Algorithms in Swift 3

使用SPM构建开发环境

(https://www.boxueio.com/series/algorithms-in-swift3/ebook/123)

BST II - 打印和遍历

(https://www.boxueio.com/series/algorithms-in-swift3/ebook/88)

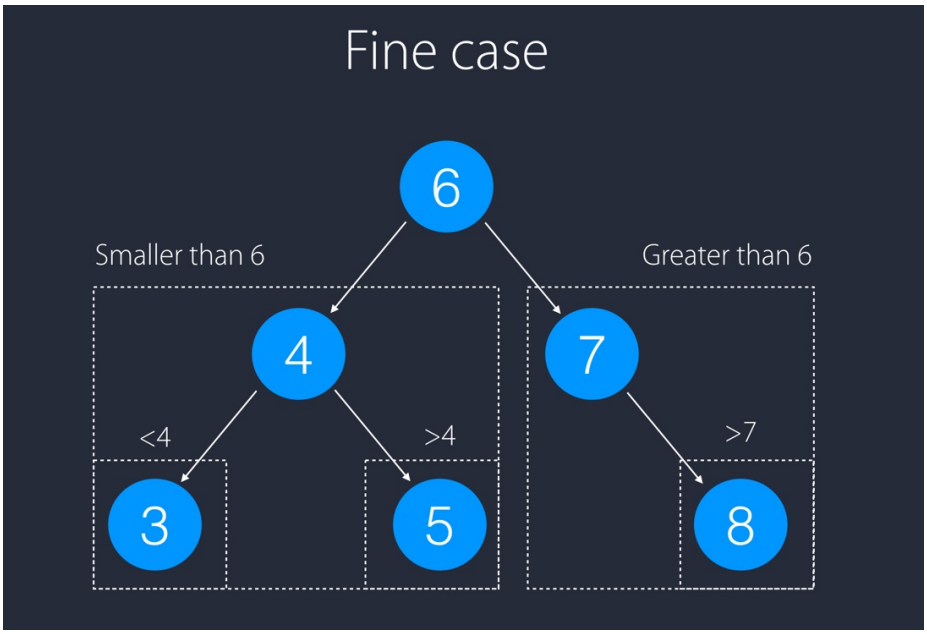
# BST I - 初始化和插入

Back to series (/series/algorithms-in-swift3)

