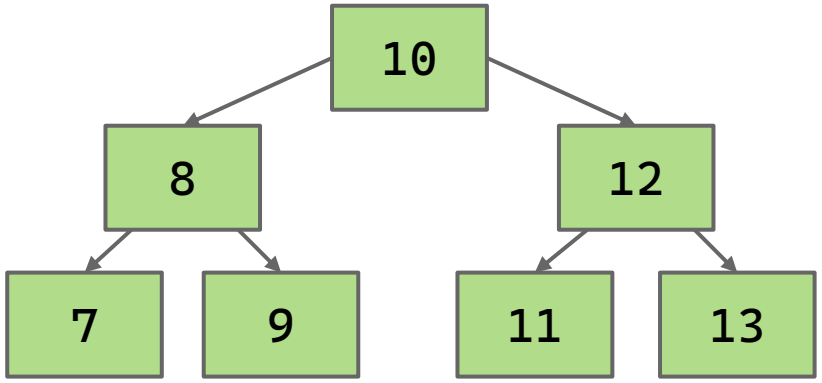
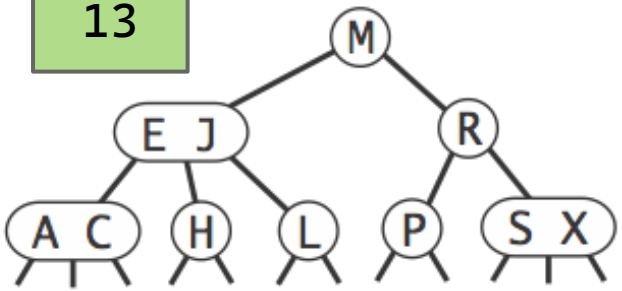
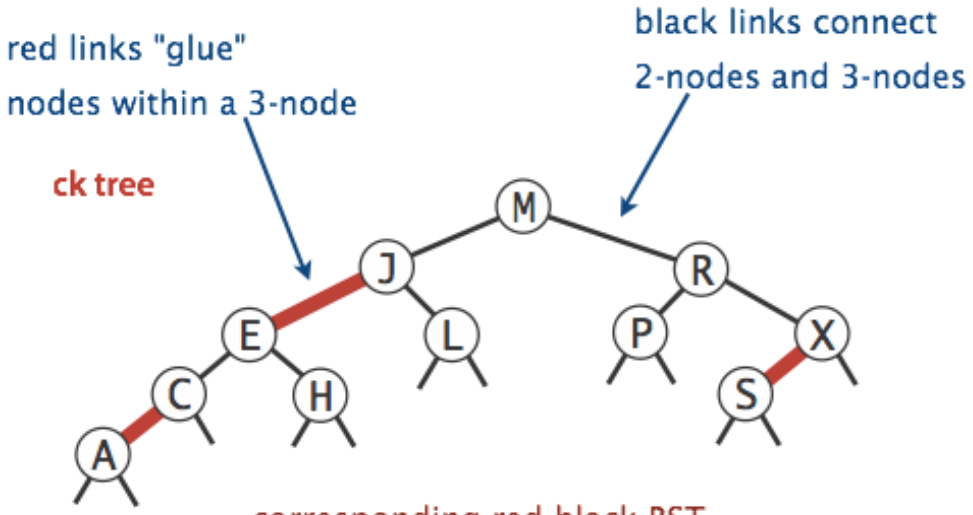


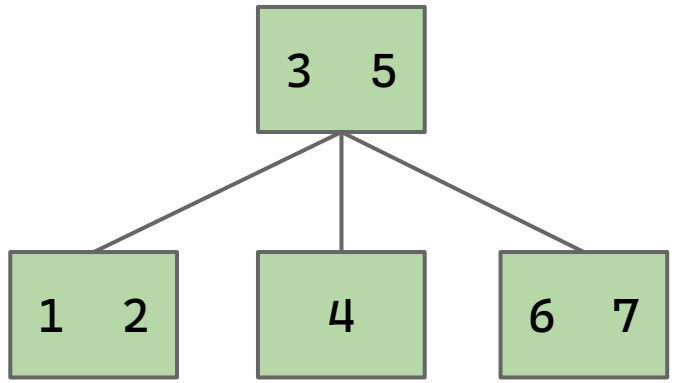


2-3 tree



corresponding red-black BST

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



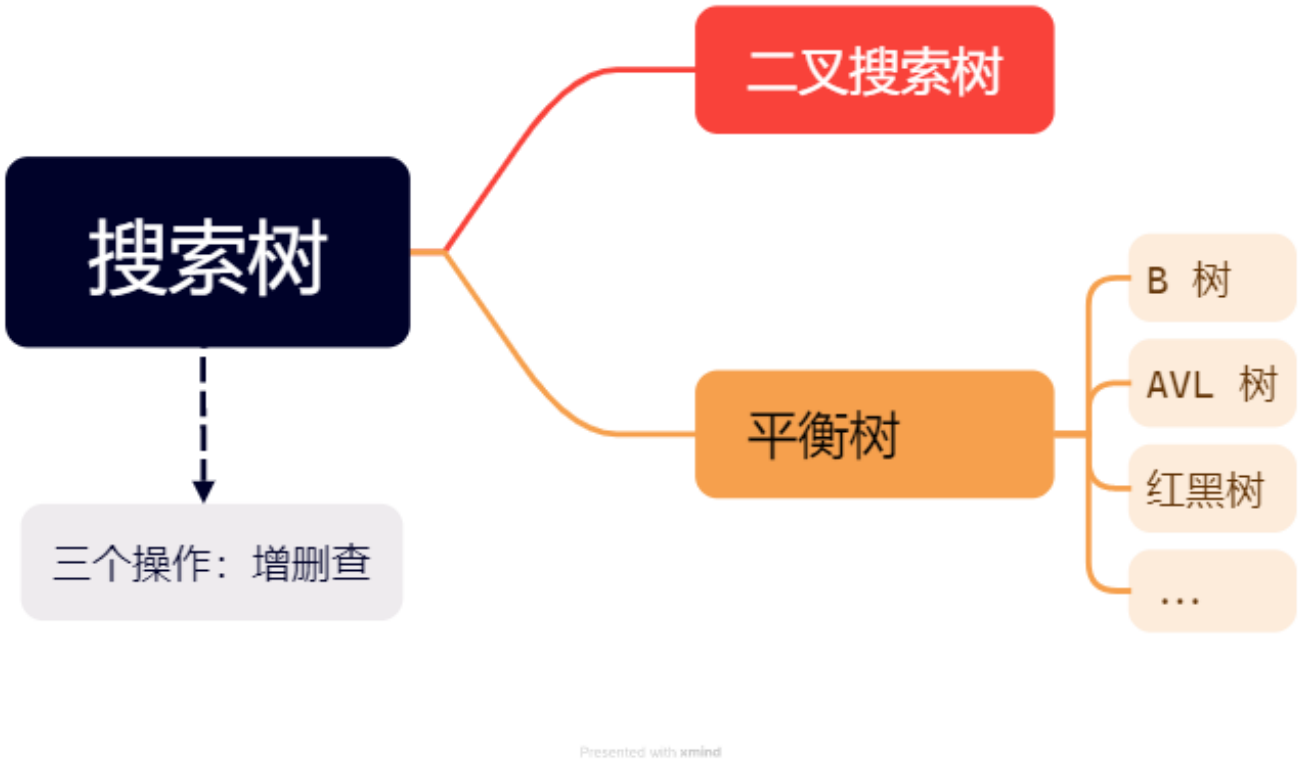
主讲人：七海Nana7mi

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！

章节目录



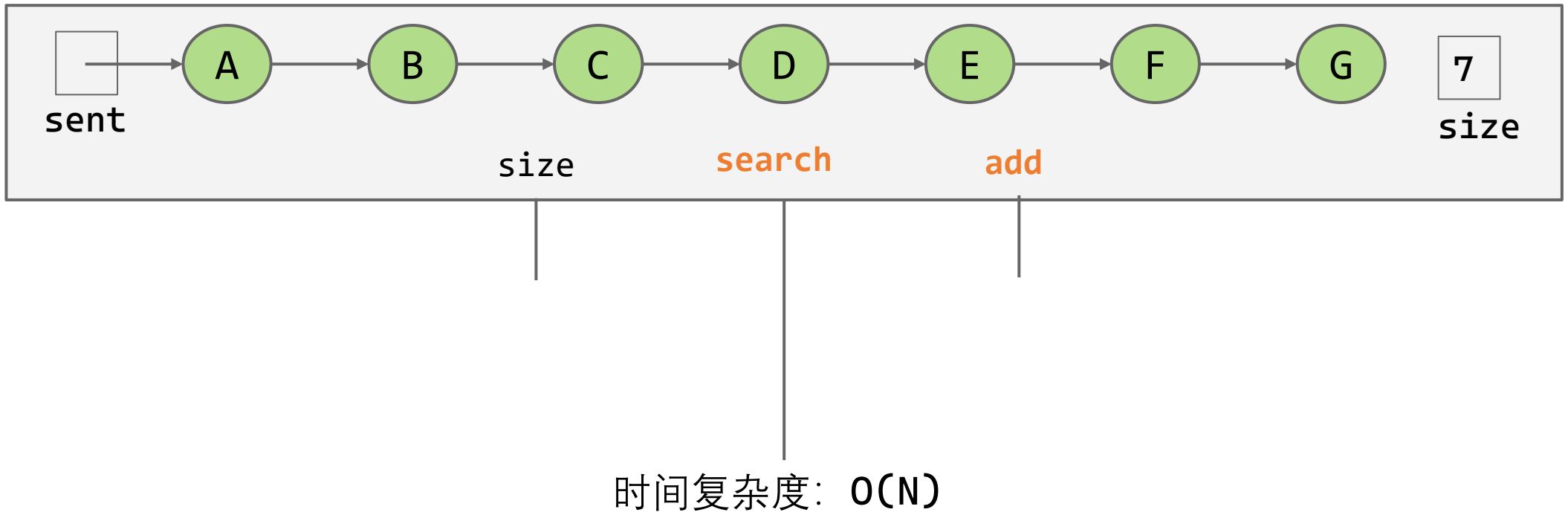
二叉搜索树：导入

Lecture 1

二叉搜索树
- 导入
- 定义
- `contains()`
- `insert()`
- `delete()`

二叉搜索树的应用

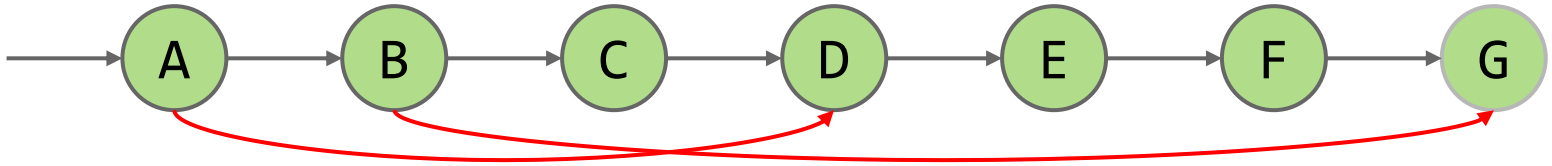
For the *order linked list* implementation below, an operation of search can take worst case linear time, Θ(N).



- How to do?

