SEGMENT TREES

ISIS 2801

Segment trees

Segment trees are a useful data structure to solve multiple problems (very common), usually to solve **range queries**

Sometimes **Interval** or **Tournament** trees

Dynamic programming is **applicable** for optimization problems that can be built **bottom-up.** These problems often present a common property (e.g., can be expressed by a recursive equation, have overlapping subproblems)

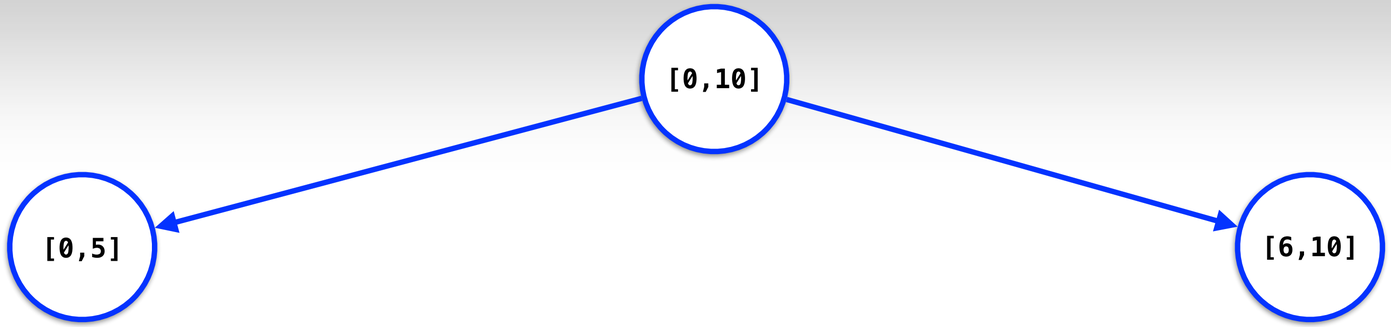
Querying ranges

index	0	1	2	3	4	5	6	7	8	9	10
value	**2**	**3**	**4**	**1**	**7**	**2**	**8**	**7**	**4**	**1**	**5**

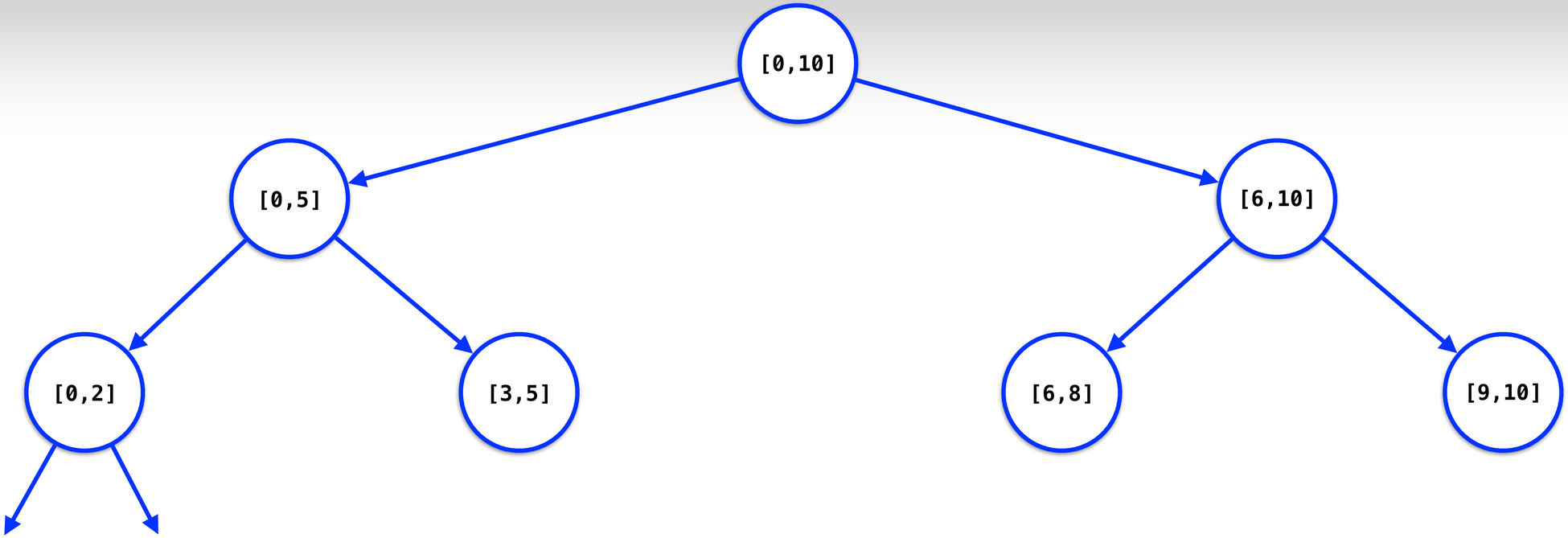
minimum of range **[2,7]**?

