平衡树

基本原理

算法竞赛中常用：**Treap**(BBST+堆)、红黑树(太麻烦，用set/map代劳)、**Splay**(最常用)、SBT(不常用)、AVL

BBST：一般权值互异；中序遍历单调；

- 本质：**动态维护有序序列**
- 操作：插入、删除、找前驱/后继（前驱后继定义在 中序遍历序列 中）、找最值
  - 前驱
    - 有左子树：则找左子树的最大值，即从左子树开始向右走到头
    - 无左子树：找第一个身为其右子树节点的父节点，即不断遍历父节点，直到父节点是向左拐的（成右子树）
  - 后驱类似
  - 以上操作在set中均有实现：insert, erase, ++/--, begin()/end()-1
- 操作：set中不存在，需手写
  - **求某个值的排名**（set中节点没有这个信息）
  - **求排名是k的数**（即nth_element)
  - **比某个数小的最大值**（lower_bound/upper_bound)
  - **比某个数大的最小值**：并非找前驱和后继，这是对值进行query，值可能不在

具体实现在Treap中
Treap：最简单的BBST

- 原理：使BST尽量随机，随机BST的期望高度O(log n)
- 节点结构

```
struct Node {
    int l, r;
    int key;  // BST中key有序
    int val;  // Heap中val满足堆序性（以大根为例）
} tr[N]; // 空间复杂度O(n)
```

- 初始化：一般会加两个哨兵，-infinity和+infinity
- 左旋zig，右旋zag：不破坏BST中序性质，但是可以交换两个节点（**从而满足堆序性**）

```
      x        zig        y
     / \       ==>       / \
    y   T3     <==      T1  x
```

```
   / \          zag          / \
  T1   T2                   T2 T3
```

这里为还是想把zig认作左旋（和yxc讲的不一样），与学ds时的认知一致；然后有**往哪侧旋，哪侧的值就会上升一层，另一侧下降**

- 删除：针对要删除的节点，不断旋转将其旋转至叶节点，然后删去；旋转时注意将大val向上转（即向大val处旋）

模板

以T254普通平衡树为模板题

您需要写一种数据结构来维护一些数，其中需要提供以下操作：

1. 插入数值 x。
2. 删除数值 x(若有多个相同的数，应只删除一个)。
3. 查询数值 x 的排名(若有多个相同的数，应输出最小的排名)。
4. 查询排名为 x 的数值。
5. 求数值 x 的前驱(前驱定义为小于 x 的最大的数)。(并非节点前后驱)
6. 求数值 x 的后继(后继定义为大于 x 的最小的数)。
   数据保证查询的结果一定存在；没有的情况模板也写了

```
const int N = 1e5 + 10, INF = 1e9;
#define LChild(p) (tr[tr[p].l])
#define RChild(p) (tr[tr[p].r])

int n;
struct Node {
    int l, r;
    int key, val;
    int cnt, size; // cnt: count of the number; size: node size of this
subtree
} tr[N];
int root, idx;

void pushup(int p) {
    tr[p].size = tr[tr[p].l].size + tr[tr[p].r].size + tr[p].cnt;
}

int createNode(int key) {
    tr[++idx].key = key;
    tr[idx].val = rand();
    tr[idx].cnt = tr[idx].size = 1;
    return idx;
}
```

```
void build() {
    createNode(-INF), createNode(INF);
    root = 1; tr[1].r = 2;
    pushup(root);
}

void zig(int& p) {
    int q = tr[p].l;
    tr[p].l = tr[q].r, tr[q].r = p, p = q;
    pushup(tr[p].r), pushup(p);
}

void zag(int& p) {
    int q = tr[p].r;
    tr[p].r = tr[q].l, tr[q].l = p, p = q;
    pushup(tr[p].l), pushup(p);
}

void insert(int& p, int key) {
    if (!p) p = createNode(key);
    else if (tr[p].key == key) tr[p].cnt++;
    else {
        if (tr[p].key > key) {
            insert(tr[p].l, key);
            if (tr[tr[p].l].val > tr[p].val) zig(p);
        } else {
            insert(tr[p].r, key);
            if (tr[tr[p].r].val > tr[p].val) zag(p);
        }
    }
    pushup(p);
}

void remove(int& p, int key) {
    if (!p) return;
    if (tr[p].key == key) {
        if (tr[p].cnt > 1) tr[p].cnt--;
        else if (tr[p].l || tr[p].r) {
            if (!tr[p].r || tr[tr[p].l].val > tr[tr[p].r].val) {
                zig(p);
                remove(tr[p].r, key);
            } else {
                zag(p);
                remove(tr[p].l, key);
            }
        } else p = 0;
    } else {
        if (tr[p].key > key) remove(tr[p].l, key);
        else remove(tr[p].r, key);
    }
```

```
    pushup(p);
}

int getRankByKey(int p, int key) {
    if (!p) return 0;
    if (tr[p].key == key) return tr[tr[p].l].size + 1;
    if (tr[p].key > key) return getRankByKey(tr[p].l, key);
    return tr[tr[p].l].size + tr[p].cnt + getRankByKey(tr[p].r, key);
}

int getKeyByRank(int p, int rank) {
    if (!p) return INF;
    if (tr[tr[p].l].size >= rank) return getKeyByRank(tr[p].l, rank);
    if (tr[tr[p].l].size + tr[p].cnt >= rank) return tr[p].key;
    return getKeyByRank(tr[p].r, rank - tr[tr[p].l].size - tr[p].cnt); //
miswrite getRankByKey
}

int getPrev(int p, int key) { // < key
    if (!p) return -INF;
    if (tr[p].key >= key) return getPrev(tr[p].l, key);
    return max(tr[p].key, getPrev(tr[p].r, key));
}

int getSucc(int p, int key) { // > key
    if (!p) return INF;
    if (tr[p].key <= key) return getSucc(tr[p].r, key);
    return min(tr[p].key, getSucc(tr[p].l, key));
}

int main() {
    build();
    scanf("%d", &n);

    int op, x;
    while (n--) {
        scanf("%d%d", &op, &x);
        switch (op) {
            case 1: insert(root, x); break;
            case 2: remove(root, x); break;
            case 3: printf("%d\n", getRankByKey(root, x) - 1); break;
            case 4: printf("%d\n", getKeyByRank(root, x + 1)); break;
            case 5: printf("%d\n", getPrev(root, x)); break;
            case 6: printf("%d\n", getSucc(root, x)); break;
            default: break;
        }
    }
    return 0;
}
```

