JavaClub 第六周集体学习课程讲义

课程内容: 树(图)DFS，BFS

DFS

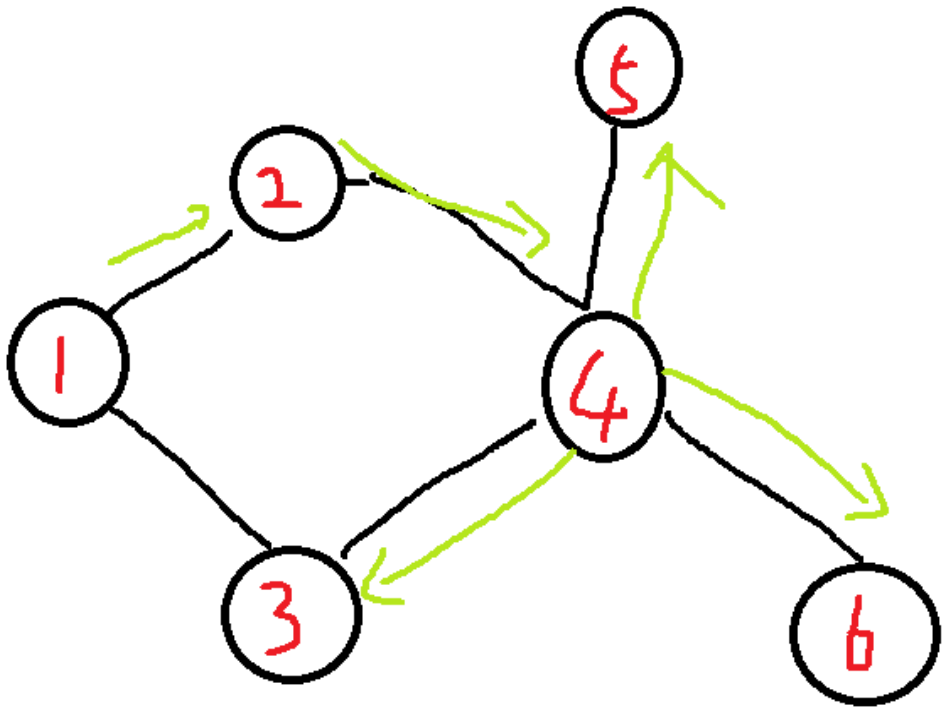
深度优先搜索算法（Depth First Search，简称DFS）：一种用于遍历或搜索树或图的算法。 沿着树的深度遍历树的节点，尽可能深的搜索树的分支。当节点v的所在边都已被探寻过或者在搜寻时结点不满足条件，搜索将回溯到发现节点v的那条边的起始节点。整个进程反复进行直到所有节点都被访问为止。

