06-C1

二叉搜索树

平衡：期望树高

邓俊辉
deng@tsinghua.edu.cn

上梁不正下梁歪

树高：最坏情况与平均情况

❖ 由以上的实现与分析，BST主要接口`search()`、`insert()`和`remove()`的运行时间

在最坏情况下，均线性正比于其高度$\mathcal{O}(h)$

❖ 若不能有效地控制**树高**，就无法体现出BST相对于向量、列表等数据结构的明显优势

❖ 比如在**最（较）坏情况**下，二叉搜索树可能彻底地（接近地）**退化**为列表

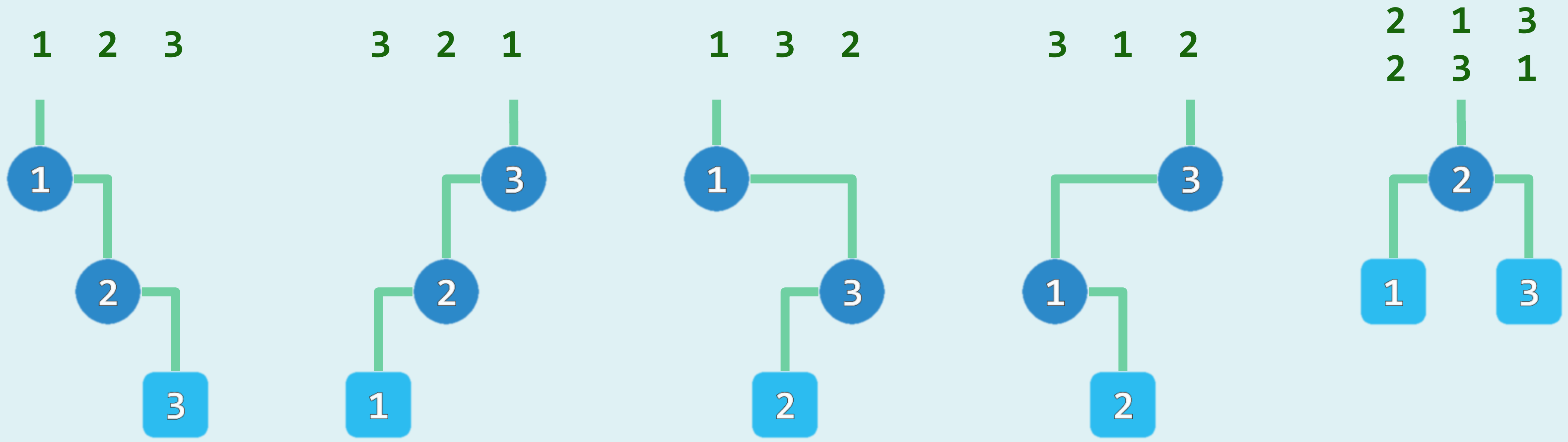
此时的性能不仅没有提高，而且因为结构更为复杂，反而会（在常系数意义上）下降

❖ 那么，出现此类最坏、较坏情况的**概率**有多大？

或者，从**平均**复杂度的角度看，二叉搜索树的性能究竟如何？

❖ 以下按两种常用的随机统计**口径**，就此做一分析和对比

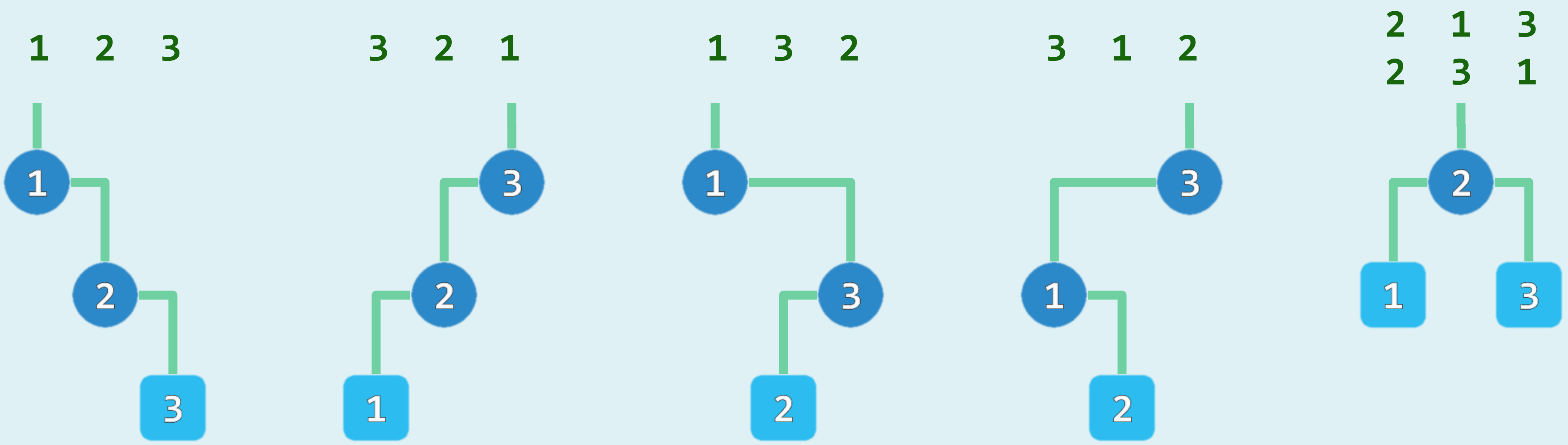
随机生成：n个词条$\{ e_1, e_2, e_3, \dots e_n \}$按随机排列$\sigma = (e_{i_1}, e_{i_2}, e_{i_3}, \dots e_{i_n})$依次插入



