二叉树与经典问题

开课吧

2021-03-31

二叉树的基础知识

基础知识部分代码

```cpp
typedef struct Node {
    int key;
    struct Node *lchild, *rchild;
} Node;

Node *getNewNode(int key) {
    Node *p = (Node *) malloc(sizeof(Node));
    p->key = key;
    p->lchild = p->rchild = nullptr;
    return p;
}

Node *random_insert(Node *root, int key) {
    if (!root) { return getNewNode(key); }

    if (rand() % 2) { root->lchild = random_insert(root->lchild,
    ↪  key); }
    else { root->rchild = random_insert(root->rchild, key); }

    return root;
}

void pre_order(Node *root) {
    if (!root) { return; }
    printf("%d ", root->key);
    pre_order(root->lchild);
    pre_order(root->rchild);
}

void in_order(Node *root) {
    if (!root) { return; }
    in_order(root->lchild);
    printf("%d ", root->key);
    in_order(root->rchild);
}

int main(int argc, char **argv) {
    if (argc != 2) { return 0; }

    int MAX_N = atoi(argv[1]);

    Node *root = nullptr;
    for (int i = 1; i <= MAX_N; ++i) {
        root = random_insert(root, i);
```

```
    }

    pre_order(root);
    printf("\n");
    in_order(root);
    printf("\n");

    return 0;
}
```

基本性质

1. 每个结点的度最多为 2。
2. 度为 0 的结点比度为 2 的结点多一个。

证明：设度为 0 的结点为 n_0，度为 1 的结点为 n_1，度为 2 的结点为 n_2。那么总结点数为 $n_0 + n_1 + n_2$，而总边数为 $0 \cdot n_0 + 1 \cdot n_1 + 2 \cdot n_2$。而我们知道总边数等于总结点数减去 1，那么有 $n_0 + n_1 + n_2 - 1 = 0 \cdot n_0 + 1 \cdot n_1 + 2 \cdot n_2$，即 $n_0 - 1 = n_2$。

遍历

根据根结点被访问的时机，分为前序遍历（根、左子树、右子树）、中序遍历（左子树、根、右子树）和后序遍历（左子树、右子树、根）。

特殊的二叉树

1. 完全二叉树 (complete binary tree)
2. 满二叉树 (full binary tree) – 指所有结点的度都是 0 或 2 的二叉树
3. 完美二叉树 (perfect binary tree)

注：几种二叉树的定义在不同的资料说明中可能存在一定差异，因此在实际场合中提到时请务必进行确认。

关于树结构的理解

结点表示集合 (set)，边表示关系 (relationship)。

学习二叉树的作用

二叉树是理解高级数据结构的基础。

1. 完全二叉树 – 堆、优先队列
2. 多叉树/森林 – 字典树、AC 自动机、并查集

3. 二叉排序树 (BST, Binary Search Tree) – AVL 树、2-3 树、红黑树、B-树、B+ 树

二叉树是练习递归技巧的最佳选择。

学习二叉树后，可以使用左孩子右兄弟法来节省空间。

二叉树的基本操作

LeetCode 144. 二叉树的前序遍历

点击查看题目

```cpp
class Solution {
public:
    void preorder(TreeNode *root, vector<int> &ans) {
        if (root == nullptr) { return; }

        ans.push_back(root->val);
        preorder(root->left, ans);
        preorder(root->right, ans);

        return;
    }

    vector<int> preorderTraversal(TreeNode *root) {
        vector<int> ans;
        preorder(root, ans);
        return ans;
    }
};
```

LeetCode 589. N 叉树的前序遍历

点击查看题目

```cpp
class Solution {
public:
    void __preorder(Node *root, vector<int> &ans) {
        if (root == nullptr) { return; }

        ans.push_back(root->val);
        for (auto x : root->children) {
            __preorder(x, ans);
        }
```

```
        return;
    }
    vector<int> preorder(Node *root) {
        vector<int> ans;
        __preorder(root, ans);
        return ans;
    }
};
```

LeetCode 226. 翻转二叉树

点击查看题目
交换左右子树，再递归翻转左右子树。

```
class Solution {
public:
    TreeNode *invertTree(TreeNode *root) {
        if (root == nullptr) { return root; }

        swap(root->left, root->right);
        invertTree(root->left);
        invertTree(root->right);

        return root;
    }
};
```

LeetCode 剑指 Offer 32 - II. 从上到下打印二叉树 II

点击查看题目
使用将行号作为参数的递归即可。也可以使用队列 BFS 来进行层序遍历。

```
class Solution {
public:
    void getResult(TreeNode *root, int k, vector<vector<int>>
    ↪  &ans) {
        if (root == nullptr) { return; }
        if (k == ans.size()) { ans.push_back(vector<int>()); }

        ans[k].push_back(root->val);
        getResult(root->left, k + 1, ans);
        getResult(root->right, k + 1, ans);

        return;
    }
```

```
    vector<vector<int>> levelOrder(TreeNode *root) {
        vector<vector<int>> ans;
        getResult(root, 0, ans);
        return ans;
    }
};
```

