数据结构与算法

Data Structure and Algorithm

XIV. 树（上）

授课人：Kevin Feng

翻译 ：梁 少 华

Review

回顾

- 数据结构与算法
- 数学回顾
- 数组
- 数组列表
- 搜索和排列
- 递归与迭代
- 二进制搜索
- 分而治之
- 链接列表
- 散列表

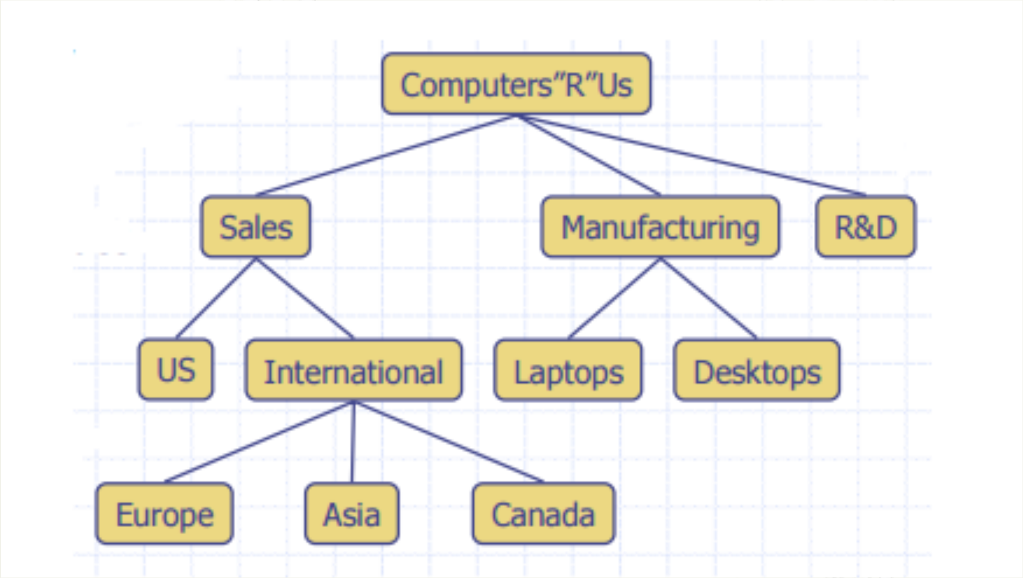
概述

- 树
  - 数据成员
  - 操作
- 二叉树
- 二进制搜索树

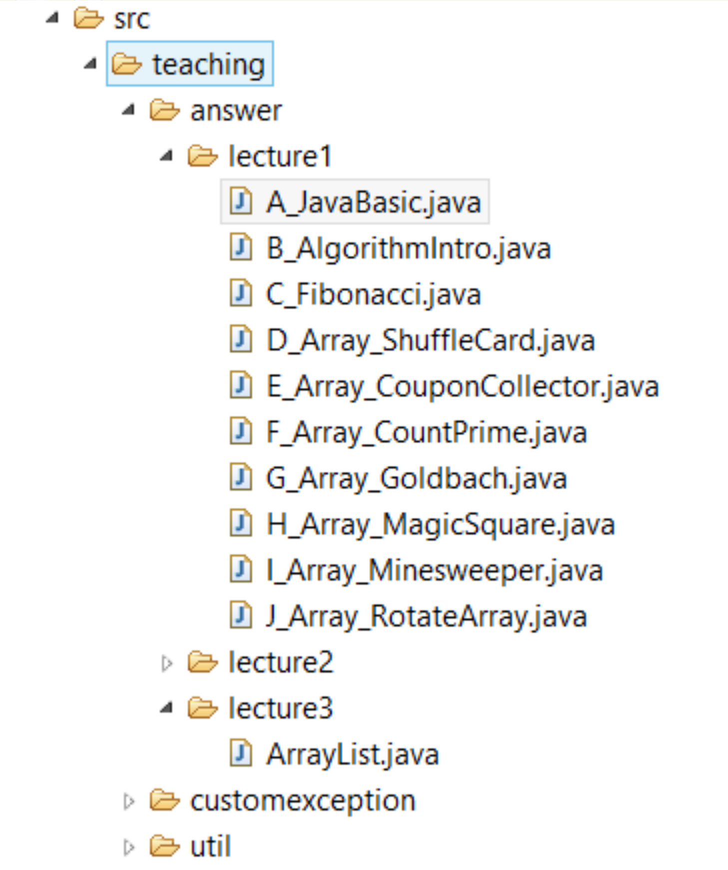
基本思想

- 我们学习链接列表
- 在计算机科学中,树是分层结构的抽象模型
- 一棵树由具有父子关系的节点组成
  - 经常为时间牺牲空间 - 所有“洞”都浪费空间
  - 依赖于元素的排序，这意味着内存块的排序
- 应用:
  - 组织结构图
  - 文件系统
  - 编程环境

例子



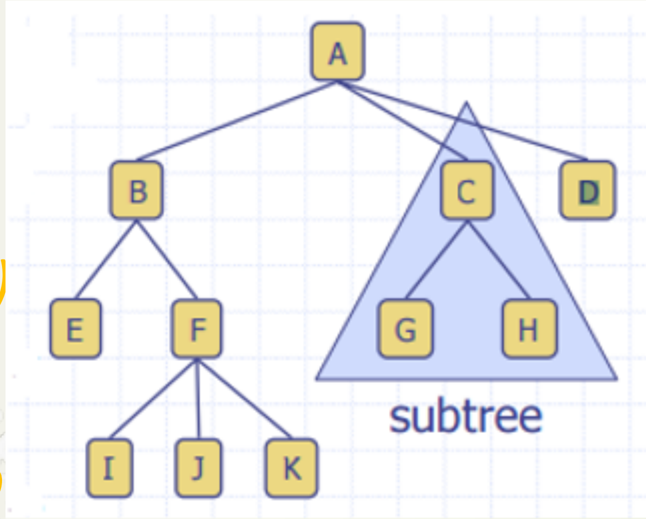
Computers"R"Us
Sales
Manufacturing
R&D
US
International
Laptops
Desktops
Europe
Asia
Canada



src
teaching
answer
lecture1
A_JavaBasic.java
B_AlgorithmIntro.java
C_Fibonacci.java
D_Array_ShuffleCard.java
E_Array_CouponCollector.java
F_Array_CountPrime.java
G_Array_Goldbach.java
H_Array_MagicSquare.java
I_Array_Minesweeper.java
J_Array_RotateArray.java
lecture2
lecture3
ArrayList.java
customexception
util

树术语

- Root: 无父节点 *(A)*
- Internal node:具有至少一个子节点的节点 *(A, B, C, F)*
- External node (a.k.a. leaf): 无子节点 *(E, I, J, K, G, H, D)*
- Ancestors of a node:父母,祖父母,祖父母,等等.
- Descendant of a node:孩子,孙子,重孙子,等等. *A B C D*
- Depth of a node:树的祖先高度数：任意节点的最大深度. *Depth(E) = 2*
- Height:任何节点的最大深度, *Height = 3*
- Sibling: C是B和D的兄弟姐妹.
- Subtree:树由节点及其后代组成
- Edge of tree:是一对节点(u，v),u是v的父节点
- Path:节点S.T的一个序列，来自边缘的任意两个连续的节点



