## 定义和复杂度

无旋 Treap 也是一种自平衡二叉搜索树，也就是说，

其节点 u 的 key(u) 满足 key(u→lson) ≤ key(u) 和 key(u) ≤ key(u→rson)。

而无旋 Treap 为每个节点 u 随机指定 prio(u)，要求 prio(u) 满足最大堆性质，也即 key(u→lson) ≤ key(u) 和 key(u→rson) ≤ key(u)。

要求数值随机的 prio(u) 满足最大堆性质，实质上就相当于将整个二叉搜索树的结构近似于随机二叉搜索树（RBST）。

这点如何证明呢？考虑构造 RBST 所需要的随机插入顺序，先插入的永远在上面，把插入顺序当作一种优先级，实际上满足的就是堆性质。

## 实现和例题

首先是 DeepSeek 小姐为我们带来的示例代码。

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <climits>
using namespace std;

struct Node {
    int x, y;
    int prio, max_y;
    Node *left, *right;
    Node(int x, int y) : x(x), y(y), prio(rand()), max_y(y), left(nullptr),
right(nullptr) {}
};

void update(Node *node) {
    if (!node) return;
    node->max_y = node->y;
    if (node->left)
        node->max_y = max(node->max_y, node->left->max_y);
    if (node->right)
        node->max_y = max(node->max_y, node->right->max_y);
}

// 按 x 分裂为 <= key 和 > key 两部分
void split(Node *root, int key, Node *&a, Node *&b) {
    if (!root) {
        a = b = nullptr;
        return;
    }
    if (root->x <= key) {
        a = root;
        split(root->right, key, a->right, b);
        update(a); // 分裂后更新子树信息
    } else {
        b = root;
        split(root->left, key, a, b->left);
        update(b);
    }
```

```
}

// 合并两棵 Treap（a 所有节点的 x <= b 的 x）
Node* merge(Node *a, Node *b) {
    if (!a) return b;
    if (!b) return a;
    if (a->prio > b->prio) {
        a->right = merge(a->right, b);
        update(a);
        return a;
    } else {
        b->left = merge(a, b->left);
        update(b);
        return b;
    }
}

// 插入或更新节点（保证 x 唯一，y 取最大值）
void insert(Node *&root, int x, int y) {
    Node *left, *mid_right;
    split(root, x - 1, left, mid_right);  // left: <=x-1, mid_right: >=x

    Node *mid, *right;
    split(mid_right, x, mid, right);      // mid: ==x, right: >x

    if (mid) {  // 若 x 已存在，更新 y 为较大值
        mid->y = max(mid->y, y);
        update(mid);  // 更新当前节点及其子树
    } else {     // 否则创建新节点
        mid = new Node(x, y);
    }

    // 合并回原树
    mid_right = merge(mid, right);
    root = merge(left, mid_right);
}

// 查询区间 [a, b] 的最大 y 值
int query_max(Node *&root, int a, int b) {
    Node *left_part, *right_part;
    split(root, b, left_part, right_part);  // left_part: <=b

    Node *left_left, *mid_part;
    split(left_part, a - 1, left_left, mid_part);  // mid_part: [a, b]

    int max_y = INT_MIN;
    if (mid_part) max_y = mid_part->max_y;

    // 合并恢复树结构
    left_part = merge(left_left, mid_part);
    root = merge(left_part, right_part);

    return max_y;
}

// 示例测试
```

```cpp
int main() {
    srand(time(nullptr));
    Node *root = nullptr;

    insert(root, 3, 5);
    insert(root, 1, 3);
    insert(root, 5, 10);
    cout << "查询 [2,4]: " << query_max(root, 2, 4) << endl;  // 输出 5

    insert(root, 3, 7);  // 更新 x=3 的 y 为 7
    cout << "查询 [2,4]: " << query_max(root, 2, 4) << endl;  // 输出 7

    insert(root, 4, 2);
    cout << "查询 [2,4]: " << query_max(root, 2, 4) << endl;  // 输出 7

    insert(root, 4, 9);  // 更新 x=4 的 y 为 9
    cout << "查询 [2,4]: " << query_max(root, 2, 4) << endl;  // 输出 9

    return 0;
}
```

题目在 **这里**，提交记录在 **这里**。