Augmented Interval Tree

Chapter 14.3

Background

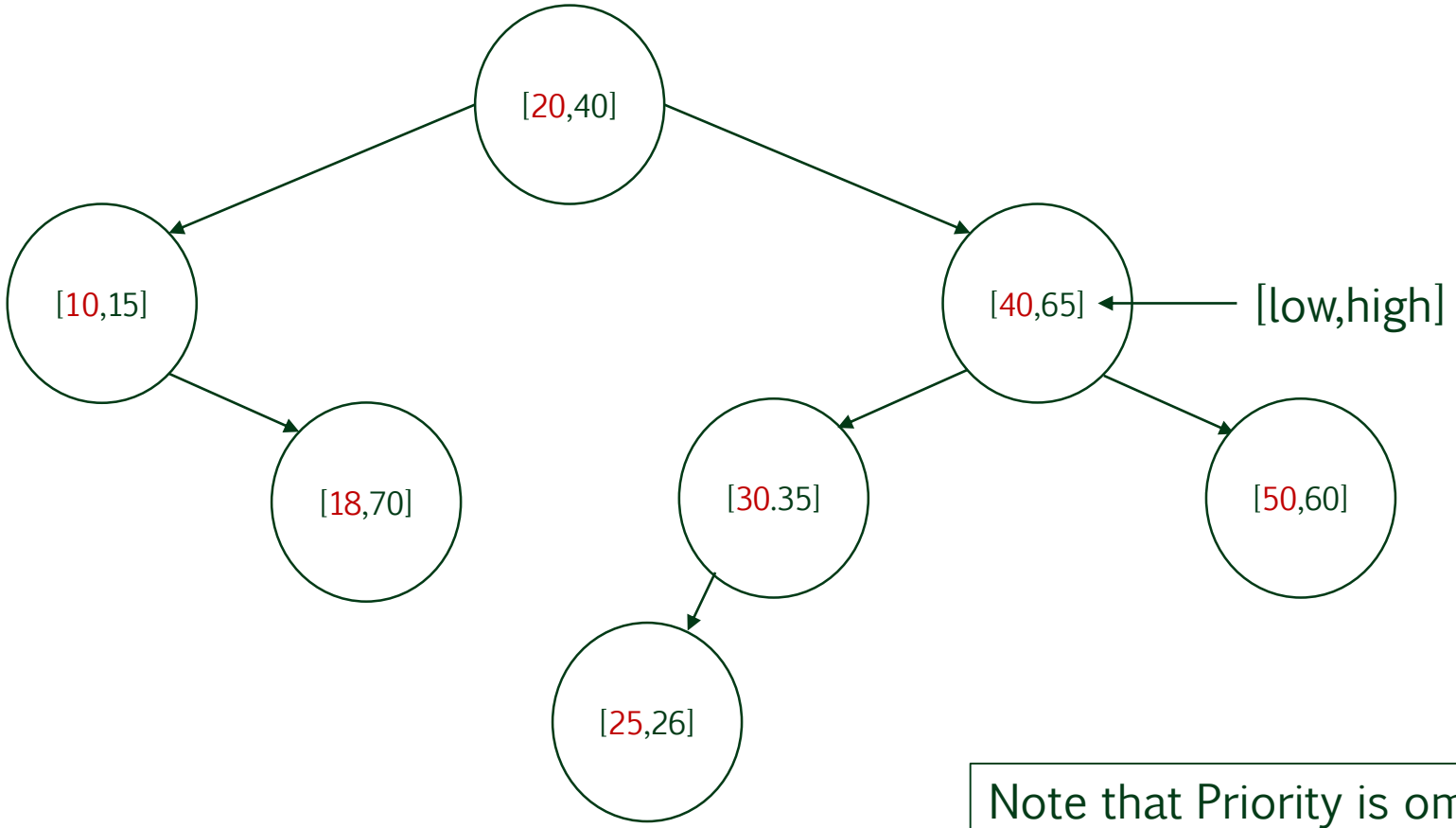
- A closed interval is an ordered pair of real numbers [low,high] where:
  - low ≤ high
  - [low,high] represents the set { t | low ≤ t ≤ high }
- Often, an interval represents a period of time

Definition

- An interval tree is binary search tree of intervals [low,high] ordered on low

- In our case, the interval tree is implemented as a treap and therefore, the expected depth of the tree is O(log n)

Example of an interval tree



Note that Priority is omitted

Rationale for an augmented interval tree

- Suppose we wish to add one method to the interval tree (treap) that returns an interval that overlaps with a given period

```
Interval Overlap(Interval period);
```

- In our previous example:
  - [19,25] overlaps with [20,40], [18,70] and [25,26]
  - [9,75] overlaps with all intervals in the tree
  - [16,17] overlaps with no intervals in the tree

Using the original interval tree

- To find an interval that overlaps with a given period may require us to traverse the entire underlying treap.
  - For example, if the given period is [66, 68], it may be tempting to explore the right subtree of the root node since the low bound of the period (66) is greater than the low bound of the root (20). But the interval that overlaps with period [66,68] occurs in the rightmost node in the left subtree of the root, i.e. [18,70]!
- The expected time complexity is O(n) since half the intervals are visited on average

Using the augmented interval tree

- By adding one data member to the original implementation of the interval tree (treap) the expected time complexity for the Overlap method can be reduced to O(log n)

- But what data member is added, where is it added, and can it be easily maintained?

New data member: MaxHigh

- The Node class is augmented with the data member MaxHigh which stores the maximum High value of any interval in the subtree rooted at Node p

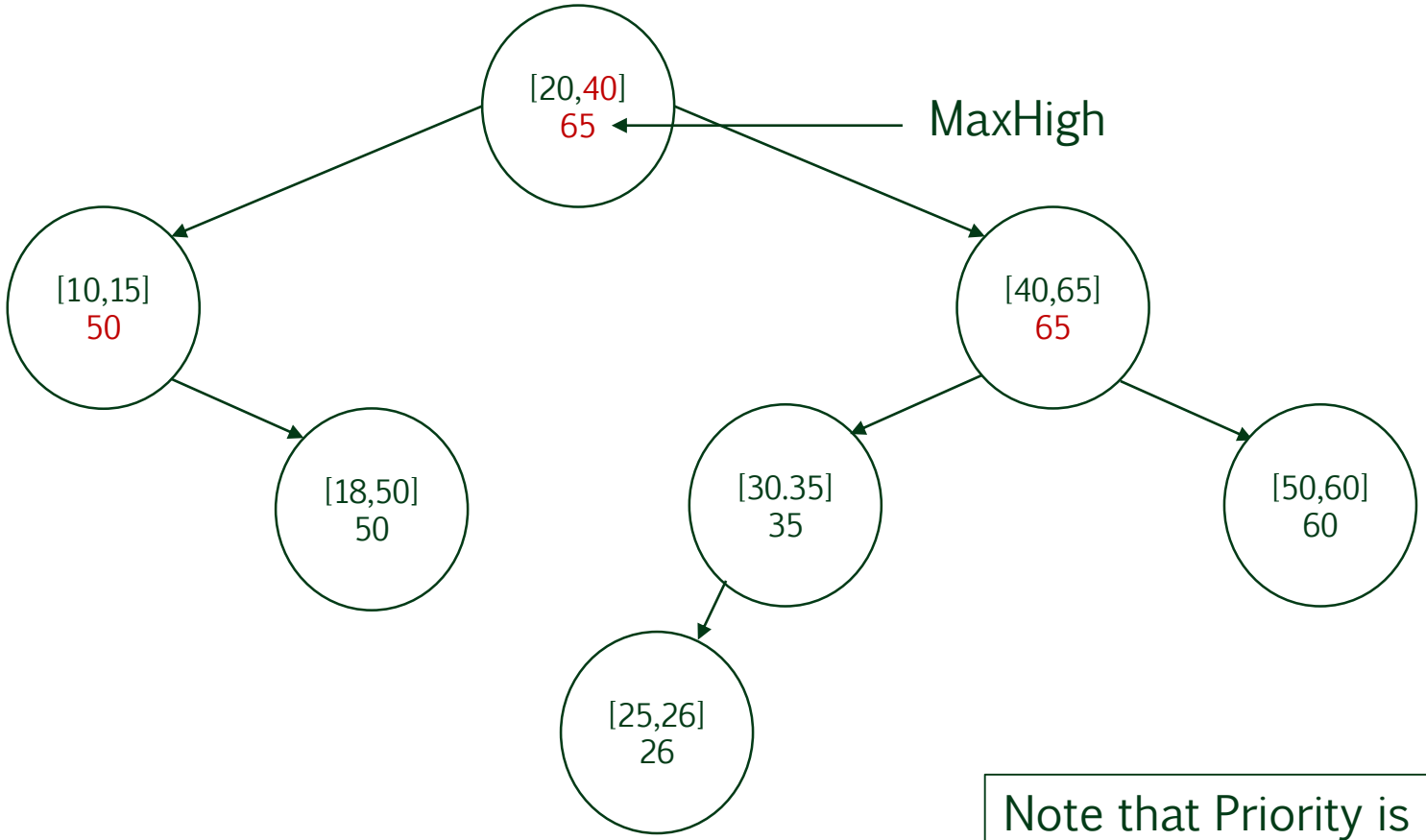
- Calculated as

```
p.MaxHigh = p.Period.High;
if (p.Left != null)
    if (p.Left.MaxHigh > p.MaxHigh)
        p.MaxHigh = p.Left.MaxHigh;
if (p.Right != null)
    if (p.Right.MaxHigh > p.MaxHigh)
        p.MaxHigh = p.Right.MaxHigh;;
```

Augmented data structure

```
public class Node
{
    private static Random R = new Random();
    public Interval Period { get; set; }
    public int Priority    { get; set; }    // Randomly generated
    public int MaxHigh     { get; set; }    // Augmented information (data)
    public Node Left       { get; set; }
    public Node Right      { get; set; }
    ...
}

class IntervalTree : ISearchable
{
    private Node Root;  // Reference to the root of the Treap
    ...
}
```

Example of an augmented interval tree



Note that Priority is omitted

Can MaxHigh be easily maintained?

- Yes
- When an interval is added to or removed from an interval tree, MaxHigh is updated in O(1) time for each node along the path from the point of insertion or removal to the root of the treap. Note that both an insertion and removal take place at a leaf node.
- An additional update is also needed whenever a rotation is performed on the way up the treap for the Add method.

Exercises (cf augmented treap)

- Explain why MaxHigh is updated for the left (right) child of the new root after a LeftRotate (RightRotate) is performed in the Add method.

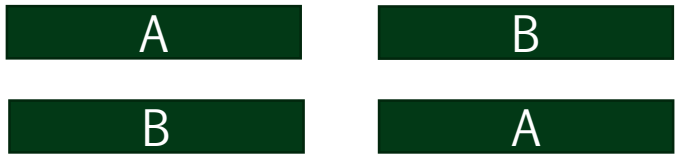
- Explain why MaxHigh need not be updated after a LeftRotate or RightRotate is performed in the Remove method.

Checking for Overlap

Given two intervals A and B, one of three properties holds:

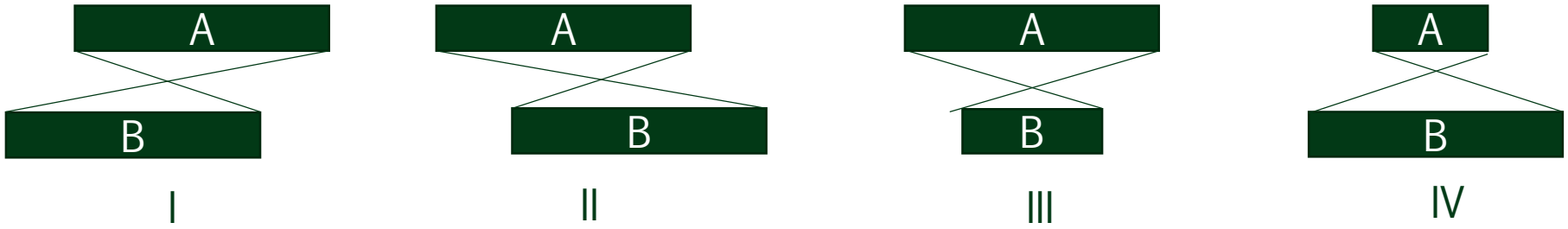
1. A.High < B.Low
2. B.High < A.Low

} no overlap



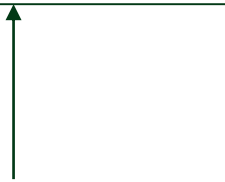
3. ! (A.High < B.Low || B.High < A.Low) ≡ (A.High >= B.Low && B.High >= A.Low)

overlap



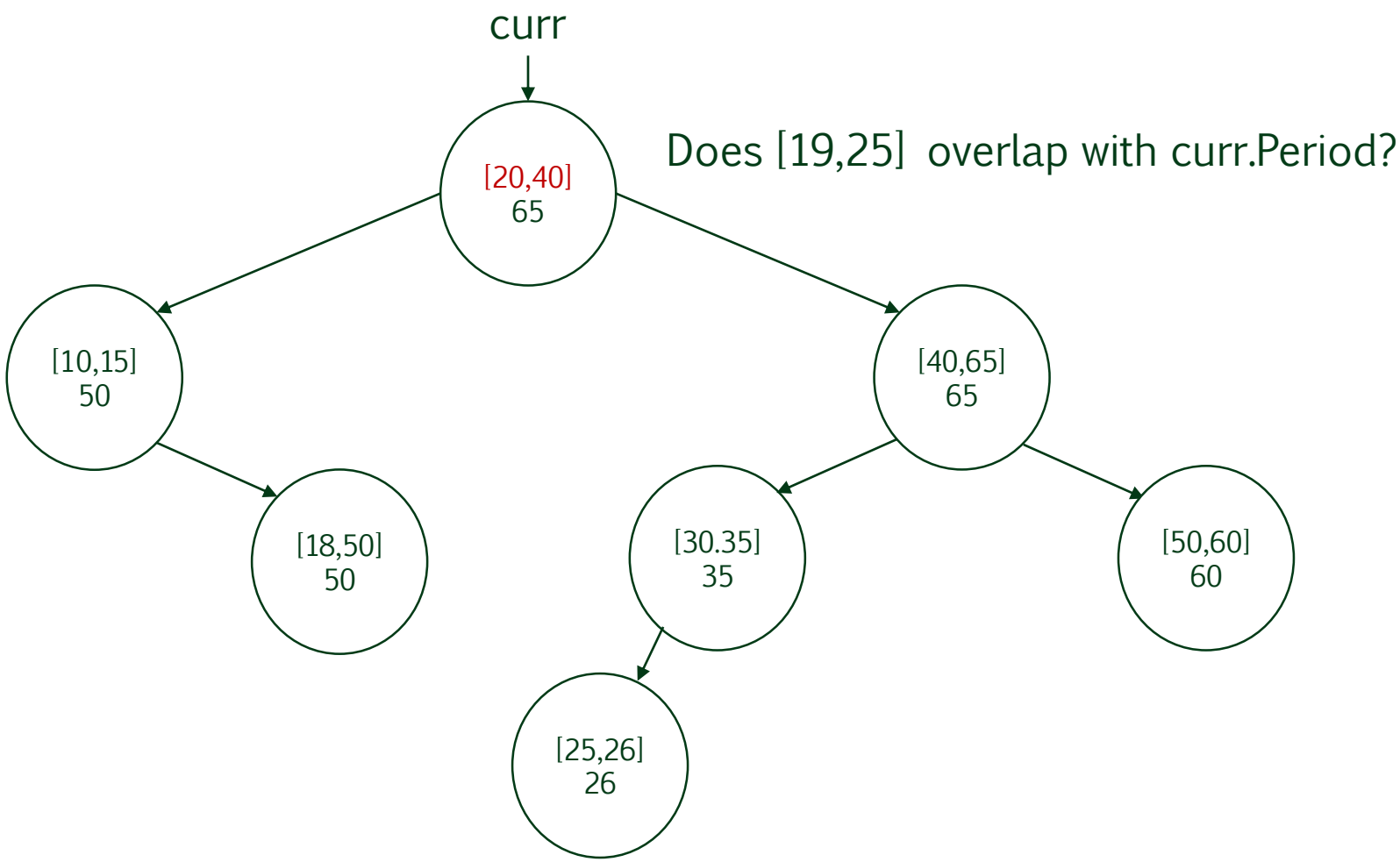
How can MaxHigh be used to support Overlap?

```
Node curr = Root;

while ((curr != null) &&    // No overlap
       (curr.Period.Low > period.High || period.Low > curr.Period.High))
{
    if ((curr.Left != null) && period.Low <= curr.Left.MaxHigh)
        curr = curr.Left;
    else
        curr = curr.Right;
}

if (curr != null)
    return curr.Period;
else
    return new Interval(0, 0);
```

