湖南大学
HUNAN UNIVERSITY

第六章 扩张的数据结构

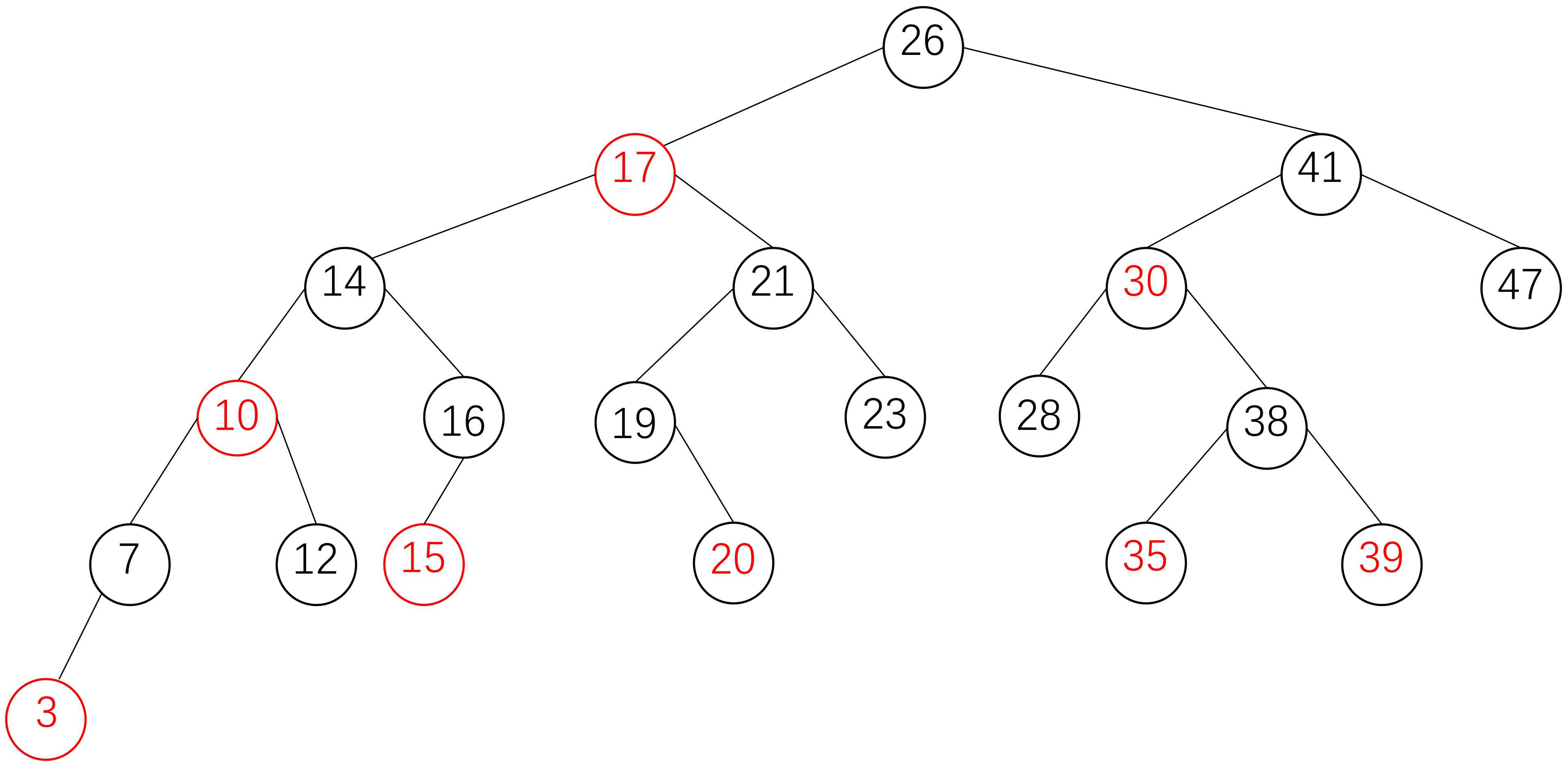
湖南大学信息科学与工程学院

红黑树回顾

一颗红黑树是满足下面红黑性质的二叉搜索树：

- 每个结点或是红色的，或是黑色的
- 根结点是黑色的
- 每个叶结点（NIL）是黑色的
- 如果一个结点是红色的，则它的两个子结点都是黑色的
- 对每个结点，从该结点到其所有后代叶结点的简单路径上，均包含相同数目的黑色结点

红黑树回顾



目录

动态顺序统计

一个元素的**秩**，即它在集合线性序中的位置。

顺序统计树T只是简单地在每个结点上存储附加信息的一棵红黑树，在红黑树的结点x中，出了通常属性的x.key、x.color、x.p、x.left和x.right之外，还包括另一个属性x.size。这个属性包含了以x为根的子树（包括x本身）的（内）结点数，即这颗子树的大小。如果定义哨兵的大小为0，也就是设置T.nil.size为0，则有等式：

$$x.size = x.left.size + x.right.size + 1$$

动态顺序统计

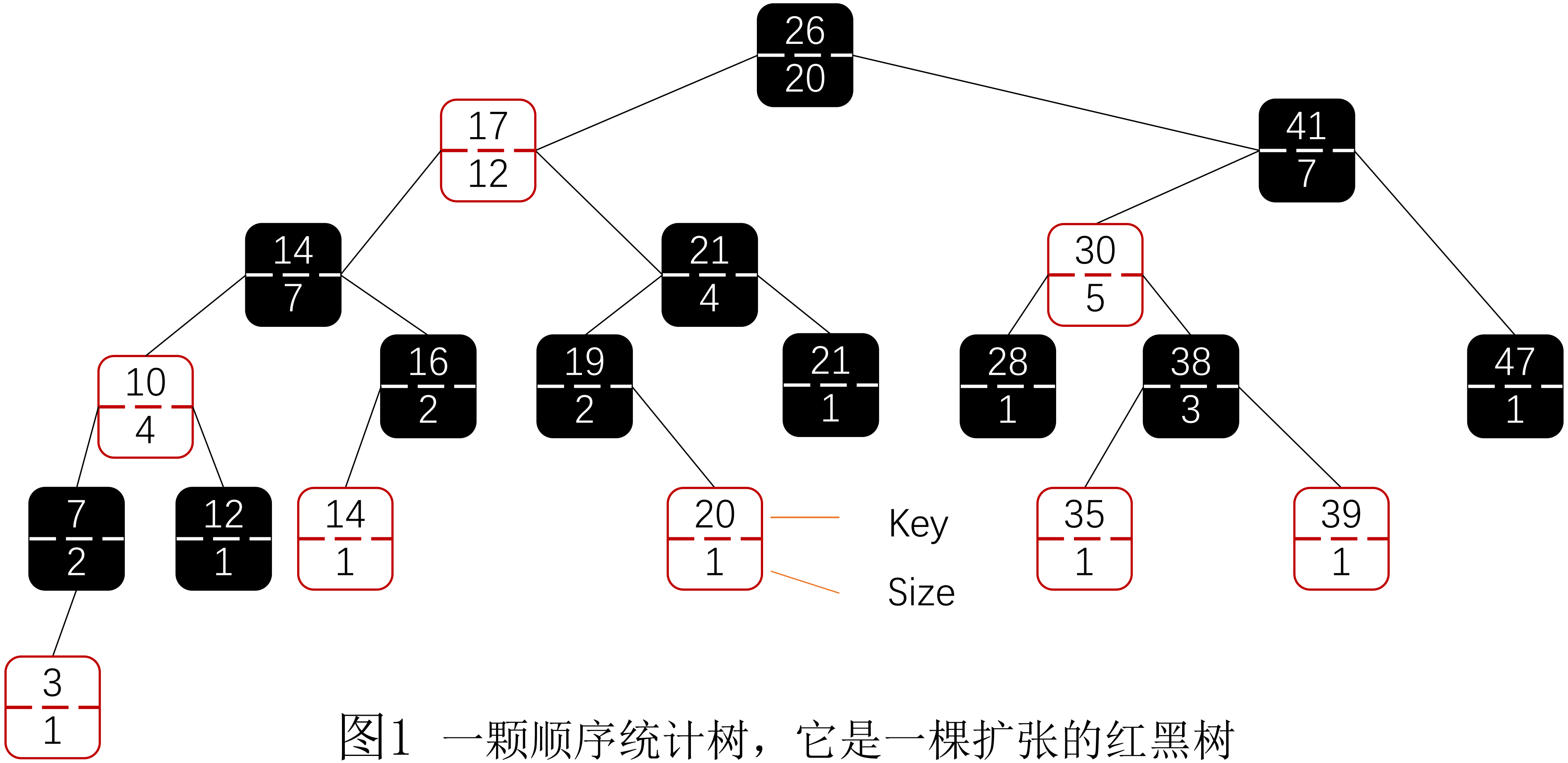


图1　一颗顺序统计树，它是一棵扩张的红黑树

动态顺序统计

查找具有给定秩的元素

在说明插入和删除过程中如何维护 size 信息之前，我们先来讨论利用这个附加信息来实现的两个顺序统计查询。首先一个操作是对具有给定秩的元素的检索。过程OS-SELECT（x,i）返回一个指针，其指向以x为根的子树中包含第i小关键字的结点。为找出顺序统计树T中的第i小关键字，我们调用过程OS-SELECT（T,root,i）。

动态顺序统计

```
OS-SELECT (x,i)
r = x.left.size + 1
if i == r
    return x
else i < r
    return OS-SELECT(x,left,i)
else return OS-SELECT(x,right,i - r)
```