LeetCode 107. 二叉树的层序遍历 II

点击查看题目

```
class Solution {
public:
    void getResult(TreeNode *root, int k, vector<vector<int>>
    ↪  &ans) {
        if (root == nullptr) { return; }
        if (k == ans.size()) { ans.push_back(vector<int>()); }

        ans[k].push_back(root->val);
        getResult(root->left, k + 1, ans);
        getResult(root->right, k + 1, ans);

        return;
    }
    vector<vector<int>> levelOrderBottom(TreeNode *root) {
        vector<vector<int>> ans;
        getResult(root, 0, ans);
        for (int i = 0, j = ans.size() - 1; i < j; i++, j--) {
            swap(ans[i], ans[j]);
        }
        return ans;
    }
};
```

LeetCode 103. 二叉树的锯齿形层序遍历

点击查看题目

```
class Solution {
public:
    void getResult(TreeNode *root, int k, vector<vector<int>>
    ↪  &ans) {
        if (root == nullptr) { return; }
        if (k == ans.size()) { ans.push_back(vector<int>()); }
```

```
        ans[k].push_back(root->val);
        getResult(root->left, k + 1, ans);
        getResult(root->right, k + 1, ans);

        return;
    }

    vector<vector<int>> levelOrder(TreeNode *root) {
        vector<vector<int>> ans;
        getResult(root, 0, ans);
        return ans;
    }

    vector<vector<int>> zigzagLevelOrder(TreeNode *root) {
        vector<vector<int>> ans = levelOrder(root);
        for (int k = 1; k < ans.size(); k += 2) {
            for (int i = 0, j = ans[k].size() - 1; i < j; i++,
            ↪  j--) {
                swap(ans[k][i], ans[k][j]);
            }
        }
        return ans;
    }
};
```

二叉树的进阶操作

LeetCode 110. 平衡二叉树

点击查看题目

对获取树高的函数进行修改，在获取树高的函数中对平衡进行判断即可。

```
class Solution {
public:
    int getHeight(TreeNode *root) {
        if (root == nullptr) { return 0; }
        int l = getHeight(root->left);
        int r = getHeight(root->right);
        if (l < 0 || r < 0) { return -2; }
        if (abs(l - r) > 1) { return -2; }
        return max(l, r) + 1;
    }

    bool isBalanced(TreeNode *root) {
        return getHeight(root) >= 0;
```

```
    }
};
```

LeetCode 112. 路径总和

点击查看题目

递归向下求值，每次减去当前结点的值，递归结束的条件是遇到叶子结点且刚好求得的值为 0。

```
class Solution {
public:
    bool hasPathSum(TreeNode *root, int targetSum) {
        if (root == nullptr) { return false; }
        if (root->left == nullptr && root->right == nullptr) {
        ↪  return targetSum == root->val; }
        targetSum -= root->val;
        return hasPathSum(root->left, targetSum) ||
        ↪  hasPathSum(root->right, targetSum);
    }
};
```

LeetCode 105. 从前序与中序遍历序列构造二叉树

点击查看题目

递归拆分。前序遍历的第一个结点是根结点，在中序遍历中找到该根结点的位置，区分出左右子树，再递归向下拆分左右子树即可。

```
class Solution {
public:
    unordered_map<int, int> ind;

    TreeNode *__buildTree(vector<int> &preorder, int l1, int r1,
    ↪  vector<int> &inorder, int l2, int r2) {
        if (l1 == r1) { return nullptr; }
        TreeNode *root = new TreeNode(preorder[l1]);
        int pos = ind[preorder[l1]], n = pos - l2 + 1;
        root->left = __buildTree(preorder, l1 + 1, l1 + n,
        ↪  inorder, l2, pos);
        root->right = __buildTree(preorder, l1 + n, r1, inorder,
        ↪  l2 + n, r2);
        return root;
    }

    TreeNode *buildTree(vector<int> &preorder, vector<int>
    ↪  &inorder) {
        for (int i = 0; i < inorder.size(); i++) {
```

```
            ind[inorder[i]] = i;
        }
        return __buildTree(preorder, 0, preorder.size(), inorder,
        ↪  0, inorder.size());
    }
};
```

LeetCode 222. 完全二叉树的结点个数

点击查看题目

递归数左子树和右子树的结点，加上根结点即可。

```
class Solution {
public:
    int countNodes(TreeNode *root) {
        if (root == nullptr) { return 0; }
        return countNodes(root->left) + countNodes(root->right) +
        ↪  1;
    }
};
```

