

「集思广益一波」

CleanBlue

2022 年 11 月 18 日

目录

1	数据结构	1
1.1	单调队列/滑动窗口	1
1.2	单调栈	1
1.3	并查集	1
2	基本算法	1
2.1	快速幂/整数类/取模	1
2.2	一二三维差分/前缀和	2
2.3	二分	2
2.4	三分	2
2.5	ST/离线区间最值	3
2.6	归并排序/逆序对	3

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。

```

1 int n, k;
2 std::cin >> n >> k;
3
4 std::vector<int> a(n);
5 for (int &i : a)
6     std::cin >> i;
7
8 std::deque<int> q;
9 for (int i = 0; i < n; ++i) { //输出最小值
10     while (!q.empty() && a[q.back()] >= a[i])
11         q.pop_back();
12     q.emplace_back(i);
13     if (i >= k - 1) {
14         while (!q.empty() && q.front() <= i - k)
15             q.pop_front();
16         std::cout << a[q.front()] << ' ';
17     }
18 }
19 std::cout << '\n';
20
21 while (!q.empty())
22     q.pop_back();
23
24 for (int i = 0; i < n; ++i) { //输出最大值
25     while (!q.empty() && a[q.back()] <= a[i])
26         q.pop_back();
27     q.emplace_back(i);
28     if (i >= k - 1) {
29         while (!q.empty() && q.front() <= i - k)
30             q.pop_front();
31         std::cout << a[q.front()] << ' ';
32     }
33 }
34 std::cout << '\n';

```

1.2 单调栈

输出右边第一个比 a_i 大的数的下标。

```

1 int n;
2 std::cin >> n;
3
4 std::vector<int> a(n), ans(n);
5 for (int &i : a)
6     std::cin >> i;
7
8 std::stack<int> s;
9 for (int i = n - 1; i >= 0; --i) {
10     while (!s.empty() && a[s.top()] <= a[i])
11         s.pop();
12     if (s.empty())
13         ans[i] = -1;
14     else
15         ans[i] = s.top();
16     s.emplace(i);
17 }

```

1.3 并查集

```

1 struct dsu {
2     // 元素范围 [0, n)
3     int n;
4     std::vector<int> f, sz;
5
6     dsu(int n) : n(n), f(n), sz(n, 1) { std::iota(f.begin(), f.end(), 0); }
7     int father(int u) {
8         if (u >= n) return -1;
9         while (u != f[u]) u = f[u] = f[f[u]];
10        return u;
11    }
12    bool same(int u, int v) {
13        int a = father(u), b = father(v);
14        return a == b && a != -1;
15    }
16    void merge(int u, int v) {
17        int a = father(u), b = father(v);
18        if (sz[a] > sz[b])
19            f[b] = a, sz[a] += sz[b];
20        else
21            f[a] = b, sz[b] += sz[a];
22    }
23    int size(int u) { return sz[father(u)]; }
24 };

```

2 基本算法

2.1 快速幂/整数类/取模

```

1 constexpr int P = 1000000007;
2 int norm(int x) {
3     if (x < 0) x += P;
4     if (x >= P) x -= P;
5     return x;
6 }
7 template <class T> constexpr T power(T a,
8     int64_t b) {
9     T res = 1;
10    for (; b; b /= 2, a *= a)
11        if (b % 2) res *= a;
12    return res;
13 }
14 struct Z {
15     int x;
16     Z(int x = 0) : x(norm(x)) {}
17     Z(int64_t x) : x(norm(x % P)) {}
18     int val() const { return x; }
19     Z operator-() const { return Z(norm(P - x)); }
20
21     Z inv() const {
22         assert(x != 0);
23         return power(*this, P - 2);
24     }
25     Z &operator*=(const Z &rhs) {
26         x = int64_t(x) * rhs.x % P;
27         return *this;
28     }
29     Z &operator+=(const Z &rhs) {
30         x = norm(x + rhs.x);