应用

以 T265营业额统计 为例

- 计算最小波动。即对于a[i]来说，寻找a[1]～a[i-1]中和a[i]差最小得一个a[j] $\Longleftrightarrow$ 找大于等于a[i]得最小数，和小于等于a[i]得最大数，两个差取min即可

使用stl，set平衡树

```
const int N = 32767 + 10, INF = 1e8;
typedef long long ll;

set<int> nums;
int n;

int main() {
    scanf("%d", &n);
    nums.insert(INF);
    nums.insert(-INF);

    int a;
    scanf("%d", &a);
    nums.insert(a); n--;
    ll res = a;
    set<int>::iterator it_min, it_max;
    while (n--) {
        scanf("%d", &a);
        it_min = nums.lower_bound(a);
        if (*it_min != a) {
            it_max = it_min;
            it_max--;  // 前驱
            res += min(abs(*it_min - a), abs(a - *it_max));
            nums.insert(a);
        }
    }
    printf("%lld\n", res);
    return 0;
}
```

手写平衡树

```
const int N = 33000 + 10, INF = 1e8;
typedef long long ll;

int n;
struct Node {
    int l, r;
    int key, val;
```

```
} tr[N];
int root, idx;

int createNode(int key) {
    tr[++idx].key = key;
    tr[idx].val = rand();
    return idx;
}

void build() {
    createNode(-INF); createNode(INF);
    root = 1, tr[1].r = 2;
}

void zig(int& p) {
    int q = tr[p].l;
    tr[p].l = tr[q].r, tr[q].r = p, p = q;
}

void zag(int& p) {
    int q = tr[p].r;
    tr[p].r = tr[q].l, tr[q].l = p, p = q;
}

void insert(int& p, int key) {
    if (!p) p = createNode(key);
    else if (tr[p].key != key) { // 变更，相等则可以不管
        if (tr[p].key > key) {
            insert(tr[p].l, key);
            if (tr[p].val < tr[tr[p].l].val) zig(p);
        } else {
            insert(tr[p].r, key);
            if (tr[p].val < tr[tr[p].r].val) zag(p);
        }
    }
}

int getPred(int p, int key) {
    if (!p) return -INF;
    if (tr[p].key > key) return getPred(tr[p].l, key); // 变更为>，可以等
    return max(tr[p].key, getPred(tr[p].r, key));
}

int getSucc(int p, int key) {
    if (!p) return INF;
    if (tr[p].key < key) return getSucc(tr[p].r, key); // 变更为>，可以等
    return min(tr[p].key, getSucc(tr[p].l, key));
}

int main() {
```

```
    build();
    scanf("%d", &n);

    int a;
    scanf("%d", &a);
    ll res = a; insert(root, a);
    for (int i = 1; i < n; ++i) {
        scanf("%d", &a);
        res += min(a - getPred(root, a), getSucc(root, a) - a);
        insert(root, a);
    }
    printf("%lld\n", res);
    return 0;
}
```