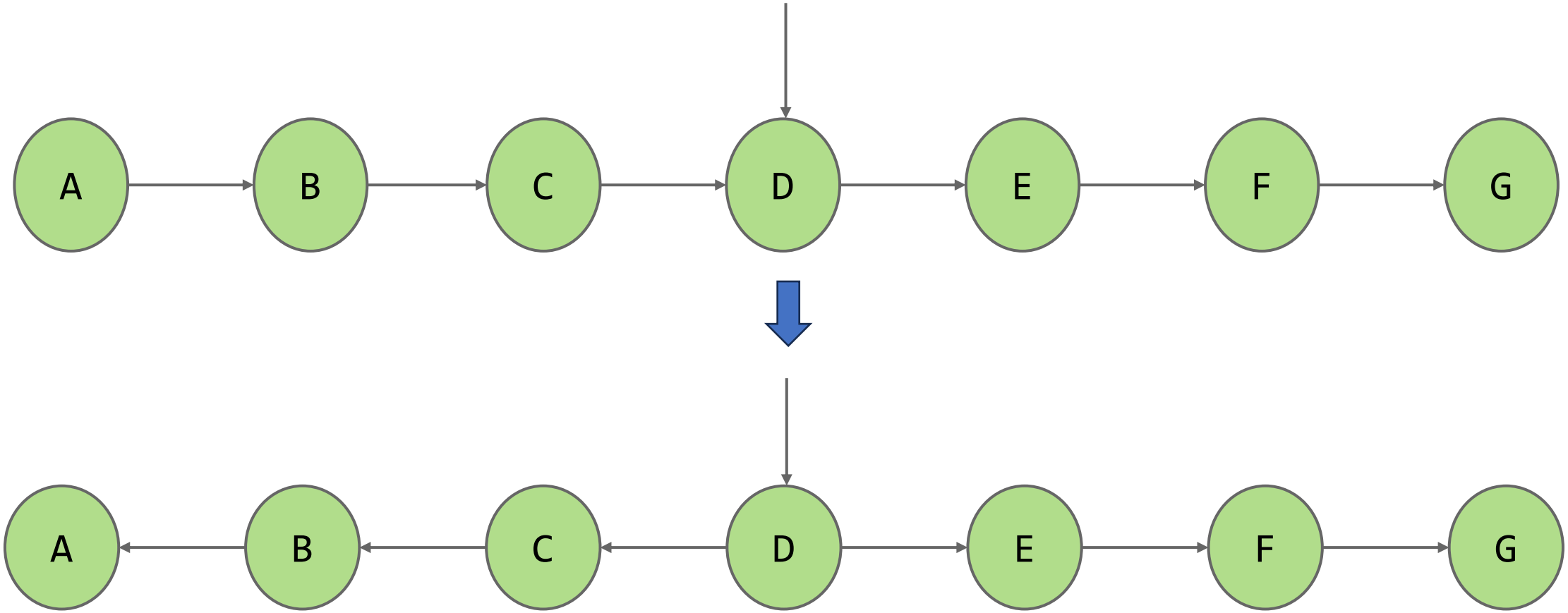
Fundamental Problem:
Slow search, even though it’s in order.

- 我们可以任意的增加不同元素之间的连接线，来缩短路程



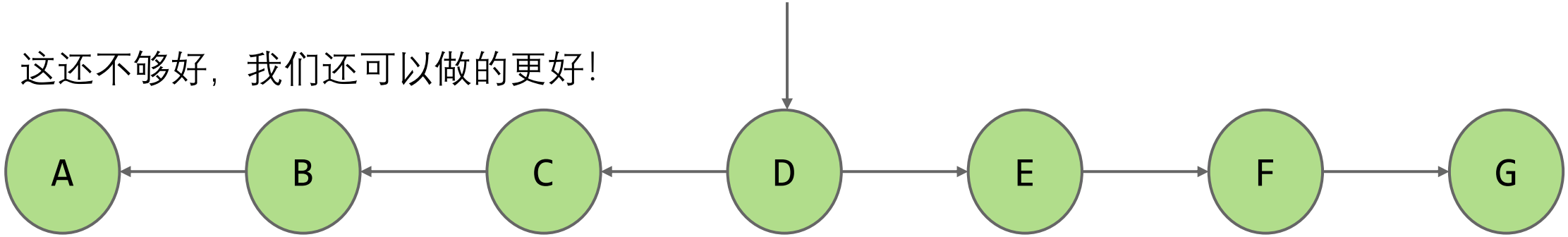
这种方法可以改进为一中数据结构也被叫做跳跃列表，我们对他的讲解讲止步于此，感兴趣的脆鲨可以自行搜索。

- 利用有序性，我们可以把入口指针指向中间的元素



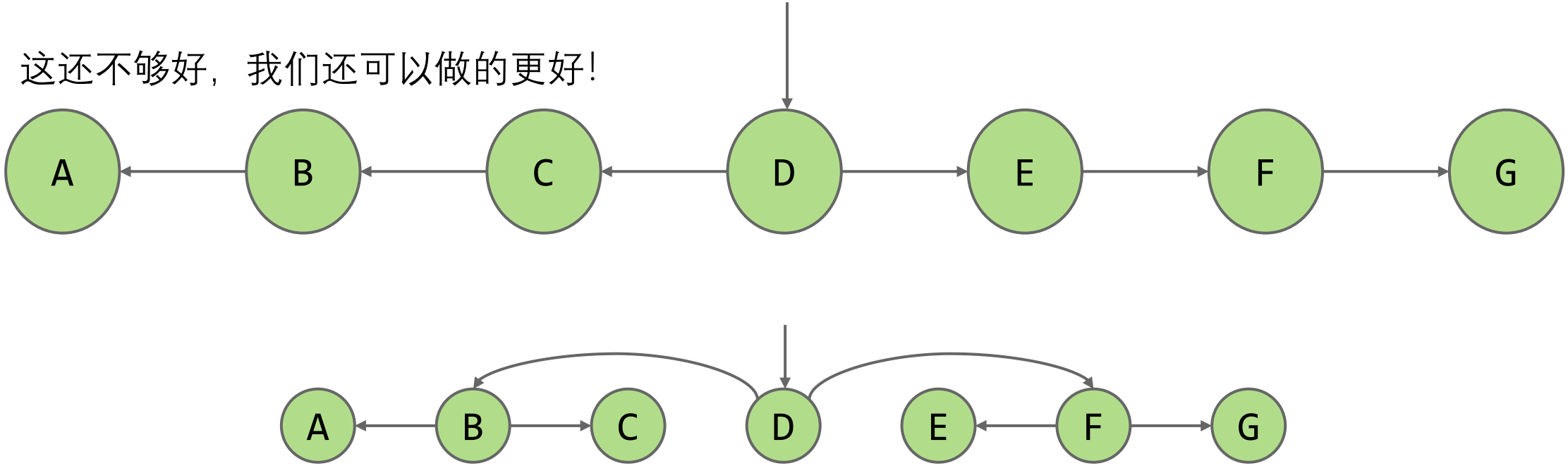
- 利用有序性，我们可以把入口指针指向中间的元素

这还不够好，我们还可以做的更好!



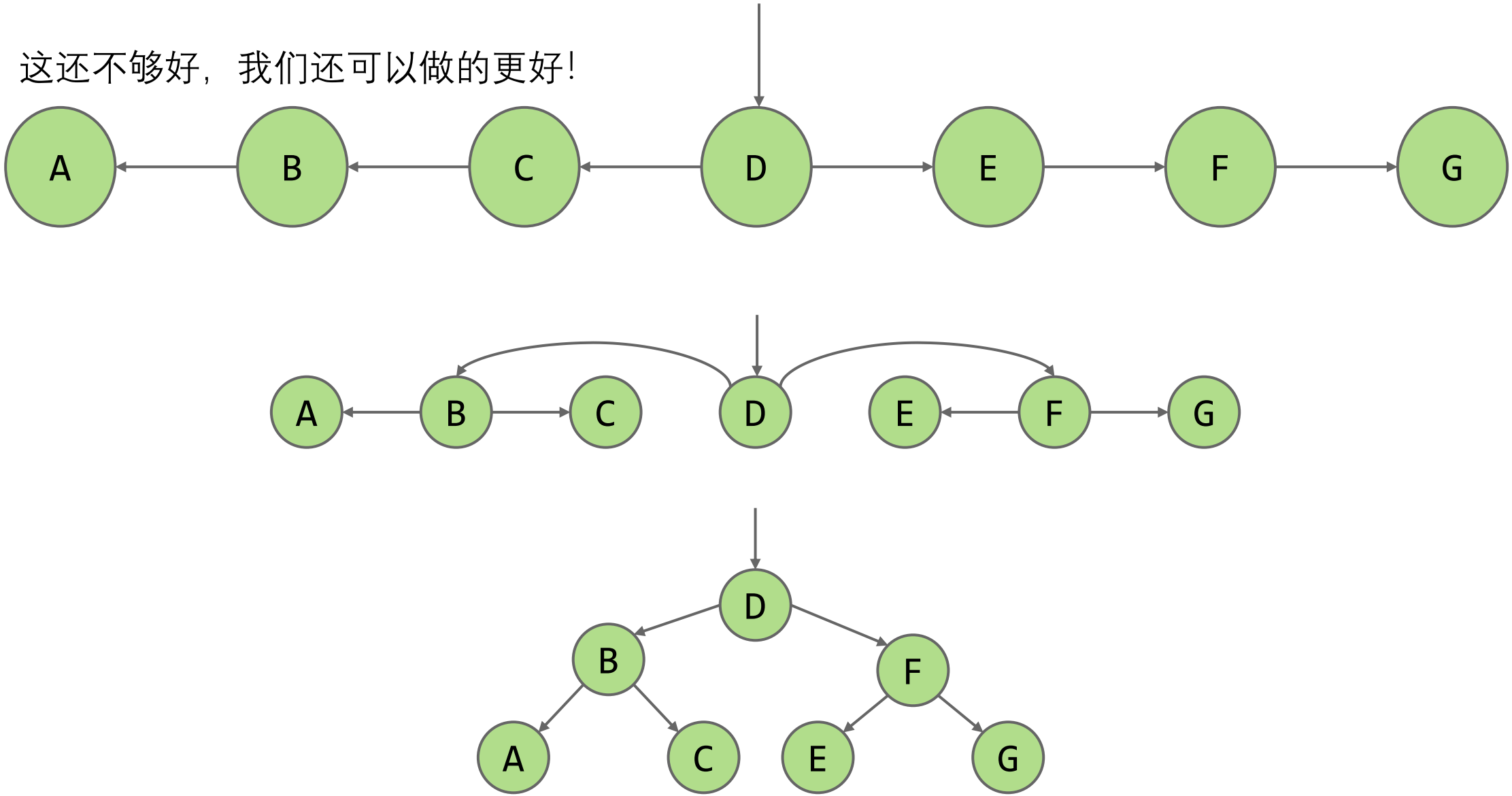
- 利用有序性，我们可以把入口指针指向中间的元素

这还不够好，我们还可以做的更好!



