「集思广益一波」

CleanBlue

2022 年 11 月 18 日

目录

1 数据结构

1.1 单调队列/滑动窗口

有一个长为 n 的序列 a，以及一个大小为 k 的窗口。现在这个从左边开始向右滑动，每次滑动一个单位，求出每次滑动后窗口中的最大值和最小值。

```cpp
int n, k;
std::cin >> n >> k;

std::vector<int> a(n);
for (int &i : a)
    std::cin >> i;

std::deque<int> q;
for (int i = 0; i < n; ++i) {  //输出最小值
    while (!q.empty() && a[q.back()] >= a[i])
        q.pop_back();
    q.emplace_back(i);
    if (i >= k - 1) {
        while (!q.empty() && q.front() <= i - k)
            q.pop_front();
        std::cout << a[q.front()] << ' ';
    }
}
std::cout << '\n';

while (!q.empty())
    q.pop_back();

for (int i = 0; i < n; ++i) {  //输出最大值
    while (!q.empty() && a[q.back()] <= a[i])
        q.pop_back();
    q.emplace_back(i);
    if (i >= k - 1) {
        while (!q.empty() && q.front() <= i - k)
            q.pop_front();
        std::cout << a[q.front()] << ' ';
    }
}
std::cout << '\n';
```

1.2 单调栈

输出右边第一个比 a_i 大的数的下标。

```cpp
int n;
std::cin >> n;

std::vector<int> a(n), ans(n);
for (int &i : a)
    std::cin >> i;

std::stack<int> s;
for (int i = n - 1; i >= 0; --i) {
    while (!s.empty() && a[s.top()] <= a[i])
        s.pop();
    if (s.empty())
        ans[i] = -1;
    else
        ans[i] = s.top();
    s.emplace(i);
}
```

1.3 并查集

```cpp
struct dsu {
    // 元素范围 [0, n)
    int n;
    std::vector<int> f, sz;

    dsu(int n) : n(n), f(n), sz(n, 1) { std::iota(f.begin(), f.end(), 0); }
    int father(int u) {
        if (u >= n) return -1;
        while (u != f[u]) u = f[u] = f[f[u]];
        return u;
    }
    bool same(int u, int v) {
        int a = father(u), b = father(v);
        return a == b && a != -1;
    }
    void merge(int u, int v) {
        int a = father(u), b = father(v);
        if (sz[a] > sz[b])
            f[b] = a, sz[a] += sz[b];
        else
            f[a] = b, sz[b] += sz[a];
    }
    int size(int u) { return sz[father(u)]; }
};
```

2 基本算法

2.1 快速幂/整数类/取模

```cpp
constexpr int P = 1000000007;
int norm(int x) {
    if (x < 0) x += P;
    if (x >= P) x -= P;
    return x;
}
template <class T> constexpr T power(T a, int64_t b) {
    T res = 1;
    for (; b; b /= 2, a *= a)
    if (b % 2) res *= a;
    return res;
}

struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    Z(int64_t x) : x(norm(x % P)) {}
    int val() const { return x; }
    Z operator-() const { return Z(norm(P - x)); }
    Z inv() const {
        assert(x != 0);
        return power(*this, P - 2);
    }
    Z &operator*=(const Z &rhs) {
        x = int64_t(x) * rhs.x % P;
        return *this;
    }
    Z &operator+=(const Z &rhs) {
        x = norm(x + rhs.x);
```

```
        return *this;
    }
    Z &operator-=(const Z &rhs) {
        x = norm(x - rhs.x);
        return *this;
    }
    Z &operator/=(const Z &rhs) { return *this
        *= rhs.inv(); }
    friend Z operator*(const Z &lhs, const Z &
        rhs) {
        Z res = lhs;
        res *= rhs;
        return res;
    }
    friend Z operator+(const Z &lhs, const Z &
        rhs) {
        Z res = lhs;
        res += rhs;
        return res;
    }
    friend Z operator-(const Z &lhs, const Z &
        rhs) {
        Z res = lhs;
        res -= rhs;
        return res;
    }
    friend Z operator/(const Z &lhs, const Z &
        rhs) {
        Z res = lhs;
        res /= rhs;
        return res;
    }
    friend std::istream &operator>>(std::istream
         &is, Z &a) {
        int64_t v;
        is >> v;
        a = Z(v);
        return is;
    }
    friend std::ostream &operator<<(std::ostream
         &os, const Z &a) { return os << a.val();
         }
};
```

2.2 一二三维差分/前缀和

一维：区间 $[l,r]$ 每个元素加上 x：

```
d[l] += x;
d[r + 1] -= x;
```

二维：$(x1,y1),(x2,y2)$ 每个元素加上 x：

```
d[x1][y1] += x;
d[x1][y2 + 1] -= x;
d[x2 + 1][y1] -= x;
d[x2 + 1][y2 + 1] += x;
```

三维：$(x1,y1,z1),(x2,y2,z2)$ 每个元素加上 x：

```
d[x1][y1][z1] += x;
d[x2 + 1][y1][z1] -= x;
d[x1][y1][z2 + 1] -= x;
d[x2 + 1][y1][z2 + 1] += x;
d[x1][y2 + 1][z1] -= x;
d[x2 + 1][y2 + 1][z1] += x;
d[x1][y2 + 1][z2 + 1] += x;
d[x2 + 1][y2 + 1][z2 + 1] -=x;
```

