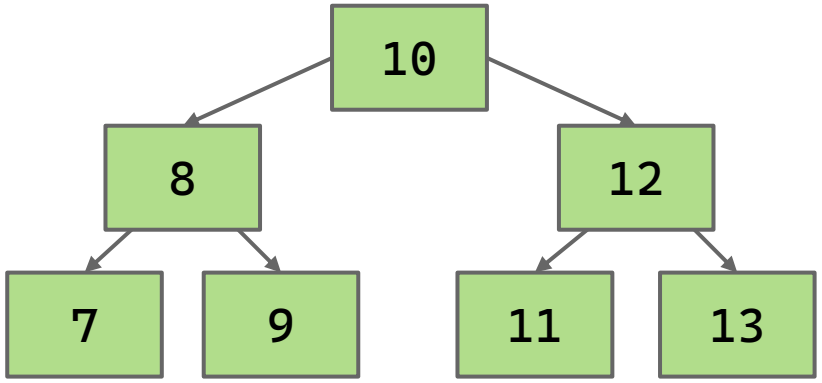
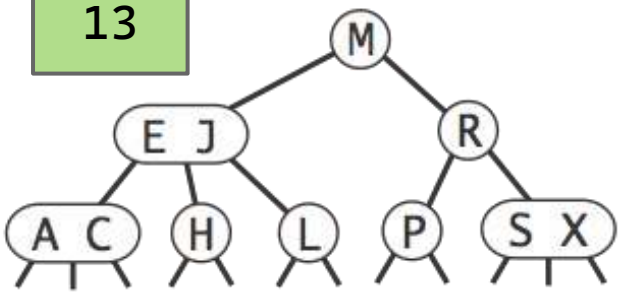
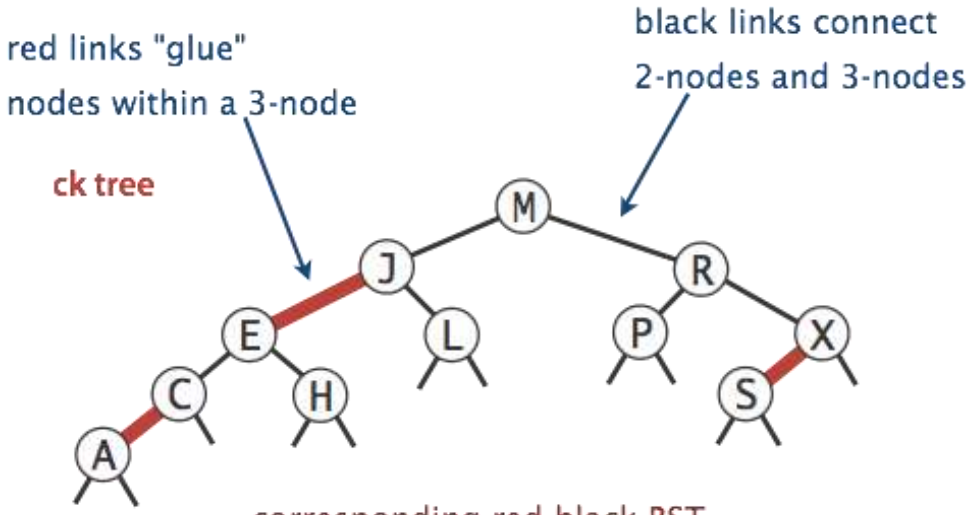


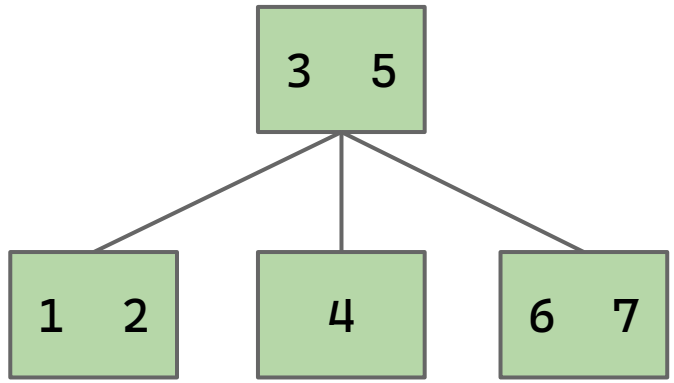


2-3 tree



corresponding red-black BST

BSTs, **B-trees**, AVL trees, Red-black trees



主讲人：七海Nana7mi

课程大纲：CS61B

课程说明：

- 课程内容基于 UC-Berkeley 的课程 CS61B-sp18 与 CS61B-fa23。可以理解为课程的汉化视频。
- 课程使用的编程语言为 Java。
- AI 语音模型来源 BiliBili 用户 Xz乔希。
- 七海也在学习中，有错误敬请指出！