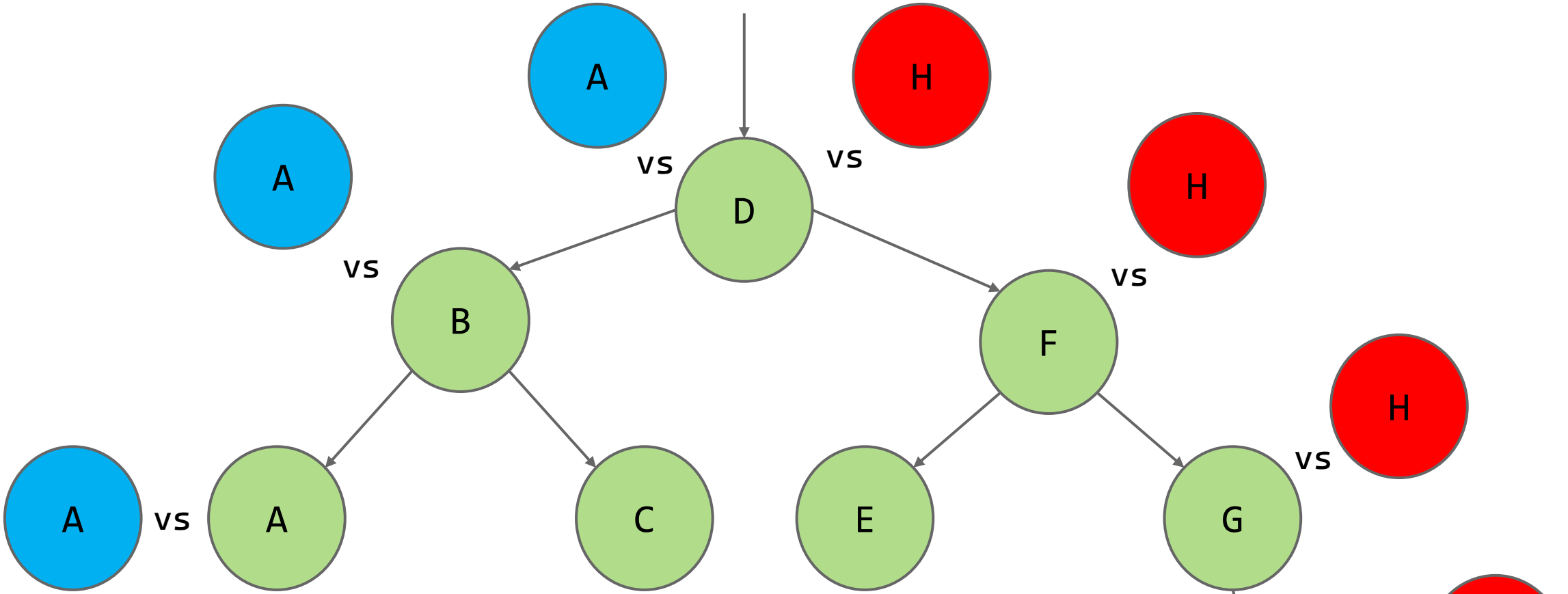
- 利用有序性，我们可以把入口指针指向中间的元素

这还不够好，我们还可以做的更好!



contains('A')?

contains('H')?



return true;

return false;

二叉搜索树：定义

Lecture 1

二叉搜索树
- 导入
- 定义
- `contains()`
- `insert()`
- `delete()`

二叉搜索树的应用

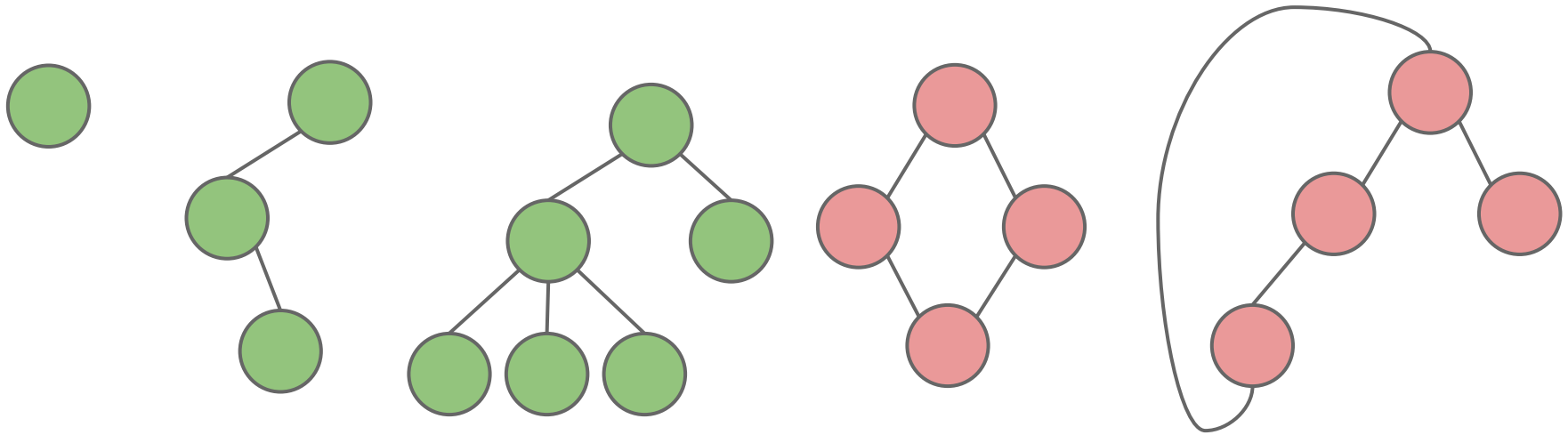
树:

A tree consists of:
- A set of nodes. -> 有限个结点的集合
- A set of edges that connect those nodes. -> 有有限个边相互连接
  - Constraint: There is exactly one path between any two nodes.

树:

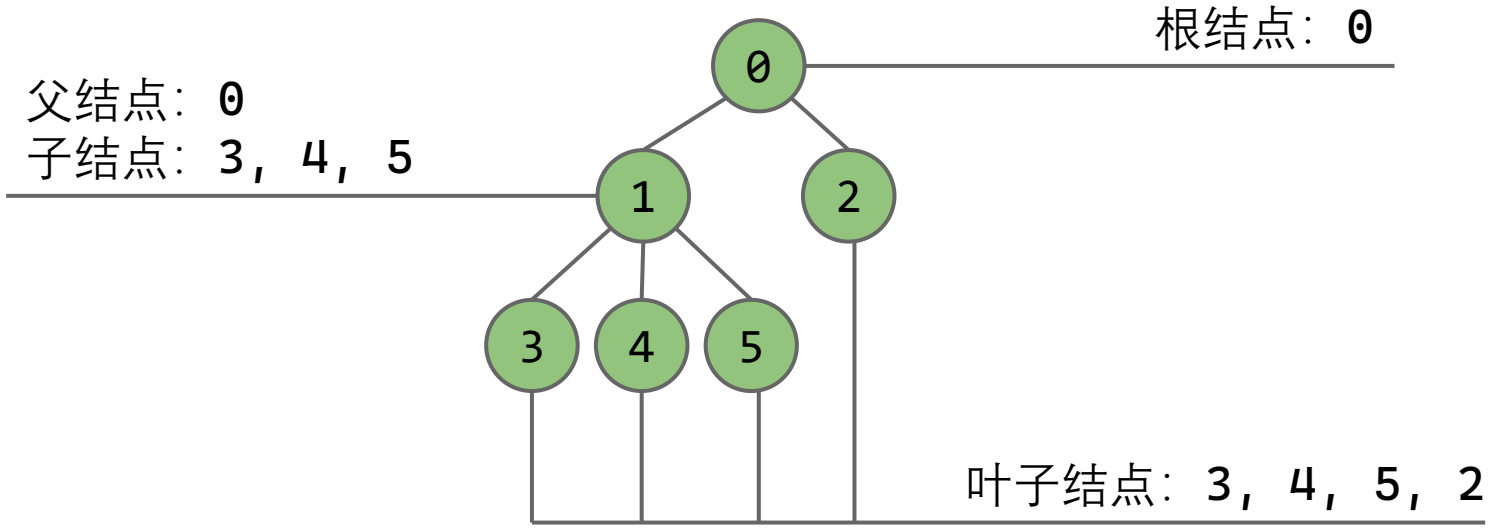
A tree consists of:

- A set of nodes. -> 有限个结点的集合
- A set of edges that connect those nodes. -> 有有限个边相互连接
  - Constraint: There is exactly one path between any two nodes.



树 & 二叉树

- Every node N except the root has exactly one parent. -> 没有父结点的结点就是根
- the root is usually depicted at the top of the tree. -> 根结点常常表示在最上方
- A node with no child is called a leaf. -> 没有子结点的结点称为叶结点



树 & 二叉树

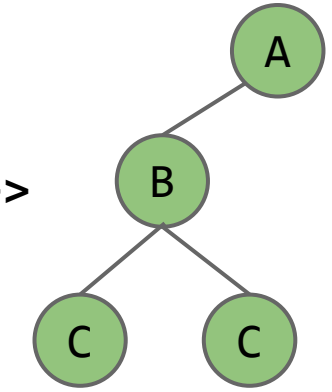
- Every node N except the root has exactly one parent. -> 没有父结点的结点就是根
- the root is usually depicted at the top of the tree. -> 根结点常常表示在最上方
- A node with no child is called a leaf. -> 没有子结点的结点称为叶结点

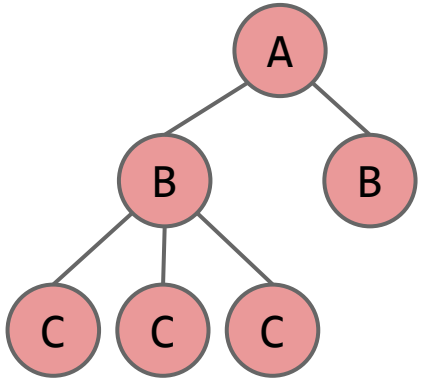
In a binary tree, every node has either 0, 1, or 2 children (subtrees).

Simple Java Code:

```
class Tree {
  int key;
  Tree leftPtr;
  Tree rightPtr;
}
```

Binary! ->





<- Not binary!

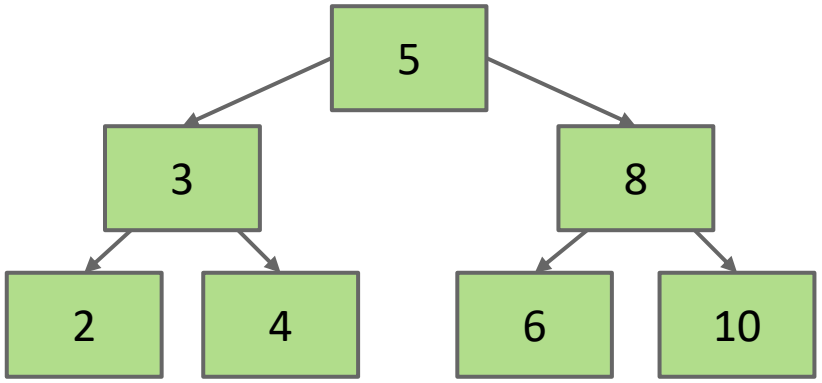
二叉搜索树

二叉搜索树就是在二叉树的定义基础上再增加一些条件限制：

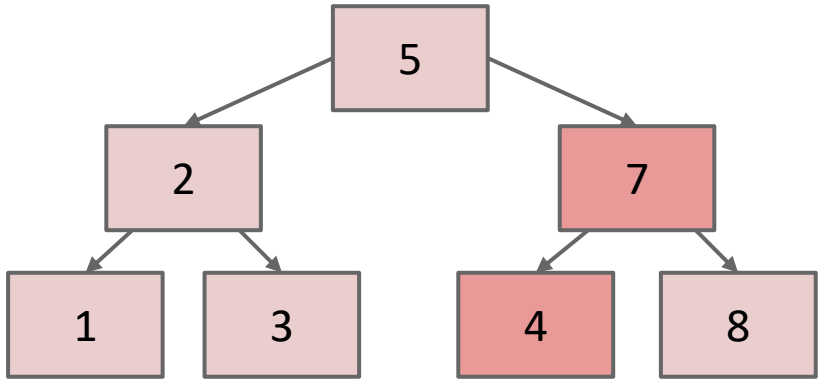
For every node X in the tree: -> 对于树中所有结点：

- Every key in the left subtree is less than X's key. -> 一个结点左子树的所有结点的值小于这个结点的值

- Every key in the right subtree is greater than X's key. -> 一个结点右子树的所有结点的值大于这个结点的值



Binary Search Tree



Binary Tree,
but not a Binary Search Tree

二叉搜索树

二叉搜索树就是在二叉树的定义基础上再增加一些条件限制：

For every node X in the tree: -> 对于树中所有结点：

- Every key in the left subtree is less than X's key. -> 一个结点左子树的所有结点的值小于这个结点的值

- Every key in the right subtree is greater than X's key. -> 一个结点右子树的所有结点的值大于这个结点的值

推论：二叉树不能有相同的值，结点的值之间没有相等的情况。

二叉搜索树的类定义

```
class BST<V> {
    V val;
    BST left;
    BST right;

    public BST(V val, BST left, BST right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }

    public BST(V val) {
        this.val = val;
    }
}
```

- `BST<String> tree1 = new BST<>("a");`
表示结点中存储的值是字符串
- `BST<Integer> tree2 = new BST<>(1);`
表示结点中存储的值是整数

二叉搜索树的类定义

```
class BST<V> {
    V val;
    BST left;
    BST right;

    public BST(V val, BST left, BST right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }

    public BST(V val) {
        this.val = val;
    }
}
```

