Data Structure and Algorithms Design

Vincent Chau （周）

2021.11.17

Summary

- Divide and Conquer √
- Dynamic programming √
- Greedy Algorithm √
- Binary Search Tree
- Branch and Bound

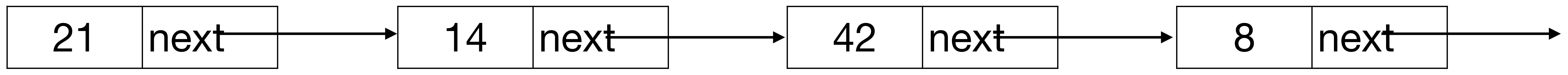
Tree structure

Linear Data Structure

- Array

21	14	42	8	30	10	11	1

- Linked list

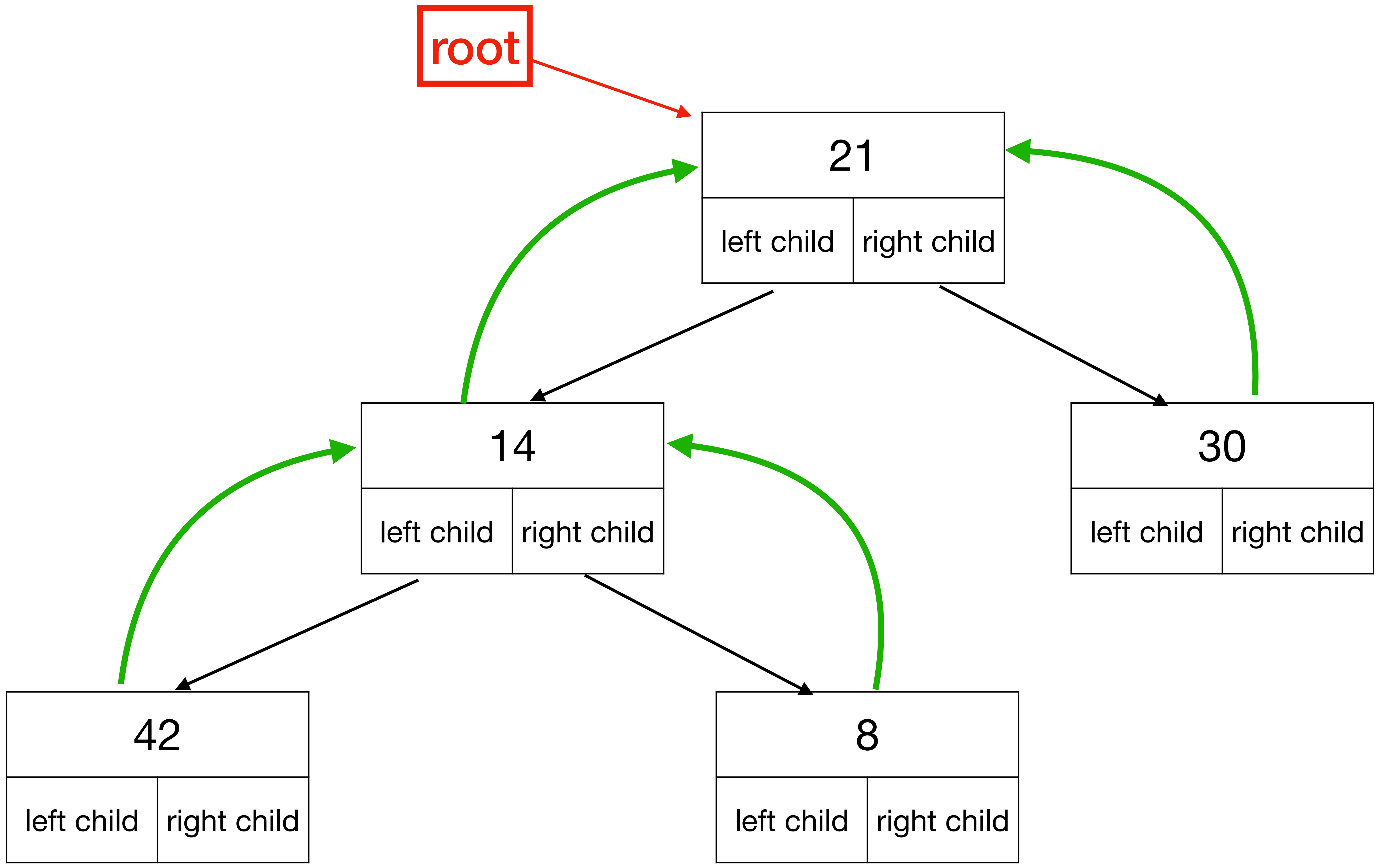


Binary Tree

Concept

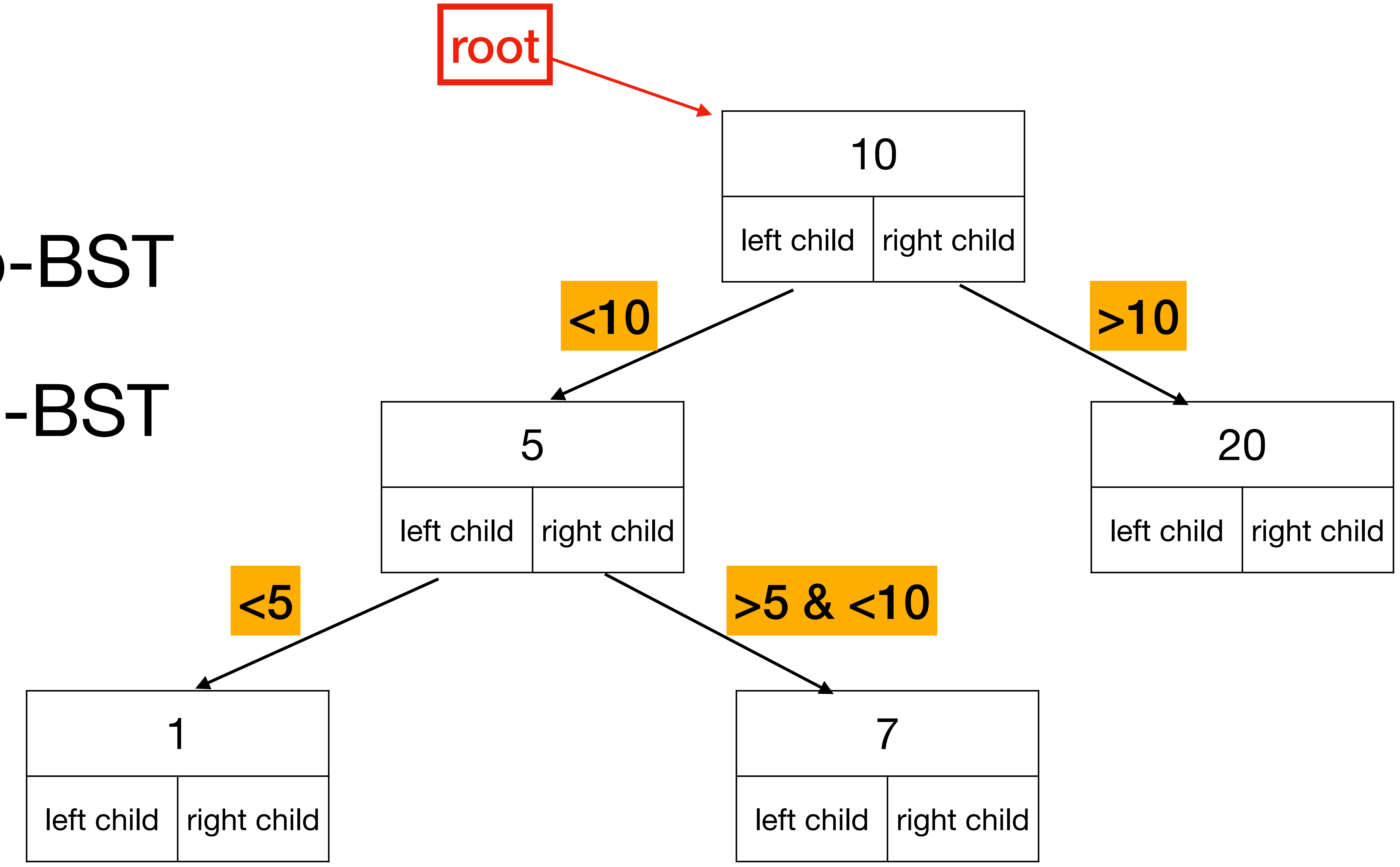
- Each node has at most two children

```
Structure node{
  key: int;
  left: *node;
  right: *node;
  parent: *node;
}
```