树的ADT

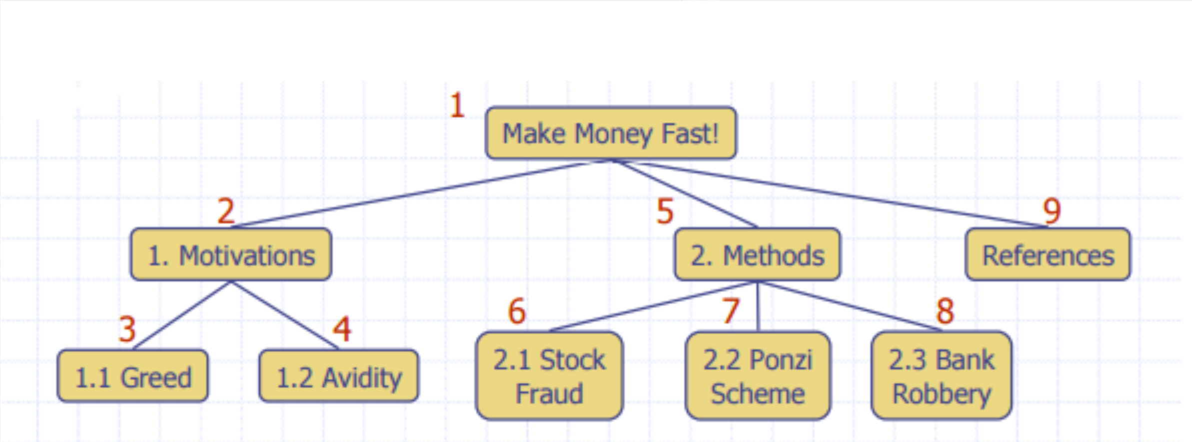
- 二叉树可以通过存储一个节点的数据加两个子指针来实现.
- 具有两个以上孩子的树可以使用链接的节点列表来实现，如下一张幻灯片所示.

- 通用方法:
  - Int size()
  - boolean isEmpty()
  - Iterator elements()
  - Iterator positions()
- 访问器方法:
  - Node root()
  - Node parent(p)
  - List<Node> children(p)
- 查询方法:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- 更新方法:
  - object replace (p, o)
- 额外的更新方法可以由实现Tree ADT的数据结构来定义

先序遍历

- 遍历以系统的方式访问树的节点
- 在前序遍历中,一个节点在它的后代之前被访问
- 应用程序：打印结构化文档

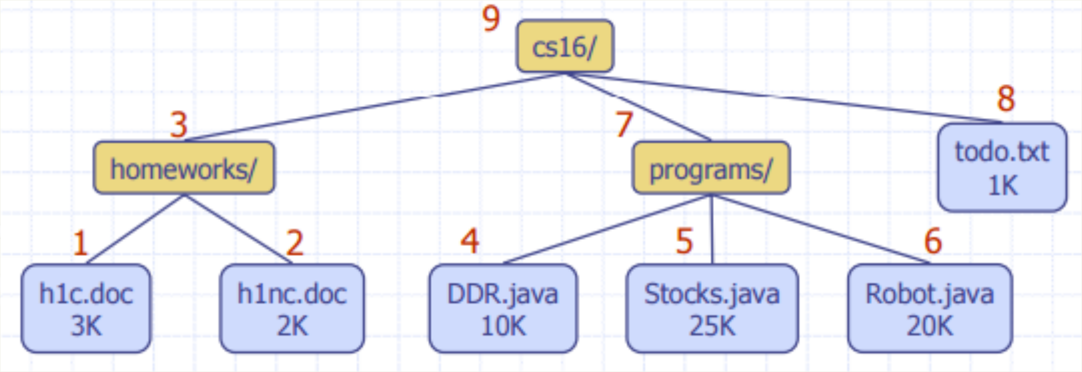
```
Algorithm preOrder(v)
    visit(v)
    for each child w of v
        preorder (w)
```



后序遍历

- 在后序遍历中,节点在其后代之后被访问
- 应用程序：计算目录及其子目录中文件使用的空间

```
Algorithm postOrder(v)
    for each child w of v
        postOrder (w)
    visit(v)
```

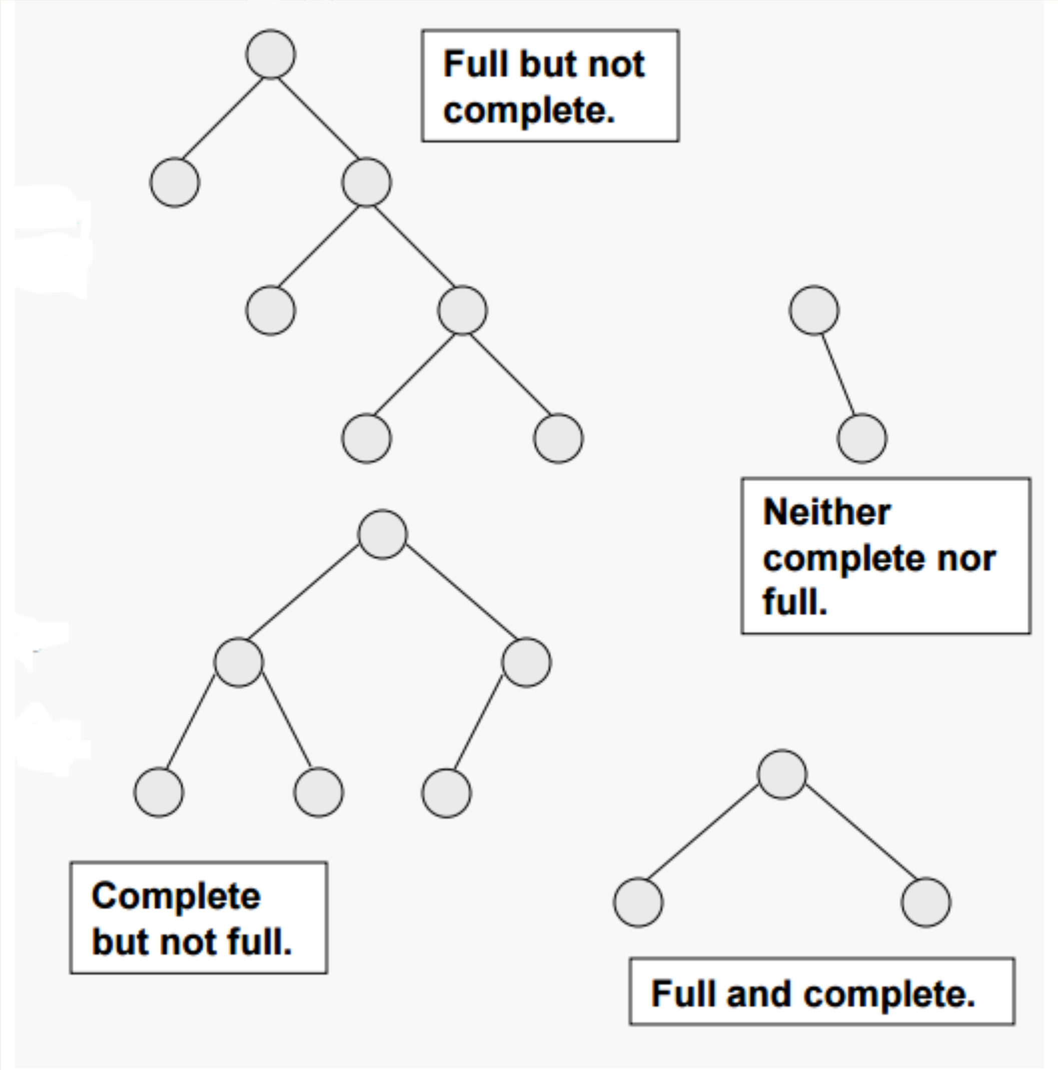


二叉树

- 二叉树是具有以下属性的树:
  - 每个内部节点至多有两个孩子（正确的二叉树恰好是两个孩子）
  - 节点的孩子是有序对
- 我们把一个内部节点的孩子称为“左子”和“右子
- 替代递归定义：二叉树
  - 由单个节点组成的树，或,
  - 一棵树，它的根有一对有序的孩子，每一棵树都是二叉树
- 应用:
  - 算术表达式
  - 决策过程
  - 搜索