- `BST<String> tree1 = new BST<>("a");`
表示结点中存储的值是字符串
- `BST<Integer> tree2 = new BST<>(1);`
表示结点中存储的值是整数

- `BST<Integer> tree1 = new BST<>(2);`
- `BST<Integer> tree2 = new BST<>(4);`
- `BST<Integer> tree3 = new BST<>(3, tree1, tree2);`

二叉搜索树: contains()

Lecture 1

二叉搜索树
- 导入
- 定义
- contains()
- insert()
- delete()

二叉搜索树的应用

contains()

```
static boolean contains(BST tree, V item) {}
```

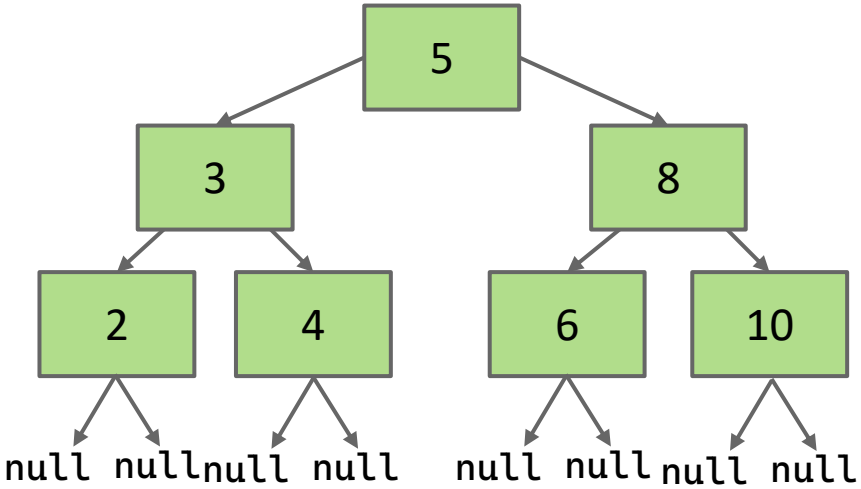

static 是一个修饰符，没学过 **Java** 的脆鲨不要在意。
意思就是这个函数不需要依靠于实例存在，是类函数。

contains()

```
static boolean contains(BST tree, V item) {}
```

If item equals T.val, return.

- If item < T.val, search T.left.
- If item > T.val, search T.right.

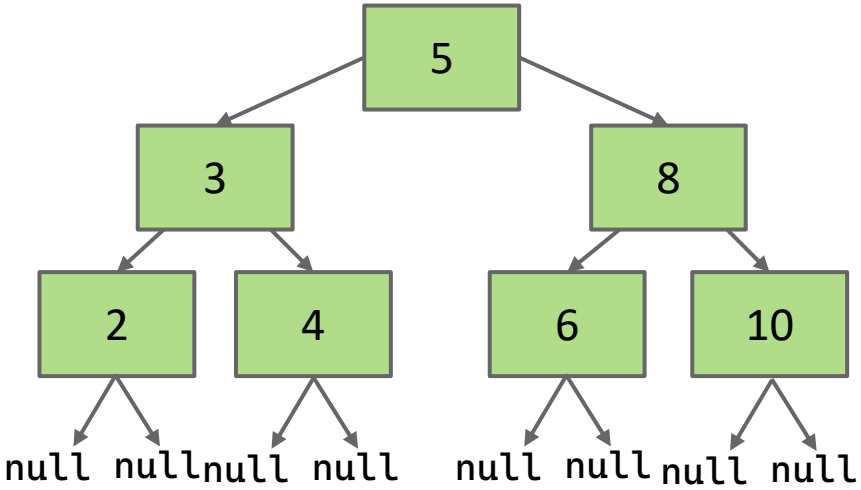


contains()

```
static boolean contains(BST tree, V item) {}
```

If item equals T.val, return.

- If item < T.val, search T.left.
- If item > T.val, search T.right.




```
1.static boolean contains(BST<V> tree, V item) {
2.     if (tree == null) {
3.         return false;
4.     }
5.     if (item.equals(tree.val)) {
6.         return true;
7.     } else if (item < tree.val) {
8.         return contains(tree.left, item);
9.     } else {
10.         return contains(tree.right, item);
11.    }
12.}
```

不能编译！！

复杂度分析

对于下面这棵“长的十分茂盛”的二叉搜索树，最差情况的时间复杂度是多少呢？（N 是结点的数量）

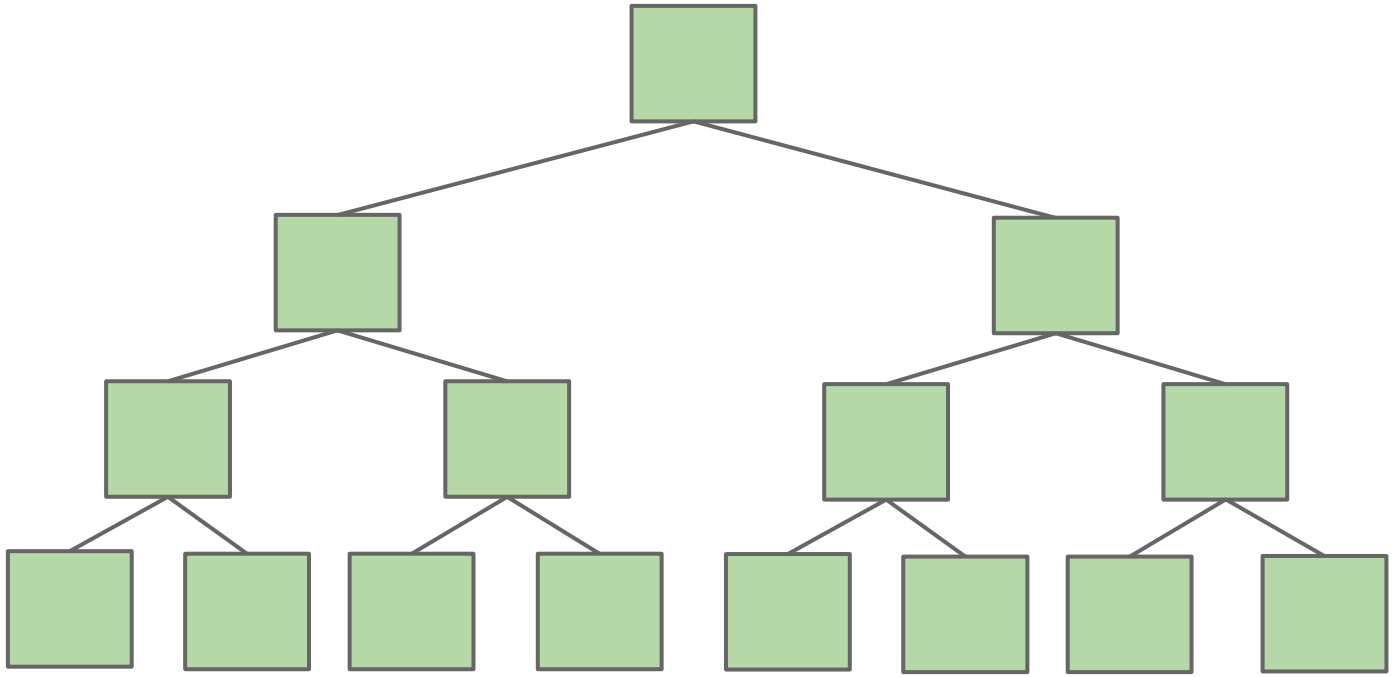
A. $\Theta(\log N)$

B. $\Theta(N)$

C. $\Theta(N \log N)$

D. $\Theta(N^2)$

E. $\Theta(2^N)$



复杂度分析

对于下面这棵“长的十分茂盛”的二叉搜索树，最差情况的时间复杂度是多少呢？（N 是结点的数量）

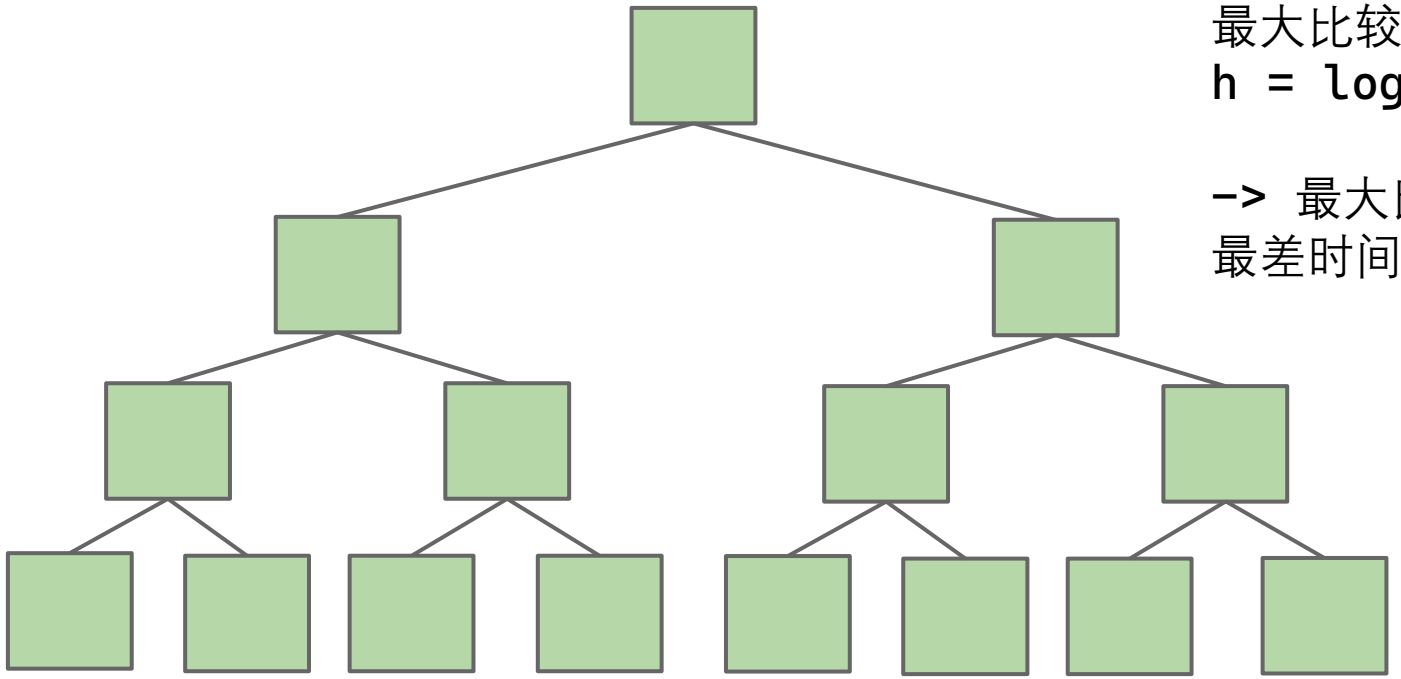
A. Θ(log N)

B. Θ(N)

C. Θ(N log N)

D. $\Theta(N^2)$

E. $\Theta(2^N)$



h – 层数
N – 结点数

最大比较次数 = h + 1
h = log_2(N+1) – 1

-> 最大比较次数 = log_2(N+1)
最差时间复杂度 = Θ(log N)

复杂度分析

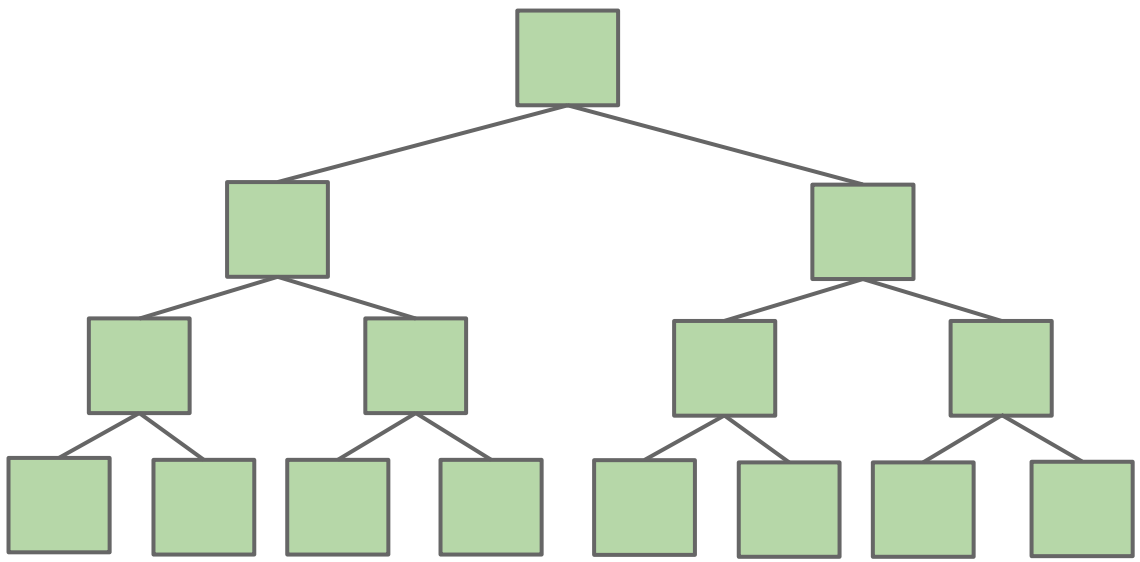
一颗“茂盛”的二叉搜索树实在是太快了!