Binary Search Tree
Properties

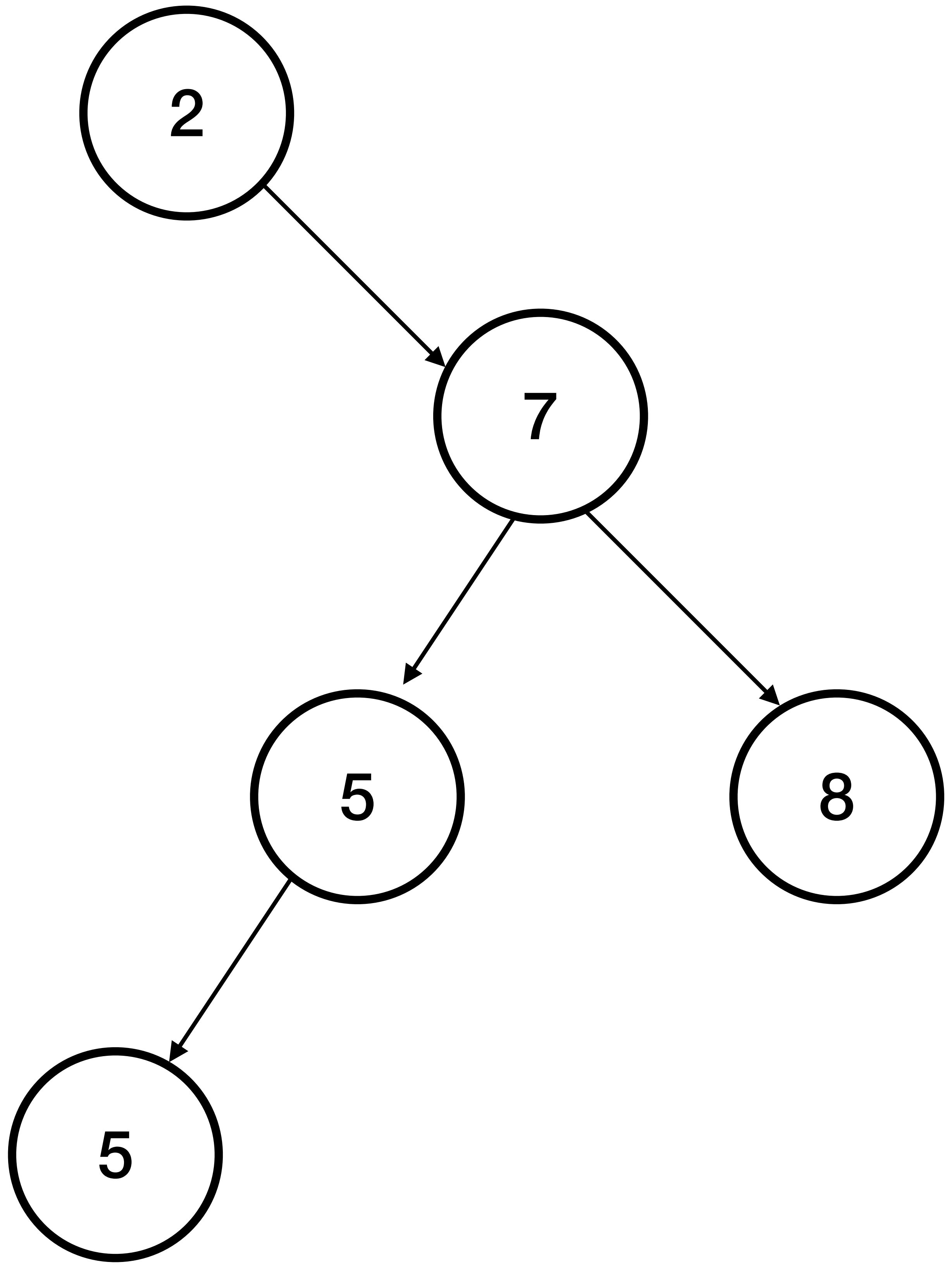
- All numbers are:
  - smaller than `key` at the left sub-BST
  - larger than `key` at the right sub-BST



Print numbers in increasing order

Use infix notation

```
Infix(tree):
if tree≠NULL
  | Infix(tree.left);
  | print(tree.key);
  | Infix(tree.right);
end if
```

