【第二十三课】手撕红黑树-删除调整

red_black_tree_all

```
/*************************************************************************
  > File Name: 1.red_black_tree.cpp
  > Author: huguang
  > Mail: hug@haizeix.com
  > Created Time:
 ************************************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <queue>
#include <stack>
#include <algorithm>
#include <string>
#include <map>
#include <set>
#include <vector>
using namespace std;

#define NIL &(node::__NIL)
struct node {
    node(int key = 0, int color = 0, node *lchild = NIL, node *rchild = NIL)
    : key(key), color(color), lchild(lchild), rchild(rchild) {}
    int key;
    int color; // 0 red, 1 black, 2 double black
    node *lchild, *rchild;
    static node __NIL;
};

node node::__NIL(0, 1);

node *getNewNode(int key) {
    return new node(key);
}

bool has_red_child(node *root) {
    return root->lchild->color == 0 || root->rchild->color == 0;
}

node *left_rotate(node *root) {
    node *temp = root->rchild;
    root->rchild = temp->lchild;
    temp->lchild = root;
    return temp;
}
```

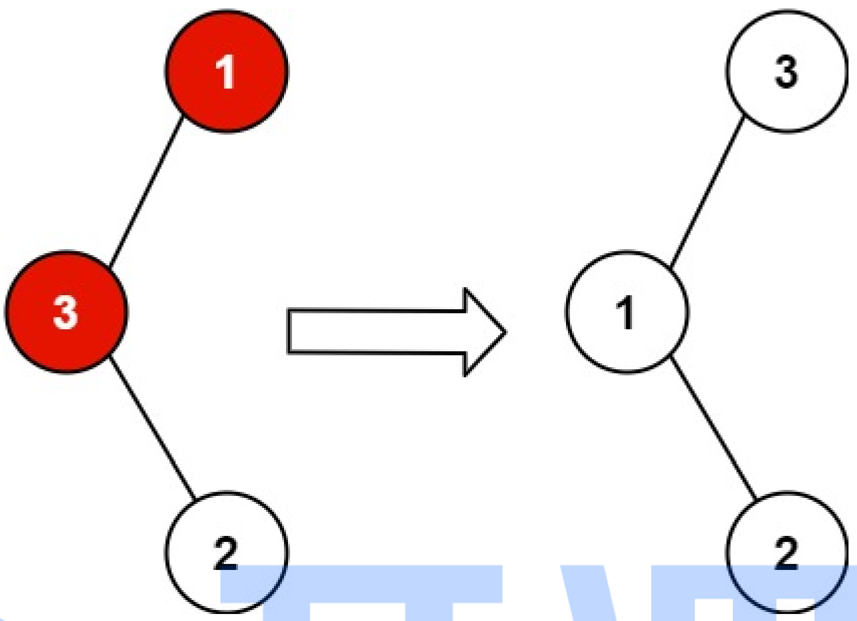
```
node *right_rotate(node *root) {
    node *temp = root->lchild;
    root->lchild = temp->rchild;
    temp->rchild = root;
    return temp;
}

node *insert_maintain(node *root) {
    int flag = 0;
    if (root->lchild->color == 0 && has_red_child(root->lchild)) flag = 1;
    if (root->rchild->color == 0 && has_red_child(root->rchild)) flag = 2;
    if (flag == 0) return root;
    if (root->lchild->color == 0 && root->rchild->color == 0) {
        root->color = 0;
        root->lchild->color = root->rchild->color = 1;
        return root;
    }
    if (flag == 1) {
        if (root->lchild->rchild->color == 0) {
            root->lchild = left_rotate(root->lchild);
        }
        root = right_rotate(root);
    } else {
        if (root->rchild->lchild->color == 0) {
            root->rchild = right_rotate(root->rchild);
        }
        root = left_rotate(root);
    }
    root->color = 0;
    root->lchild->color = root->rchild->color = 1;
    return root;
}

node *__insert(node *root, int key) {
    if (root == NIL) return getNewNode(key);
    if (key == root->key) return root;
    if (key < root->key) {
        root->lchild = __insert(root->lchild, key);
    } else {
        root->rchild = __insert(root->rchild, key);
    }
    return insert_maintain(root);
}

node *insert(node *root, int key) {
    root = __insert(root, key);
    root->color = 1;
    return root;
}

node *erase_maintain(node *root) {
    if (root->lchild->color != 2 && root->rchild->color != 2) return root;
    int flag = 0;
    if (has_red_child(root)) {
        root->color = 0;
        if (root->lchild->color == 0) {
            root = right_rotate(root); flag = 1;
        } else {
            root = left_rotate(root); flag = 2;
        }
        root->color = 1;