完整 vs. 完成

- 完整的二叉树是一棵树，树叶中的每个节点都有两个孩子
- 一个完成的二叉树是一棵二叉树，其中除了最后一个以外的每一层都被完全填充，并且所有节点尽可能地离开

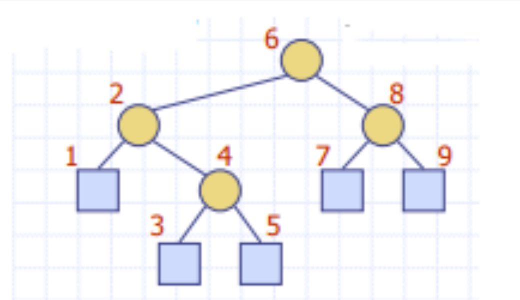


二叉树 ADT

- 二叉树ADT扩展了树ADT，即它继承了树ADT的所有方法
- 其他方法
  - Node left(p)
  - Node right(p)
  - boolean hasLeft(p)
  - boolean hasRight(p)

顺序遍历

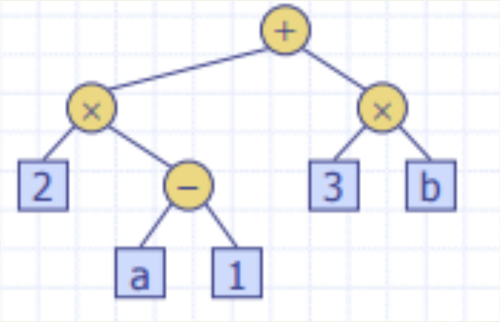
- 在顺序遍历中，在左子树和右子树之前访问节点
- 应用:




```
Algorithm inOrder(v)
    if hasLeft (v)
        inOrder (left (v))
    visit(v)
    if hasRight (v)
        inOrder (right (v))
```

打印算术表达式

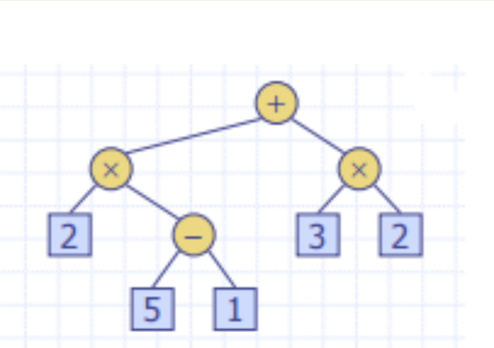
- 顺序遍历的专业化
  - 在访问节点时打印操作数或操作符
  - 在遍历左子树之前打印“(“
  - 在遍历右子树之后打印“)“




```
Algorithm printExpression(v)
    if hasLeft (v)
        print("(")
        inOrder (left(v))
    print(v.element ())
    if hasRight (v)
        inOrder (right(v))
        print (")")
```

评估算术表达式

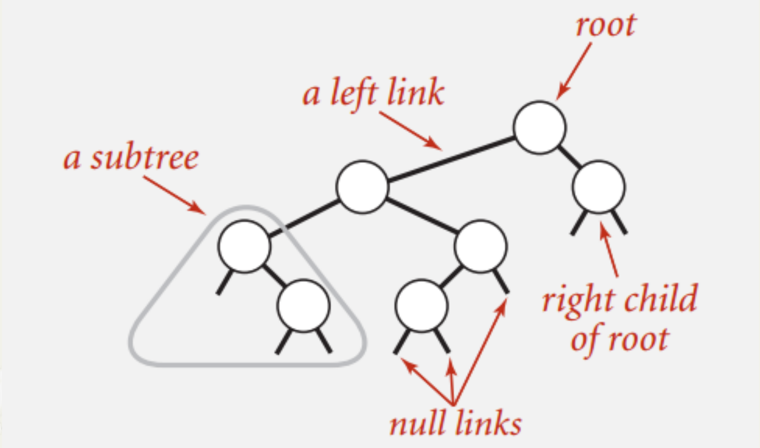
- 后序遍历的专业化
  - 递归方法返回一个子树的值
  - 当访问内部节点时，合并子树的值

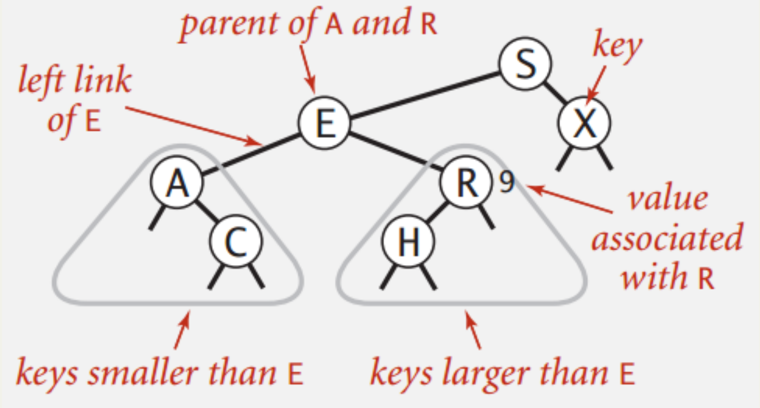



```
Algorithm evalExpr(v)
    if isExternal (v)
        return v.element ()
    else
        x ← evalExpr(leftChild (v))
        y ← evalExpr(rightChild (v))
        ◊ ← operator stored at v
        return x ◊ y
```

二进制搜索树

- 定义：BST是对称顺序的二叉树
- 二叉树也是:
  - 空
  - 两个不相交的二叉树（左和右）
- 对称顺序
  - 每个节点都有一个密钥，每个节点的密钥都是
  - 大于其左侧子树中的所有密钥
  - 小于其右侧子树中的所有密钥