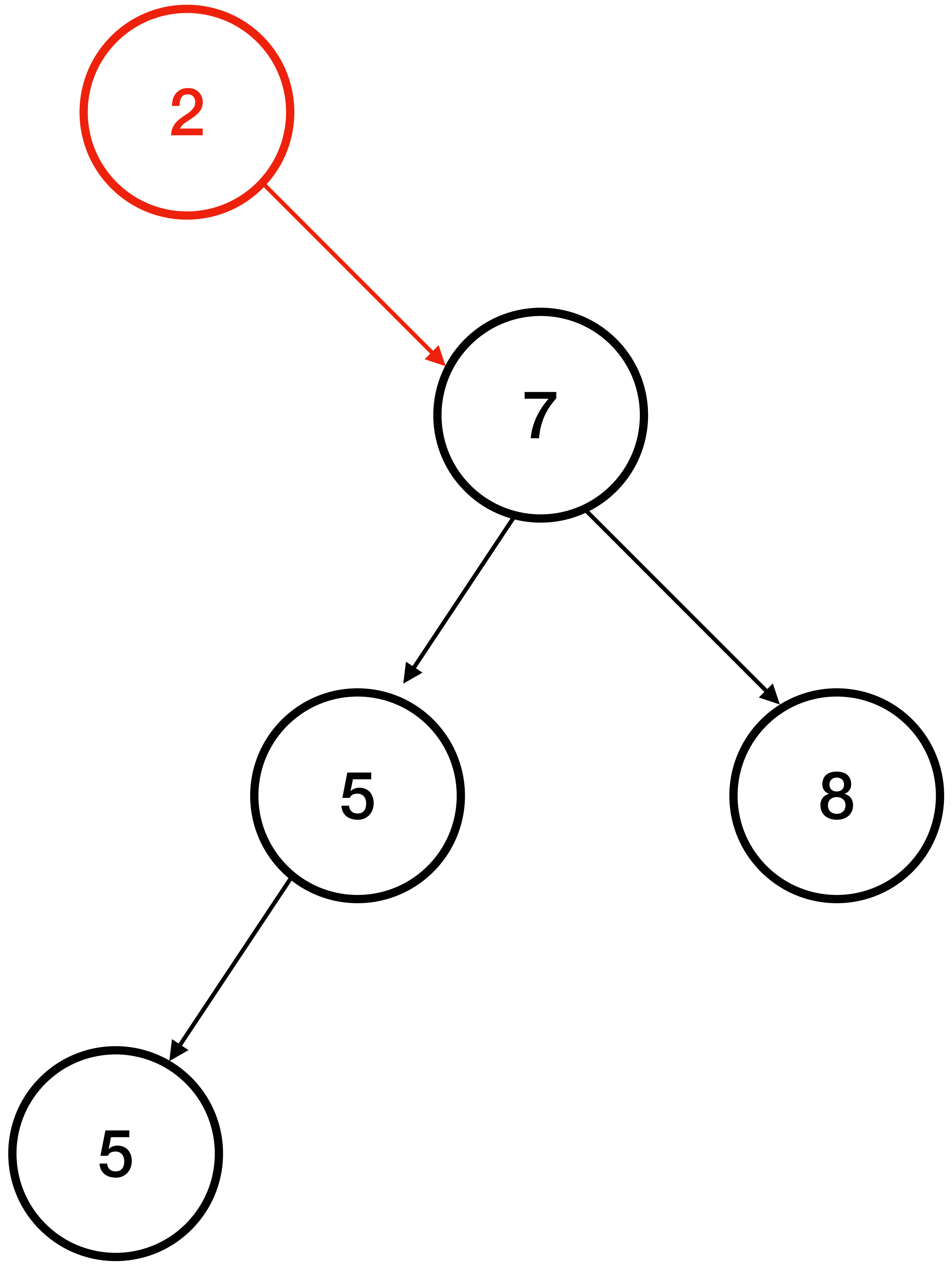


Print numbers in increasing order

Use infix notation

```
Infix(2)
    Infix(NULL)
    print(2)
    Infix(7)
```

```
Infix(tree):
if tree≠NULL
    Infix(tree.left);
    print(tree.key);
    Infix(tree.right);
end if
```

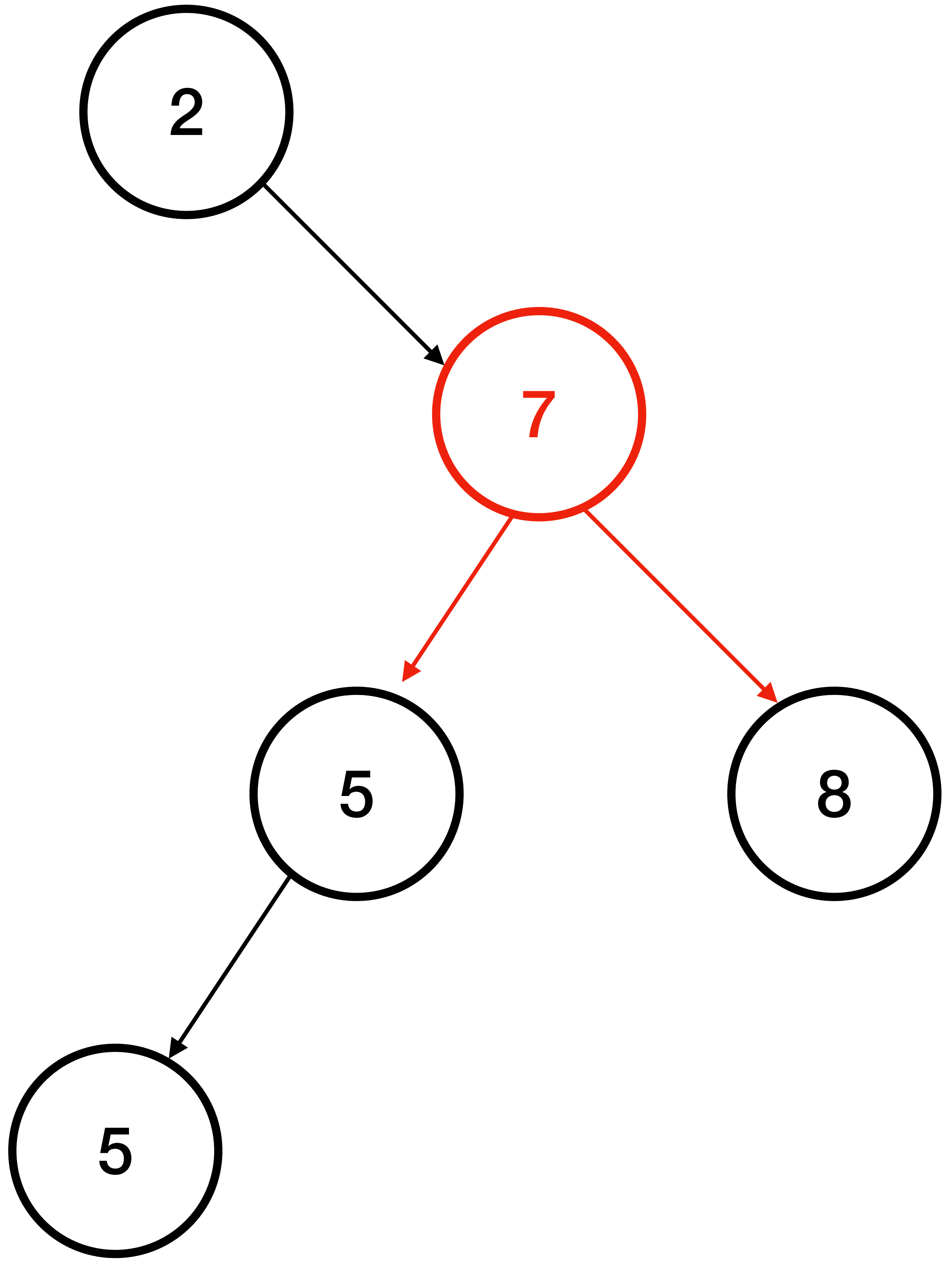


Print numbers in increasing order

Use infix notation

```
Infix(2)
    Infix(NULL)
    print(2)
    Infix(7)
        Infix(5)
        print(7)
        Infix(8)
```

```
Infix(tree):
if tree≠NULL
    Infix(tree.left);
    print(tree.key);
    Infix(tree.right);
end if
```