- 第一期视频：【CS61B汉化】七海讲数据结构-二叉搜索树【七海Nana7mi】

二叉搜索树的高度

Lecture 2

二叉搜索树的高度

平衡树

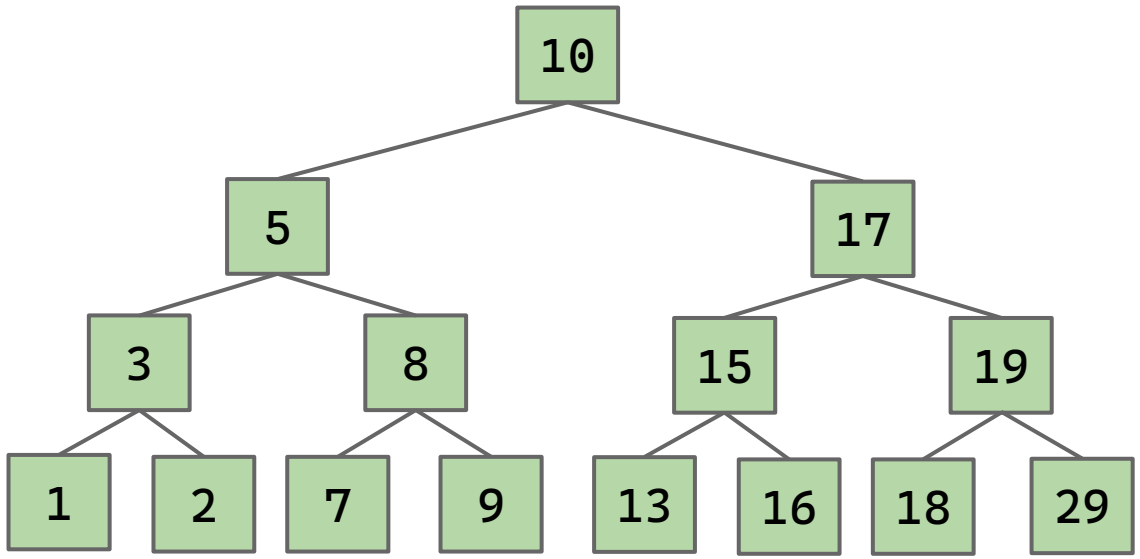
B树

- 节点分裂
- 连锁分裂
- 正式定义
- 性质
- 性能

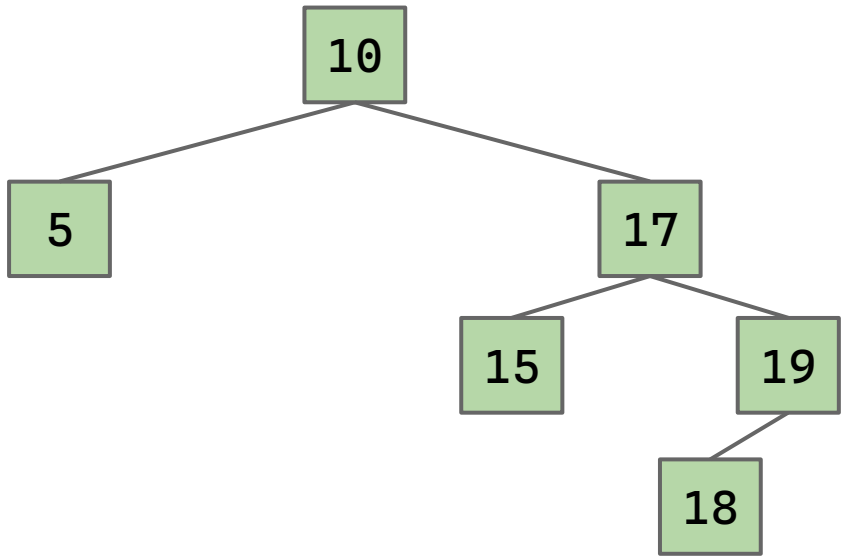
二叉搜索树的高度

Height is just max depth -> 树的高度就是最大深度

H=3

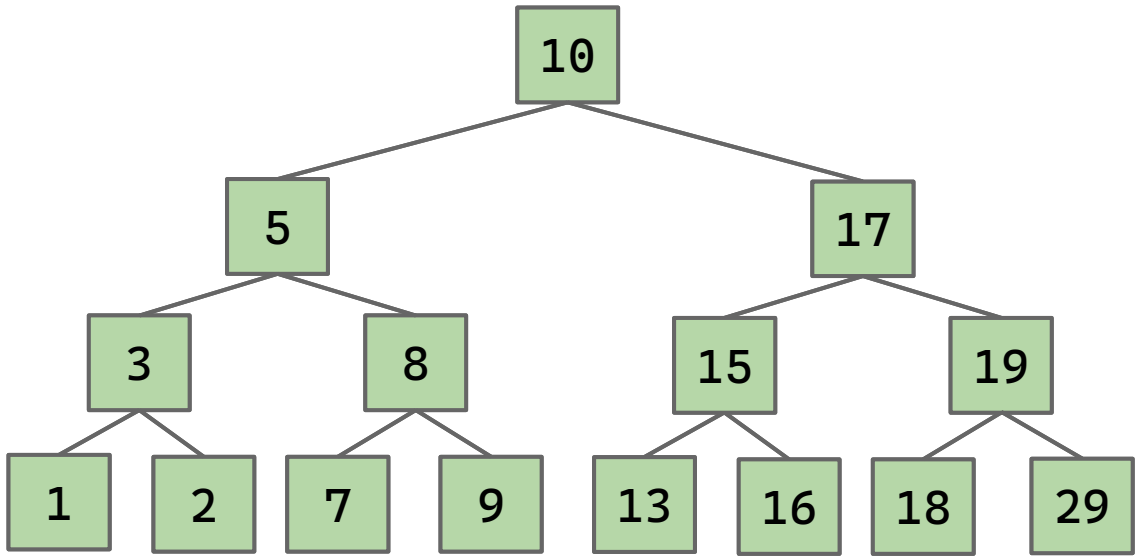


H=3



二叉搜索树的高度

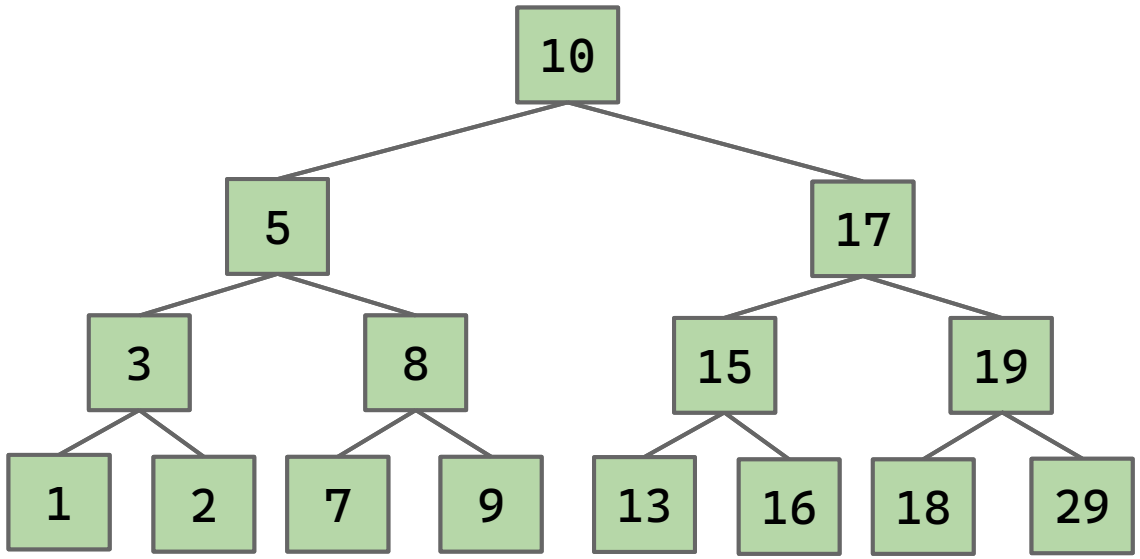
H=3



H = θ(log(N))

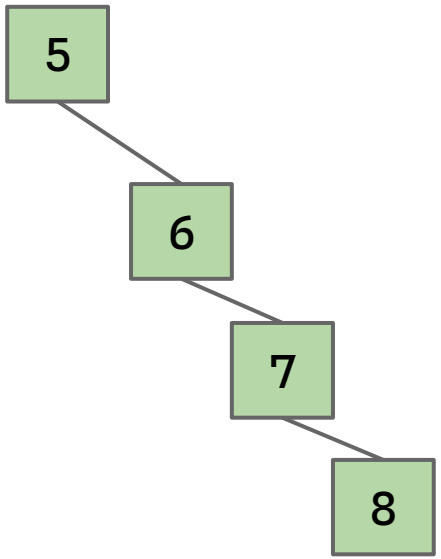
二叉搜索树的高度

H=3



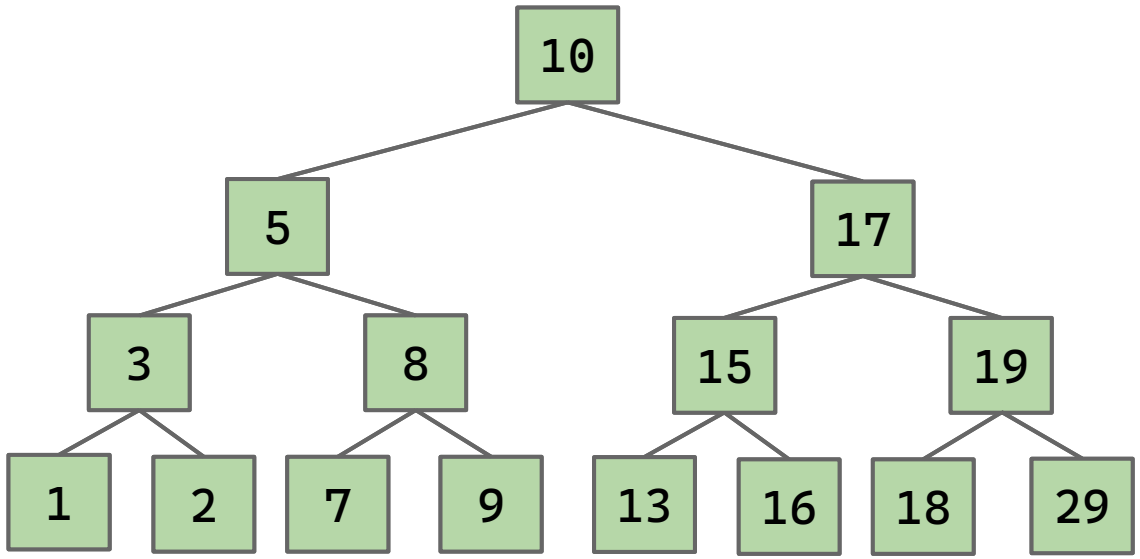
H = θ(log(N))

H=3



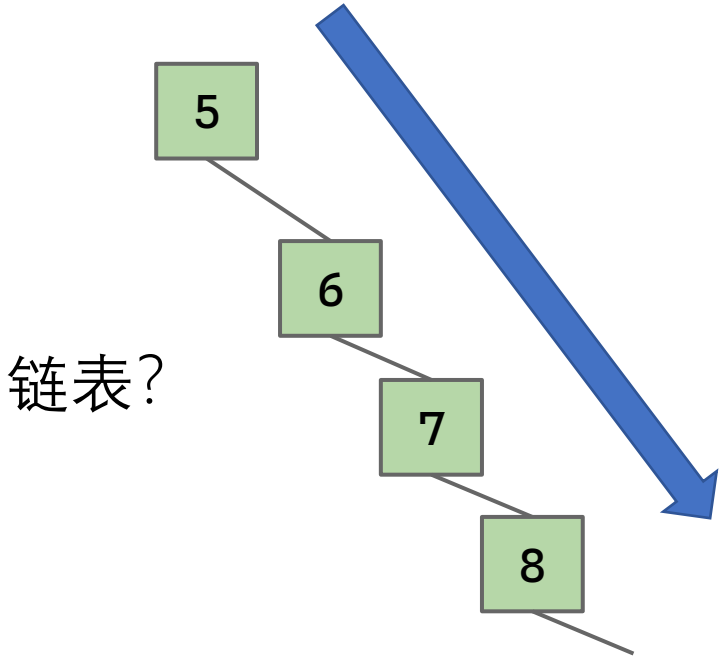
二叉搜索树的高度

H=3



H = θ(log(N))

H=3



链表?

H = θ(N)

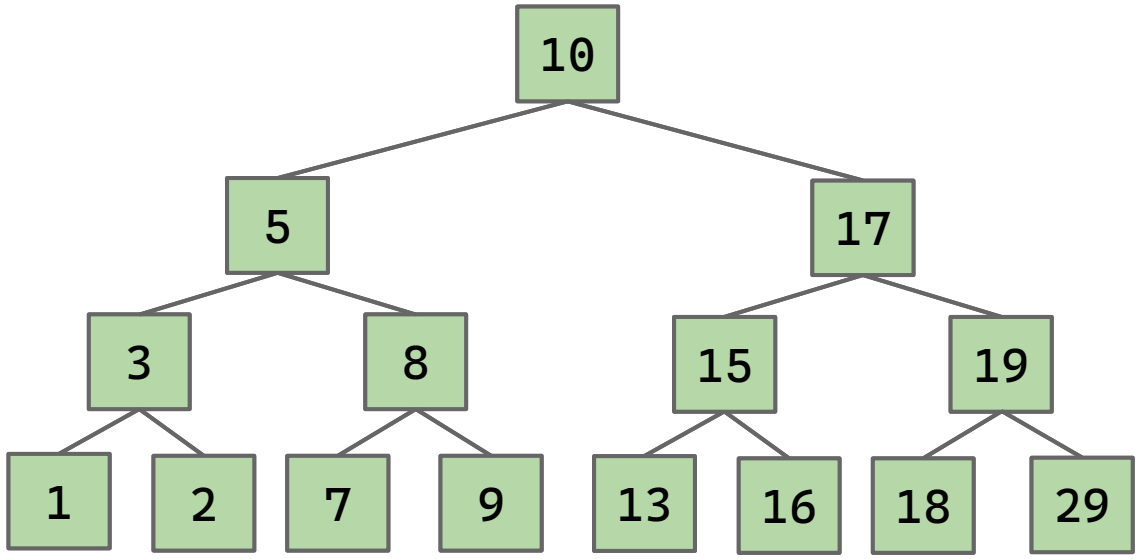
...

二叉搜索树的高度

BST height is both of these:

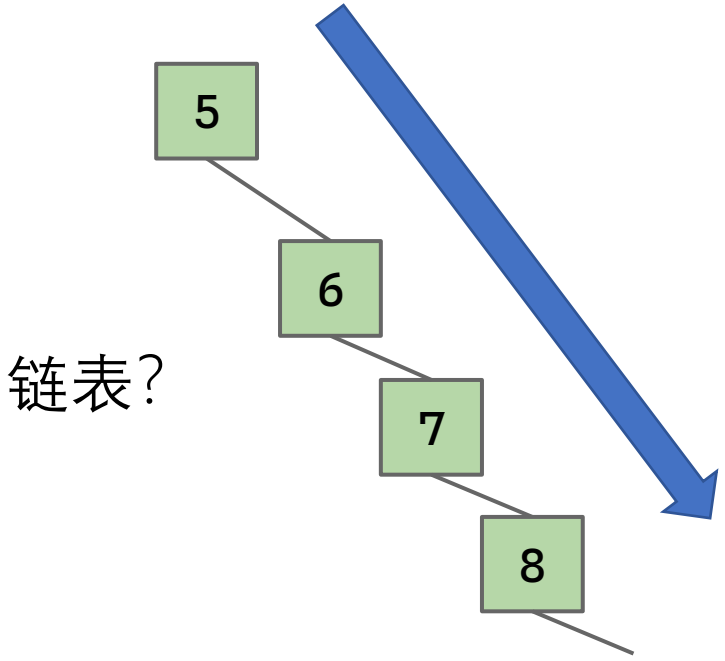
- θ(log N) in the best case -> 最佳情况，树的高度是 θ(log N) ，插入、删除、修改和查找时间都和结点数成对数关系（插入、删除、修改都包括查找操作）
- θ(N) in the worst case -> 最差情况，BST 退化为链表，坏!