        if (flag == 1) root->rchild = erase_maintain(root->rchild);
        else root->lchild = erase_maintain(root->lchild);
        return root;
    }
    if ((root->lchild->color == 1 && !has_red_child(root->lchild))
        || (root->rchild->color == 1 && !has_red_child(root->rchild))) {
        root->lchild->color -= 1;
        root->rchild->color -= 1;
        root->color += 1;
        return root;
    }
    if (root->lchild->color == 1) {
        root->rchild->color = 1;
        if (root->lchild->lchild->color != 0) {
            root->lchild->color = 0;
            root->lchild = left_rotate(root->lchild);
            root->lchild->color = 1;
        }
        root->lchild->color = root->color;
        root = right_rotate(root);
    } else {
        root->lchild->color = 1;
        if (root->rchild->rchild->color != 0) {
            root->rchild->color = 0;
            root->rchild = right_rotate(root->rchild);
            root->rchild->color = 1;
        }
        root->rchild->color = root->color;
        root = left_rotate(root);
    }
    root->lchild->color = root->rchild->color = 1;
    return root;
}

node *predecessor(node *root) {
    node *temp = root->lchild;
    while (temp->rchild != NIL) temp = temp->rchild;
    return temp;
}

node *__erase(node *root, int key) {
    if (root == NIL) return root;
    if (key < root->key) {
        root->lchild = __erase(root->lchild, key);
    } else if (key > root->key) {
        root->rchild = __erase(root->rchild, key);
    } else {
        if (root->lchild == NIL || root->rchild == NIL) {
            node *temp = root->lchild == NIL ? root->rchild : root->lchild;
            temp->color += root->color;
            delete root;
            return temp;
        } else {
            node *temp = predecessor(root);
            root->key = temp->key;
            root->lchild = __erase(root->lchild, temp->key);
        }
    }
    return erase_maintain(root);
}

node *erase(node *root, int key) {
    root = __erase(root, key);
    root->color = 1;
    return root;
}

void clear(node *root) {
    if (root == NIL) return ;
    clear(root->lchild);
    clear(root->rchild);
    delete root;
    return ;
}

void print(node *root) {
    printf("( %d | %d, %d, %d )\n",
        root->color, root->key,
        root->lchild->key, root->rchild->key
    );
    return ;
}

void output(node *root) {
    if (root == NIL) return ;
    print(root);
    output(root->lchild);
    output(root->rchild);
    return ;
}

int main() {
    int op, val;
    node *root = NIL;
    while (cin >> op >> val) {
        switch (op) {
            case 1: root = insert(root, val); break;
            case 2: root = erase(root, val); break;
        }
        cout << endl << "==== rbtree print ====" << endl;
        output(root);
        cout << "==== rbtree print done ====" << endl;
    }
    return 0;
}
```