Building segment trees

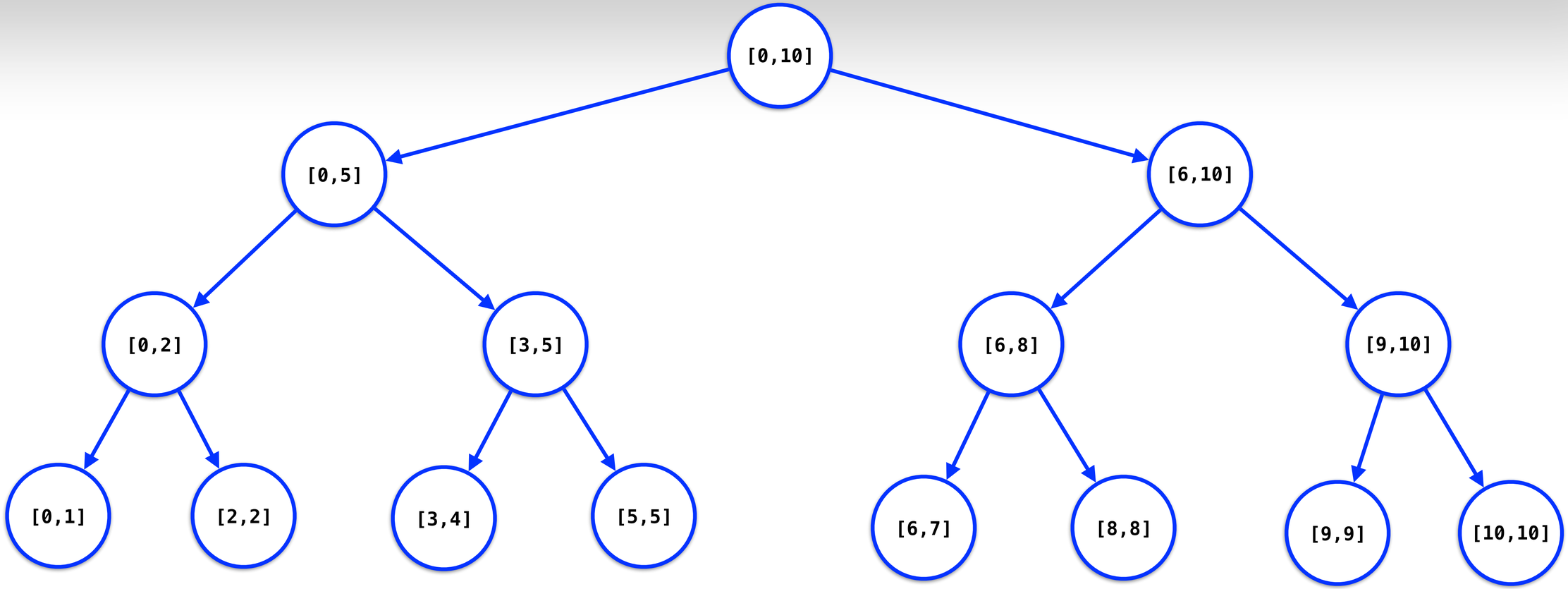
Building segment trees



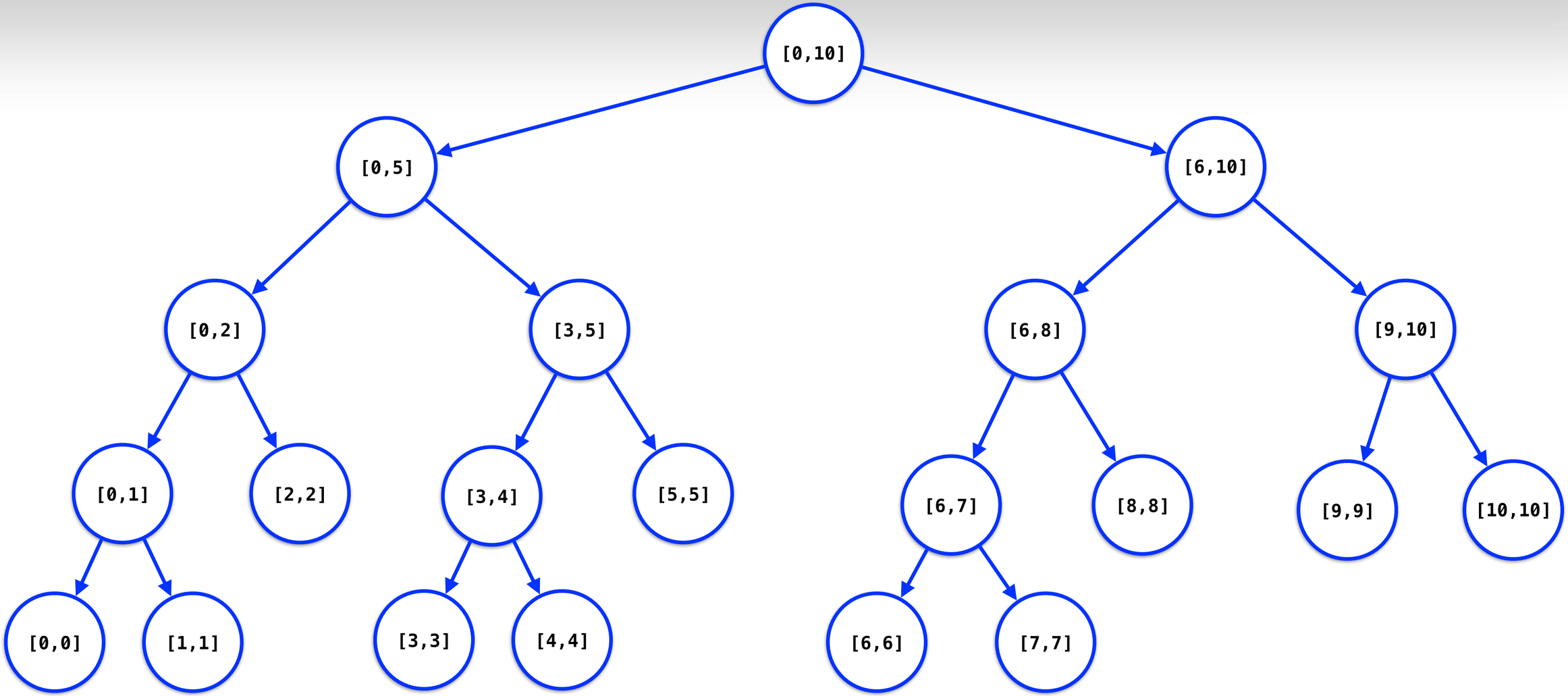
Building segment trees



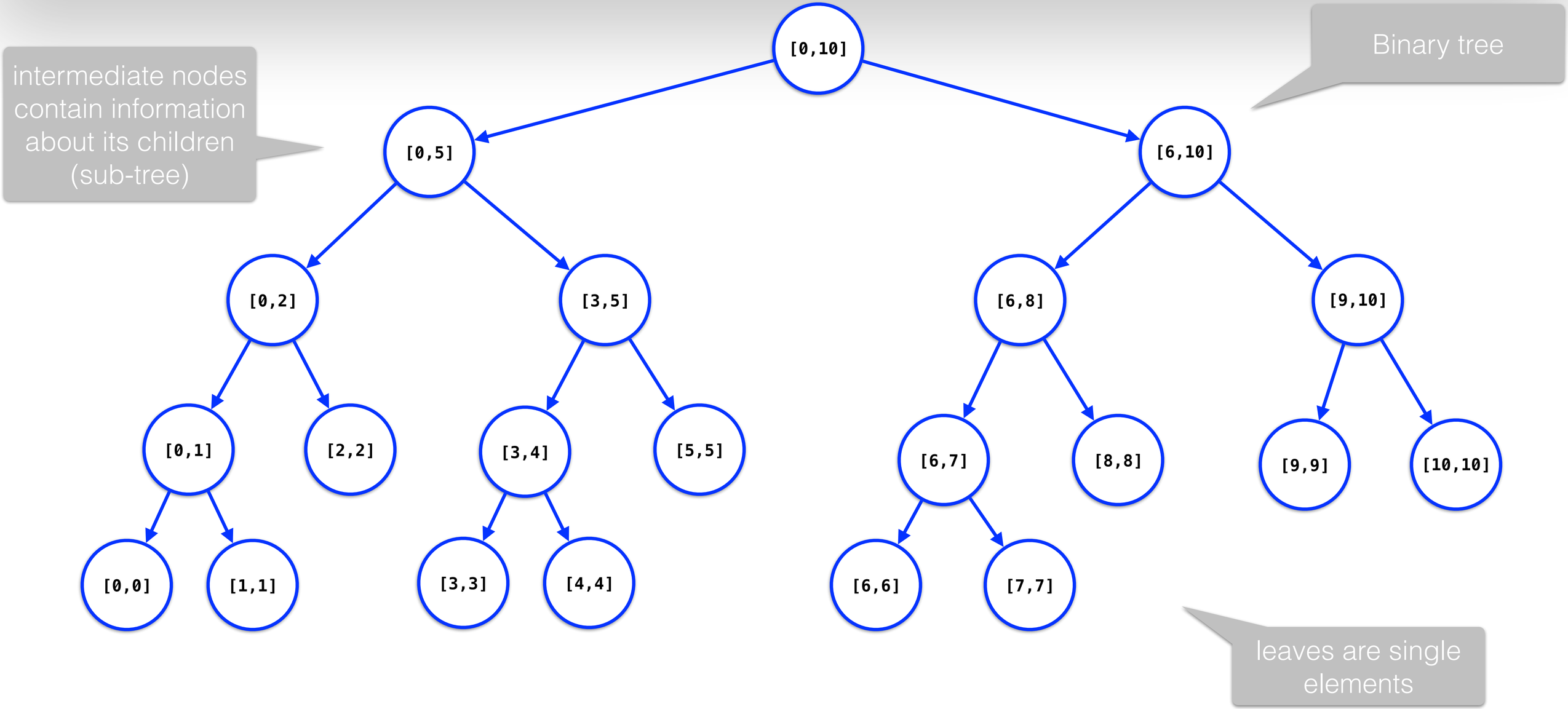
Building segment trees



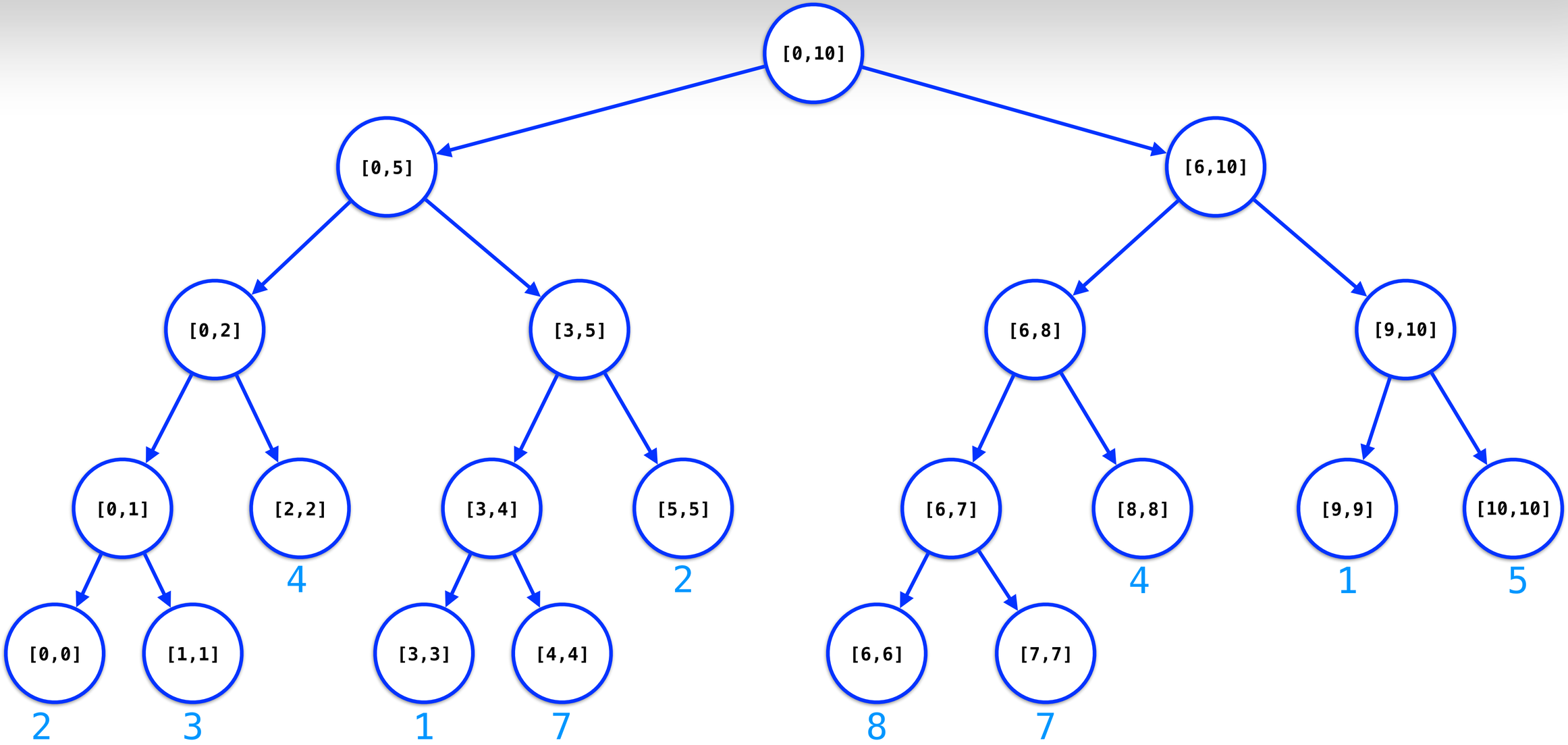
Building segment trees



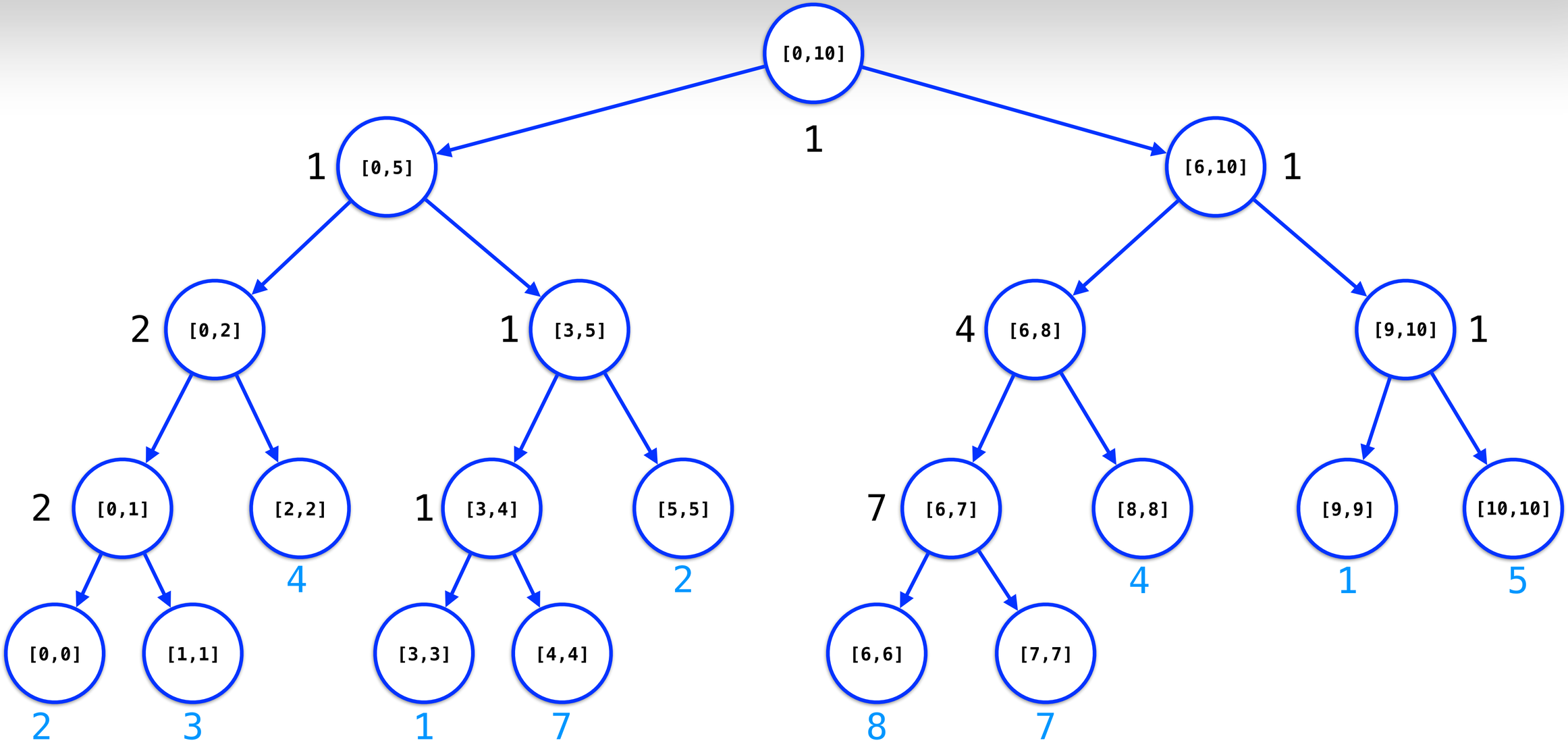
Building segment trees



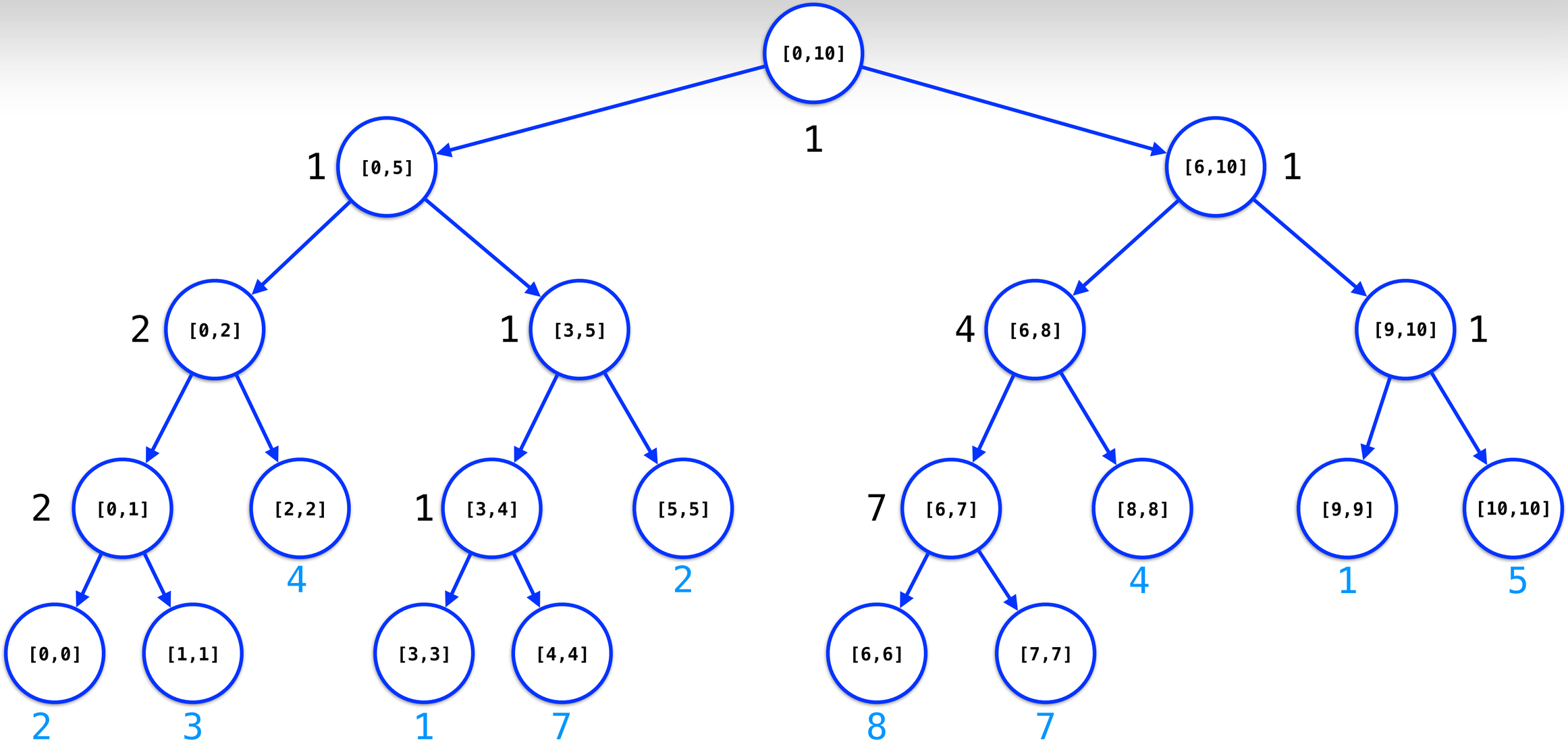
Calculate the minimum of a range



Calculate the minimum of the tree



Calculate the minimum of the tree