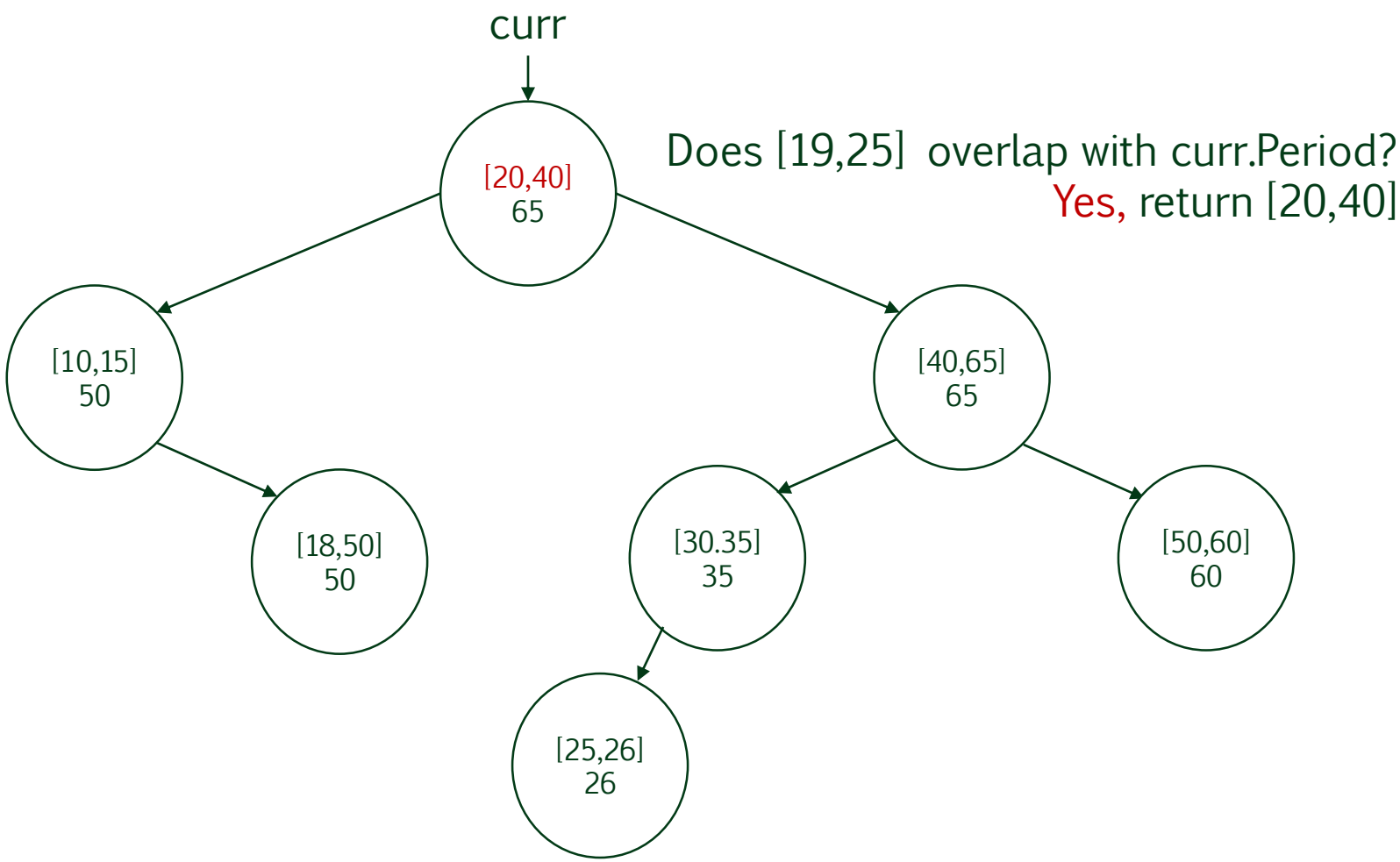


Go left or right?

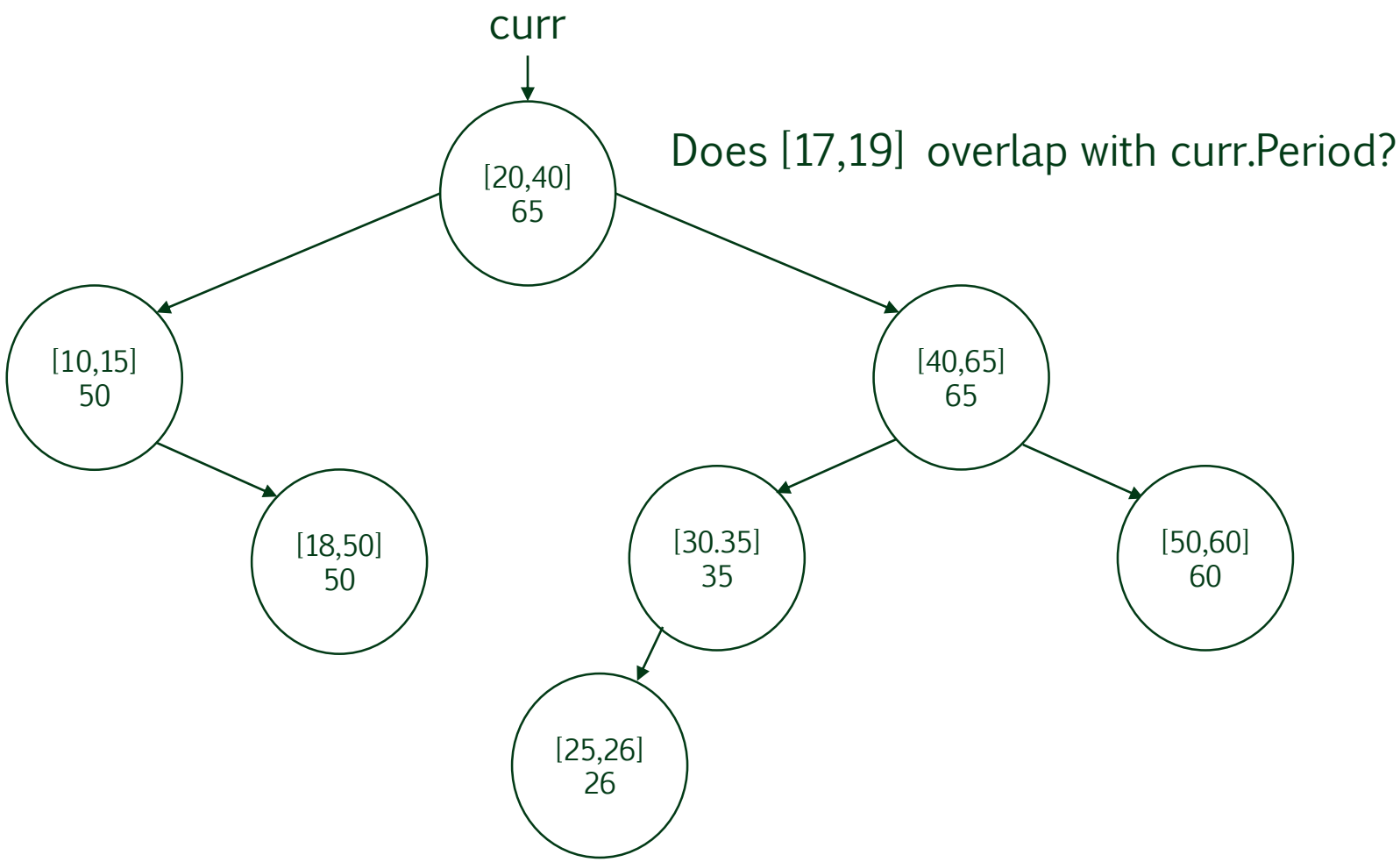
Return interval that overlaps with [19,25]



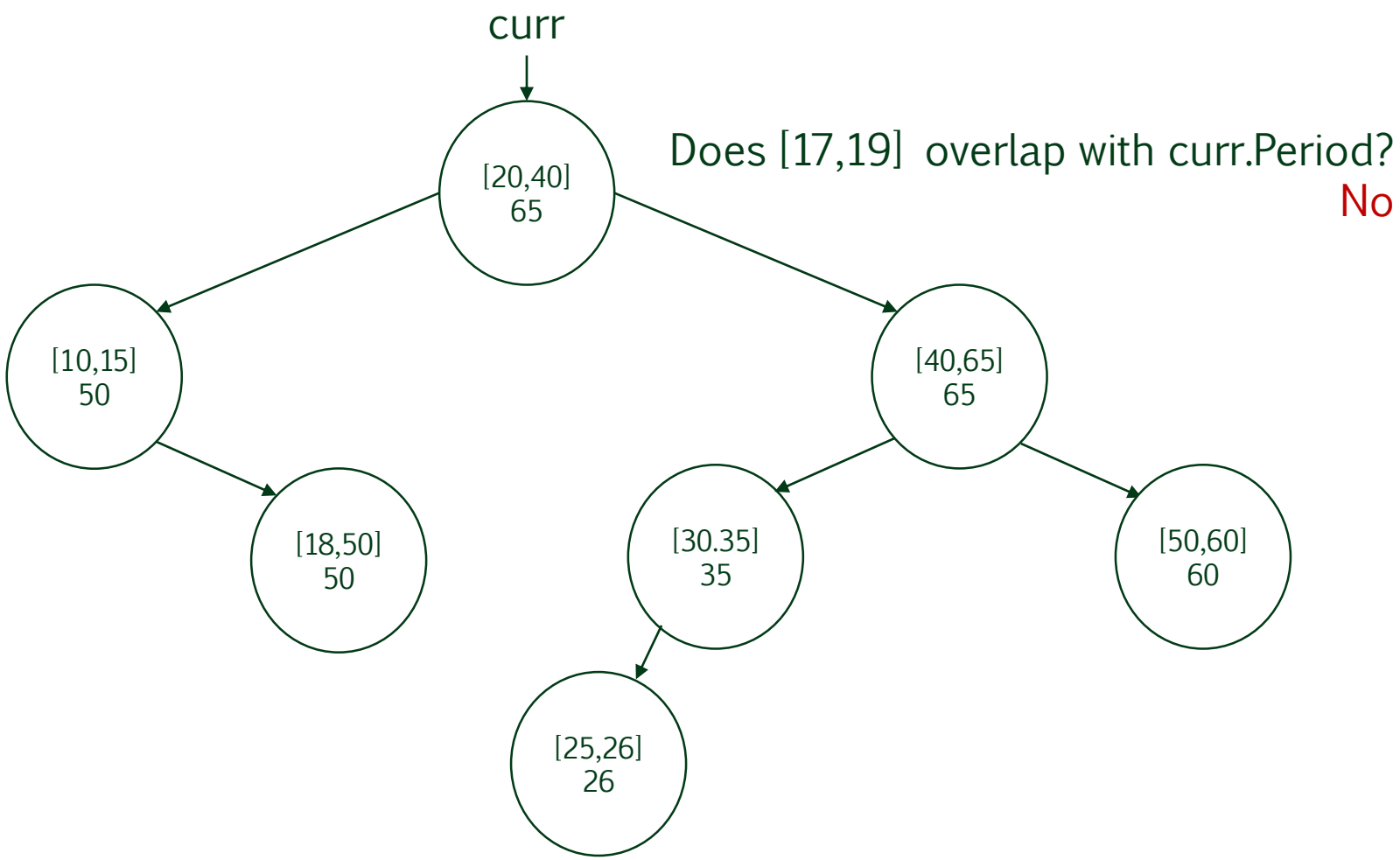
Return interval that overlaps with [19,25]



Return interval that overlaps with [17,19]



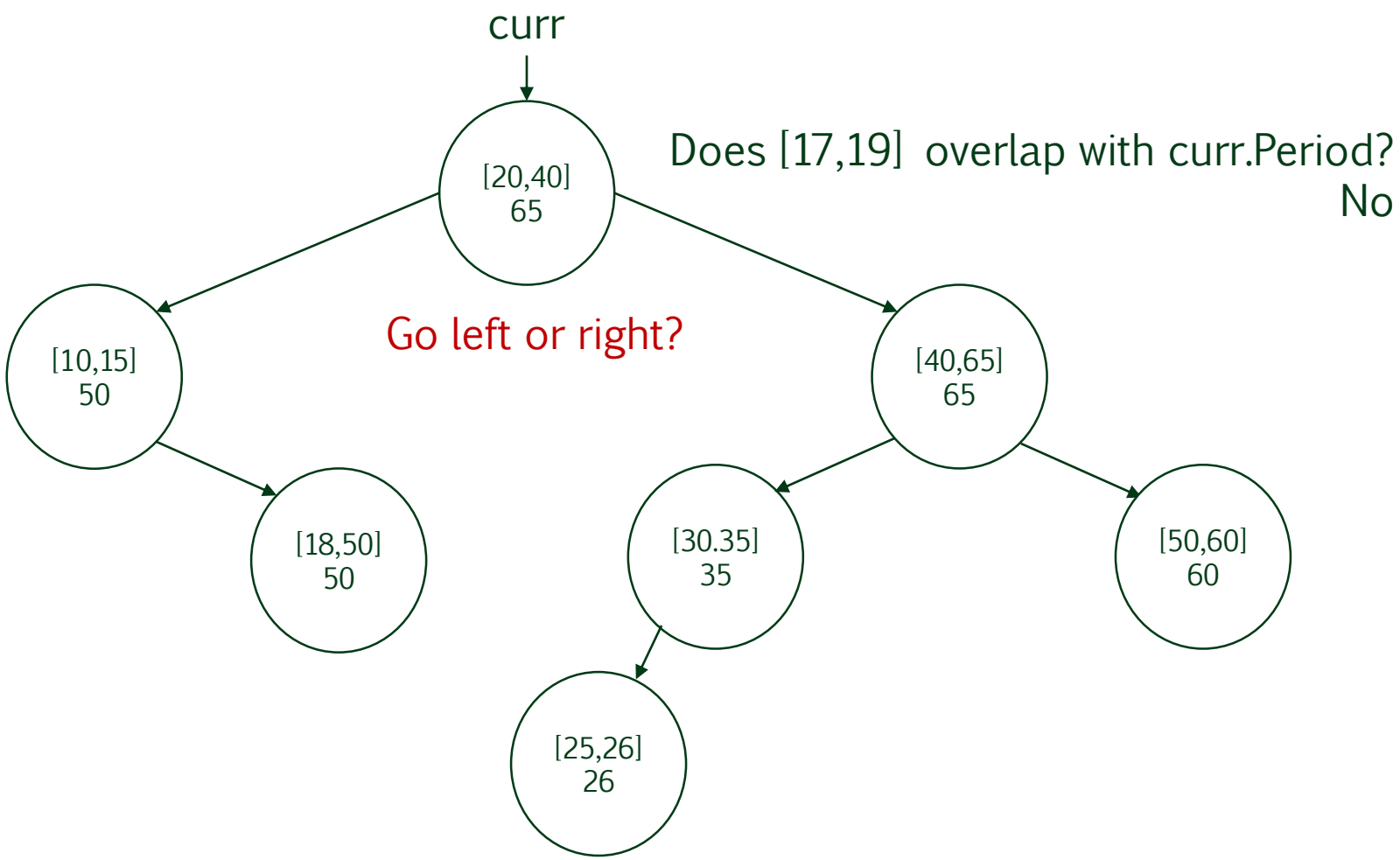
Return interval that overlaps with [17,19]



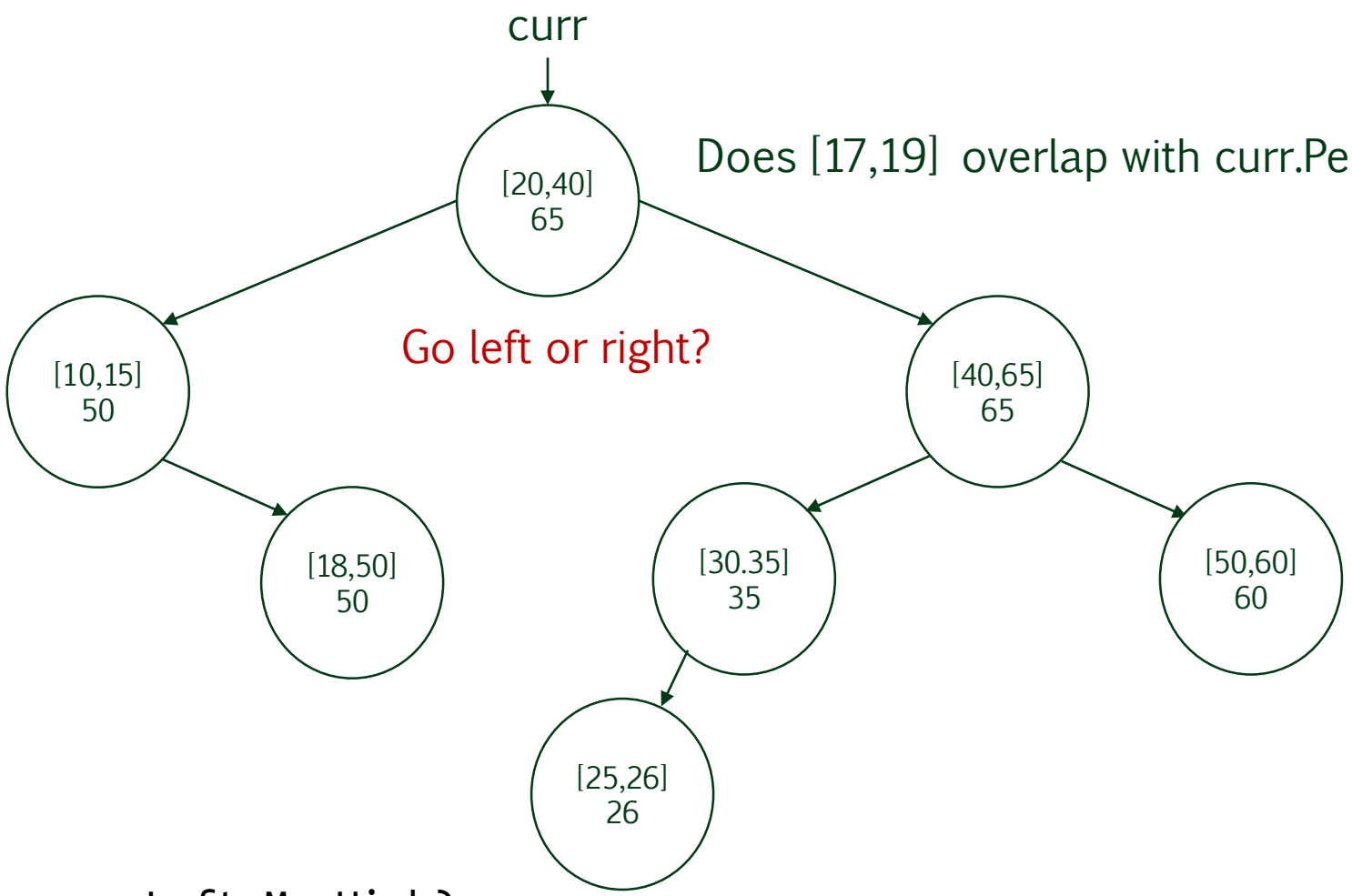
Does [17,19] overlap with curr.Period?