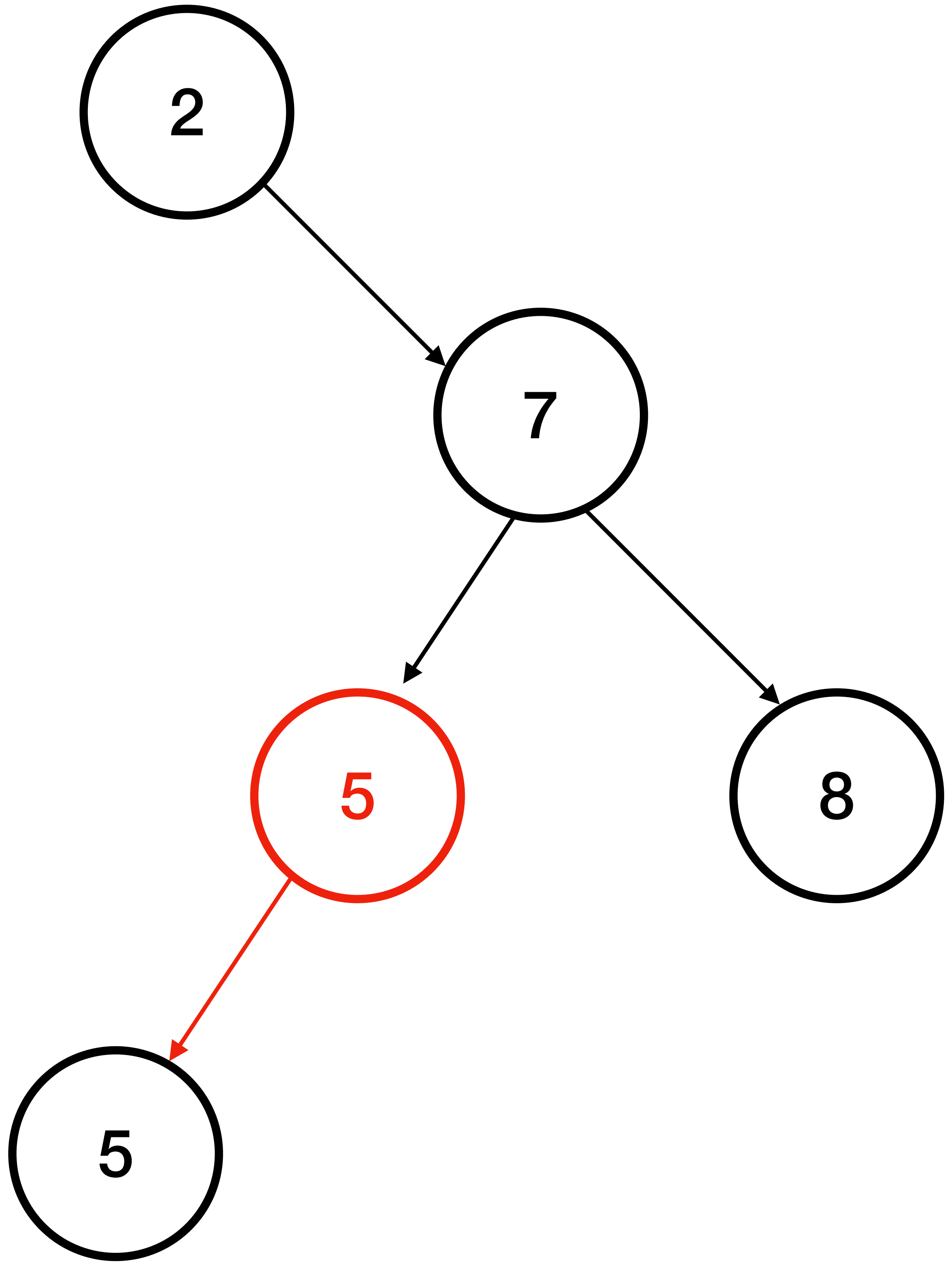


Print numbers in increasing order

Use infix notation

```
Infix(2)
    Infix(NULL)
    print(2)
    Infix(7)
        Infix(5)
            Infix(5)
            print(5)
            Infix(NULL)
        print(7)
        Infix(8)
```

```
Infix(tree):
if tree≠NULL
    Infix(tree.left);
    print(tree.key);
    Infix(tree.right);
end if
```

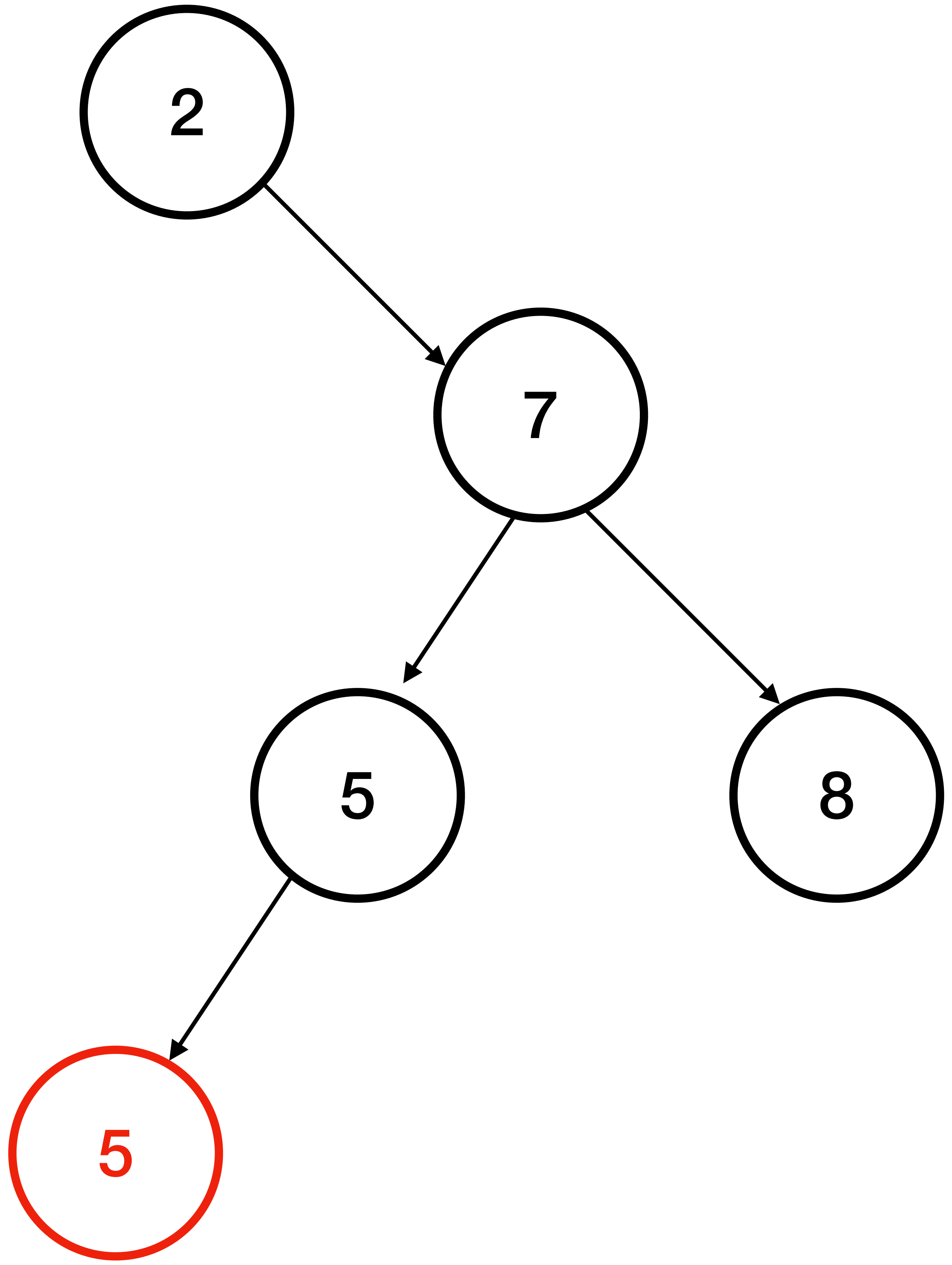


Print numbers in increasing order

Use infix notation

```
Infix(2)
    Infix(NULL)
    print(2)
    Infix(7)
        Infix(5)
            Infix(5)
                Infix(NULL)
                print(5)
                Infix(NULL)
            print(5)
            Infix(NULL)
        print(7)
        Infix(8)
```

```
Infix(tree):
if tree≠NULL
    Infix(tree.left);
    print(tree.key);
    Infix(tree.right);
end if
```

