回滚莫队

有些题目在区间转移时，可能会出现增加或者删除无法实现的问题。在只有增加不可实现或者只有删除不可实现的时候，就可以使用回滚莫队在 $O(n\sqrt{n})$ 的时间内解决问题。回滚莫队的核心思想就是既然我只能实现一个操作，那么我就只使用一个操作，剩下的交给回滚解决。

回滚莫队分为只使用增加操作的回滚莫队和只使用删除操作的回滚莫队。以下仅介绍只使用增加操作的回滚莫队，只使用删除操作的回滚莫队和只使用增加操作的回滚莫队只在算法实现上有一点区别，故不再赘述。

例题 JOISC 2014 Day1 历史研究 [https://loj.ac/problem/2874]

给你一个长度为 n 的数组 A 和 m 个询问 $(1 \leq n, m \leq 10^5)$，每次询问一个区间 $[L, R]$ 内重要度最大的数字，要求 **输出其重要度** 。一个数字 i 重要度的定义为 i 乘上 i 在区间内出现的次数。

在这个问题中，在增加的过程中更新答案是很好实现的，但是在删除的过程中更新答案是不好实现的。因为如果增加会影响答案，那么新答案必定是刚刚增加的数字的重要度，而如果删除过后区间重要度最大的数字改变，我们很难确定新的重要度最大的数字是哪一个。所以，普通的莫队很难解决这个问题。

具体算法

- 对原序列进行分块，对询问按以左端点所属块编号升序为第一关键字，右端点升序为第二关键字的方式排序
- 按顺序处理询问
  - 如果询问左端点所属块 B 和上一个询问左端点所属块的不同，那么将莫队区间的左端点初始化为 B 的右端点加 1, 将莫队区间的右端点初始化为

B 的右端点

- 如果询问的左右端点所属的块相同，那么直接扫描区间回答询问
- 如果询问的左右端点所属的块不同
  - 如果询问的右端点大于莫队区间的右端点，那么不断扩展右端点直至莫队区间的右端点等于询问的右端点
  - 不断扩展莫队区间的左端点直至莫队区间的左端点等于询问的左端点
  - 回答询问
  - 撤销莫队区间左端点的改动，使莫队区间的左端点回滚到 B 的右端点加 1

复杂度证明

假设左右端点同属于一个块的询问个个数为 C_1 ，左右端点不属于同一个块的询问的个数为 C_2 。

回答一个左右端点同属于一个块的询问的时间复杂度为 $O(\sqrt{n})$ ，回答所有左右端点同属于一个块的询问的时间复杂度为 $O(C_1\sqrt{n})$ 。

对于左右端点不属于同一个块的询问，将其按左端点所属块分类。对于一类询问，假设属于这一类询问的个数为 c_i 。在回答这一类询问的时候莫队区间右端点至多扩展 n 次；回答这一类问题中的一个的时候，左端点扩展和回滚的复杂度为 $O(\sqrt{n})$ 。由此，回答一类问题的复杂度为 $O(n+c_i\sqrt{n})$ 。总共有 $\sqrt{n}$ 类询问，所以回答左右端点不属于同一个块的询问的时间复杂度为 $O(C_2\sqrt{n}+n\sqrt{n})$ 。

综上，这个算法的复杂度 $T(n)=O(C_2\sqrt{n}+n\sqrt{n})+O(C_1\sqrt{n})=O(n\sqrt{n}+m\sqrt{n})$ 。

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 5;
int n, q;
int x[N], t[N], m;

```

```cpp
struct Query {
  int l, r, id;
} Q[N];
int pos[N], L[N], R[N], sz, tot;
int cnt[N], __cnt[N];
ll ans[N];

inline bool cmp(const Query& A, const Query& B) {
  if (pos[A.l] == pos[B.l]) return A.r < B.r;
  return pos[A.l] < pos[B.l];
}

void build() {
  sz = sqrt(n);
  tot = n / sz;
  for (int i = 1; i <= tot; i++) {
    L[i] = (i - 1) * sz + 1;
    R[i] = i * sz;
  }
  if (R[tot] < n) {
    ++tot;
    L[tot] = R[tot - 1] + 1;
    R[tot] = n;
  }
}

inline void Add(int v, ll& Ans) {
  ++cnt[v];
  Ans = max(Ans, 1LL * cnt[v] * t[v]);
}

inline void Del(int v) { --cnt[v]; }

int main() {
  scanf("%d %d", &n, &q);
  for (int i = 1; i <= n; i++) scanf("%d", &x[i]), t[++m]
= x[i];
  for (int i = 1; i <= q; i++) scanf("%d %d", &Q[i].l,
&Q[i].r), Q[i].id = i;

  build();

  // 对询问进行排序
  for (int i = 1; i <= tot; i++)
    for (int j = L[i]; j <= R[i]; j++) pos[j] = i;
  sort(Q + 1, Q + 1 + q, cmp);

  // 离散化
  sort(t + 1, t + 1 + m);
```

```
  m = unique(t + 1, t + 1 + m) - (t + 1);
  for (int i = 1; i <= n; i++) x[i] = lower_bound(t + 1, t
+ 1 + m, x[i]) - t;

  int l = 1, r = 0, last_block = 0, __l;
  ll Ans = 0, tmp;
  for (int i = 1; i <= q; i++) {
    // 询问的左右端点同属于一个块则暴力扫描回答
    if (pos[Q[i].l] == pos[Q[i].r]) {
      for (int j = Q[i].l; j <= Q[i].r; j++)
++__cnt[x[j]];
      for (int j = Q[i].l; j <= Q[i].r; j++)
        ans[Q[i].id] = max(ans[Q[i].id], 1LL * t[x[j]] *
__cnt[x[j]]);
      for (int j = Q[i].l; j <= Q[i].r; j++) --
__cnt[x[j]];
      continue;
    }

    // 访问到了新的块则重新初始化莫队区间
    if (pos[Q[i].l] != last_block) {
      while (r > R[pos[Q[i].l]]) Del(x[r]), --r;
      while (l < R[pos[Q[i].l]] + 1) Del(x[l]), ++l;
      Ans = 0;
      last_block = pos[Q[i].l];
    }

    // 扩展右端点
    while (r < Q[i].r) ++r, Add(x[r], Ans);
    __l = l;
    tmp = Ans;

    // 扩展左端点
    while (__l > Q[i].l) --__l, Add(x[__l], tmp);
    ans[Q[i].id] = tmp;

    // 回滚
    while (__l < l) Del(x[__l]), ++__l;
  }
  for (int i = 1; i <= q; i++) printf("%lld\n", ans[i]);
  return 0;
}
```

参考资料