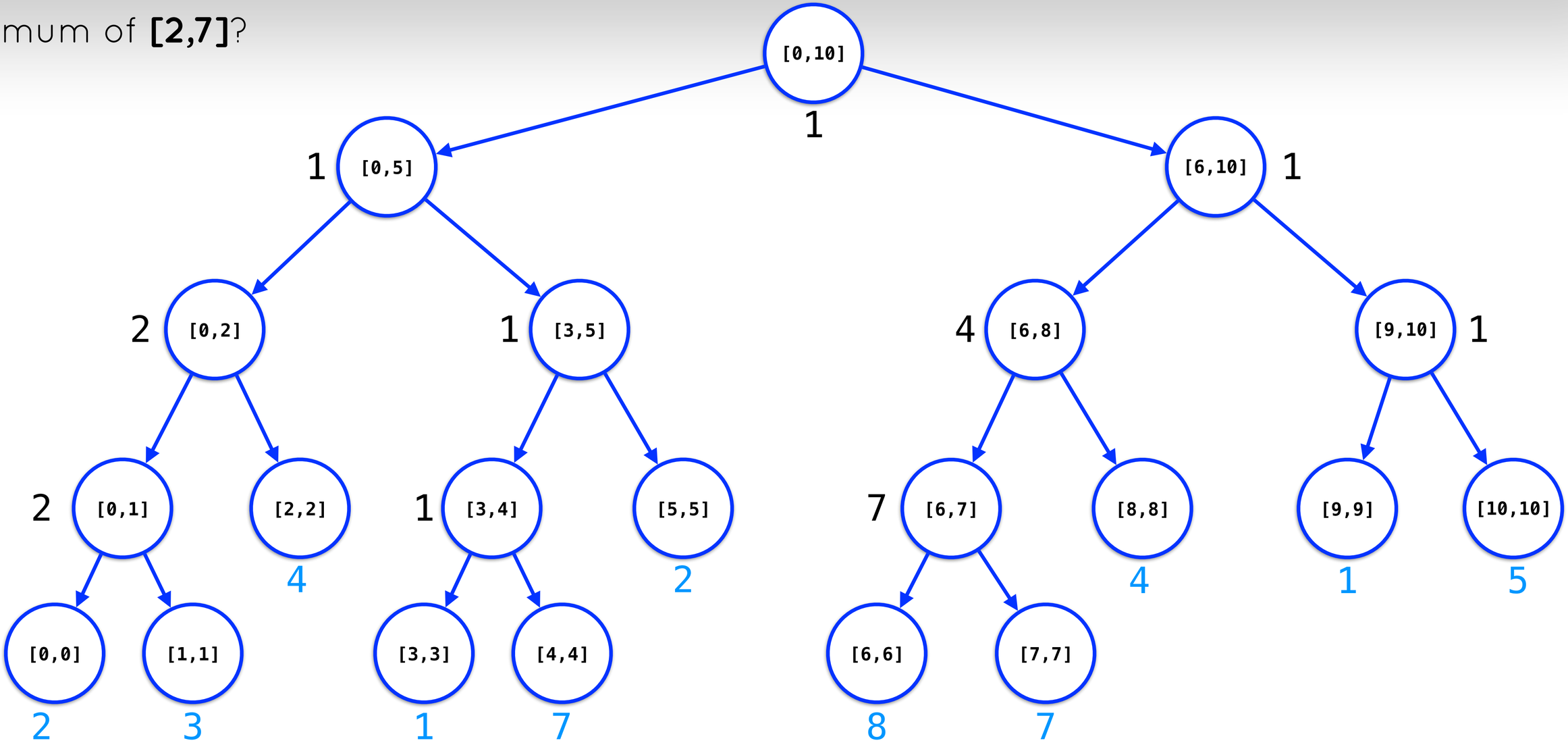
Range minimum queries

Given a range, find its minimum element? `-> min([i,j])`

To do this, we must find sets completely covered by the given range

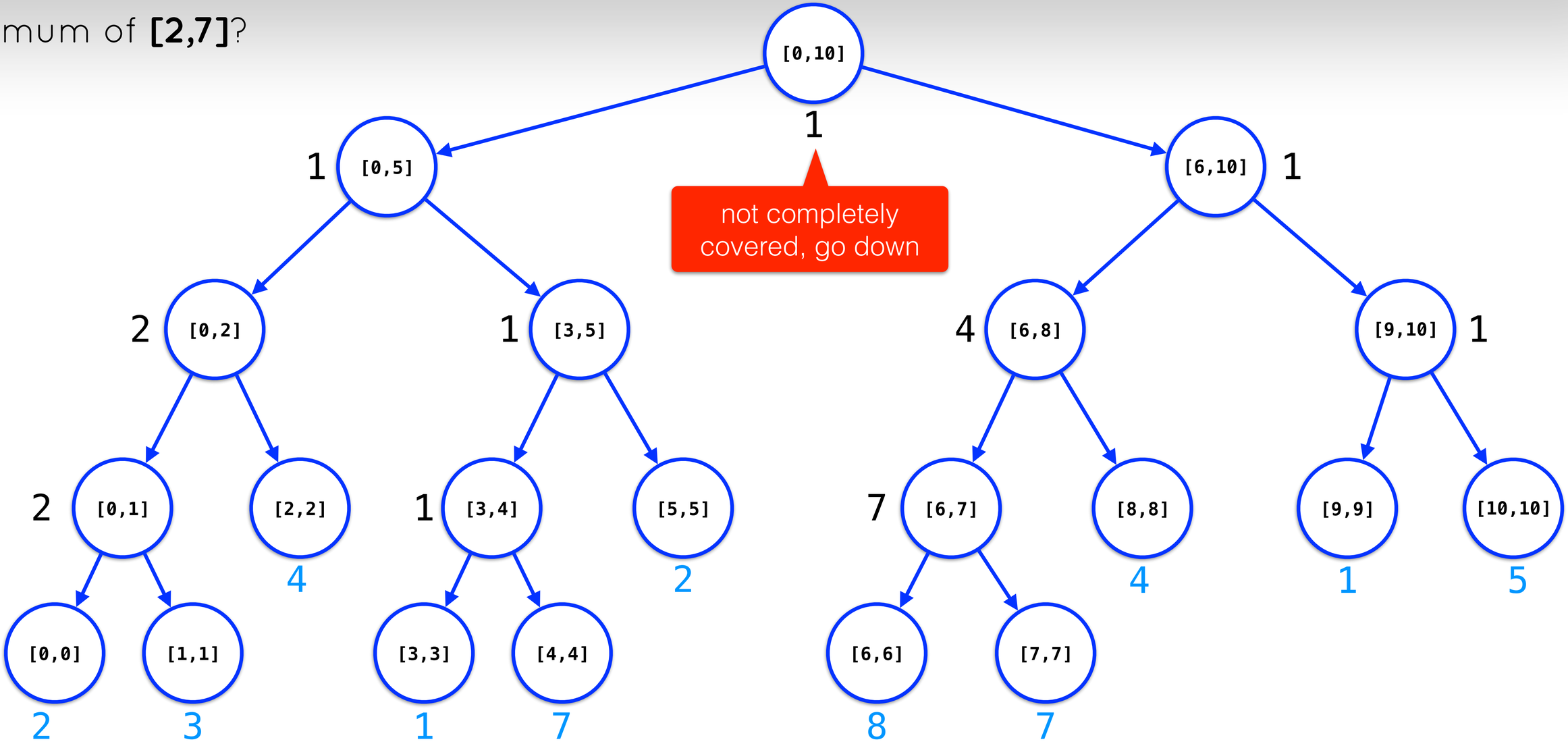
Range minimum queries

minimum of **[2,7]**?



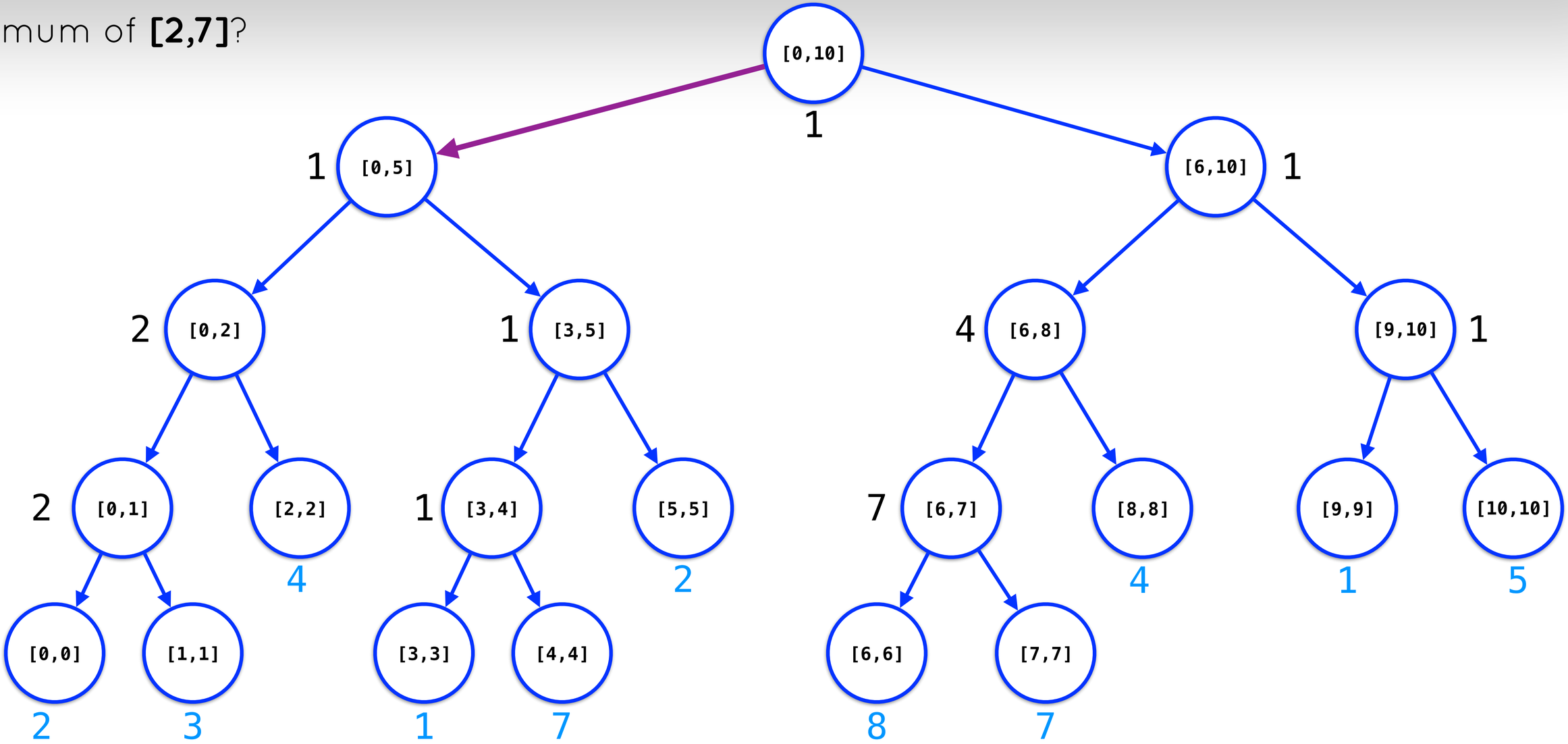
Range minimum queries

minimum of **[2,7]**?



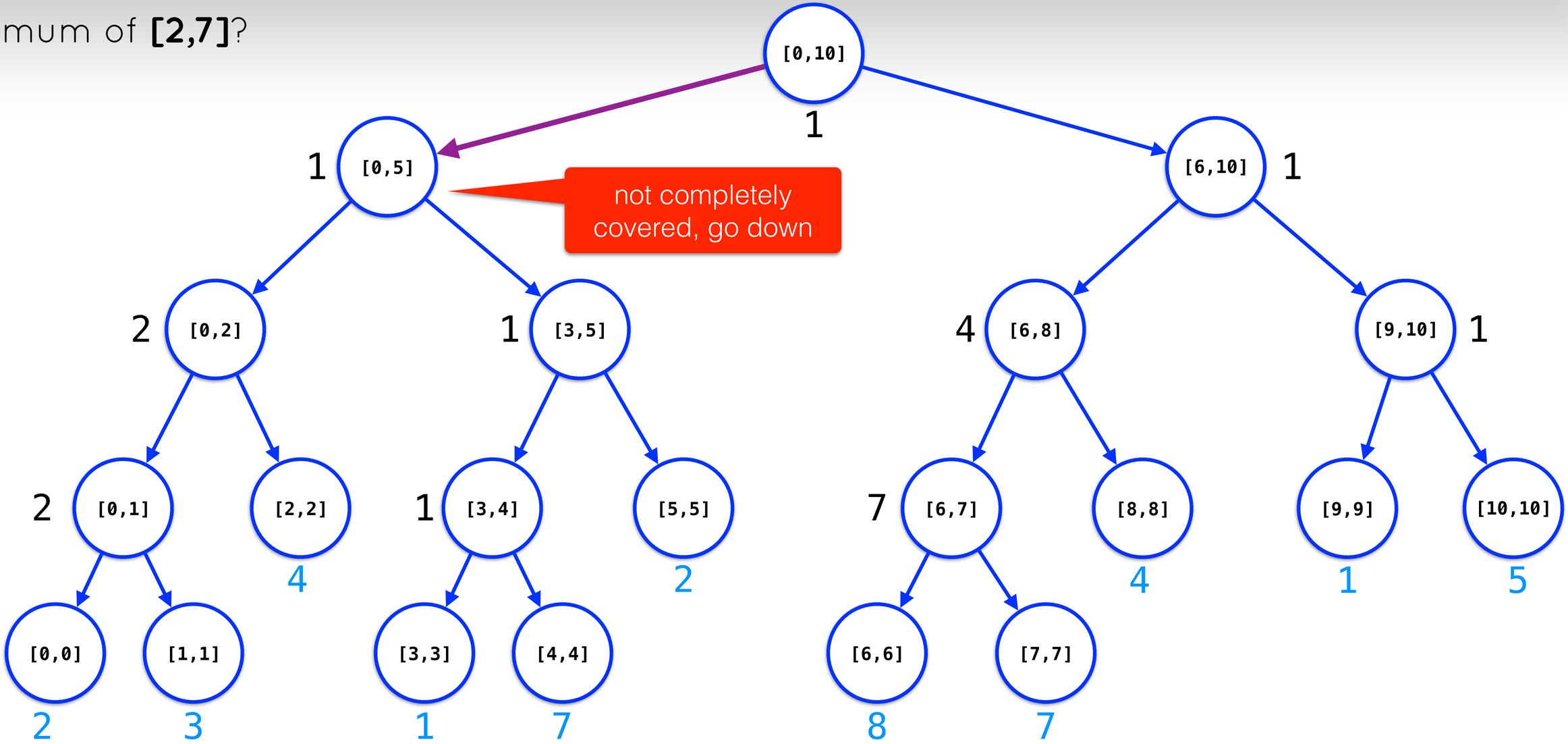
Range minimum queries

minimum of **[2,7]**?



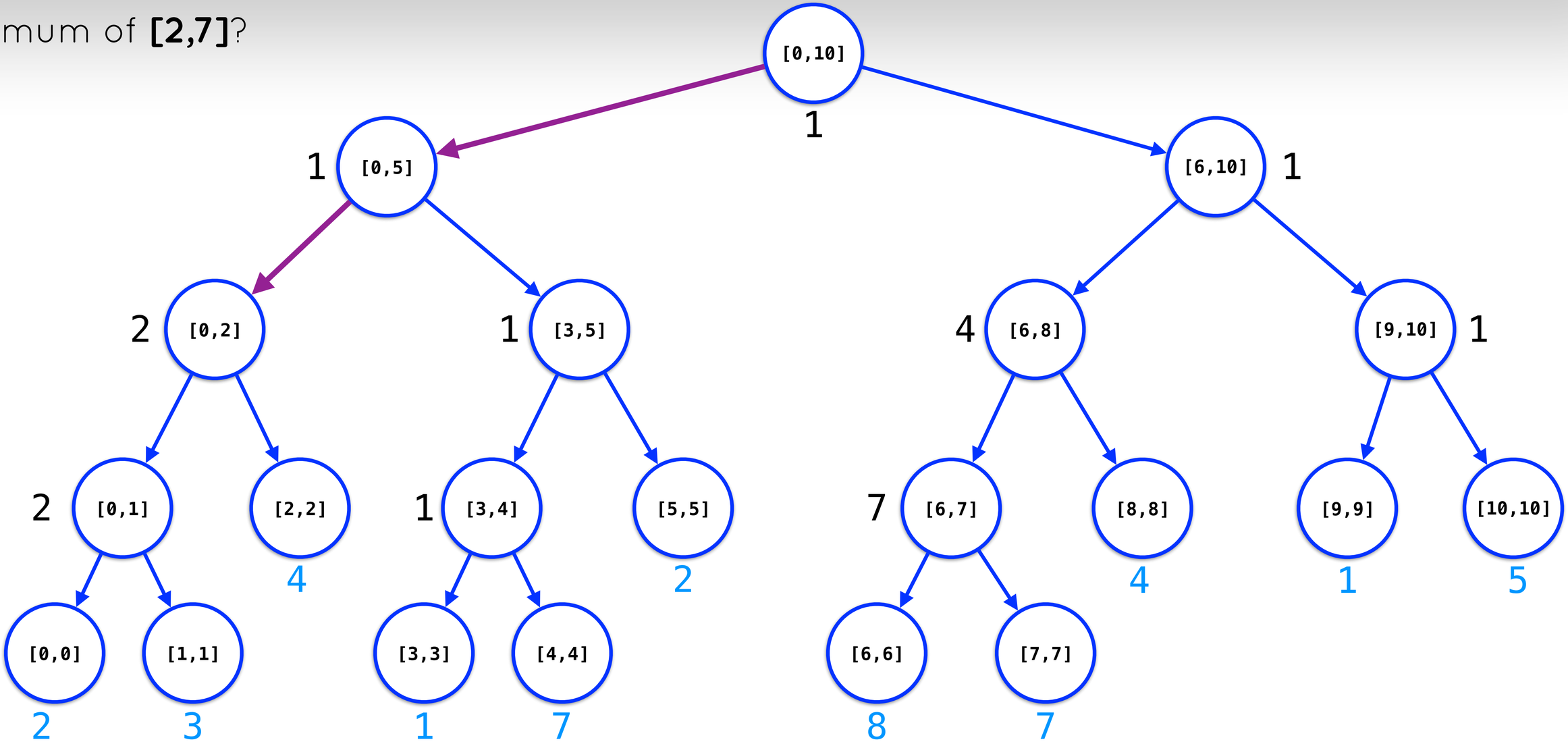
Range minimum queries

minimum of **[2,7]**?