H=3



H = θ(log(N))

H=3



H = θ(N)

...

二叉搜索树的高度

二叉搜索树插入可能会导致层数变得不必要的多，甚至退化成链表。

我们应该怎么做插入呢？

二叉搜索树的高度

Good news: BSTs have great performance if we insert items randomly. Performance is Θ(log N) per operation. -> 将所有要插入的数据随机插入会得到 Θ(log N) 的高度，换取 Θ(log N) 的性能。

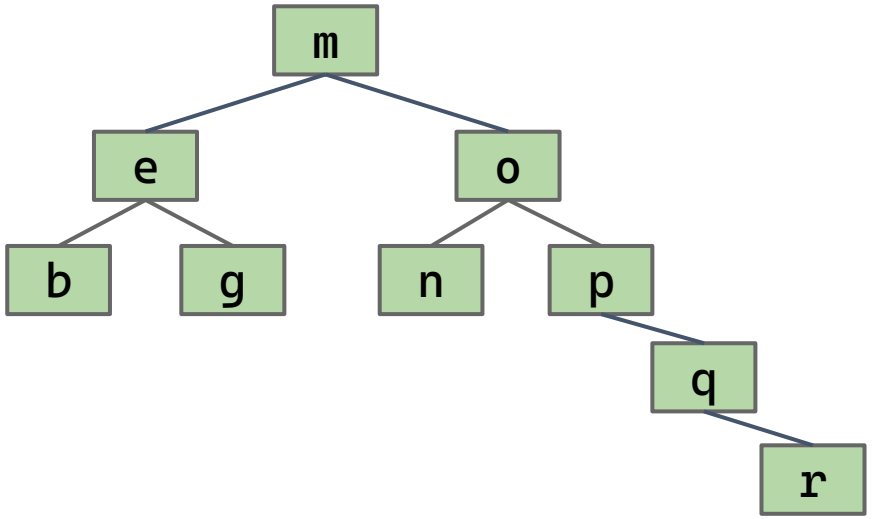
二叉搜索树的高度

Good news: BSTs have great performance if we insert items randomly. Performance is Θ(log N) per operation. -> 将所有要插入的数据随机插入会得到 Θ(log N) 的高度，换取 Θ(log N) 的性能。

Bad News: We can't always insert our items in a random order. Why?

- Data comes in over time, don't have all at once.
  - Example: Storing dates of events.
    - add("01-Jan-2019, 10:31:00")
    - add("01-Jan-2019, 18:51:00")
    - add("02-Jan-2019, 00:05:00")
    - add("02-Jan-2019, 23:10:00")

    -> 数据获取往往有先后性



平衡树

Lecture 2

二叉搜索树的高度

平衡树

B树

- 节点分裂
- 连锁分裂
- 正式定义
- 性质
- 性能

平衡树

> 改进自 BST 的，为了实现更高效的查询的一种搜索树

平衡性：

- 所有叶子的深度趋于平衡。
- 在树上所有可能查找的均摊复杂度偏低。
- 以 T 为根节点的树，每一个结点的左子树和右子树高度差最多为 1。

平衡树

> 改进自 BST 的，为了实现更高效的查询的一种搜索树

得到一颗具有平衡性的树：

已经有一颗不平衡的树 -> 经过调整

只有一个结点的树 -> 在插入，删除等操作中保持自平衡

平衡树

> 改进自 BST 的，为了实现更高效的查询的一种搜索树

得到一颗具有平衡性的树：
已经有一颗不平衡的树 -> 经过调整
只有一个结点的树 -> 在插入，删除等操作中保持自平衡

- AVL树
- B树
- 红黑树
- 伸展树

...

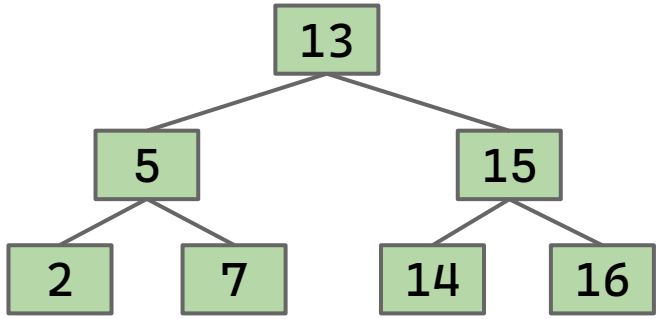
B树：节点分裂

Lecture 2

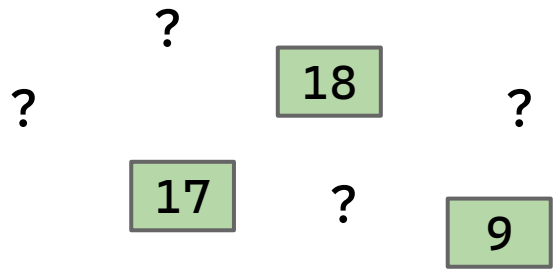
二叉搜索树的高度
平衡树
B树
- 节点分裂
- 连锁分裂
- 正式定义
- 性质
- 性能

拒绝不平衡！

假设有右图这样的一个已经平衡的平衡树，我们要插入元素。

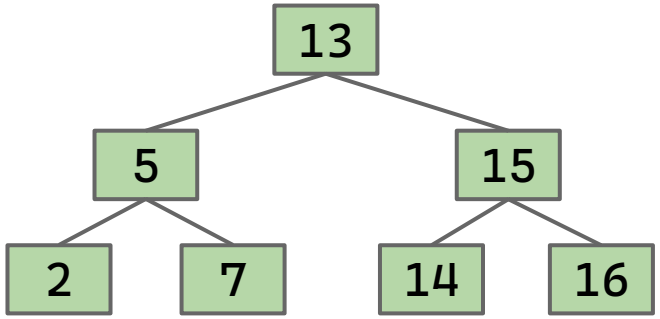


该如何插入？？



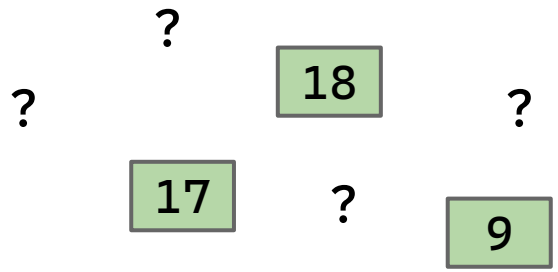
拒绝不平衡!

假设有右图这样的一个已经平衡的平衡树，我们要插入元素。



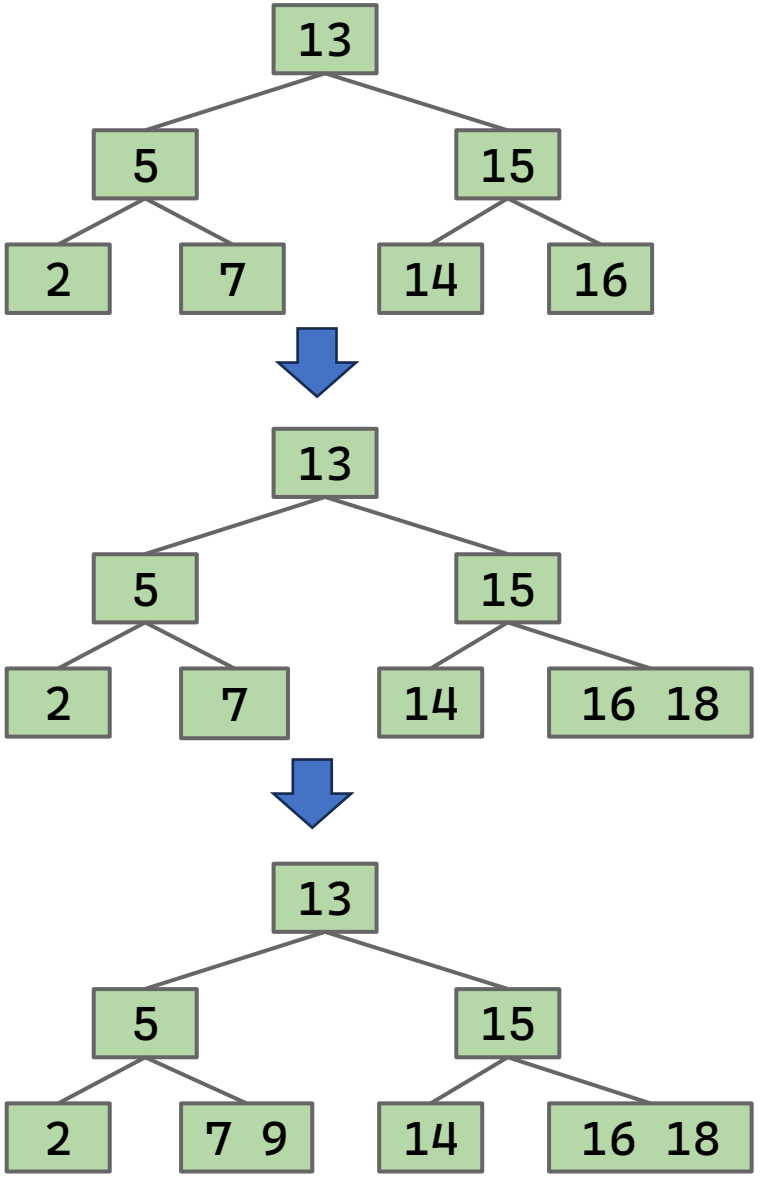
- 树能不能别变高?
  -> 那就不添加新结点了，这样树永远不会变高!

- 那新插入的元素放哪?