动态顺序统计

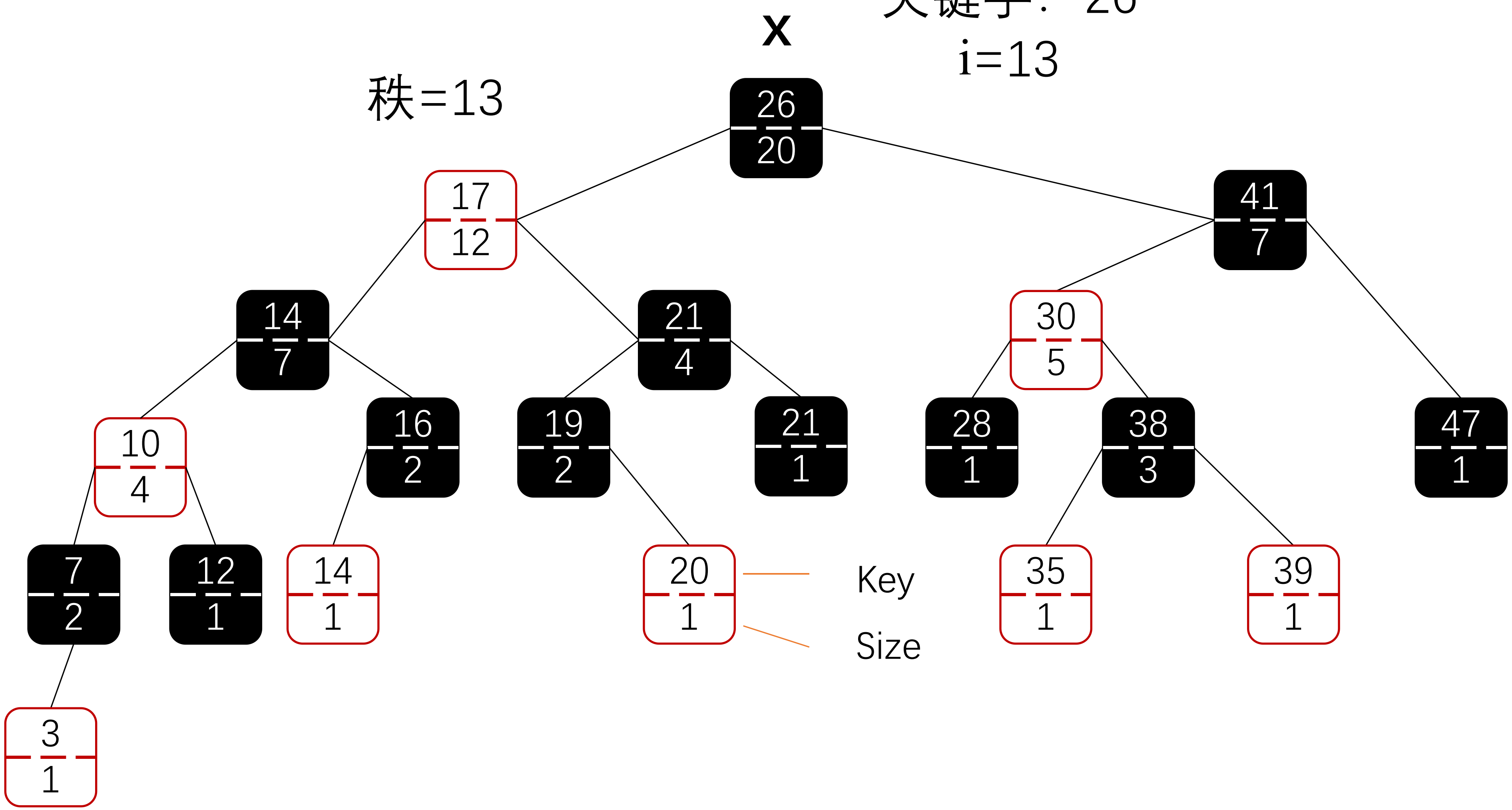
查找第17小的元素

- 以 x 为根开始，其关键字为 26，i=17。因为 26 的左子树的大小为 12，故它的秩为 13。因此，秩为 17 的结点是 26 的右子树中第17—13=4 小的元素。
- 递归调用后，x 为关键字 41 的结点，i=4。因为 41 的左子树大小为 5，故它的秩为 6。这样，可以知道秩为 4 的结点是 41 的左子树中第 4 小元素。
- 再次递归调用后，x 为关键字30的结点，在其子树中它的秩为 2。
- 再进行一次递归调用，就能找到以关键字 38 的结点为根的子树中第 4-2=2 小的元素。它的左子树大小为 1，这意味着它就是第 2 小元素。
- 最终，该过程返回一个指向关键字为 38 的结点的指针。

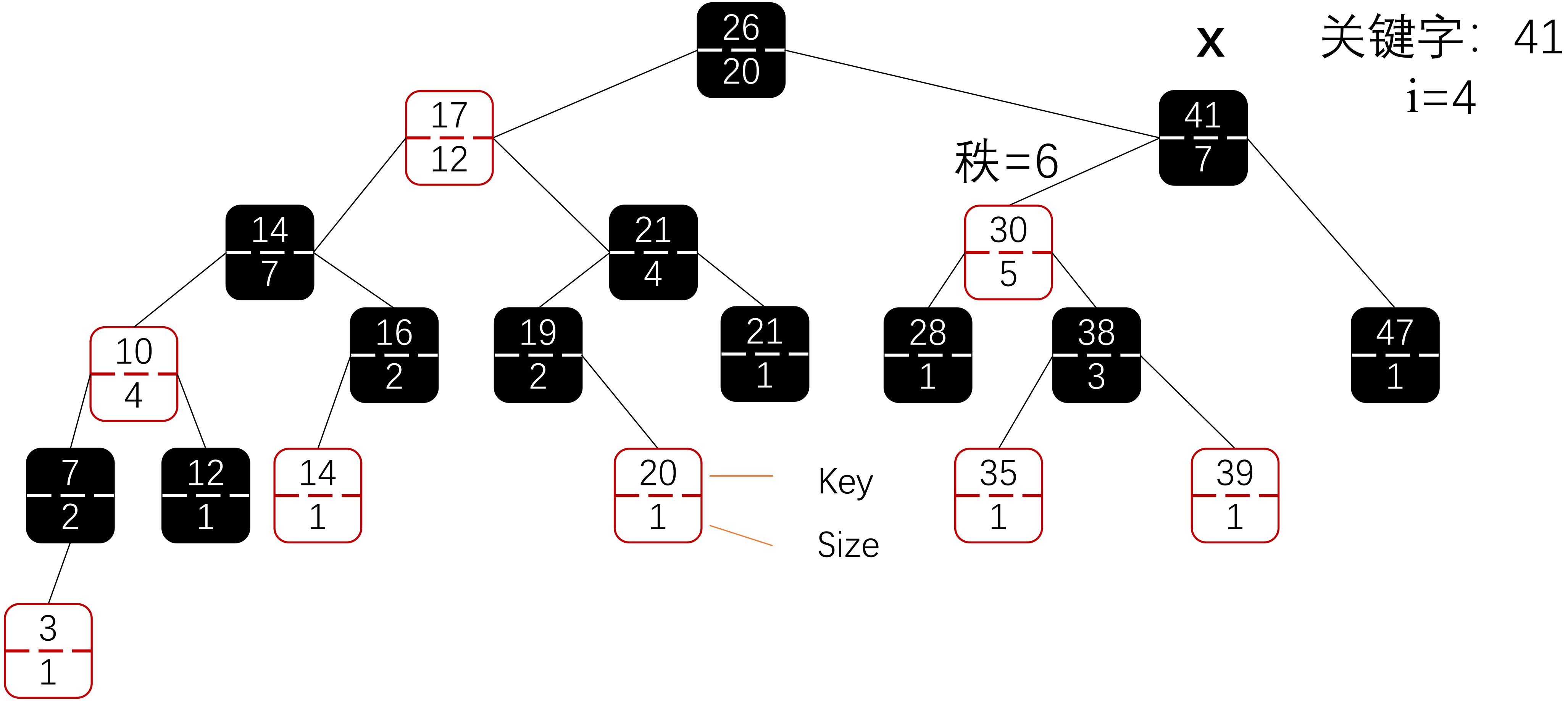
动态顺序统计

关键字：26

i=13

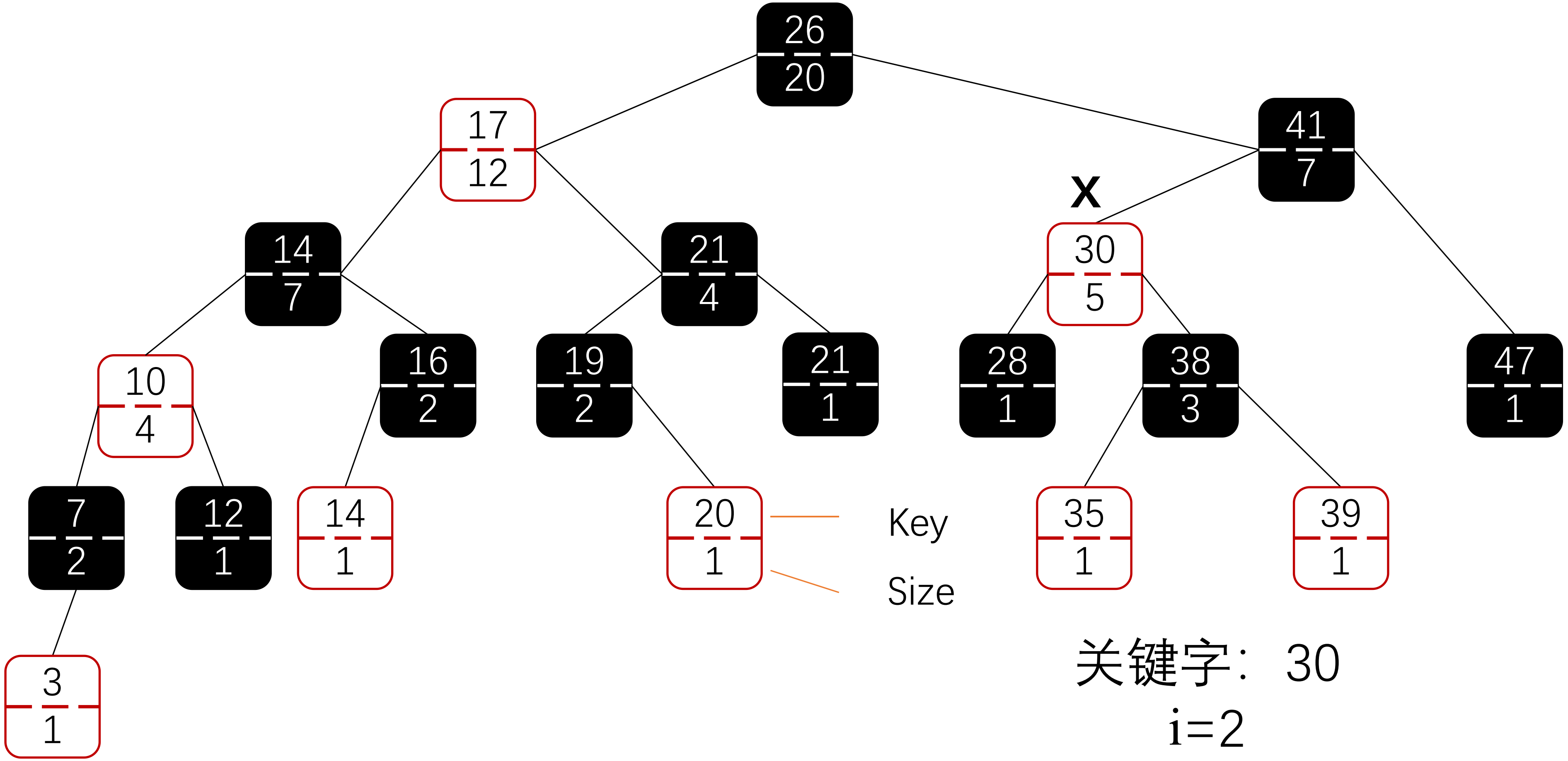


动态顺序统计



26
20
17
12
14
7
21
4
10
4
16
2
19
2
21
1
7
2
12
1
14
1
20
1
3
1
Key
Size
X
关键字：41
i=4
41
7
秩=6
30
5
28
1
38
3
47
1
35
1
39
1