BFS

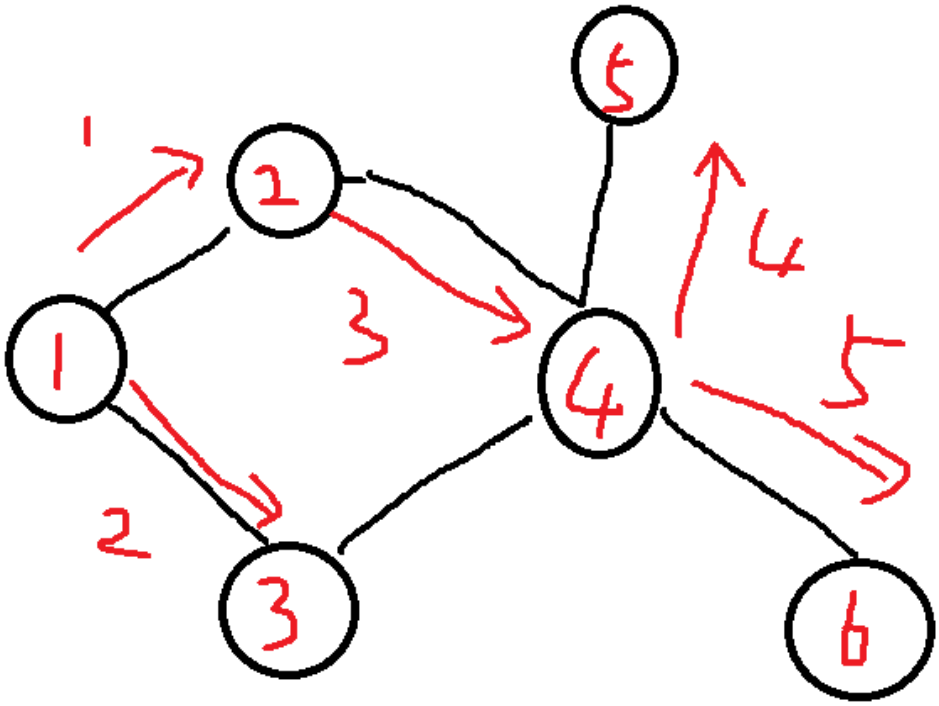
广度优先搜索算法（Breadth-First Search，缩写为 BFS），又称为宽度优先搜索，是一种图形搜索算法。简单的说，BFS 是从根结点开始，沿着树的宽度遍历树的结点。如果所有结点均被访问，则算法中止。