No

Return interval that overlaps with [17,19]

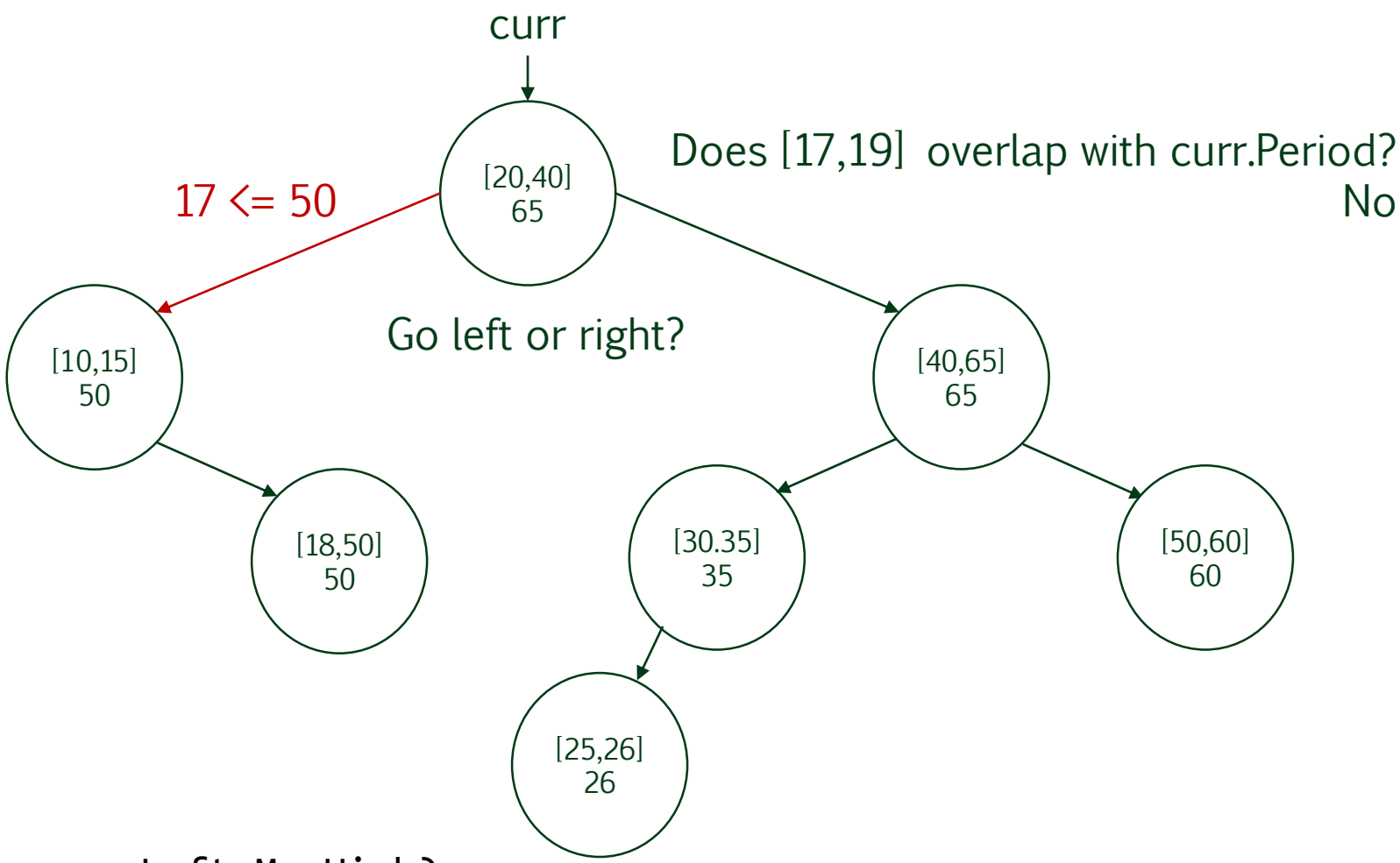


Return interval that overlaps with [17,19]




```
period.Low <= curr.Left.MaxHigh?
```

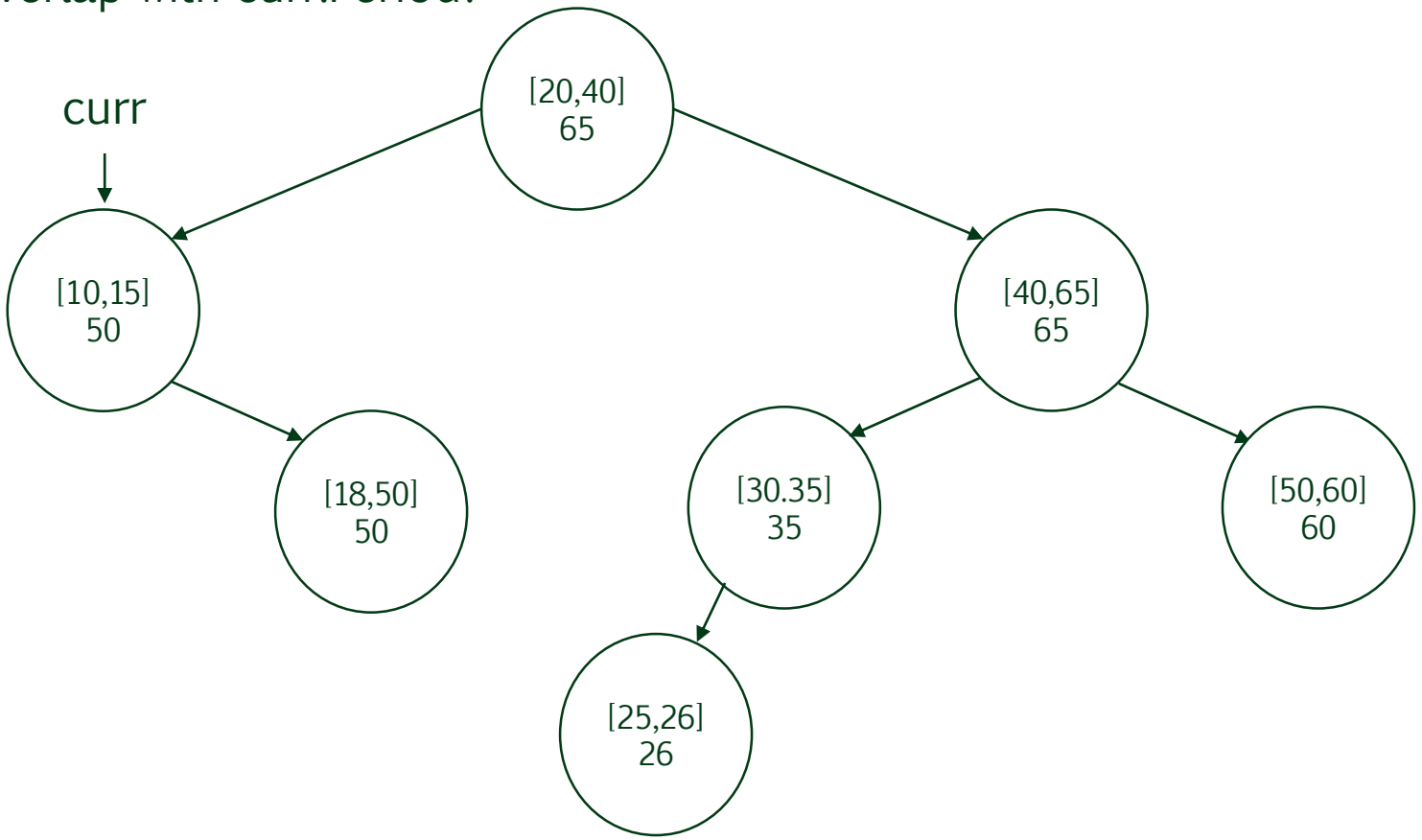
Return interval that overlaps with [17,19]




```
period.Low <= curr.Left.MaxHigh?
```

Return interval that overlaps with [17,19]

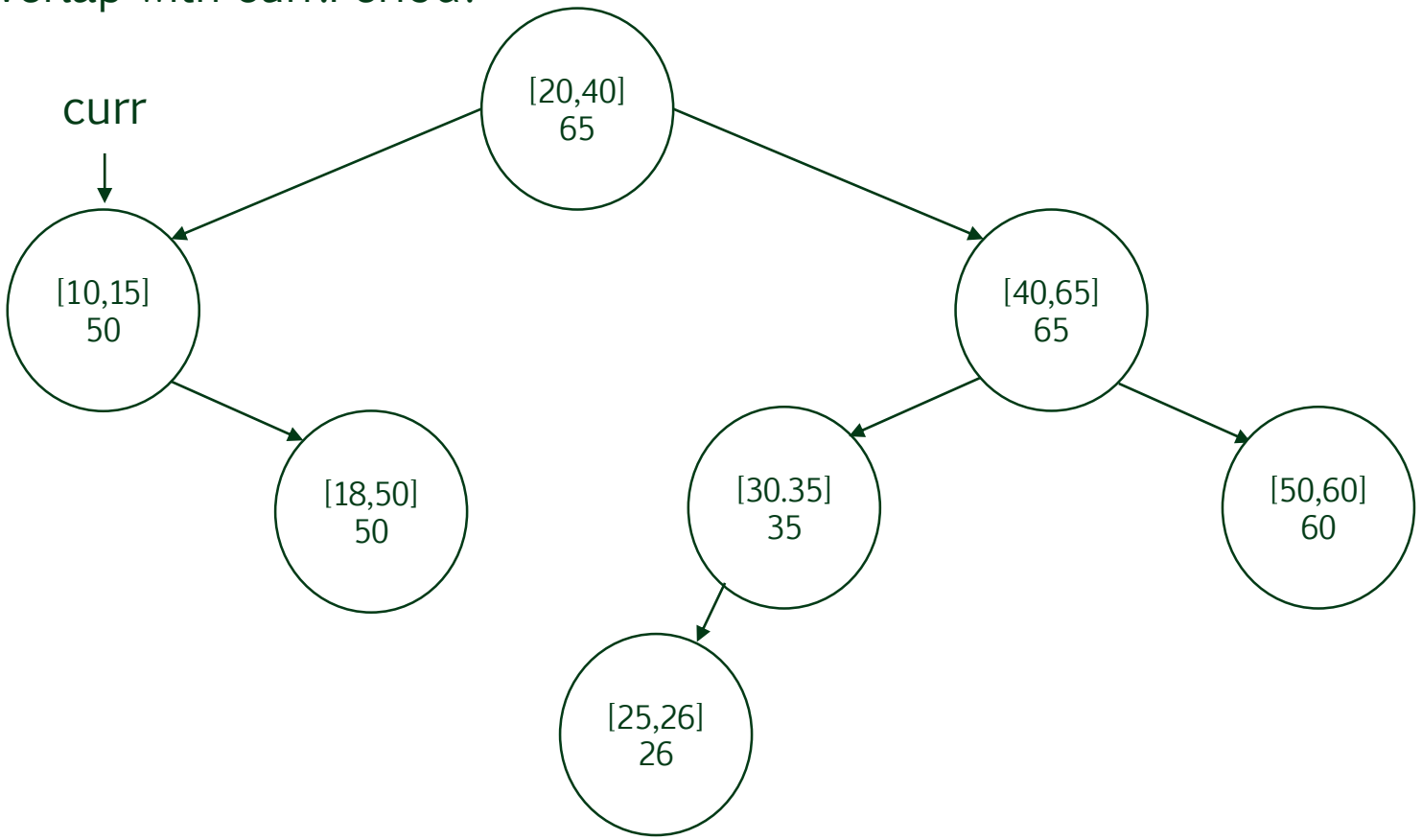
Does [17,19] overlap with curr.Period?



Return interval that overlaps with [17,19]

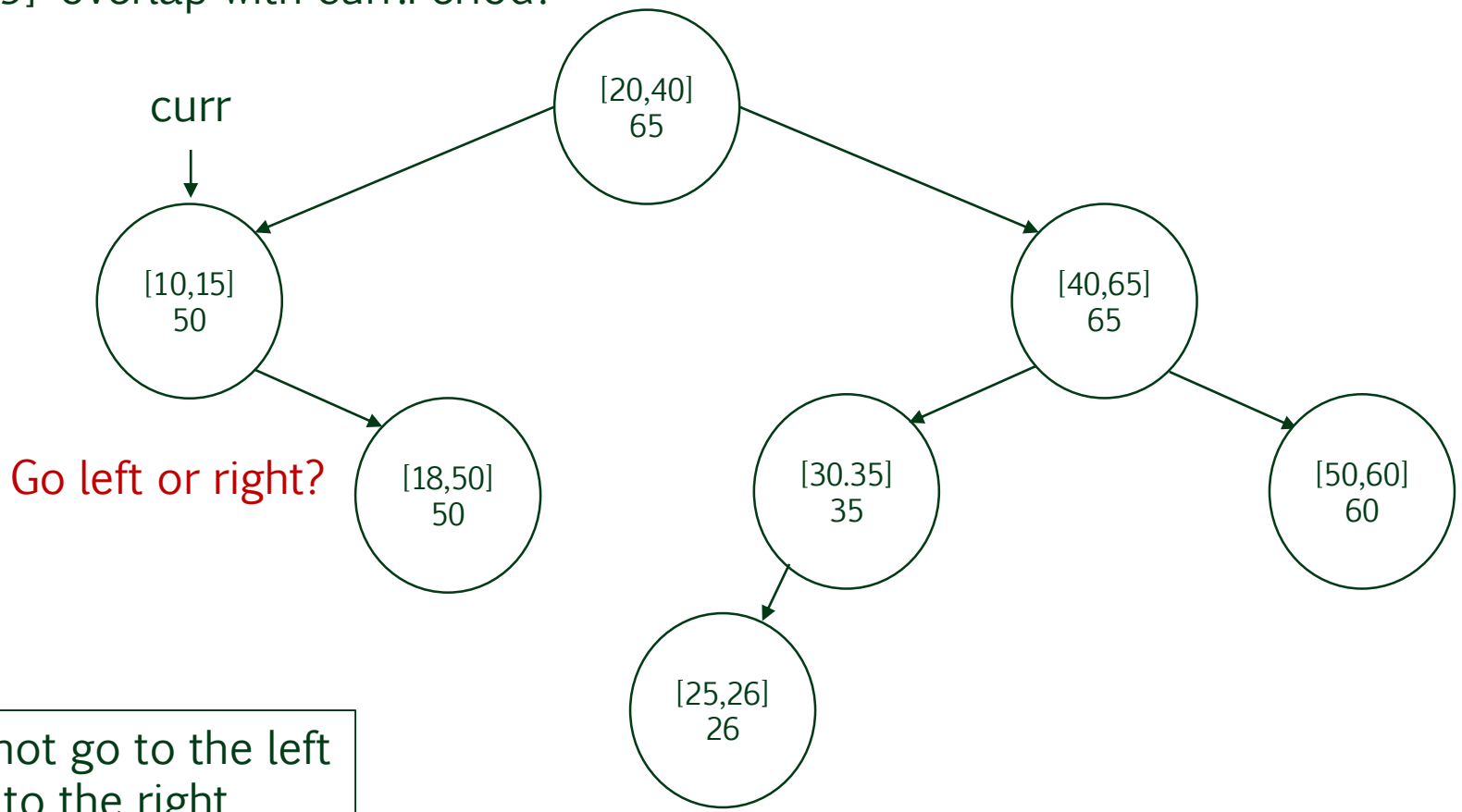
Does [17,19] overlap with curr.Period?

