]);
  }

  T query(int x, int y, int siz) {  // 查询以(x,y)为左上角、边长为siz的正方形中的最值
    return query(x, y, x + siz - 1, y + siz - 1);
  }
};

void solve() {
  int a, b, n; cin >> a >> b >> n;
  vector<vector<int>> matrix(a + 1, vector<int>(b + 1));
  for (int i = 1; i <= a; i++)
    for (int j = 1; j <= b; j++) cin >> matrix[i][j];

  auto rmqmax = RMQ2D<int>(a, b, matrix, [&](int a, int b, int c, int d) {
    return max({ a,b,c,d });
    });
  auto rmqmin = RMQ2D<int>(a, b, matrix, [&](int a, int b, int c, int d) {
    return min({ a,b,c,d });
    });
```

```
50
51    int ans = INF;
52    for (int i = 1; i + n - 1 <= a; i++) {
53      for (int j = 1; j + n - 1 <= b; j++)
54        ans = min(ans, rmqmax.query(i, j, n) - rmqmin.query(i, j, n));
55    }
56    cout << ans << endl;
57  }
58
59  int main() {
60    solve();
61  }
```