1. https://leetcode.com/problems/same-tree/description/

```
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if(p == NULL && q == NULL) return true;
        if(p == NULL || q == NULL) return false;
        if(p->val != q->val) return false;

        bool l = isSameTree(p->left, q->left);
        bool r = isSameTree(p->right, q->right);

        return l && r;
    }
};
```

2. https://leetcode.com/problems/binary-tree-tilt/description/

```
class Solution {
public:
    int res = 0;

    int findSum(TreeNode *root){
        if(root == NULL) return 0;
        int l = findSum(root->left);
        int r = findSum(root->right);

        res += abs(l-r);
        return l + r + root->val;
    }
    int findTilt(TreeNode* root) {
        findSum(root);
        return res;
    }

};
```

3. https://leetcode.com/problems/leaf-similar-trees/description/

```
class Solution {
public:



    void fun(TreeNode *&r, vector<int> &v){

        if(r->left == NULL && r->right == NULL)
        {
            v.push_back(r->val);
        }
        if(r->left)fun(r->left, v);
        if(r->right)fun(r->right, v);
    }

    bool leafSimilar(TreeNode* root1, TreeNode* root2) {
        vector<int> v1;
        vector<int> v2;
        fun(root1, v1);
        fun(root2, v2);
        return v1 == v2;
    }
};
```

4. https://leetcode.com/problems/reverse-odd-levels-of-binary-tree/description/

Approach 1:

```
class Solution {
public:
    TreeNode* reverseOddLevels(TreeNode* root) {
        queue<TreeNode *> q;
        bool odd = false;
        q.push(root);

        while(!q.empty()){
            vector<TreeNode *> v;
            int sz = q.size();
```

```
            while(sz--){
                TreeNode *node = q.front(); q.pop();
                if(odd == true){
                    v.push_back(node);
                }
                if(node->left)
                {
                    q.push(node->left);
                }
                if(node->right) q.push(node->right);
            }
            if(v.size() > 0 && odd ){
                int i = 0;
                int j = v.size() - 1;
                while(i < j){
                    swap(v[i]->val, v[j]->val);
                    i++;
                    j--;
                }
            }
            odd = !odd;
        }
        return root;
    }
};
```

Approach 2:

```
class Solution {
public:
    void swapNode(TreeNode *p, TreeNode *q, int level){
        if(p == NULL || q == NULL) return;
        if(level % 2 == 1){
            swap(p->val, q->val);
        }
        swapNode(p->left, q->right, level+1);
        swapNode(p->right, q->left, level+1);
    }
    TreeNode* reverseOddLevels(TreeNode* root) {
        swapNode(root->left, root->right, 1);
        return root;
    }
};
```