No



Return interval that overlaps with [17,19]

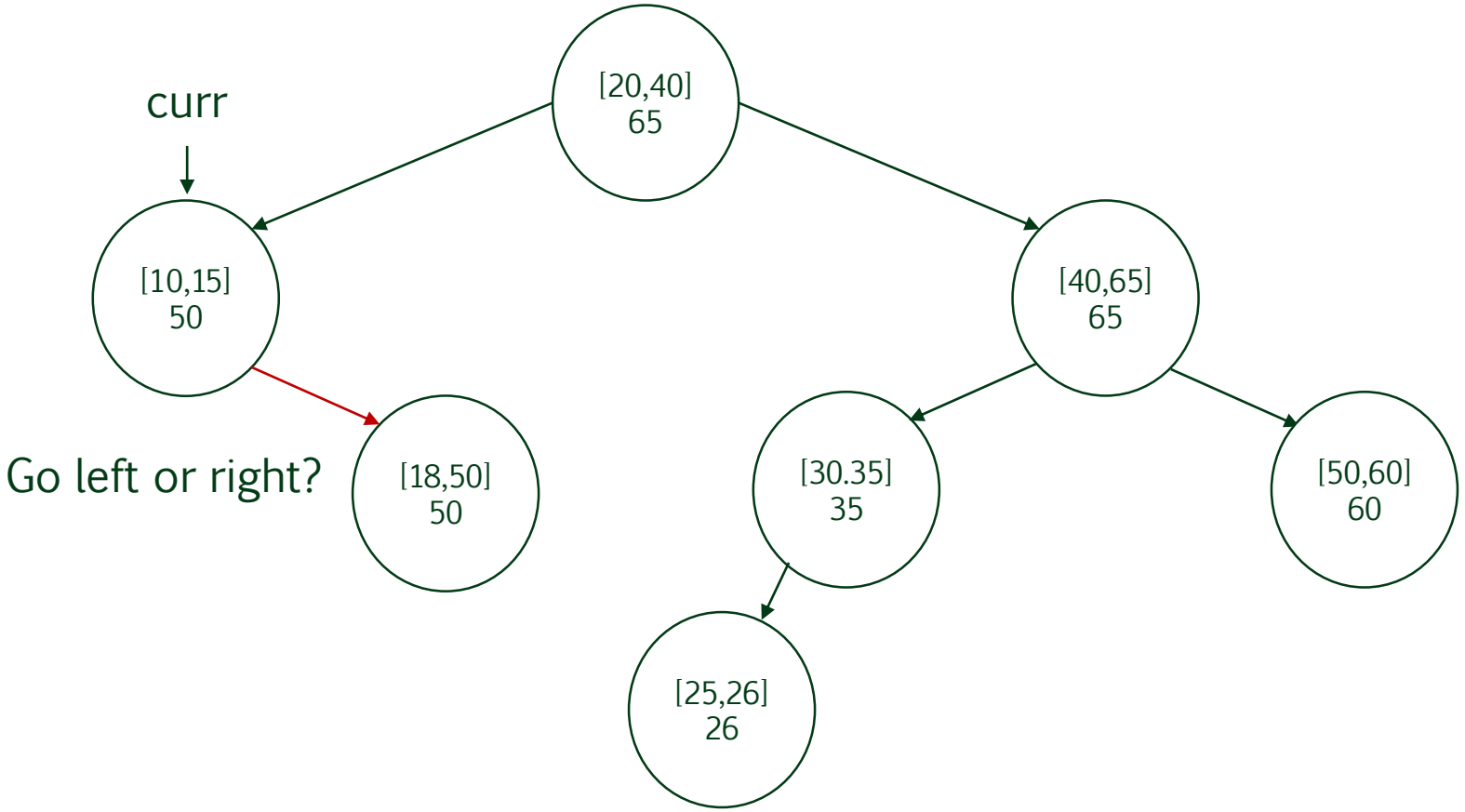
Does [17,19] overlap with curr.Period?
No



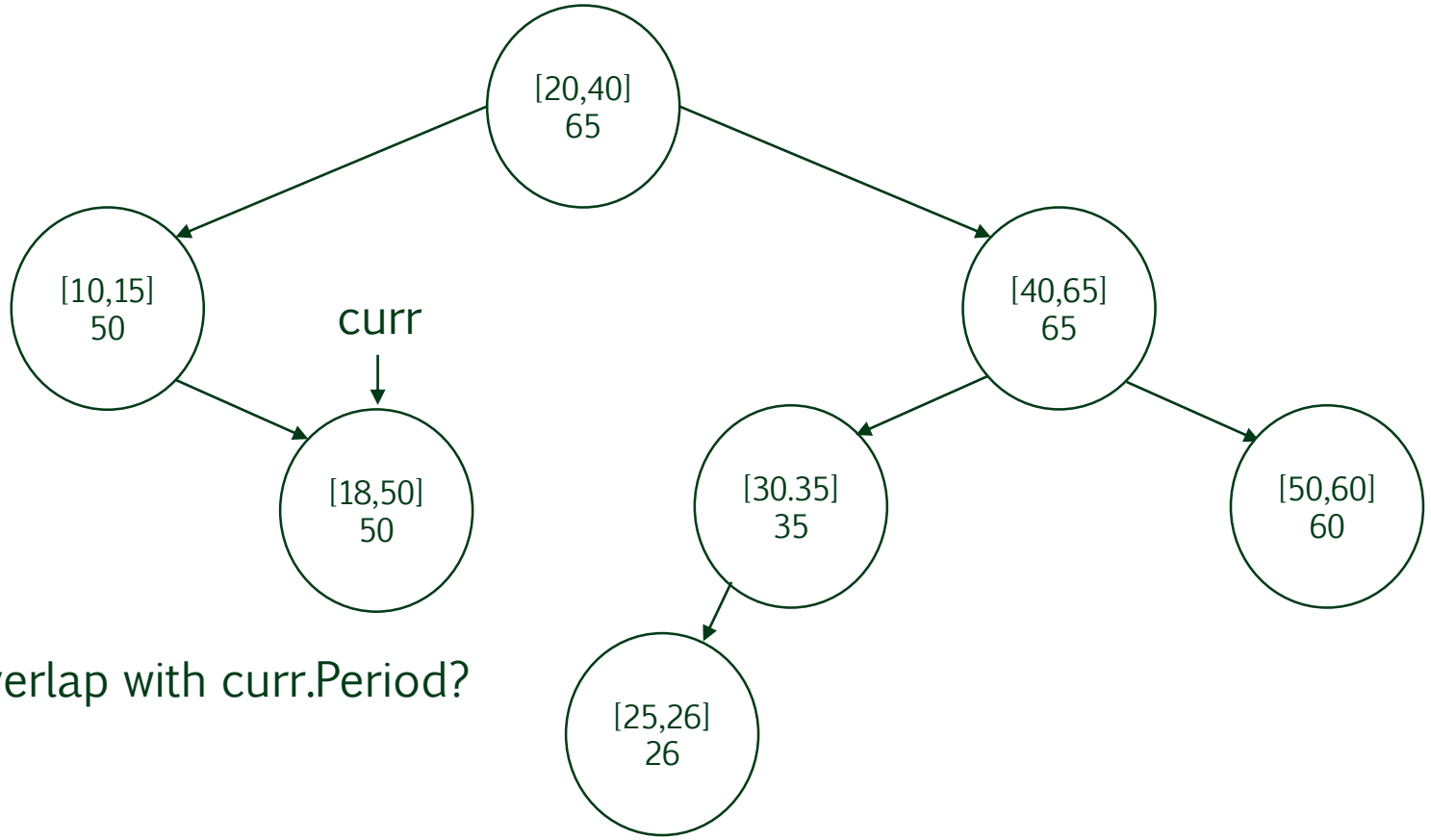
Go left or right?

If you cannot go to the left always go to the right

Return interval that overlaps with [17,19]

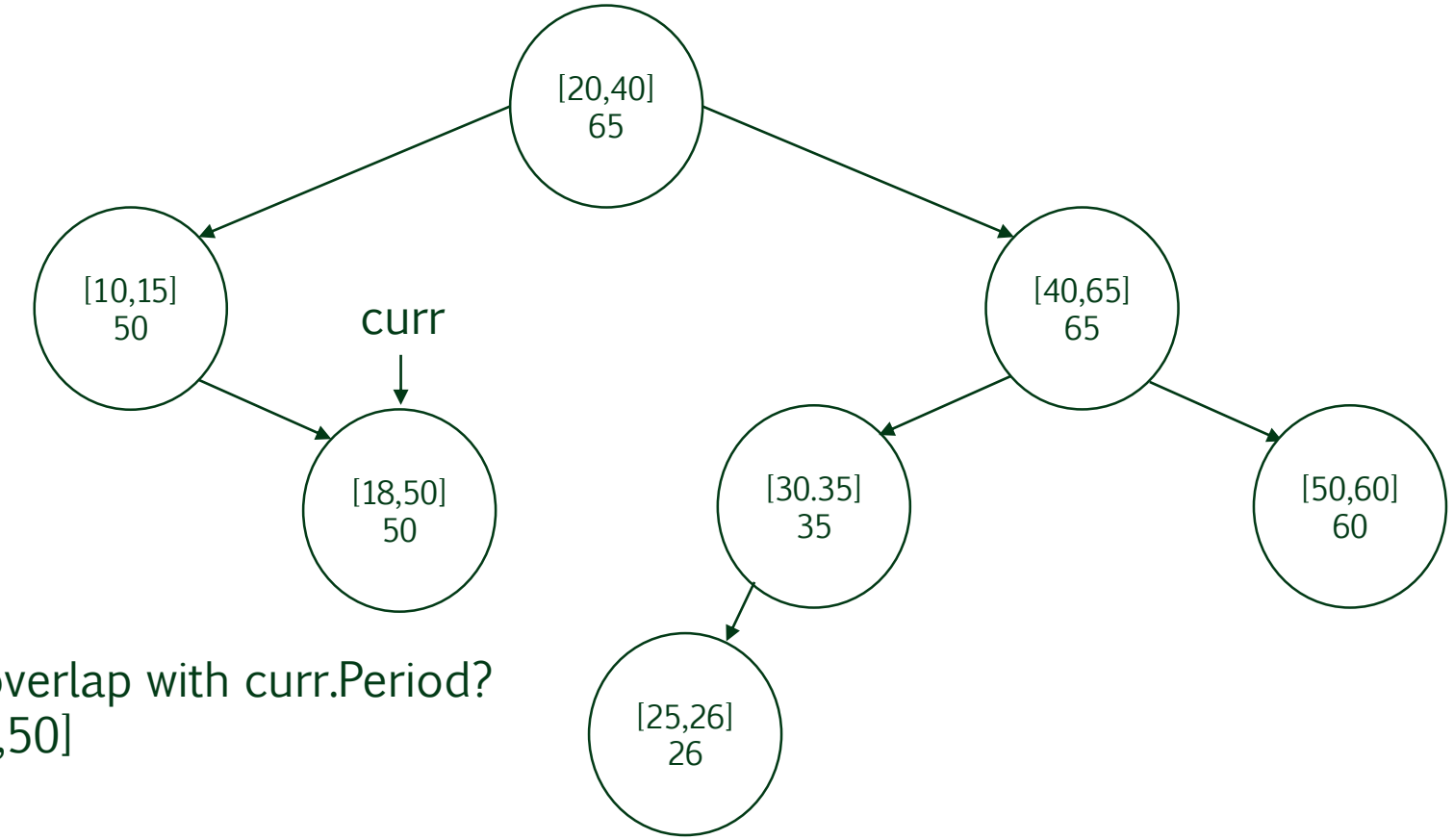


Return interval that overlaps with [17,19]



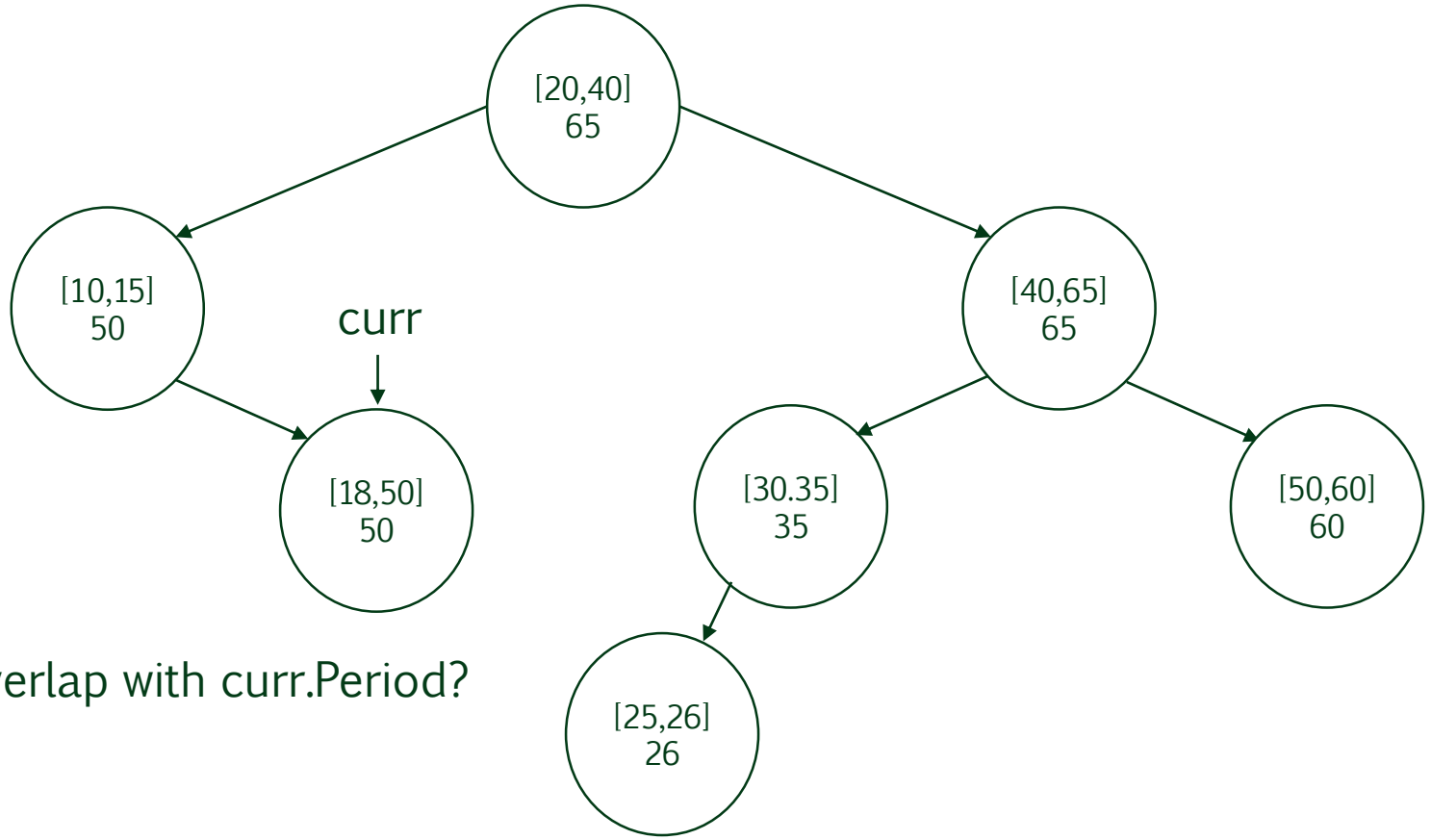
Does [17,19] overlap with curr.Period?

Return interval that overlaps with [17,19]



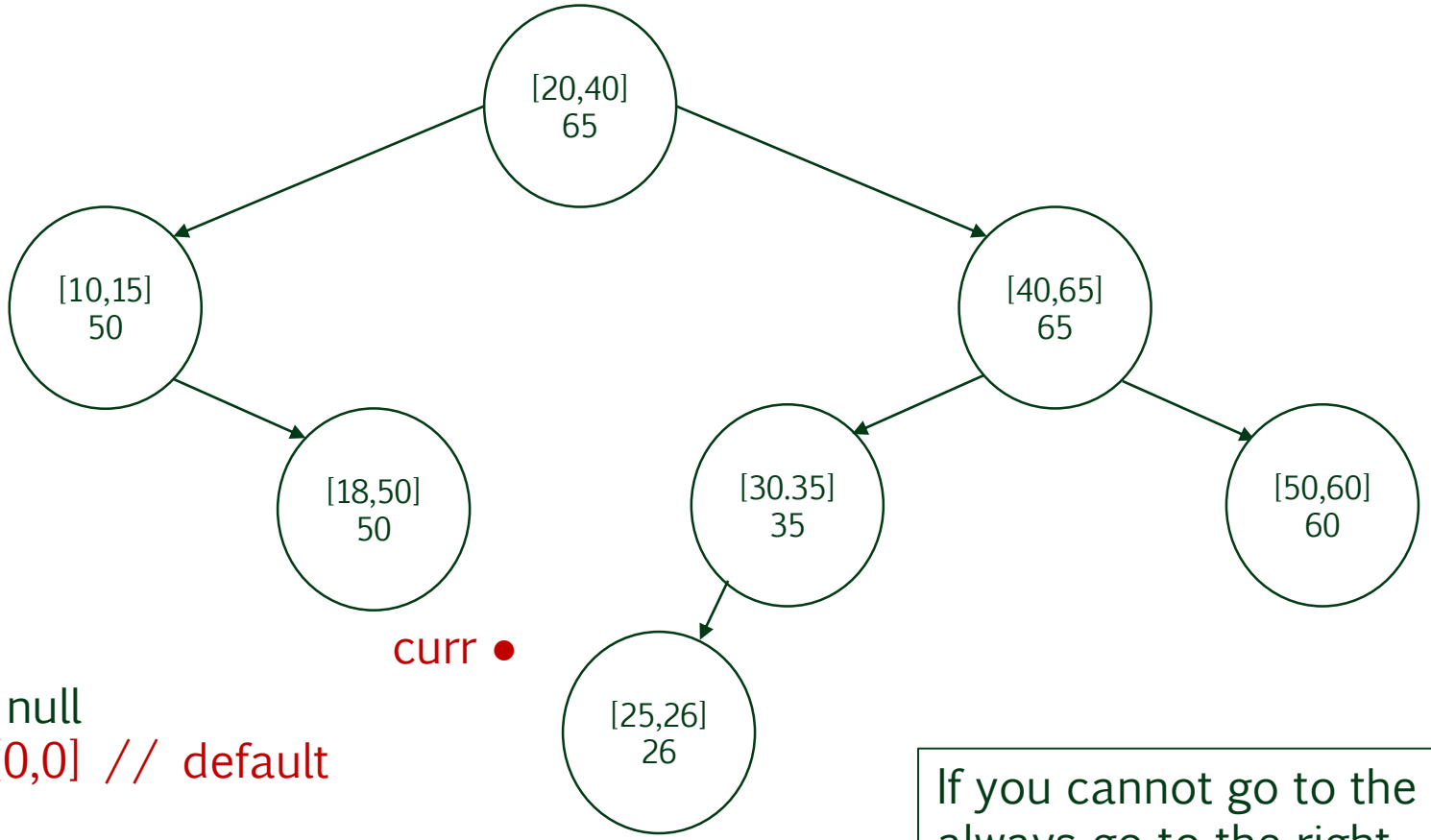
Does [17,19] overlap with curr.Period?
Yes, return [18,50]

