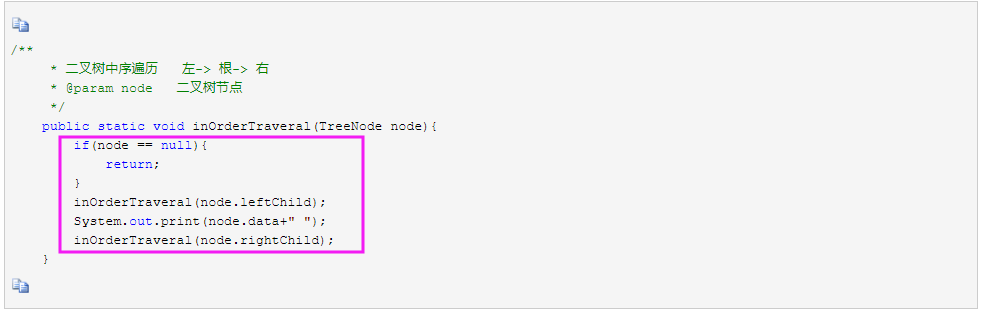
算法思想总结

1. 二叉树的先中后序非递归遍历（Access the binary tree）

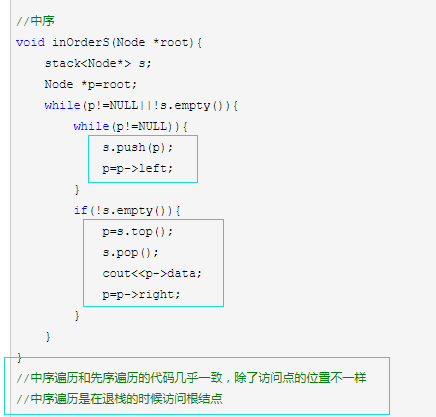
先序遍历：沿着左支走到底，将走的节点输出同时入站，当走到左支结点的尾部时，弹出栈，将跟结点置为弹出栈结点的右孩子，继续入栈。



//中序遍历和先序遍历的代码几乎一致，除了访问点的位置不一样

//中序遍历是在退栈的时候访问根结点







1. 蛇形打印二叉树（SnakePrintBinaryTree）

分析：(两个栈一个存基数层从左往右，一个存偶数层从右往左)

使用两个栈。一个栈存储奇数层节点，另一个栈存储偶数层节点。

若是奇数层，则打印顺序是从左往右，即先添加左子树，再添加右子树，即先保存左子节点，再保存右子节点。若是偶数层，打印顺序是从右向左，先添加右子树，再添加左子树。因为栈是先进后出，所以需要注意左右子树入栈的顺序。

1. 给定一棵二叉树，想象自己站在它的右侧，按照从顶部到底部的顺序，返回从右侧所能看到的节点值。

层次遍历每次取每层的最后一个

1. 给定一个二叉树，原地将它展开为链表（dfs或者for循环）
2. 最长不重复子串（双指针移动）
3. 判断是否是完全二叉树

每一个节点都有四种情况

我们开始寻找规律：

1：如果当前访问的节点的左右孩子是情况3，说明不是完全二叉树，直接返回false。

2：如果当前访问的节点的左右孩子是情况1，继续访问其他节点。

3：如果当前访问的节点的左右孩子是情况2或者情况4，那么我们定义一个状态（接下来访问的所有节点必须全部是叶子节点）。只要遇到情况2或者情况4，这个状态就开启了

1. 链表判断是否有环

设置p1为慢指针，p2为快指针，两者初始时都指向链表的头结点 ，慢指针p1每次前进1步，快指针p2每次前进2步。如果链表存在环，则快指针p2肯定先进入环，慢指针p1后进入环，两个指针必定会相遇。如果不存在环，则快指针会先行到达链表的尾部变为None。

1. 稀疏向量的点乘 要求：尽量高效地实现，需要同时考虑时空复杂度。
2. 区间问题

首先，我们将列表按上述方式排序。然后，我们将第一个区间插入 merged 数组中，然后按顺序考虑之后的每个区间：如果当前区间的左端点在前一个区间的右端点之后，那么他们不会重合，我们可以直接将这个区间插入 merged 中；否则，他们重合，我们用当前区间的右端点更新前一个区间的右端点 end 如果前者数值比后者大的话。