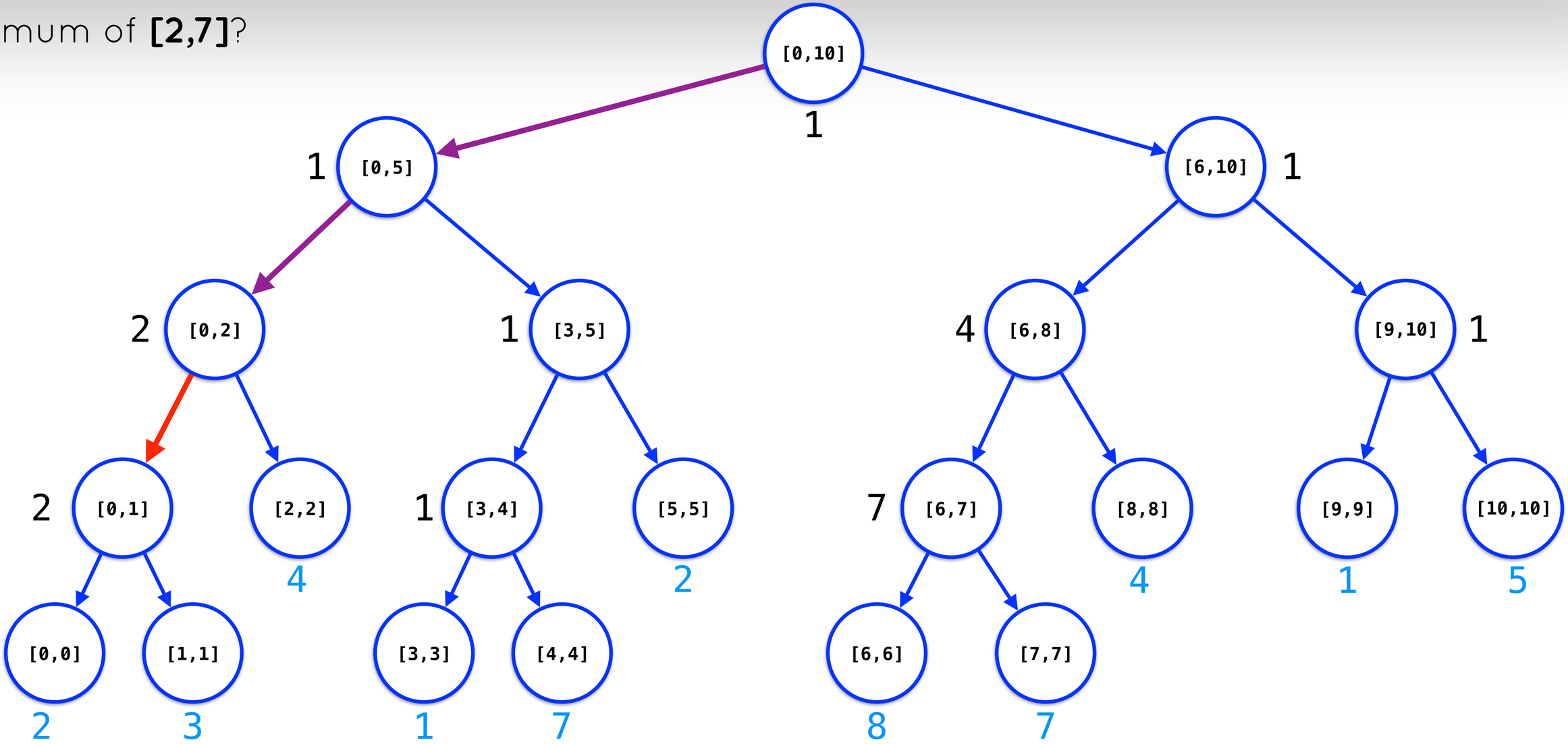
Range minimum queries

minimum of **[2,7]**?



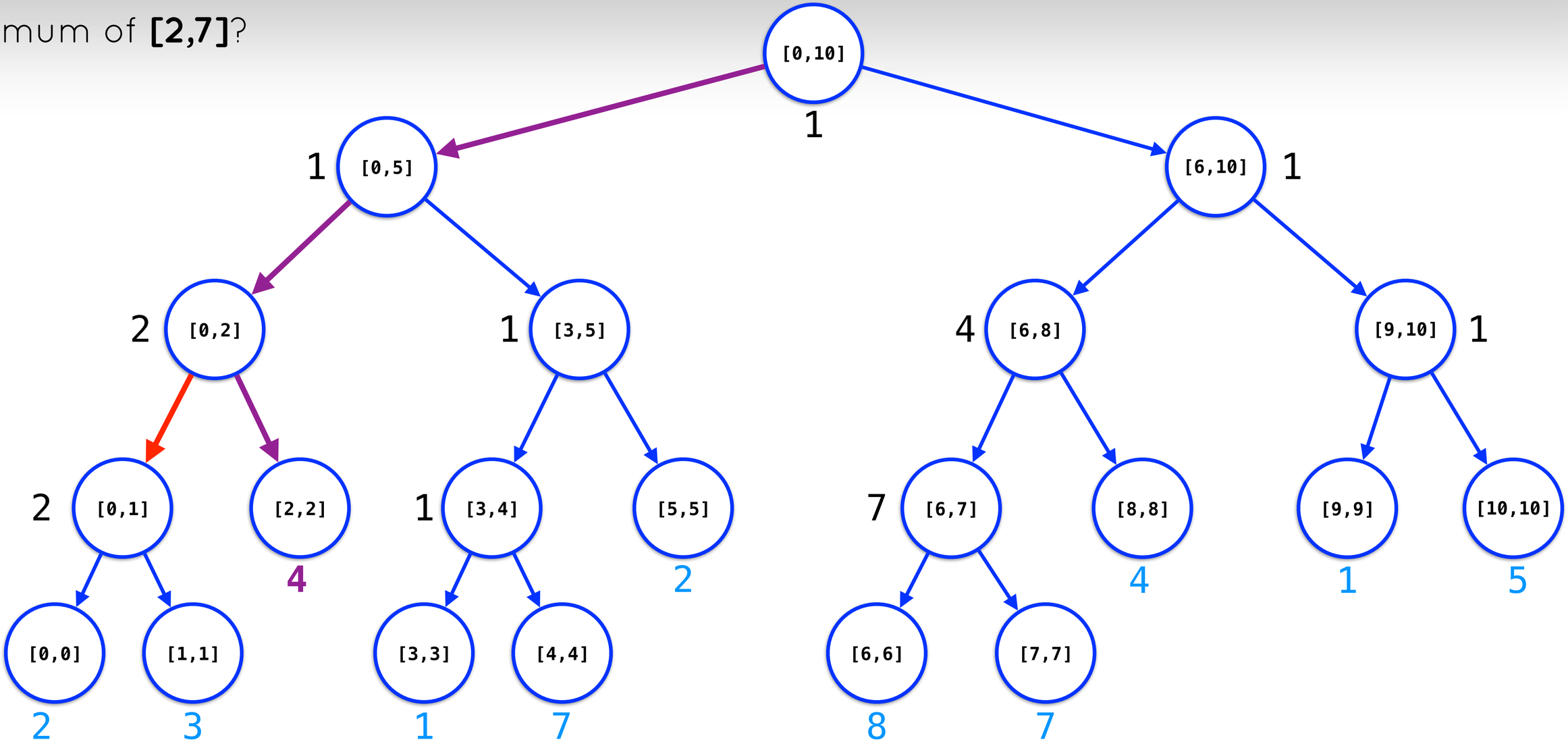
Range minimum queries

minimum of **[2,7]**?



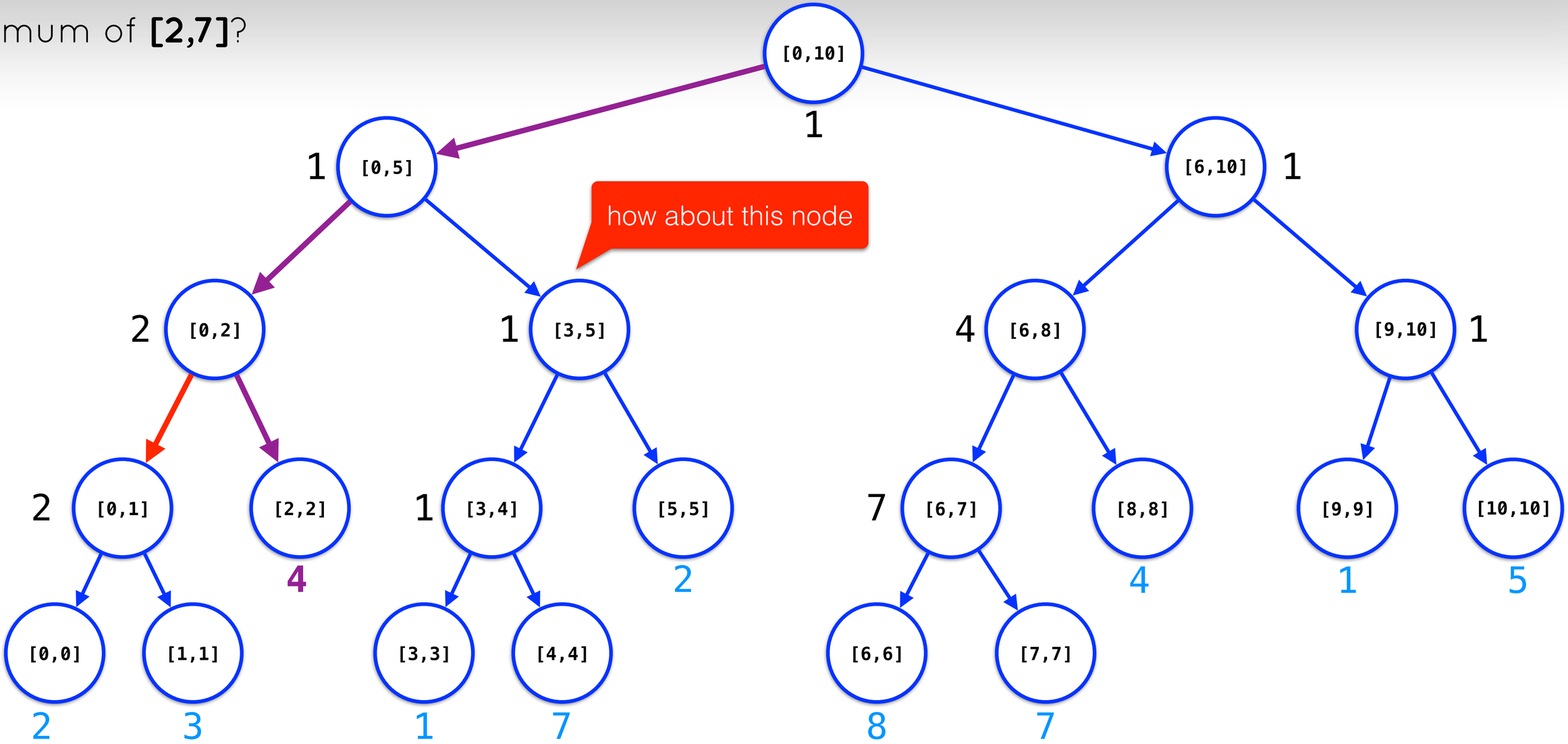
Range minimum queries

minimum of **[2,7]**?



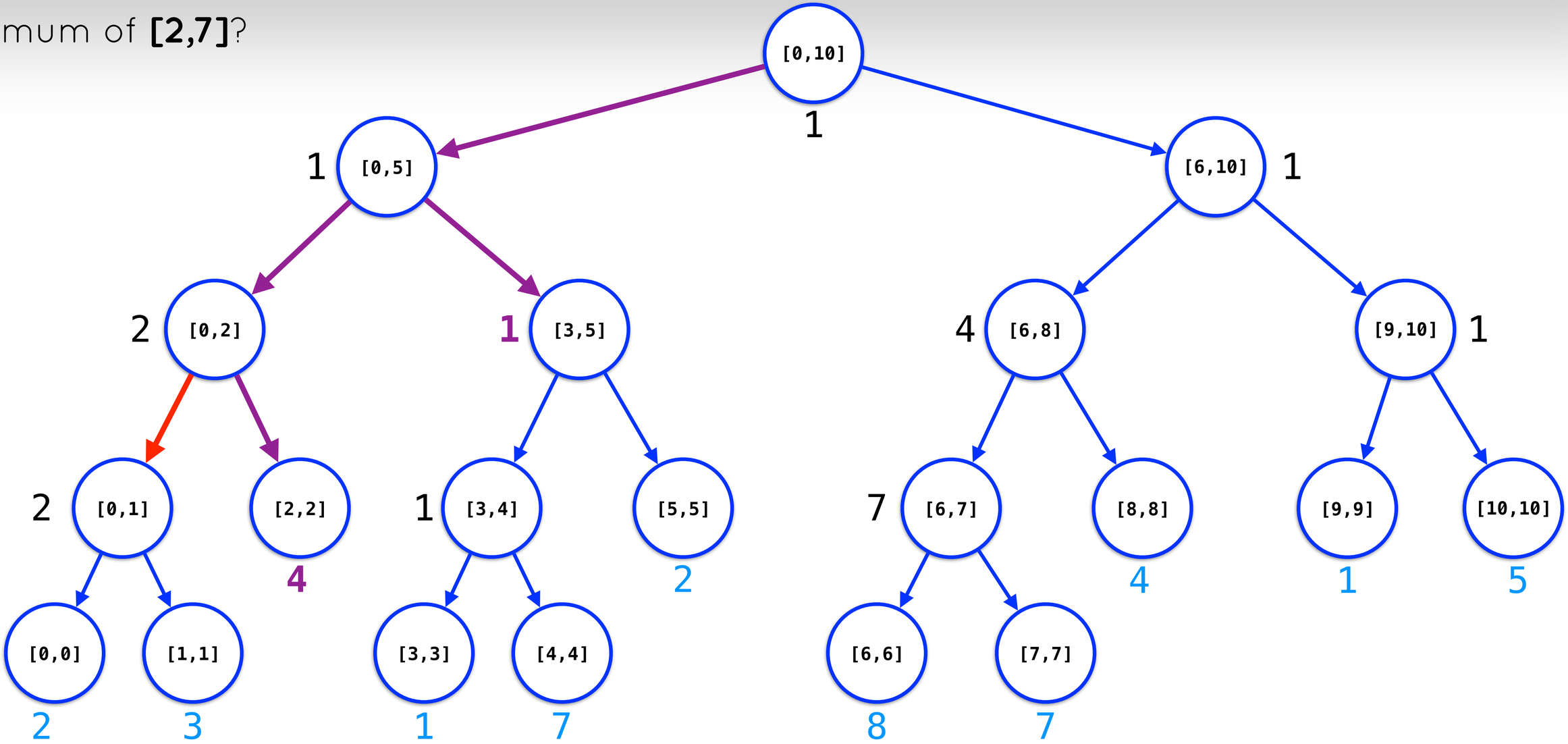
Range minimum queries

minimum of **[2,7]**?



