南京工業大學
NANJING TECH UNIVERSITY

ICPC Template Manual

作者: 贺梦杰

August 24, 2019

Contents

Chapter 1

基础

Chapter 2

搜索

Chapter 3

动态规划

Chapter 4

字符串

Chapter 5

数据结构

5.1 归并树

5.1.1 简介

归并排序，自底向上越来越有序。而归并树则是将归并排序的中间结果保留了下来。
为什么要这么做呢？因为这样当询问区间时，总是可以将询问区间二分成某一次或几次归并排序的中间结果。

5.1.2 v 在区间内的排名

5.1.2.1 问题简述

给定 n 个数，m 次查询，每次查询 [l,r] 内从小到大第 k 个数，输出这个数。

5.1.2.2 解决方案

对这 n 个数先排序，然后二分尝试，将二分得到的中间值代入归并树中，看是否排名为 k，**对所有排名小于等于 k 的取最大值**。
建树时间复杂度 O(NlogN)，查询时间复杂度 O(logNlogN)。

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int N = 1e5 + 10, INF = 1e9 + 8;

int a[N], n, m;

struct Node {
    int l, r, m;
    vector<int> a;
} t[N * 3];

// 由a[]数组初始化归并树
void build(int l, int r, int x) {
    Node& cur = t[x];
    cur.l = l, cur.r = r, cur.m = (l + r) / 2;
    cur.a.clear();
    if (r - l == 1) {
        cur.a.push_back(a[cur.l]);
        return;
    }
    build(l, cur.m, x * 2);
    build(cur.m, r, x * 2 + 1);
    Node &lch = t[x * 2], &rch = t[x * 2 + 1];
    vector<int>::iterator it1, it2;
    it1 = lch.a.begin(), it2 = rch.a.begin();
    // 合并左右子树
    while (it1 != lch.a.end() && it2 != rch.a.end()) {
        if (*it1 <= *it2)
            cur.a.push_back(*it1), it1++;
        else
            cur.a.push_back(*it2), it2++;
    }
    while (it1 != lch.a.end())
        cur.a.push_back(*it1), it1++;
    while (it2 != rch.a.end())
        cur.a.push_back(*it2), it2++;
}

// 查询在区间[l,r)内比v小的数的个数，稍加修改便可查询大于等于k的最小值
int query(int l, int r, int v, int x) {
    Node& cur = t[x];
    if (cur.l == l && cur.r == r)
        return lower_bound(cur.a.begin(), cur.a.end(), v) - cur.a.begin();
    if (r <= cur.m)
        return query(l, r, v, x * 2);
    else if (l >= cur.m)
```

```
49          return query(l, r, v, x * 2 + 1);
50      else
51          return query(l, cur.m, v, x * 2) + query(cur.m, r, v, x * 2 + 1);
52  }
53
54  // 查询区间[l,r)区间内的从小到大第k个值
55  int QR(int ql, int qr, int k) {
56      int l = 1, r = n + 1, m, ans = -INF;
57      while (l < r) {
58          m = (l + r) / 2;
59          int rank = query(ql, qr, a[m], 1) + 1;
60          if (rank <= k)  // 可能有相等的值，所以需要<=
61              ans = max(ans, a[m]);
62          if (rank <= k)
63              l = m + 1;
64          else
65              r = m;
66      }
67      return ans;
68  }
69
70  int main() {
71      int i, l, r, k;
72      while (scanf("%d%d", &n, &m) != EOF) {
73          for (i = 1; i <= n; i++)
74              scanf("%d", &a[i]);
75          build(1, n + 1, 1);
76          sort(a + 1, a + 1 + n);
77          for (i = 1; i <= m; i++) {
78              scanf("%d%d%d", &l, &r, &k);
79              printf("%d\n", QR(l, r + 1, k));
80          }
81      }
82
83      return 0;
84  }
```

5.1.2.3 思考

如何取区间内不重复的第 k 大?
结点中用两个 vector，其中一个是原来的不变，另一个用于存放不重复的数，查询的时候在后者中查。

5.1.3 区间内大于等于 v 的最小值

5.1.3.1 问题简述

给定 n 个数，m 次查询，每次查询 [l,r] 内比 v 大的最小值，输出这个最小值。

5.1.3.2 解决方案

归并树天生适合的问题。对于对应区间，直接返回 lower_bound 找到的第一个数注意，如果没找到返回一个无限大，对于其余祖先节点，返回左右子树中找到的最小值。

```
1   int query(int l, int r, int v, int x) {
2       Node& cur = t[x];
3       if (cur.l == l && cur.r == r) {
4           vector<int>::iterator it;
5           it = lower_bound(cur.a.begin(), cur.a.end(), v);
6           if (it == cur.a.end())
7               return n + 1;
8           return *it;
9       }
10      if (r <= cur.m)
11          return query(l, r, v, x * 2);
12      else if (l >= cur.m)
13          return query(l, r, v, x * 2 + 1);
```

```
14      else
15          return min(query(l, cur.m, v, x * 2), query(cur.m, r, v, x * 2 + 1));
16  }
```

Chapter 6

图论