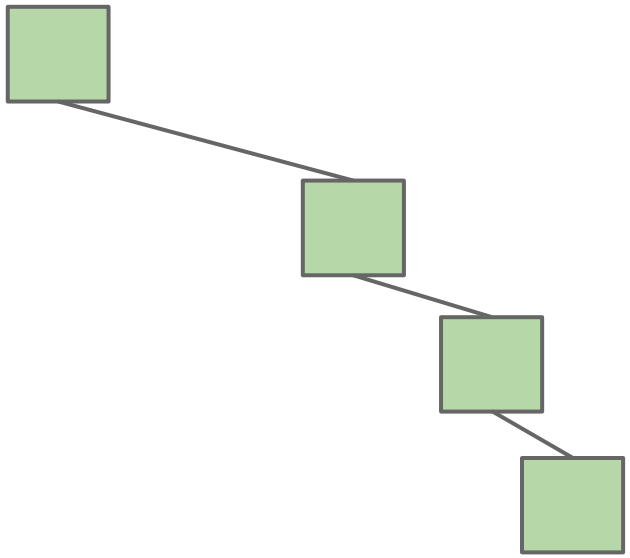
At 1 microsecond per operation, can find something from a tree of size 10^{300000} in one second.

复杂度分析

一颗“茂盛”的二叉搜索树实在是太快了!

At 1 microsecond per operation, can find something from a tree of size 10^{300000} in one second.





二叉搜索树: insert()

Lecture 1

二叉搜索树
- 导入
- 定义
- contains()
- insert()
- delete()

二叉搜索树的应用

insert()

查找要插入的值

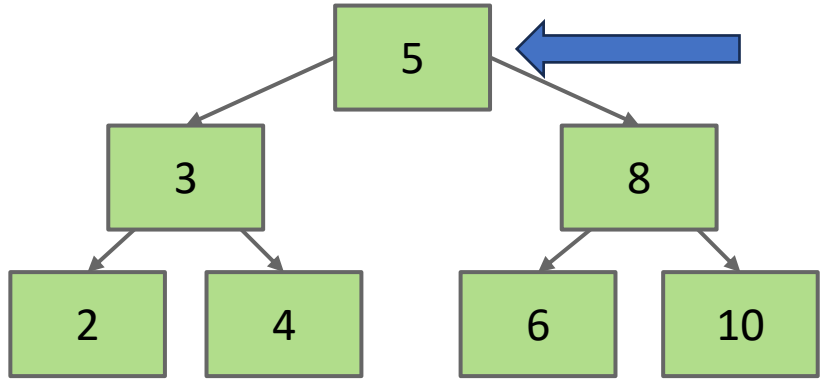
- 如果找到了，就什么都不做 -> 二叉搜索树不能有重复元素
- 如果没找到：（此时一定在某个叶子结点的左侧或者右侧）
  - 创建一个新的结点
  - 用正确的指针连接到正确的位置

insert()

查找要插入的值

- 如果找到了，就什么都不做 -> 二叉搜索树不能有重复元素
- 如果没找到：（此时一定在某个叶子结点的左侧或者右侧）
  - 创建一个新的结点
  - 用正确的指针连接到正确的位置

比如我们要在这个二叉搜索树中插入 **7**

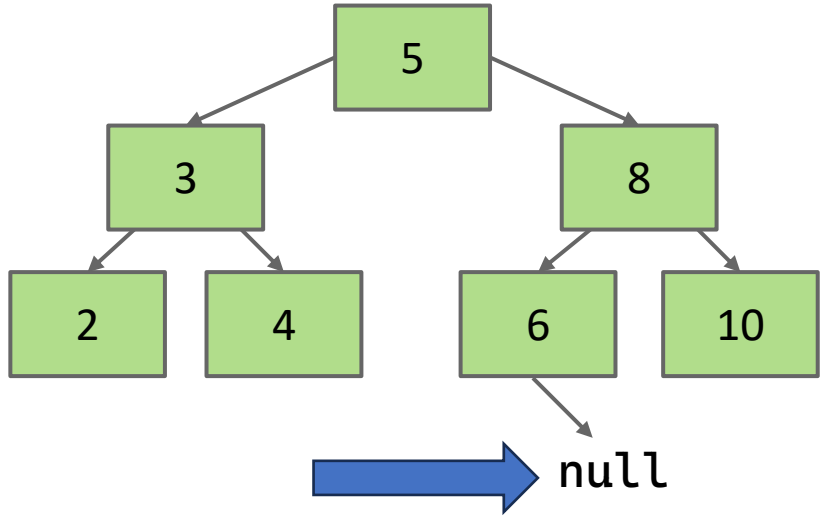


insert()

查找要插入的值

- 如果找到了，就什么都不做 -> 二叉搜索树不能有重复元素
- 如果没找到：（此时一定在某个叶子结点的左侧或者右侧）
  - 创建一个新的结点
  - 用正确的指针连接到正确的位置

比如我们要在这个二叉搜索树中插入 **7**

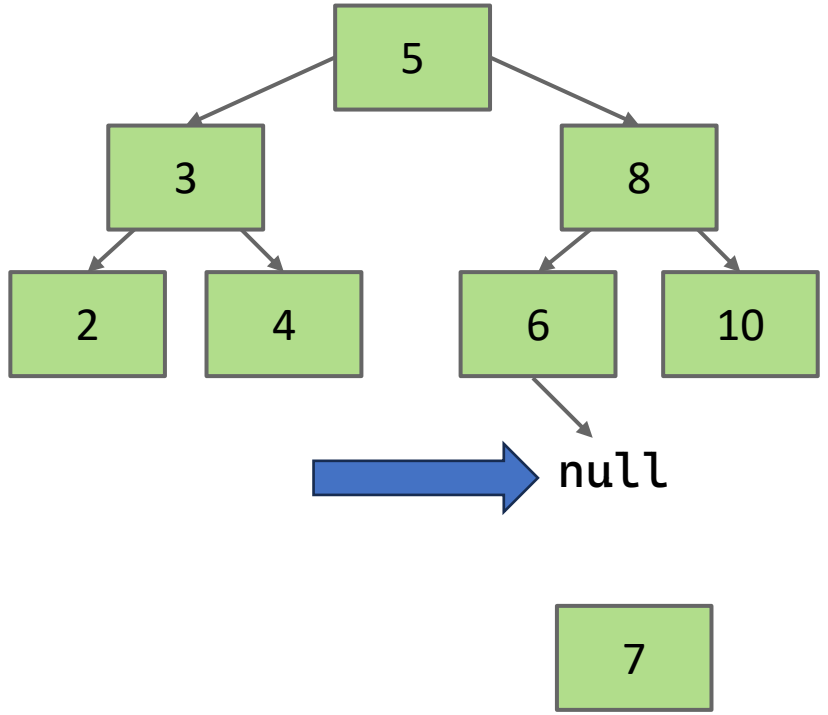


insert()

查找要插入的值

- 如果找到了，就什么都不做 -> 二叉搜索树不能有重复元素
- 如果没找到：（此时一定在某个叶子结点的左侧或者右侧）
  - 创建一个新的结点
  - 用正确的指针连接到正确的位置

比如我们要在这个二叉搜索树中插入 **7**

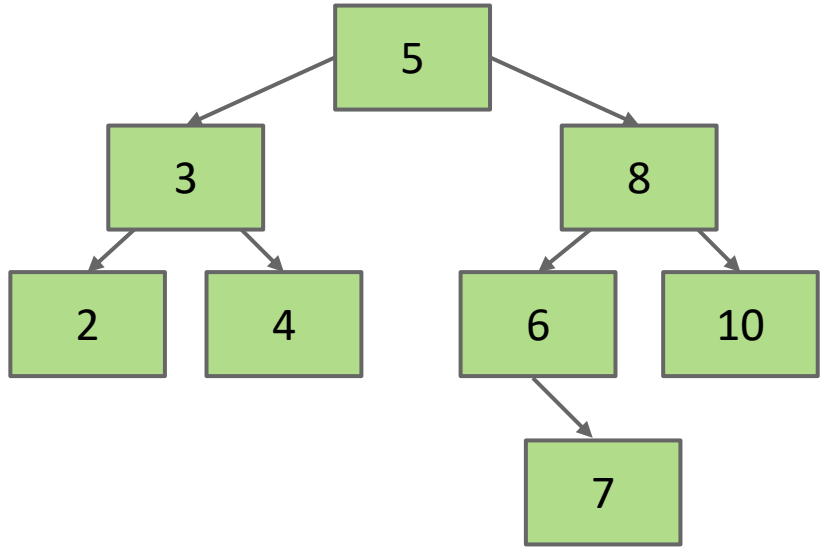


insert()

查找要插入的值

- 如果找到了，就什么都不做 -> 二叉搜索树不能有重复元素
- 如果没找到：（此时一定在某个叶子结点的左侧或者右侧）
  - 创建一个新的结点
  - 用正确的指针连接到正确的位置

比如我们要在这个二叉搜索树中插入 **7**



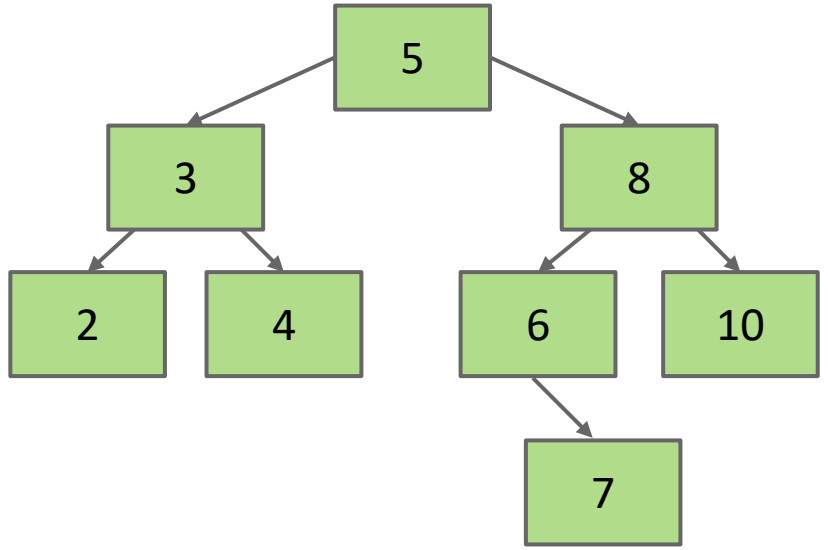
insert()

