fhq平衡树

原题链接

P3369 【模板】普通平衡树

节点定义

```
struct Node
{
    int val, key;
    int left, right;
    int size;
};
```

新建节点

```
inline int newNode(int val)
{
    tree[++cnt].val = val;
    tree[cnt].key = rand();
    tree[cnt].size = 1;
    return cnt;
}
```

更新节点信息

```
inline void update(int now)
{
    tree[now].size = tree[tree[now].left].size + tree[tree[now].right].size + 1;
}
```

分裂操作

将以now为根节点的子树分裂成两棵子树，根节点分别为ls，rs

ls子树上所有值小于等于val，rs子树上所有值大于val

```
void split(int now, int val, int& ls, int& rs)
{
    if (!now)
        ls = rs = 0;
    else
    {
        if (tree[now].val <= val)
        {
            ls = now;
            split(tree[now].right, val, tree[now].right, rs);
        }
```

```
        else
        {
            rs = now;
            split(tree[now].left, val, ls, tree[now].left);
        }
        update(now);
    }
}
```

合并操作

将两颗子树合并并返回新树的根节点

```
int merge(int ls, int rs)
{
    if (!ls || !rs) //如果某棵子树为空则返回另一颗子树的索引
        return ls + rs;
    if (tree[ls].key > tree[rs].key)
    {
        tree[ls].right = merge(tree[ls].right, rs);
        update(ls);
        return ls;
    }
    else
    {
        tree[rs].left = merge(ls, tree[rs].left);
        update(rs);
        return rs;
    }
}
```

插入操作

从根节点开始，把整颗树按照val拆成两颗子树

先把左子树和val合并再和右子树合并

```
void insert(int val)
{
    int ls, rs;
    split(root, val, ls, rs);
    root=merge(merge(ls, newNode(val)), rs);
}
```

删除操作

把树拆成三颗，左子树小于val，中子树等于val，右子树大于val

把中子树的左右子树合并，即舍弃中子树根节点

把三颗子树合并

```
void del(int val)
{
    int ls, ms, rs;
    split(root, val, ls, rs);
    split(ls, val - 1, ls, ms);
    ms = merge(tree[ms].left, tree[ms].right);
    root = merge(merge(ls, ms), rs);
}
```

获取排名

```
int getRank(int val)
{
    int ls, rs;
    split(root, val - 1, ls, rs);
    int ret = tree[ls].size + 1;
    root = merge(ls, rs);
    return ret;
}
```

根据排名获取值

```
int getVal(int rank)
{
    int now = root;
    while (now)
    {
        if (tree[tree[now].left].size + 1 == rank)
            break;
        else if (tree[tree[now].left].size >= rank)
            now = tree[now].left;
        else
        {
            rank -= tree[tree[now].left].size + 1;
            now = tree[now].right;
        }
    }
    return tree[now].val;
}
```

获取前驱

```
int getPre(int val)
{
    int ls, rs;
    split(root, val - 1, ls, rs);
    int now = ls;
    while (tree[now].right)
        now = tree[now].right;
    int ret = tree[now].val;
    root=merge(ls, rs);
    return ret;
}
```

获取后驱

```
int getNext(int val)
{
    int ls, rs;
    split(root, val, ls, rs);
    int now = rs;
    while (tree[now].left)
        now = tree[now].left;
    int ret = tree[now].val;
    root = merge(ls, rs);
    return ret;
}
```

完整代码

```
#include <bits/stdc++.h>
#define endl '\n'

using namespace std;

typedef long long ll;
typedef vector<int> VI;
typedef pair<int, int> PII;

const int maxn = 1e5 + 5;
const ll mod = 1e9 + 7;

struct Node
{
    int val, key;
    int left, right;
    int size;
};


Node tree[maxn];
int cnt, root;

inline int newNode(int val)
{
```

```
    tree[++cnt].val = val;
    tree[cnt].key = rand();
    tree[cnt].size = 1;
    return cnt;
}

inline void update(int now)
{
    tree[now].size = tree[tree[now].left].size + tree[tree[now].right].size + 1;
}

void split(int now, int val, int& ls, int& rs)
{
    if (!now)
        ls = rs = 0;
    else
    {
        if (tree[now].val <= val)
        {
            ls = now;
            split(tree[now].right, val, tree[now].right, rs);
        }
        else
        {
            rs = now;
            split(tree[now].left, val, ls, tree[now].left);
        }
        update(now);
    }
}

int merge(int ls, int rs)
{
    if (!ls || !rs) //如果某棵子树为空则返回另一颗子树的索引
        return ls + rs;
    if (tree[ls].key > tree[rs].key)
    {
        tree[ls].right = merge(tree[ls].right, rs);
        update(ls);
        return ls;
    }
    else
    {
        tree[rs].left = merge(ls, tree[rs].left);
        update(rs);
        return rs;
    }
}

void insert(int val)
{
    int ls, rs;
    split(root, val, ls, rs);
    //把整棵树按照val分成两颗子树
    root=merge(merge(ls, newNode(val)), rs);
```

```
    //先把val和左子树合并后再和右子树合并
}

void del(int val)
{
    int ls, ms, rs;
    split(root, val, ls, rs);
    split(ls, val - 1, ls, ms);
    ms = merge(tree[ms].left, tree[ms].right);
    root = merge(merge(ls, ms), rs);
}

int getRank(int val)
{
    int ls, rs;
    split(root, val - 1, ls, rs);
    int ret = tree[ls].size + 1;
    root = merge(ls, rs);
    return ret;
}

int getVal(int rank)
{
    int now = root;
    while (now)
    {
        if (tree[tree[now].left].size + 1 == rank)
            break;
        else if (tree[tree[now].left].size >= rank)
            now = tree[now].left;
        else
        {
            rank -= tree[tree[now].left].size + 1;
            now = tree[now].right;
        }
    }
    return tree[now].val;
}

int getPre(int val)
{
    int ls, rs;
    split(root, val - 1, ls, rs);
    int now = ls;
    while (tree[now].right)
        now = tree[now].right;
    int ret = tree[now].val;
    root=merge(ls, rs);
    return ret;
}

int getNext(int val)
{
    int ls, rs;
    split(root, val, ls, rs);
```

```
    int now = rs;
    while (tree[now].left)
        now = tree[now].left;
    int ret = tree[now].val;
    root = merge(ls, rs);
    return ret;
}

void solve()
{
    int T;
    scanf("%d", &T);
    while (T--)
    {
        int opt, x;
        scanf("%d %d", &opt, &x);
        if (opt == 1)
            insert(x);
        else if (opt == 2)
            del(x);
        else if (opt == 3)
            printf("%d\n", getRank(x));
        else if (opt == 4)
            printf("%d\n", getVal(x));
        else if (opt == 5)
            printf("%d\n", getPre(x));
        else if (opt == 6)
            printf("%d\n", getNext(x));
    }
}

int main()
{
    int T = 1;
    while (T--)
        solve();
    return 0;
}
```