动态顺序统计

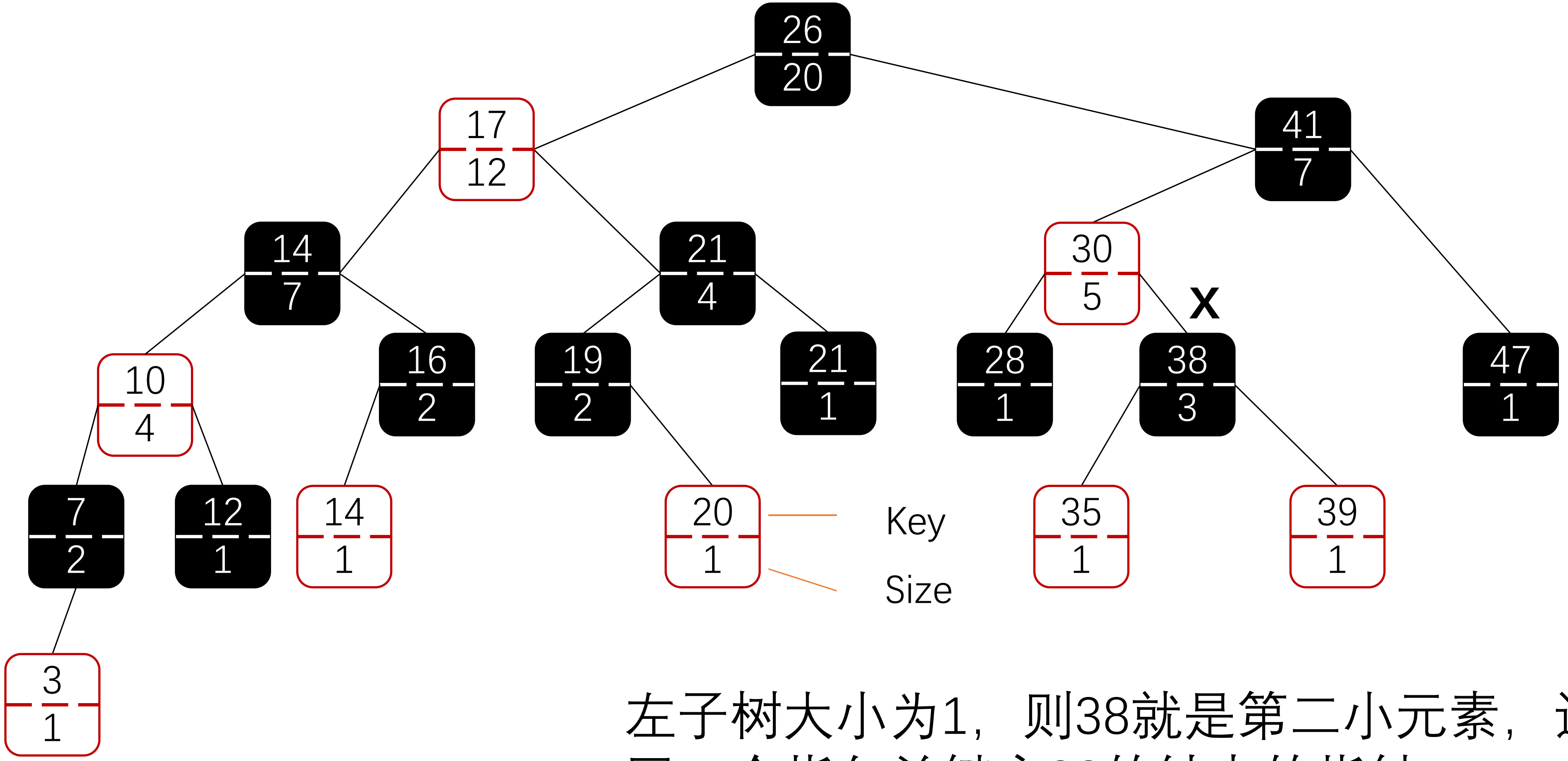


关键字：30

i=2

秩=2

动态顺序统计



左子树大小为1，则38就是第二小元素，返回一个指向关键字38的结点的指针

动态顺序统计

```
OS-SELECT (x,i)
r = x.left.size + 1
if i == r
    return x
else i < r
    return OS-SELECT(x,left,i)
else return OS-SELECT(x,right,i - r)
```

因为每次递归调用都在顺序统计树中下降一层，OS-SELECT 的总时间最差与树的高度成正比。又因为该树是一棵红黑树，其高度为 O(lg n)，其中n为数的结点数。所以，对于n个元素的动态集合，OS-SELECT 的运行时间为 **O(lg n)**。

动态顺序统计

确定一个元素的秩

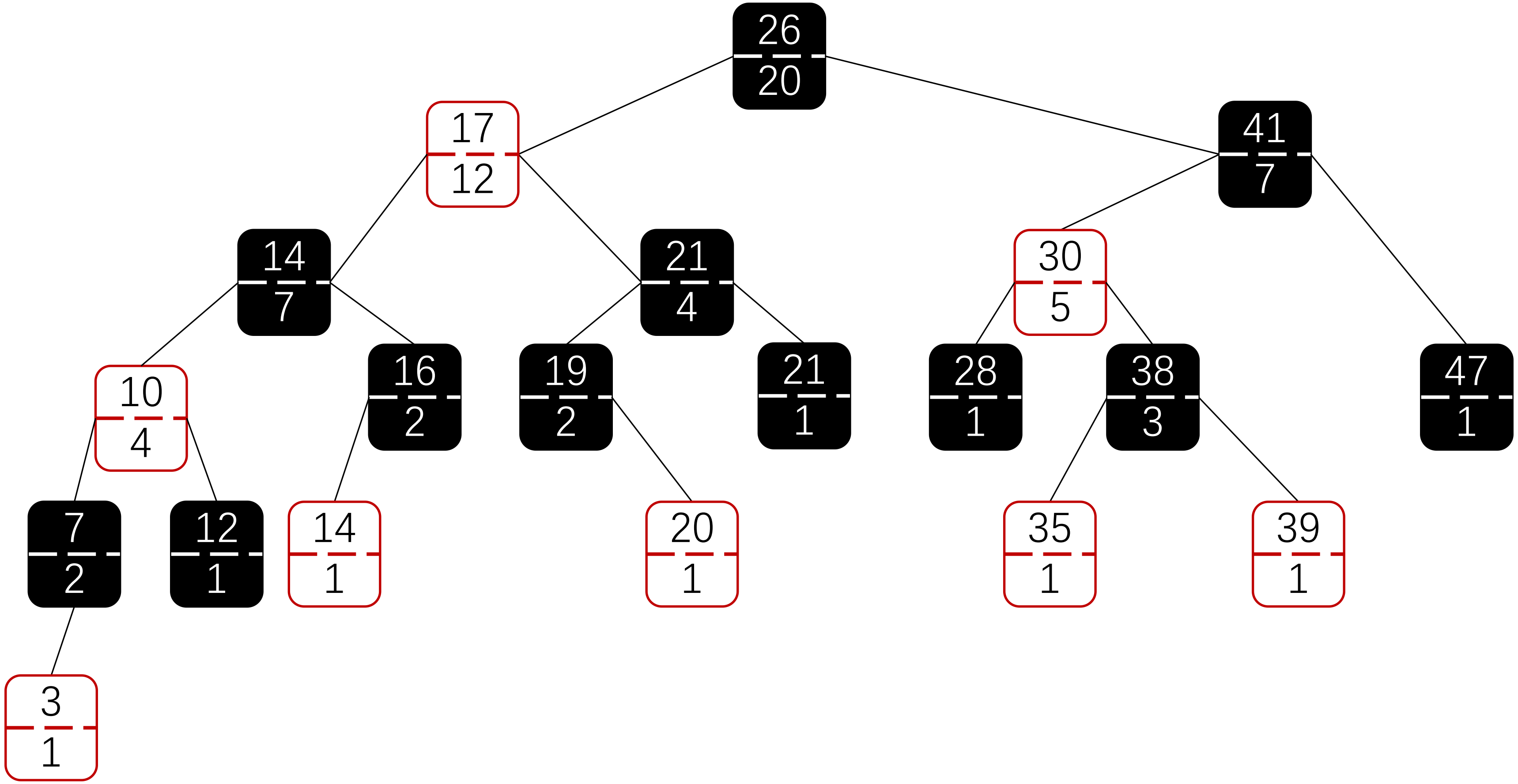
给定指向顺序统计树 T 中结点 x 的指针，过程 OS-RANK 返回对 T 中序遍历对应的线性序中 x 的位置

OS-RANK(T,x)

```
r = x.left.size + 1
y = x
while y ≠ T.root
    if y == y.p.right
        r = r + y.p.left.size + 1
    y = y.p
return r
```