Return interval that overlaps with [16,17]



Does [16,17] overlap with curr.Period?
No

Return interval that overlaps with [16,17]

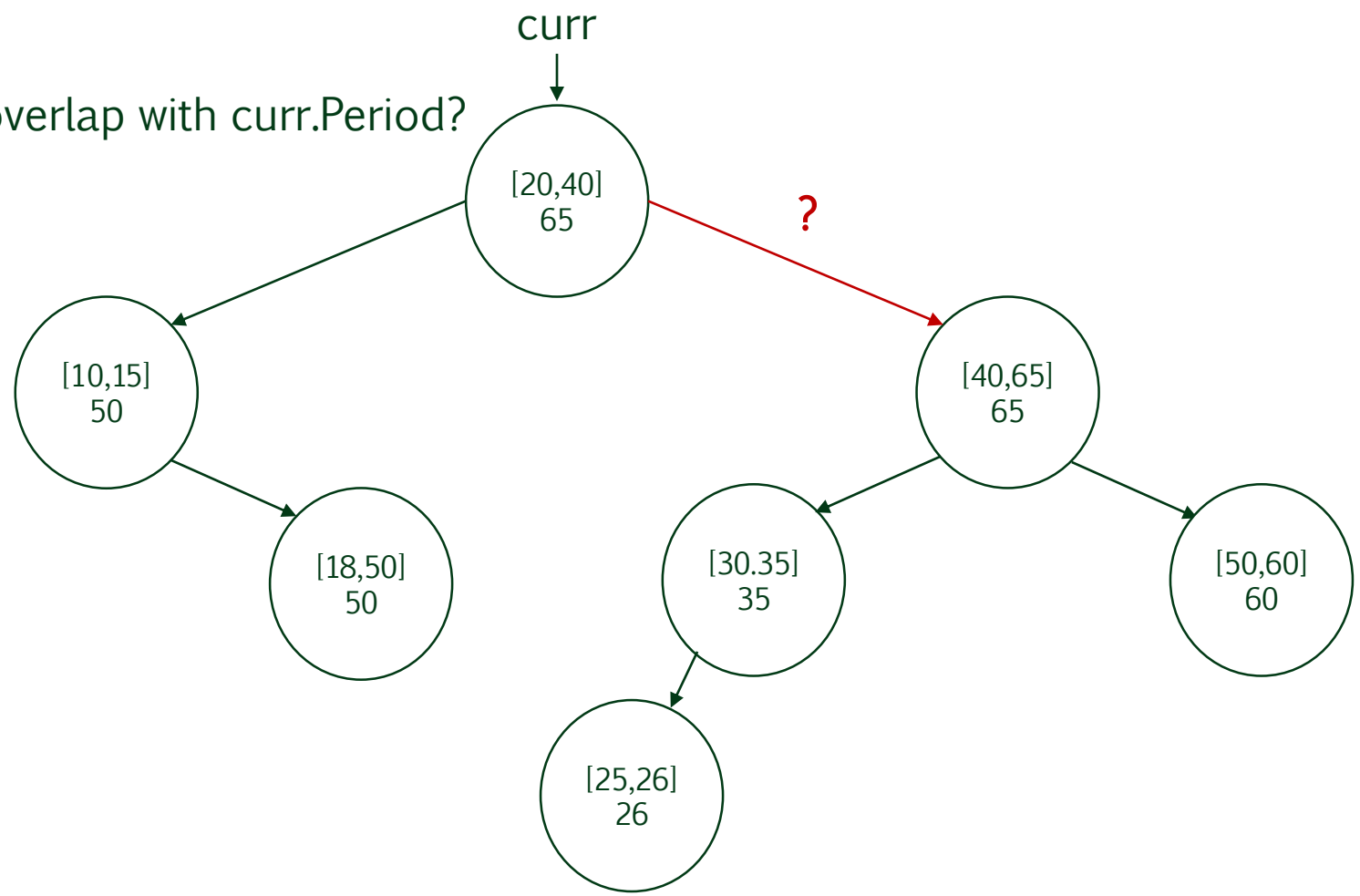


curr == null
Return [0,0] // default

If you cannot go to the left always go to the right

But could [16,17] overlap in the right subtree?

Does [16,17] overlap with curr.Period?
No



Theorem

If no overlap is found in the left subtree when

```
period.Low <= curr.Left.MaxHigh
```

then no overlap will be found in the entire tree *.

* Assuming no overlap is found at curr.

Proof

In the subtree rooted at curr.Left there is an interval A whose High value is MaxHigh.

Hence, period.Low <= A.High.

Also, period.High < A.Low since there is no overlap.

Because the interval tree is ordered on Low,

period.High < X.Low

for all intervals X in the subtree rooted at curr.Right.

Hence, there can be no overlaps in the right subtree either.

Theorem

If no overlap is found in the right subtree when

```
period.Low > curr.Left.MaxHigh
```

then no overlap will be found in the entire tree *.

* Assuming no overlap is found at curr.

Proof

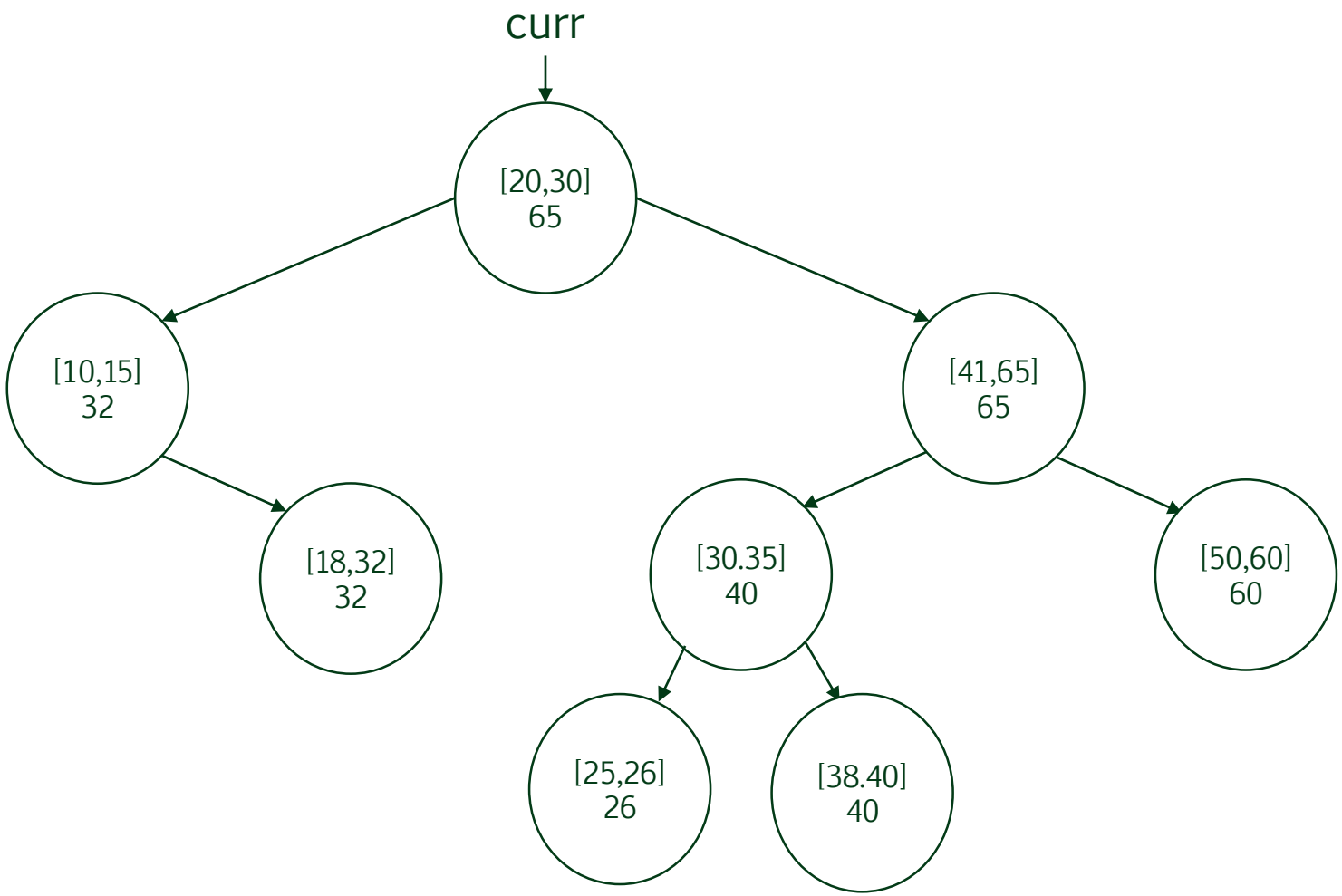
If period.Low is greater than curr.Left.MaxHigh then the interval period begins after all intervals in the left subtree rooted at curr.Left.

Hence, there can be no overlaps in the left subtree either.

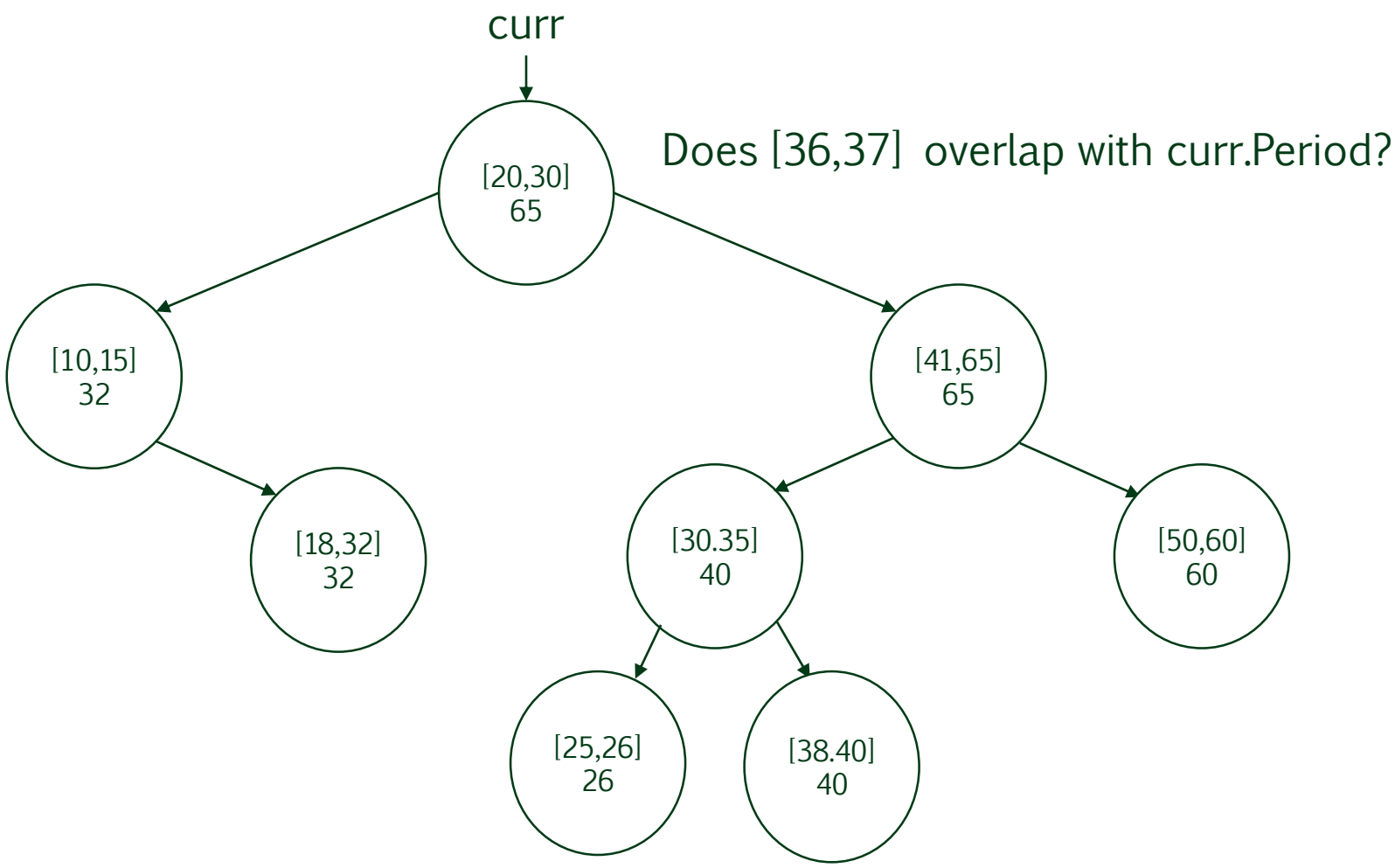
Implication of the theorem

Only one path from the root need be followed to determine if an interval overlaps with a given period.

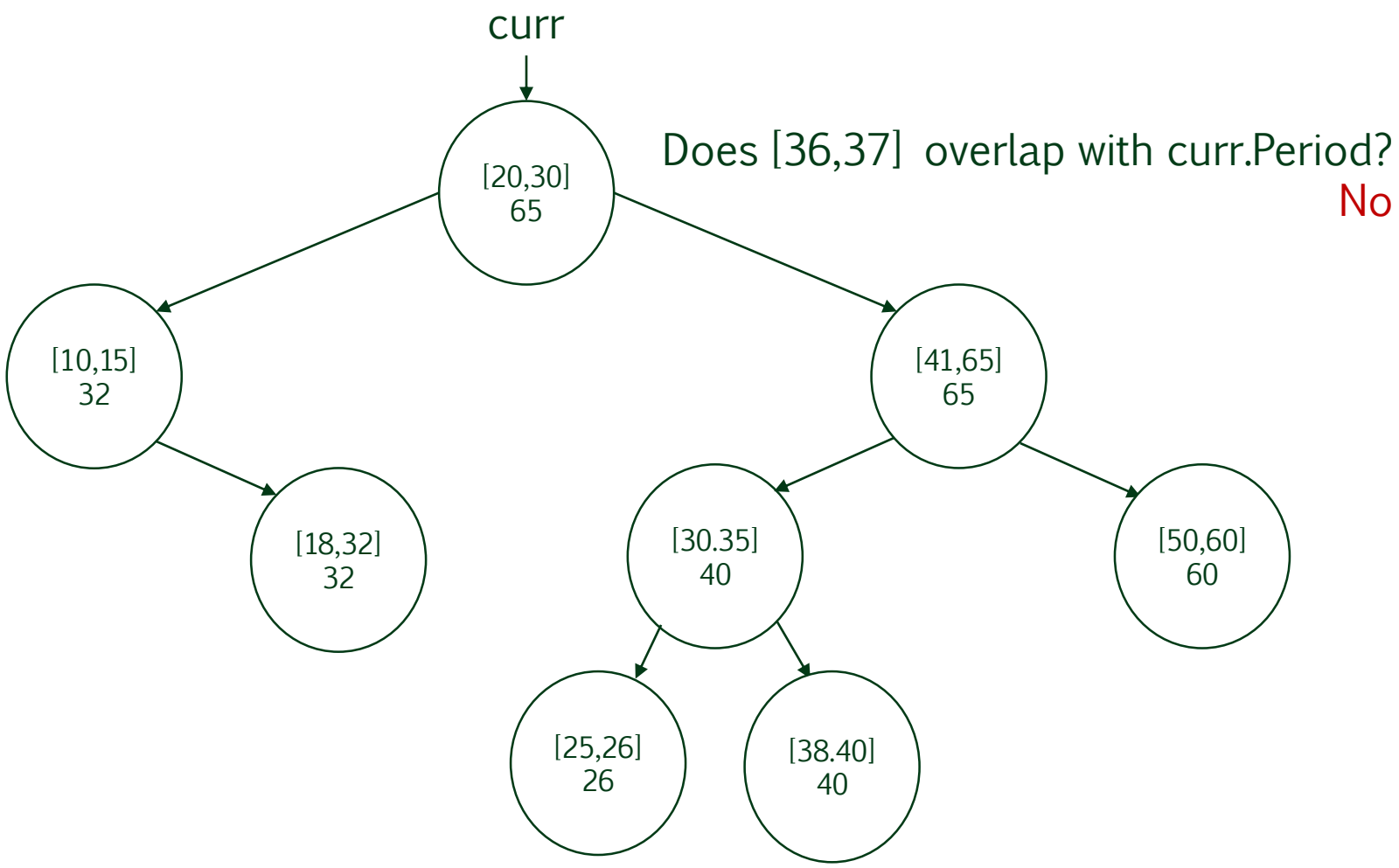
Consider the following interval tree



Return interval that overlaps with [36,37]