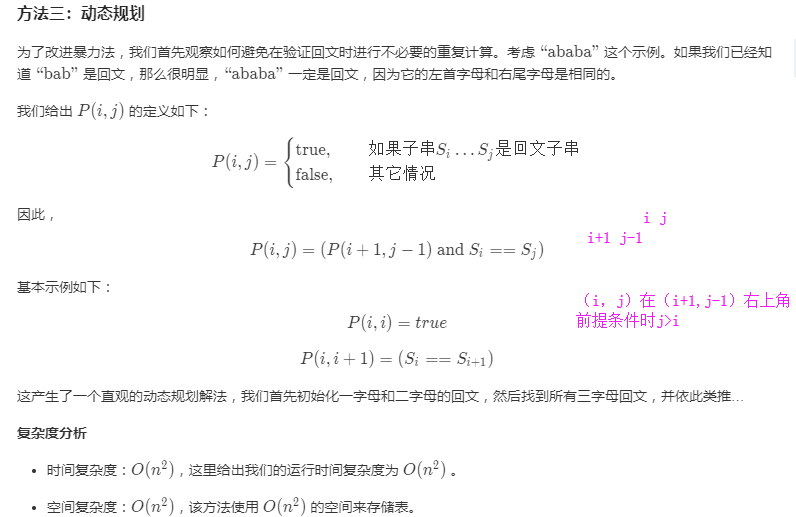
1. 买卖股票的最佳时机，只能一次买入和一次卖出，要求输出买入的时机和卖出的时机和赚取的最大利润。

初始化一个最小值和最大值，随后动态更新数组中的最小值和最大值。最小值和最大值的初始化值

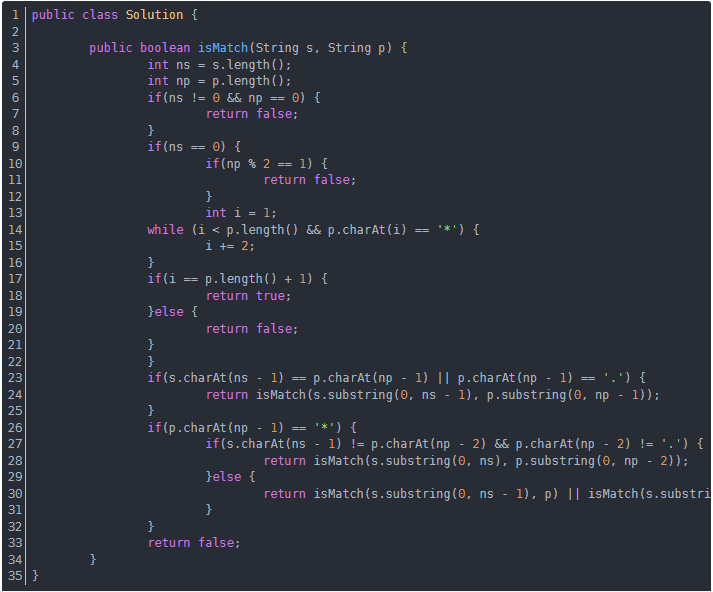
1. 求二叉树的最长路径长度。

树中任意两个节点之间，连接起来的路径最长。方法就是求出每个节点的左子树和右子树的高度，两者相加就是当前节点的最长路径，然后比较每个节点的最长路径，最大的就是结果

1. 求二叉树的最大路径和（把11题的单位1换成权值）
2. 求最长回文子串



1. 正则表达式（没理清）（RegularExpression）



15组合公式

C（M-1，N-1）+C（M-1，N）=C（M，N）

C(M,N)表示从m中选n个，m<n返回0，m==n或者n==0返回1

16两点最短距离（3层for循环，中间的那层for循环作为中间件）

16最大和的连续子串



16 01背包问题

定义：m[i][j]表示考虑到编号为i的物品、当背包容量为j时的最大可容纳价值。（其中i从0开始编号，考虑到i即从0-i的物品都考虑进去）

17dfs

深度优先搜索的思想：

关键在于解决当下改如何做，至于下一步如何做则与当下改如何做是一样的

基本模板

void dfs(int step)

#1.判断边界（满足边界条件的话记得最后返回最后一次调用dfs的地方）

#2.尝试每一种可能

for i in range(1,n+1):

