图论面试题精讲

七月算法　**曹鹏**

2015年4月25日

提纲

- 图论简介
- 面试题总体分析
- 一些例题
  - 例1 给定二叉树前中序遍历，构造二叉树
  - 例2 二叉树高度、最小深度、二叉搜索树判断、对称判断、平衡判断
  - 例3 二叉树与链表转换
  - 例4 无向图复制
  - 例5 直角路线遍历棋盘
- 总结

图论简介

- 图结构
  - 节点
  - 边
- 分类
  - 有向图
  - 无向图
- 特殊的图
  - 二叉树： 二叉搜索树
  - 普通树 （并查集）
  - 堆

面试题总体分析

- 图
  - 连通性（割点、边）
  - 最小生成树
  - 最短路
  - 搜索(BFS,DFS)
  - 欧拉回路
  - 哈密尔顿回路
  - 拓扑排序
- 树
  - 树的定义与判断
  - 平衡、二查搜索树、最大（小）高度、最近公共祖先

例1 遍历序列

- 给定二叉树前、中序遍历，构造二叉树（Leetcode 105)
  - 分析：
    - 前序遍历序列第一个是根节点x
    - 从中序遍历序列中找到根节点x
    - 中序遍历中x的左边序列对应等长的前序遍历序列
      - 左子树
    - 中序遍历中x的右边序列对应等长的前序遍历序列
      - 右子树

例1 续

- 思考题 Leetcode 106

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode *help(vector<int> &preorder, vector<int> &inorder, int fromp, int fromi, int length) {
        if (length == 0) {
            return 0;
        }
        TreeNode *root = new TreeNode(preorder[fromp]);
        int i;
        for (i = fromi; inorder[i] != preorder[fromp]; ++i)
        ;
        root->left = help(preorder, inorder, fromp + 1, fromi, i - fromi);
        root->right = help(preorder, inorder, fromp + 1 + i - fromi , i + 1, length - 1 - i + fromi);
        return root;
    }
    TreeNode *buildTree(vector<int> &preorder, vector<int> &inorder) {
        // IMPORTANT: Please reset any member data you declared, as
        // the same Solution instance will be reused for each test case.
        return help(preorder, inorder, 0, 0, preorder.size());

    }
};
```

例2 二叉树相关问题——递归

- 一般思路
  - 递归：根节点、左子树、右子树（前、中、后续遍历）
- (Leetcode 124) 二叉树每个节点有一个整数，返回和最大的路径。
  - 左子树延伸下去的路径
  - 右子树延伸下去的路径
  - 通过根节点的路径
  - 注意返回值和最大值的关系

例2 续1

```
class Solution {
public:
    int help(TreeNode *root, int &m) {
        if (root == 0) {
            return 0;
        }
        int left = help(root->left, m);
        int right = help(root->right, m);
        int ret = max(max(left, right), 0) + root->val;
        m = max(max(m, ret), left + right + root->val);
        return ret;


    }
    int maxPathSum(TreeNode *root) {
        // IMPORTANT: Please reset any member data you declared, as
        // the same Solution instance will be reused for each test case.
        if (root == 0) {
            return 0;
        }
        int result = root->val;
        help(root, result);
        return result;

    }
};
```

例2 续2

- 二叉树最小深度 (Leetcode 111) 注意空子树

```
class Solution {
public:
    int minDepth(TreeNode *root) {
        // IMPORTANT: Please reset any member data you declared, as
        // the same Solution instance will be reused for each test case.
        if (root == 0) {
            return 0;
        }
        if (root->left) {
            if (root->right) {
                return min(minDepth(root->left), minDepth(root->right)) + 1;
            }
            else {
                return minDepth(root->left) + 1;
            }
        }
        else if (root->right) {
            return minDepth(root->right) + 1;

        }
        else {
            return 1;
        }

    }
};
```

例2 续3

- 判断平衡 (Leetcode 110)

```
class Solution {
public:
    bool help(TreeNode *root,int &height) {
        if (root == 0) {
            height = 0;
            return true;
        }
        int height1, height2;
        if (!help(root->left, height1)) {
            return false;
        }
        if (!help(root->right, height2)) {
            return false;
        }
        height = max(height1, height2) + 1;
        return (height1 >= height2 - 1) && (height1 <= height2 + 1);



    }
    bool isBalanced(TreeNode *root) {
        // IMPORTANT: Please reset any member data you declared, as
        // the same Solution instance will be reused for each test case.
        int height;
        return help(root,height);

    }
```

例2 续4

- 最大深度（Leetcode 104) 注意和最小深度不同

```
class Solution {
public:
    int maxDepth(TreeNode *root) {
        // IMPORTANT: Please reset any member data you declared, as
        // the same Solution instance will be reused for each test case.
        return root?(max(maxDepth(root->left), maxDepth(root->right)) + 1):0;