2.3 二分

```
int binary_search_1(const std::vector<int> &a,
    int x) {
    // a 单调递增
    // 在 [0, n) 中查找 x 或 x 的后继的位置
    int n = a.size(), l = 0, r = n;
    while (l < r) {
        int mid = (l + r) / 2;
        if (a[mid] >= x)
            r = mid;
        else
            l = mid + 1;
    }
    return l;
}

int binary_search_2(const std::vector<int> &a,
    int x) {
    // a 单调递增
    //在 [0, n) 中查找 x 或 x 的前驱的位置
    int n = a.size(), l = 0, r = n;
    while (l < r) {
        int mid = (l + r + 1) / 2;
        if (a[mid] <= x)
            l = mid;
        else
            r = mid - 1;
    }
    return l;
}

double float_binary_search(/*...*/) {
    constexpr double eps = 1e-7;
    double l = 0, r = 1e18;
    while (r - l > eps) {
        double mid = (l + r) / 2;
        if (/*...*/)
            l = mid;
        else
            r = mid;
    }
    return l;
}
```

2.4 三分

如何取 mid1 和 mid2？有以下两种基本方法。

(1) 三等分：mid1 和 mid2 为 $[l,r]$ 的三等分点，区间每次可以缩小 1 / 3。

(2) 近似三等分：计算 $[l,r]$ 的中间点 mid $=(l+r)/2$，然后让 mid1 和 mid2 非常接近 mid，如 mid1 = mid - eps, mid2 = mid + eps。区间每次可以缩小接近一半。

近似三等分比三等分要稍微快一点。不过，在有些情况下 eps 过小可能导致 mid1 和 mid2 算出的结果相等，从而判断错方向。

三等分：

```
while (r - l > eps) {
```

```
    double k = (r - l) / 3;
    double mid1 = l + k, mid2 = r - k;
    if (/*...*/)
        r = mid2;
    else
        l = mid1;
}
```

近似三等分：

```
while (r - l > eps) {
    double mid1 = (l + r) / 2;
    double mid2 = mid1 + eps;
    if (/*...*/)
        r = mid2;
    else
        l = mid1;
}
```

整数三分：

```
while (r - l > 2) {
    int mid1 = l + (r - l) / 3;
    int mid2 = r - (r - l) / 3;
    if (/*...*/)
        r = mid2;
    else
        l = mid1;
}
```

2.5 ST/离线区间最值

定义 $dp[i][k]$ 表示左断点为 i，区间长度为 2^k 的区间最值。递推关系为:

$$dp[i][k] = merge\{dp[i][k-1], dp[i+(1<<(k-1))][k]\}$$

$O(nlogn)$ 预处理、$O(1)$ 查询。

```
int N, M;
std::cin >> N >> M;

std::vector<int> a(N);
for (int &i : a) std::cin >> i;

std::vector<int> LOG2(N + 1);
LOG2[0] = -1;
for (int i = 1; i <= N; ++i)
    LOG2[i] = LOG2[i / 2] + 1;

int dp[N][21];
for (int i = 0; i < N; ++i)
    dp[i][0] = a[i];

int p = LOG2[N];
// 可倍增区间的最大次方: 2^p <= N

for (int k = 1; k <= p; ++k)
    for (int i = 0; i + (1 << k) - 1 < N; ++i)
        dp[i][k] = std::max(dp[i][k - 1], dp[i +
            (1 << (k - 1))][k - 1]);

while (M--) {
    int l, r;
    //求 [l, r] 中最大值
    std::cin >> l >> r;
    --l;

    int len = r - l;
    int k = LOG2[len];

    std::cout << std::max(dp[l][k], dp[r - (1 <<
        k)][k]) << '\n';
}
```

ST 类：

```
struct ST {
    size_t n;
    std::vector<std::vector<int>> info;

    int merge(const int &lhs, const int &rhs) {
        // TODO: 修改merge
        return std::max(lhs, rhs);
    }

    ST(const std::vector<int> &init) : n(init.
        size()), info(init.size() + 1, std::
        vector<int>(std::__lg(n) + 1, INT_MIN)) {
          //TODO : 修改初值
        for (size_t x = 0; x <= std::__lg(n); ++
            x)
        for (int i = 0; i + (1ull << x) <= n; ++
            i)
        if (x == 0)
        info[i][x] = init[i];
        else
        info[i][x] = merge(info[i][x - 1], info[
            i + (1 << (x - 1))][x - 1]);
    }

    int query(int l, int r) {  //[l, r)
        int len = r - l;
        assert(len > 0);
        return merge(info[l][std::__lg(len)],
            info[r - (1 << std::__lg(len))][std::
            __lg(len)]);
    }
};
```

2.6 归并排序/逆序对

```
int n;
std::cin >> n;

std::vector<int> a(n), b(n);
for (int &i : a) std::cin >> i;

int64_t ans = 0;
std::function<void(int, int)> merge_sort = [&](
    int l, int r) {
    //排序区间 [l, r)
    if (r - l <= 1) return;
    int mid = (l + r) / 2;
    merge_sort(l, mid);
    merge_sort(mid, r);
    for (int i = l, j = mid, k = l; k < r; ++k)
        {
        if (j == r || (i < mid && a[i] <= a[j]))
            b[k] = a[i++];
```

```
        else {
            ans += mid - i;
            b[k] = a[j++];
        }
    }
    for (int i = l; i < r; ++i) a[i] = b[i];
};

merge_sort(0, n);
std::cout << ans << '\n';
```