- 若假设各排列出现的概率**均等**（1/n!），则BST**平均**高度为$\Theta(\log n)$
- 的确，多数实际应用中的BST**总体上**都是如此生成和演化的

  即便计入remove()，也可通过随机使用succ()和pred()，避免逐渐**倾侧**的趋势
- 然而问题恰恰在于，所有排列出现的概率的确**均等**吗？

随机组成：n个互异节点，在遵守顺序性的前提下，随机确定拓扑联接关系



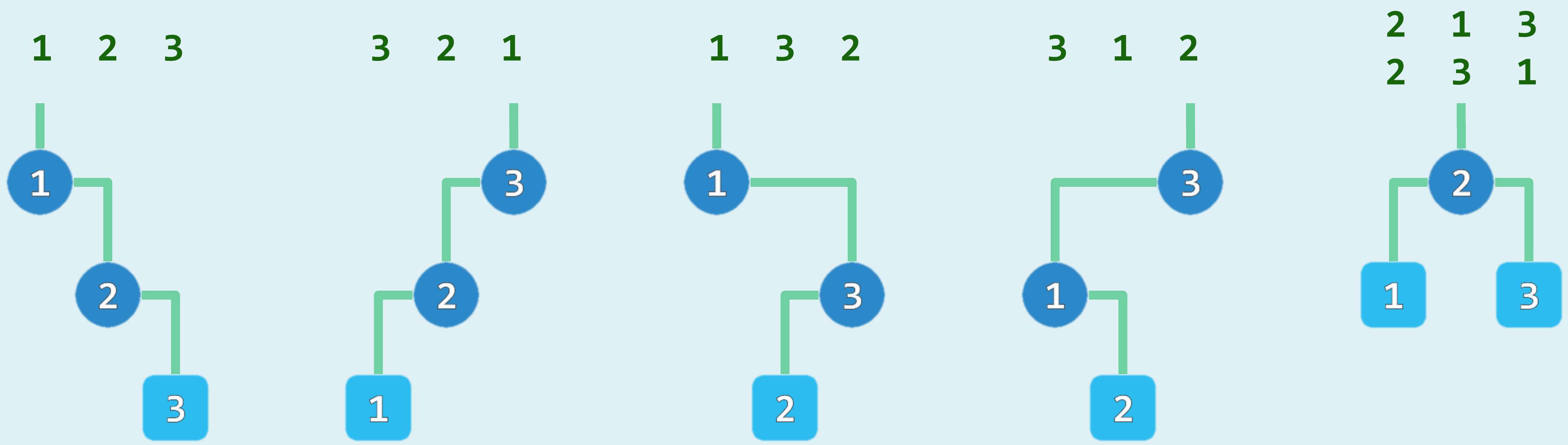
- 由n个互异节点随机**组成**的BST，若共计S(n)棵，则有

$$S(n) = \sum_{k=1}^{n} S(k-1) \cdot S(n-k) = catalan(n) = (2n)!/(n+1)!/n!$$

- 假定所有BST**等概率**地出现，则其**平均**高度为 $\Theta(\sqrt{n})$

- 在Huffman编码之类的应用中，二叉树（尽管还不是BST）的确是逐渐**拼合**而成的

logn vs. sqrt(n)



❖ 两种口径所估计出的平均高度差异极大——谁更可信？谁更接近于真实情况？

❖ 后者未免太**悲观**，但前者则过于**乐观**：BST越低，**权重**越大；而最根本的原因在于...

❖ **理想随机**在实际中**绝难出现**：局部性、关联性、（分段）单调性、（近似）周期性、...

较高甚至极高的BST频繁出现，不足为怪；平衡化处理很有必要！