    }
};
```

- 判断相同 (Leetcode 100)

```
class Solution {
public:
    bool isSameTree(TreeNode *p, TreeNode *q) {
        // IMPORTANT: Please reset any member data you declared, as
        // the same Solution instance will be reused for each test case.
        if (p == 0) {
            return q == 0;
        }
        if (q == 0) {
            return false;
        }
        return (p->val == q->val) && isSameTree(p->left, q->left) && isSameTree(p->right, q->right);

    }
};
```

例2 续5

- 判断对称（Leetcode 101)

```
class Solution {
public:
    bool help(TreeNode *root1, TreeNode *root2) {
        if (root1 == 0) {
            return root2 == 0;
        }
        if (root2 == 0) {
            return false;
        }
        return (root1->val == root2->val) && help(root1->left, root2->right) && help(root1->right,root2->left);
    }
    bool isSymmetric(TreeNode *root) {
        // IMPORTANT: Please reset any member data you declared, as
        // the same Solution instance will be reused for each test case.
        if (root == 0) {
            return true;
        }
        return help(root->left, root->right);

    }
};
```

例2 续6

- 判断二查搜索树 (Leetcode 98)

```
class Solution {
public:
    bool help(TreeNode *root,bool &first, int &last) {
        if (root == 0) {
            return true;
        }
        if (!help(root->left, first, last)) {
            return false;
        }
        if (first) {
            first = false;
            last = root->val;
        }
        else if (last >= root->val) {
            return false;
        }
        last = root->val;
        return help(root->right, first, last);
    }
    bool isValidBST(TreeNode *root) {
        // Note: The Solution object is instantiated only once and is reused by each test case.
        bool mark = true;
        int val = 0;
        return help(root, mark, val);

    }
};
```

例3 二叉树与链表

- 二叉树转链表 (Leetcode 114)

```
class Solution {
public:
    void help(TreeNode * &root, TreeNode *&last) {
        last = root;
        if (root == 0) {
            return;
        }
        TreeNode *maylast, *temp = root->right;
        help(root->left, maylast);
        if (maylast) {
            last = maylast;
            last->right = temp;
            root->right = root->left;
            root->left = 0;
        }
        help(temp, maylast);
        if (maylast) {
            last = maylast;
        }
    }
    void flatten(TreeNode *root) {
        // IMPORTANT: Please reset any member data you declared, as
        // the same Solution instance will be reused for each test case.
        TreeNode *last;
        help(root, last);

    }
};
```

例3 续1

- 链表转(平衡)二叉树

(Leetcode 109)

  - 方法1 O(nlogn)

因为链表不能随机访问

```
class Solution {
public:
    TreeNode *help(ListNode *head, int length) {
        if (length == 0) {
            return 0;
        }
        ListNode *now = head;
        for (int i = (length - 1) >> 1; i; --i) {
            now = now->next;
        }
        TreeNode *root = new TreeNode(now->val);
        root->left = help(head, (length - 1) >> 1);
        root->right = help(now->next, length >> 1);
        return root;

    }
    TreeNode *sortedListToBST(ListNode *head) {
        // IMPORTANT: Please reset any member data you declared, as
        // the same Solution instance will be reused for each test case.
       ListNode *temp = head;
       int length;
       for (length = 0; temp; temp = temp->next, ++length)
       ;
       return help(head, length);


    }
};
```

例3 续2

- 方法2: 优化

同时移动指针 O(n)

- 思考题

有序数组转(平衡)二叉树

Leetcode108

```
class Solution {
public:
    TreeNode *help(ListNode *&head, int length) {
        if (length == 0) {
            return 0;
        }
        int num = (length - 1) >> 1;
        TreeNode *left = help(head, num);
        TreeNode *root = new TreeNode(head->val);
        root->left = left;
        head = head->next;
        root->right = help(head, length - num - 1);
        return root;

