题目地址（987. 二叉树的垂序遍历）

https://leetcode-cn.com/problems/vertical-order-traversal-of-a-binary-tree

标签

- 哈希表
- 树
- 排序

难度

- 中等

入选理由

1. 很有意思的一个题目，一个很特殊的遍历方式。但不管怎么奇葩的遍历，我们都可以用基础的遍历方式来解决

题目描述

给定二叉树，按垂序遍历返回其结点值。

对位于 (X, Y) 的每个结点而言，其左右子结点分别位于 (X-1, Y-1) 和 (X+1, Y-1)。

把一条垂线从 X = -infinity 移动到 X = +infinity ，每当该垂线与结点接触时，我们按从上到下的顺序报告结点的值（Y 坐标递减）。

如果两个结点位置相同，则首先报告的结点值较小。

按 X 坐标顺序返回非空报告的列表。每个报告都有一个结点值列表。

示例 1：

```
输入：[3,9,20,null,null,15,7]
输出：[[9],[3,15],[20],[7]]
解释：
在不丧失其普遍性的情况下，我们可以假设根结点位于 (0, 0)：
然后，值为 9 的结点出现在 (-1, -1)；
```

值为 3 和 15 的两个结点分别出现在 (0, 0) 和 (0, -2)；
值为 20 的结点出现在 (1, -1)；
值为 7 的结点出现在 (2, -2)。
示例 2：

输入：[1,2,3,4,5,6,7]
输出：[[4],[2],[1,5,6],[3],[7]]
解释：
根据给定的方案，值为 5 和 6 的两个结点出现在同一位置。
然而，在报告 "[1,5,6]" 中，结点值 5 排在前面，因为 5 小于 6。