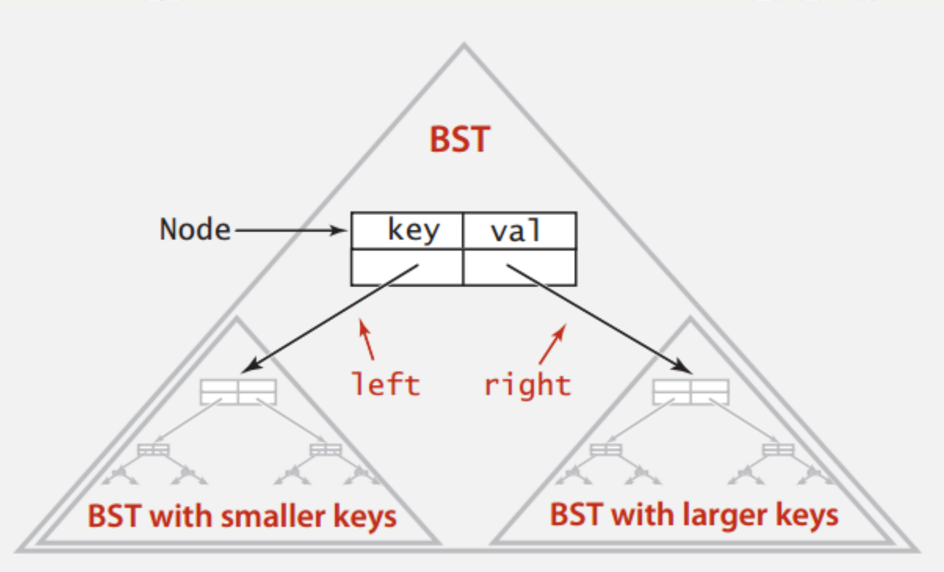


BST 用Python表示

- BST是对根节点的引用
- 节点由四个字段组成
  - 一个 *Key* 和 一个 *Value*
  - 对左子树(较小的密钥)和右子树(较大的密钥)的引用

```python
class Node:
    __slots__ = '_item' , '_left' , '_right'

    def __init__ (self, item, left=None, right=None):
        self._item = item
        self._left = left
        self._right = right
```



BST 搜索

- 获取：返回给定键对应的值;如果没有这样的键，则返回null

```python
# Get methods
def get(self, key):
    return self.__get(self._root, key);

def __get(self, node, key): # helper
    if (node is None):
        return None
    if (key == node._item):
        return node._item
    if (key < node._item):
        return self.__get(node._left, key)
    else:
        return self.__get(node._right, key)
```

BST-Search$(x,\ k)$

1: $y \leftarrow x$
2: **while** $y \neq \mathbf{nil}$ **do**
3:     **if** $key[y] = k$ **then return** y
4:     **else if** $key[y] < k$ **then** $y \leftarrow right[y]$
5:     **else** $y \leftarrow left[y]$
6: **return** ("NOT FOUND")

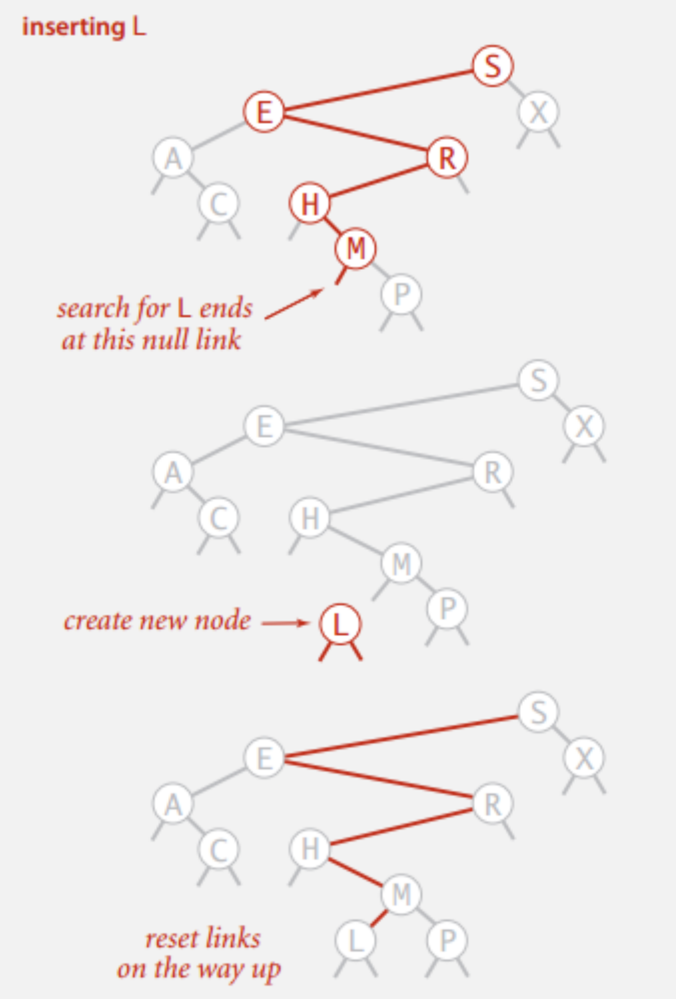
- 成本：比较的数量等于1 +节点的深度

BST 插入

- 添加：将关联值与关键字关联
- 搜索密钥，然后搜索两个案例
  - 键入树⇒重置值
  - 密钥不在树中⇒添加新节点

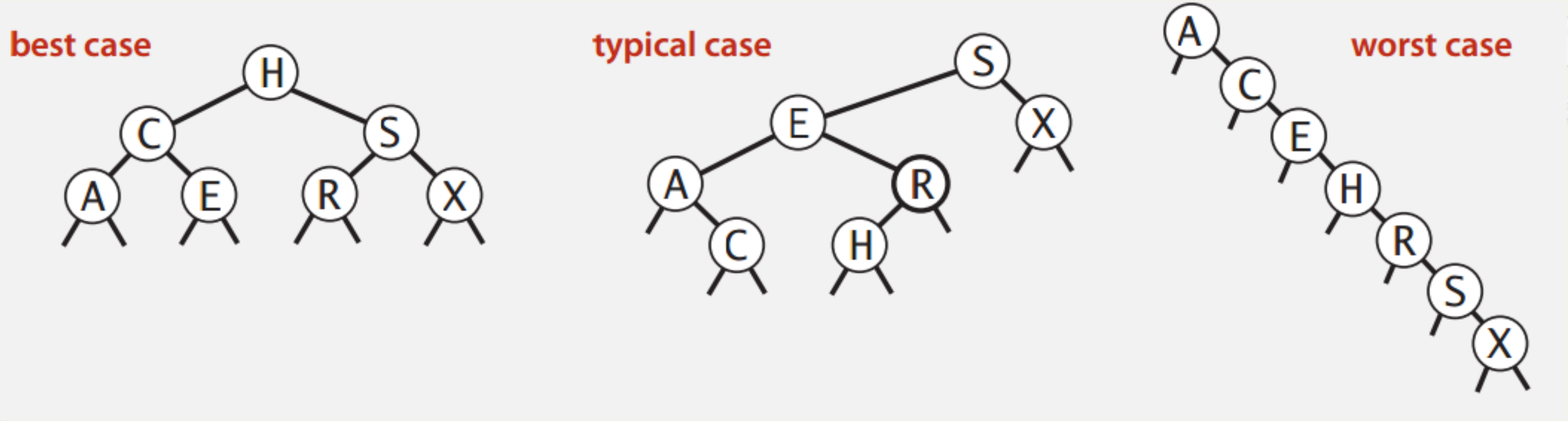
```
BST-Insert(x, z, k)
if x = nil then return "Error"
y ← x
while true do {
    if key[y] < k
    then z ← left[y]
    else z ← right[y]
    if z = nil break
}
if key[y] > k then left[y] ← z
else right[p[y]] ← z
```

