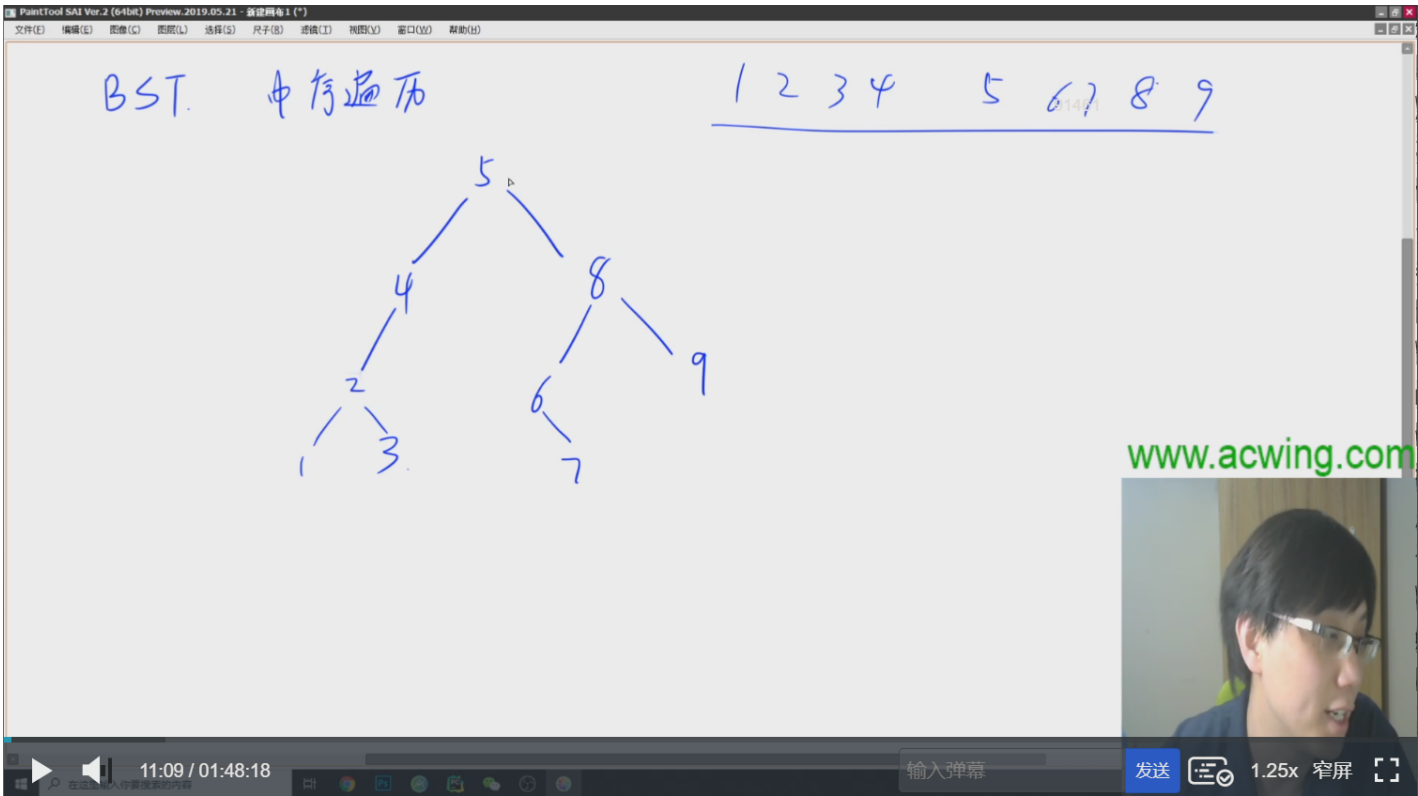


# L2C4 Lesson6

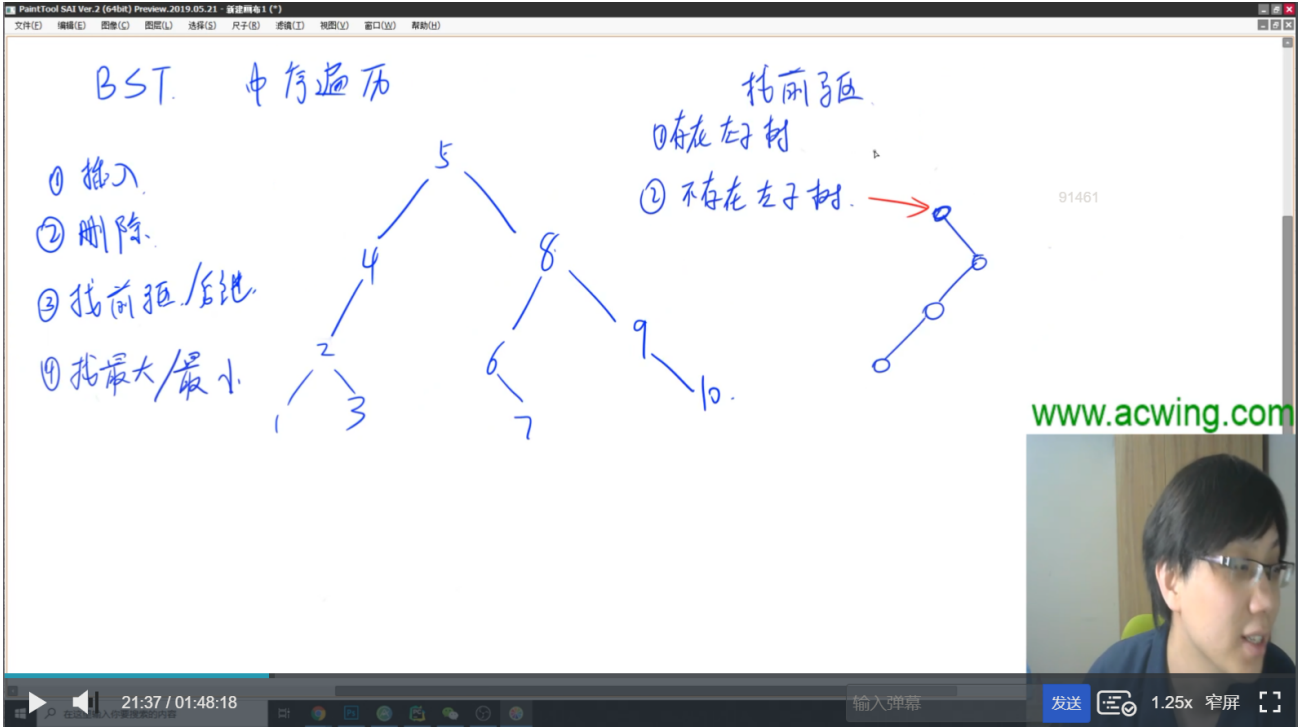
- 平衡树 (treap)
- tree + heap
- 同类型: 红黑树, splay (重点,) , sbt, AVL
- 红黑树太麻烦, 难写, 其中splay, treap常用

## 二叉搜索树 (Binary Search tree BST)

- 性质:
- 1. 当前结点左子树中的任意一个点的权值 < 当前结点权值
  - 2. 当前结点右子树中的任意一个点的权值 > 当前结点权值
  - 3. 一般情况下没有相同权值, 若有相同权值, 额外为每个结点维护一个cnt, 记录该权值出现多少次
  - 4. 主要看中序(根的位置)遍历, 即按照左->根->右的顺序遍历
    - 1. BST中序遍历, 最终结果一定是从小到大排序



- 5. 平衡树本质上, 主要作用是动态维护一个有序序列 (集合)
  - 6. 主要操作:
    - 1. 插入
    - 2. 删除: 在treap和splay里将结点换到叶结点上并删除
    - 3. 找前驱, 后继
      - 前驱: 中序遍历中当前结点前一个位置的数, 首元素无前驱
- 如何找前驱:
- 1. 当前结点存在左子树, 则前驱为左子树中的最大值, 走到左子树, 再一直往右走, 即为当前结点的前驱, 假设当前结点是p
    - left = p->left;
    - while (left->right) left = left->right; //最终left即为前驱
  - 2. 当前结点不存在左子树, 红色箭头所指结点即为尾结点前驱