    }
    TreeNode *sortedListToBST(ListNode *head) {
        // IMPORTANT: Please reset any member data you declared, as
        // the same Solution instance will be reused for each test case.
       ListNode *temp = head;
       int length;
       for (length = 0; temp; temp = temp->next, ++length)
       ;
       return help(head, length);
    }
};
```

例4 无向图复制

- 复制一个有向图（临接表存储）(Leetcode 133)
  - 分析：DFS.
  - 图可能有圈

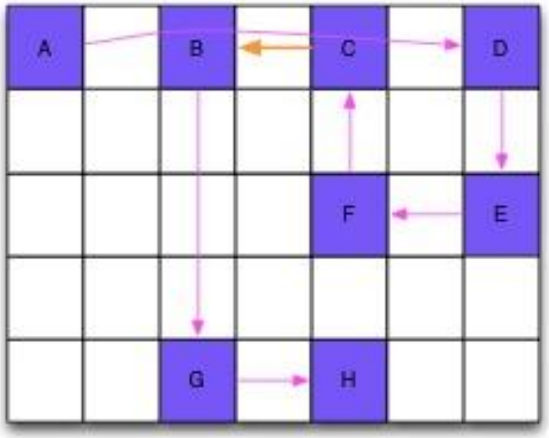
```
class Solution {
public:
    UndirectedGraphNode * dfs(const UndirectedGraphNode *node,map<int,UndirectedGraphNode *> &have) {
        map<int,UndirectedGraphNode *>::iterator t = have.find(node->label);
        if (t == have.end()) {
            UndirectedGraphNode *newnode = new UndirectedGraphNode(node->label);
            have.insert(make_pair(node->label, newnode));
            for (int i = 0; i < node->neighbors.size(); ++i) {
                newnode->neighbors.push_back(dfs(node->neighbors[i], have));
            }
            return newnode;
        }
        else {
            return t->second;
        }
    }
    UndirectedGraphNode *cloneGraph(UndirectedGraphNode *node) {
        // IMPORTANT: Please reset any member data you declared, as
        // the same Solution instance will be reused for each test case.
        map<int, UndirectedGraphNode *> have;
        if (node == 0) {
            return 0;
        }
        return dfs(node,have);


    }
};
```

例5 直角遍历棋盘

- 给定矩形棋盘，再给你若干个位置(x,y)，你可以从任何给定的位置出发，只能在给定位置之间移动。每次移动只能是沿着水平和竖直方向走，并且这次走的方向和上次不同(交错方向)，每个位置只能经过一次，是否可行？(直角遍历棋盘)
- 例 右图ADEFCBGH是可行的！
- 分析：每个点只经过一次，

这是哈密尔顿路？

- “直角”的特殊性
  - 拆点:把所有x放到一起X，所有y放到一
  - 原先点(x,y)，我们连一条无向边(x,y)
  - 我们走的路一定是从集合X到Y交错
  - 原先有一条路对对应于走遍这个（二分）图所有的边！
  - 这是一个欧拉路判断的问题！



例5 续

- 思考题(密码锁问题)
- 一个密码锁，密码是

4位数字，操作是

(1) 扔掉高位数字

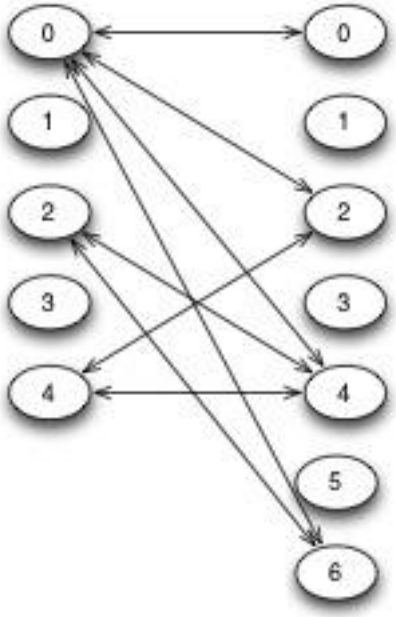
(2) 把低位数字移动到高位

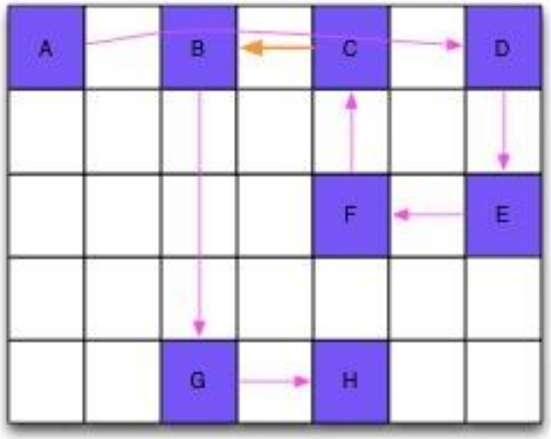
(3) 添加任意低位数字

即abcd变为bcde,问从

任意数字开始，是否可以经过

0000-9999仅一次？





提示:

节点:3位数字“abc”

边: 后两位等于前两位 abc->bcd 相当于一条边 代表abcd的组合

总结

- 理解递归
- 熟悉树的遍历(递归、非递归)
- 其他问题
  - 最近公共祖先
    - 二叉树
    - 非二叉树
    - 二查搜索树
    - 离线算法－在线算法
  - (隐式）图搜索 （bfs/dfs)——（强）连通分量
    - 自己建图
  - 拓扑排序

谢谢大家

- 更多算法视频尽在：
  - http://www.julyedu.com/
    - 免费视频
    - 直播课程
    - 面试问答
- Contact us：微博
  - @七月算法
  - @七月问答
  - @曹鹏博士