图的DFS BFS两种遍历方法

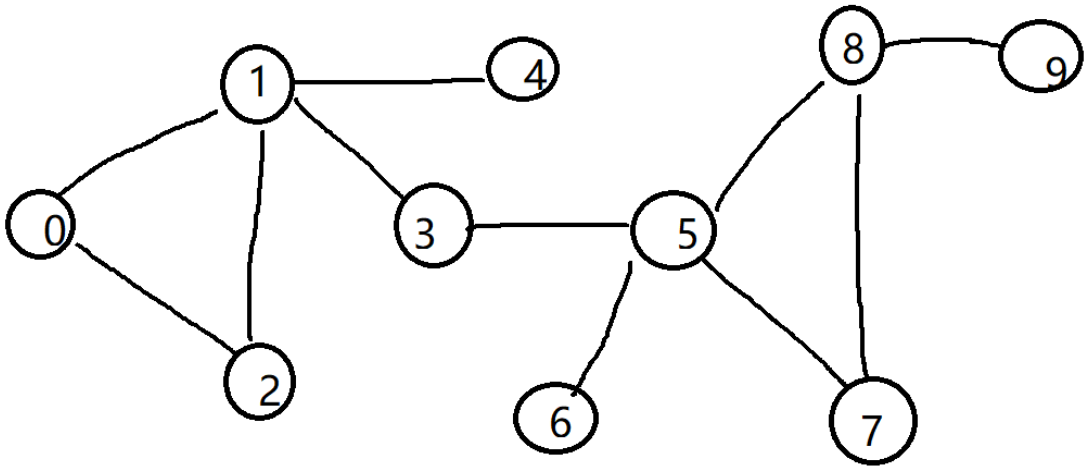
DFS： 1 2 4 5 6 3



BFS： 1 2 3 4 5 6



课堂练习：



DFS：0123456789

BFS：0123456789

问题：如果存在无法到达的边怎么遍历？

比如两个连通片段

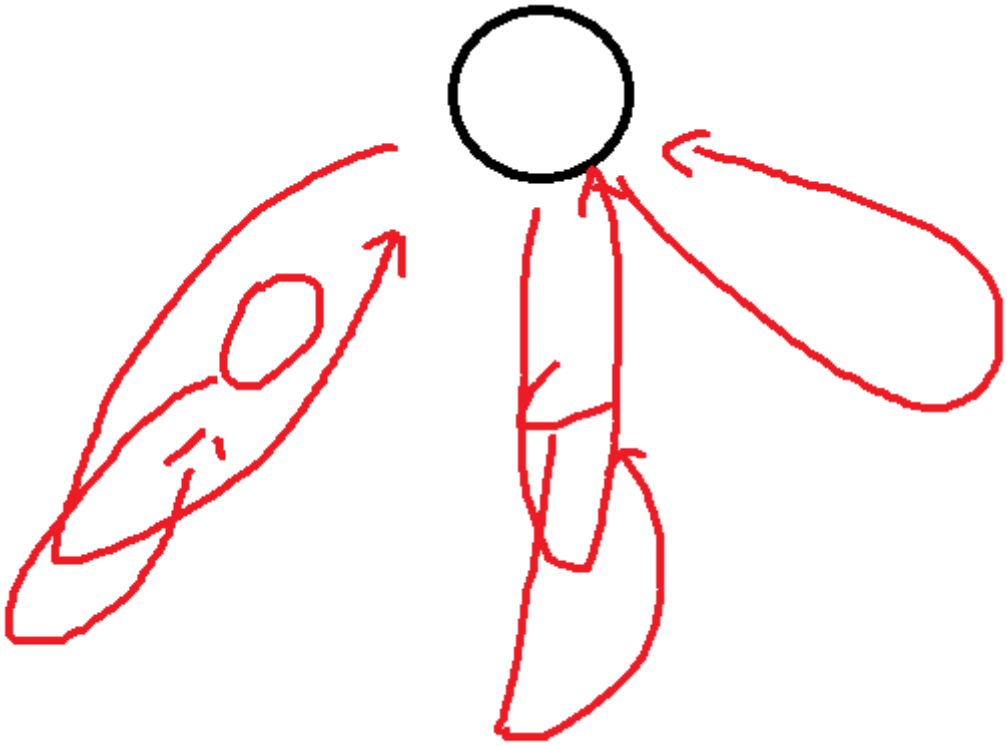
A——B——C　　D---E

将所有点装入数组，dfs后对**未访问**的再dfs

外层再套一次循环

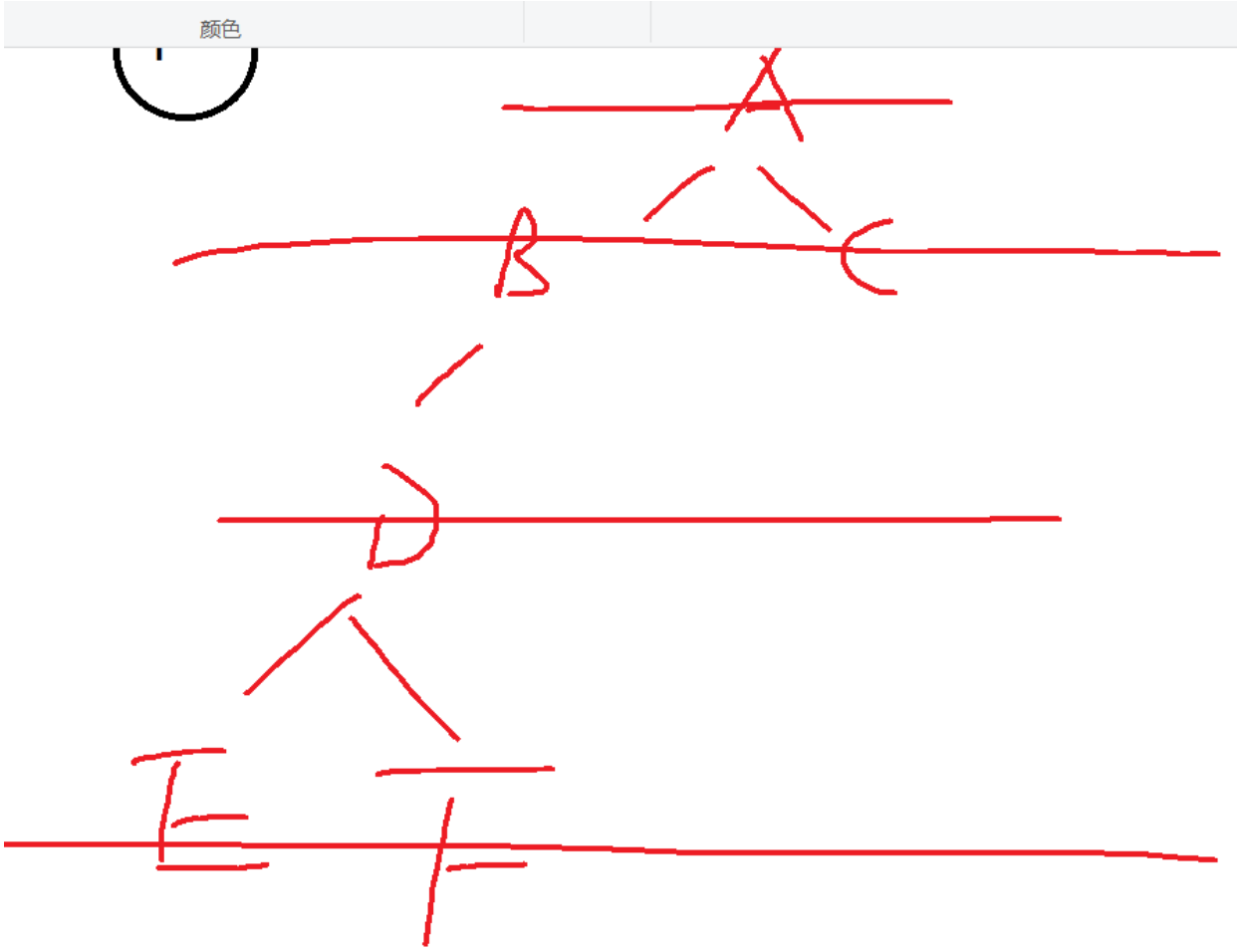
通过上面的讲解啊，我们可以看出图的遍历其实是这样的

DFS：



树的中后序遍历相似

BFS:



树的层次遍历相似

下面我们就这两个特点来讲一下树的遍历

树的遍历

中序遍历（DFS）

中序遍历：

```
i、中序遍历左子树；
ii、访问根结点；
iii、中序遍历右子树
```

94. 二叉树的中序遍历

```java
public List<Integer> inorderTraversal(TreeNode root) {
    List<Integer> list = new ArrayList<>();
    addNode(root,list);
    return list;
}

private void addNode(TreeNode root, List<Integer> list) {
    if(root==null) return;
    //首先遍历到最左边 那么怎么停止呢