- 后继：中序遍历中当前结点后一个位置的数，尾元素无后继
- 4. 找最大值（一直往右走）/最小值（一直往左走），1-4操作在Set中已经实现（insert, erase, ++/--, begin/end-1），若一题只涉及前四种操作，则可直接是使用Set来做
- 5. 求某个值的排名
- 6. 求排名是k（中序遍历中第k个数）的数是啥
- 7. lower-bound（比某数大的最小值，输入的值在平衡树中可能不存在，找到在平衡树中满足条件的值）
- 8. upper-bound（比某数小的最大值）

## Treap

Tree + heap

对于BST而言，每个操作一般而言时间复杂度和树高成正比，最坏情况退化成一条链，此时为 $O(n)$ 。

结论：一颗结构随机的BST，期望树高度是 $O(\log n)$ 级别

因此，treap的思想是让BST变得尽可能随机，从而让整棵树期望高度接近 $\log n$

结点性质：

```
1 Node {
2   int l, r;    //l和r存储左右儿子的编号
3   int key, value; //key存储用于BST排序的关键字，value是大根堆中的优先级
4   //同时满足两个条件，一是BST：key（中序遍历）有序满足BST条件
5   //二是大根堆：每个结点的value值>=左右儿子的value，且value是随机值
6 }tr[N];
```

性质：

- 只要所有key不同，所有value不同，则通过key, value就可以唯一确定一颗BST的结构，递归证明。如果对value随机，则整棵BST结构也会随机。随机BST的树高是 $\log N$ 。
- BST的结点个数是N，一个点对应一个数，线段树一个点可能对应多个数。因此，BST空间复杂度是 $O(N)$ 级别

具体操作：

- 初始化：BST一般加入两个哨兵，防止出现边界问题，减少判断  
即根节点为 $-\infty$ ，下一个结点为 $+\infty$ ，保证所有用到的值都处于 $[-\infty, +\infty]$ 之间
- 插入：通过递归将新结点插到某个叶子上，并给新节点赋值随机value，并通过回溯更新位置。所有BST都有左旋/右旋操作，即交换父结点与儿子的顺序。左右旋之后不会影响中序遍历的顺序，但是可以让父子结点的顺序发生交换。

BST. treap. Node

```

{
    int l, r;
    int key, val;
} tr[N];

```

插入  
删除  
找前驱/后继  
找最大/最小

求某个值的排名  
排名是卡的数是多少  
比某个数小的最大值  
比某个数大的最小值

右旋. zig  
左旋. zag

www.acwing.com

41:25 / 01:48:18

BST. treap. Node

```

{
    int l, r;
    int key, val;
} tr[N];

```

插入  
删除  
找前驱/后继  
找最大/最小

求某个值的排名  
排名是卡的数是多少  
比某个数小的最大值  
比某个数大的最小值

右旋. zig  
左旋. zag

www.acwing.com

- 右旋(zig)
- 左旋(zag)
- 删除：将要删除的结点左右旋至叶结点，并删除
  - 左儿子value大->右旋，右儿子value大->左旋

普通平衡树（模板题）

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  #include <algorithm>
5
6  using namespace std;          //253. 普通平衡树
7
8  const int N = 100010, INF = 1e8;
9
10 int n;
11 struct Node {
12     int l, r;    //l, r分别表示左右儿子的编号
13     int key, val; //key表示BST排序的权值, val表示大根堆的权值
14     int cnt, size; //cnt表示当前结点出现的次数, size表示左右子树和当前结点cnt之和, 用于求排名rank
15 } tr[N];
16
17 int root, idx; //root表示根结点, idx表示当前使用到了第几个结点, 和链表中idx同义
18
19 void pushup(int p) { //利用p的左右儿子信息维护p的信息, 信息往上传递
20     tr[p].size = tr[tr[p].l].size + tr[tr[p].r].size + tr[p].cnt;
21 }
22
23 int get_node(int key) { //创建一个结点
24     tr[++idx].key = key;
25     tr[idx].val = rand(); //创建结点时, 随机赋value值
26     tr[idx].cnt = tr[idx].size = 1; //创建结点时, 为叶结点, cnt = size = 1

```