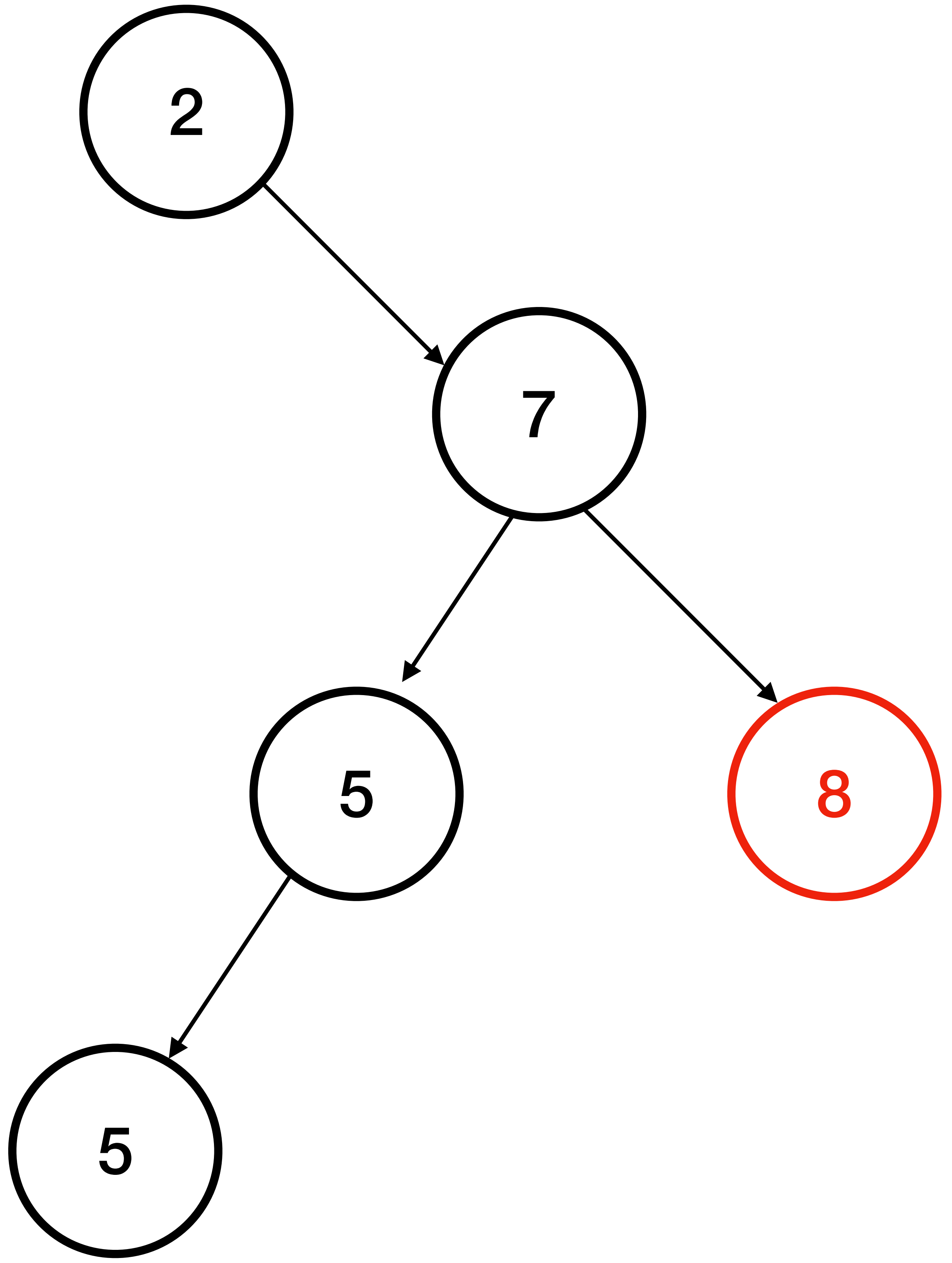


Print numbers in increasing order

Use infix notation

```
Infix(2)
    Infix(NULL)
    print(2)
    Infix(7)
        Infix(5)
            Infix(5)
                Infix(NULL)
                print(5)
                Infix(NULL)
            print(5)
            Infix(NULL)
        print(7)
        Infix(8)
            Infix(NULL)
            print(8)
            Infix(NULL)
```

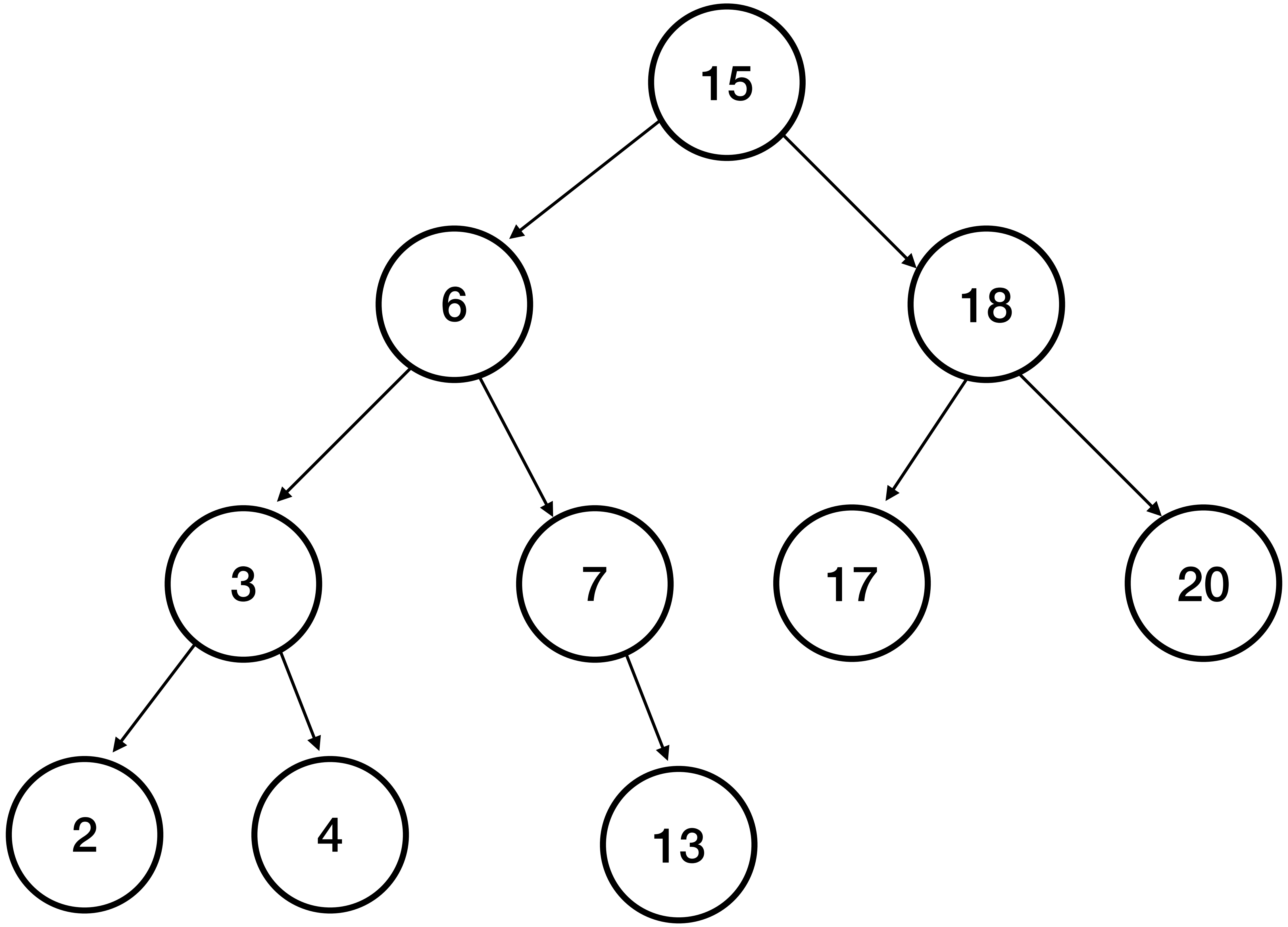
```
Infix(tree):
if tree≠NULL
    Infix(tree.left);
    print(tree.key);
    Infix(tree.right);
end if
```