    addNode(root.left,list);
    list.add(root.val);
    addNode(root.right,list);
}
```

顺便问一下，为什么这样写?

List list = new ArrayList<>();

这样写的原因是使用了泛型和多态的概念，以提高代码的灵活性和可维护性。

多态：Java中的许多对象（一般都是具有父子类关系的父类对象）在运行时都会出现两种类型：编译时类型和运行时类型，例如：Person person = new Student();这行代码将会生成一个person变量，该变量的编译时类型是Person，运行时类型是Student。

说明一下编译时类型和运行时类型：

> Java的引用变量有两个类型，一个是编译时类型，一个是运行时类型，编译时类型由声明该变量时使用的类型决定，运行时类型由实际赋给该变量的对象决定。如果编译时类型和运行时类型不一致，会出现所谓的多态。因为[子类](https://so.csdn.net/so/search?q=子类&spm=1001.2101.3001.7020)其实是一种特殊的父类，因此java允许把一个子类对象直接赋值给一个父类引用变量，无须任何类型转换，或者被称为向上转型，由系统自动完成。
>
> 引用变量在编译阶段只能调用其编译时类型所具有的方法，但运行时则执行它运行时类型所具有的方法，因此，编写Java代码时，引用变量只能调用声明该变量所用类里包含的方法。与方法不同的是，对象的属性则不具备多态性。通过引用变量来访问其包含的实例属性时，系统总是试图访问它编译时类所定义的属性，而不是它运行时所定义的属性。

