网站爬取深度优先与广度 优先原理

大家好，我们之前已经了解过了如何利用爬虫来爬取我们想要的数据，但是仅限于爬取有限的URL，这样严格来说并不算是爬虫，我们利用爬虫要爬取的应该是大量的数据，同时就对应着要爬取大量的URL。

例如，现在有这样一个需求，爬取拉钩上的所有职位信息，然后对获取到的数据做数据分析，我们该如何选择爬取策略呢？

网站的爬取一般有两种爬取策略：深度优先和广度优先。

我们先来讨论下深度优先，它的原理是这样的：

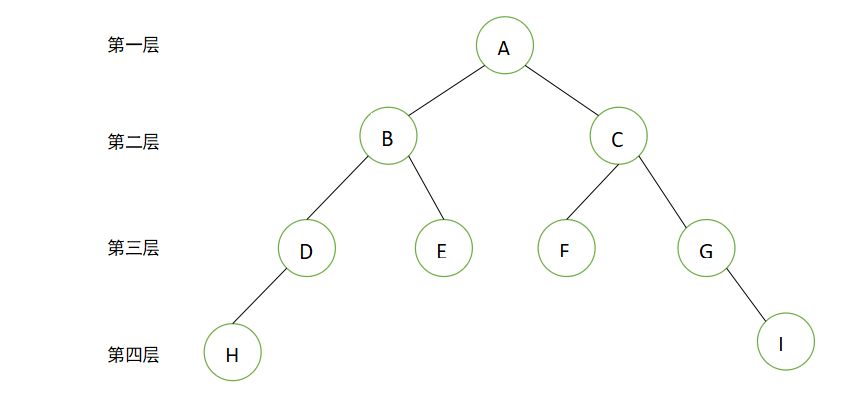
进入拉钩首页后，这个页面可能有100个URL，先选择第一个URL发起请求，我们将这个URL暂时叫做url\_level1，然后就进入了url\_level1的界面。

再在这个界面中选择第一个URL，判断是否已经向这个URL发起过请求，如果有，就选择第二个URL，依次类推，找到未向其发起请求的第一个URL，然后向它发起请求，我们将这个URL暂时叫做url\_level2，然后就进入了url\_level2的界面。

再在这个界面中选择第一个URL，判断是否已经向这个URL发起过请求，如果有，就选择第二个URL，依次类推，找到未向其发起请求的第一个URL，然后向它发起请求，我们将这个URL暂时叫做url\_level3，然后就进入了url\_level3的界面。

同理，我们一层层深入下去，直到最后一个界面中的URL都被发起过请求，然后就倒退回上一层，继续往下深入。如此反复，就能爬取到网站的所有URL。

接下来我们来看一个图，再来理解下深度优先的过程是怎么样的：



本图是一个二叉树（数据结构的相关知识，了解下就可以），我们可以简单看作是一个网页的所有URL分布，深度优先最终得到的结果是什么呢？

从A开始，先进入B，然后进入D，再进入H，发现已经到底了，然后退回D，发现D下的H已经处理过，也没有其他内容了，就退回B，发现B还有E没有遍历，就进入E，又到底了，就退回B，同理，又退回到A，进入C，再进入F，到底了，退回C，进入G，发现G左边没有节点，就进入I，到这里就已经完全结束了。