Requests in BST

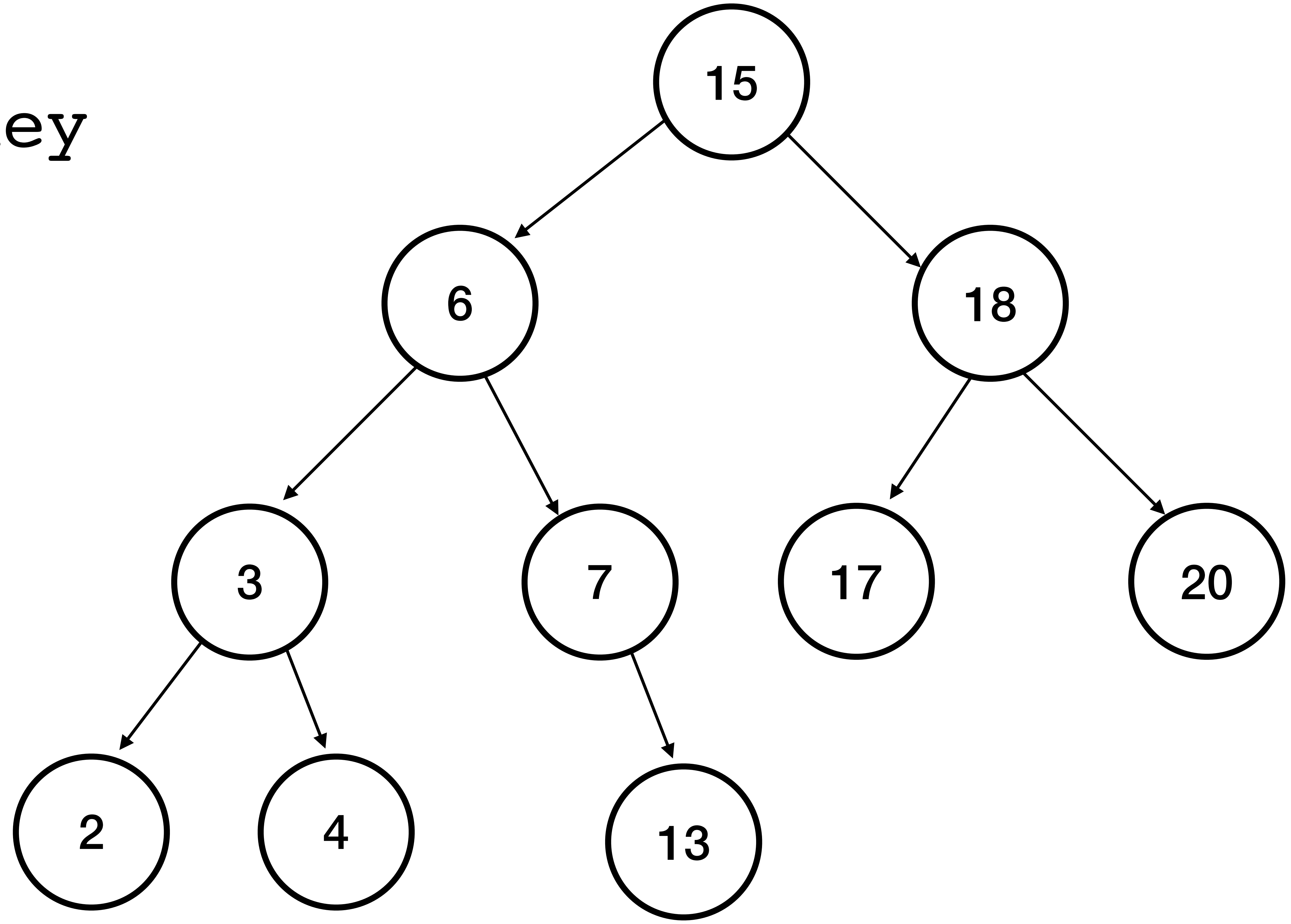
Search (recursive)

```
search(tree,k):
if tree = NULL or k = tree.key
|  return tree
end if
if k < tree.key
|  search(tree.left,k);
| else
|  search(tree.right,k);
end if
```



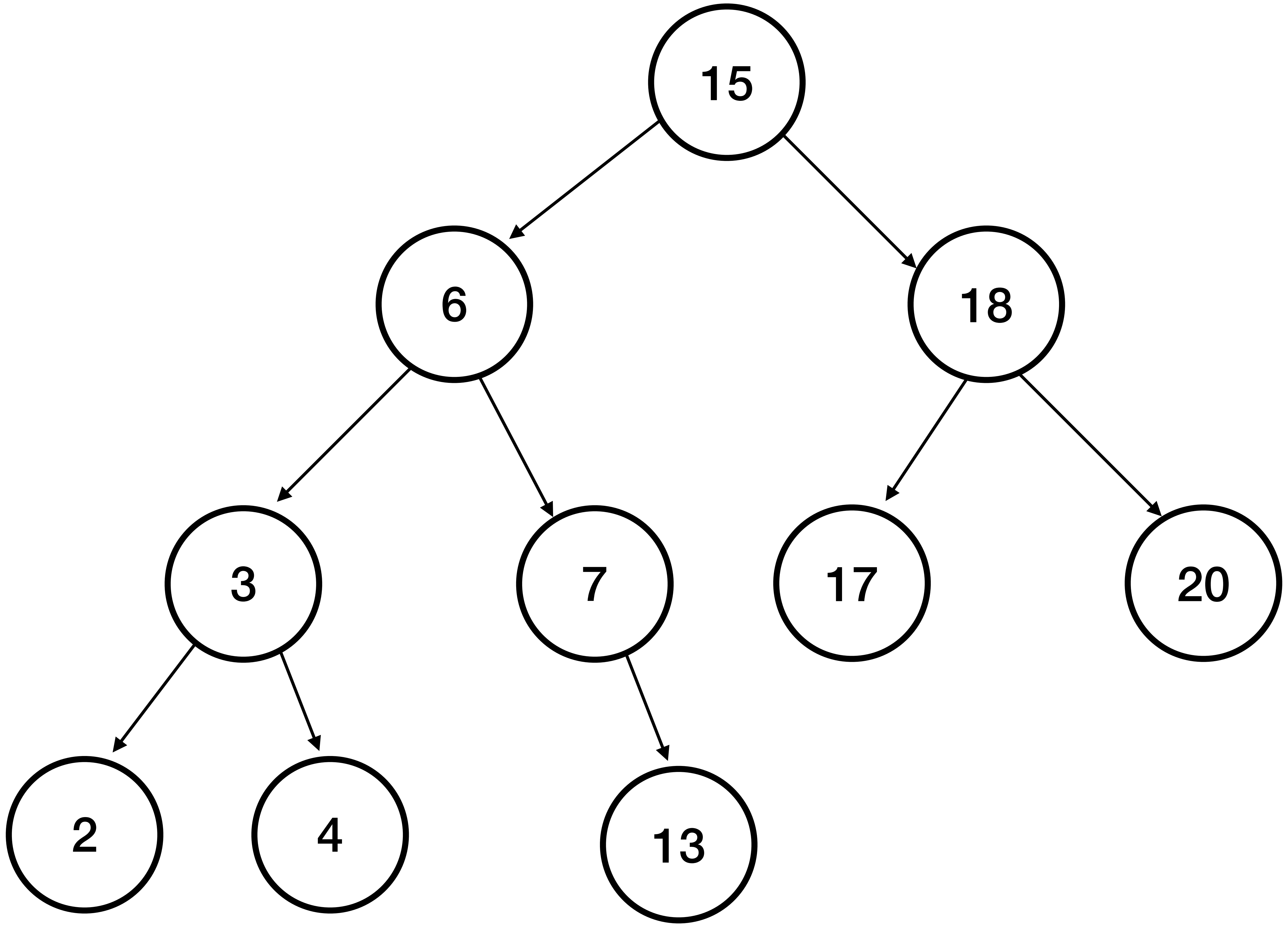
Search (iterative)

```
search(tree,k):
while tree ≠ NULL and k ≠ tree.key
    if k < tree.key
        tree := tree.left
    else
        tree := tree.right
    end if
end while
return tree
```