查找要插入的值

- 如果找到了，就什么都不做 -> 二叉搜索树不能有重复元素
- 如果没找到：（此时一定在某个叶子结点的左侧或者右侧）
  - 创建一个新的结点
  - 用正确的指针连接到正确的位置

```
1.static BST insert(BST T, V ik) {
2.     if (T == null)
3.          return new BST(ik);
4.     if (ik < T.key)
5.          T.left = insert(T.left, ik);
6.     else if (ik > T.key)
7.          T.right = insert(T.right, ik);
8.     return T;
9.}
```

比如我们要在这个二叉搜索树中插入 **7**

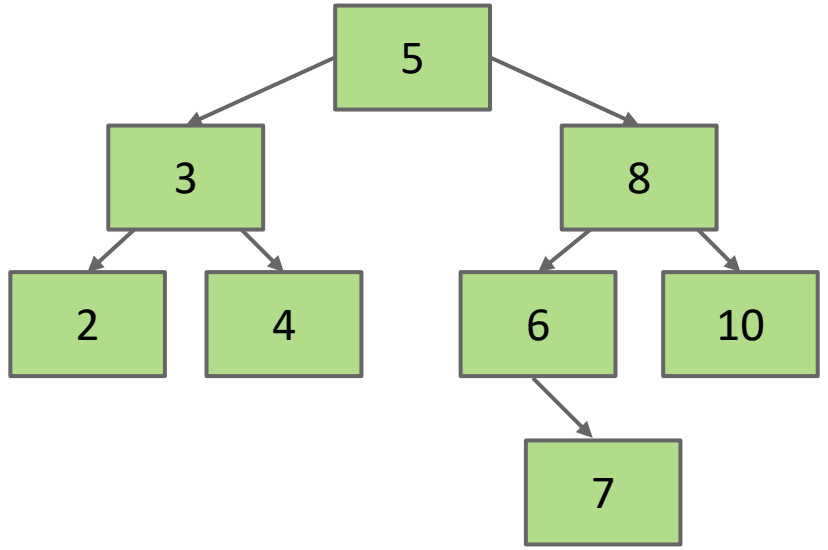


insert()

查找要插入的值

- 如果找到了，就什么都不做 -> 二叉搜索树不能有重复元素
- 如果没找到：（此时一定在某个叶子结点的左侧或者右侧）
  - 创建一个新的结点
  - 用正确的指针连接到正确的位置

比如我们要在这个二叉搜索树中插入 **7**




```
1.static BST insert(BST T, V ik) {
2.     if (T == null)
3.         return new BST(ik);
4.     if (ik < T.key)
5.         T.left = insert(T.left, ik);
6.     else if (ik > T.key)
7.         T.right = insert(T.right, ik);
8.     return T;
9.}
```

```
1.if (T.left == null)
2.     T.left = new BST(ik);
3.else if (T.right == null)
4.     T.right = new BST(ik);
```

二叉搜索树: delete()

Lecture 1

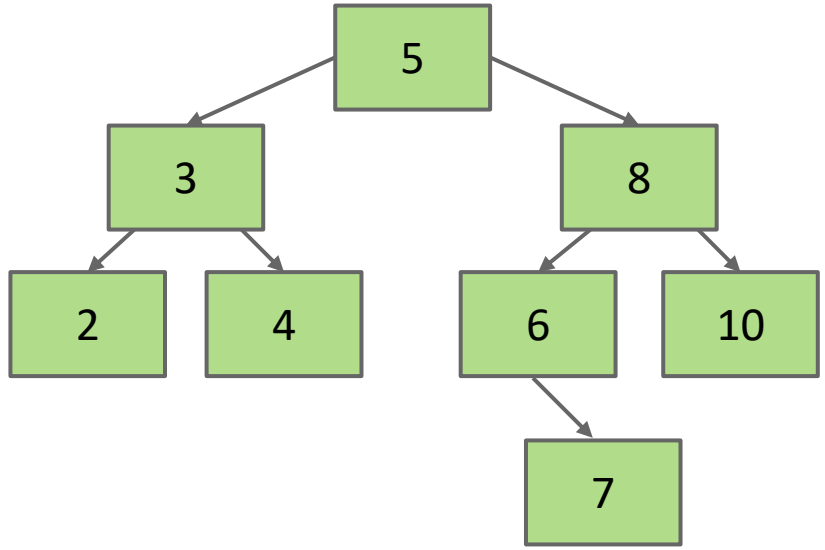
二叉搜索树
- 导入
- 定义
- contains()
- insert()
- delete()

二叉搜索树的应用

delete()

3 Cases:

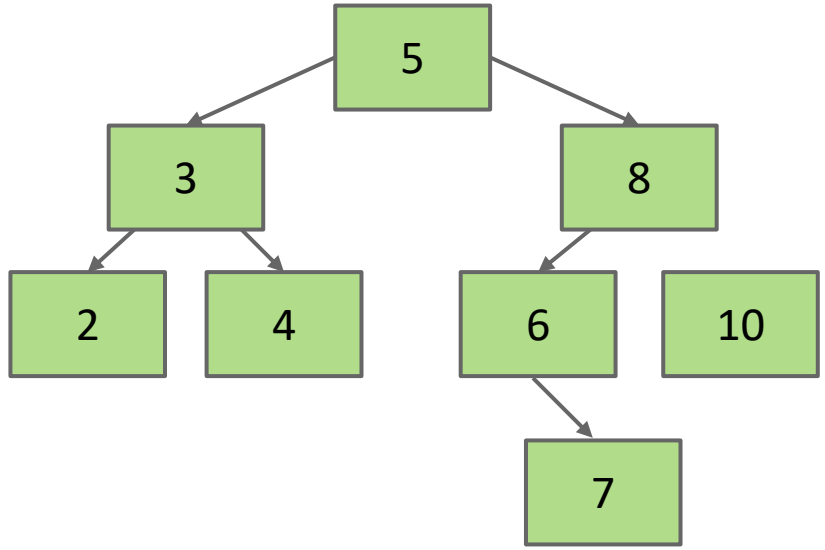
- Deletion key has no children. -> 要被删除的结点没有子结点
- Deletion key has one child. -> 要被删除的结点有一个子结点
- Deletion key has two children. -> 要被删除的结点有两个子结点



delete()

- Deletion key has no children. -> 要被删除的结点没有子结点

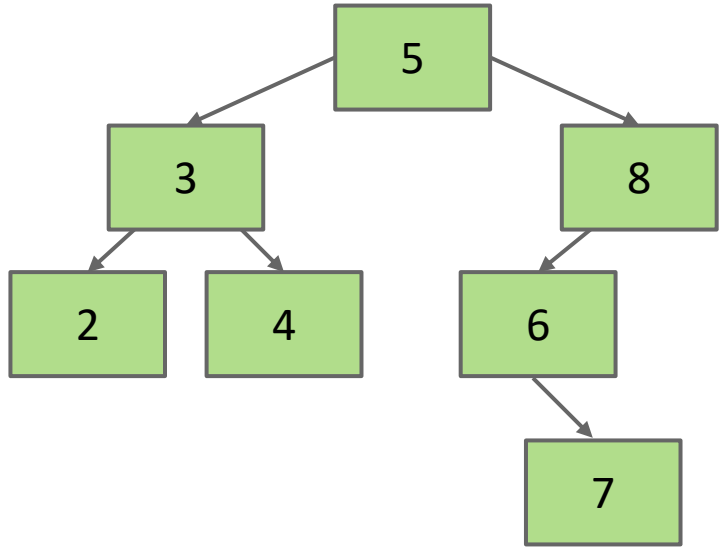
Deletion key has no children (10):
- Just delete the parent’s link.
- What happens to “glut” node?
  - Garbage collected.



delete()

- Deletion key has one child. -> 要被删除的结点有一个子结点

保持二叉搜索树的特征

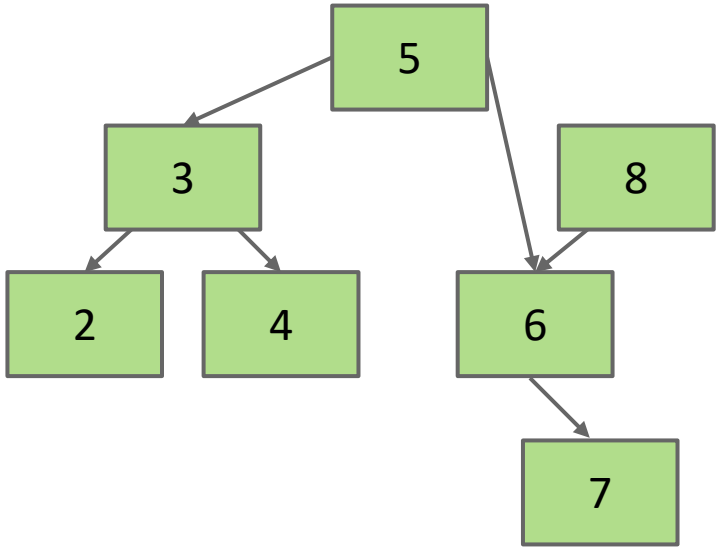


delete()

- Deletion key has one child. -> 要被删除的结点有一个子结点

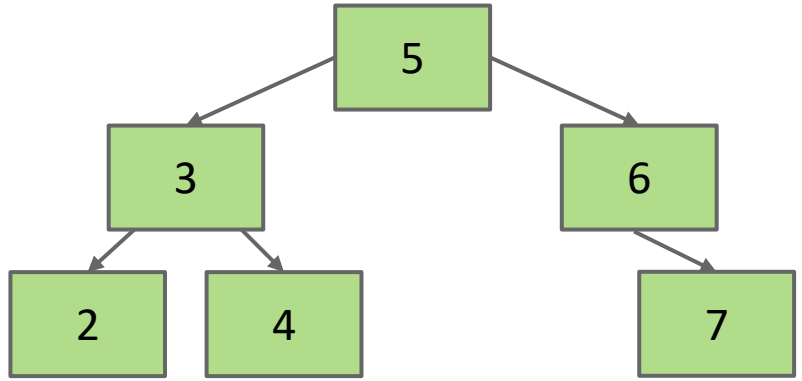
保持二叉搜索树的特征

让被删除结点的父结点指向它的子结点即可。



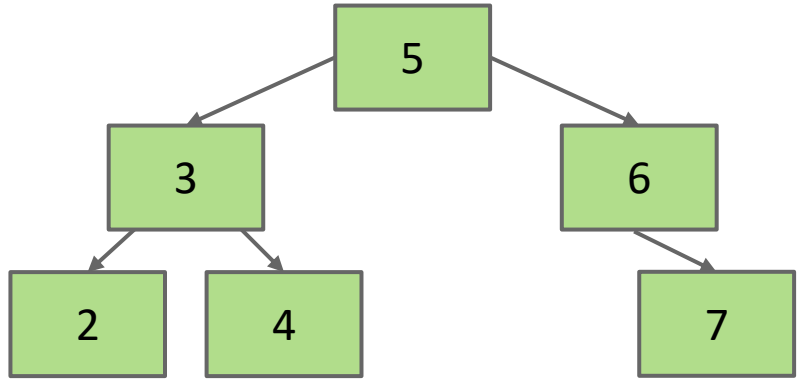
delete()

- Deletion key has two children. -> 要被删除的结点有两个子结点



delete()

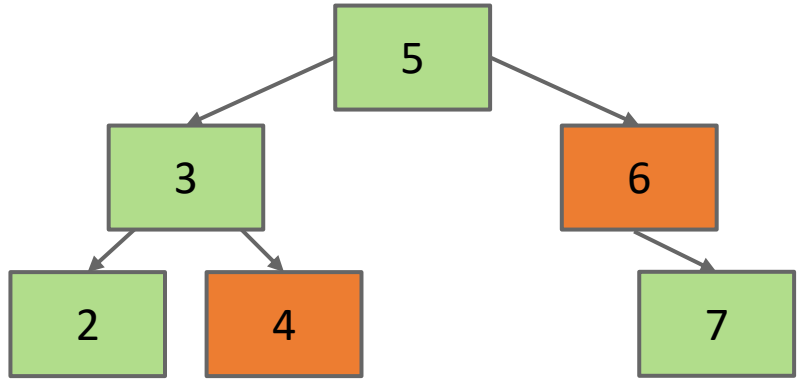
- Deletion key has two children. -> 要被删除的结点有两个子结点