```

27     return idx;    //返回创建节点的编号
28 }
29
30 void zig(int &p) {    //对结点p右旋，需要更改实参的值，故传引用
31     int q = tr[p].l;    //旋转后p的左儿子q将成为父结点
32     tr[p].l = tr[q].r, tr[q].r = p, p = q;    //p左结点=q右节点，q右节点=p，q成为新父结点
33     pushup(tr[p].r), pushup(p);    //更新原父结点与新父结点的信息，注意顺序从下往上
34
35 }
36
37 void zag(int &p) {    //对结点p左旋
38     int q = tr[p].r;
39     tr[p].r = tr[q].l, tr[q].l = p, p = q;    //对称，p右节点=q左结点，q左结点=p，q成为新父结点
40     pushup(tr[p].l), pushup(p);
41 }
42
43 void build() {    //初始化树
44     get_node(-INF), get_node(INF);    //创建两个哨兵结点(1, 2)，简化边界判断，0表示空结点
45     root = 1, tr[root].r = 2;    //初始化树结构，-INF为根结点，INF为右儿子
46     pushup(root);    //更新root结点信息
47
48     if (tr[1].val < tr[2].val) zag(root);    //满足大根堆
49 }
50
51 void insert(int &p, int key) {    //插入一个key，需要更改实参的值，故传引用，递归实现
52     if (!p) p = get_node(key);    //到达空叶子位置，新建一个叶子结点，记得更新p
53     else if (tr[p].key == key) tr[p].cnt++;    //当前结点key==插入key，cnt++
54     else if (tr[p].key > key) {
55         insert(tr[p].l, key);    //在左子树中去插入key
56         if (tr[tr[p].l].val > tr[p].val) zig(p);    //插入后如果左子树val>父val，右旋父结点
57     }
58     else {
59         insert(tr[p].r, key);    //对称，左子树中去插入key
60         if (tr[tr[p].r].val > tr[p].val) zag(p);    //右子树val>父val，左旋父结点
61     }
62
63     pushup(p);    //从下往上更新p结点信息
64 }
65
66 void remove(int &p, int key) {    //删除一个key，需要更改实参的值，故传引用，递归实现
67     if (!p) return;    //删除的key不存在，直接return
68     if (tr[p].key == key) {
69         if (tr[p].cnt > 1) tr[p].cnt--;    //当前结点个数>1，cnt--;
70         else if (tr[p].l || tr[p].r) {    //当前结点存在左子树或右子树
71             if (!tr[p].r || tr[tr[p].l].val > tr[tr[p].r].val) {
72                 //当前结点右子树不存在 || 左儿子val > 右儿子val -> 右旋
73                 zig(p);    //右旋，右旋后原p结点 = 新p结点.r
74                 remove(tr[p].r, key);    //将原p结点递归旋转至叶子结点，并删除
75             }
76             else {    //否则左旋
77                 zag(p);    //左旋
78                 remove(tr[p].l, key);
79             }
80         }
81         else p = 0;    //已经旋转至叶子结点，令p=0直接删除
82     }
83     else if (tr[p].key > key) remove(tr[p].l, key);    //去左子树中删
84     else remove(tr[p].r, key);    //去右子树中删
85
86     pushup(p);    //从下往上更新p结点信息
87 }
88
89 int get_rank_by_key(int p, int key) {    //根据key找排名，不涉及修改操作，不用传引用
90     if (!p) return 0;    //所找key不存在，返回rank=0，本题中不存在此情况
91     if (tr[p].key == key) return tr[tr[p].l].size + 1;    //左子树size+1，即为rank
92     if (tr[p].key > key) return get_rank_by_key(tr[p].l, key);    //左子树中找
93     //左子树size+父结点cnt+右子树中找
94     return tr[tr[p].l].size + tr[p].cnt + get_rank_by_key(tr[p].r, key);
95 }
96
97 int get_key_by_rank(int p, int rank) {    //通过rank找key
98     if (!p) return INF;    //所找rank不存在，返回key=INF，本题中不存在此情况
99     if (tr[tr[p].l].size >= rank) return get_key_by_rank(tr[p].l, rank);    //左子树中找，注意>=
100     if (tr[tr[p].l].size + tr[p].cnt >= rank) return tr[p].key;    //此时p.key即为所找
101     return get_key_by_rank(tr[p].r, rank - tr[tr[p].l].size - tr[p].cnt);    //右子树中找，记得更新rank
102 }
103
104 int get_prev(int p, int key) {    //找前驱，严格小于key的最大数
105     if (!p) return -INF;
106     if (tr[p].key >= key) return get_prev(tr[p].l, key);    //左子树中找

```