字号
字号
默认主题
金色主题
暗色主题

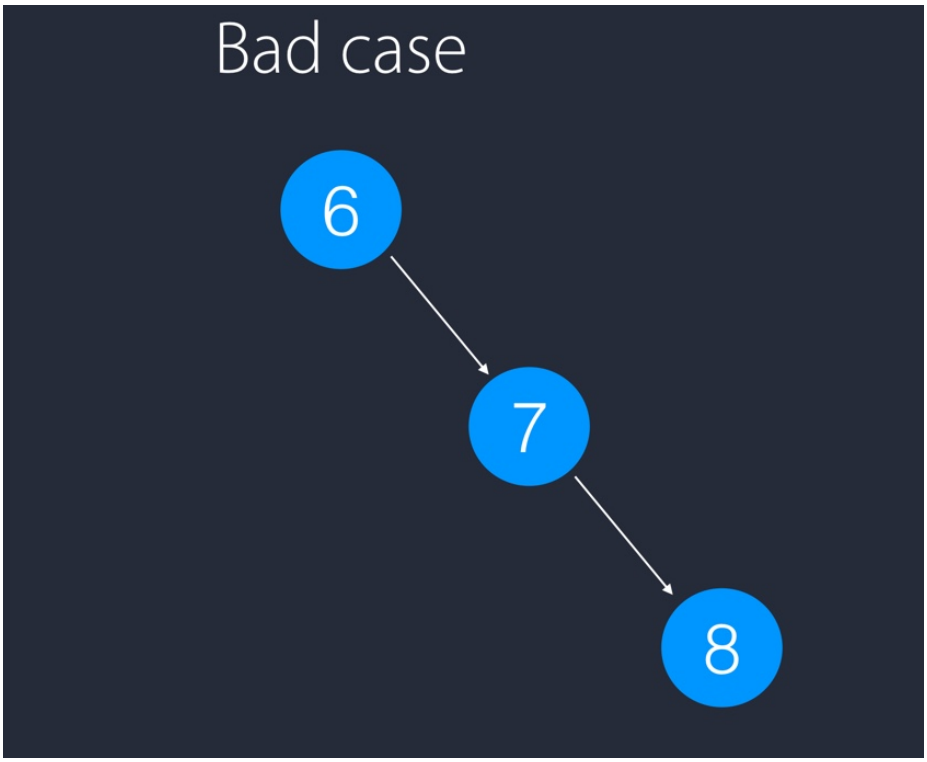
## BST - I

二叉搜索树（BST）是一种常用的保持所有元素都有序的二叉树。如图所示：




首先，它是一个二叉树，每一个节点至多有两个子节点；
其次，在每一个节点，**所有**左子树的节点值都小于父节点，**所有**右子树的节点值都大于父节点；

这样的有序结构在绝大多数时候，都带来的良好的算法执行性能。但也有一些很糟糕的情况，例如：




此时，二叉树就退化成了一个链表。为了避免这样的情况，在后面的视频里，我们会介绍一些更复杂的二叉树。了解了二叉搜索树的核心思想之后，我们就通过Swift来实现它。

## 在使用场景设计API

在Swift API设计指南里，首先提到的一点就是，API是否易用，是要在使用场景里进行评估的，而不是在API定义的时候评估的。毕竟一个方法，只定义一次，却要被反复使用无数次。

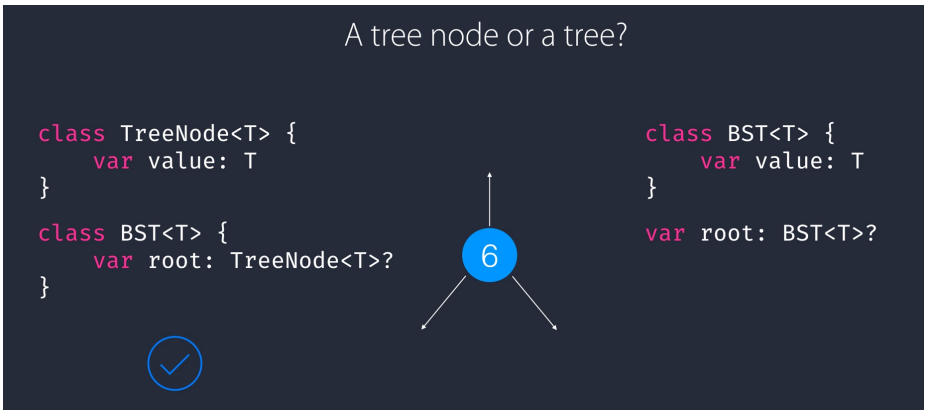
因此，在实现具体的算法之前，我们应该想一下要如何设计BST的各种API呢？我们可以按照操作的性质和API的表现形式大体规划一些场景：

```
// 1. How to create a BST
// let tree = BST<Int>(10)
// let tree1 = BST<Int>([6, 4, 7, 3, 5, 8])

// 2. Formated output
// print(tree1)
// debugPrint(tree1)

// 3. Modification
// tree1.insert(value: 12)

// 4. Traverse BST
```

看似要做的事情不少，别急，我们来一步步实现它们。

## 如何定义一个BST

在实现之前，我们要先做一个选择，如何看待树中的节点呢？




对这个问题的理解关乎到实现BST的方式。

**如果我们认为树中的节点是一个独立的类型**，那么诸如：

- 节点值;
- 子节点和父节点的引用;
- 和节点相关的所有操作；

都应该封装在这个独立的类型里。对于BST来说，它只保存一个属性，就是表示树根的 TreeNode 。

我们也可以淡化 TreeNode 的存在，**把访问一个节点就理解为是直接访问由这个节点和它的子节点构成的BST**。这样，关于节点值、子节点和父节点的信息，都直接保存在BST对象里。只是这样，我们就没有一个明确的“树根”概念了。当我们创建一个新BST时，新创建的对象就是“树根”。

当然，它们没有绝对的对错，只是思考问题的方式不一样罢了。在这段视频里，我们采用左边的方式。

新建一个叫做BST.swift的文件，添加下面的代码：

```
open class TreeNode<T: Comparable> {
    fileprivate var value: T
    fileprivate var parent: TreeNode?
    fileprivate var left: TreeNode?
    fileprivate var right: TreeNode?
}

open class BST<T: Comparable> {
    fileprivate(set) public var root: TreeNode<T>?
}
```

其中，有四点值得说明一下：

- 为了排序，放入 BST 的元素必须是可比较的；
- **在一个泛型类型内部使用类型自身时，可以省略掉泛型参数**，表示它们的泛型参数类型和类对象是一样的。例如 TreeNode 中的parent / left / right，我们直接写了 TreeNode ， 而不是 TreeNode<T> ， 表示如果我们定义一个 TreeNode<Double> ， 那么parent / left / right的类型，也是 TreeNode<Double> ；
- **在一个泛型类型内部使用其他类型时，不可以省略泛型参数**。例如，在 BST 中定义root的时候，我们必须写 TreeNode<T> ，表示一个和 BST 元素类型相同的节点类型；
- 在设计 TreeNode 和 BST 时，我们都希望它们之中的一些算法可以被更复杂的二叉树算法改写，因此，我们使用了 open 修饰这两个类型；
- 对于 TreeNode 的属性，我们希望只在定义它们的文件内被访问和修改，因此把它们定义为了 fileprivate 。而对于 BST ， 我们通过 fileprivate(set) public 让 root 属性达到了只读的效果；