删除五 -> 找一个新的结点代替五
- 新结点大于左子树所有值
- 新结点小于右子树所有值

delete()

- Deletion key has two children. -> 要被删除的结点有两个子结点

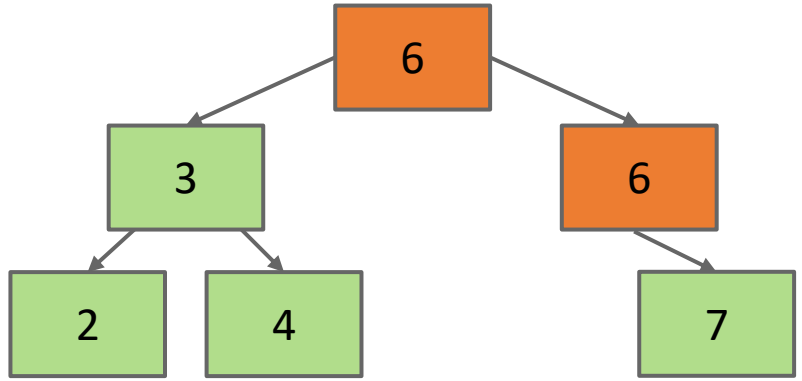


删除五 -> 找一个新的结点代替五
- 新结点大于左子树所有值
- 新结点小于右子树所有值

-> 选择 4 或者 6 作为新的结点

delete()

- Deletion key has two children. -> 要被删除的结点有两个子结点

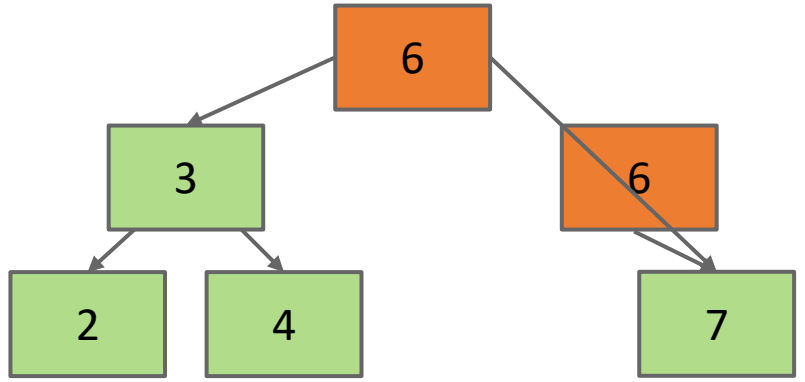


删除五 -> 找一个新的结点代替五
- 新结点大于左子树所有值
- 新结点小于右子树所有值

-> 选择 4 或者 6 作为新的结点（我们在例子中选择 6）
  -> 替换后，再删除原来的 6:

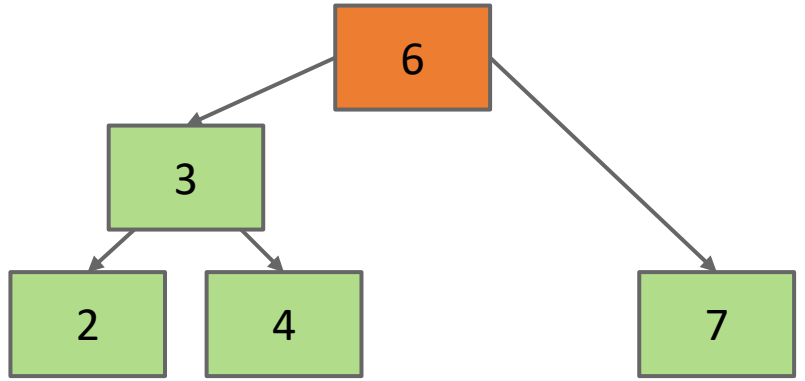
delete()

- Deletion key has two children. -> 要被删除的结点有两个子结点



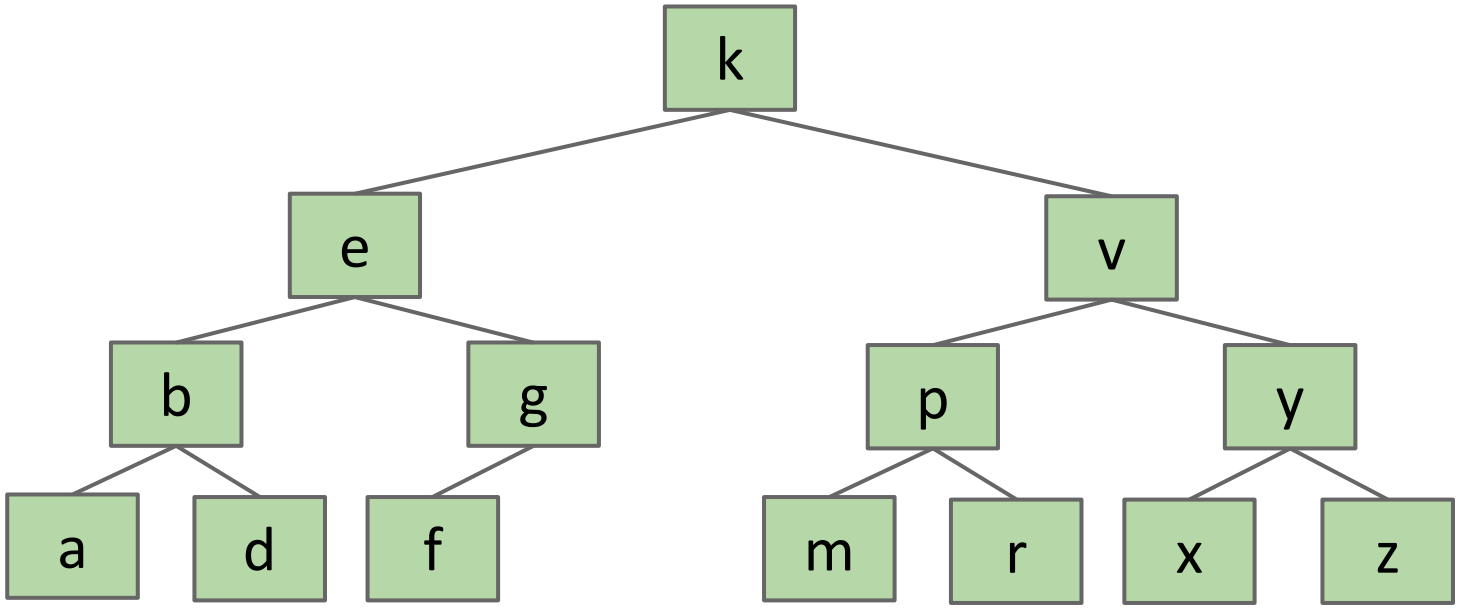
删除五 -> 找一个新的结点代替五
- 新结点大于左子树所有值
- 新结点小于右子树所有值

-> 选择 4 或者 6 作为新的结点（我们在例子中选择 6）
  -> 替换后，再删除原来的 6:
    -> 删除原来的 6 要么是情况一，要么是情况二!
      -> 再回到情况一，二做删除操作即可

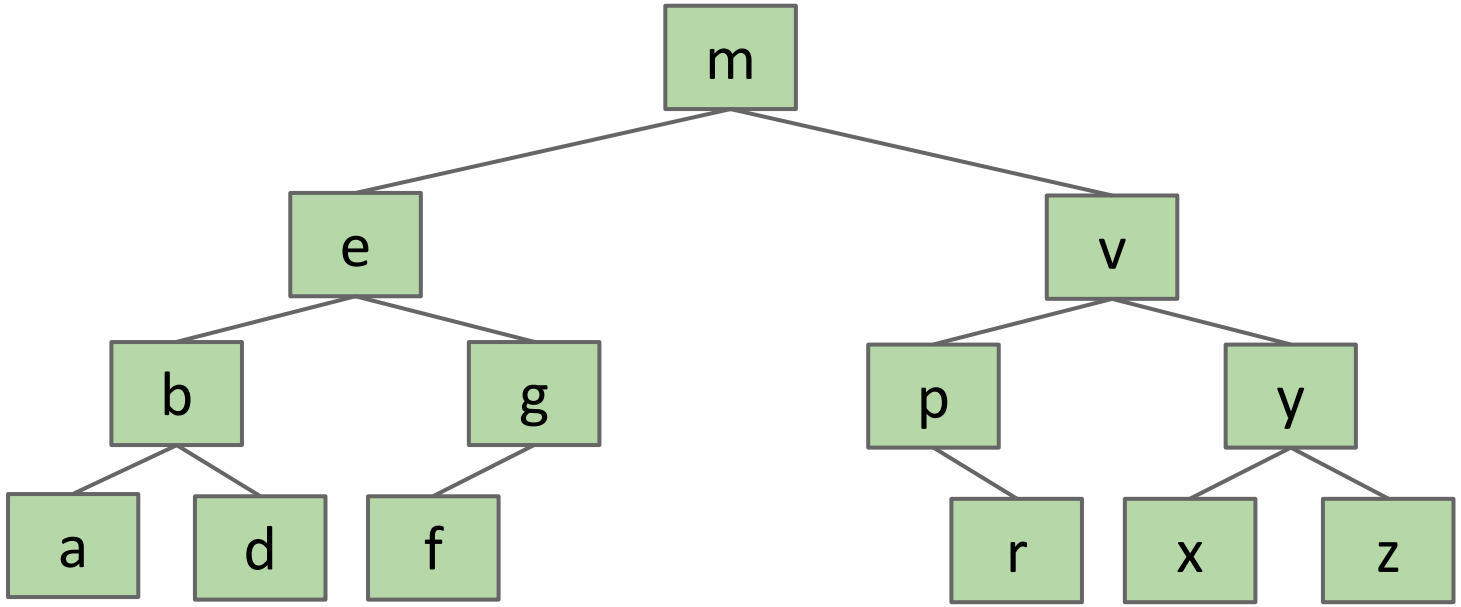


delete()

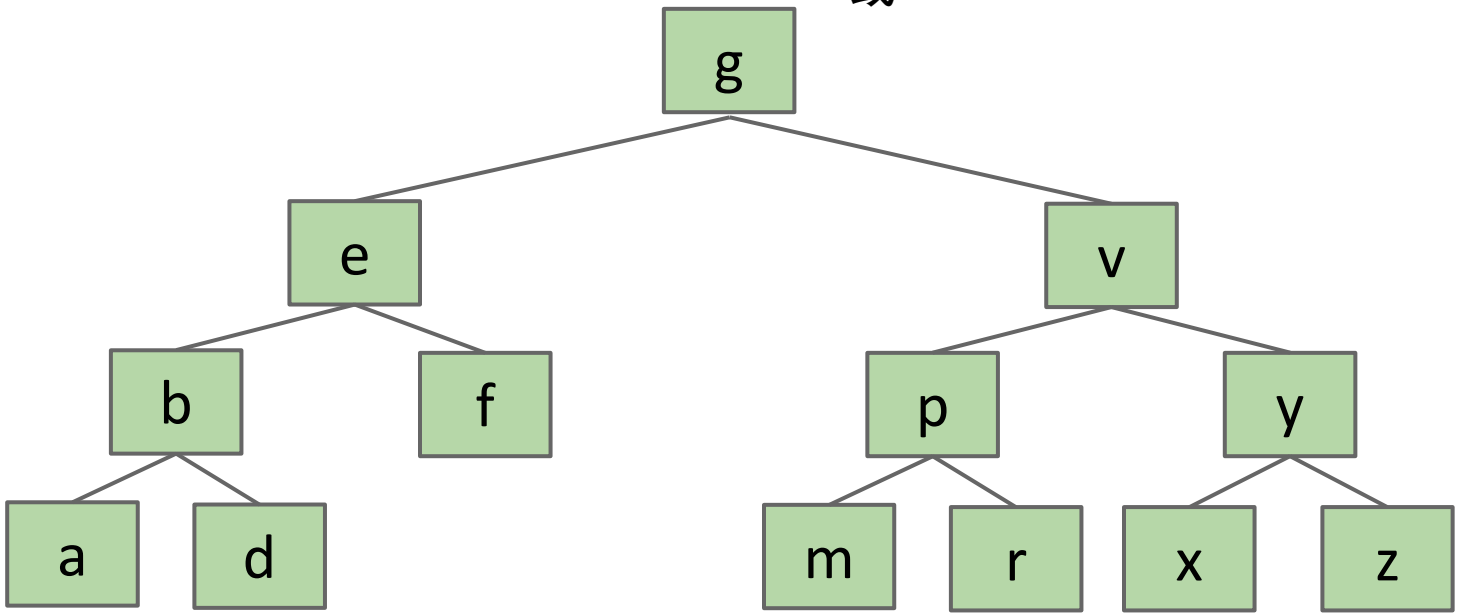
删除 k



delete()