提示：

树的结点数介于 1 和 1000 之间。
每个结点值介于 0 和 1000 之间。

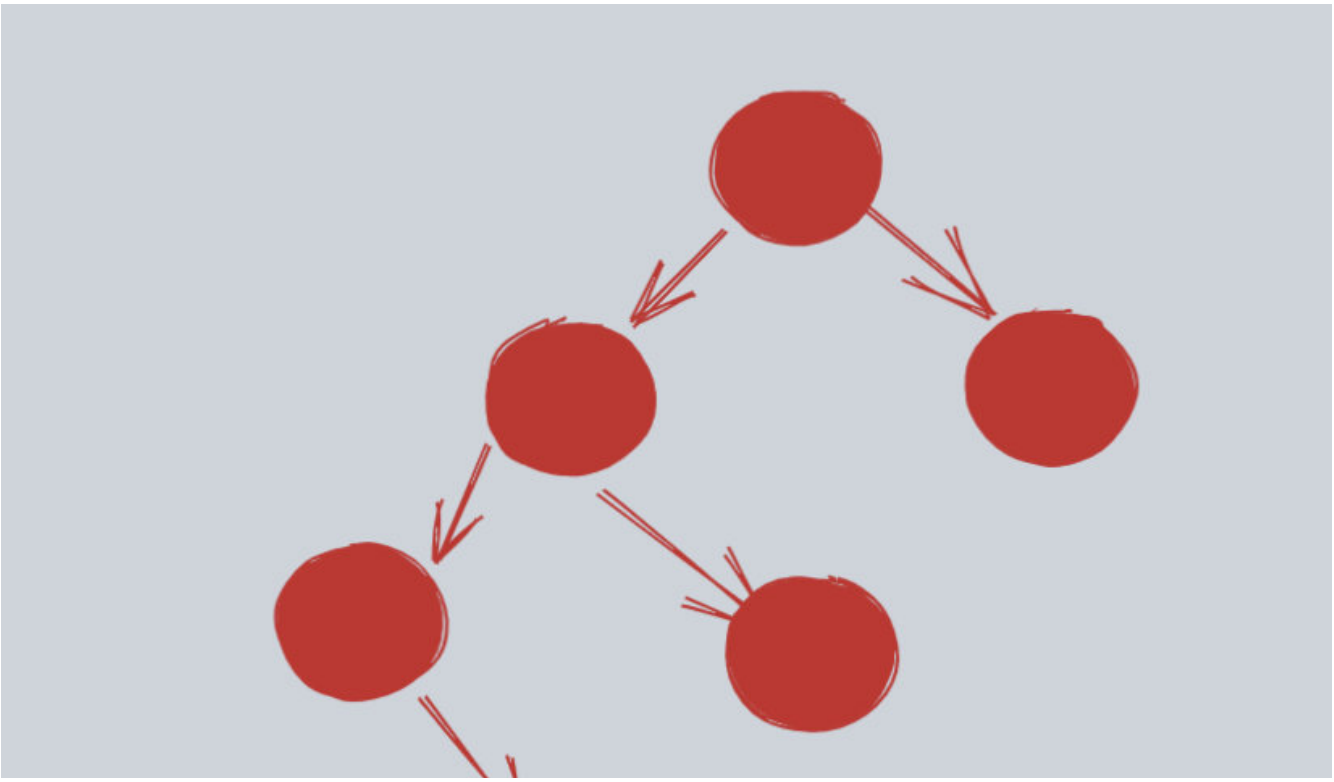
前置知识

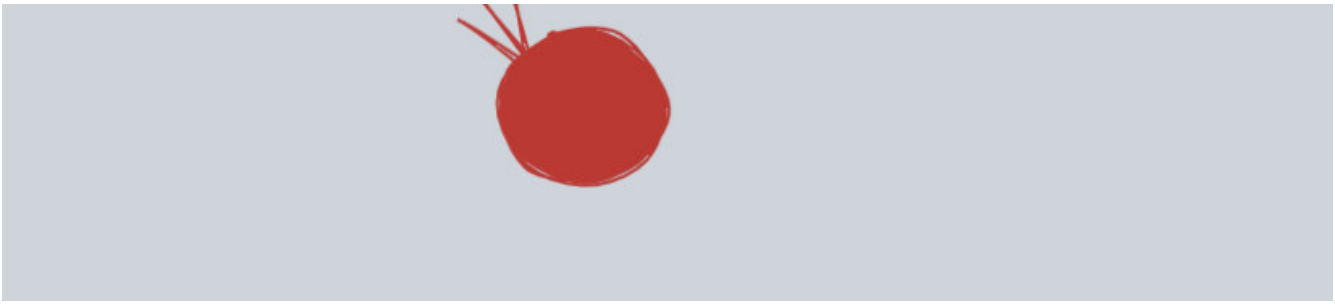
- DFS
- 排序

思路

经过前面几天的学习，希望大家对 DFS 和 BFS 已经有了一定的了解了。

我们先来简化一下问题。假如题目没有 从上到下的顺序报告结点的值（Y 坐标递减） ，甚至也没有 如果两个结点位置相同，则首先报告的结点值较小。 的限制。是不是就比较简单了？





如上图，我们只需要进行一次搜索，不妨使用 DFS（没有特殊理由，我一般都是 DFS），将节点存储到一个哈希表中，其中 key 为节点的 x 值，value 为横坐标为 x 的节点值列表（不妨用数组表示）。形如：

```
{
    1: [1,3,4]
    -1: [5]
}
```

> 数据是瞎编的，不和题目例子有关联

经过上面的处理， 这个时候只需要对哈希表中的数据进行一次排序输出即可。

ok，如果这个你懂了，我们尝试加上面的两个限制加上去。

1. 从上到下的顺序报告结点的值（Y 坐标递减）

2. 如果两个结点位置相同，则首先报告的结点值较小。

关于第一个限制。其实我们可以再哈希表中再额外增加一层来解决。形如：

```
{
    1: {
        -2,[1,3,4]
        -3,[5]

    },
    -1: {
        -3: [6]
    }
}
```