泛型：

最简单的好处，让List里面的数据类型相同，若是不符合直接回爆红，无需等到编译时才发现出错。

课堂练习（3min）：

145. 二叉树的后序遍历

144. 二叉树的前序遍历

层序遍历(BFS)

102. 二叉树的层序遍历

接下来我们再来介绍二叉树的另一种遍历方式：层序遍历。

层序遍历一个二叉树。就是从左到右一层一层的去遍历二叉树。这种遍历的方式和我们之前讲过的都不太一样。

需要借用一个辅助数据结构即队列来实现，**队列先进先出，符合一层一层遍历的逻辑，而用栈先进后出适合模拟深度优先遍历也就是递归的逻辑。**

而这种层序遍历方式就是图论中的广度优先遍历，只不过我们应用在二叉树上。

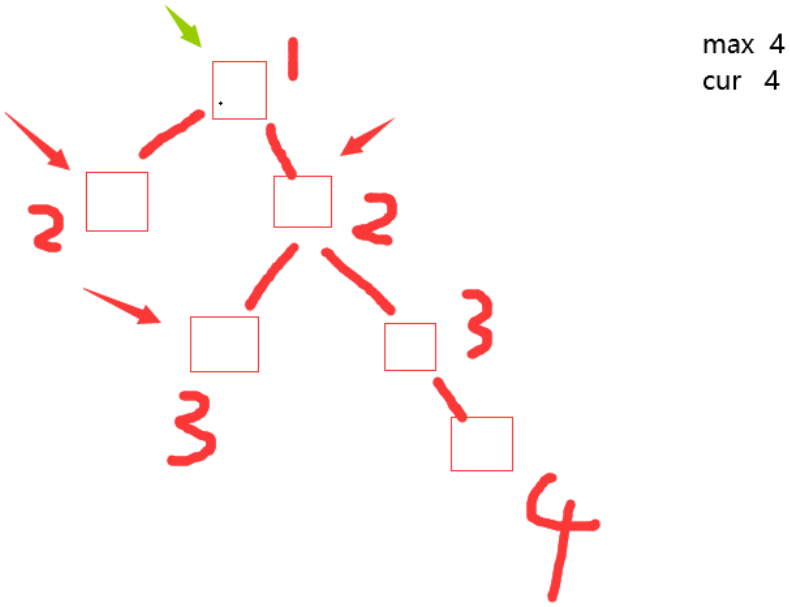
课堂练习：

199. 二叉树的右视图

树的深度（DFS）

104. 二叉树的最大深度

思路一 定义一个maxDepth记录最大深度，**每次**与当前深度比较 有必要吗？**没有**



由于深度是根节点到叶子结点的距离，所以只需要在到达叶子结点时比较即可

```
int ans; // 变量用于存储最大深度
public int maxDepth(TreeNode root) {
    int start = 0; // 起始深度为0
    dfsForMaxDepth(root, start); // 调用深度优先搜索函数
    return ans; // 返回最大深度
}

void dfsForMaxDepth(TreeNode node, int currentDepth) {
    if (node == null) {
        return; // 如果节点为空，从函数中返回
    }
    currentDepth++; // 增加当前深度
    // 检查当前节点是否为叶子节点（没有左子节点和右子节点）
    if (node.left == null && node.right == null) {
        ans = Math.max(ans, currentDepth); // 使用最大深度更新ans变量
    }
    // 递归调用左子节点和右子节点的深度优先搜索函数
    dfsForMaxDepth(node.left, currentDepth);
    dfsForMaxDepth(node.right, currentDepth);
```

```
}
```

官方题解

```
public int maxDepth(TreeNode root) {
    // 如果根节点为空，表示当前子树为空树，深度为0
    if (root == null) {
        return 0;
    } else {
        // 递归计算左子树的深度
        int leftHeight = maxDepth(root.left);
        // 递归计算右子树的深度
        int rightHeight = maxDepth(root.right);
        // 返回左子树深度和右子树深度的最大值，再加上根节点的深度1
        return Math.max(leftHeight, rightHeight) + 1;
    }
}
```

if-else 语句可以换成三目运算符 ? :

```
return root==null?0:Math.max(maxDepth(root.left).maxDepth(root.right))+1;
```