- 成本：比较的数量等于1 +节点的深度



树的形状

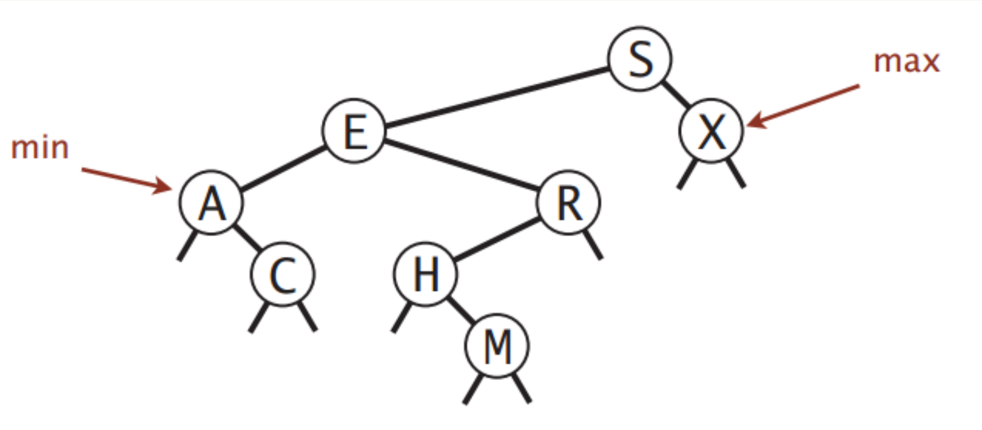
- 许多BST对应于相同的一组键
- 搜索/插入的比较数等于1 +节点的深度



- 所以平均来说，它是*$log_2(n)$*

最小值与最大值

- 问题.如何找到最小和最大值?



BST-Minimum(x)

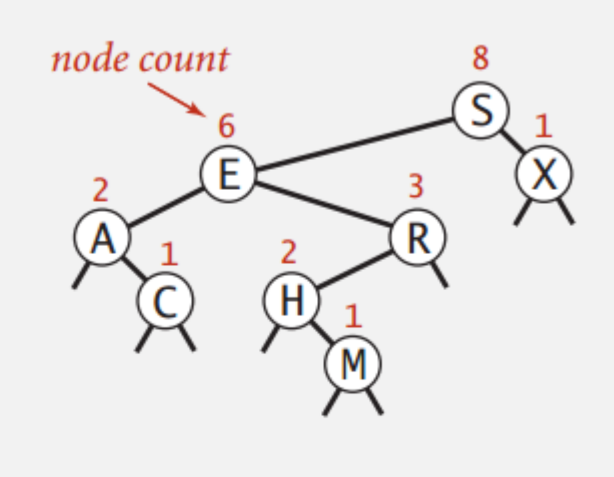
if $x =$ nil **then return** ("Empty Tree")
$y \leftarrow x$
while $left[y] \neq$ nil **do** $y \leftarrow left[y]$
return $(key[y])$

BST-Maximum(x)

if $x =$ nil **then return** ("Empty Tree")
$y \leftarrow x$
while $right[y] \neq$ nil **do** $y \leftarrow right[y]$
return $(key[y])$

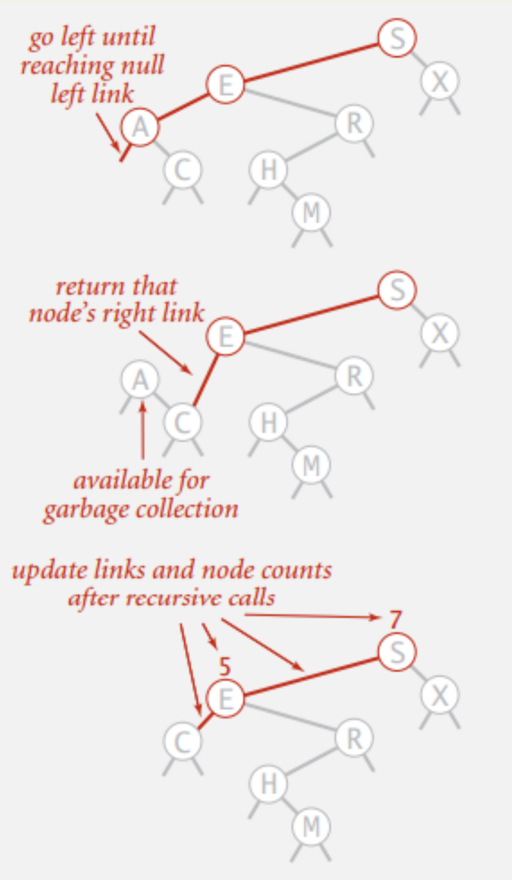
排名和选择

- 问题.如何有效地实现rank( )和select( )?
- A. 在每个节点中，我们存储以该节点为根的子树中的节点数; 要实现size( )，请在根上返回计数



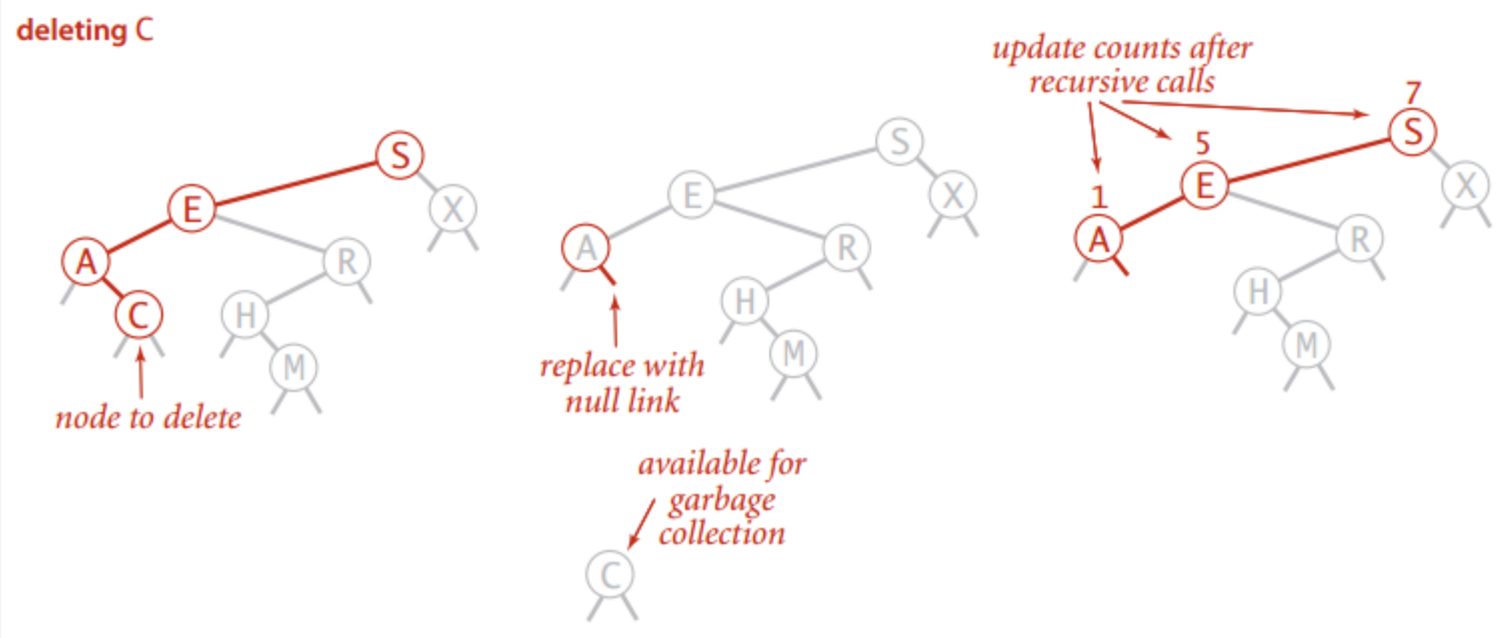
BST 删除

- 最复杂的二叉搜索树操作
- 我们必须确保当我们移除一个元素时,我们维护二叉搜索树属性
- 让我们先看看DeleteMin ...
- 删除最小密钥
  - 向左移动直到找到带有空左链接的节点
  - 用正确的链接替换该节点
  - 更新子树计数.



