南京工業大學
NANJING TECH UNIVERSITY

ICPC Template Manual

作者: 贺梦杰

September 19, 2019

Contents

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

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

using namespace std;
const int N = 1e3 + 10;

int n, a[N];
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

inline void solve() {
    int i, p, len = 0;
    for (i = 1; i <= n; i++) {
        p = lower_bound(c + 1, c + 1 + len, a[i]) - c;
        b[i] = p, c[p] = a[i], len = max(len, p);
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int i;
    cin >> n;
    for (i = 1; i <= n; i++)
        cin >> a[i];
    solve();
    int ans = 0;
    for (i = 1; i <= n; i++)
        ans = max(ans, b[i]);
    cout << ans << endl;

    return 0;
}
```

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

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

using namespace std;
const int N = 1e5 + 10;

int n, a[N];
int b[N];    // b[i]是以a[i]为右端点的最长上升子序列的长度
int i2x[N];  // 离散化
int m;       // 离散化的长度
int t[N];    // 树状数组

inline int x2i(int x) {
    return lower_bound(i2x + 1, i2x + 1 + m, x) - i2x;
}

inline int lowbit(int x) {
    return -x & x;
}

inline void upd(int p, int v) {
    for (; p <= m; p += lowbit(p))
```

```
        t[p] = max(t[p], v);
}

inline int qry(int p) {
    int ret = 0;
    for (; p > 0; p -= lowbit(p))
        ret = max(ret, t[p]);
    return ret;
}

inline void solve() {
    int i;
    for (i = 1; i <= n; i++) {
        b[i] = qry(x2i(a[i]) - 1) + 1;
        upd(x2i(a[i]), b[i]);
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int i;
    cin >> n;
    for (i = 1; i <= n; i++)
        cin >> a[i], i2x[i] = a[i];
    sort(i2x + 1, i2x + 1 + n);
    m = unique(i2x + 1, i2x + 1 + n) - i2x - 1;

    solve();

    int ans = 0;
    for (i = 1; i <= n; i++)
        ans = max(ans, b[i]);
    cout << ans << endl;

    return 0;
}
```

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

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

using namespace std;
const int N = 1e5 + 10;

int n, a[N];
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

// 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。
// 所以无论是lower还是upper都不带等号
bool cmp(const int& lhs, const int& rhs) {
    return lhs > rhs;
}

inline void solve() {
    int i, len, mx = 0;
    for (i = n; i >= 1; i--) {
        len = lower_bound(c + 1, c + 1 + mx, a[i], cmp) - c;
        d[i] = len, c[len] = a[i], mx = max(mx, len);
    }
```

```
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int i;
    cin >> n;
    for (i = 1; i <= n; i++)
        cin >> a[i];

    solve();

    int mx = 0;
    for (i = 1; i <= n; i++)
        mx = max(mx, d[i]);

    cout << mx << endl;

    int cur = mx, last = 0;
    for (i = 1; i <= n; i++) {
        if (d[i] == cur && a[i] > last) {
            if (cur != mx)
                cout << " ";
            cout << i;  // 输出下标
            --cur, last = a[i];
        }
    }
    cout << endl;

    return 0;
}
```

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。

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

using namespace std;
const int N = 1e5 + 10;

int n, a[N];
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
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
int ans[N];
// a[] b[] pre[] 下标一致；c[] pos[] 下标一致

inline void solve() {
    int i, len, mx = 0;
    for (i = 1; i <= n; i++) {
        len = lower_bound(c + 1, c + 1 + mx, a[i]) - c;
        b[i] = len, pre[i] = pos[len - 1];
        c[len] = a[i], pos[len] = i;
        mx = max(mx, len);
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int i, T;
    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