111. 二叉树的最小深度

那我们直接用原来的代码修改一下max—>min是不是就搞定了哇;

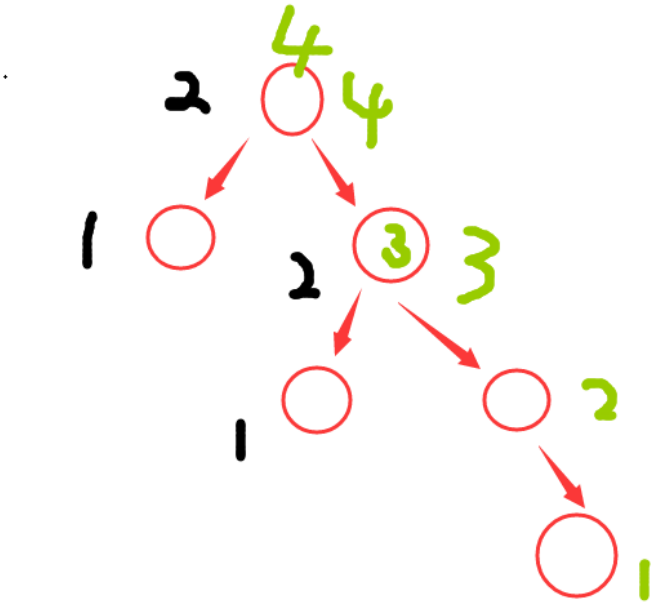
```
return root==null?0:Math.min(maxDepth(root.left).maxDepth(root.right))+1;
```

提交测试

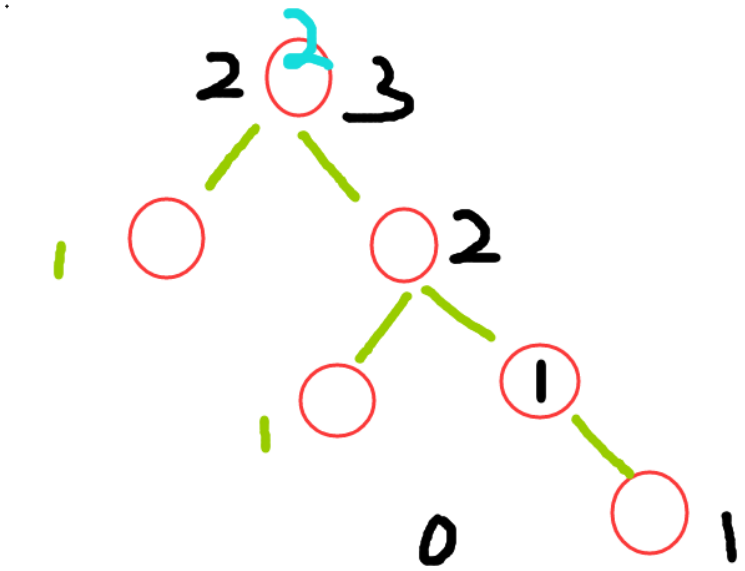
为什么? **提问**

我们来分析一下过程

原来求最大的深度时

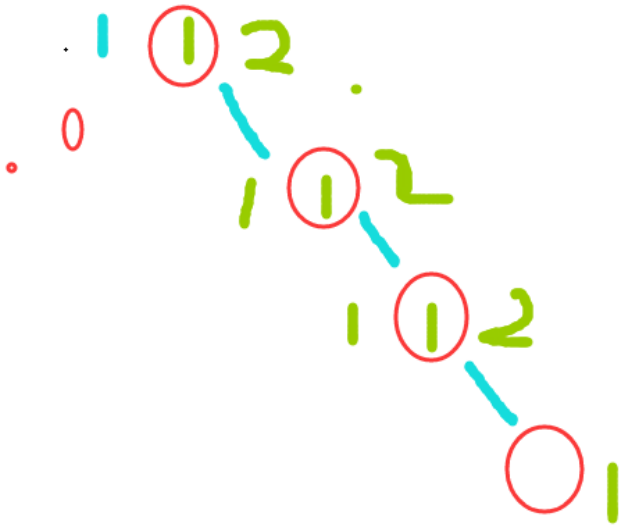


我们换成最小会出现什么问题呢?