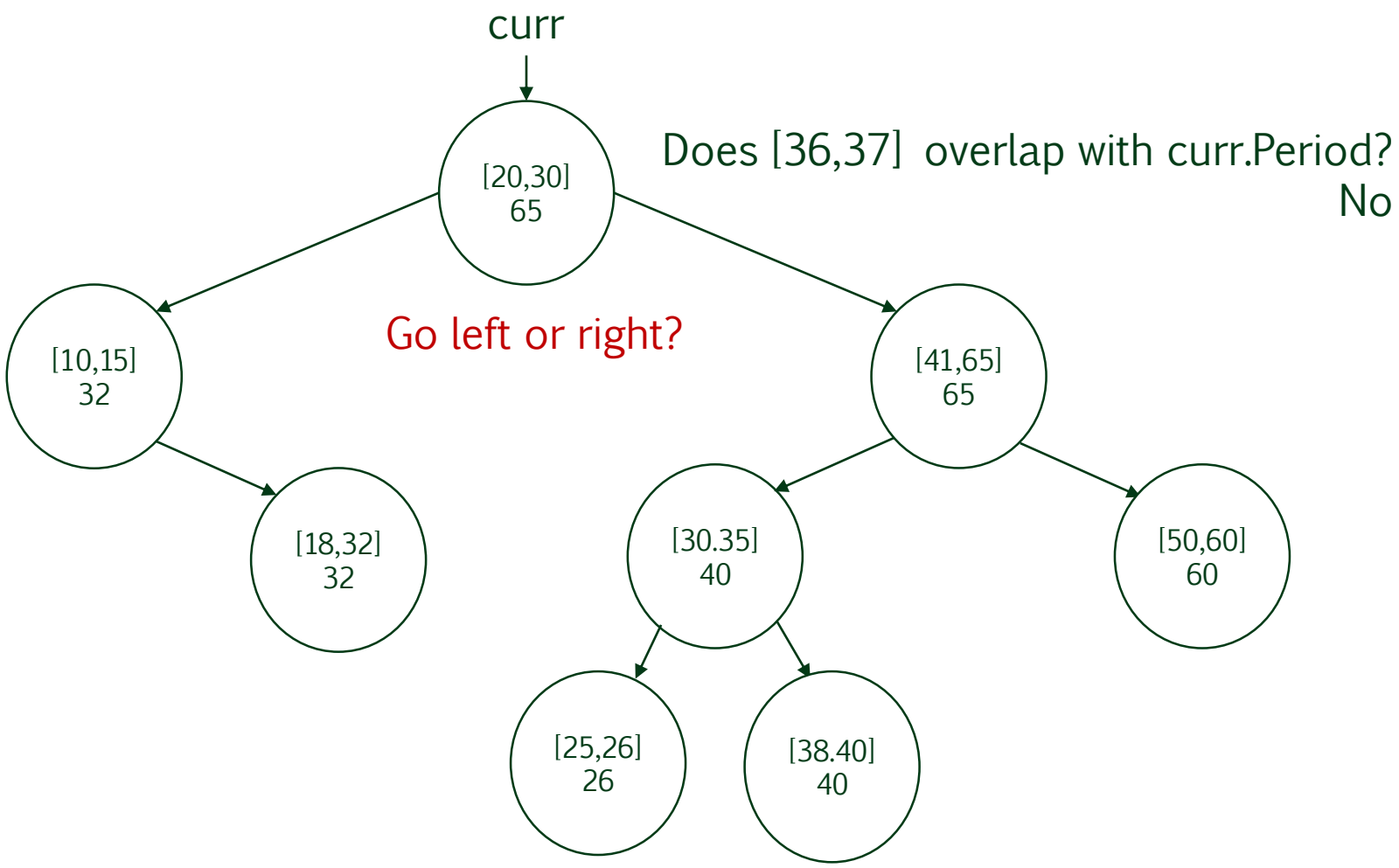
Return interval that overlaps with [36,37]



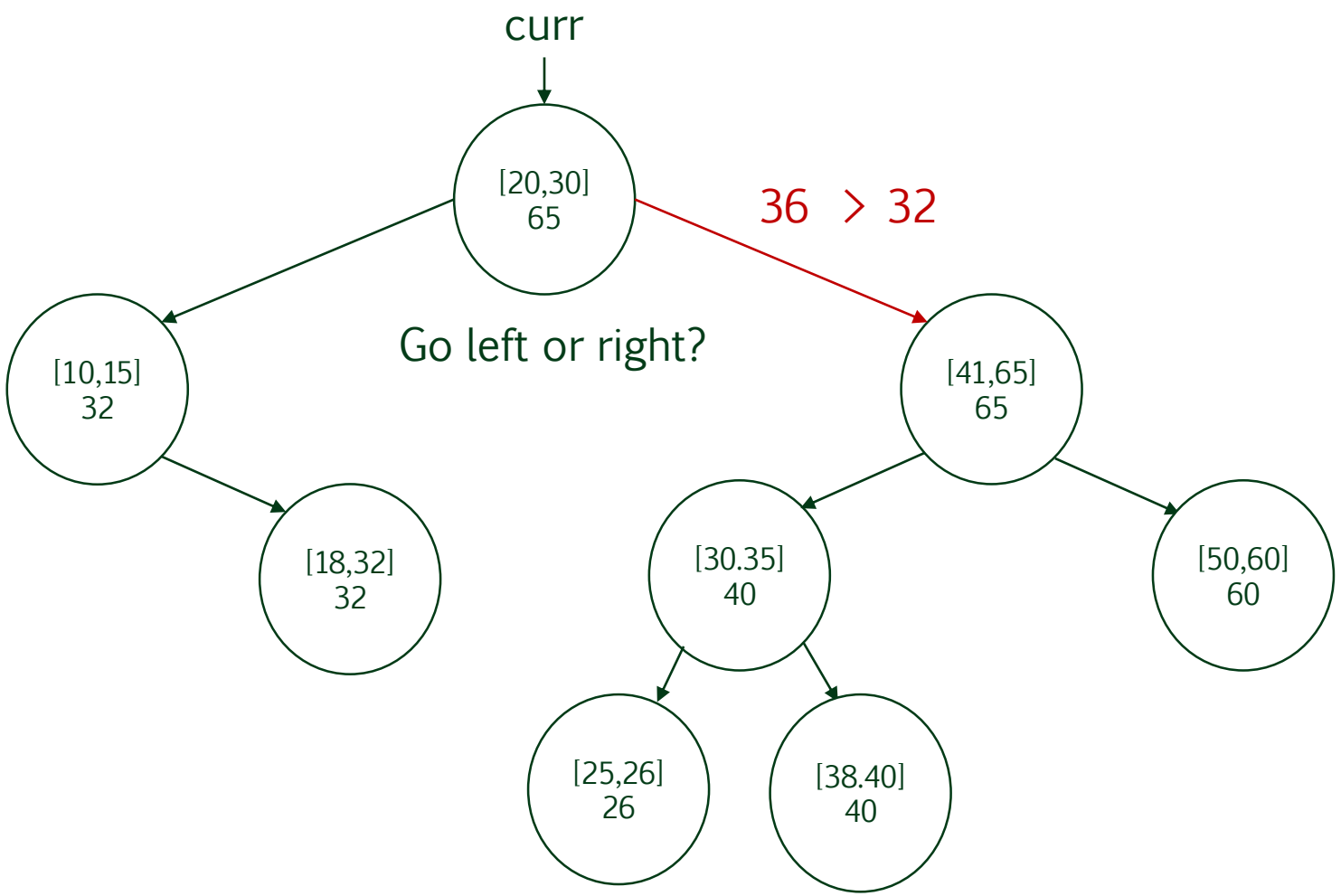
Does [36,37] overlap with curr.Period?

No

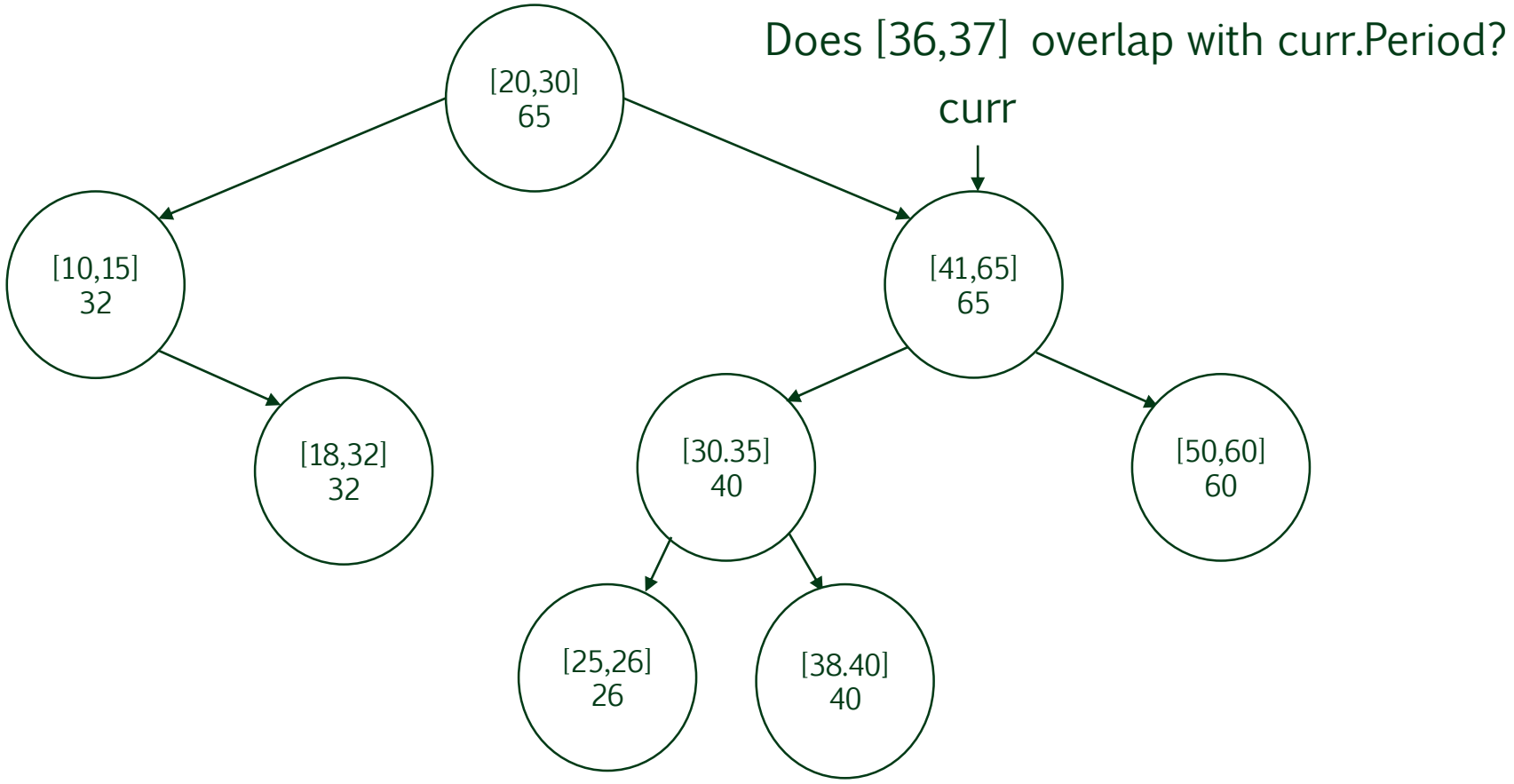
Return interval that overlaps with [36,37]



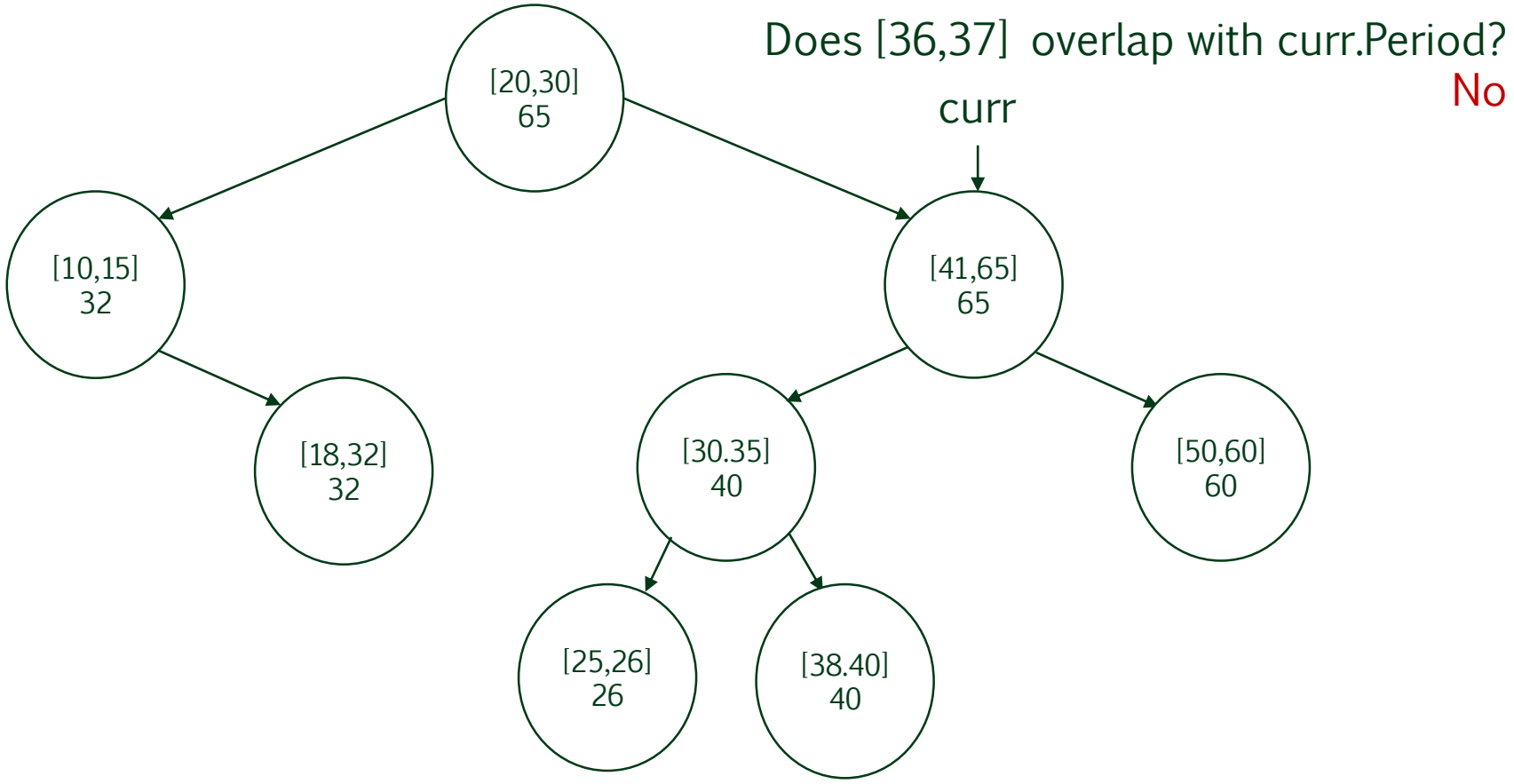
Return interval that overlaps with [36,37]



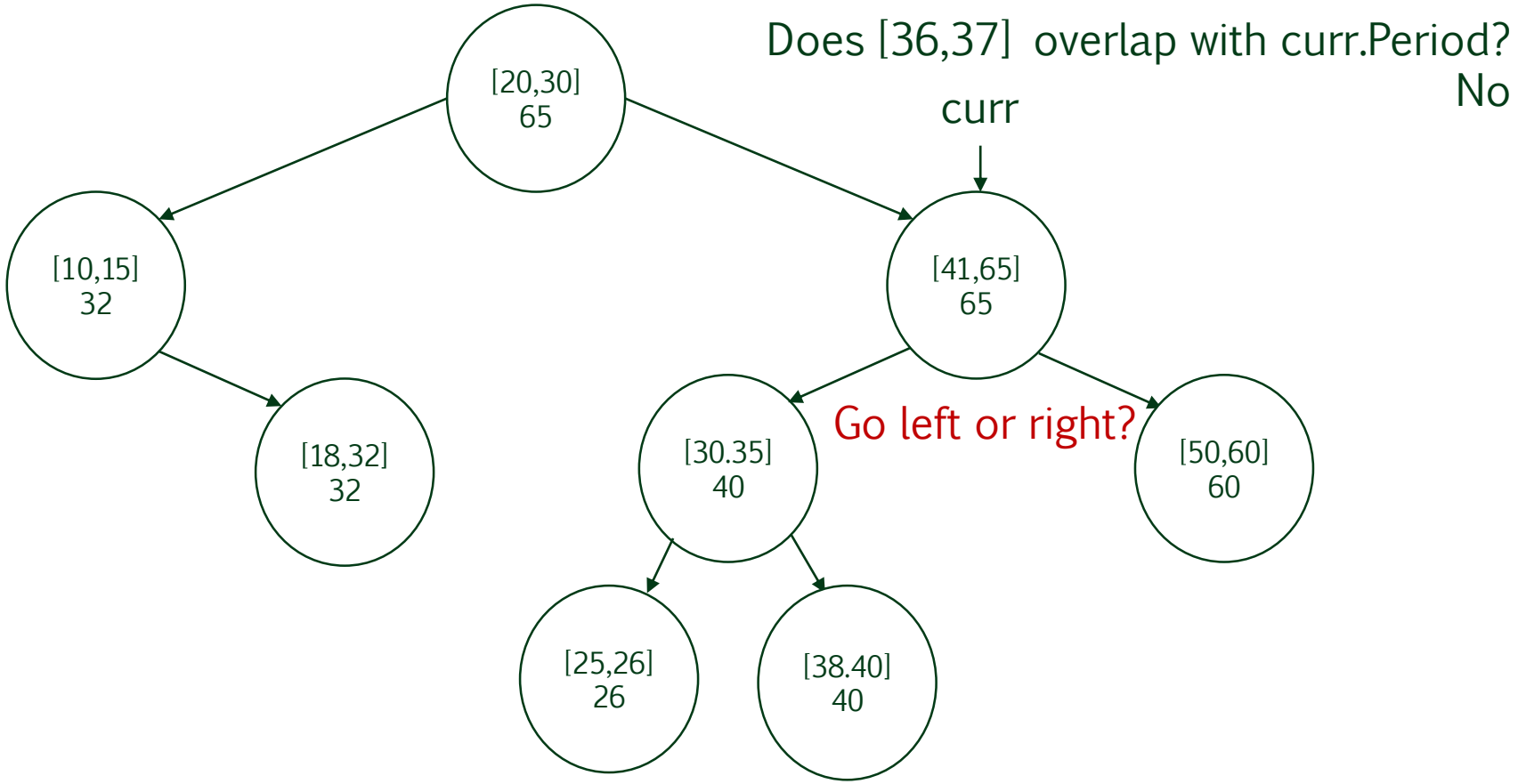
Return interval that overlaps with [36,37]



