**[[LeetCode 124] - 二叉树最大路径和(Binary Tree Maximum Path Sum)](https://www.cnblogs.com/shawnhue/archive/2013/06/08/leetcode_124.html)**

**问题**

给出一个二叉树，找到其中的最大路径和。

路径可以从树中任意一个节点开始和结束。

例如：

给出如下二叉树，

       1

      / \

    2    3

返回6。

**初始思路**

为了简化分析，我们先假设二叉树中所有节点的值都是正数。通过观察可以发现，一棵二叉树的最大路径，就是其左子树的最大路径加上右子树的最大路径。看起来可以从根节点出发通过深度优先递归来求解：

**函数 查找路径**

**如果是叶子节点，返回叶子节点的值**

**如果不是叶子节点**

**左子树路径和 ＝ 查找路径（左子树）**

**右子树路径和 ＝ 查找路径（右子树）**

**如果左子树路径＋右子树路径和＋当前节点值 > 当前最大路径，更新最大路径**

**返回左子树路径＋右子树路径和＋当前节点值**

用题目中的简单例子来验证，是可以得出答案的。但是使用复杂一点的树来验证后就发现其中的问题了，如

              1

           /      \

         2         3

      /      \

    4        5

使用前面的伪代码得出的结果是15，但是其实答案应该是11，由3，1，2，5或者2，4，5得到。分析可以发现问题在于计算2，4，5这棵子树时，它的最长路径为11，这是正确的。但是当它作为左子树向父节点返回最长路径时，因该返回7而不是11。因为从1出发不走重复路径不可能同时到达4或5的－通常二叉树节点路径的定义是每个节点只能访问一次，通过测试数据也可以验证题目就是这样要求的。因此我们需要两个最大值，一个是当前树的最大路径，即前面伪代码算出来的那个值；另一个是当前树向父节点提供的最大路径，这个值应该是根节点的值加上路径最长的子树那边的最大路径。我们向上层递归函数返回这个值。

好了，现在全是正数的情况解决了。让我们开始把负数引入。负数引入后，将会导致以下几个变化：

* 叶子节点的值也有可能成为最大路径。在全是正数的情形下，叶子节点的值肯定不可能会是最大路径，因为加上父节点的值后必然会变大。有了负数以后，这个情况就不成立了，如：

                －1

              /

            3

        这时最大路径就是3。

* 当前树最大路径的计算方法。有了负数以后不能简单的把左子树返回的值，右子树返回的值及当前的值相加了。这里我们把各种情况列举出来：
  + 当前值为正，子树返回值都为正：全相加
  + 当前值为正，子树返回值有一个为正：当前值＋正的那个值，因为负值只会让结果变小。
  + 当前值为正，子树返回值都是负：只取当前值，负值越加越小。
  + 当前值为负，子树返回的值都为正：全相加，虽然值会变小，但是没有当前节点左右就不能联通。
  + 当前值为负，子树返回值有一个为正：当前值＋正的那个值。
  + 当前值为负，子树返回值都为负：当前值，负值越加越小。
* 向父节点提供的最大路径的计算方法。和当前树最大路径计算方法基本一样。就是仍然要左子树右子树的值只能取大的那个。

将上面分析转换成代码，并加入一些细节如没有左（右）子树的判断。请注意由于节点的取值范围并没有限定，所以不能使用某个特殊值作为没有左（右）子树的标志