动态顺序统计

确定关键字为38的结点的秩



动态顺序统计

确定关键字为38的结点的秩

迭代	y.key	r
1	38	2
2	30	4
3	41	4
4	26	17

返回的秩为17

动态顺序统计

确定一个元素的秩

因为 while 循环的每次迭代耗费 O(1)时间，且 y 在每次迭代中沿树上升一层，所以最坏情况下 OS-RANK 的运行时间与树的高度成正比：在n个结点的顺序统计树上为O(lg n)。

OS-RANK(T,x)

```
r = x.left.size + 1
y = x
  while y ≠ T.root
    if y == y.p.right
          r = r + y.p.left.size + 1
     y = y.p
 return r
```

对子树规模的维护

给定每个结点的 size 属性后，OS-SELECT 和 OS-RANK 能迅速计算出所需的顺序统计信息。然而除非能用红黑树上经过修改的基本操作对 size 属性加以有效的维护，否则，我们的工作将变得没意义。下面就来说明在不影响插入和删除操作的渐近运行时间的前提下，如何维护子树规模。

对子树规模的维护

红黑树上的**插入**操作包括两个阶段。

第一阶段从根开始沿树下降，将新结点插人作为某个已有结点的孩子。

第二阶段沿树上升，做一些变色和旋转操作来保持红黑树性质。

对子树规模的维护

在第一阶段中为了维护子树的规模，对由根至叶子的路径上遍历的每一个结点飞，都增加x.size属性。新增加结点的 size 为 1。由于一条遍历的路径上共有O(lg n)个结点，故维护 size 属性的额外代价为 O(lg n)。

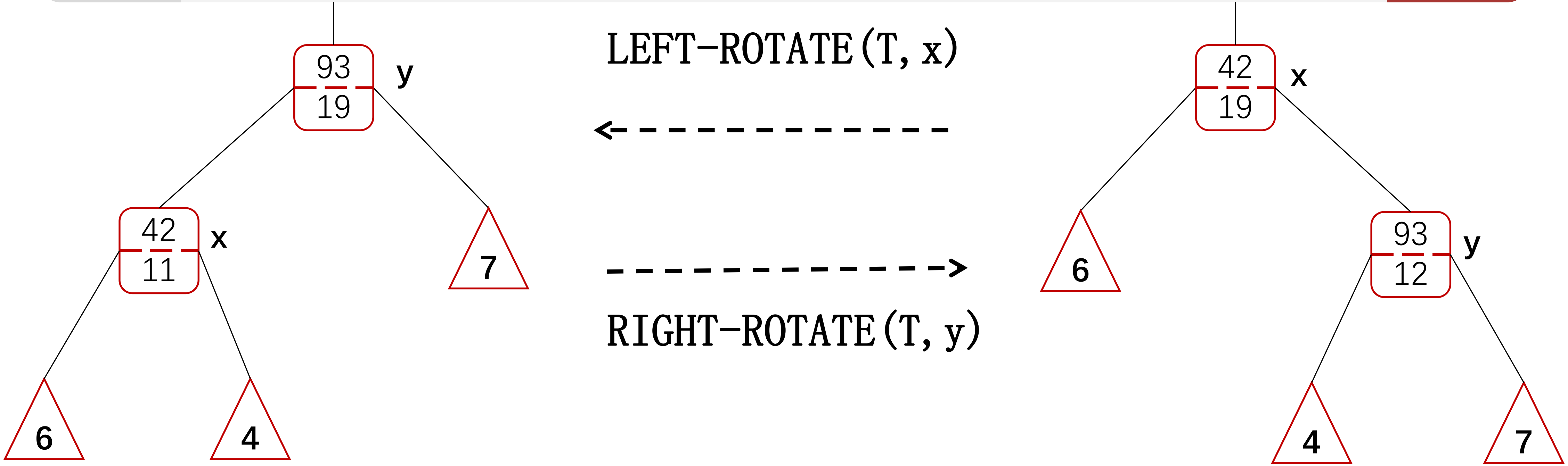
动态顺序统计

对子树规模的维护

在第二阶段，对红黑树结构上的改变仅仅是由旋转所致，旋转次数至多为2。此外，旋转是一种局部操作：它仅会使两个结点的 size属性失效，而围绕旋转操作的链就是与这两个结点关联。于是增加以下两行代码：

```
1  y.size = x.size
2  x.size = x.left.size + x.right.size + 1
```

动态顺序统计



在旋转过程中修改子树的大小。与围绕旋转的链相关联的两个结点，它们的 size 属性要更新。这些更新是局部的，仅需要存储在x和y中的size信息，以及图中三角形子树的根中的size信息。

动态顺序统计

因为在红黑树的插入过程中至多进行两次旋转，所以在第二阶段更新size属性只需要O(1) 的额外时间。因此，对一颗有n个结点的顺序统计树插入元素所需要的总时间为O(1g n)。

动态顺序统计

红黑树上的**删除**操作也包括两个阶段。

第一阶段对搜索树进行操作，第二阶段至多做三次旋转，其他对结构没有任何影响。第一阶段中，要么将结点y从树中删除，要么将它在树中上移。为了更新子树的规模，我们只需要遍历一条由结点y至根的简单路径，并减少路径上每个结点的size属性的值。

第一阶段维护 size 属性所耗费的额外时间为 O(1g n)，第二阶段采取与插入相同的方式来处理删除操作中的O(1)次旋转，所以总共需要O(1g n)的时间。

湖南大学 HUNAN UNIVERSITY

目录

如何扩张数据结构

扩张一种数据结构可分为4个步骤

1、选择一种基础数据结构

2、确定基础数据结构中要维护的附加信息

3、检验基础数据结构上的基本修改操作能否维护附加信息

4、设计一些新操作

如何扩张数据结构

对红黑树的扩张

定理1 设f是n个结点的红黑树T扩张的属性，且假设对任一结点x，f的值仅依赖于结点x、x.left和x.right的信息，还可能包括x.left.f和x.right.f。那么，我们可以在插入和删除操作期间对T的所有结点的f值进行维护，并且不影响这两个操作的O(1g n)渐近时间性能。

证明： 证明的主要思想是，对树中某结点x的f属性的变动只会影响到x 的祖先。也就是说，修改x,f只需要更新x.p.f，改变x.p.f的值只需要更新x.p.p.f，如此沿树向上。一旦更新到T.root• f，就不再有其他任何结点依赖于新值，于是过程结束。因为红黑树的高度为O(lg n)，所以改变某结点的f属性要耗费O(lg n)时间，来更新被该修改所影响的所有结点。

目录

区间树

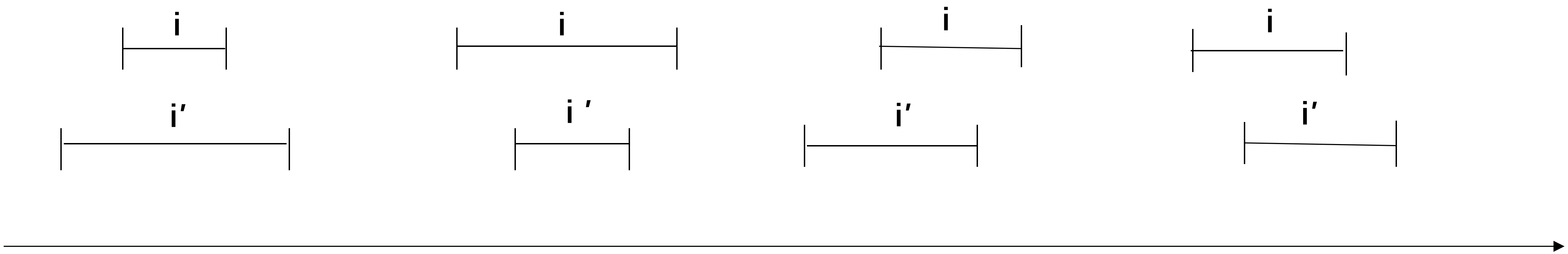
我们可以把一个区间[t_1，t_2]表示成一个对象i，其中属性i.low = t_1为**低端点**(low endpoint)，属性i.high=t_2 为**高端点**（high endpoint）。我们称区间 i 和 i′ **重叠**(overlap），如果$i \cap i' \neq \emptyset$，即如果$i.low \leq i'.high$且$i'.low \leqq i.high$。如图所示，任何两个区间i和i′满足区间三分律(interval trichotomy)，即下面三条性质之一成立：

a. i和i′重叠

b. i在i′的左边(也就是i.high< i′.low）

c. i在i′的右边(也就是i′.high<i.low）

区间树

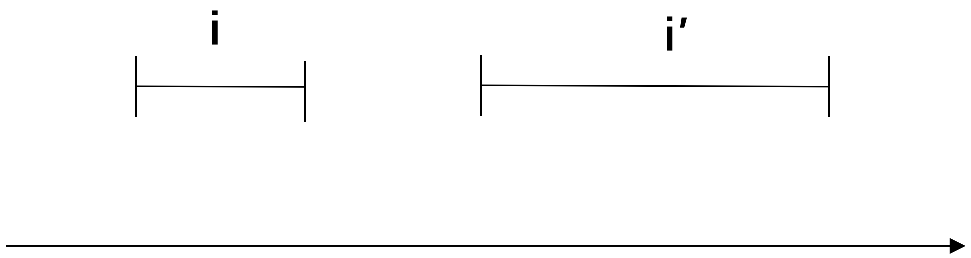


(a)

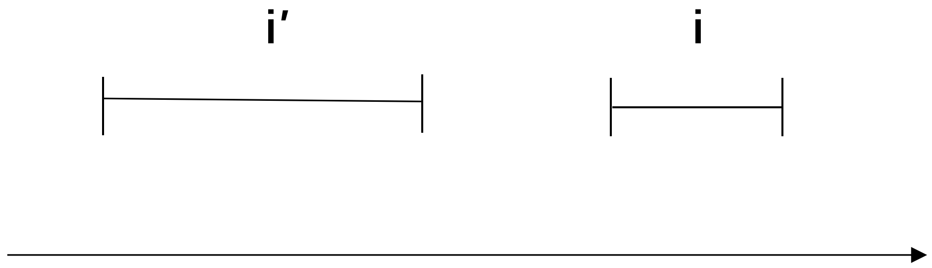
a. 如果i和i′重叠，又分为4种情况，每种情况都有

$\mathrm{i.low} \leq \mathrm{i}'.high$且$\mathrm{i}'.low \leqq i.high$

区间树



(b)