拒绝不平衡！

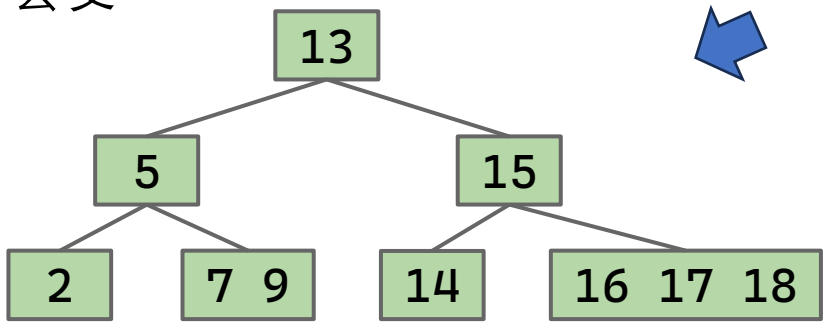
假设有右图这样的一个已经平衡的平衡树，我们要插入元素。



- 树能不能别变高？
  -> 那就不添加新结点了，这样树永远不会变高！

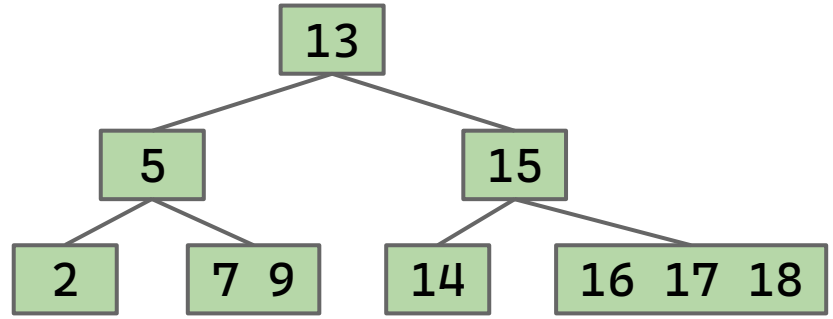
- 那新插入的元素放哪？
  -> 强制往叶子结点加元素，让叶子结点“过载”
    - 这种过载的树高度永远不会变

1. add(18)
2. add(9)
3. add(17)



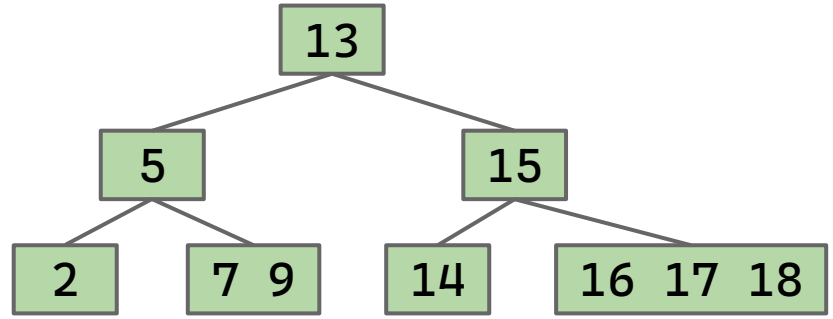
用“过载”拒绝不平衡

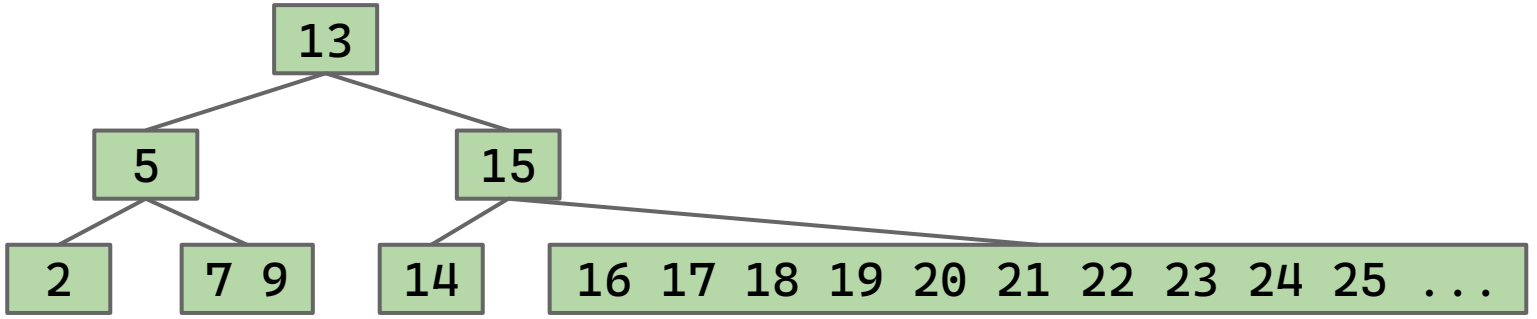
```
contains(18):
- Is 18 > 13? Yes, go right.
- Is 18 > 15? Yes, go right.
- Is 16 = 18? No.
- Is 17 = 18? No.
- Is 18 = 18? Yes! Found it.
```



用“过载”拒绝不平衡

```
contains(18):
- Is 18 > 13? Yes, go right.
- Is 18 > 15? Yes, go right.
- Is 16 = 18? No.
- Is 17 = 18? No.
- Is 18 = 18? Yes! Found it.
```

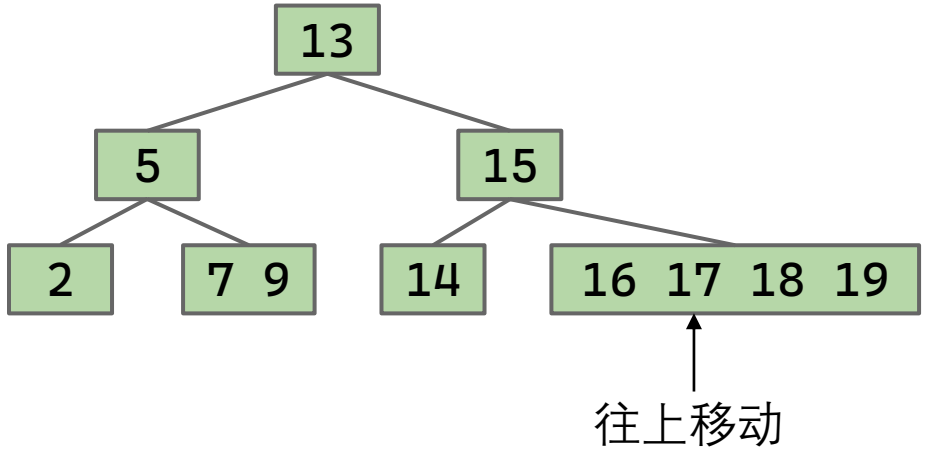




用“向上移动”解决过载树问题

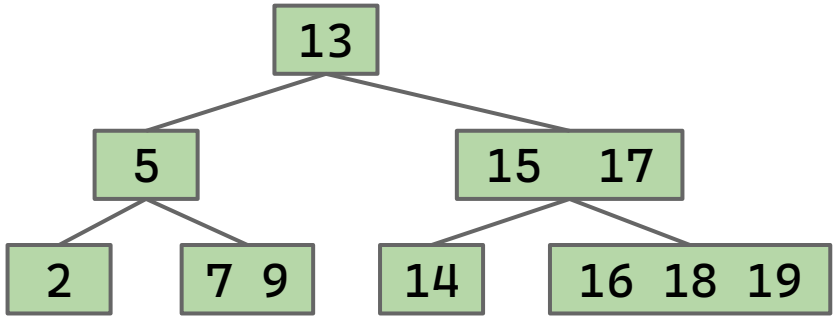
高度恒定了，但我们有新的问题：

- 叶结点装的太太太多了，这不还是步了退化链表的后尘？



解决方法？

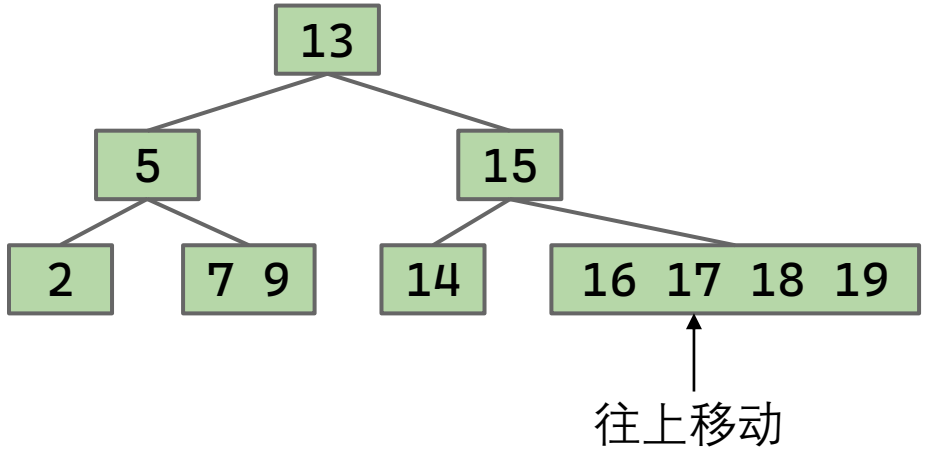
- 给结点包含的数据量设置一个限制 L，假设每个结点可以包含元素量的最大值为 L = 3。
  - 如果一个结点含有的元素超过了 L（如右图最右下角结点），就把这个结点的一个元素给父结点
    - 哪个元素？**我们规定是中间的元素（如果中间元素有两个，选取两个中偏左的）**



用“向上移动”解决过载树问题

高度恒定了，但我们有新的问题：

- 叶结点装的太太太多了，这不还是步了退化链表的后尘？

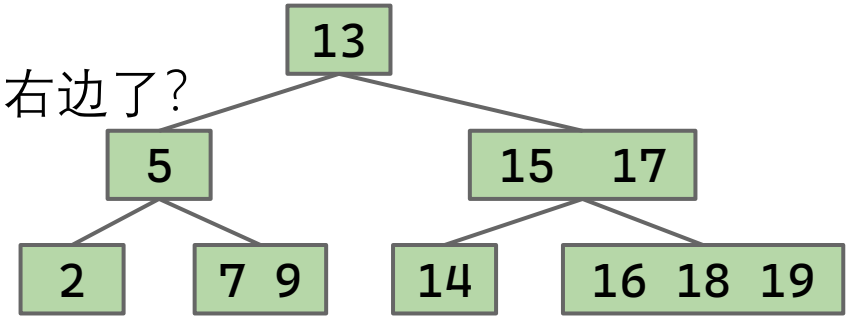


解决方法？

- 给结点包含的数据量设置一个限制 L，假设每个结点可以包含元素量的最大值为 L = 3。
  - 如果一个结点含有的元素超过了 L（如右图最右下角结点），就把这个结点的一个元素给父结点
    - 哪个元素？**我们规定是中间的元素（如果中间元素有两个，选取两个中偏左的）**