Hibbard 删除

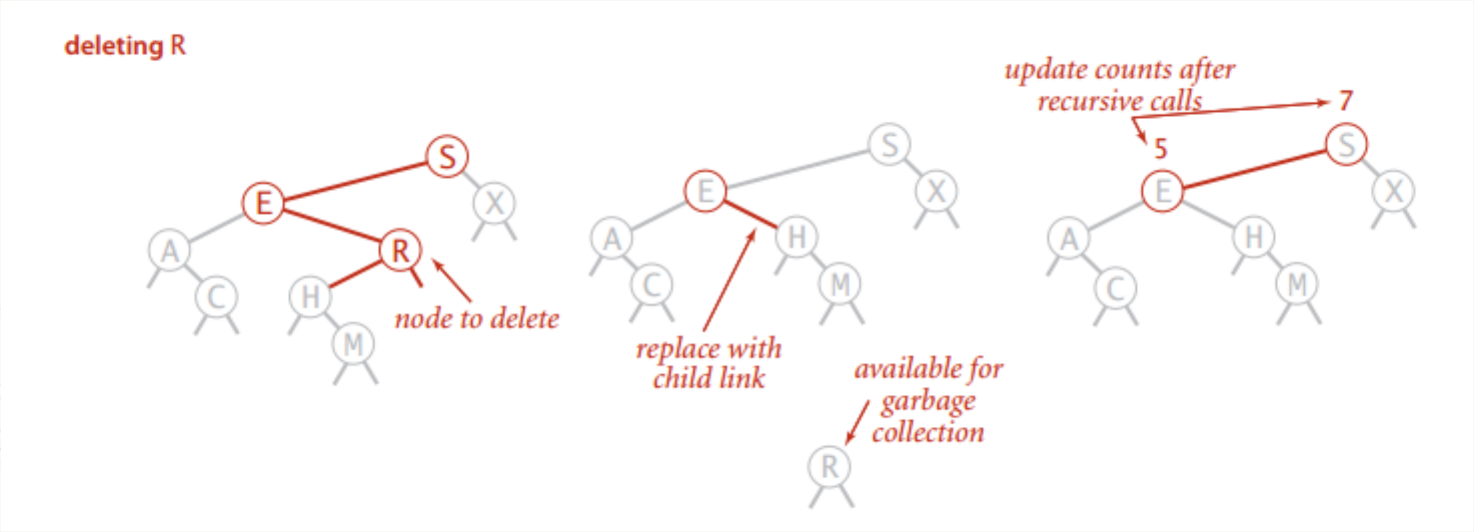
- 用键k删除节点：搜索包含键k的节点t
- Case 1. *[0 children]* 通过将父链接设置为空来删除t



Hibbard 删除

CONTINUED

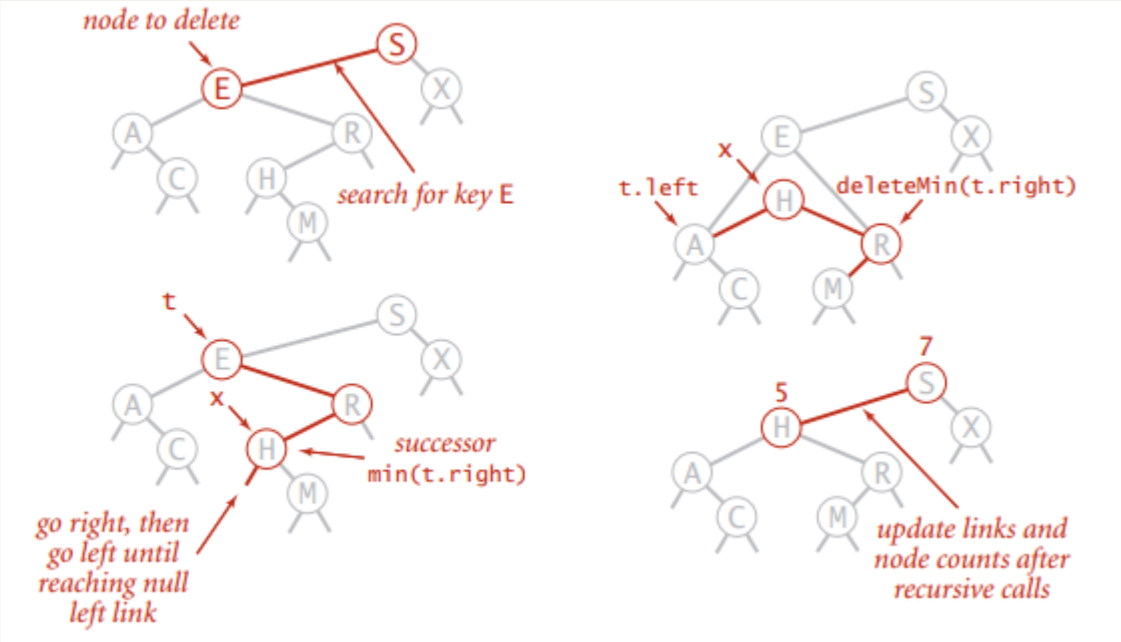
- 用键k删除节点：搜索包含键k的节点t
- Case 2. *[1 children]* 通过替换父链接来删除t.



Hibbard删除

CONTINUED

- 用键k删除节点：搜索包含键k的节点t
- Case 3. *[2 children]*
  - 找到t的后继x
  - 删除t的右侧子树中的最小值
  - 把x放在t的地方



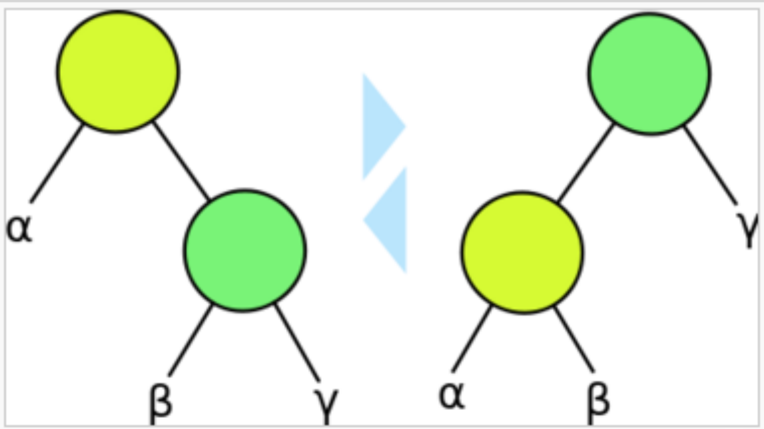
性能摘要

	Binary Search	ArrayList	LinkedList	BST
Search	lg N	N	N	*h*
Insert	N	N	N	*h*
min/max	1	N	N	*h*
floor/Ceiling	lg N	N	N	*h*
Rank	lg N	N	N	*h*
Get	1	1	N	*h*
Ordered Iteration	N	N lg N	N lg N	N

h = height of BST
(proportional to log N if keys inserted in random order)