```

```

30     return *this;
31 }
32 Z &operator--(const Z &rhs) {
33     x = norm(x - rhs.x);
34     return *this;
35 }
36 Z &operator/=(const Z &rhs) { return *this
    *= rhs.inv(); }
37 friend Z operator*(const Z &lsh, const Z &
    rhs) {
38     Z res = lsh;
39     res *= rhs;
40     return res;
41 }
42 friend Z operator+(const Z &lsh, const Z &
    rhs) {
43     Z res = lsh;
44     res += rhs;
45     return res;
46 }
47 friend Z operator-(const Z &lsh, const Z &
    rhs) {
48     Z res = lsh;
49     res -= rhs;
50     return res;
51 }
52 friend Z operator/(const Z &lsh, const Z &
    rhs) {
53     Z res = lsh;
54     res /= rhs;
55     return res;
56 }
57 friend std::istream &operator>>(std::istream
    &is, Z &a) {
58     int64_t v;
59     is >> v;
60     a = Z(v);
61     return is;
62 }
63 friend std::ostream &operator<<(std::ostream
    &os, const Z &a) { return os << a.val();
64 };

```

2.2 一二三维差分/前缀和

一维: 区间 $[l, r]$ 每个元素加上 x :

```

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

```

二维: $(x_1, y_1), (x_2, y_2)$ 每个元素加上 x :

```

1 d[x1][y1] += x;
2 d[x1][y2 + 1] -= x;
3 d[x2 + 1][y1] -= x;
4 d[x2 + 1][y2 + 1] += x;

```

三维: $(x_1, y_1, z_1), (x_2, y_2, z_2)$ 每个元素加上 x :

```

1 d[x1][y1][z1] += x;
2 d[x2 + 1][y1][z1] -= x;
3 d[x1][y1][z2 + 1] -= x;
4 d[x2 + 1][y1][z2 + 1] += x;
5 d[x1][y2 + 1][z1] -= x;

```

```

6 d[x2 + 1][y2 + 1][z1] += x;
7 d[x1][y2 + 1][z2 + 1] += x;
8 d[x2 + 1][y2 + 1][z2 + 1] -= x;

```

2.3 二分

```

1 int binary_search_1(const std::vector<int> &a,
    int x) {
2     // a 单调递增
3     // 在  $[0, n)$  中查找  $x$  或  $x$  的后继的位置
4     int n = a.size(), l = 0, r = n;
5     while (l < r) {
6         int mid = (l + r) / 2;
7         if (a[mid] >= x)
8             r = mid;
9         else
10            l = mid + 1;
11    }
12    return l;
13 }
14
15 int binary_search_2(const std::vector<int> &a,
    int x) {
16     // a 单调递增
17     // 在  $[0, n)$  中查找  $x$  或  $x$  的前驱的位置
18     int n = a.size(), l = 0, r = n;
19     while (l < r) {
20         int mid = (l + r + 1) / 2;
21         if (a[mid] <= x)
22             l = mid;
23         else
24             r = mid - 1;
25    }
26    return l;
27 }
28
29 double float_binary_search(/*...*/) {
30     constexpr double eps = 1e-7;
31     double l = 0, r = 1e18;
32     while (r - l > eps) {
33         double mid = (l + r) / 2;
34         if (/*...*/)
35             l = mid;
36         else
37             r = mid;
38    }
39    return l;
40 }

```

2.4 三分

如何取 mid_1 和 mid_2 ? 有以下两种基本方法。

(1) 三分: mid_1 和 mid_2 为 $[l, r]$ 的三等分点, 区间每次可以缩小 $1/3$ 。

(2) 近似三分: 计算 $[l, r]$ 的中间点 $mid = (l + r)/2$, 然后让 mid_1 和 mid_2 非常接近 mid , 如 $mid_1 = mid - eps$, $mid_2 = mid + eps$ 。区间每次可以缩小接近一半。