99. 恢复二叉搜索树

给你二叉搜索树的根节点 `root` ，该树中的两个节点被错误地交换。请在不改变其结构的情况下，恢复这棵树。

进阶：使用 O(*n*) 空间复杂度的解法很容易实现。你能想出一个只使用常数空间的解决方案吗？

示例 1：



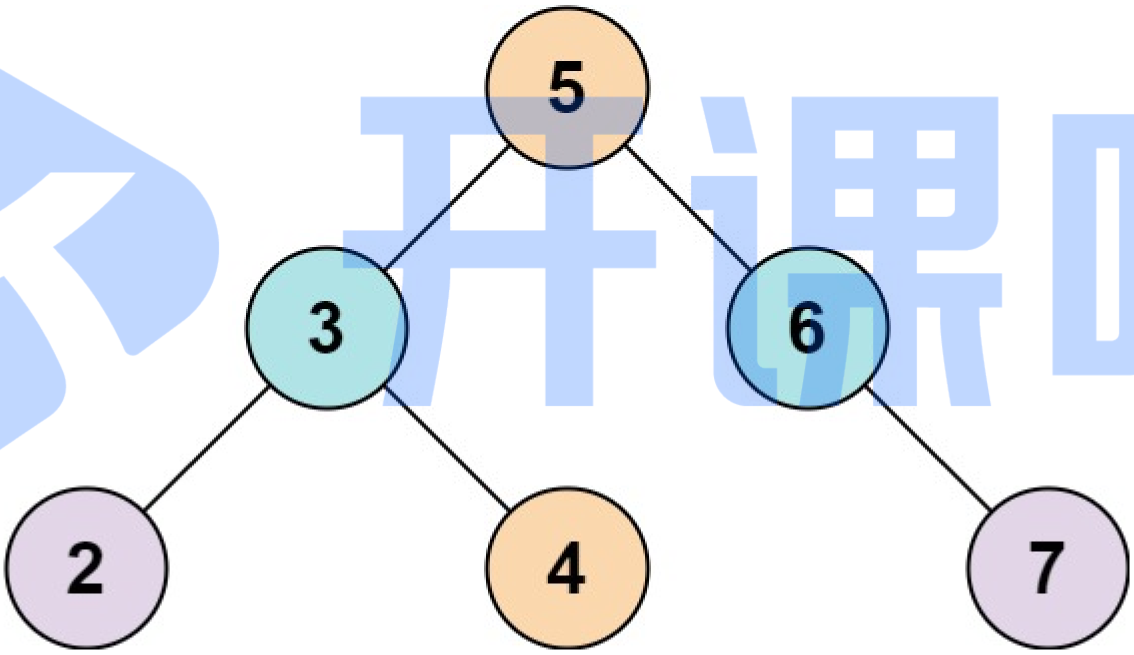
```
输入：root = [1,3,null,null,2]
输出：[3,1,null,null,2]
解释：3 不能是 1 左孩子，因为 3 > 1 。交换 1 和 3 使二叉搜索树有效。
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(ri
ght) {}
 * };
 */
class Solution {
public:
    TreeNode *pre, *p, *q;
    void inorder(TreeNode *root) {
        if (root == nullptr) return ;
        inorder(root->left);
        if (pre && root->val < pre->val) {
            if (p == nullptr) p = pre;
            q = root;
        }
        pre = root;
        inorder(root->right);
        return ;
    }
    void recoverTree(TreeNode* root) {
        pre = p = q = nullptr;
        inorder(root);
        swap(p->val, q->val);
        return ;
    }
};
```

653. 两数之和 IV - 输入 BST

给定一个二叉搜索树 `root` 和一个目标结果 `k`，如果 BST 中存在两个元素且它们的和等于给定的目标结果，则返回 `true`。

示例 1：



```
输入: root = [5,3,6,2,4,null,7], k = 9
输出: true
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(ri
ght) {}
 * };
 */
class Solution {
public:
    void inorder(TreeNode *root, vector<int> &ret) {
        if (root == nullptr) return ;
        inorder(root->left, ret);
        ret.push_back(root->val);
        inorder(root->right, ret);
        return ;
    }
    bool findTarget(TreeNode* root, int k) {
        vector<int> ret;
        inorder(root, ret);
        int p = 0, q = ret.size() - 1;
        while (p < q && ret[p] + ret[q] - k) {
            if (ret[p] + ret[q] < k) p += 1;
            else q -= 1;
        }
        return p < q;
    }
};
```