(c)

b. 区间没有重叠且i.high< i′.low

c.区间没有重叠且i′.high<i.low

区间树

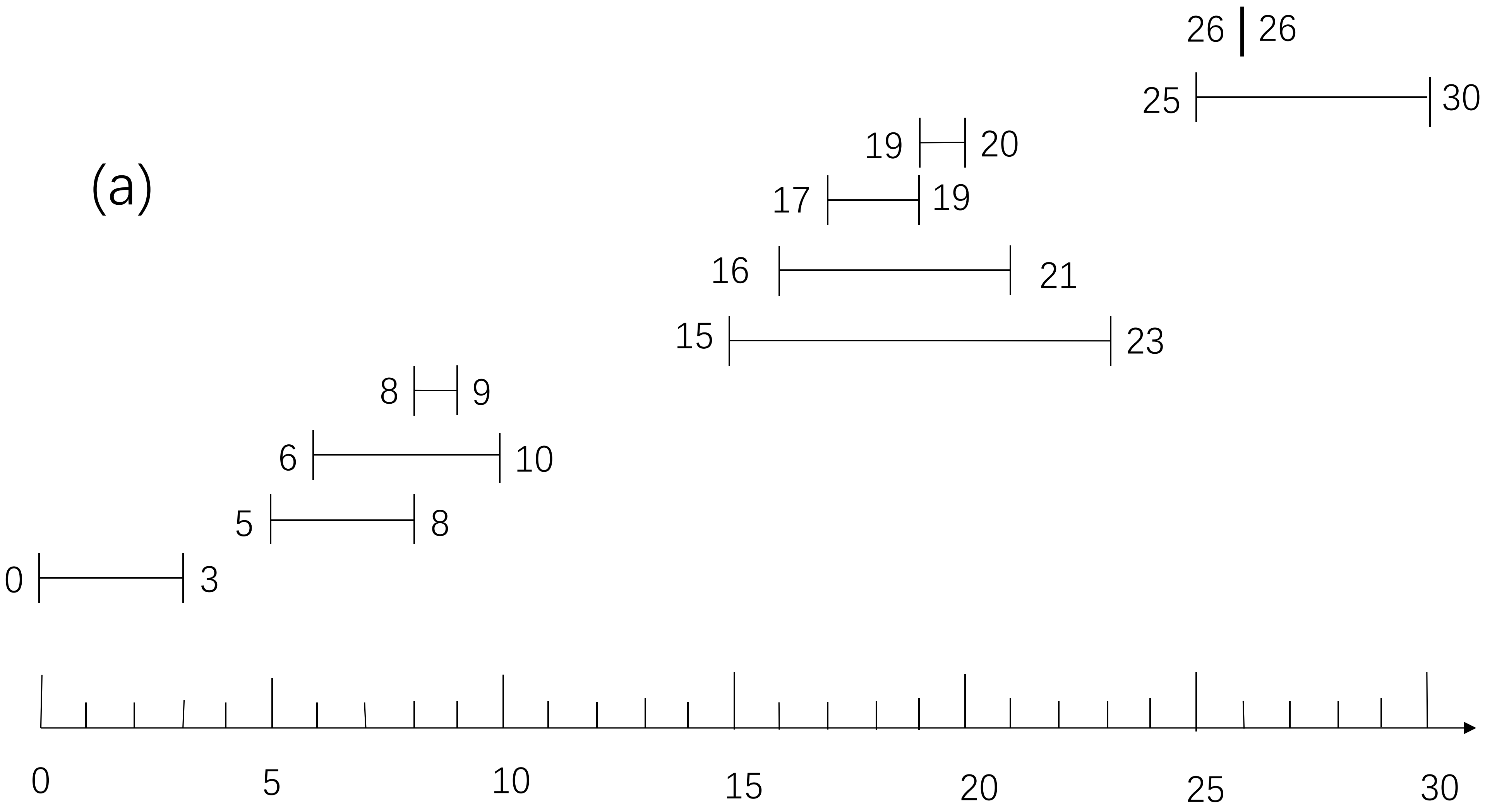
区间树是一种对动态集合进行维护的红黑树，其中每个元素 x 都包含了一个区间 x.int。区间树支持下列操作：

INTERVAL-INSERT(T,x):将包含区间属性int的元素 x插入到区间树T 中。

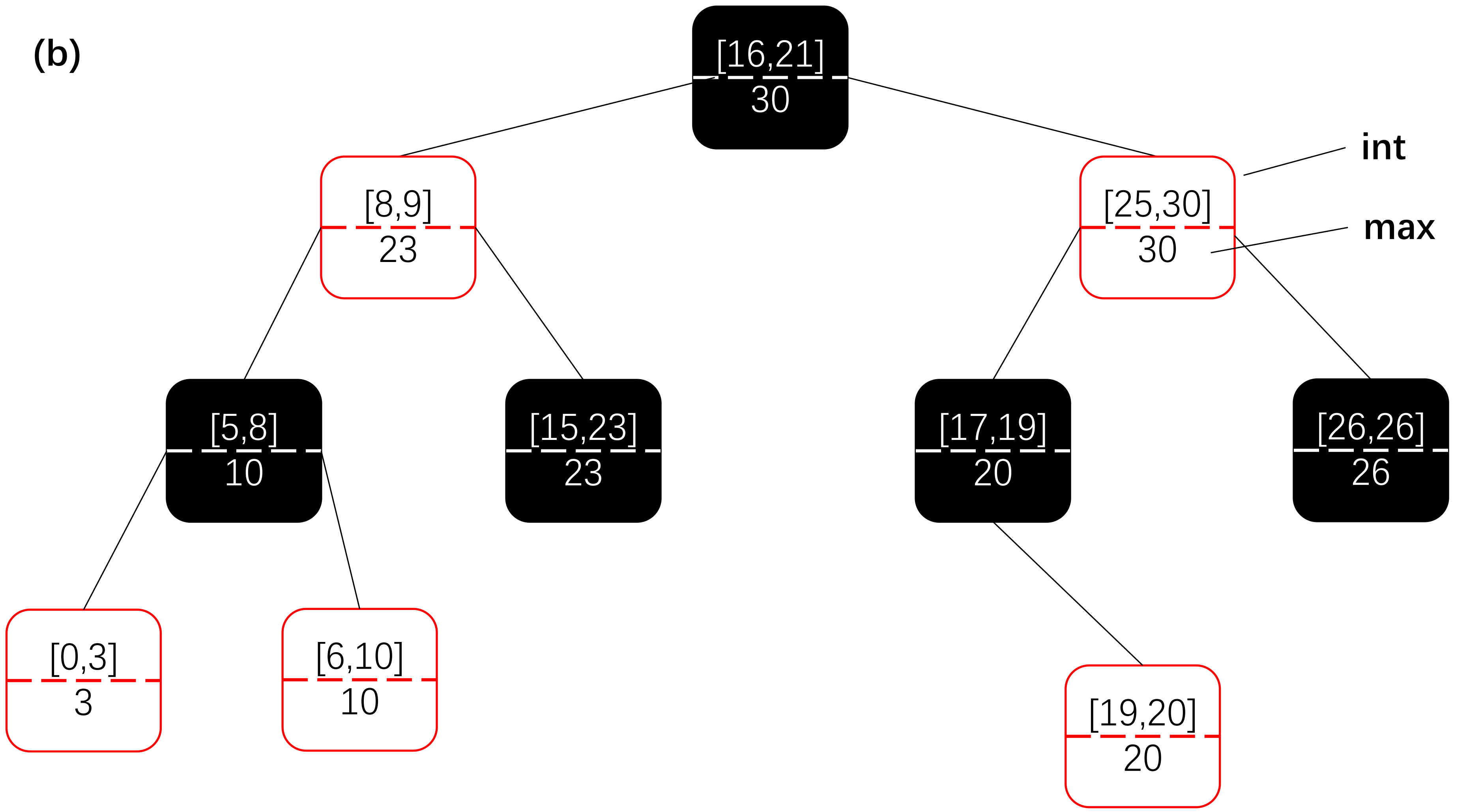
INTERVAL-DELETE(T,x):从区间树T中删除元素x。

INTERVAL-SEARCH(T,i):返回一个指向区间树T中元素 x 的指针，使x.int与 i 重叠；若此元素不存在，则返回T.nil。

用区间树表达一个集合



用区间树表达一个集合


(b)
[16,21]
30
[8,9]
23
[25,30]
30
int
max
[5,8]
10
[15,23]
23
[17,19]
20
[26,26]
26
[0,3]
3
[6,10]
10
[19,20]
20

区间树

步骤1：基础数据结构

选择这样一颗红黑树，其每个结点x包含一个区间x.int，且x的关键字为区间的低端点的次序排列的各区间。

步骤2：附加信息

每个结点x中除了自身区间信息之外，还包含一个值x.max，它是以x为根的子树中所有区间的端点的最大值。

区间树

步骤3：对信息的维护

必须验证n个结点的区间树上的插入和删除操作能否在O(lg n)时间内完成。通过给定区间x.int和结点x的子结点的 max值，可以确定x.max值：

$$x.\mathrm{max} = \mathrm{max}(x.int.high,\ x.left.max,\ x.rightmax)$$

区间树

步骤4：设计新的操作

这里我们仅需要唯一的一个新操作INTERVAL-SEARCH(T,i)，它是用来找出树T中与区间i重叠的那个结点。若树中与i重叠的结点不存在，则算法过程返回指向哨兵T.nil的指针。

```
INTERVAL-SEARCH (T,i)
x = T.root
while x ≠ T.nil and i does not overlap x.int
    if x.left ≠ T.nil and x.left.max ≥ i.low
        x == x.left
    else x == x.right
return x
```

区间树

```
INTERVAL-SEARCH (T,i)
x = T.root
while x ≠ T.nil and i does not overlap x.int
    if x.left ≠ T.nil and x.left.max ≥ i.low
        x == x.left
    else x == x.right
return x
```

时间复杂度：O(lg n)