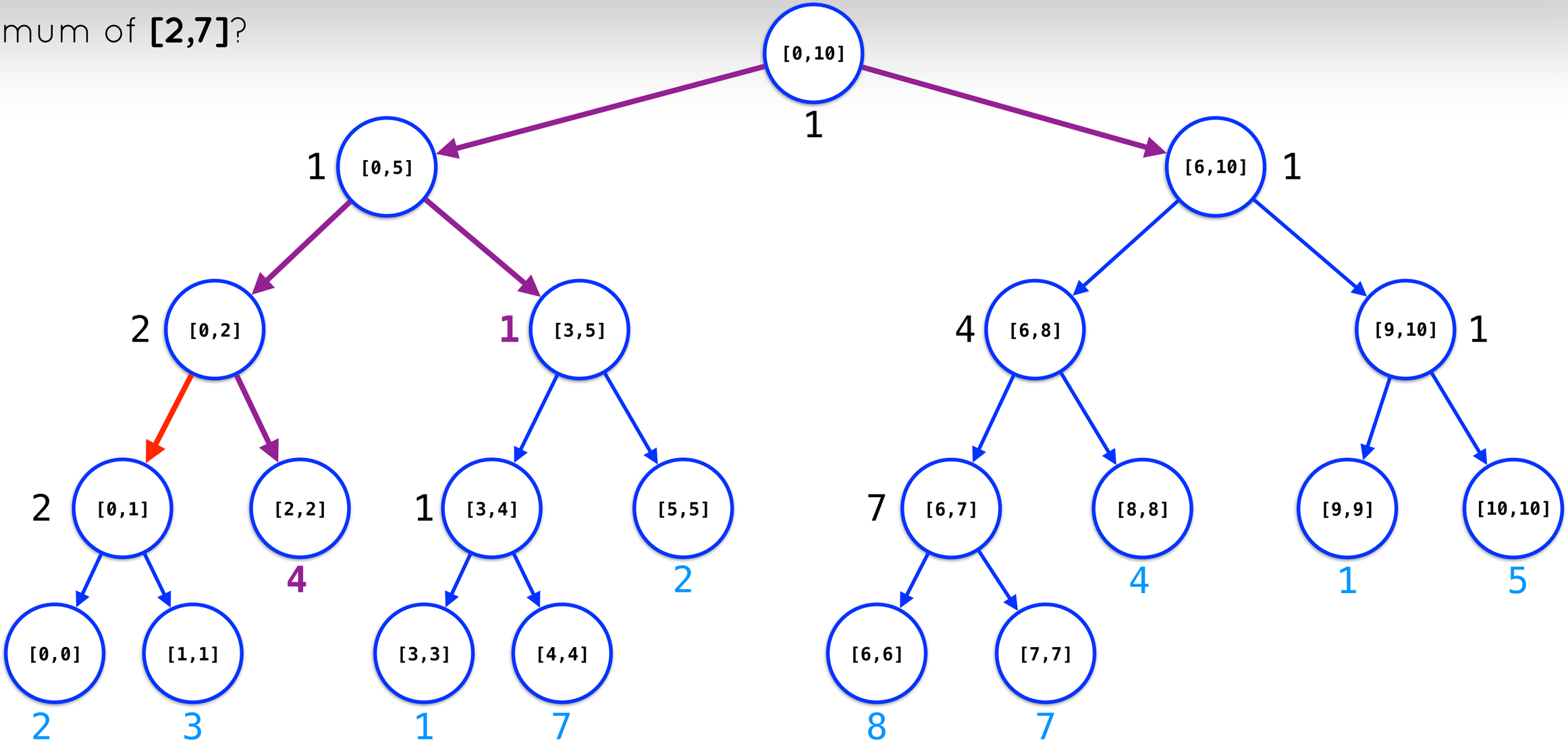
Range minimum queries

minimum of **[2,7]**?



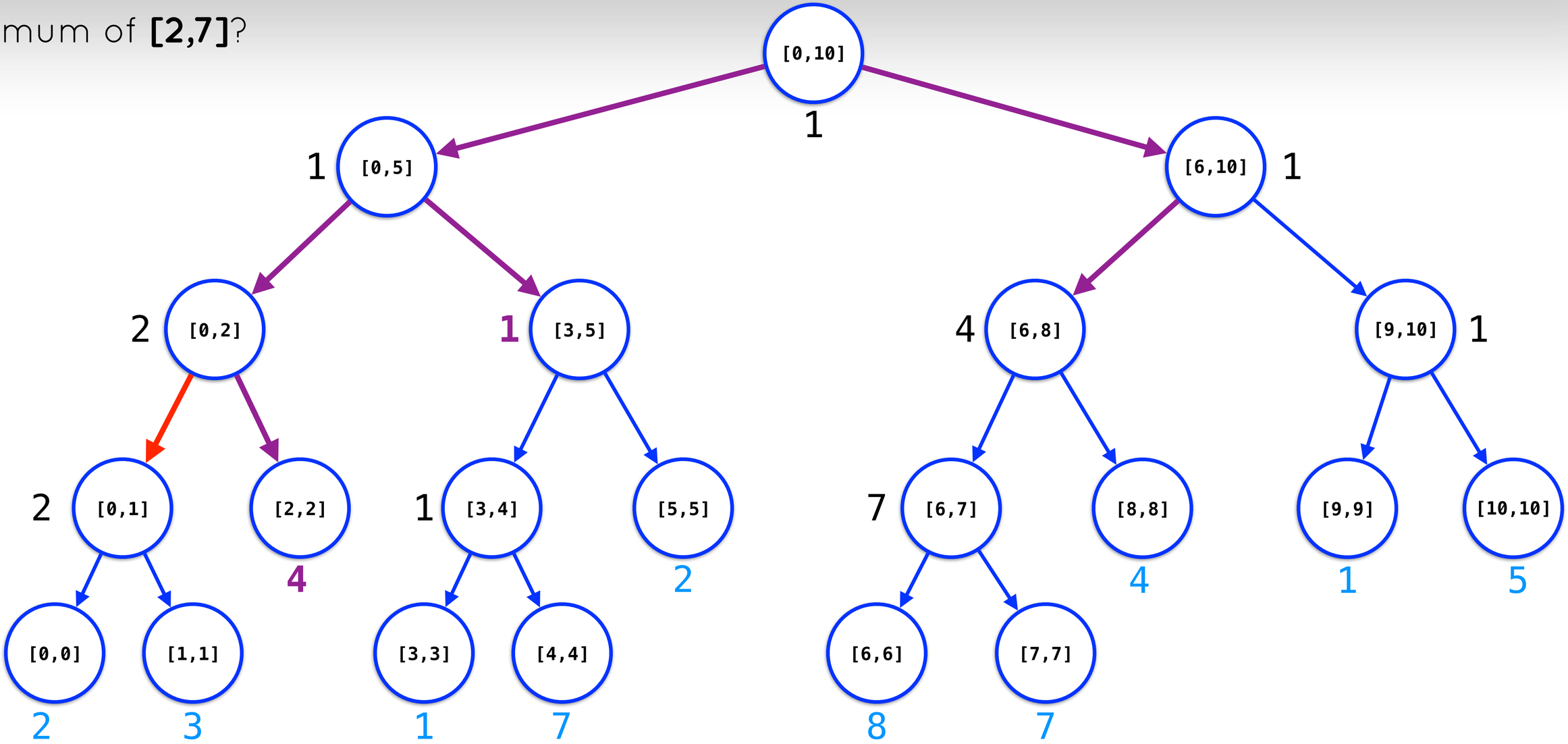
Range minimum queries

minimum of **[2,7]**?



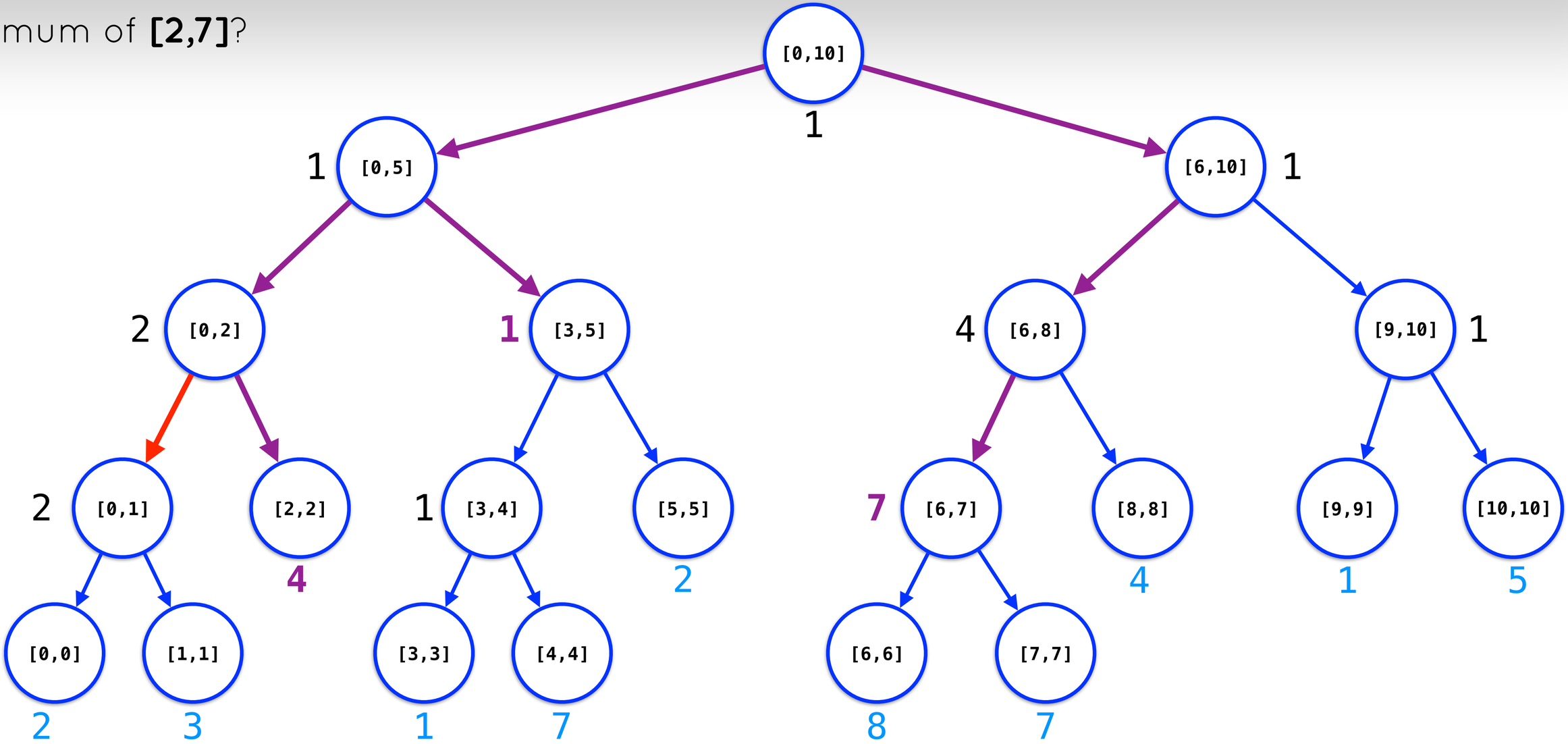
Range minimum queries

minimum of **[2,7]**?



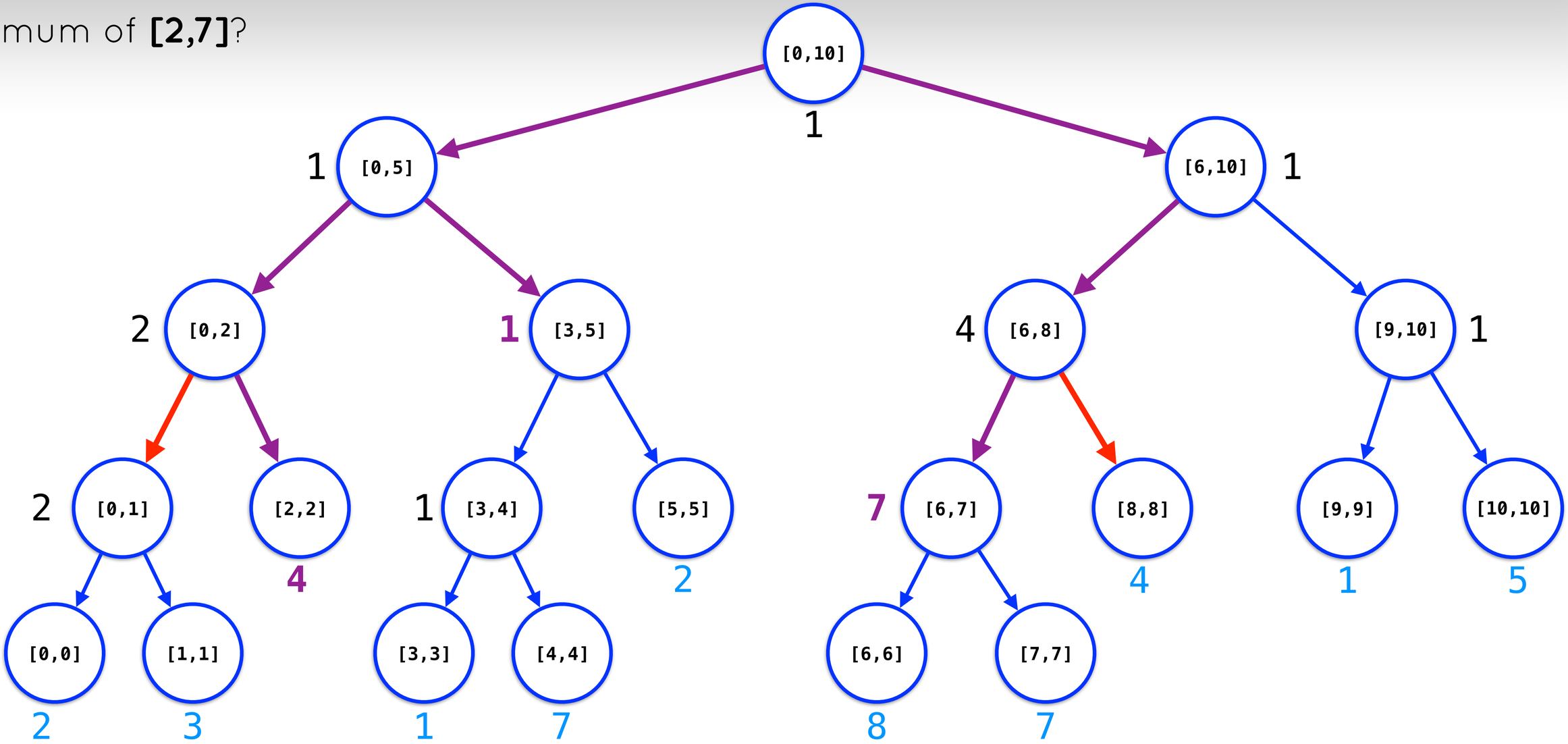
Range minimum queries

minimum of **[2,7]**?