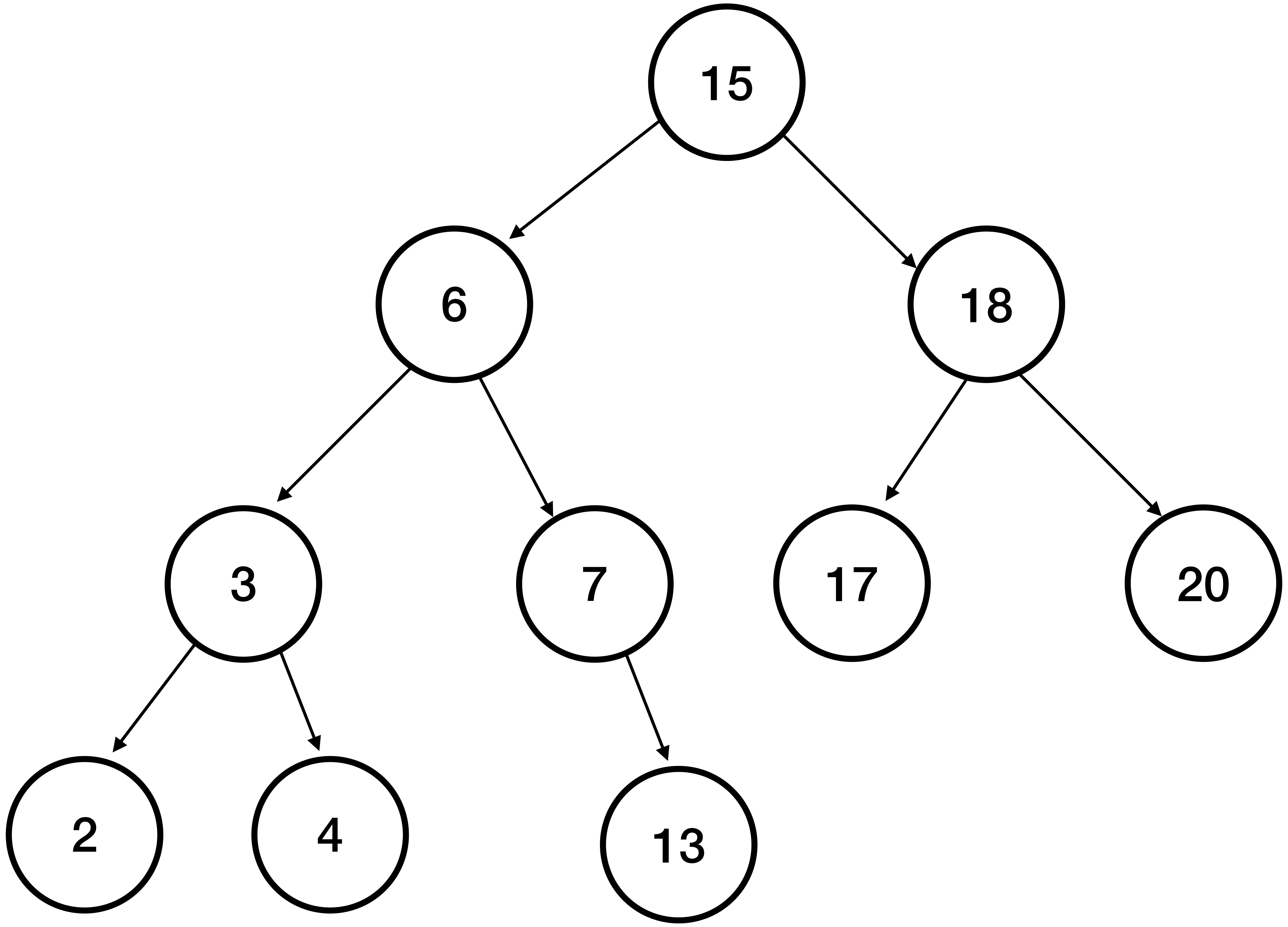
Minimum

```
minimum(tree):
while tree.left ≠ NULL
  | tree := tree.left
end while
return tree.key
```



Maximum

```
minimum(tree):
while tree.right ≠ NULL
  | tree := tree.right
end while
return tree.key
```



Successor and Predecessor

- Successor of a node x is the node with the smallest key such that it is larger than x.key

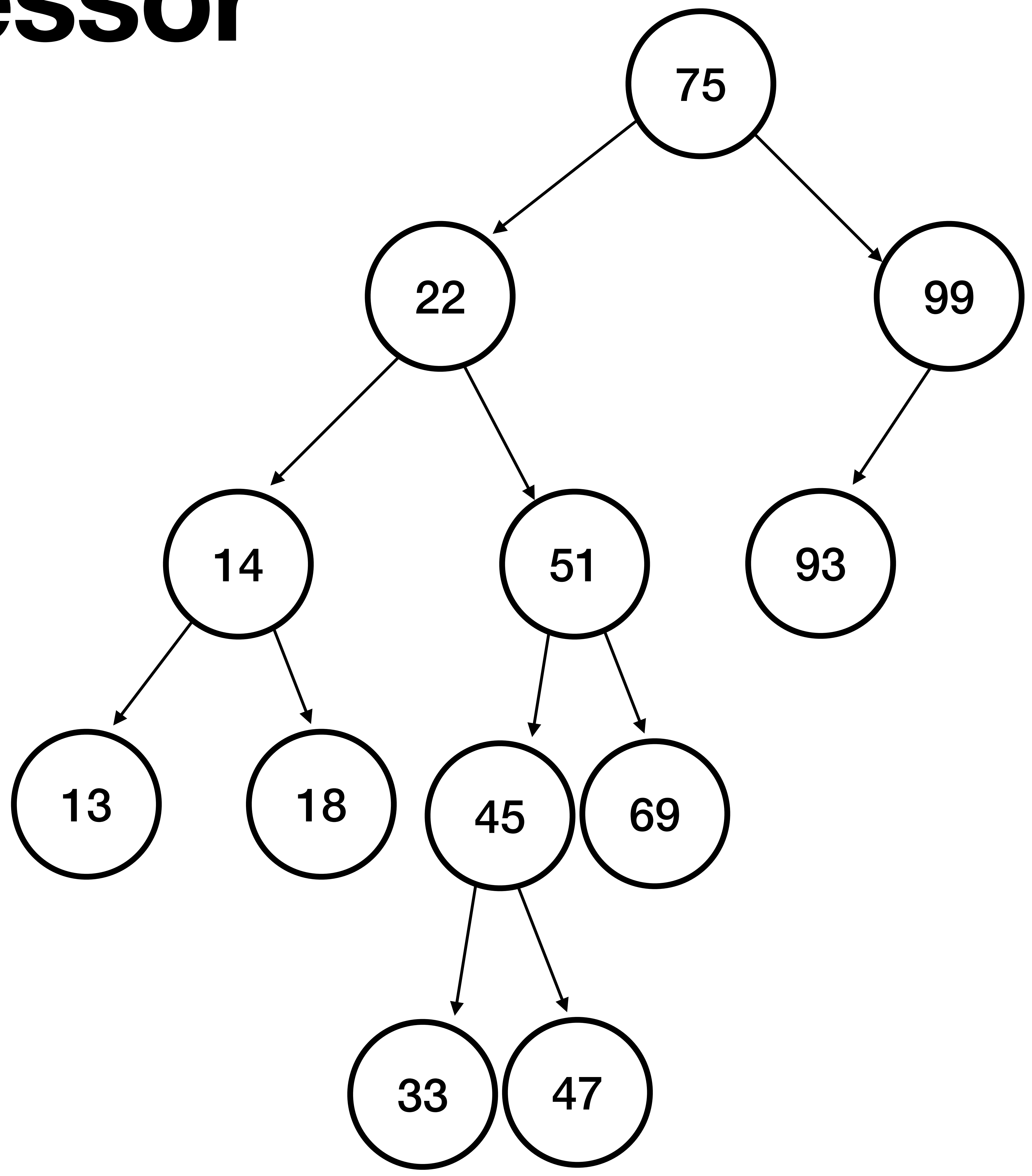
Successor of 51 is

Successor of 93 is

- Predecessor of a node x is the node with the largest key such that it is smaller than x.key