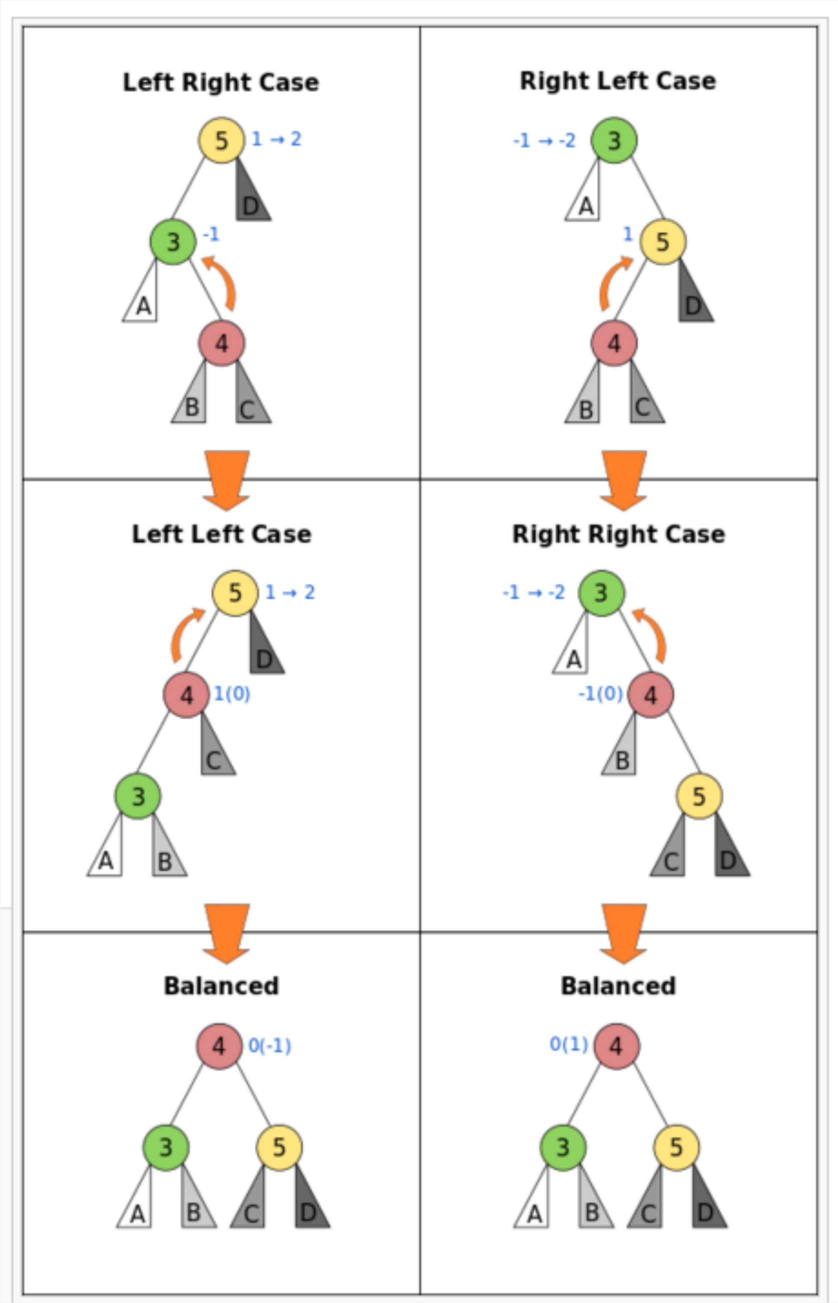
AVL 树

- Georgy Adelson-Velsky and Evgenii Landis' tree, 1962
- 任何节点的两个子树的高度相差最多一个
- 如果在任何时候它们相差多于一个，则重新平衡以恢复这个属性
- 查找，插入和删除都在平均和最差情况下都需要O(log n)时间
- 树旋转



Tree rotations

树旋转

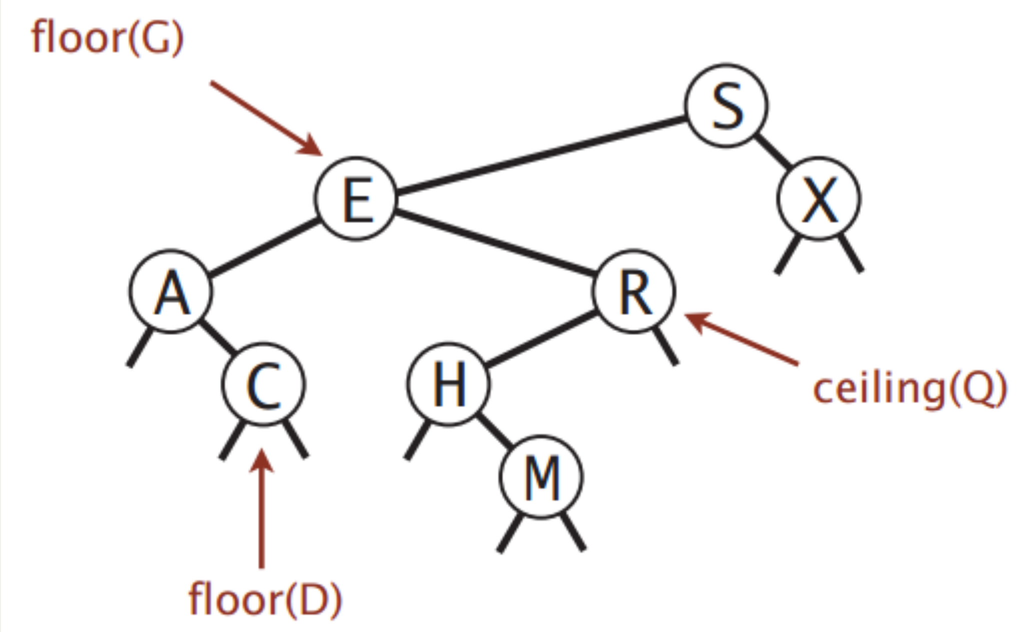

Left Right Case
Right Left Case
Left Left Case
Right Right Case
Balanced
Balanced
1 → 2
-1
-1 → -2
1
1(0)
-1(0)
0(-1)
0(1)

练习 I

- 树的大小
- 最大深度 (104E)
  - 给定一个二叉树,计算它的 “maxDepth” - 从根节点到最远的叶节点沿最长路径的节点数.
- 是树的平衡树? (101E)
  - 平衡树定义为树，使得两个叶节点在距离根上的距离不超过一个.