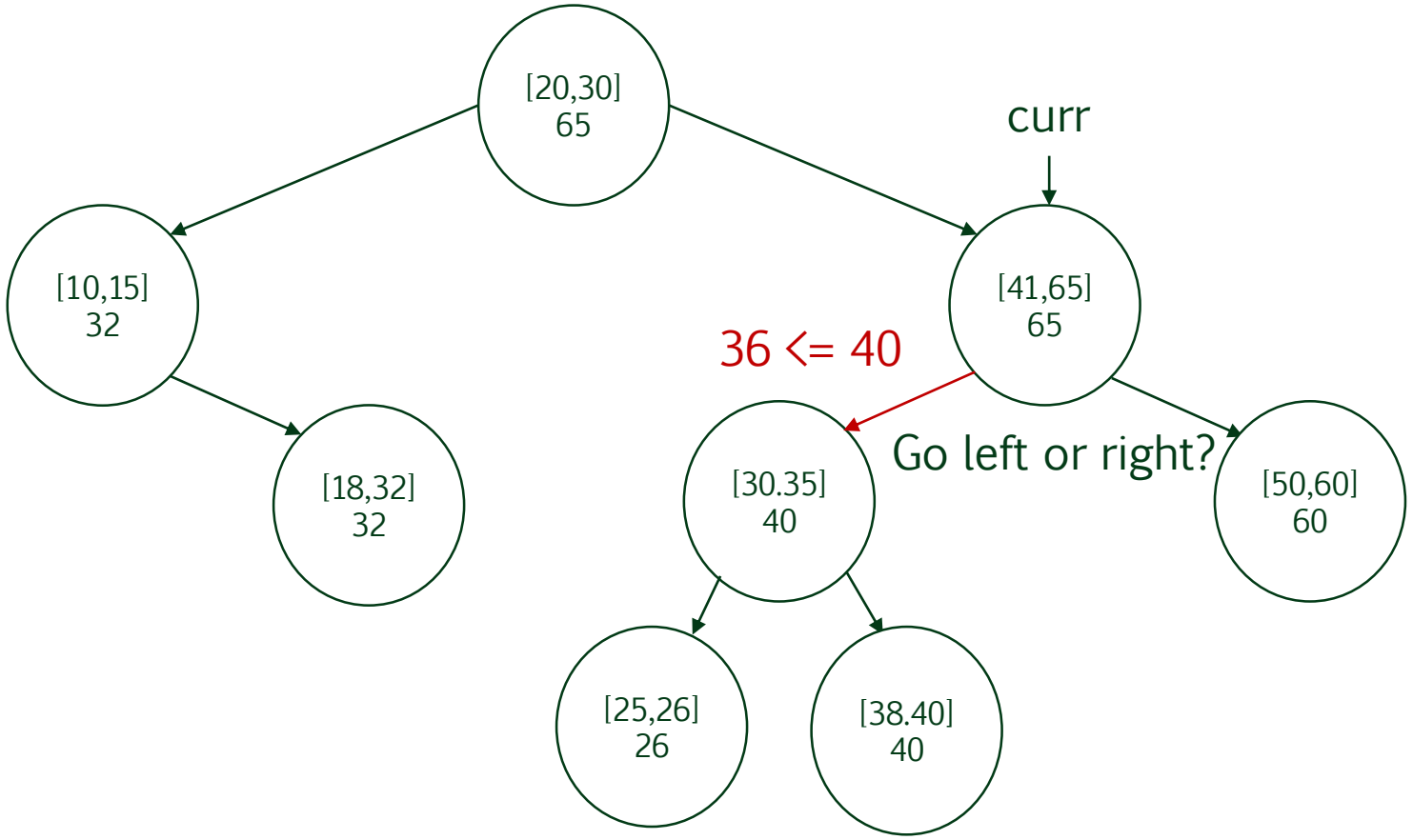
Return interval that overlaps with [36,37]



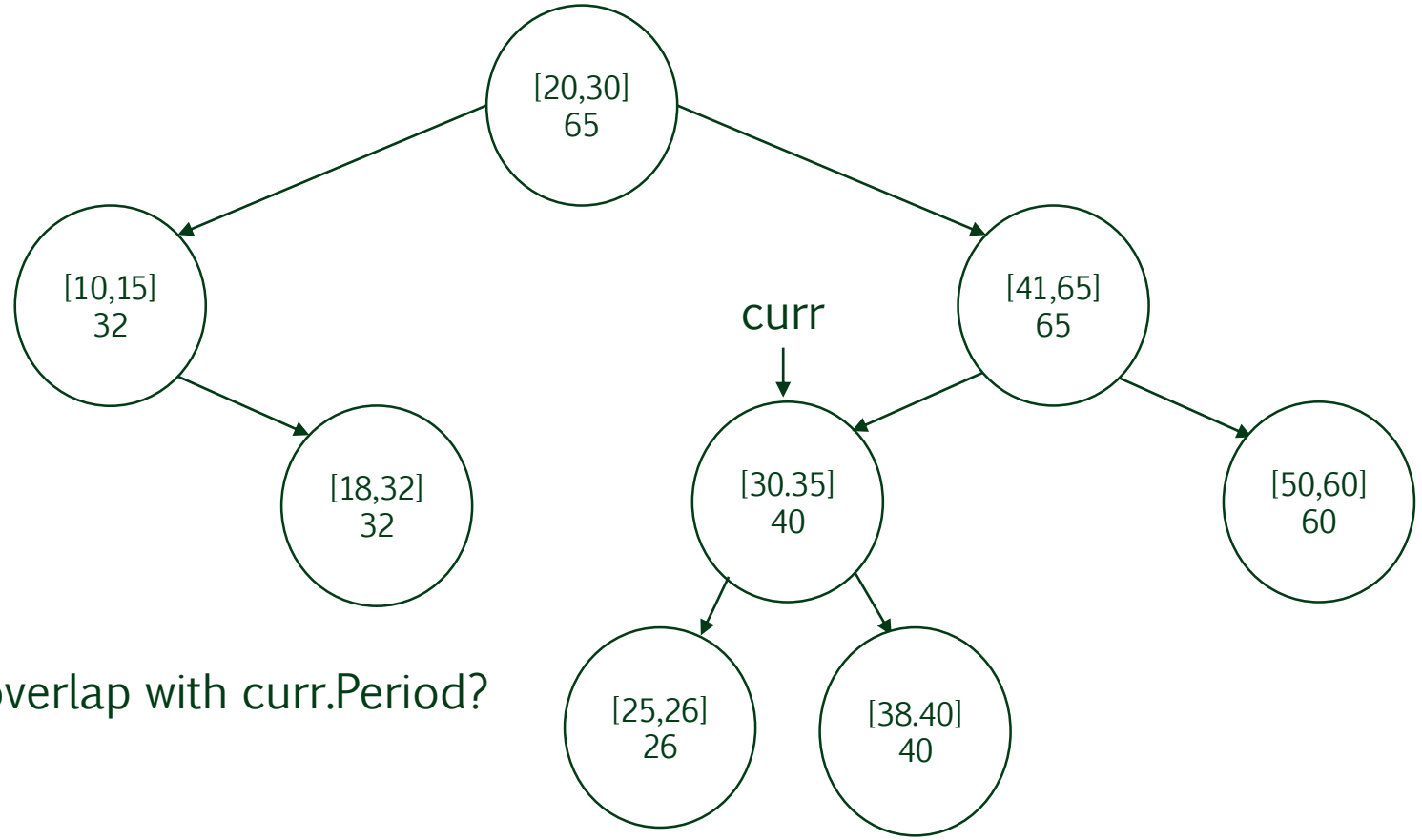
Return interval that overlaps with [36,37]



Return interval that overlaps with [36,37]

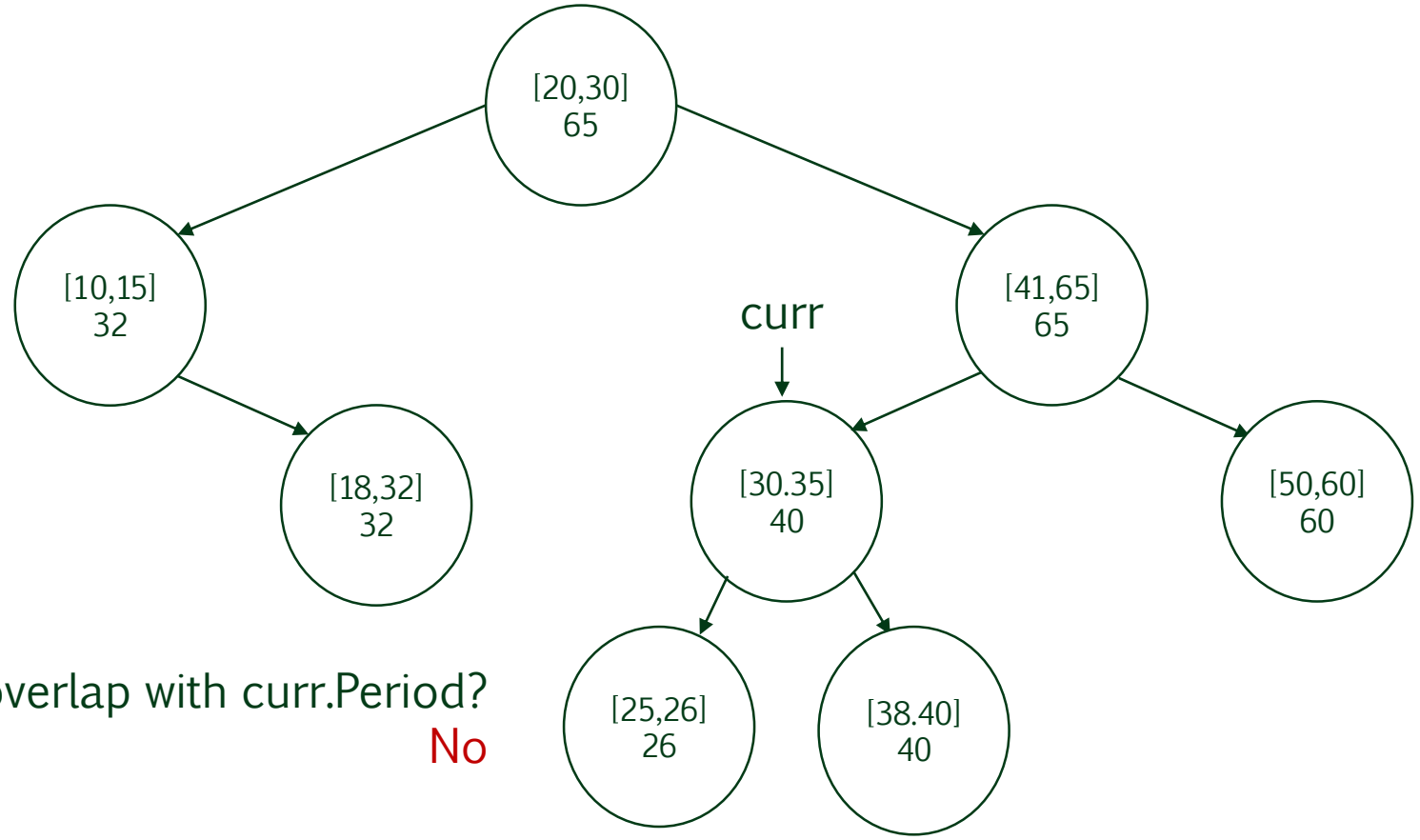


Return interval that overlaps with [36,37]



Does [36,37] overlap with curr.Period?

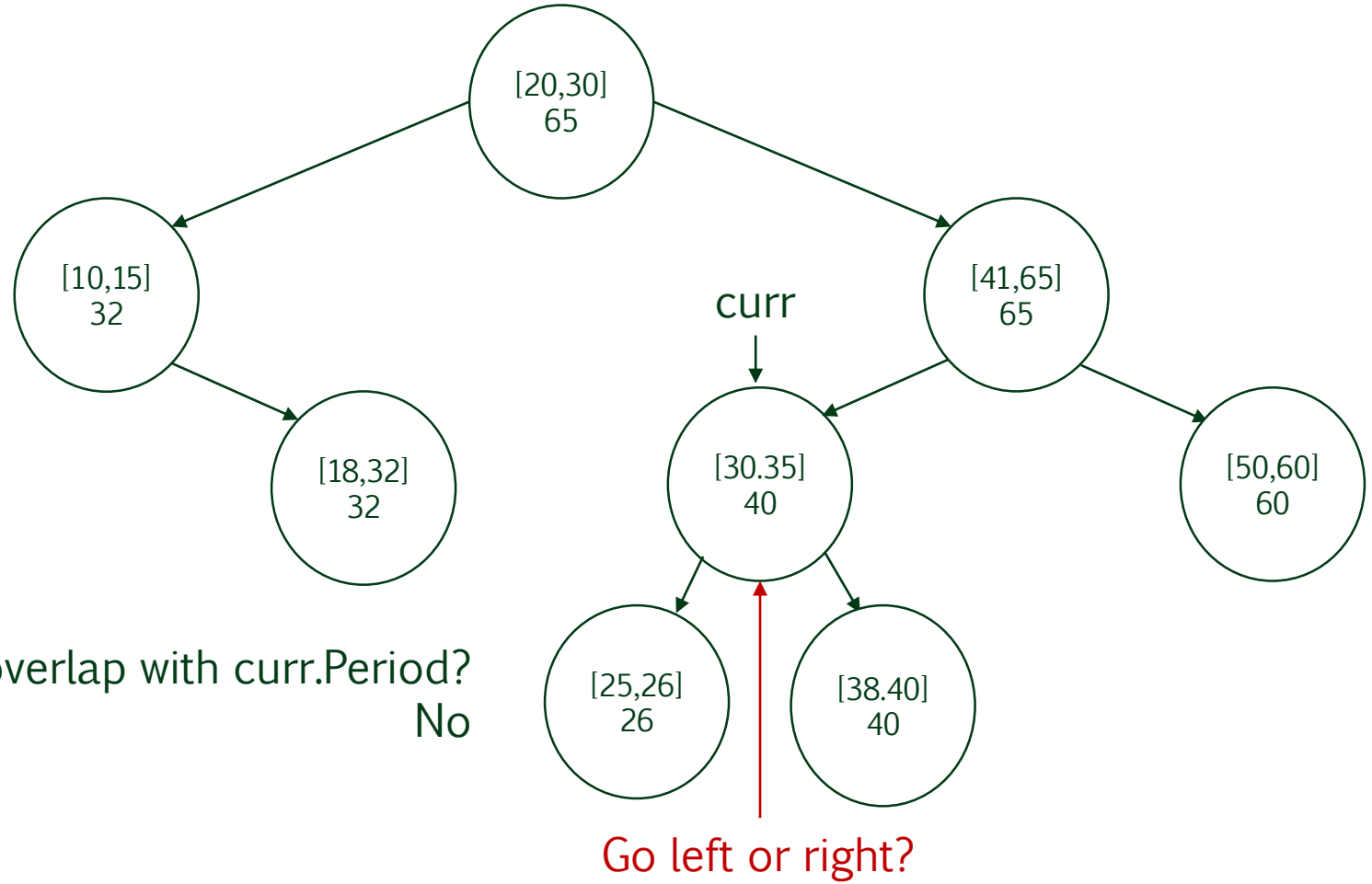
Return interval that overlaps with [36,37]



Does [36,37] overlap with curr.Period?

