Sunday • 更新于 2022-10-19

上一节 11：局部总结：... 13：源码逻辑：... 下一节

# 12：前置知识：场景五：最长递增子序列

在场景五的 `diff` 中， `vue` 使用了 **最长递增子序列** 这样的一个概念，所以想要更好地理解场景五，那么我们需要先搞明白，两个问题：

1. 什么是最长递增子序列?
2. 最长递增子序列在 `diff` 中的作用是什么?

## 什么是最长递增子序列

> 维基百科 - 最长递增子序列
> 在一个给定的数值序列中，找到一个子序列，使得这个子序列元素的数值依次递增，并且这个子序列的长度尽可能地大。

只看概念可能难以理解，我们来看一个具体的例子。

假设，我们现在有一个这样两组节点：

代码块

```
旧节点：1,2,3,4,5,6
新节点：1,3,2,4,6,5
```

我们可以根据 **新节点** 生成 **递增子序列（非最长）（注意：并不是惟一的）**，其结果为：

1. `1、3、6`
2. `1、2、4、6`
3. ...

## 最长递增子序列在 `diff` 中的作用是什么

那么现在我们成功得到了 递增子序列，那么下面我们来看，这两个递增子序列在我们接下来的 `diff` 中起到了什么作用。

根据我们之前的四种场景可知，所谓的 `diff`，其实说白了就是对 **一组节点** 进行 **添加、删除、打补丁** 的对应操作。那么除了以上三种操作之外，其实还有最后一种操作方式，那就是 **移动。**

对于以上的节点对比而言，如果我们想要把 **旧节点转化为新节点**，那么将要涉及到节点的 **移动**，所以问题的重点是：如何进行移动。

大家看到这里可以，先暂停，想一下：**如果是你的话，那么你会如何移动节点，才能以最少的移动次数，完成更新?**

---

那么接下来，我们来分析一下移动的策略，整个移动根据递增子序列的不同，将拥有两种移动策略：

1. `1、3、6` 递增序列下：
    1. 因为 `1、3、6` 的递增已确认，所以它们三个是不需要移动的，那么我们所需要移动的节点无非就是 **三** 个 `2、4、5` 。
    2. 所以我们需要经过 **三次** 移动

2. 1、2、4、6 递增序列下：

    1. 因为 1、2、4、6 的递增已确认，所以它们四个是不需要移动的，那么我们所需要移动的节点无非就是 **两个** 3、5 。
    2. 所以我们需要经过 **两次** 移动

所以由以上分析，我们可知：**最长递增子序列的确定，可以帮助我们减少移动的次数**

所以，当我们需要进行节点移动时，移动需要事先构建出最长递增子序列，以保证我们的移动方案。

11：局部总结：前四种 diff 场景的总结与乱... 上一节 下一节 13：源码逻辑：场景五：求解最长递增子序列

我要提出意见反馈