



南京工業大學
NANJING TECH
UNIVERSITY

ICPC Template Manual



作者: 贺梦杰

September 19, 2019

Contents

1 基础	2
1.1 测试	3
2 搜索	4
3 动态规划	5
3.1 最长上升子序列	6
3.1.1 基本实现	6
3.1.2 另一种解法：权值线段树	6
3.1.3 输出最小字典序的下标	7
3.1.4 输出值的最小字典序	8
3.1.5 最长不降子序列	9
4 字符串	10
5 数据结构	11
6 图论	12

Chapter 1

基础

1.1 测试

Chapter 2

搜索

Chapter 3

动态规划

3.1 最长上升子序列

3.1.1 基本实现

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  const int N = 1e3 + 10;
5
6  int n, a[N];
7  int b[N], c[N]; // b[i]是以a[i]为右端点的最长上升子序列的长度, c[i]是长度为i的最长上升子序列的右端点的
    最小值
8
9  inline void solve() {
10     int i, p, len = 0;
11     for (i = 1; i <= n; i++) {
12         p = lower_bound(c + 1, c + 1 + len, a[i]) - c;
13         b[i] = p, c[p] = a[i], len = max(len, p);
14     }
15 }
16
17 int main() {
18     ios::sync_with_stdio(0);
19     cin.tie(0);
20
21     int i;
22     cin >> n;
23     for (i = 1; i <= n; i++)
24         cin >> a[i];
25     solve();
26     int ans = 0;
27     for (i = 1; i <= n; i++)
28         ans = max(ans, b[i]);
29     cout << ans << endl;
30
31     return 0;
32 }

```

3.1.2 另一种解法：权值线段树

对于 $a[i]$ ，我们可以通过查找以比 $a[i]$ 小的数为结尾的最长长度，然后 $+1$ 即可得到以 $a[i]$ 为结尾的最长长度。权值线段树维护以某权值为结尾的最大长度。注意需要离散化。由于是查前缀，所以权值线段树可以换用树状数组

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  const int N = 1e5 + 10;
5
6  int n, a[N];
7  int b[N]; // b[i]是以a[i]为右端点的最长上升子序列的长度
8  int i2x[N]; // 离散化
9  int m; // 离散化的长度
10 int t[N]; // 树状数组
11
12 inline int x2i(int x) {
13     return lower_bound(i2x + 1, i2x + 1 + m, x) - i2x;
14 }
15
16 inline int lowbit(int x) {
17     return -x & x;
18 }
19
20 inline void upd(int p, int v) {
21     for (; p <= m; p += lowbit(p))

```

```

22         t[p] = max(t[p], v);
23     }
24
25     inline int qry(int p) {
26         int ret = 0;
27         for (; p > 0; p -= lowbit(p))
28             ret = max(ret, t[p]);
29         return ret;
30     }
31
32     inline void solve() {
33         int i;
34         for (i = 1; i <= n; i++) {
35             b[i] = qry(x2i(a[i]) - 1) + 1;
36             upd(x2i(a[i]), b[i]);
37         }
38     }
39
40     int main() {
41         ios::sync_with_stdio(0);
42         cin.tie(0);
43
44         int i;
45         cin >> n;
46         for (i = 1; i <= n; i++)
47             cin >> a[i], i2x[i] = a[i];
48         sort(i2x + 1, i2x + 1 + n);
49         m = unique(i2x + 1, i2x + 1 + n) - i2x - 1;
50
51         solve();
52
53         int ans = 0;
54         for (i = 1; i <= n; i++)
55             ans = max(ans, b[i]);
56         cout << ans << endl;
57
58         return 0;
59     }

```

3.1.3 输出最小字典序的下标

原问题是记录了以 $a[i]$ 为结尾的最长上升子序列的长度，而此问题是记录以 $a[i]$ 为**开头**的最长上升子序列的长度

PS: 若要求输出最大字典序的下标，只要还改成记录以 $a[i]$ 为**结尾**的最长上升子序列的长度，然后倒过来筛就可以了

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  const int N = 1e5 + 10;
5
6  int n, a[N];
7  int d[N], c[N]; // d[i]是以a[i]为左端点的最长上升子序列的长度, c[i]用于记录从右向左长度为i的最小左端点
8
9  // 此处定义的是小于号, 注意, 不可以有等于。lower_bound配上<=就是upper_bound。
10 // 所以无论是lower还是upper都不带等号
11 bool cmp(const int& lhs, const int& rhs) {
12     return lhs > rhs;
13 }
14
15 inline void solve() {
16     int i, len, mx = 0;
17     for (i = n; i >= 1; i--) {
18         len = lower_bound(c + 1, c + 1 + mx, a[i], cmp) - c;
19         d[i] = len, c[len] = a[i], mx = max(mx, len);
20     }

```



```

21 }
22
23 int main() {
24     ios::sync_with_stdio(0);
25     cin.tie(0);
26
27     int i;
28     cin >> n;
29     for (i = 1; i <= n; i++)
30         cin >> a[i];
31
32     solve();
33
34     int mx = 0;
35     for (i = 1; i <= n; i++)
36         mx = max(mx, d[i]);
37
38     cout << mx << endl;
39
40     int cur = mx, last = 0;
41     for (i = 1; i <= n; i++) {
42         if (d[i] == cur && a[i] > last) {
43             if (cur != mx)
44                 cout << " ";
45             cout << i; // 输出下标
46             --cur, last = a[i];
47         }
48     }
49     cout << endl;
50
51     return 0;
52 }

```

3.1.4 输出值的最小字典序

与上面问题不同的是，这里要求的是值的最小字典序。

例如：6 7 8 9 1 2 3 4，我们要输出的是 1 2 3 4。

对于这个问题，我们只要记录每个数的前驱即可。

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  const int N = 1e5 + 10;
5
6  int n, a[N];
7  int b[N], c[N]; // b[i]是以a[i]为右端点的最长上升子序列的长度，c[i]用于记录长度为i的最小端点
8  int pos[N], pre[N]; // pos[i]是与c[i]对应的下标，pre[i]是a[i]的前驱的下标
9  int ans[N];
10 // a[] b[] pre[] 下标一致；c[] pos[] 下标一致
11
12 inline void solve() {
13     int i, len, mx = 0;
14     for (i = 1; i <= n; i++) {
15         len = lower_bound(c + 1, c + 1 + mx, a[i]) - c;
16         b[i] = len, pre[i] = pos[len - 1];
17         c[len] = a[i], pos[len] = i;
18         mx = max(mx, len);
19     }
20 }
21
22 int main() {
23     ios::sync_with_stdio(0);
24     cin.tie(0);
25
26     int i, T;
27     cin >> T;

```

```
28     while (T--) {
29         cin >> n;
30         for (i = 1; i <= n; i++)
31             cin >> a[i];
32
33         solve();
34
35         int mx = 0;
36         for (i = 1; i <= n; i++)
37             mx = max(mx, b[i]);
38
39         cout << mx << endl;
40
41         int p = pos[mx], len = mx;
42         while (p) {
43             ans[len--] = a[p];
44             p = pre[p];
45         }
46
47         for (i = 1; i <= mx; i++) {
48             if (i > 1)
49                 cout << " ";
50             cout << ans[i];
51         }
52         cout << endl;
53     }
54
55     return 0;
56 }
```

3.1.5 最长不降子序列

```
1  p = upper_bound(c + 1, c + 1 + len, a[i]) - c;
```

Chapter 4

字符串

Chapter 5

数据结构

Chapter 6

图论