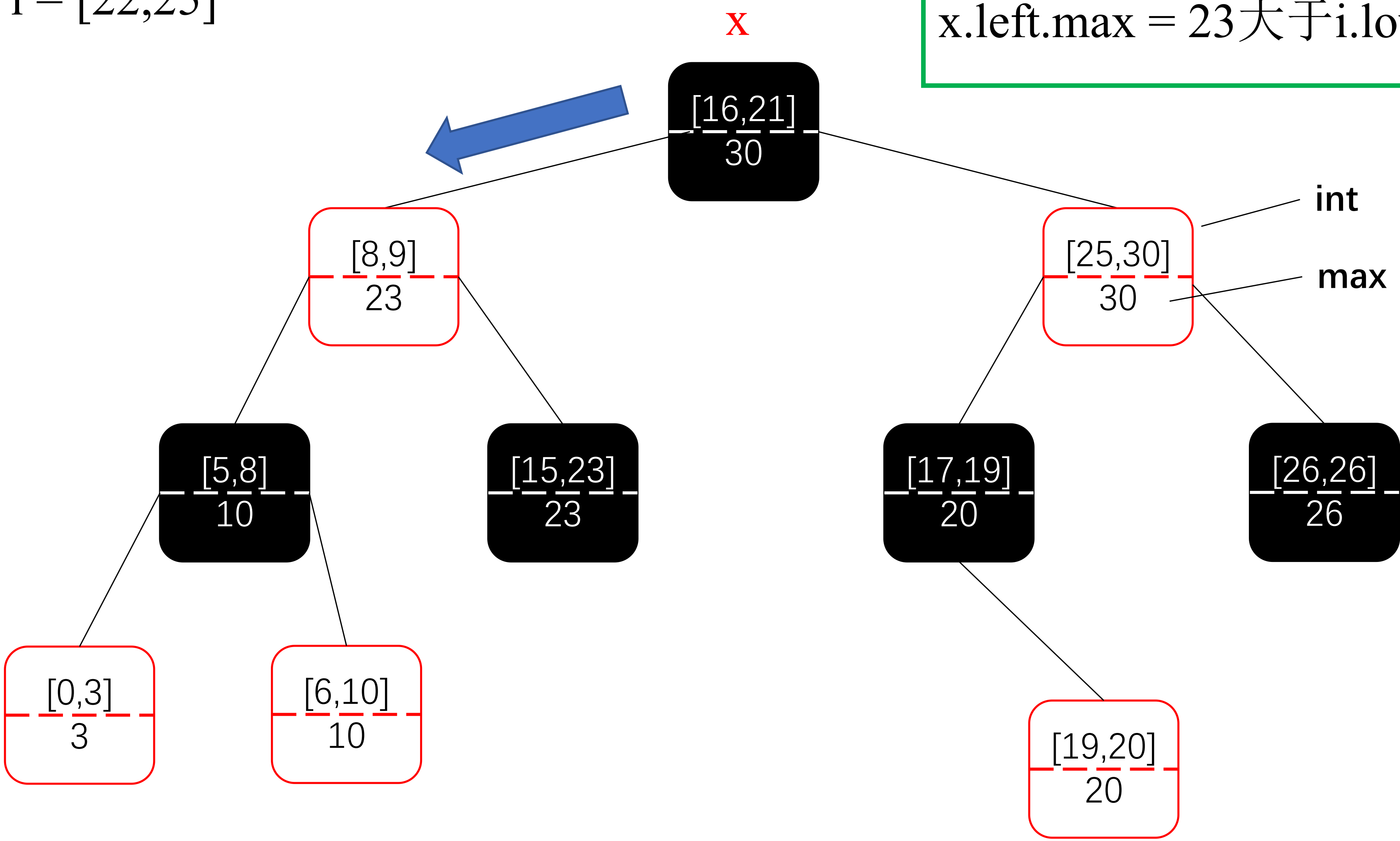
区间树

查找成功

查找一个与区间i = [22,25]重叠的区间。开始时以x为根结点，它包含区间[16，21]，与i不重叠。由于x.left.max = 23大于i.low = 22,所以这时以这棵树根的左孩子作为x继续循环。现在结点x包含区间[8，9]，仍不与i重叠。此时， x.left.max = 10小于i.low = 22,因此以x的右孩子作为新的x继续循环。现在，由于结点x所包含的区间[15，23]与i重叠，过程结束并返回这个结点。

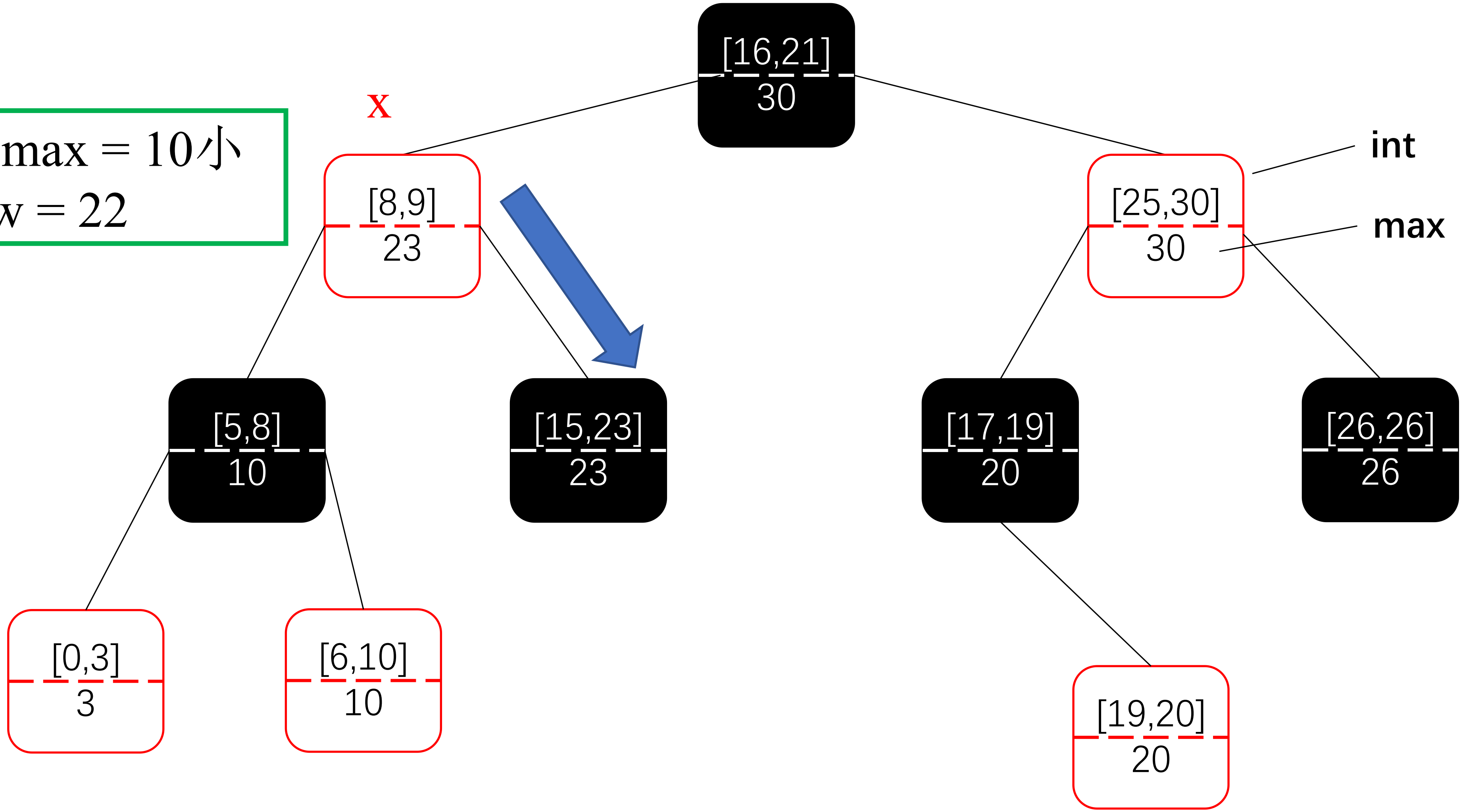
i = [22,25]