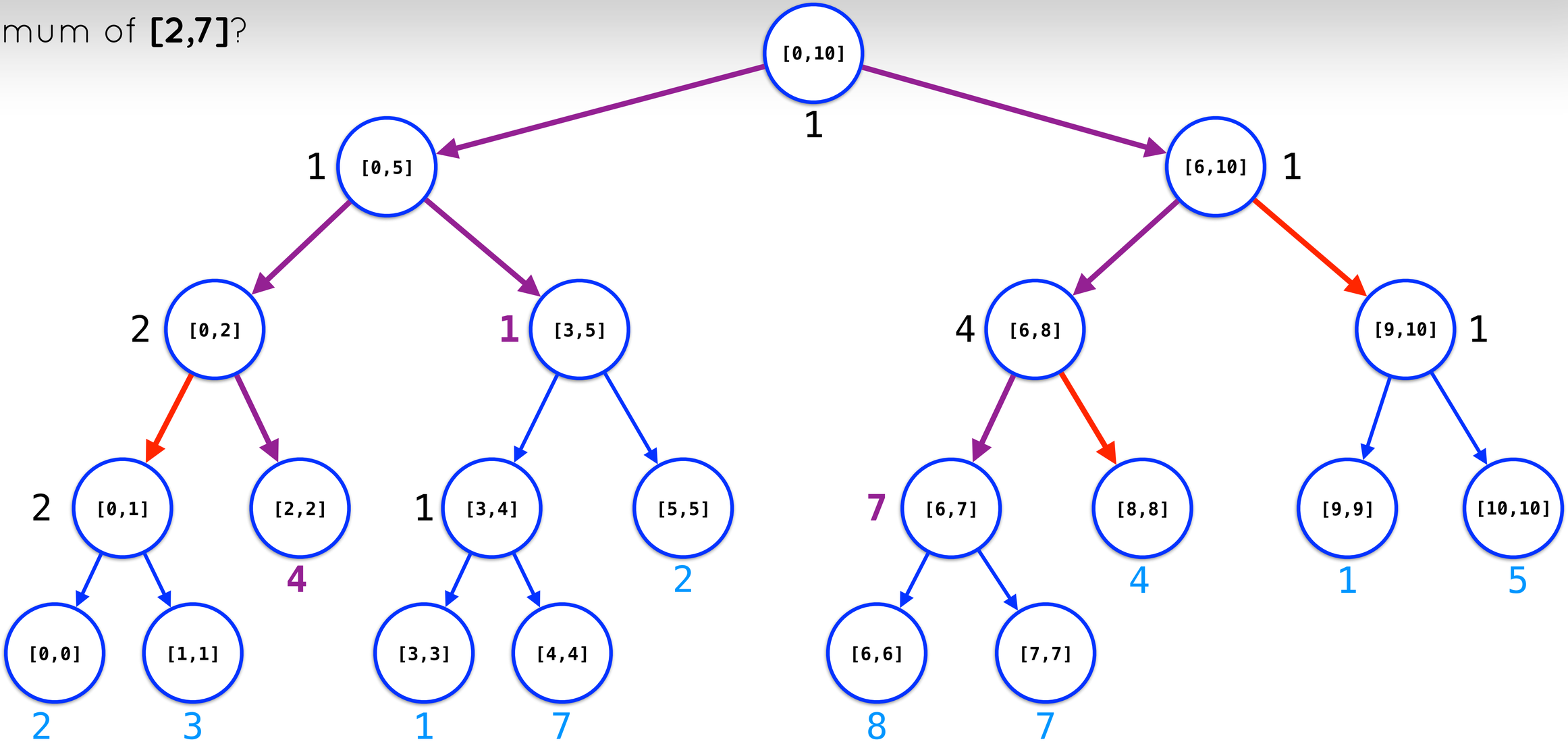
Range minimum queries

minimum of **[2,7]**?



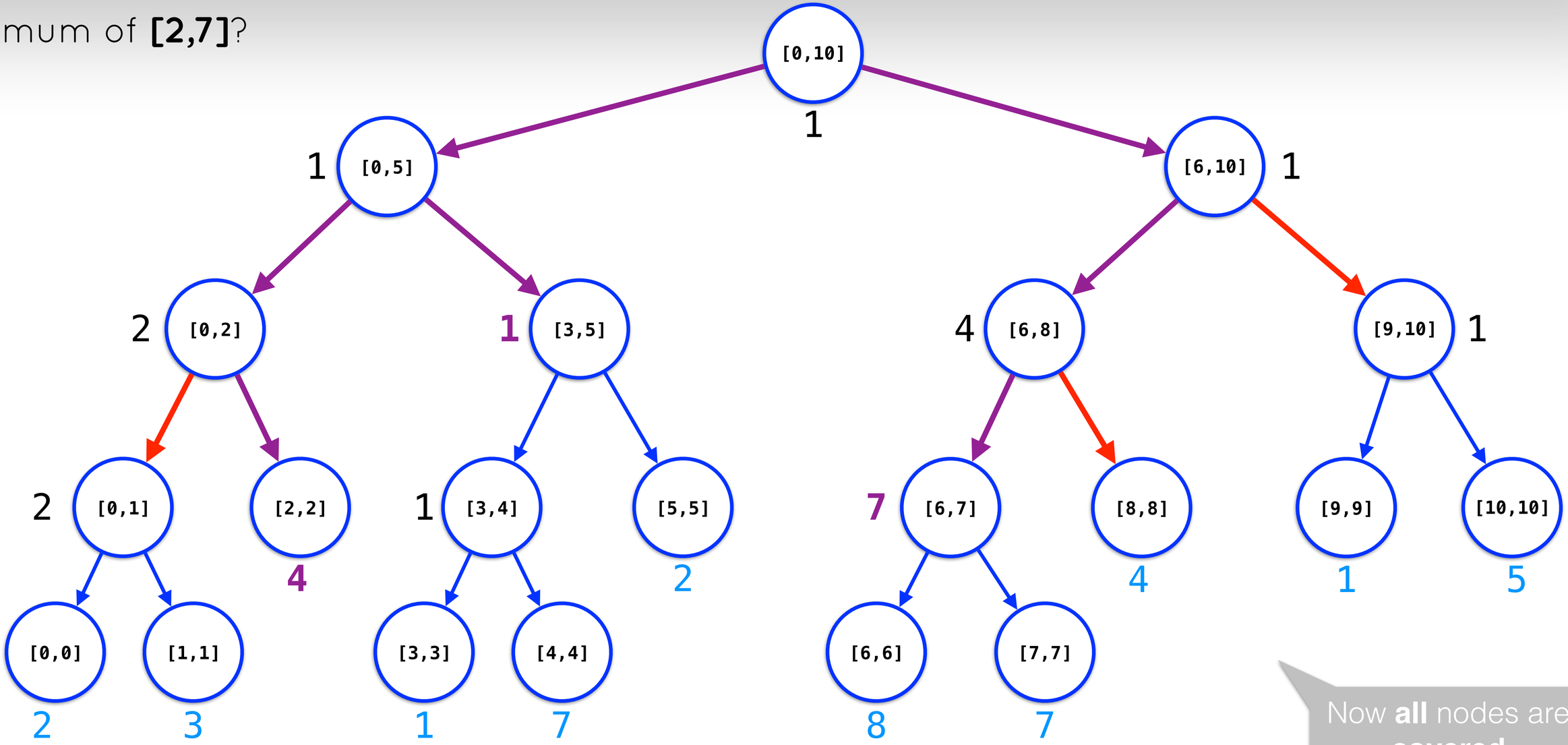
Range minimum queries

minimum of **[2,7]**?



Range minimum queries

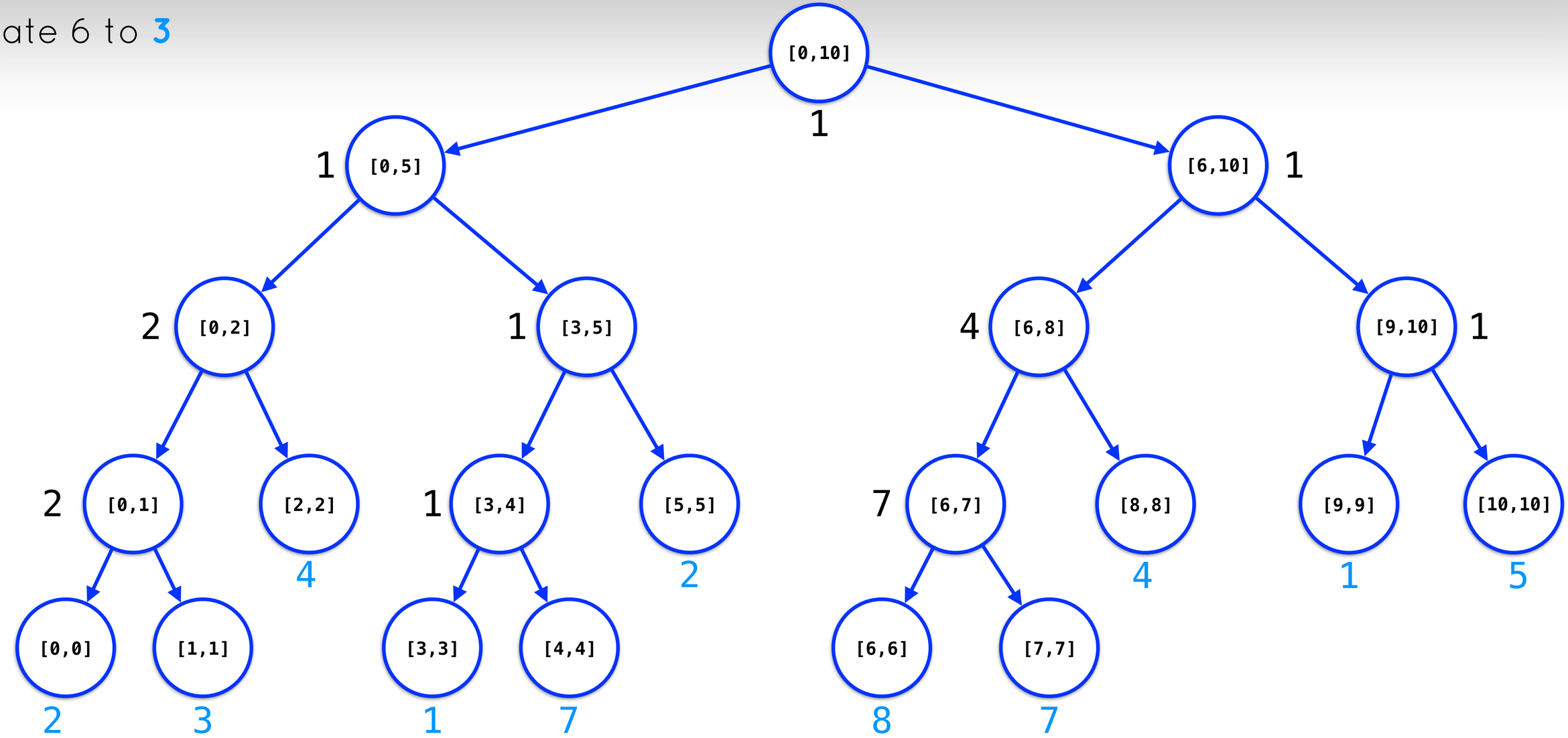
minimum of **[2,7]**?