LeetCode 剑指 Offer 54. 二叉搜索树的第 k 大结点

点击查看题目

统计右子树的结点个数 cnt_r，递归处理即可。如果 $k = cnt_r + 1$，那么是根结点，如果 $k \leq cnt_r$，那么在右子树中且是第 k 大，否则在左子树中且是第 $k - cnt_r - 1$ 大。

```
class Solution {
public:
    int countNodes(TreeNode *root) {
        if (root == nullptr) { return 0; }
        return countNodes(root->left) + countNodes(root->right) +
        ↪  1;
    }

    int kthLargest(TreeNode *root, int k) {
        int n = countNodes(root->right);
        if (n >= k) { return kthLargest(root->right, k); }
        if (n == k - 1) { return root->val; }
        return kthLargest(root->left, k - n - 1);
    }
};
```

LeetCode 剑指 Offer 26. 树的子结构

点击查看题目
先和根结点比较，再递归地和左右子树比较即可。

```cpp
class Solution {
public:
    bool isMatch(TreeNode *A, TreeNode *B) {
        if (B == nullptr) { return true; }
        if (A == nullptr) { return false; }
        return A->val == B->val && isMatch(A->left, B->left) &&
        ↪  isMatch(A->right, B->right);
    }

    bool isSubStructure(TreeNode *A, TreeNode *B) {
        if (A == nullptr || B == nullptr) { return false; }
        if (A->val == B->val && isMatch(A, B)) { return true; }
        return isSubStructure(A->left, B) ||
        ↪  isSubStructure(A->right, B);
    }
};
```

LeetCode 968. 监控二叉树

点击查看题目
dp 的第一维表示“父结点”是否放置摄像头，第二维表示“当前结点”是否放置摄像头。例如 dp[0][0] 表示父结点不放置摄像头，当前结点也不放置摄像头的情况下，覆盖整棵树所需要的最少摄像头数。

对放置情况进行分情况讨论。以 dp[0][0] 为例，有：

```cpp
dp[0][0] = min(
                left[0][1] + right[0][0],
                left[0][0] + right[0][1],
                left[0][1] + right[0][1]
            )
```

这表示，父节点和当前结点都不放摄像头的情况下，所需要的最少摄像头数，需要取以下三种情况的最小值：
(1) 左子树的父节点不放置摄像头 + 左子树自身放置摄像头 + 右子树的父节点不放置摄像头 + 右子树自身不放置摄像头
(2) 左子树的父节点不放置摄像头 + 左子树自身不放置摄像头 + 右子树的父节点不放置摄像头 + 右子树自身放置摄像头
(3) 左子树的父节点不放置摄像头 + 左子树自身放置摄像头 + 右子树的父节点不放置摄像头 + 右子树自身放置摄像头

对所有情况进行讨论，具体逻辑可以参考随堂代码。

```cpp
class Solution {
public:
    void getDP(TreeNode *root, int dp[2][2]) {
        if (root == nullptr) {
            dp[0][0] = 0;
            dp[0][1] = 10000;
            dp[1][0] = 0;
            dp[1][1] = 10000;
            return;
        }
        if (root->left == nullptr && root->right == nullptr) {
            dp[0][0] = 10000;
            dp[0][1] = 1;
            dp[1][0] = 0;
            dp[1][1] = 1;
            return;
        }
        int l[2][2], r[2][2];
        getDP(root->left, l);
        getDP(root->right, r);
        dp[0][0] = min(min(l[0][0] + r[0][1], l[0][1] + r[0][0]),
        ↪  l[0][1] + r[0][1]);
        dp[0][1] = min(min(l[1][0] + r[1][0], l[1][0] + r[1][1]),
        ↪  min(l[1][1] + r[1][0], l[1][1] + r[1][1])) + 1;
        dp[1][0] = min(dp[0][0], l[0][0] + r[0][0]);
        dp[1][1] = dp[0][1];
        return;
    }

    int minCameraCover(TreeNode *root) {
        int dp[2][2];
        getDP(root, dp);
        return min(dp[0][0], dp[0][1]);
    }
};
```

LeetCode 662. 二叉树最大宽度

点击查看题目

按照完美二叉树的形式，对所有的结点进行编号。层序遍历后同一层内编号最大结点和最小编号结点的编号之差加一即为所求。

```cpp
class Solution {
public:
    typedef pair<TreeNode *, int> PNI;

    int widthOfBinaryTree(TreeNode *root) {
```

```
        queue<PNI> q;
        q.push(PNI(root, 0));
        int ans = 0;
        while (!q.empty()) {
            int cnt = q.size();
            int l = q.front().second, r;
            for (int i = 0; i < cnt; i++) {
                PNI &temp = q.front();
                r = temp.second - l;
                if (temp.first->left) {
                ↪  q.push(PNI(temp.first->left, r * 2)); }
                if (temp.first->right) {
                ↪  q.push(PNI(temp.first->right, r * 2 + 1)); }
                q.pop();
            }
            ans = max(ans, r + 1);
        }

        return ans;
    }
};
```