一个新的问题：16 比 17 小，怎么还放到含有 17 的父结点右边了？

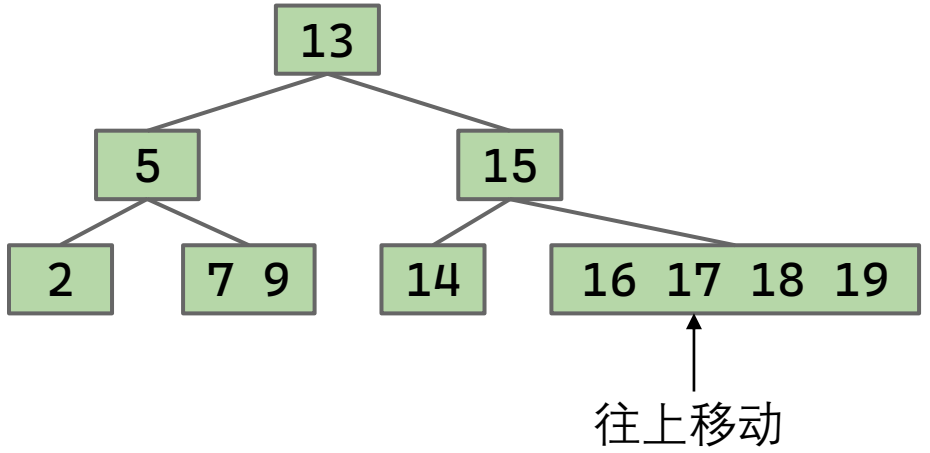
- 这该怎么解决？（不能修改向上移动哪个元素的规定）



用“分裂结点”解决过载树问题

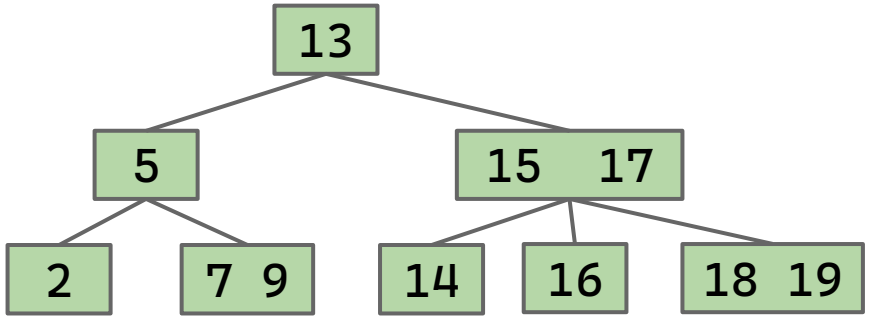
高度恒定了，但我们有新的问题：

- 叶结点装的太太太多了，这不还是步了退化链表的后尘？



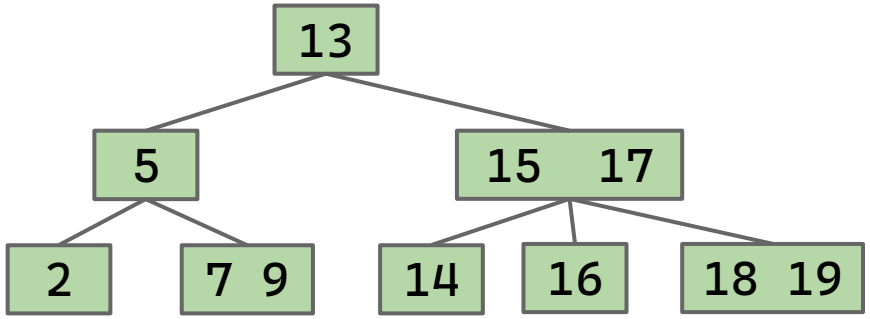
解决方法？

- 给结点包含的数据量设置一个限制 L，假设每个结点可以包含元素量的最大值为 L = 3。
  - 如果一个结点含有的元素超过了 L（如右图最右下角结点），就把这个结点的一个元素给父结点
    - 然后，我们将结点沿中间元素左右分裂（一边小于 17，一边大于 17）
    - 现在，父结点有三个子结点了！



用“分裂结点”解决过载树问题

```
contains(18):
- 18 > 13, so go right
- 18 > 15, so compare vs. 17
- 18 > 17, so go right
- found it!
```



每个结点最多有 O(L) 个元素，也就是说搜素时经过每个结点都最多在结点中遍历 L 个元素。

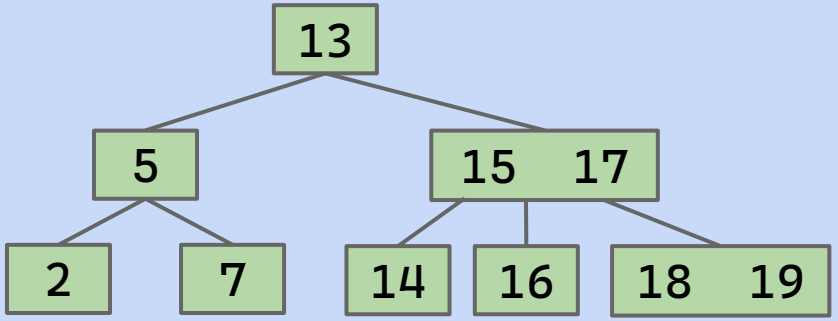
当我们搜索一个元素时，最差时间复杂度是 O(L * log(N))，但因为 L 是个常数，所以可以忽略，由此，最差时间复杂度仍然是 O(log(N))！

用“分裂结点”解决过载树问题

- 给结点包含的数据量设置一个限制 L，假设每个结点可以包含元素量的最大值为 L = 3。
  - 如果一个结点含有的元素超过了 L（如右图最右下角结点），就把这个结点的一个元素给父结点
    - 然后，我们将结点沿中间元素左右分裂（一边小于 17，一边大于 17）
    - 现在，父结点有三个子结点了!

练习：用我们刚刚讲到的策略，对下面这个树依次插入元素：20 21，结束插入后这棵树变成了什么样?

L（一个结点可以包含的最大元素数量）= 3

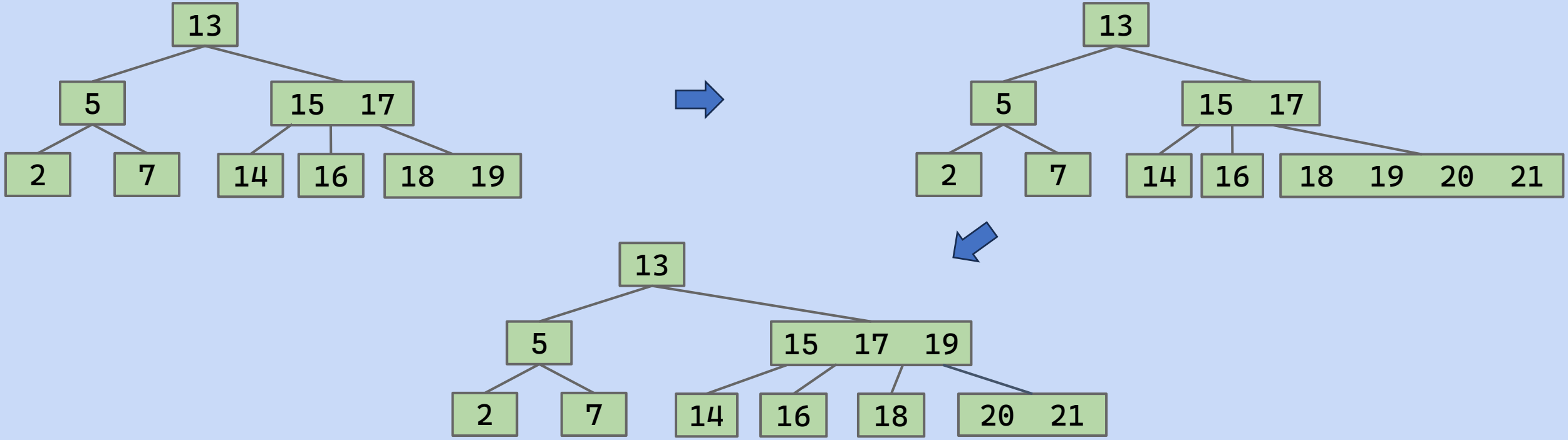


用“分裂结点”解决过载树问题

- 给结点包含的数据量设置一个限制 L，假设每个结点可以包含元素量的最大值为 L = 3。
  - 如果一个结点含有的元素超过了 L（如右图最右下角结点），就把这个结点的一个元素给父结点
    - 然后，我们将结点沿中间元素左右分裂（一边小于 17，一边大于 17）
    - 现在，父结点有三个子结点了!

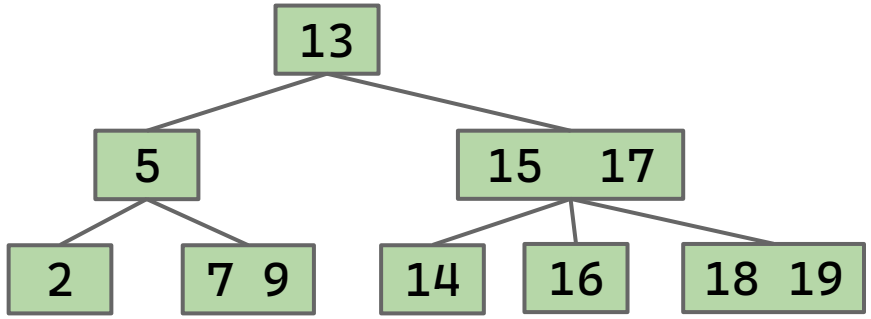
练习：用我们刚刚讲到的策略，对下面这个树依次插入元素：20 21，结束插入后这棵树变成了什么样?