204. 计数质数

统计所有小于非负整数 n 的质数的数量。

示例 1：

```
输入：n = 10
输出：4
解释：小于 10 的质数一共有 4 个, 它们是 2, 3, 5, 7 。
```

```
class Solution {
public:
    int countPrimes(int n) {
        int *prime = new int[n + 1];
        for (int i = 0; i < n; i++) prime[i] = 0;
        for (int i = 2; i * i < n; i++) {
            if (prime[i]) continue;
            for (int j = 2 * i; j < n; j += i) {
                prime[j] = 1;
            }
        }
        int cnt = 0;
        for (int i = 2; i < n; i++) cnt += (prime[i] == 0);
        delete[] prime;
        return cnt;
    }
};
```

504. 七进制数

给定一个整数 `num`，将其转化为 **7 进制**，并以字符串形式输出。

示例 1:

```
输入: num = 100
输出: "202"
```

```
class Solution {
public:
    int countPrimes(int n) {
        int *prime = new int[n + 1];
        for (int i = 0; i < n; i++) prime[i] = 0;
        for (int i = 2; i * i < n; i++) {
            if (prime[i]) continue;
            for (int j = 2 * i; j < n; j += i) {
                prime[j] = 1;
            }
        }
        int cnt = 0;
        for (int i = 2; i < n; i++) cnt += (prime[i] == 0);
        delete[] prime;
        return cnt;
    }
};
```

461. 汉明距离

两个整数之间的 汉明距离 指的是这两个数字对应二进制位不同的位置的数目。

给你两个整数 `x` 和 `y`，计算并返回它们之间的汉明距离。

示例 1：

```
输入：x = 1, y = 4
输出：2
解释：
1   (0 0 0 1)
4   (0 1 0 0)
       ↑   ↑
上面的箭头指出了对应二进制位不同的位置。
```

```
class Solution {
public:
    int hammingDistance(int x, int y) {
        x ^= y;
        int cnt = 0;
        while (x) {
            x &= (x - 1);
            cnt += 1;
        }
        return cnt;
    }
};
```

528. 按权重随机选择

给定一个正整数数组 `w` ，其中 `w[i]` 代表下标 `i` 的权重（下标从 `0` 开始），请写一个函数 `pickIndex` ，它可以随机地获取下标 `i`，选取下标 `i` 的概率与 `w[i]` 成正比。

例如，对于 `w = [1, 3]`，挑选下标 `0` 的概率为 `1 / (1 + 3) = 0.25` （即，25%），而选取下标 `1` 的概率为 `3 / (1 + 3) = 0.75`（即，75%）。

也就是说，选取下标 `i` 的概率为 `w[i] / sum(w)` 。

示例 1：

```
输入：
["Solution","pickIndex"]
[[[1]],[]]
输出：
[null,0]
解释：
Solution solution = new Solution([1]);
solution.pickIndex(); // 返回 0，因为数组中只有一个元素，所以唯一的选择是返回下标 0。
```

```
class Solution {
public:
    int n;
    vector<int> sum;
    Solution(vector<int>& w) : sum(w) {
        for (int i = 1; i < sum.size(); i++) sum[i] += sum[i - 1];
        n = sum[sum.size() - 1];
    }

    int pickIndex() {
        int x = rand() % n;
        int head = 0, tail = sum.size() - 1;
        while (head < tail) {
            int mid = (head + tail) >> 1;
            if (sum[mid] <= x) head = mid + 1;
            else tail = mid;
        }
        return head;
    }
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution* obj = new Solution(w);
 * int param_1 = obj->pickIndex();
 */
```