虽然中途有问题但是答案是对了

为什么会出错? **提问**



那怎么办？**提问** 是不是要判断当前是不是叶子结点啊，也就是要处理中间节点

叶子节点的定义是左孩子和右孩子都为 null 时叫做叶子节点
当 root 节点左右孩子**都为空**时，返回 1
当 root 节点左右孩子有**一个为空**时，返回**不为空**的孩子节点的深度
当 root 节点左右孩子**都不为空**时，返回左右孩子较小深度的节点值（和最大一样）

让同学们写一下代码（5 min）

（交换1/2的顺序）1的判断恒定为false 为啥？**提问**

修改后代码

```
public int minDepth(TreeNode root) {
    if(root == null) return 0;
    //这道题递归条件里分为三种情况
    //1.左孩子和有孩子都为空的情况，说明到达了叶子节点，直接返回1即可
    if(root.left == null && root.right == null) return 1;
    //2.如果左孩子和由孩子其中一个为空，那么需要返回比较大的那个孩子的深度
    int m1 = minDepth(root.left);
    int m2 = minDepth(root.right);
    //这里其中一个节点为空，说明m1和m2有一个必然为0，所以可以返回m1 + m2 + 1;
    if(root.left == null || root.right == null) return m1 + m2 + 1;
    //3.最后一种情况，也就是左右孩子都不为空，返回最小深度+1即可
    return Math.*min*(m1,m2) + 1;
}
```

平衡二叉树

课堂练习：

110. 平衡二叉树

`平衡二叉树`：在一棵搜索二叉树中每个节点的左右子树的高度差的`绝对值`不超过1。左右子树的高度差被称为`平衡因子`（平衡因子=右子树高度-左子树高度）。若一颗平衡二叉树的节点个数为n，那么其高度为logN。

```
public class P110 {
    public boolean isBalanced(TreeNode root) {
        if (root == null) {
            return true;
        }
        //只有当左右子树平衡并且自身平衡才是平衡
        return isBalanced(root.left)&&isBalanced(root.right)&&Math.abs(depth(root.left)-
depth(root.right))<=1;
    }
    private int depth(TreeNode root){
      return root==null?0:Math.max(depth(root.left),depth(root.right))+1;
    }
}
```

迷宫问题(DFS解法)

HJ43迷宫问题 *牛客题霸* 牛客网 (nowcoder.com) （DFS）

1. 题目理解：
    - 给定一个迷宫地图，地图由 n 行 m 列的矩阵表示，矩阵中的每个元素为 0 或 1，其中 0 表示通路，1 表示障碍物。
    - 从迷宫的左上角出发，找到一条从起点到终点的路径，路径可以在迷宫中向上、向下、向左、向右移动，不可以斜着移动，也不能经过障碍物。
2. 解题思路：
    - 使用深度优先搜索（DFS）算法来搜索可行路径。
    - 确定结束时的情况。
    - 从起点开始，递归地尝试向下、向右、向上、向左四个方向移动，直到到达终点或者无法继续移动。
    - 在递归的过程中，使用一个列表 `path` 来存储当前路径经过的位置。
    - 当找到一条可行路径时，将路径上的位置输出。
3. 代码讲解：
    - 首先，在 `main` 方法中读取输入的迷宫地图的行数和列数，并构建一个二维数组 `map` 来表示地图。
    - 接下来，定义一个空的列表 `path` 来存储路径上的位置。
    - 调用 `dfs` 方法进行深度优先搜索，初始位置为起点 `(0, 0)`。
    - 在 `dfs` 方法中，首先将当前位置 `(x, y)` 加入路径列表 `path` 中，并将该位置标记为已走过（将 `map[x][y]` 置为 1）。
    - 然后，检查是否到达终点，如果是，则返回 true，表示找到了一条可行路径。
    - 否则，依次尝试向下、向右、向上、向左移动，并在每个移动方向上递归调用 `dfs` 方法。
    - 如果在某个方向上找到了一条可行路径，则返回 true。