L（一个结点可以包含的最大元素数量）= 3



用“分裂结点”解决过载树问题

```
contains(18):
- 18 > 13, so go right
- 18 > 15, so compare vs. 17
- 18 > 17, so go right
- found it!
```



每个结点最多有 `O(L)` 个元素，也就是说搜素时经过每个结点都最多在结点中遍历 `L` 个元素。

当我们搜索一个元素时，最差时间复杂度是 `O(L * log(N))`，但因为 `L` 是个常数，所以可以忽略，由此，最差时间复杂度仍然是 `O(log(N))`！

看看如果非叶子结点过载了，该怎么分裂？

B树：连锁分裂

Lecture 2

二叉搜索树的高度

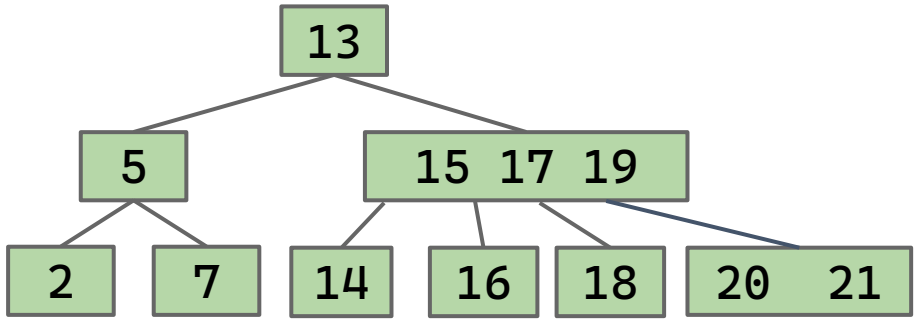
平衡树

B树

- 节点分裂
- 连锁分裂
- 正式定义
- 性质
- 性能

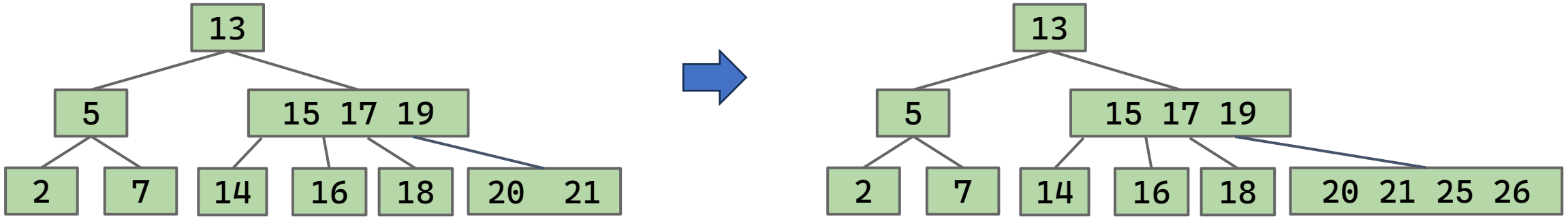
连锁分裂

在练习题的树的基础上再插入元素：25 26



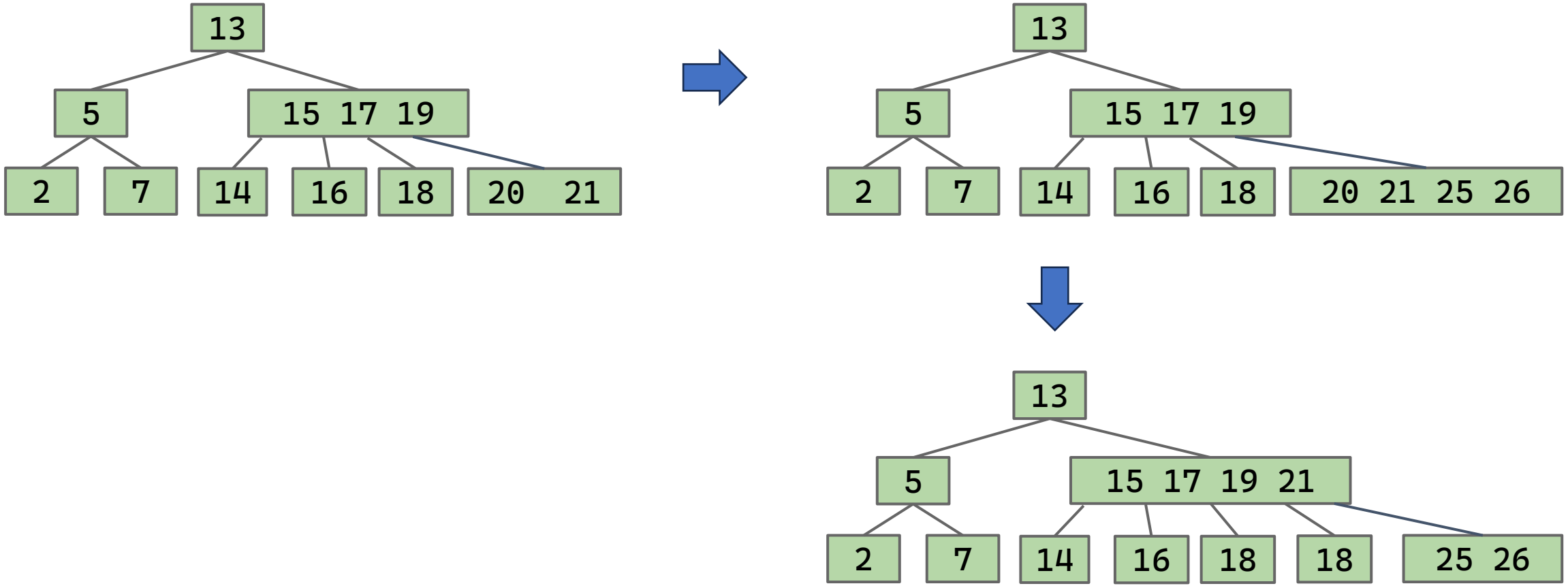
连锁分裂

在练习题的树的基础上再插入元素：25 26



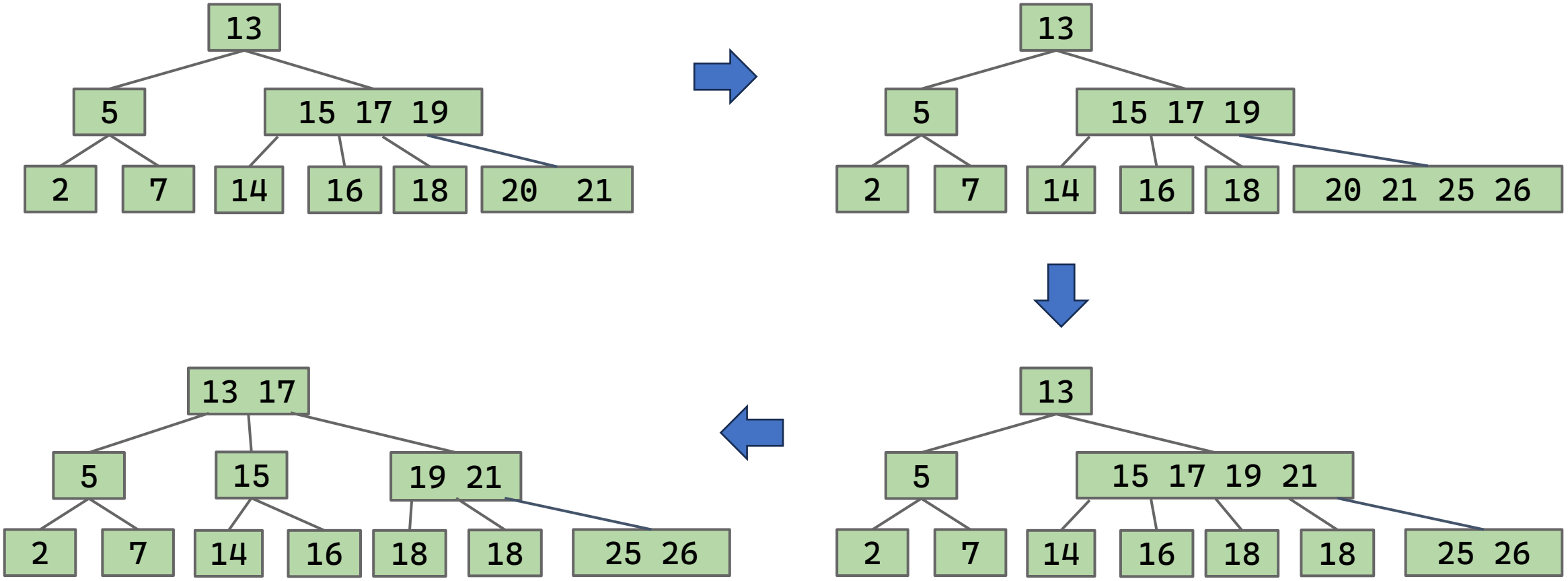
连锁分裂

在练习题的树的基础上再插入元素：25 26



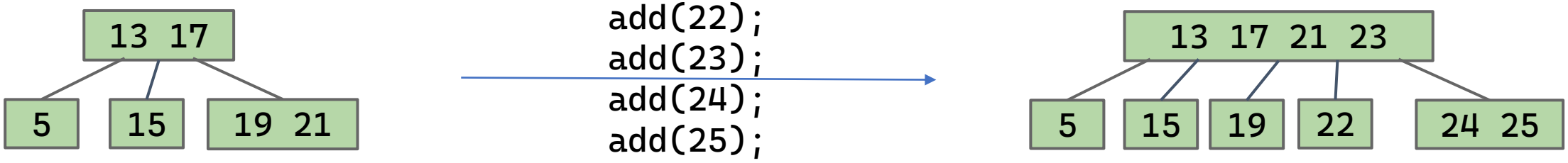
连锁分裂

在练习题的树的基础上再插入元素：25 26。（L = 3）



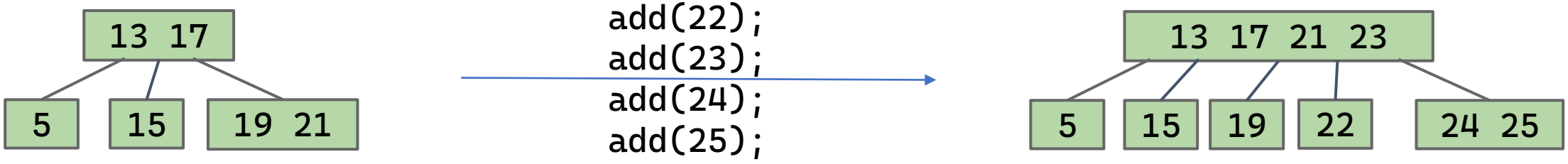
根结点分裂

经过下面的插入操作后，根结点也过载了，这下怎么办？（L = 3）



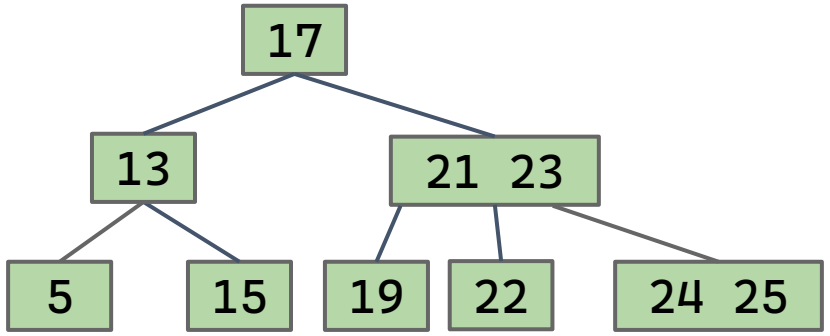
根结点分裂

经过下面的插入操作后，根结点也过载了，这下怎么办？（L = 3）



继续遵守**分裂法则**，但是事已至此了，只能增加树的高度了：

恭喜 **17**，成为了新的根结点！



B树：正式定义

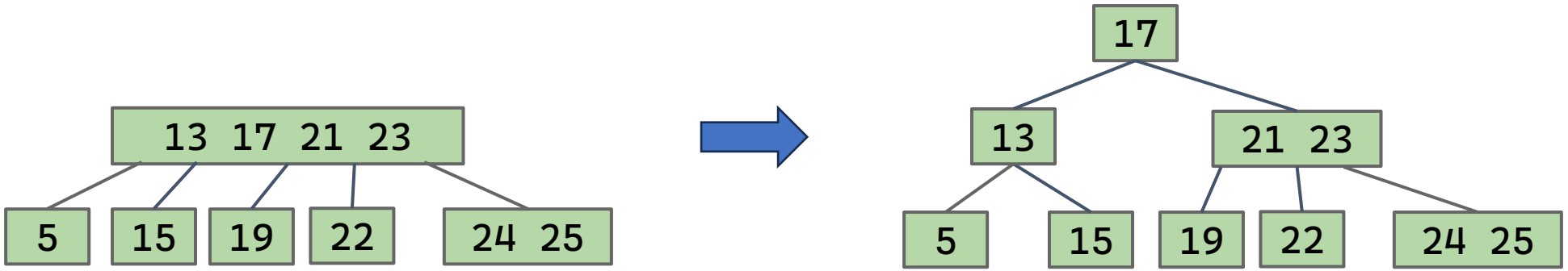
Lecture 2

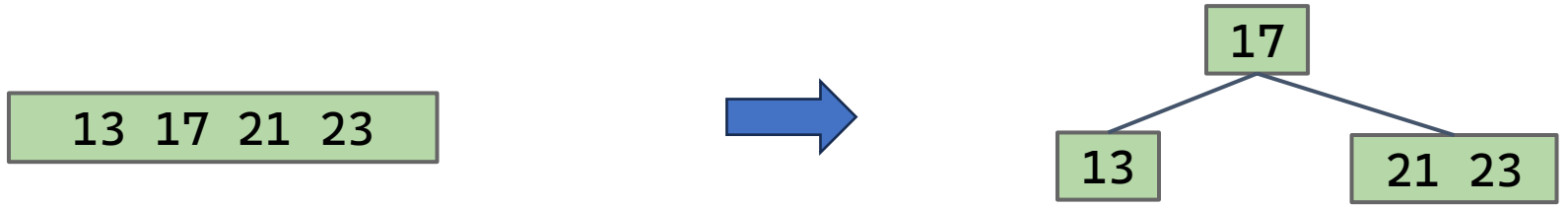
二叉搜索树的高度
平衡树
B树
- 节点分裂
- 连锁分裂
- 正式定义
- 性质
- 性能

回顾

我们刚刚在演示中介绍了如何在这种（L = 3的）树中插入元素。

- 当插入的元素最终确定在了某个非根结点的结点中时，高度不会变化。
- 高度当且仅当我们分裂根结点时会增长，**且所有叶子结点都会因此加深一层**：





B-Tree Visualization (usfca.edu)

这种树，就是B树

讲了这么多，是时候解释这种“分裂树”真面纱，就是B树。B树的建立需要指定每个结点最多可以包含的值得数量 L。

m阶B树，m表示这个树的每一个结点最多可以拥有的子结点个数。M = L + 1。

- L = 3 的B树（四阶B树）也就是我们今天举例子用到的，也叫 2-3-4树，或者 2-4树。
  - 这个名称的意思是：一个结点可以有 2，3 或 4 个子结点（当然可以有1，2）。
- L = 2 的B树（三阶B树），也可以被叫做 2-3树。

这种树，就是B树

讲了这么多，是时候解释这种“分裂树”真面纱，就是B树。B树的建立需要指定每个结点最多可以包含的值得数量 L。

m阶B树，m表示这个树的每一个结点最多可以拥有的子结点个数。M = L + 1。