```

107     return max(tr[p].key, get_prev(tr[p].r, key)); //注意需加上max，因为递归终点可能返回-INF
108 }
109
110 int get_next(int p, int key) { //找后继，严格大于key的最小数
111     if (!p) return INF;
112     if (tr[p].key <= key) return get_next(tr[p].r, key);
113     return min(tr[p].key, get_next(tr[p].l, key));
114 }
115
116 int main(void) {
117     build();
118
119     scanf("%d", &n);
120     while (n--) {
121         int opt, x;
122         scanf("%d%d", &opt, &x);
123
124         if (opt == 1) insert(root, x);
125         else if (opt == 2) remove(root, x);
126         else if (opt == 3) printf("%d\n", get_rank_by_key(root, x) - 1); // -1考虑左哨兵
127         else if (opt == 4) printf("%d\n", get_key_by_rank(root, x + 1)); // +1考虑左哨兵
128         else if (opt == 5) printf("%d\n", get_prev(root, x));
129         else printf("%d\n", get_next(root, x));
130     }
131
132     return 0;
133 }
134

```

营业额统计（平衡树的简单应用）

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  #include <algorithm>
5
6  using namespace std; //265. 营业额统计
7
8  typedef long long LL;
9
10 const int N = 33010, INF = 1e7;
11
12 int n;
13 struct Node {
14     int l, r;
15     int key, val;
16 }tr[N];
17
18 int root, idx;
19
20 int get_node(int key) {
21     tr[++idx].key = key;
22     tr[idx].val = rand();
23     return idx;
24 }
25
26 void zig(int &p) {
27     int q = tr[p].l;
28     tr[p].l = tr[q].r, tr[q].r = p, p = q;
29 }
30
31 void zag(int &p) {
32     int q = tr[p].r;
33     tr[p].r = tr[q].l, tr[q].l = p, p = q;
34 }
35
36
37 void build() {
38     get_node(-INF), get_node(INF);
39     root = 1, tr[1].r = 2;
40
41     if (tr[2].val > tr[1].val) zag(root);
42 }
43
44 void insert(int &p, int key) {
45     if (!p) p = get_node(key);
46     else if (tr[p].key == key) return; //不需要维护cnt和size
47     else if (tr[p].key > key) {

```

```

48     insert(tr[p].l, key);
49     if (tr[tr[p].l].val > tr[p].val) zig(p);
50 }
51 else {
52     insert(tr[p].r, key);
53     if (tr[tr[p].r].val > tr[p].val) zag(p);
54 }
55 }
56
57 int get_prev(int p, int key) {
58     if (!p) return -INF;
59     if (tr[p].key > key) return get_prev(tr[p].l, key);    //注意找<=的最大值，因此条件为>
60     return max(tr[p].key, get_prev(tr[p].r, key));
61 }
62
63 int get_next(int p, int key) {
64     if (!p) return INF;
65     if (tr[p].key < key) return get_next(tr[p].r, key);
66     return min(tr[p].key, get_next(tr[p].l, key));
67 }
68
69
70 int main(void) {
71     build();
72
73     scanf("%d", &n);
74     LL res = 0;
75     for (int i=0; i<n; i++) {
76         int x;
77         scanf("%d", &x);
78
79         if (i == 0) res += x;
80         else res += min(x - get_prev(root, x), get_next(root, x) - x);
81
82         insert(root, x);
83     }
84
85     printf("%lld\n", res);
86
87
88     return 0;
89 }

```

**堆 (heap 使用大根堆)**