3.继续下一步

dfs(step+1）

4.取消当前操作

#5返回

18三角形面积 s=(a+b+c)/2 S=根号（s(s-a)(s-b)(s-c)）

**400 - Bad Request 错误请求； 401 - Unauthorized 未授权；**

**402 - 需要付款**

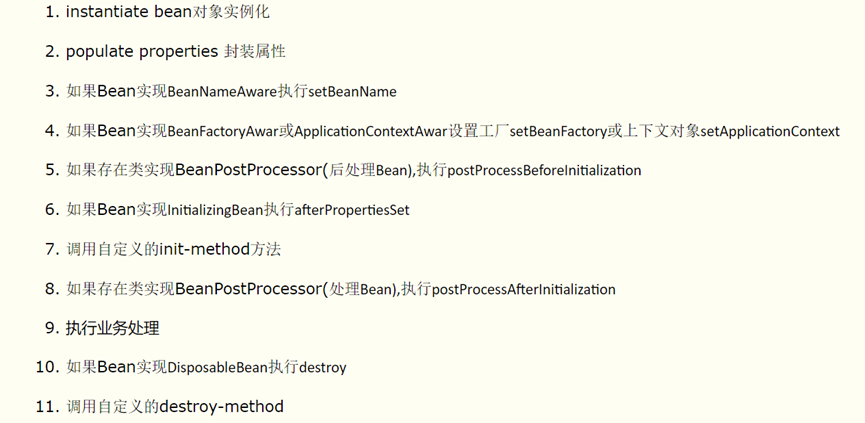
**403 - Forbidden 禁止访问；**

**501 - Not Implemented 没有实现；**

**502 - Bad Gateway 错误的网关；**

**503 - Service Unavailable 服务不可用；**

**504 - Gateway Timeout 网关超时；**

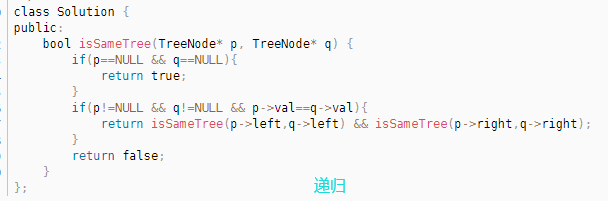


第一步就是对实例化bean，调用构造函数来创建实例，第二步是根据配置，进行相应属性的设置，依赖注入就是在这一步完成的。

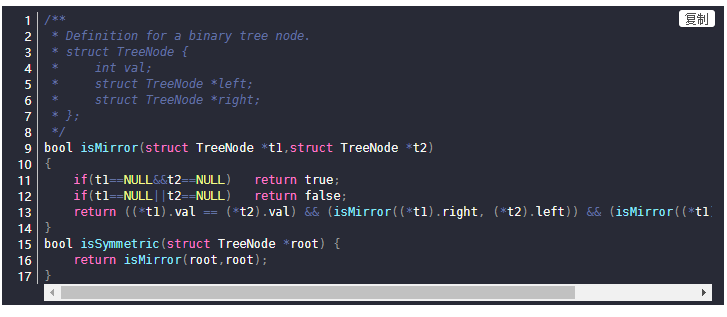
第三步和第四步是让spring去了解咱们的spring容器，第五步和第八步可以针对指定的Bean进行功能增强，这时一般是采用的动态代理，（两种动态代理方式：jdk动态代理和cglib动态代理）。第六步和第十步是通过实现指定的接口来完成init（初始化）和destory（销毁）操作。但是我们在通常情况下不会使用这两步，因为我们可以通过第七步和第十一步，在配置文件中设置相应的初始化和销毁方法。

# 树

1. 给定两个二叉树，编写一个函数来检验它们是否相同。



1. 给定一个二叉树，检查它是否是镜像对称的



# 技巧类型

# 给定一个非负整数数组，你最初位于数组的第一个位置。数组中的每个元素代表你在该位置可以跳跃的最大长度。判断你是否能够到达最后一个位置。

每到一个位置，就去寻找当前步数能去的最远距离，然后检测是否到达最大限度，例如上面的案例，第一步到0号位置，也就是两步，然后两步之内能去的最远距离中是1号元素的3，那么就去1号位置，然后就开始递归的找下去。。。。