- L = 3 的B树（四阶B树）也就是我们今天举例子用到的，也叫 2-3-4树，或者 2-4树。
  - 这个名称的意思是：一个结点可以有 2，3 或 4 个子结点（当然可以有1，2）。
- L = 2 的B树（三阶B树），也可以被叫做 2-3树。

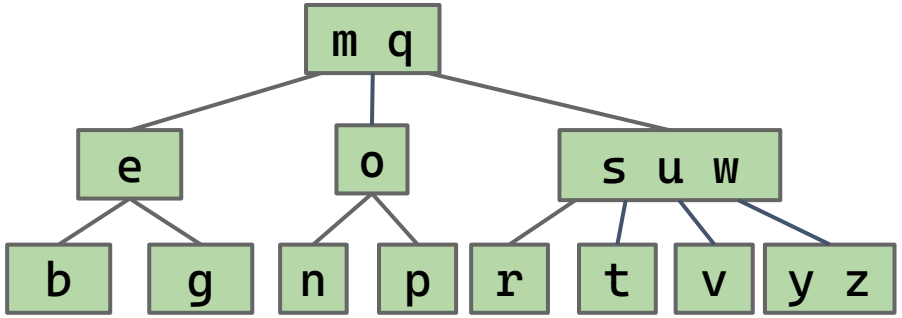
The origin of "B-tree" has never been explained by the authors. As we shall see, "balanced," "broad," or "bushy" might apply. Others suggest that the "B" stands for Boeing. Because of his contributions, however, it seems appropriate to think of B-trees as "Bayer"-trees.

- Douglas Corner (The Ubiquitous B-Tree)

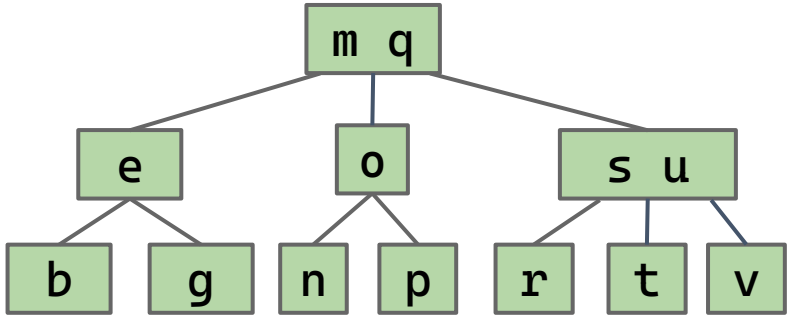
这种树，就是B树

B树一般在以下两种语境中很常见：

- L 比较小（L=2 or L=3）的时候：
  - 常被用作演示简单的平衡搜索树（像我们今天这样）
- L 比较大（成百上千）的时候：
  - 可以做操作系统的文件索引和数据库索引



2-3-4 a.k.a. 2-4 Tree (L=3):
Max 3 items per node.
Max 4 non-null children per node.



2-3 Tree (L=2):
Max 2 items per node.
Max 3 non-null children per node.

B树：性质

Lecture 2

二叉搜索树的高度

平衡树

B树

- 节点分裂
- 连锁分裂
- 正式定义
- 性质
- 性能

B树：性质

B树的性质有很多，我们在这提供两个很棒的性质，可以帮助你更深一步了解B树：

- `All leaves must be the same distance from the root.` -> 所有的叶子结点深度（距离根节点的边数）相等

- `A non-leaf node with k items must have exactly k+1 children.` -> 一个非叶子结点的有 `k` 个元素的结点，一定有 `k+1` 个子结点

B树：性质

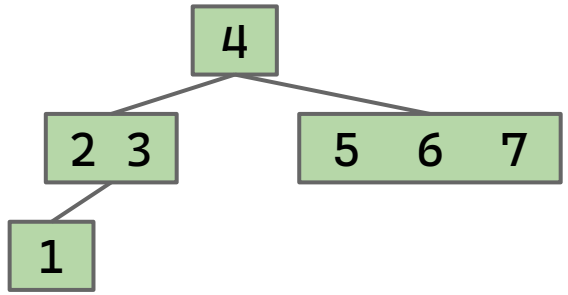
B树的性质有很多，我们在这提供两个很棒的性质，可以帮助你更深一步了解B树：

- All leaves must be the same distance from the root. -> 所有的叶子结点深度（距离根节点的边数）相等

- A non-leaf node with k items must have exactly k+1 children. -> 一个非叶子结点的有 k 个元素的结点，一定有 k+1 个子结点

Example: The tree given below is impossible.

- Leaves ([1] and [5 6 7]) are a different distance from the source.
- Non-leaf node [2 3] has two items but only one child. Should have three children.