- 如果所有的方向都无法找到可行路径，则进行回溯：从路径列表 `path` 中移除最后一个位置，并将该位置标记为未走过（将 `map[x][y]` 置为 0）。
- 最后，在 `main` 方法中，输出路径列表 `path` 中的所有位置。

```
import java.util.*;

public class HJ43 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        // 注意 hasNext 和 hasNextLine 的区别
        while (in.hasNextInt()) { // 注意 while 处理多个 case
            int n = in.nextInt(); // 读取迷宫的行数
            int m = in.nextInt(); // 读取迷宫的列数
            // 构造迷宫
            int[][] map = new int[n][m]; // 创建一个二维数组来表示迷宫
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < m; j++) {
                    map[i][j] = in.nextInt(); // 读取迷宫的每个位置的值
                }
            }
            // 路径存储的数组
            List<Pos> path = new ArrayList<>(); // 创建一个列表来存储路径上的位置
            // DFS 搜索路径
            dfs(map, 0, 0, path); // 调用深度优先搜索方法，从起点 (0, 0) 开始搜索路径
            // 输出路径
            for (Pos p : path) {
                System.out.println("(" + p.x + "," + p.y + ")"); // 输出路径上的每个位置
            }
        }
    }

    // 返回值 标记是否找到可通行的路劲
    public static boolean dfs(int[][] map, int x, int y, List<Pos> path) {
        int length = map.length - 1; // 迷宫的行数
        int width = map[0].length; // 迷宫的列数
        // 添加路径并标记已走
        path.add(new Pos(x, y)); // 将当前位置添加到路径列表中
        map[x][y] = 1; // 将当前位置标记为已走过
        // 结束标志
        if (x == length && y == width) { // 如果当前位置是终点，则找到了一条可行路径
            return true;
        }
        // 向下能走时
        if (x + 1 < length && map[x + 1][y] == 0) { // 如果向下的位置可以通行且未走过
            if (dfs(map, x + 1, y, path)) { // 递归调用深度优先搜索，从向下的位置开始继续搜索路径
                return true;
            }
        }
        // 向右能走时
        if (y + 1 < length && map[x][y + 1] == 0) { // 如果向右的位置可以通行且未走过
            if (dfs(map, x, y + 1, path)) { // 递归调用深度优先搜索，从向右的位置开始继续搜索路径
                return true;
            }
        }
        // 向上能走时
        if (x - 1 > -1 && map[x - 1][y] == 0) { // 如果向上的位置可以通行且未走过
```

```
            if (dfs(map, x - 1, y, path)) { // 递归调用深度优先搜索，从向上的位置开始继续搜索路径
                return true;
            }
        }
        // 向左能走时
        if (y - 1 > -1 && map[x][y - 1] == 0) { // 如果向左的位置可以通行且未走过
            if (dfs(map, x, y - 1, path)) { // 递归调用深度优先搜索，从向左的位置开始继续搜索路径
                return true;
            }
        }
        // 回溯
        path.remove(path.size() - 1); // 从路径列表中移除最后一个位置
        map[x][y] = 0; // 将当前位置标记为未走过
        return false;
    }

    public static class Pos {
        int x;
        int y;

        public Pos(int x, int y) {
            this.x = x;
            this.y = y;
        }
    }
}
```

作业

429. N 叉树的层序遍历

113. 路径总和 II

129. 求根节点到叶节点数字之和 （要求DFS和BFS两种方式）

HJ43迷宫问题 *牛客题霸* 牛客网 (nowcoder.com) （BFS）