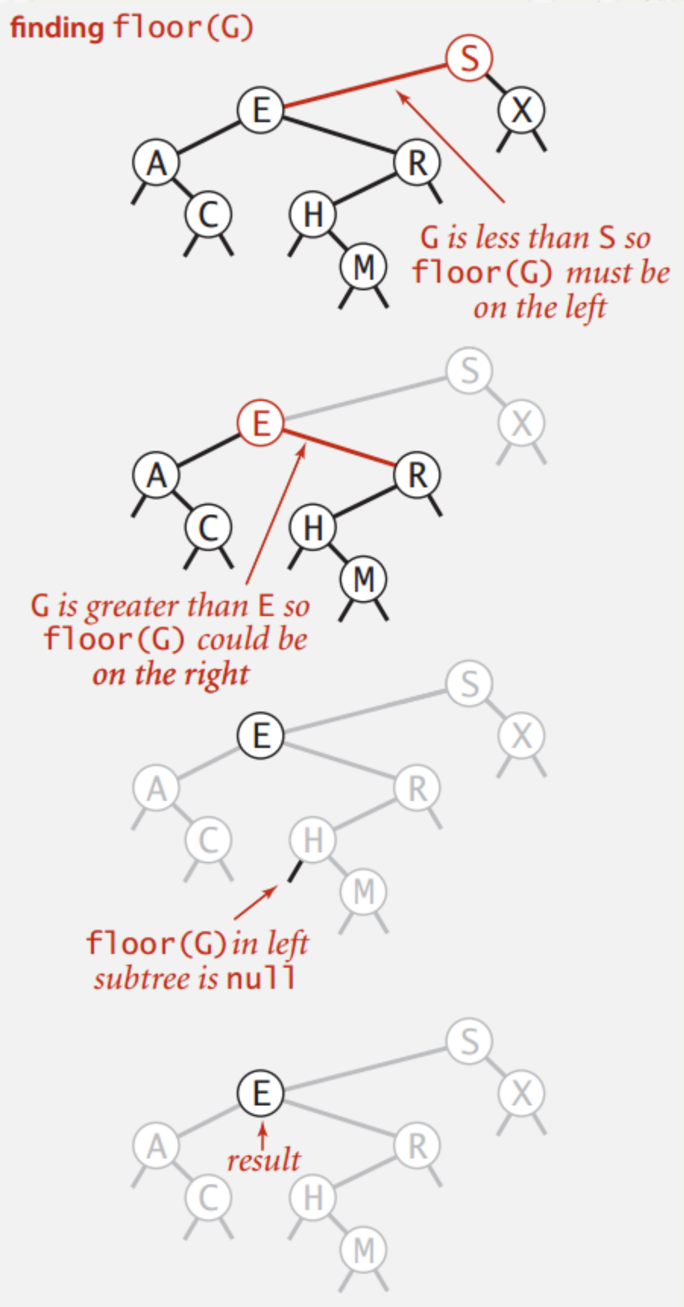
练习 II

- Floor.最大密钥≤给定密钥.
- Ceiling.最小密钥≥给定密钥
- Q. 如何找到 floor / ceiling?



计算 Floor

- Floor.最大密钥≤给定密钥.
- Q.如何找到floor / ceiling?
- Case 1. [k equals the key in the node]
  - k的floor是k.
- Case 2. [k is less than the key in the node]
  - k的底部位于左侧子树中.
- Case 3. [k is greater than the key in the node]
  - k的底层位于右侧子树(如果右侧子树中有任何密钥≤k);
  - 否则它是节点中的关键.

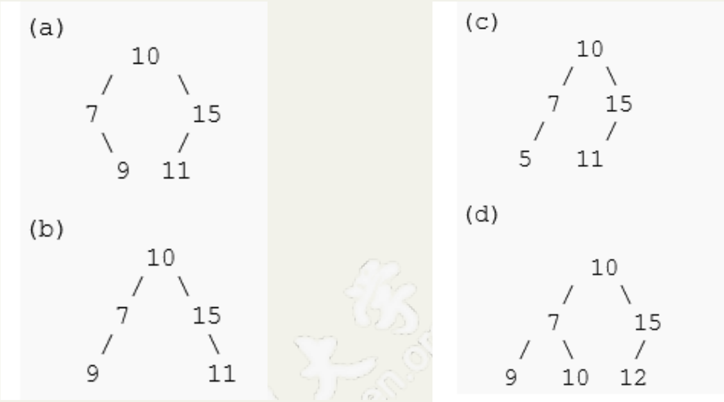


练习 III

CONTINUED

- 是一棵树BST? (98M)
- 镜像
  - 更改一棵树，以便在每个节点交换左指针和右指针的角色

- 同一树 (100E)
  - 给定两个二叉树，如果它们在结构上相同，则返回true - 它们由具有相同值的节点组成
- 可折叠的树
  - 如果树的左右子树是彼此结构清晰的镜像，树可以折叠。 一棵空树被认为是可折叠的



练习 IV

CONTINUED

- 迭代获取方法
- 迭代加法
- 迭代中间遍历
- 迭代预遍历
- 迭代后序遍历

练习 V

CONTINUED

- 二叉树级别遍历 (102E)
  - 给定一个二叉树，返回其节点值的级别遍历。 (即从左到右，逐级).
  - 例如：给定二叉树 {3,9,20,#,#,15,7}

```
    3
   / \
  9  20
    /  \
   15   7
```

```
[
  [3],
  [9,20],
  [15,7]
]
```

练习 VI

- 二叉树级别遍历 II (107E)
  - 给定一个二叉树，返回其节点值的自底向上的级别遍历。(即从左到右，从叶到根的层次).
  - 例如：给定二叉树 {3,9,20,#,#,15,7},

```
   3
  / \
 9  20
   /  \
  15   7
```

```
[
  [15,7]
  [9,20],
  [3],
]
```

练习 VII

CONTINUED

- 二叉树锯齿水平顺序遍历 (103M)
  - 给定一个二叉树，返回其节点值的锯齿级顺序遍历.(即从左到右，然后从右到左进入下一个等级并在其间交替).

```
    3
   / \
  9  20
    /  \
   15   7
```

```
[
  [3],
  [20,9],
  [15,7]
]
```

练习 VIII

- 从二阶和二阶遍历构造二叉树
  - 给定树的前序和逆序遍历，构造二叉树.
  - preorder = [3,9,20,15,7]
  - inorder = [9,3,15,20,7]

```
    3
   / \
  9  20
    /  \
   15   7
```

- 从二阶和后序遍历构造二叉树
  - 给定一个树的后序遍历和后序遍历，构造二叉树.
  - inorder = [9,3,15,20,7]
  - postorder = [9,15,7,20,3]

```
    3
   / \
  9  20
    /  \
   15   7
```

练习 IX

- 有路径和?
  - 给定一棵二叉树和一个和，确定树是否有根到叶的路径，这样沿路径加起来的所有值等于给定的总和.

```
      5
     / \
    4   8
   /   / \
  11  13  4
 /  \      \
7    2      1
```

- path 1: 5 4 11 7
- path 2: 5 4 11 2
- path 3: 5 8 13
- path 4: 5 8 4 1
- Give 27, return path 1

- 有路径和 II?
  - 给定一个二叉树和一个和，找到每个路径的总和等于给定总和的所有根到叶路径.

```
      5
     / \
    4   8
   /   / \
  11  13  4
 /  \    / \
7    2  5   1
```

- Given sum = 22
- [[5,4,11,2], [5,8,4,5]]

练习 X

- 有路径和 III?
  - 给你一个二叉树，其中每个节点都包含一个整数值.
  - 查找总和给定值的路径数量.
  - 路径不需要在根或叶上开始或结束，但必须向下（仅从父节点到子节点）.

```
      10
     /  \
    5   -3
   / \    \
  3   2   11
 / \   \
3  -2   1
```

- Sum = 8, Return 3

练习 VIII

CONTINUED

- 将数组排序为最小高度的二叉树 (108M)
- 二叉搜索树的第一个共同祖先 (235E)
  - 给定*二叉搜索树中两个节点的值，编写一个程序来查找最低的共同祖先
- 二叉树的第一个共同祖先 (236M)
- 子树
  - 你有两个非常大的二叉树：T1和数百万个节点，T2和数百个节点。 创建一个算法来决定T2是否是T1的子树

回顾

- 树
  - 数据成员
  - 操作
- 二叉树
- 二叉搜索树
- 进一步阅读：红黑树

数据结构与算法

Data Structure and Algorithm

XIV. 树
结束

授课人：Kevin Feng

翻译　：梁少华