Now **all** nodes are **covered**

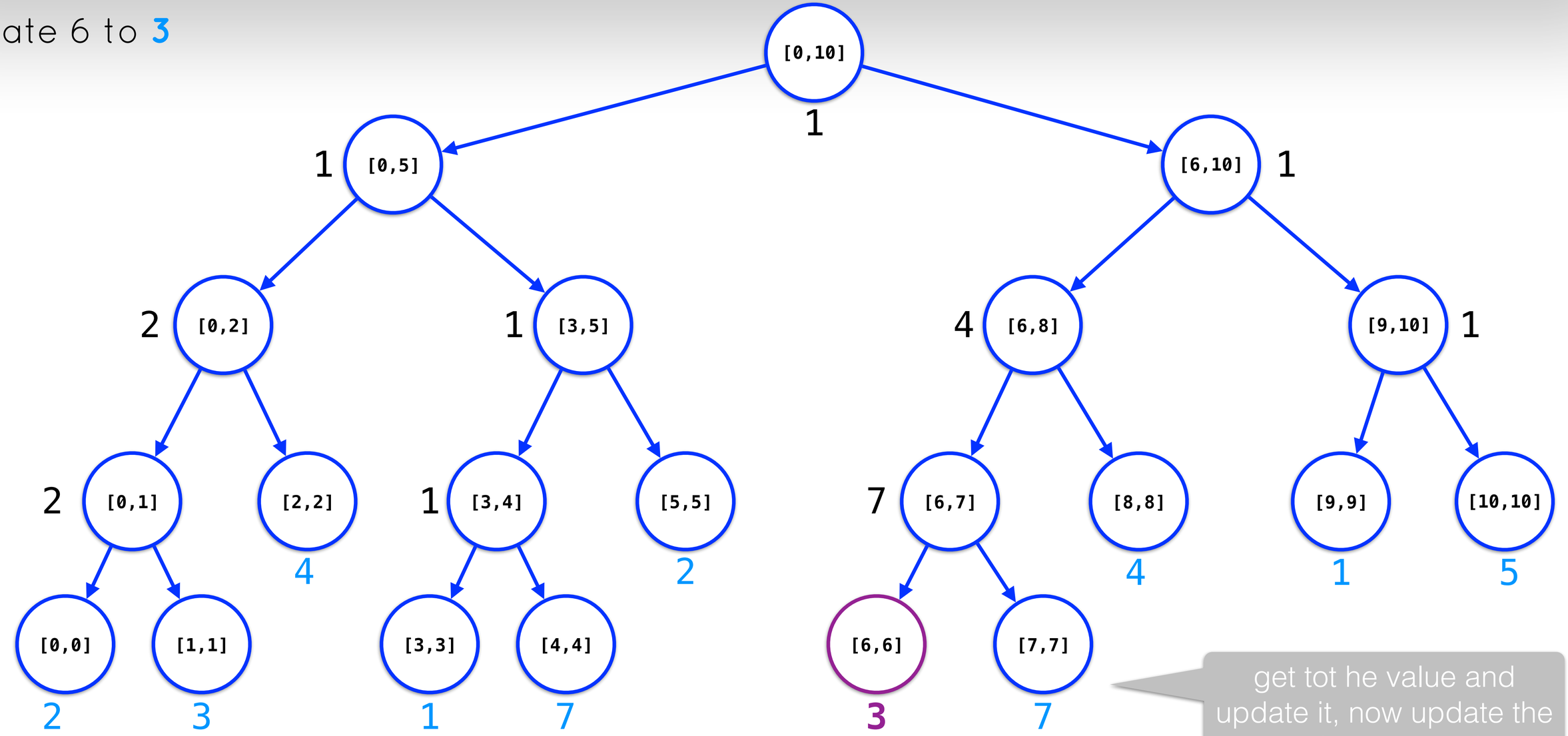
Updates

update 6 to 3



Updates

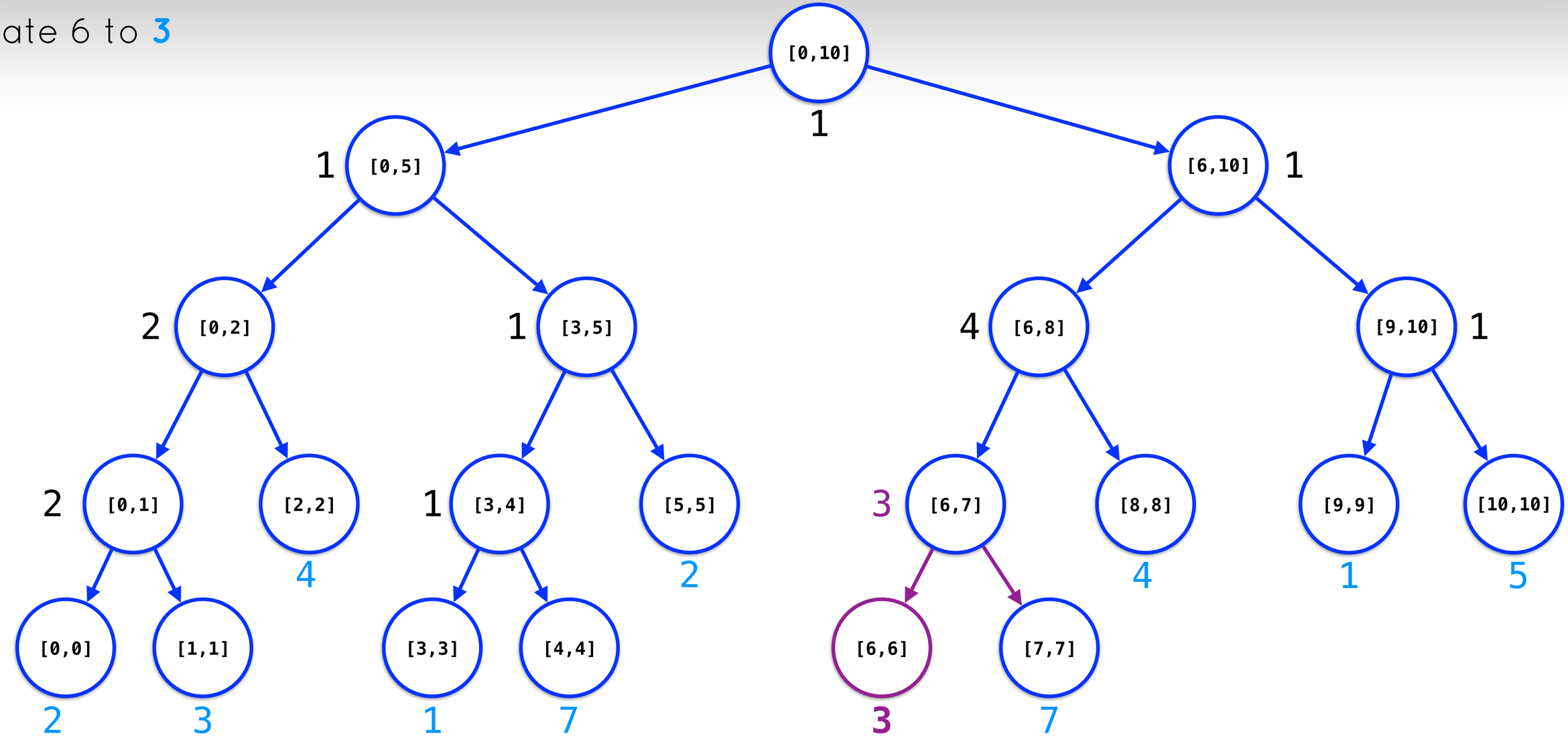
update 6 to 3



get tot he value and update it, now update the values upward

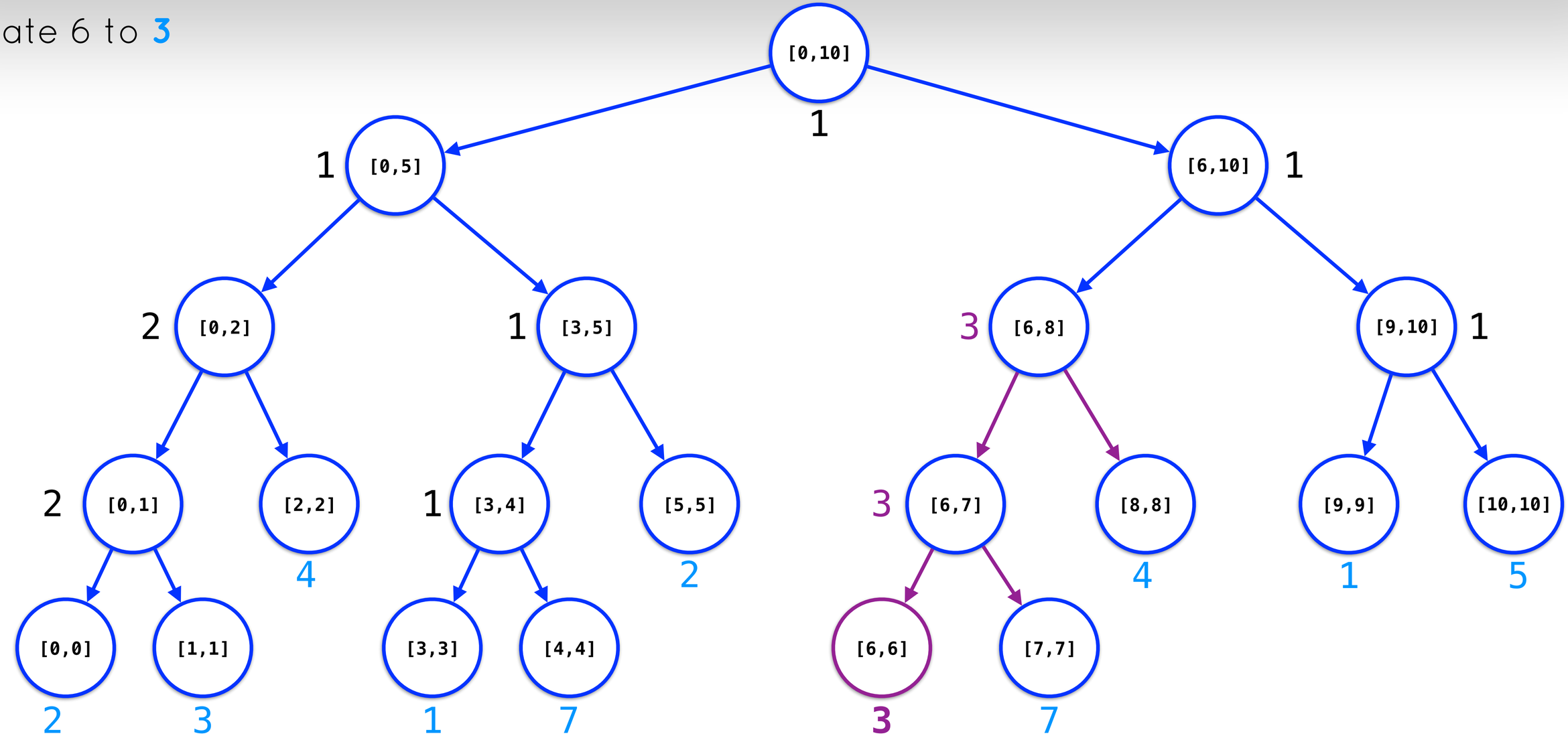
Updates

update 6 to **3**



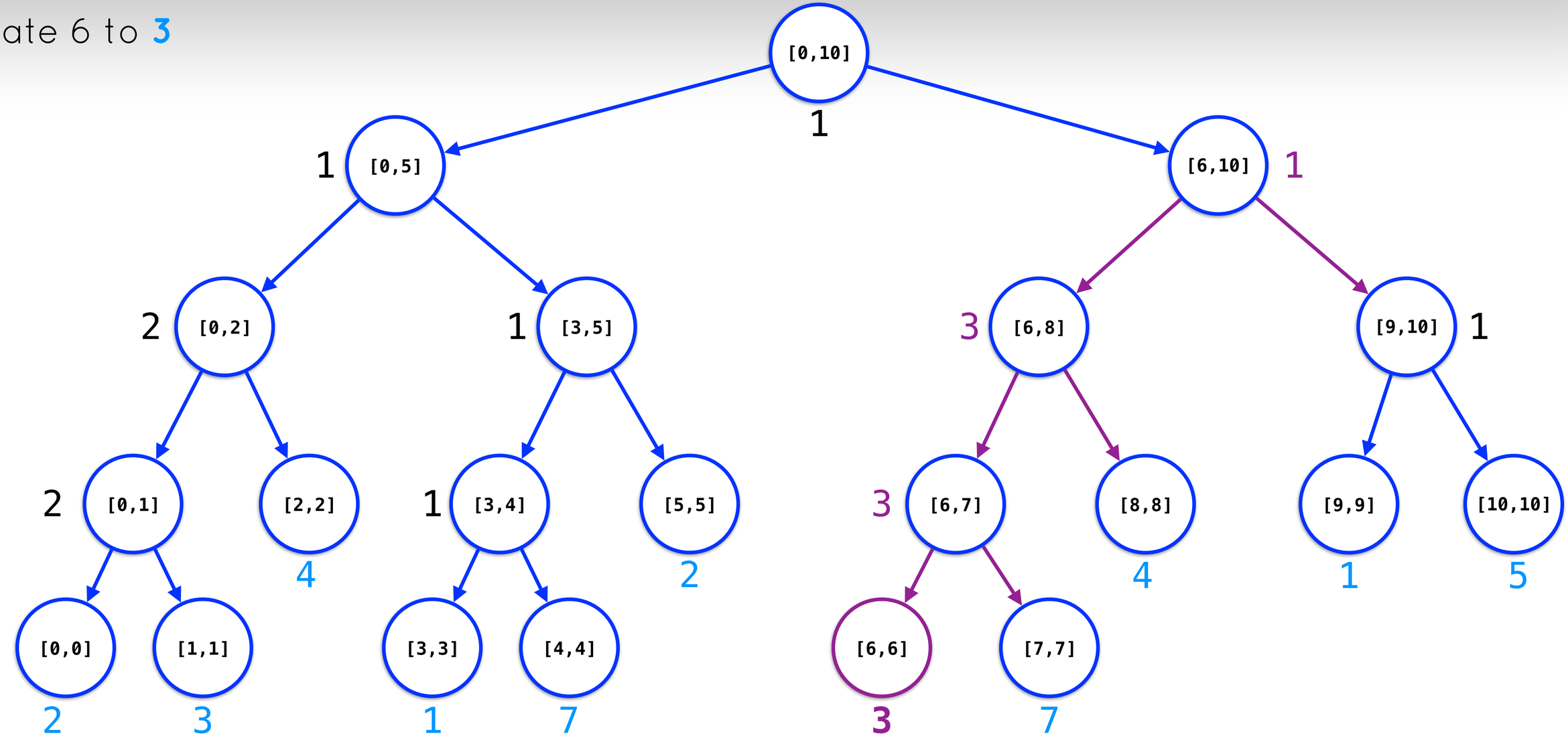
Updates

update 6 to **3**



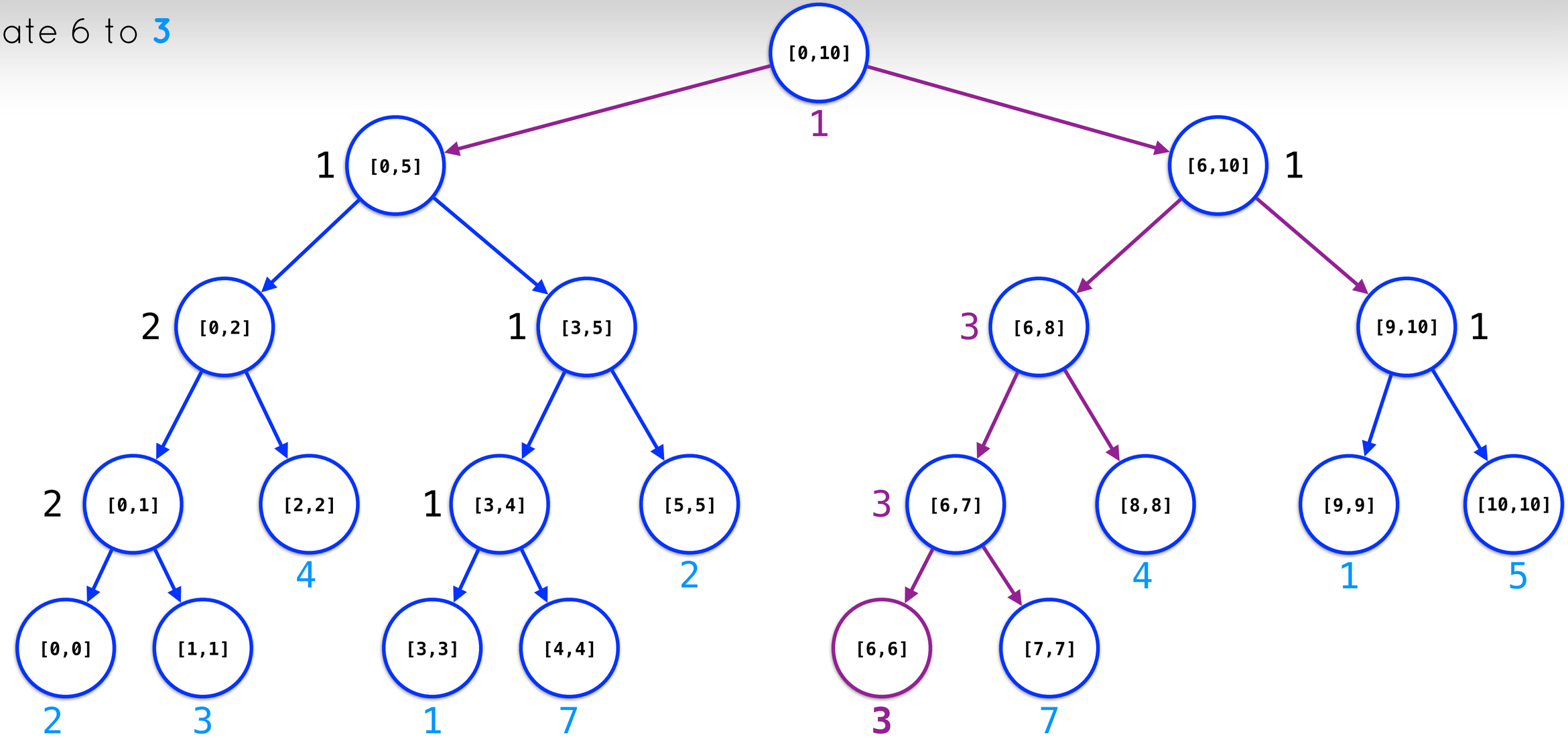
Updates

update 6 to **3**



Updates

update 6 to **3**



Build

```
int tree[4*array.length];
void build(int array[], int current, int left, int right) {
    if (left == right) {
        tree[current] = array[left];
    } else {
        int mid = (left + right) / 2;
        build(array, 2*current, left, mid);
        build(array, 2*current+1, mid+1, right);
        tree[current] = t[2*current] + t[2*current+1];
    }
}
```

Query

```
int tree[4*array.length];
int query(int current, int leftB, int rightB, int left, int right) {
    if (left > right) {
        return 0;
    if (left == leftB && right == rightB) {
        return tree[current];
    }
    int mid = (left + right) / 2;
    return query(2*current, leftB, mid, left, min(right, mid)) op
           query(2*current+1, mid+1, rightB, max(left, mid+1), right);
}
```

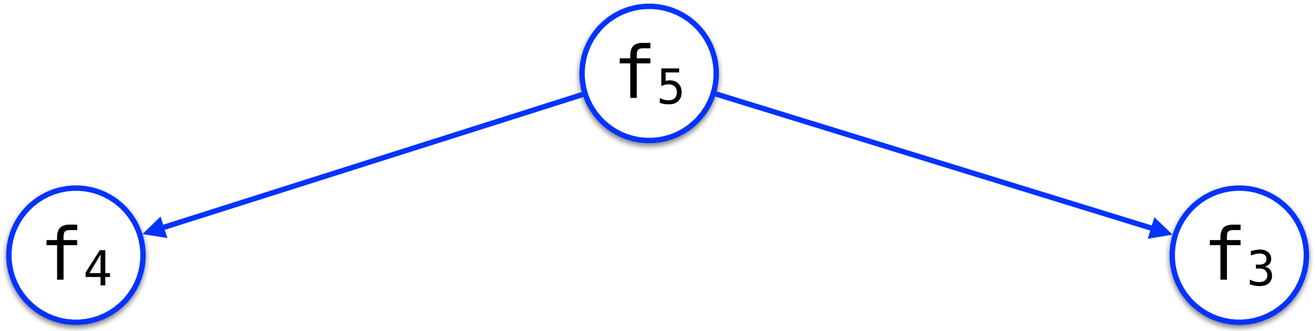
Update

```
int tree[4*array.length];
void update(int current, int left, int right, int pos, int new_val) {
    if (left == right) {
        tree[v] = new_val;
    } else {
        int mid = (left + right) / 2;
        if (pos <= mid)
            update(2*current, left, mid, pos, new_val);
        else
            update(2*current+1, mid+1, right, pos, new_val);
        tree[current] = tree[2*current] op tree[2*current+1];
    }
}
```