或



二叉搜索树：应用

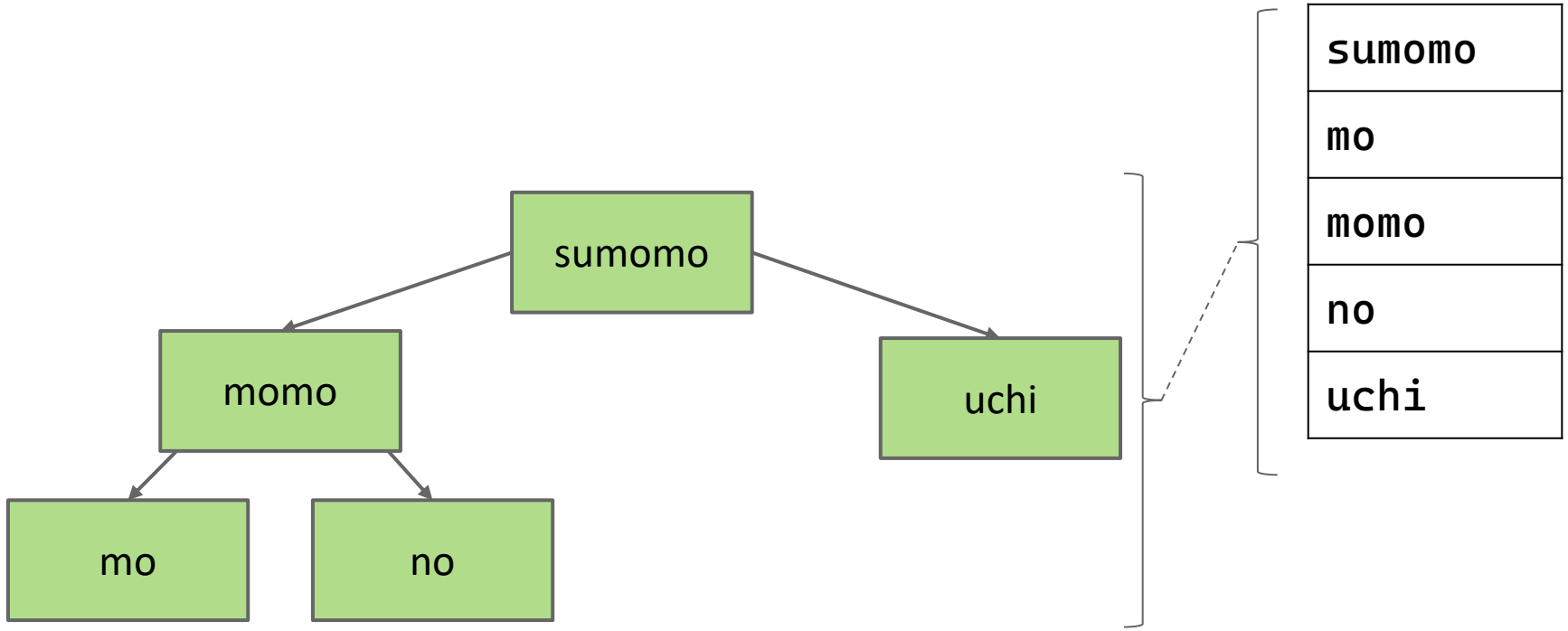
Lecture 1

二叉搜索树
- 导入
- 定义
- contains()
- insert()
- delete()

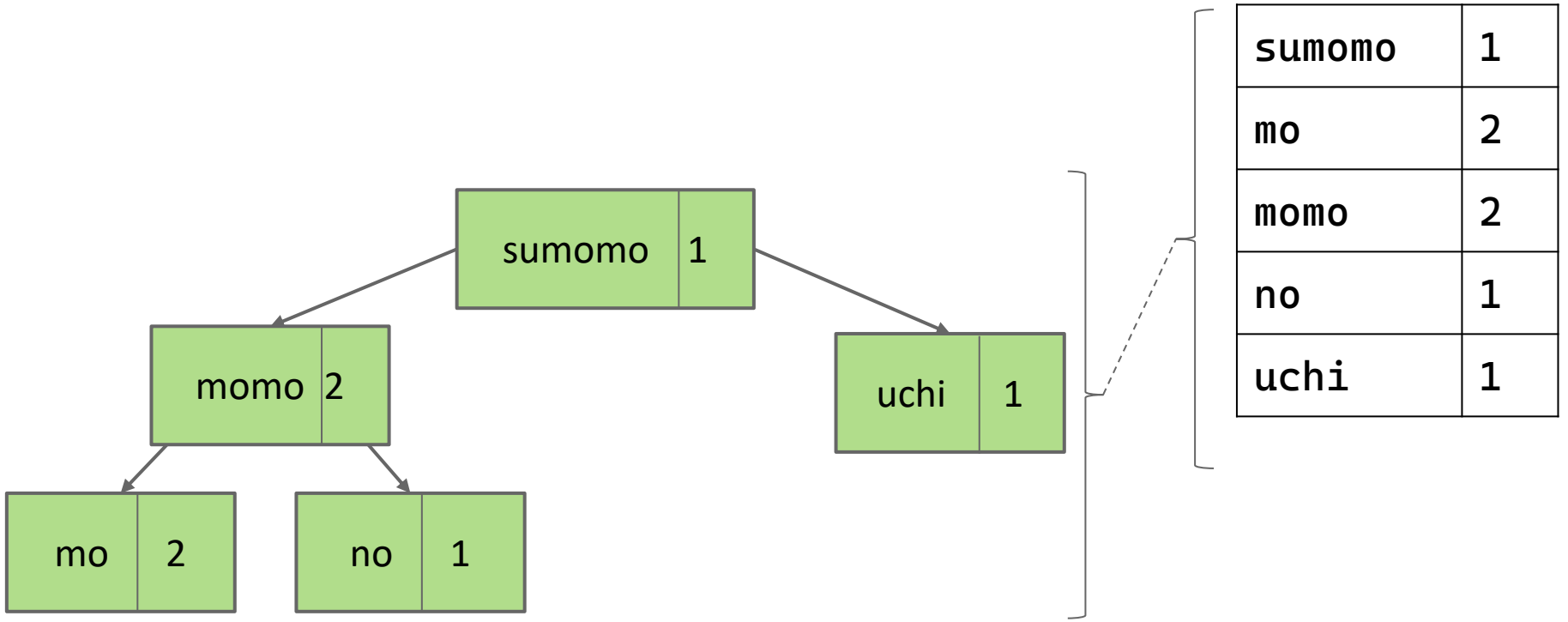
二叉搜索树的应用

Sets & Maps

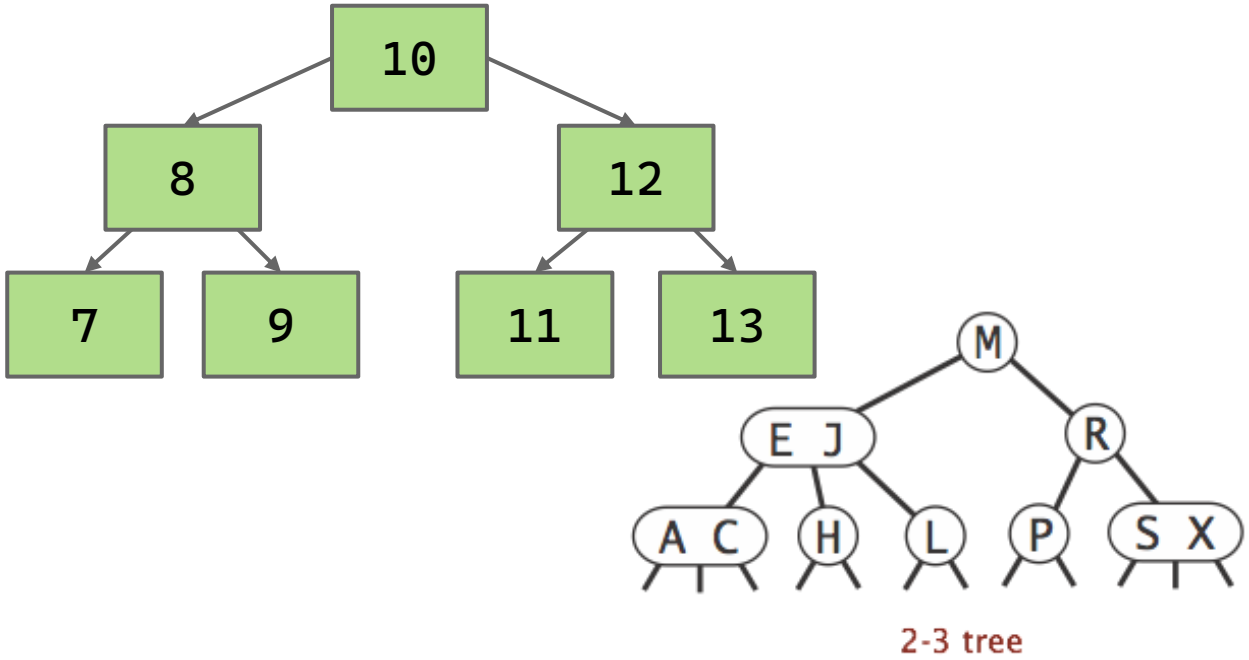
{“mo”, “no”, “sumomo”, “uchi”, “momo”}

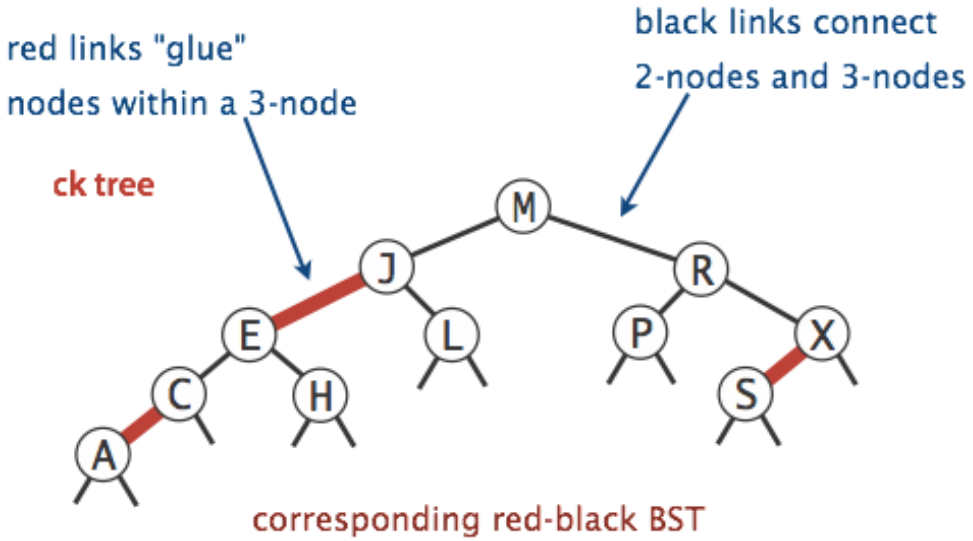


Sets & Maps



sumomo	1
mo	2
momo	2
no	1
uchi	1





谢谢大家

主讲人：七海Nana7mi

课程大纲：CS61B