> Swift里没有像C++一样的友元函数的概念。

定义好它们之后，我们先来解决初始化问题。这是所有对象的开始。

## 初始化方法

首先是 TreeNode：

```
public init(value: T,
   parent: TreeNode? = nil,
   left: TreeNode? = nil,
   right: TreeNode? = nil) {

   self.value = value
   self.parent = parent
   self.left = left
   self.right = right
}
```

很简单，既可以按成员初始化；也可以生成一个只有一个值的”树根“节点。然后，来实现一开始提到过的 BST 的初始化方法：

```
// let tree = BST<Int>(10)
// let tree1 = BST<Int>([6, 4, 7, 3, 5, 8])
```

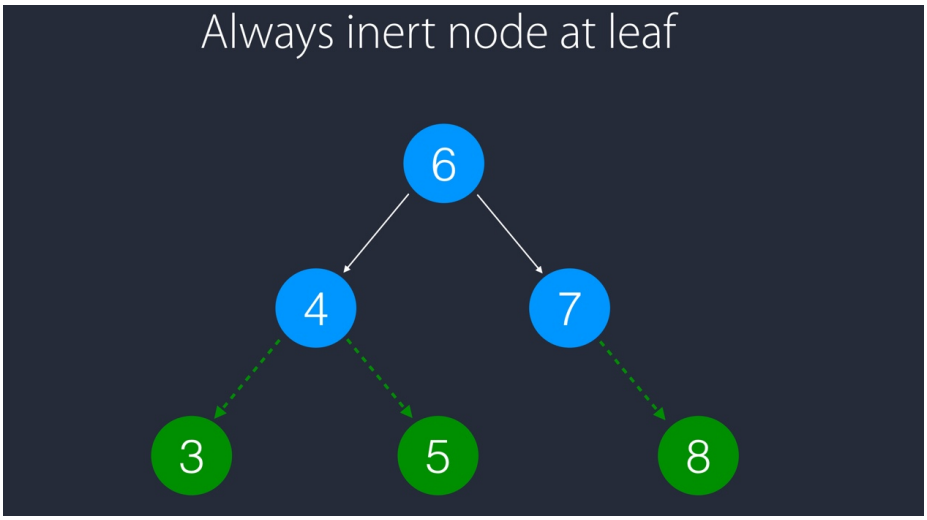
第一种很简单，直接生成一个root为参数值的 BST 就好了：

```
public init(_ value: T) {
   self.root = TreeNode<T>(value: value)
}
```

为了实现第二种初始化方法，我们先了解如何在 BST 中插入元素。

## 插入元素

在 BST 中插入元素只要记住一点就好：**我们一定能在所有的叶子节点中，给要插入的元素找到一个位置。插入操作永远不会发生在中间节点。**




例如上图中的例子里，插入3，5，8，都可以在叶子节点找到合适的位置。有了这个想法之后，实现插入元素就很简单了。我们先给 BST 添加一个 `extension`，所有的辅助方法，都写在这个 `extension` 里。

```
extension BST {

}
```

然后，在其中添加在指定位置插入节点的方法：

```
fileprivate func insert(value: T, under parent: TreeNode<T>) {
   if value < parent.value {
       if let left = parent.left {
           insert(value: value, under: left)
       }
       else {
           let node = TreeNode<T>(value: value)
           parent.left = node
           node.parent = parent
       }
   }
   else if value > parent.value {
       if let right = parent.right {
           insert(value: value, under: right)
       }
       else {
           let node = TreeNode<T>(value: value)
           parent.right = node
           node.parent = parent
       }
   }
}
```

这样，我们就可以在 BST 的 open 部分添加 `insert` 方法了：

```
open func insert(value: T) {
   self.insert(value: value, under: self.root!)
}
```

之后，实现通过数组初始化 BST 也是个很简单的事情，无非就是创建一个root之后，不断插入新值罢了：

```
public init(_ array: [T]) {
   precondition(array.count > 0)

   self.root = TreeNode<T>(value: array[0])

   for value in array[1 ..< array.count] {
       insert(value: value)
   }
}
```

完成后， Command + B 构建一次项目，如果没有任何错误，就表示数组版的 init 方法也开始工作了。

## Next?

但是，我们构建的 BST 真的如我们想象么？为了确认生成的结果，在下一段视频中，我们将向大家介绍自定义打印 BST 的方法，以及三种常用的遍历 BST 的方式。

⏮ 使用SPM构建开发环境

(https://www.boxueio.com/series/algorithms-in-swift3/ebook/123)

BST II - 打印和遍历 ⏭

(https://www.boxueio.com/series/algorithms-in-swift3/ebook/88)