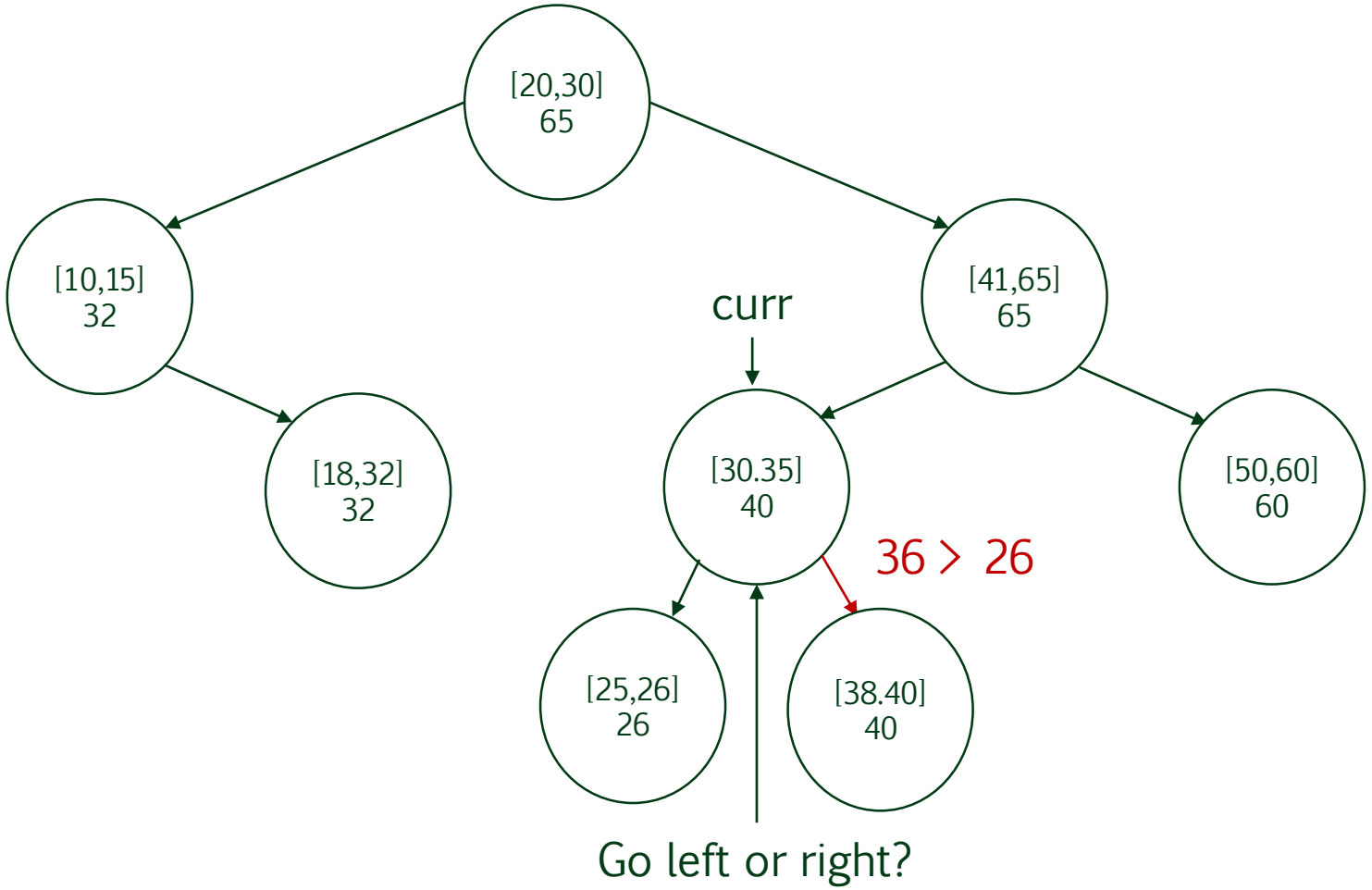
No

Return interval that overlaps with [36,37]

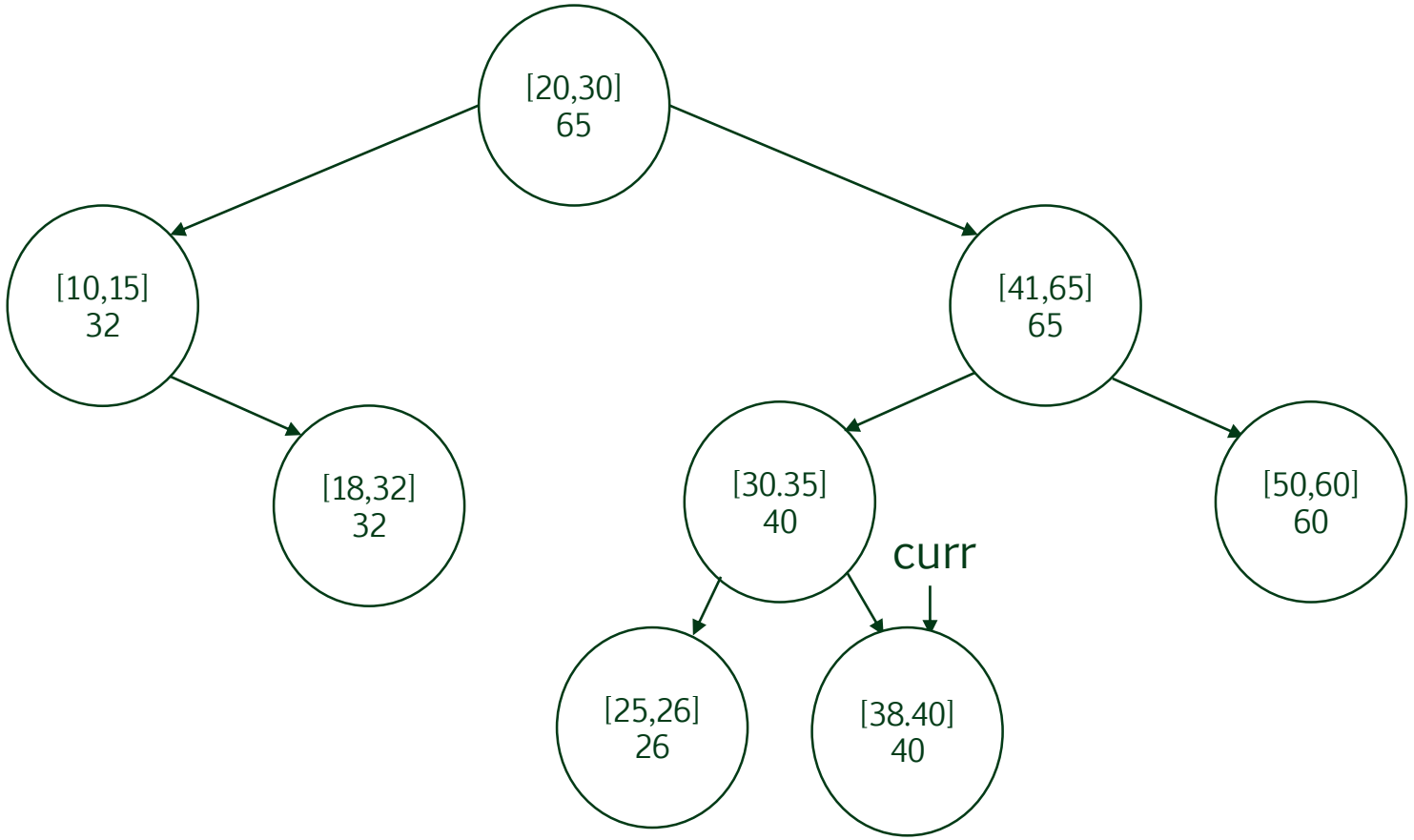


Does [36,37] overlap with curr.Period?
No

Return interval that overlaps with [36,37]

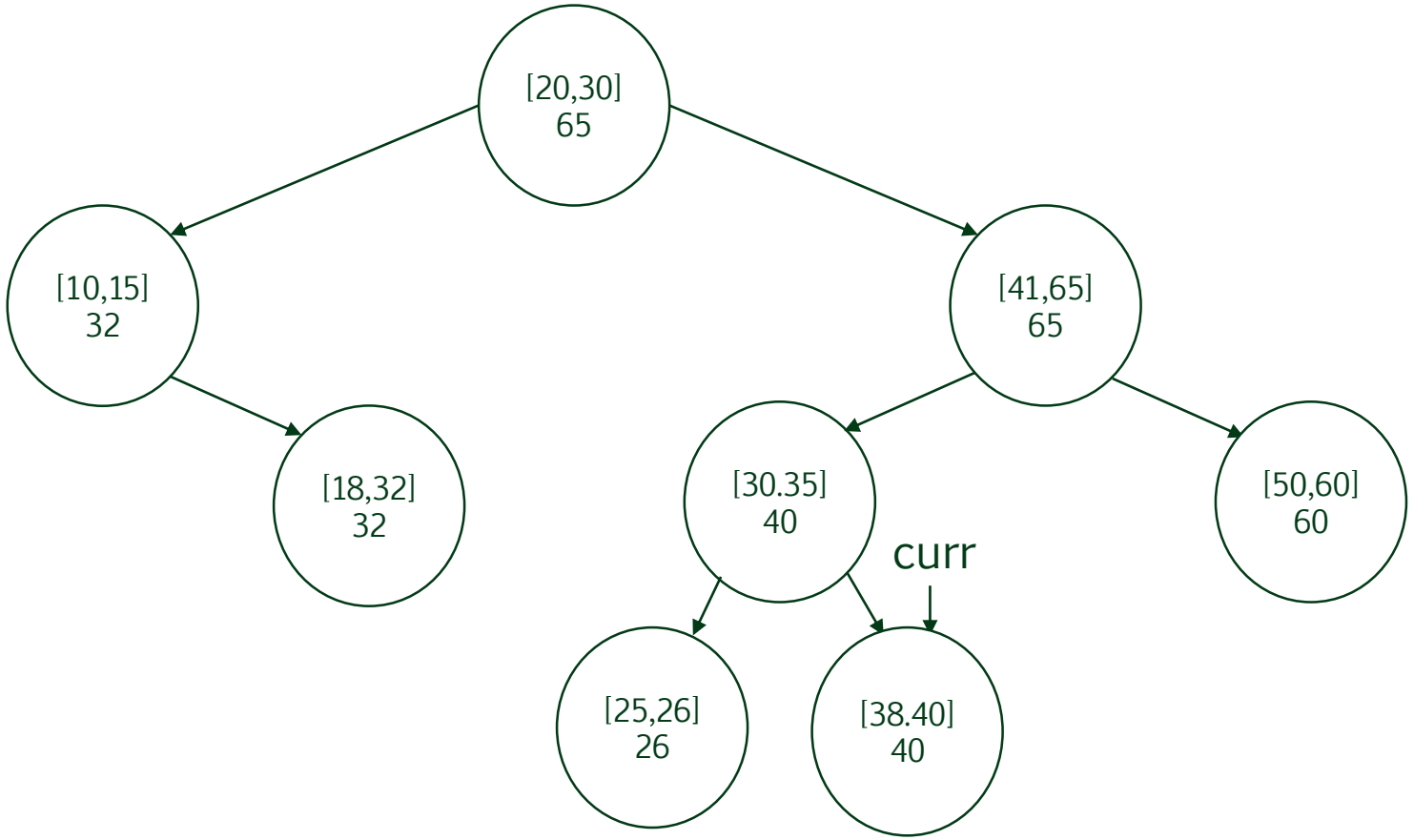


Return interval that overlaps with [36,37]



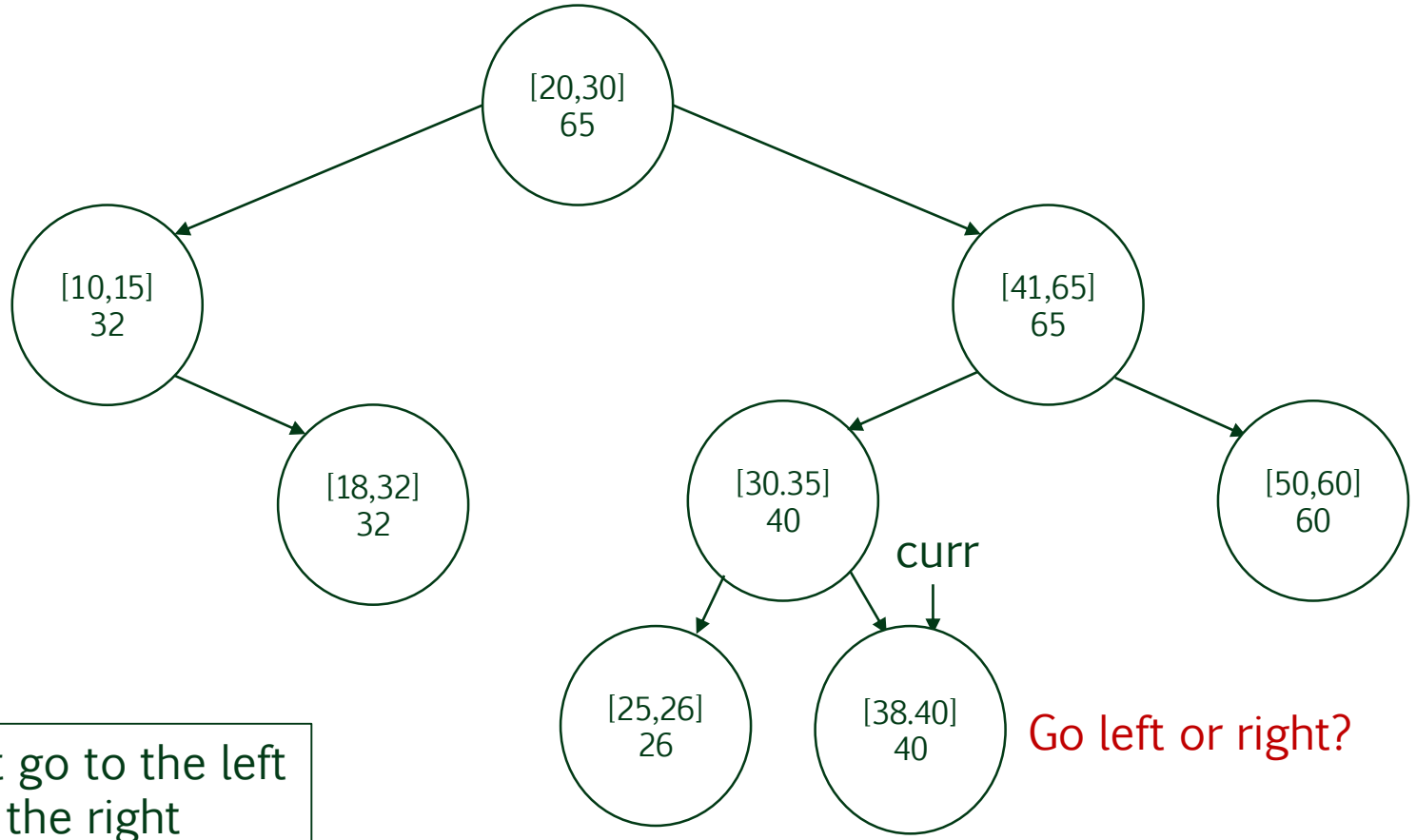
Does [36,37] overlap with curr.Period?

Return interval that overlaps with [36,37]



Does [36,37] overlap with curr.Period? No

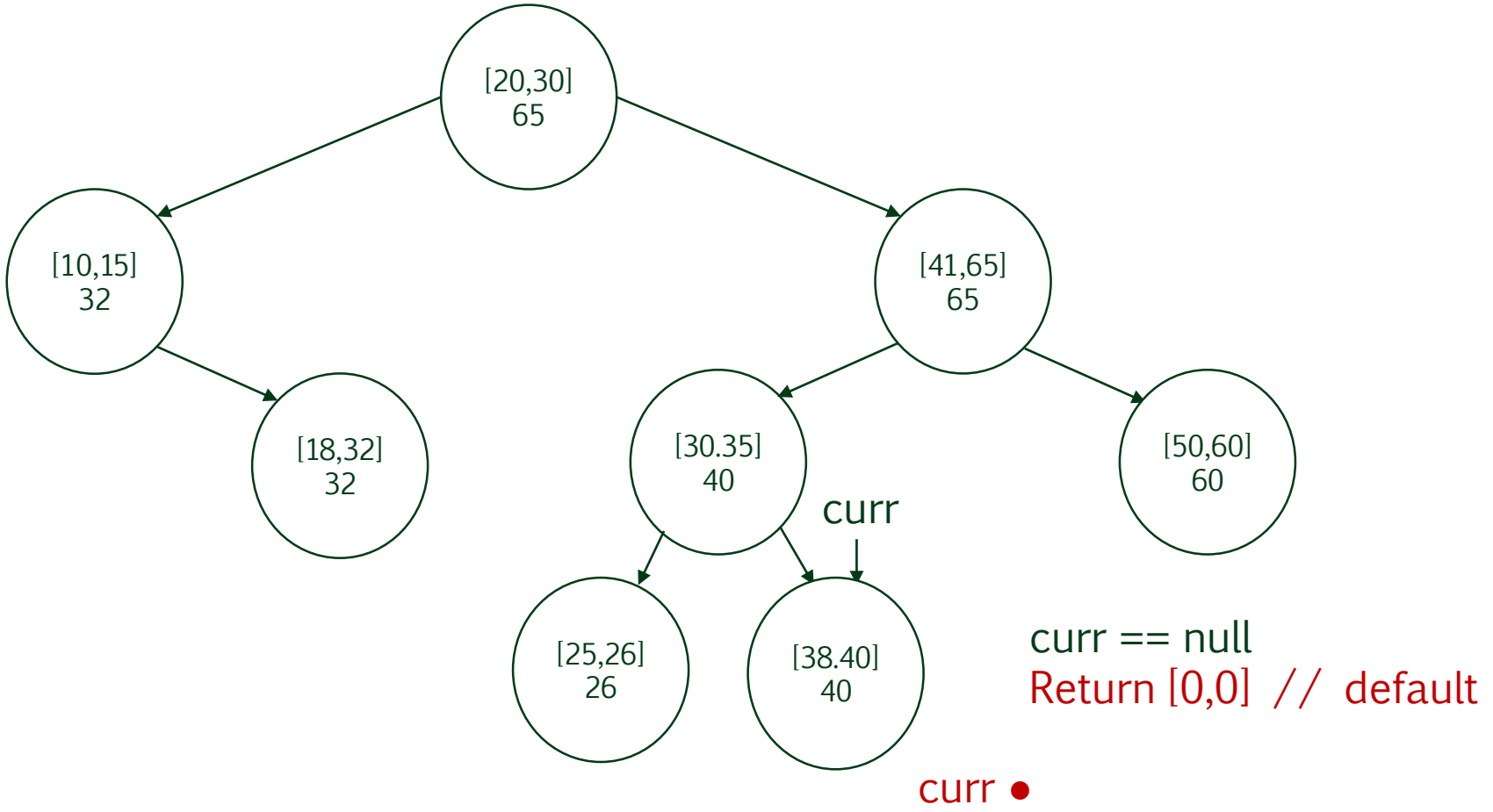
Return interval that overlaps with [36,37]



If you cannot go to the left always go to the right

Does [36,37] overlap with curr.Period? No

Return interval that overlaps with [36,37]



Time complexity of Overlap

- The while loop executes a maximum of h+1 iterations where h is the height of the interval tree (treap)

- Since the expected height of the interval tree is O(log n), the expected time complexity of Overlap is O(log n)