B树：性质

B树的性质有很多，我们在这提供两个很棒的性质，可以帮助你更深一步了解B树：

- `All leaves must be the same distance from the root.` -> 所有的叶子结点深度（距离根节点的边数）相等

- `A non-leaf node with k items must have exactly k+1 children.` -> 一个非叶子结点的有 `k` 个元素的结点，一定有 `k+1` 个子结点

以上两个性质，确保了B树一定是“茂盛”的（感性的认知）。

B树：性能

Lecture 2

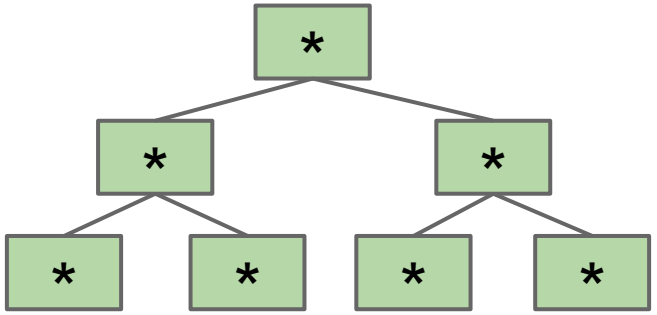
二叉搜索树的高度
平衡树
B树
- 节点分裂
- 连锁分裂
- 正式定义
- 性质
- 性能

B树的高度

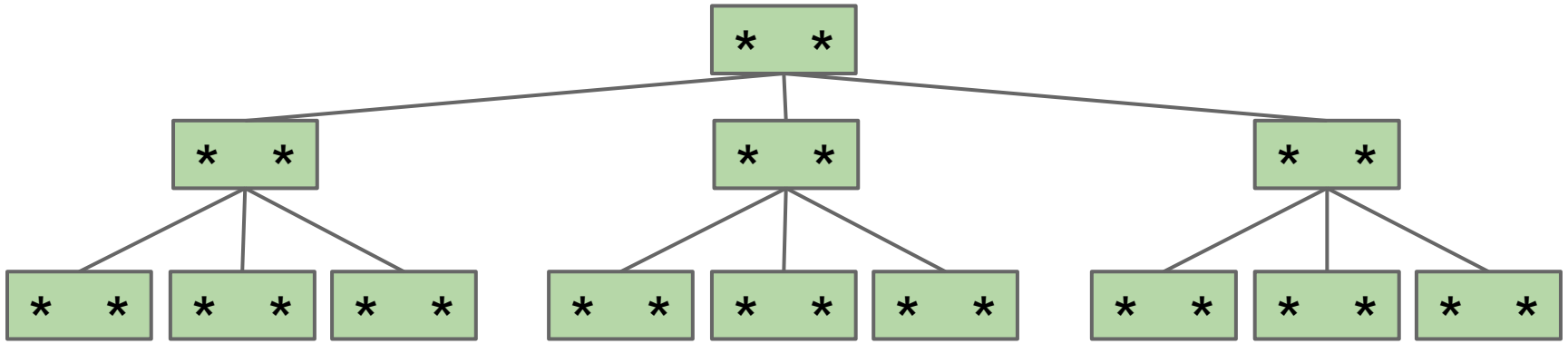
L：每个结点最多可以包含的值（示例中 L = 2）

高度：Between $\Theta(\log_{L+1}(N))$ and $\Theta(\log_2(N))$

- 最坏情况：所有非叶子结点只有一个值（右图）
- 最好情况：所有结点都有 L 个值（下图）
- 所以，B树的渐进高度就是 $\Theta(\log N)$



N: 8 items
L: 2 max per node
H: 2
Height grows with $\log_2(N)$



N: 26 items
L: 2 max per node
H: 2
Height grows with $\log_3(N)$

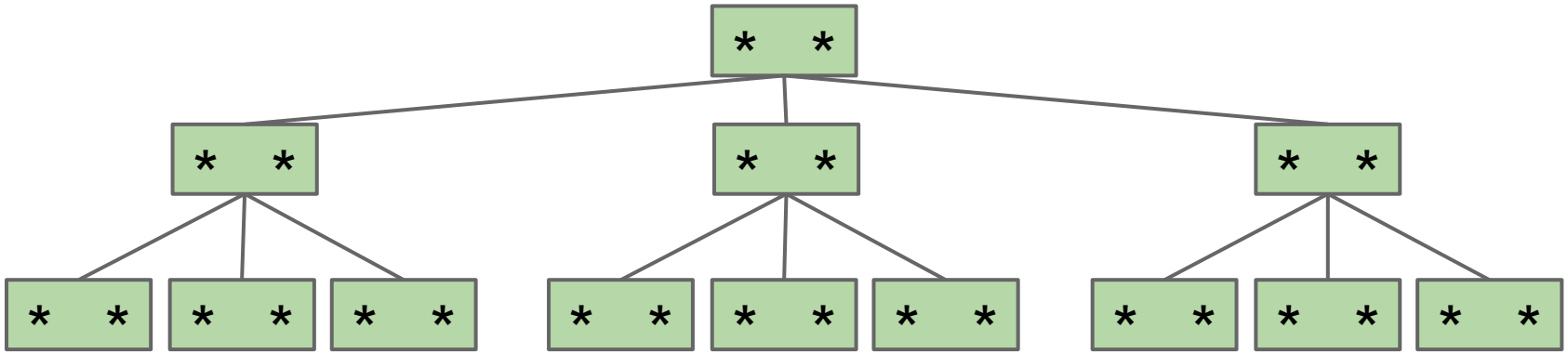
查找的时间复杂度

查找操作的时间复杂度

- 最坏情况下需要查找的结点数：H + 1
- 最坏情况下每个结点内需要查看的元素个数：L
- 整体时间复杂度：O(HL)

因为 H = Θ(log N)，整体时间复杂度：O(L log N)

- 因为 L 是个常数值，所以查找操作的时间复杂度是：O(log N).



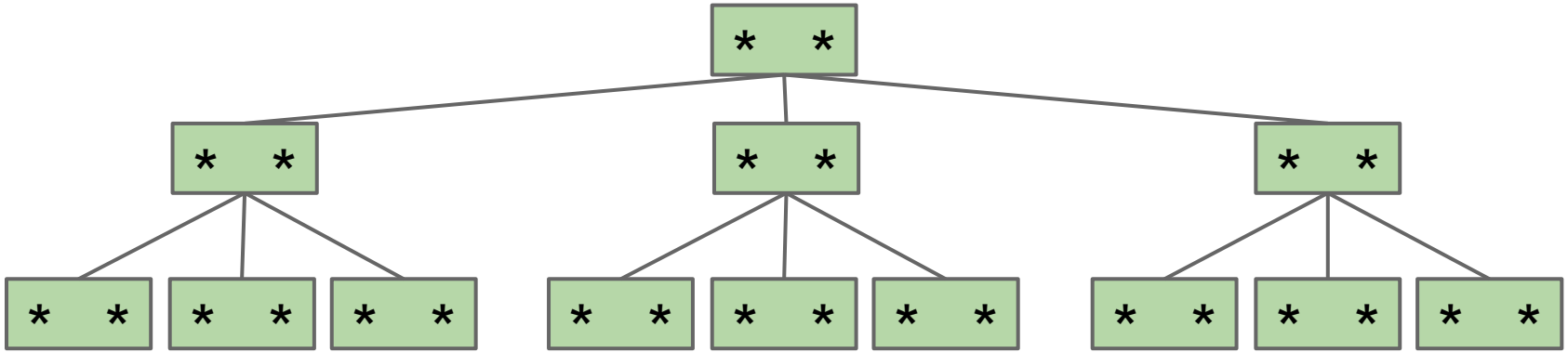
添加元素的时间复杂度

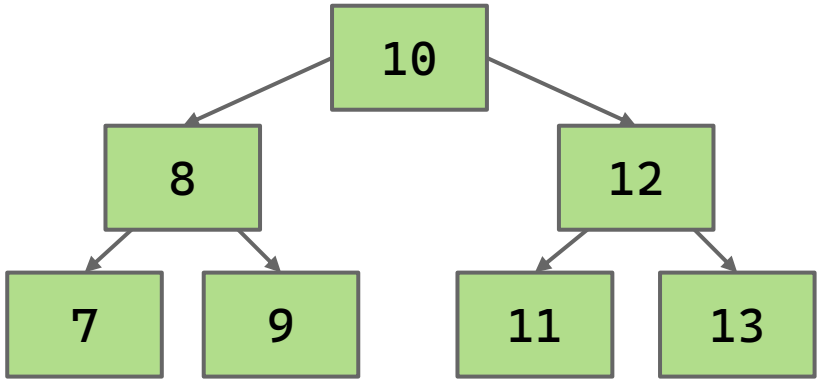
添加元素操作的时间复杂度

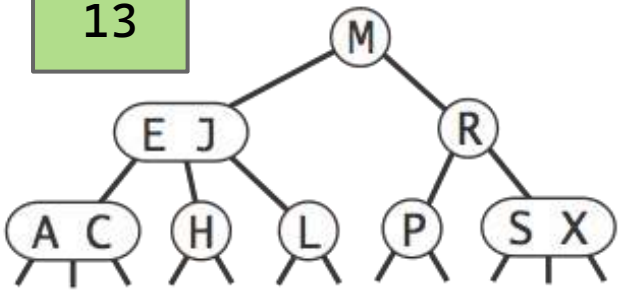
- 最坏情况下需要查找的结点数：H + 1
- 最坏情况下每个结点内需要查看的元素个数：L
- 最坏情况下的分裂结点次数：H + 1
- 整体时间复杂度：O(HL)

因为 H = Θ(log N)，整体时间复杂度：O(L log N)

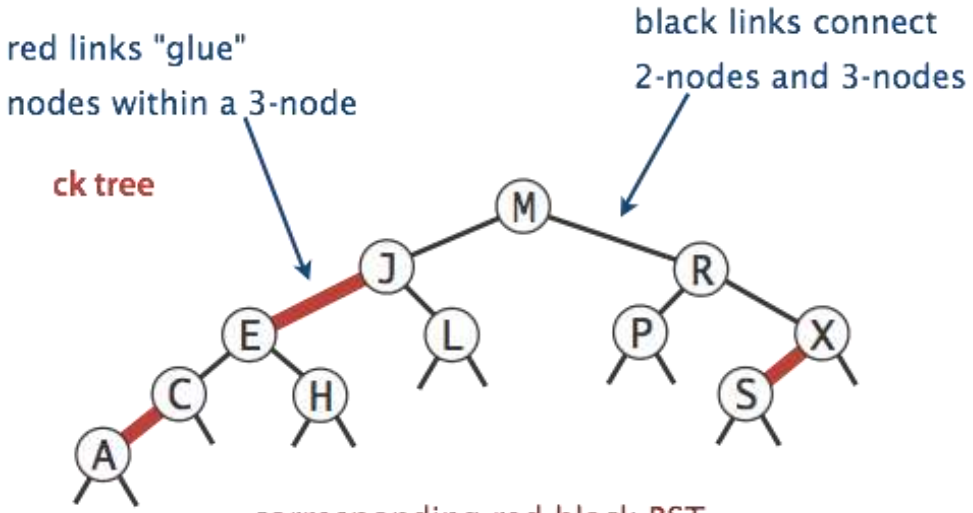
- 因为 L 是个常数值，所以查找操作的时间复杂度是：O(log N).







2-3 tree



corresponding red-black BST

谢谢大家

主讲人：七海Nana7mi

课程大纲：CS61B