这样我们除了对 x 排序，再对里层的 y 排序即可。

再来看第二个限制。其实看到上面的哈希表结构就比较清晰了，我们再对值排序即可。

总的来说，我们需要进行三次排序，分别是对 x 坐标，y 坐标 和 值。

那么时间复杂度是多少呢？我们来分析一下：

- 哈希表最外层的 key 总个数是最大是树的宽度。
- 哈希表第二层的 key 总个数是树的高度。
- 哈希表值的总长度是树的节点数。

也就是说哈希表的总容量和树的总的节点数是同阶的。因此空间复杂度为 $O(N)$， 排序的复杂度大致为 $NlogN$，其中 N 为树的节点总数。

代码

代码支持：Python，JS，CPP

Python Code:

```
class Solution(object):
    def verticalTraversal(self, root):
        seen = collections.defaultdict(
            lambda: collections.defaultdict(list))

        def dfs(root, x=0, y=0):
            if not root:
                return
            seen[x][y].append(root.val)
            dfs(root.left, x-1, y+1)
            dfs(root.right, x+1, y+1)

        dfs(root)
        ans = []
        # x 排序、
        for x in sorted(seen):
            level = []
            # y 排序
            for y in sorted(seen[x]):
                # 值排序
                level += sorted(v for v in seen[x][y])
            ans.append(level)

        return ans
```

JS Code(by @suukii)：

```
/**
 * Definition for a binary tree node.
```

```
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number[][]}
 */
var verticalTraversal = function (root) {
  if (!root) return [];

  // 坐标集合以 x 坐标分组
  const pos = {};
  // dfs 遍历节点并记录每个节点的坐标
  dfs(root, 0, 0);

  // 得到所有节点坐标后，先按 x 坐标升序排序
  let sorted = Object.keys(pos)
    .sort((a, b) => +a - +b)
    .map((key) => pos[key]);

  // 再给 x 坐标相同的每组节点坐标分别排序
  sorted = sorted.map((g) => {
    g.sort((a, b) => {
      // y 坐标相同的，按节点值升序排
      if (a[0] === b[0]) return a[1] - b[1];
      // 否则，按 y 坐标降序排
      else return b[0] - a[0];
    });
    // 把 y 坐标去掉，返回节点值
    return g.map((el) => el[1]);
  });

  return sorted;

  // ********************************
  function dfs(root, x, y) {
    if (!root) return;

    x in pos || (pos[x] = []);
    // 保存坐标数据，格式是：[y, val]
    pos[x].push([y, root.val]);

    dfs(root.left, x - 1, y - 1);
    dfs(root.right, x + 1, y - 1);
  }
};
```

CPP(by @Francis-xsc):

```
class Solution {
public:
    struct node
    {
        int val;
        int x;
        int y;
        node(int v,int X,int Y):val(v),x(X),y(Y){};
    };
    static bool cmp(node a,node b)
    {
        if(a.x^b.x)
            return a.x<b.x;
        if(a.y^b.y)
            return a.y<b.y;
        return a.val<b.val;
    }
    vector<node> a;
    int minx=1000,maxx=-1000;
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        dfs(root,0,0);
        sort(a.begin(),a.end(),cmp);
        vector<vector<int>>ans(maxx-minx+1);
        for(auto xx:a)
        {
            ans[xx.x-minx].push_back(xx.val);
        }
        return ans;
    }
    void dfs(TreeNode* root,int x,int y)
    {
        if(root==nullptr)
            return;
        if(x<minx)
            minx=x;
        if(x>maxx)
            maxx=x;
        a.push_back(node(root->val,x,y));
        dfs(root->left,x-1,y+1);
        dfs(root->right,x+1,y+1);
    }
};
```

复杂度分析

- 时间复杂度：$O(N\log N)$，其中 N 为树的节点总数。

- 空间复杂度：$O(N)$，其中 N 为树的节点总数。

上一页 下一页