近似三分比三分要稍微快一点。不过, 在有些情况下 eps 过小可能导致 mid_1 和 mid_2 算出的结果相等, 从而判断错方向。

三分:

```

1 while (r - l > eps) {

```

```

2 double k = (r - l) / 3;
3 double mid1 = l + k, mid2 = r - k;
4 if (/*...*/)
5     r = mid2;
6 else
7     l = mid1;
8 }

```

近似三等分:

```

1 while (r - l > eps) {
2     double mid1 = (l + r) / 2;
3     double mid2 = mid1 + eps;
4     if (/*...*/)
5         r = mid2;
6     else
7         l = mid1;
8 }

```

整数三分:

```

1 while (r - l > 2) {
2     int mid1 = l + (r - l) / 3;
3     int mid2 = r - (r - l) / 3;
4     if (/*...*/)
5         r = mid2;
6     else
7         l = mid1;
8 }

```

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:

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

$O(n \log n)$ 预处理、 $O(1)$ 查询。

```

1 int N, M;
2 std::cin >> N >> M;
3
4 std::vector<int> a(N);
5 for (int &i : a) std::cin >> i;
6
7 std::vector<int> LOG2(N + 1);
8 LOG2[0] = -1;
9 for (int i = 1; i <= N; ++i)
10     LOG2[i] = LOG2[i / 2] + 1;
11
12 int dp[N][21];
13 for (int i = 0; i < N; ++i)
14     dp[i][0] = a[i];
15
16 int p = LOG2[N];
17 // 可倍增区间的最大次方:  $2^p \leq N$ 
18
19 for (int k = 1; k <= p; ++k)
20     for (int i = 0; i + (1 << k) - 1 < N; ++i)
21         dp[i][k] = std::max(dp[i][k-1], dp[i + (1 << (k-1))][k-1]);
22
23 while (M--) {
24     int l, r;
25     // 求 [l, r] 中最大值
26     std::cin >> l >> r;

```

```

27 --l;
28
29 int len = r - l;
30 int k = LOG2[len];
31
32 std::cout << std::max(dp[l][k], dp[r - (1 << k)][k]) << '\n';
33 }

```

ST 类:

```

1 struct ST {
2     size_t n;
3     std::vector<std::vector<int>> info;
4
5     int merge(const int &lhs, const int &rhs) {
6         // TODO: 修改 merge
7         return std::max(lhs, rhs);
8     }
9
10    ST(const std::vector<int> &init) : n(init.size()), info(init.size() + 1, std::vector<int>(std::lg(n) + 1, INT_MIN)) {
11        // TODO: 修改初值
12        for (size_t x = 0; x <= std::lg(n); ++x)
13            for (int i = 0; i + (1ull << x) <= n; ++i)
14                if (x == 0)
15                    info[i][x] = init[i];
16                else
17                    info[i][x] = merge(info[i][x-1], info[i + (1 << (x-1))][x-1]);
18    }
19
20    int query(int l, int r) { // [l, r)
21        int len = r - l;
22        assert(len > 0);
23        return merge(info[l][std::lg(len)], info[r - (1 << std::lg(len))][std::lg(len)]);
24    }
25 };

```

2.6 归并排序/逆序对

```

1 int n;
2 std::cin >> n;
3
4 std::vector<int> a(n), b(n);
5 for (int &i : a) std::cin >> i;
6
7 int64_t ans = 0;
8 std::function<void(int, int)> merge_sort = [&](
9     int l, int r) {
10     // 排序区间 [l, r)
11     if (r - l <= 1) return;
12     int mid = (l + r) / 2;
13     merge_sort(l, mid);
14     merge_sort(mid, r);
15     for (int i = l, j = mid, k = l; k < r; ++k)
16         if (j == r || (i < mid && a[i] <= a[j]))
17             b[k] = a[i++];

```

```
17         else {
18             ans += mid - i;
19             b[k] = a[j++];
20         }
21     }
22     for (int i = l; i < r; ++i) a[i] = b[i];
23 };
24
25 merge_sort(0, n);
26 std::cout << ans << '\n';
```