Fibonacci sequence

$$\text{Fib}(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ \text{Fib}(n-1) + \text{Fib}(n-2) & \text{otherwise.} \end{cases}$$

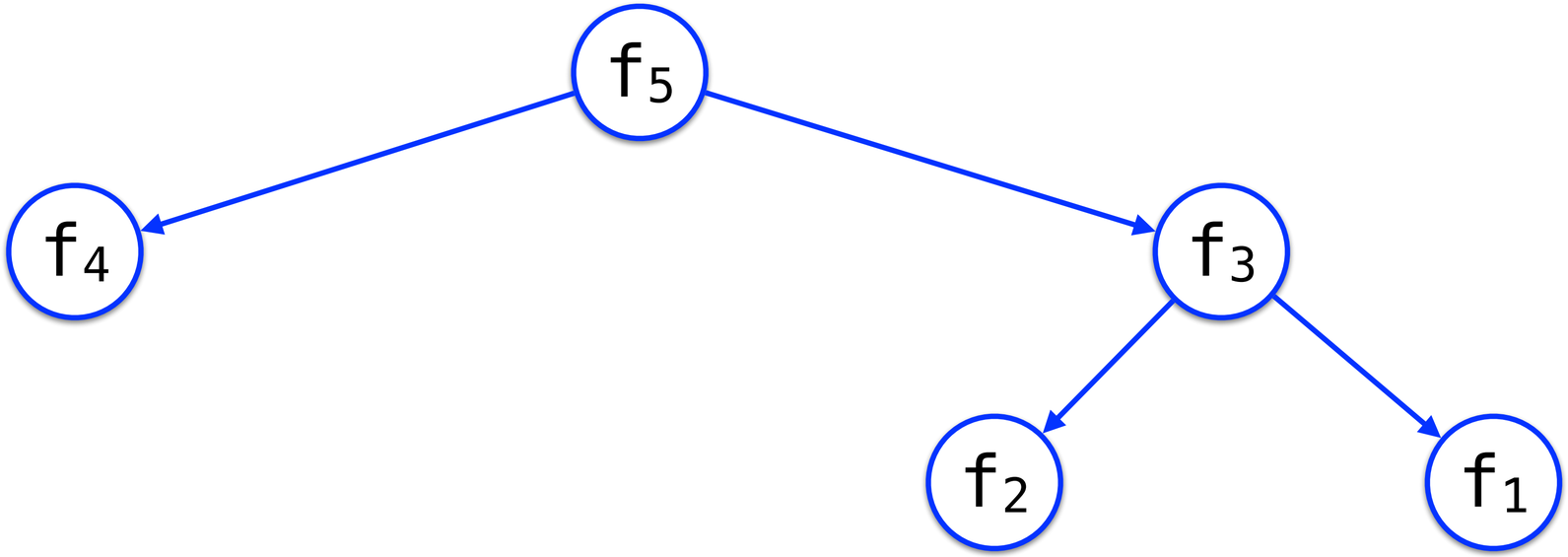
0	1	2	3	4	5
0	**1**	**0**	**0**	**0**	**0**



Fibonacci sequence

$$\text{Fib}(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ \text{Fib}(n-1) + \text{Fib}(n-2) & \text{otherwise.} \end{cases}$$

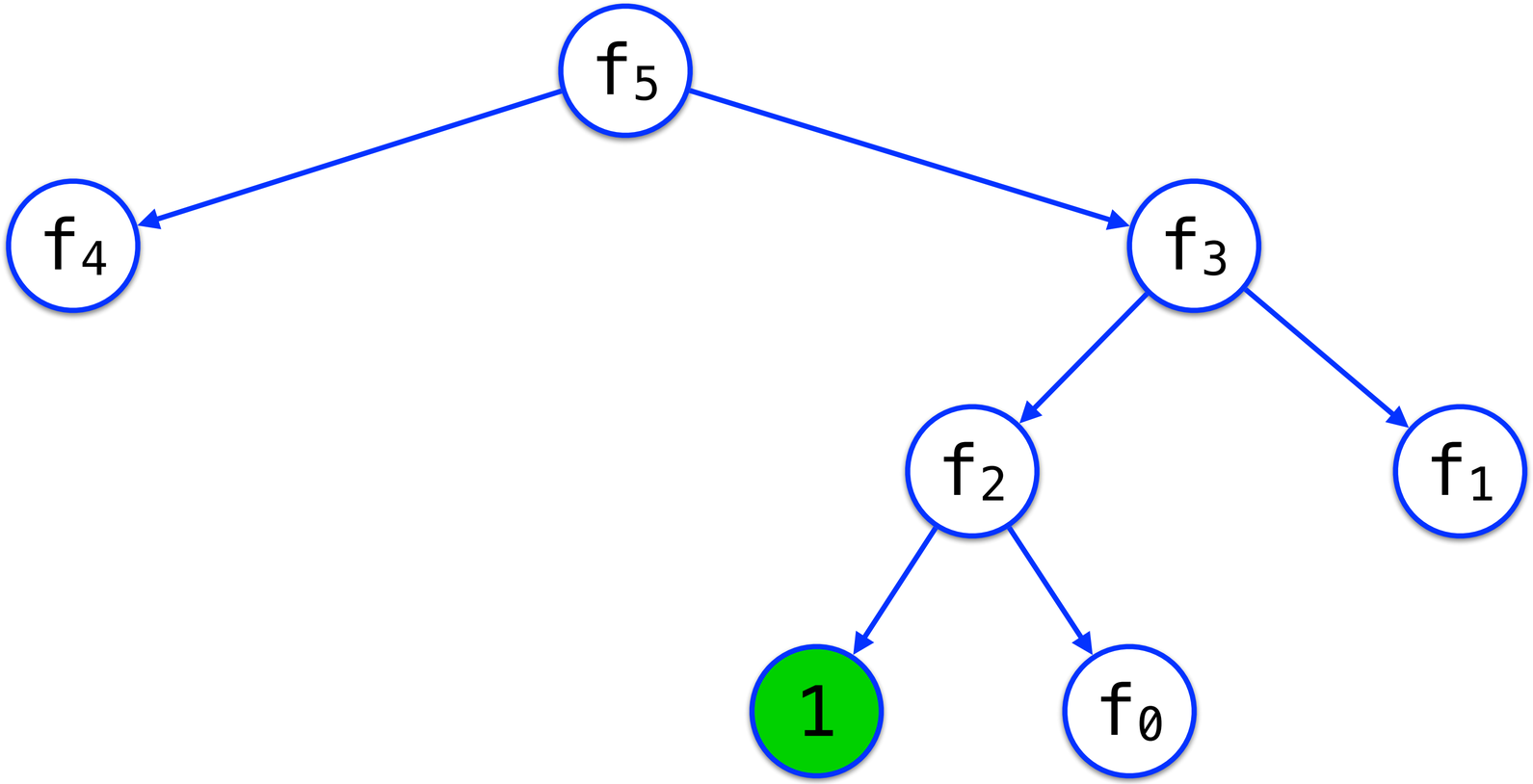
0	1	2	3	4	5
0	**1**	**0**	**0**	**0**	**0**



Fibonacci sequence

$$\text{Fib}(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ \text{Fib}(n-1) + \text{Fib}(n-2) & \text{otherwise.} \end{cases}$$

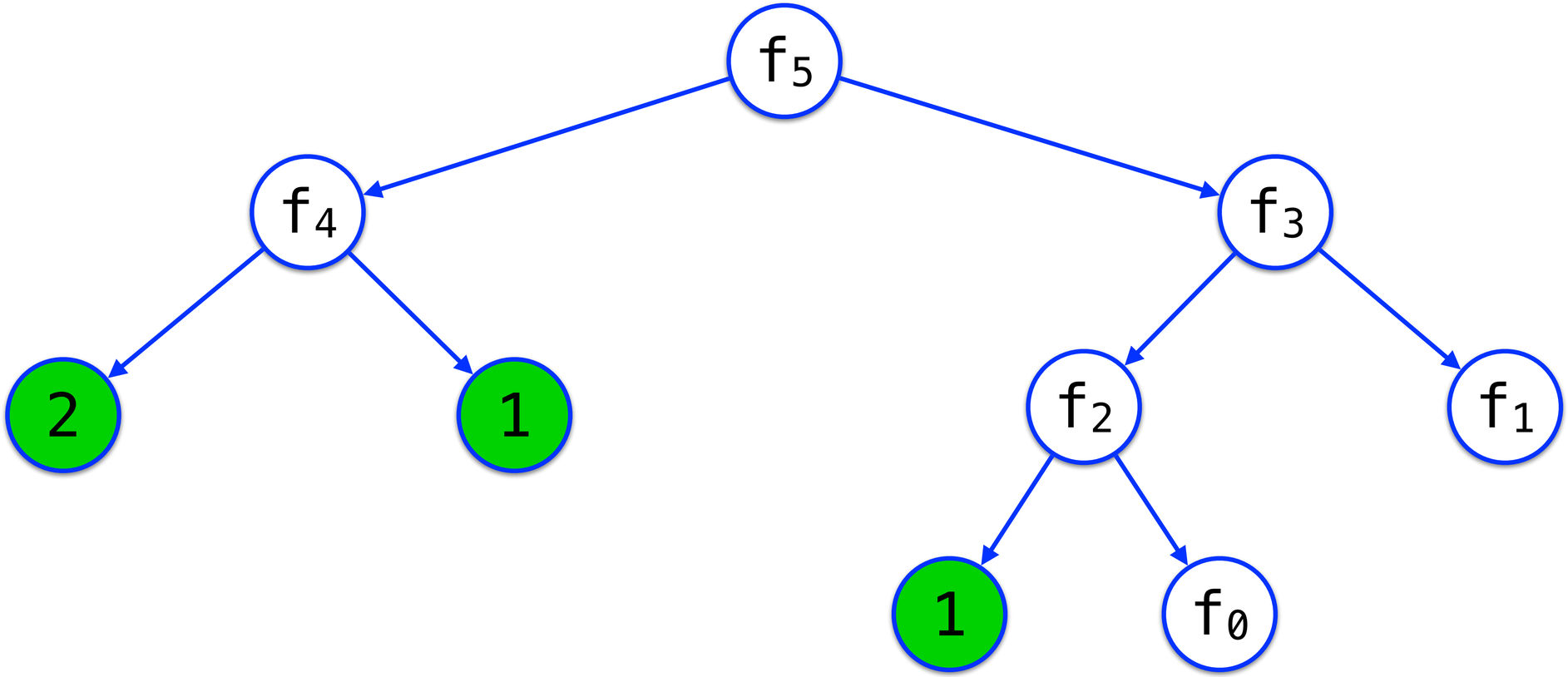
0	1	2	3	4	5
0	**1**	**1**	**2**	**0**	**0**



Fibonacci sequence

$$\mathrm{Fib}(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ \mathrm{Fib}(n-1) + \mathrm{Fib}(n-2) & \text{otherwise.} \end{cases}$$

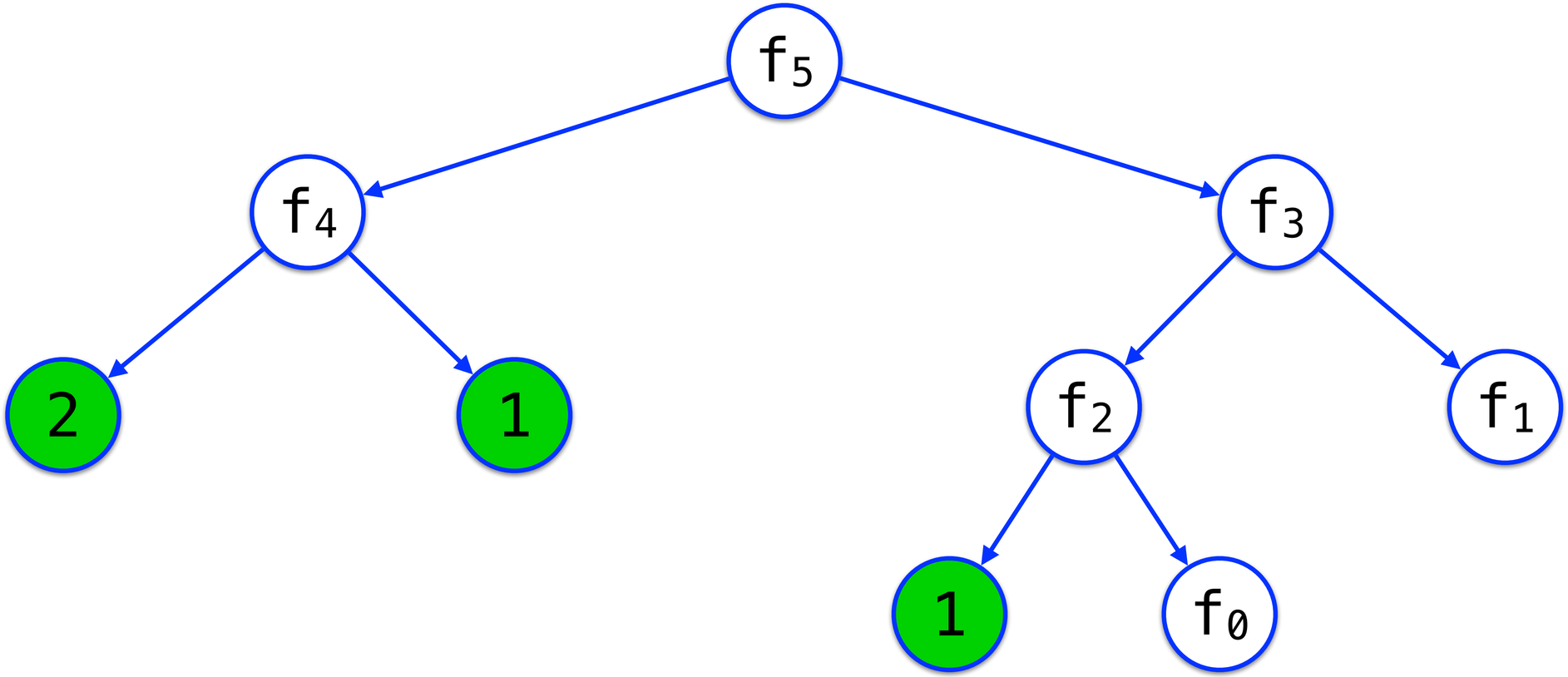
0	1	2	3	4	5
0	**1**	**1**	**2**	**3**	**0**



Fibonacci sequence

$$\mathrm{Fib}(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ \mathrm{Fib}(n-1) + \mathrm{Fib}(n-2) & \text{otherwise.} \end{cases}$$

0	1	2	3	4	5
0	**1**	**1**	**2**	**3**	**5**



Segment tree applications

- Pre-calculate partial results
- Aggregate results
- Queries about operations (min, max, sum, etc) over ranges
- Queries with intermediate updates on the queried data

Segment tree applications

- Pre-calculate partial results
- Aggregate results
- Queries about operations (min, max, sum, etc) over ranges
- Queries with intermediate updates on the queried data

commutative operations