得到的最终结果就是A-B-D-H-E-C-F-G-I。再退回去理解一下网站爬取的深度优先策略，是不是就容易了一点。

接下来，我们再来讨论下广度优先，它的原理是这样的：

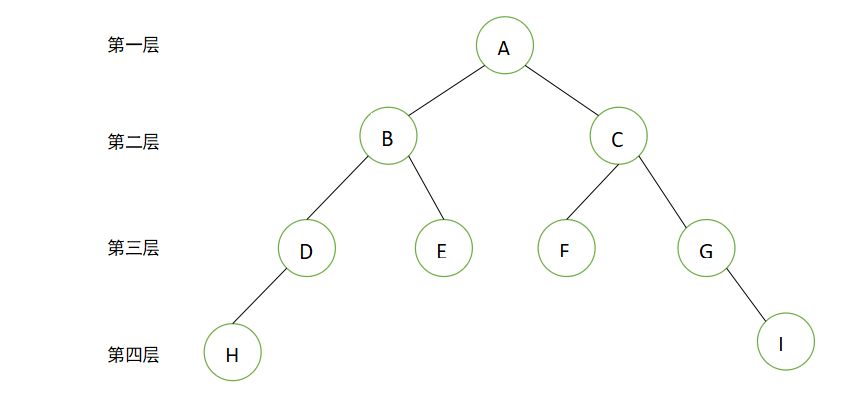
进入拉钩首页后，这个页面可能有100个URL，依次向这100个URL发起请求，我们将这些URL暂时叫做url\_level1\_one，url\_leve1\_two.....url\_level1\_hundred

然后进入url\_level1\_one的页面，这个页面可能有80个URL，然后依次向这80个URL发起请求（先判断是否已经发起过请求，如果已发起过，就略过，不再发起请求），我们将这个URL暂时叫做url\_leve2\_one\_1，url\_level2\_one\_2...url\_level2\_one\_eighty

然后再进入url\_level1\_two的页面，这个页面可能有60个URL，然后依次向这60个URL发起请求（先判断是否已经发起过请求，如果已发起过，就略过，不再发起请求），我们将这个URL暂时叫做url\_leve2\_two\_1，url\_level2\_two\_2...url\_level2\_two\_sixty

同理，我们先爬取当前页面的所有URL，然后进入当前页面的第一个URL，爬取这个URL下的所有URL，然后进入第二个URL，爬取这个URL下的所有URL，依次类推，就能获取到该网站下所有的URL。

同样的，还是来看上面的图，再来理解下广度优先的过程是怎么样的：



从A开始，先进入B，再进入C，再回到B，进入B下面的D，再进入E，再回到C，进入C下面的F，再进入G，再回到D，进入D下的H，发现D下已经没有元素了，就回到E，发现E下没有元素，就回到F，同样的，F下没有元素，就回到G，发现G下面左边没有节点，就进入I，这样就已经结束了。

得到的最终结果是A-B-C-D-E-F-G-H-I。再退回去理解一下网站爬取的广度优先策略，是不是就容易了一点。

好了，今天的分享本来到这里就结束了，但是还是想多唠叨一点东西，那就是用python来实现深度优先和广度优先，接下来的内容属于扩展性内容，不理解也没有关系，主要理解其中的思想就可以：

我们先来看下深度优先如何用python进行实现：

# 实现功能：深度优先遍历（只是作为某一个数据结构的一部分，并不能运行，只是演示原理）

# 采用方法：递归实现（对递归有兴趣的童鞋可以去了解下，自己实现数据结构时经常用到）

**def** **depth\_first**(Root\_Node):

**if** Root\_Node **is** **None**:

# 递归到底的情况是：如果当前结果为None，就直接return

**return**

print(Root\_Node.value)

**if** Root\_Node.Left\_Node:

# 如果当前节点有左子节点，那么就递归，将当前节点的左子节点作为参数传递进去

**return** depth\_first(Root\_Node.Left\_Node)

**if** Root\_Node.Right\_Node:

# 如果当前节点有右子节点，那么就递归，将当前节点的右子节点作为参数传递进去

**return** depth\_first(Root\_Node.Right\_Node)

再来看下广度优先如何用python进行实现：

**def** **breadth\_first**(Root\_Node):

**if** Root\_Node **is** **None**:

# 递归到底的情况是：如果当前结果为None，就直接return

**return**

queue = []

queue.append(Root\_Node)

# 将当前节点加入到队列中

**while** queue:

# 只要列表不为空，就一直循环

Node = queue.pop(0)

# 每次循环都将列表的第一个元素pop出去

print(Node.value)

**if** Root\_Node.Left\_Node:

# 如果当前节点的左子节点存在，那么就添加到列表的尾部

queue.append(Root\_Node.Left\_Node)

**if** Root\_Node.Right\_Node:

# 如果当前节点的右子节点存在，那么就添加到列表的尾部

queue.append(Root\_Node.Right\_Node)

好了，到这里，我们今天的分享就结束了，大家加油。

二十五  阅