x.left.max = 23大于i.low = 22

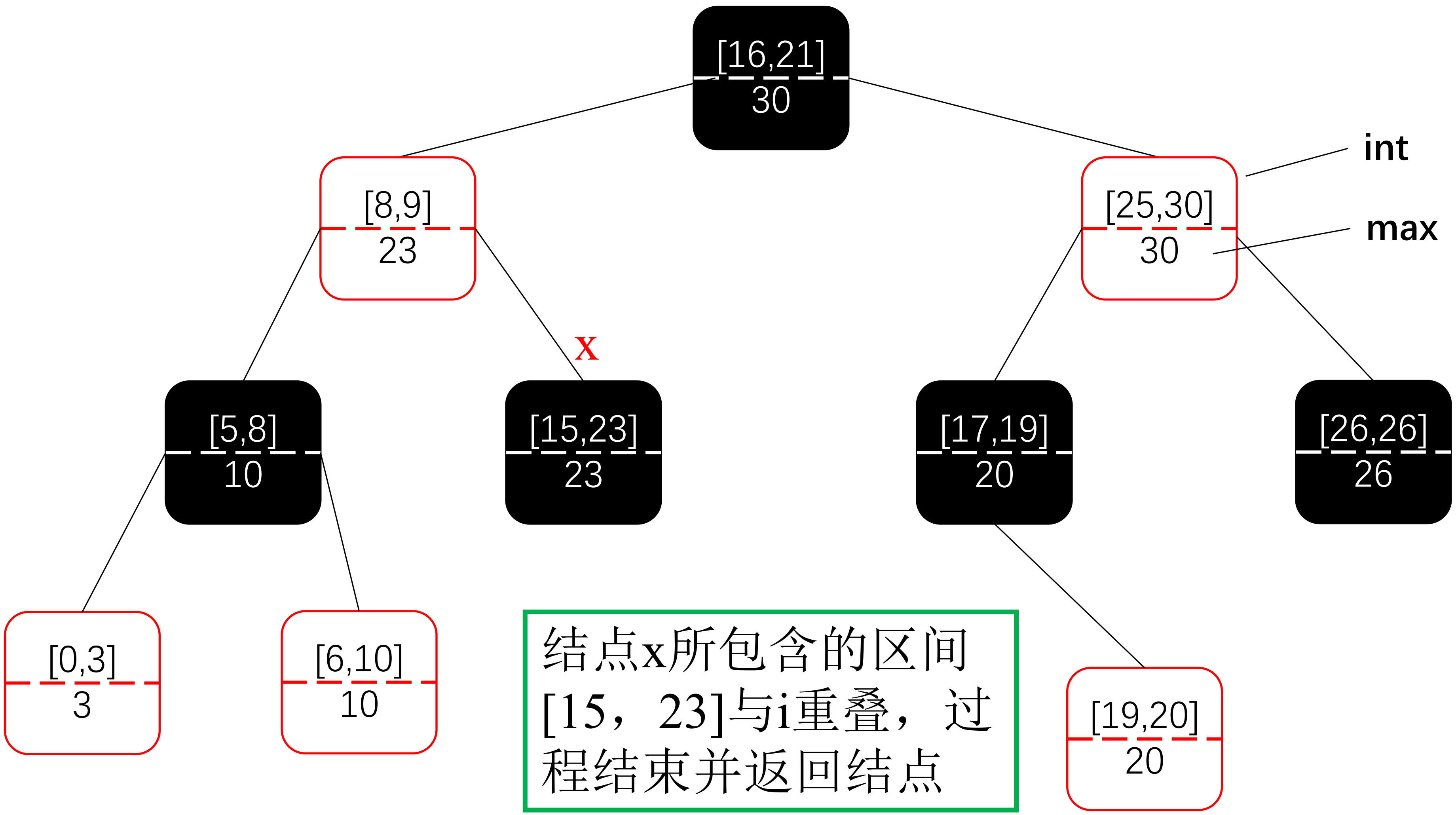


i = [22,25]

x.left.max = 10小于i.low = 22



i = [22,25]



结点x所包含的区间[15，23]与i重叠，过程结束并返回结点

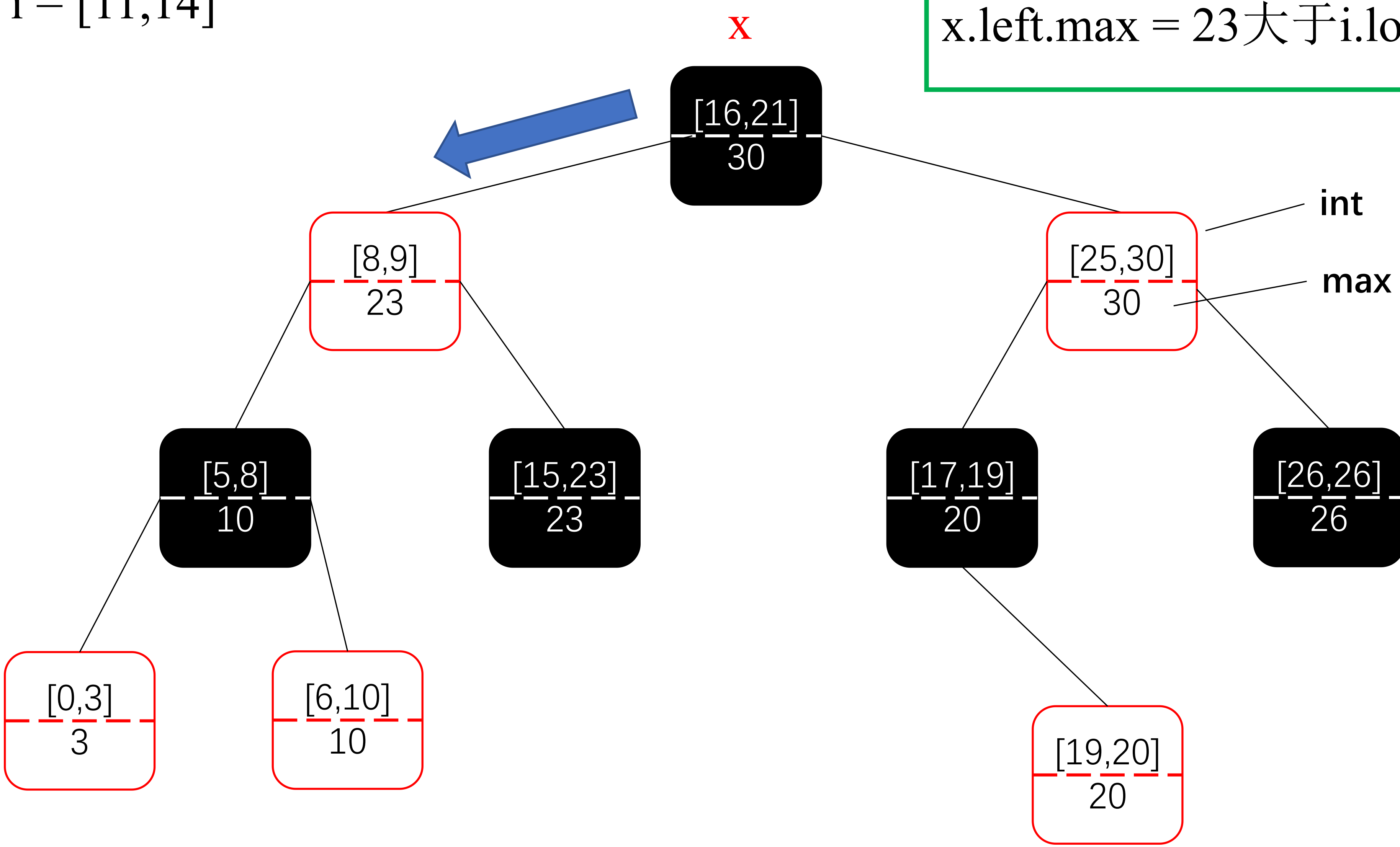
区间树

查找不成功

查找一个与区间i = [11,14]重叠的区间。再一次，开始时以x为根结点，它包含区间[16，21]，与i不重叠。且x.left.max = 23大于i.low = 11,则转向左边包含区间[8，9]，仍不与i重叠，且 x.left.max = 10小于i.low = 11,因此我们转向右子树。现在，由于结点x所包含的区间[15，23]仍不与i重叠，且它的左孩子为T.nil，故向右转，循环结束，返回T.nil。

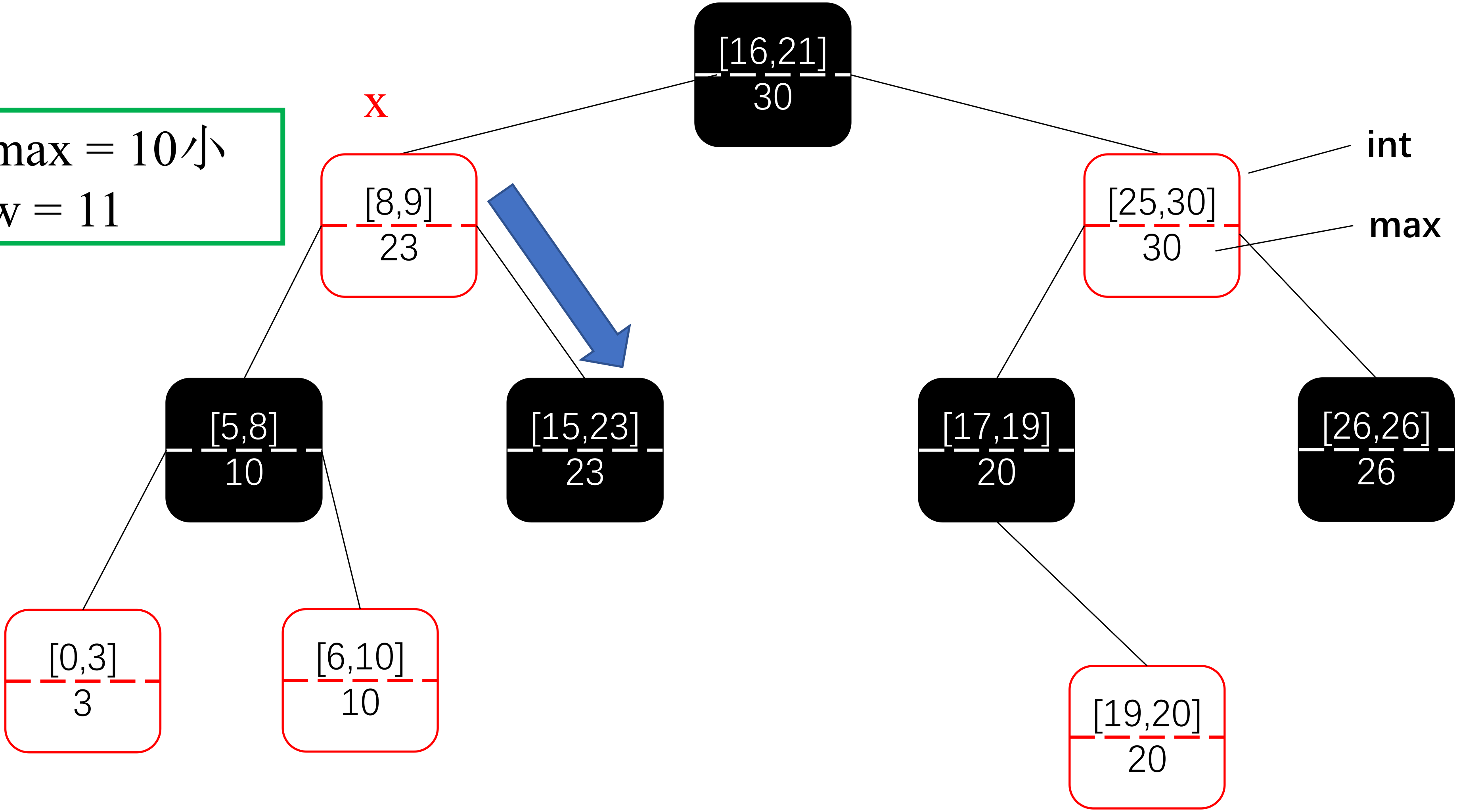
$i = [11,14]$

$x.left.max = 23$大于$i.low = 11$

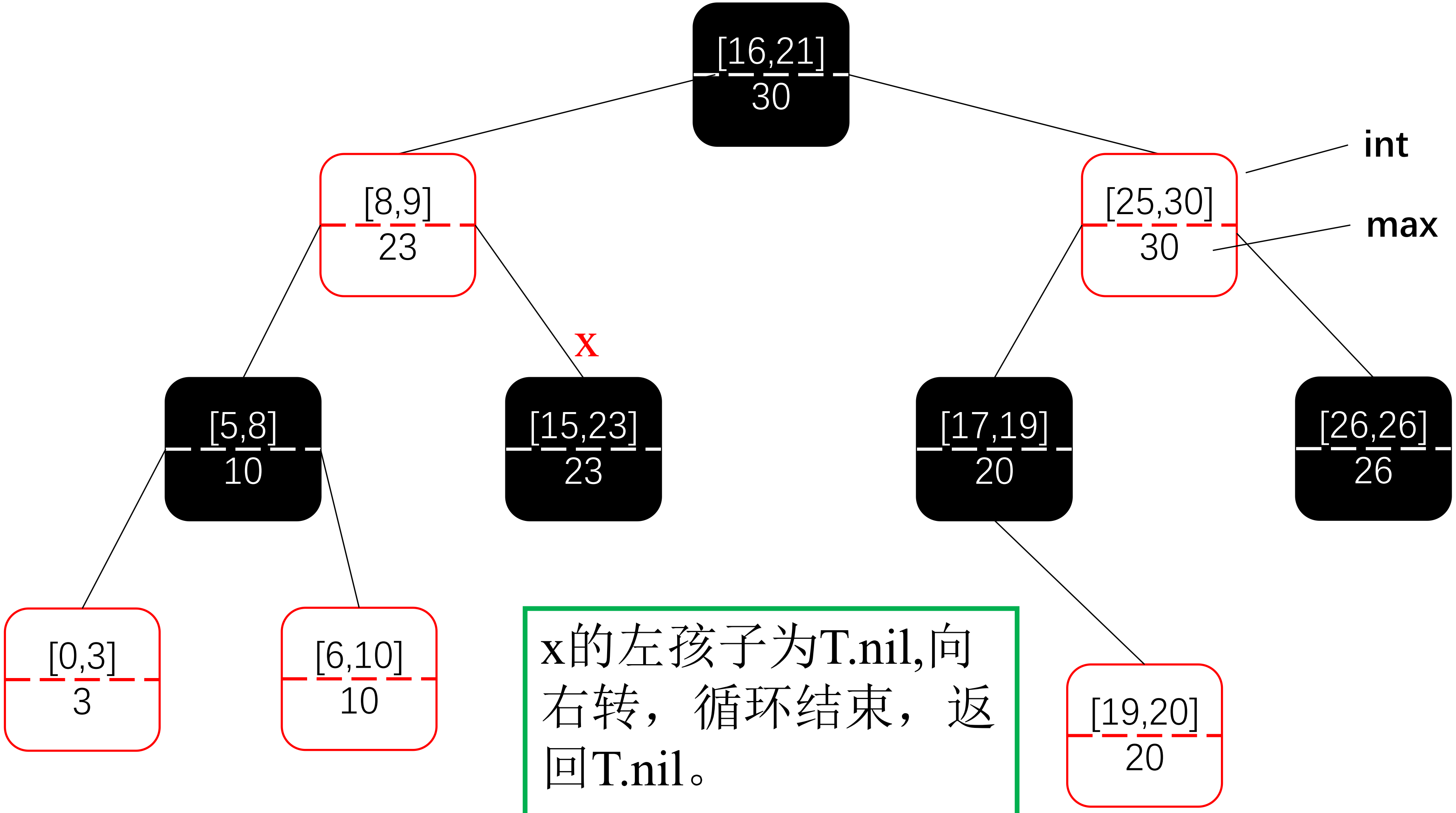


i = [11,14]

x.left.max = 10小于i.low = 11



i = [11,14]



x的左孩子为T.nil,向右转，循环结束，返回T.nil。

区间树正确性证明

定理2 INTERVAL-SEARCH(T,i)的任意一次执行，或者返回一个其区间与i重叠的结点，或者返回T.nil, 此时树T中没有任何结点的区间与i重叠。

证明： 当 x = T.nil或x.int 重叠时，第2-5行的while循环终止。后一种情况，过程返回x，显然是正确的。因此，主要考虑前一种情况，也就是当 x = T.nil时 while循环终止的情况。

对第2-5行的while 循环使用如下的性质：按照第2-5行剪枝掉左右子树是正确的

区间树正确性证明

初始化：在第一次迭代之前，第1行置x为T的根，性质成立。

保持：在while循环的每次迭代中，第4行或第5行被执行。下面将证明性质在这两种情况下都能成立。

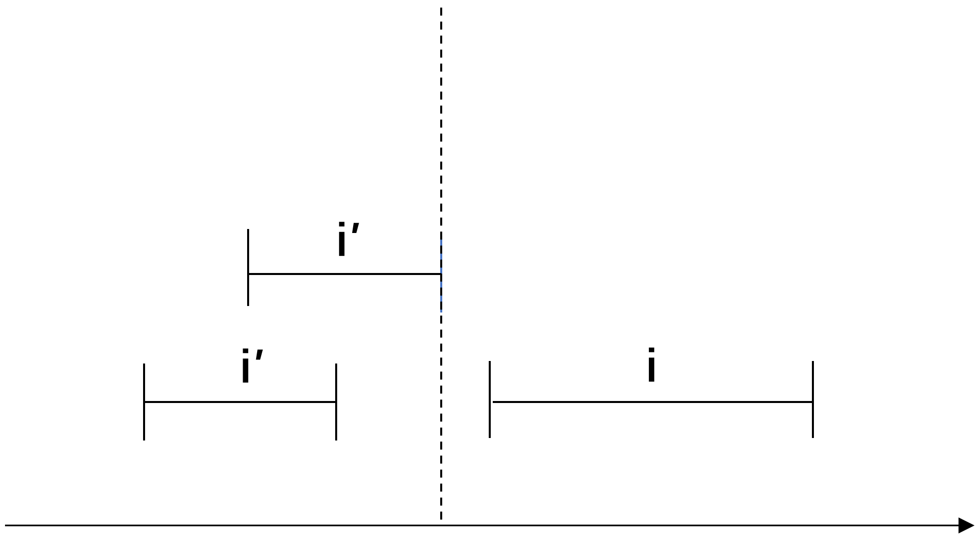
区间树正确性证明

情况(a)： 如果执行第5行，则由于第3行的分支条件有 x.left =T.nil或x.left.max<i.low 。如果x.left =T.nil，则以x.left 为根的子树显然不包含于i重叠的区间，所以置x为x.right以保持这个不变式。

因此，假设x.left ≠T.nil且x.left.max < i.low ，则如(a)所示，对x左子树的任一区间i′，都有

$$\mathrm{i}'.high \leq x.left.max < i.low$$

区间树正确性证明



(a)

根据区间三分律，i和i不重叠，因此，x的左子树不包含与i重叠的任何区间，置x为x.right，使循环不等式保持成立。

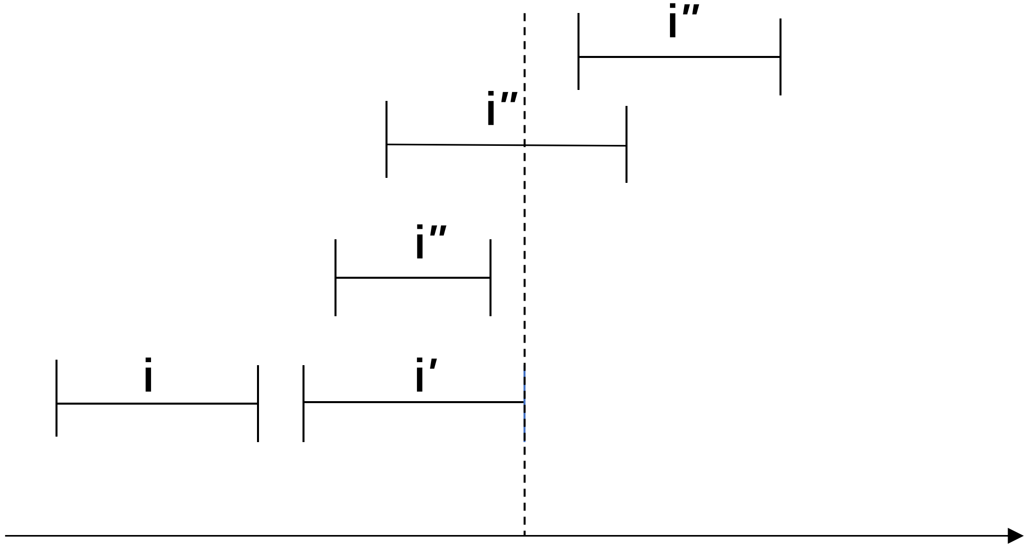
区间树正确性证明

情况(b)： 如果是第4行被执行，我们将证明性质的另一面情况，也就是说，如果在以x.left为根的子树中没有与i重叠的区间，则树的其他部分也不会包含与i重叠的区间。因为第4行被执行，是由于第3行的分支情况导致的，所以有些x.left.max ≥i.low。根据max属性的定义，在x的左子树中必定存在某区间i′，满足

$$\mathrm{i}'.high = \mathrm{x.left.max} \ \geq \mathrm{i.low}$$

区间树正确性证明

下图显示了这种情况。因为i和i′不重叠，又因为i′.high<i.low不成立，所以根据区间三分律有。区间树是以区间的低端点为关键字的，所以搜索树性质隐含了对x右子树中的任意区间i″，有 $i.high < \mathrm{i}'.low \leqq i''.low$



(b)

区间树正确性证明

根据区间三分律，i和i″不重叠。我们得出这样的结论，即不管x的左子树中是否存在与i重叠的区间，置x为x.left保持性质成立。

区间树正确性证明

终止：如果循环在x=T.nil 时终止，则表明在以x为根的子树中，没有与i重叠的区间。循环不变式的对等情况说明了 T中不包含与i重叠的区间，故返回x=T.nil是正确的。

因此，过程 INTERVAL-SEARCH 是正确的。

湖南大学
HUNAN UNIVERSITY

HNU

谢谢观赏

实 事 求 是　敢 为 人 先