Predecessor of 51 is

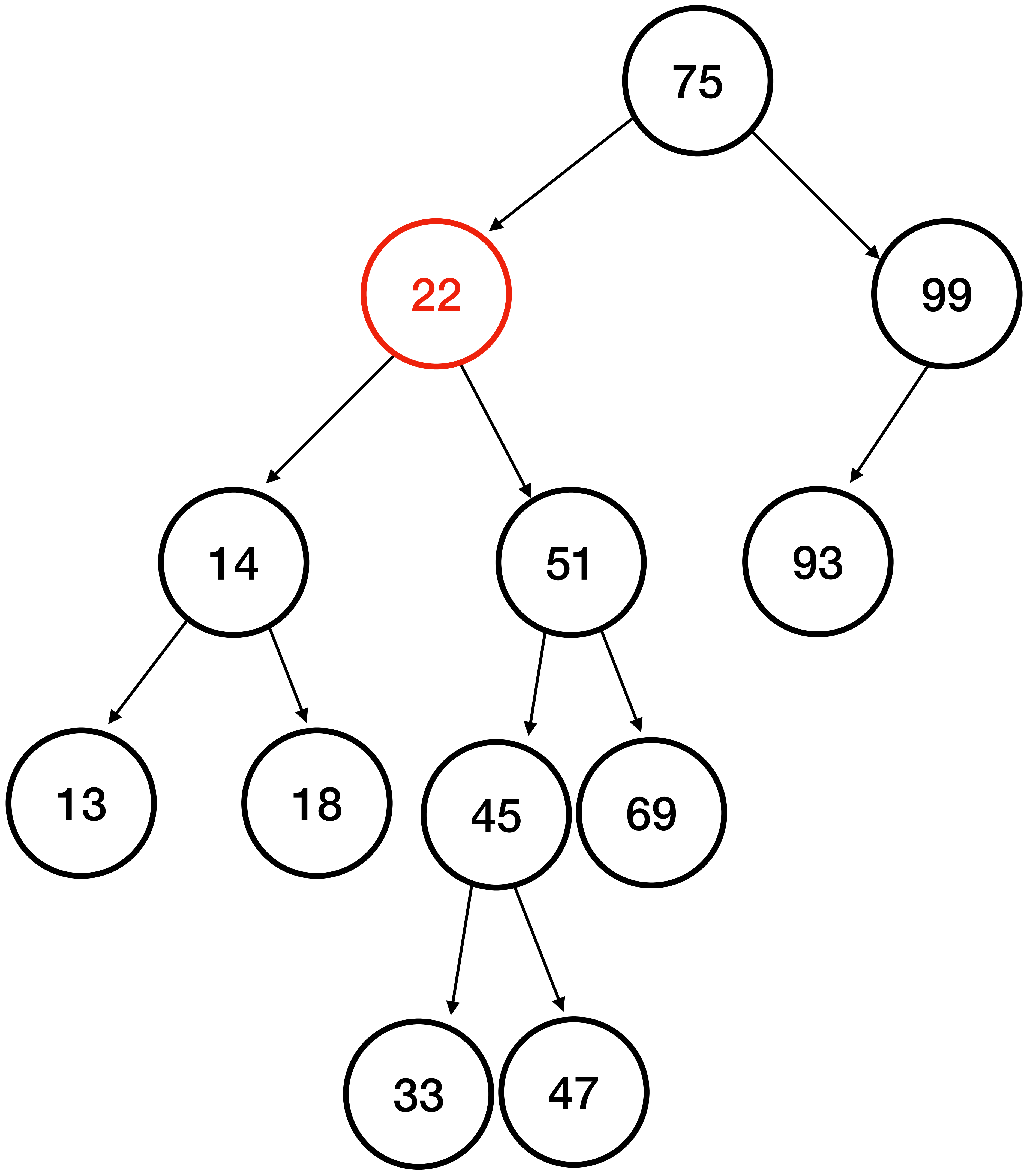
Predecessor of 93 is



Successor

Cases

- The minimum of the right sub-tree (if there is one)



Successor

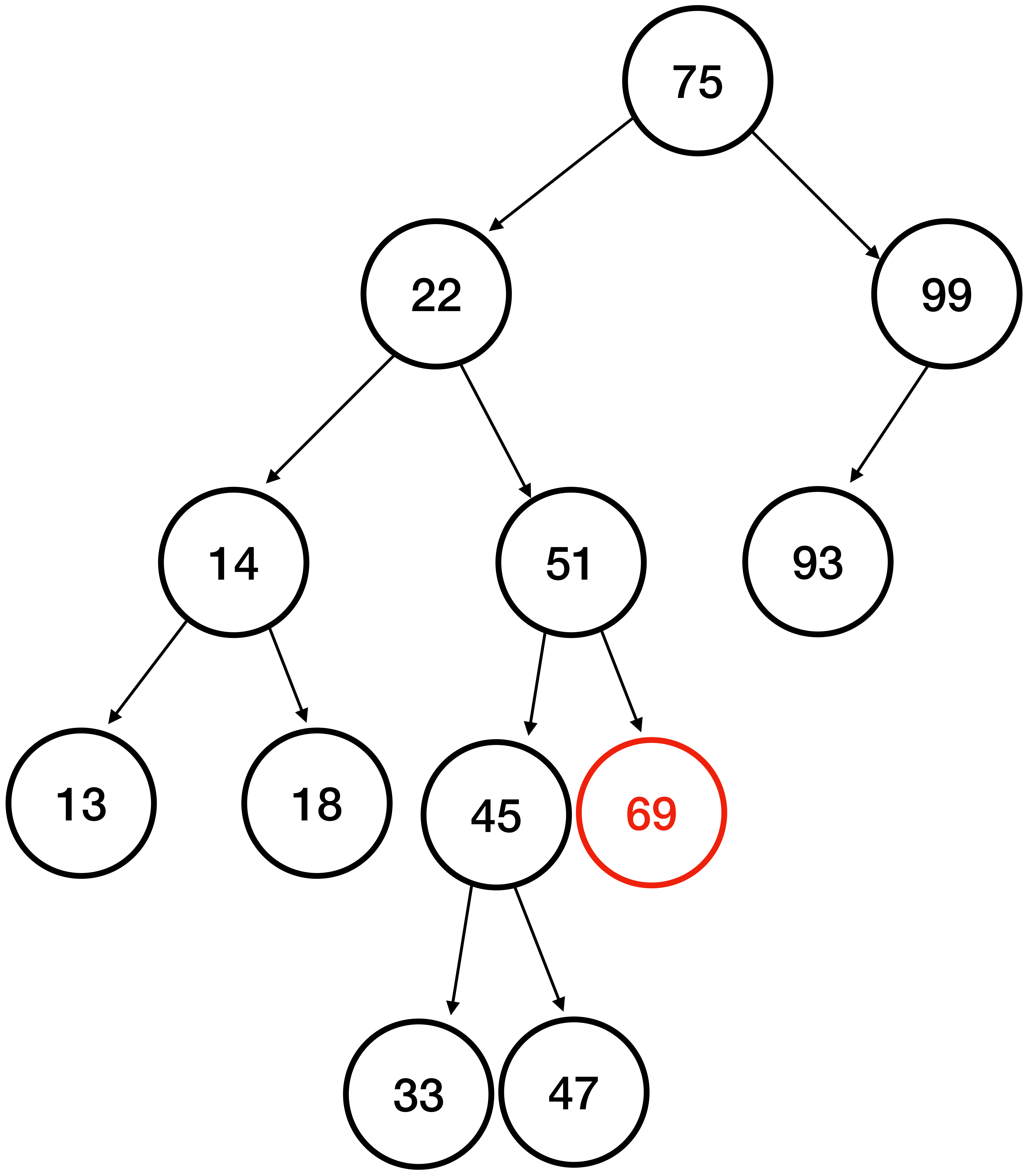
Cases

- The minimum of the right sub-tree (if there is one)
- The first ancestor of x such that its left child is an ancestor of x

Ancestor of 69: 51, 22, 75

```
successor(tree):
if tree.right ≠ NULL
    return minimum(tree)
end if
y := x.parent
while y ≠ NULL and x = y.right
    x := y
    y := y.parent
end while
return y
```

symmetric for predecessor



Insertion and deletion

Insertion

Two ways

- Insert at the root
    - Modify the structure of the BST
- Insert at the leave
    - Modify the height of the BST

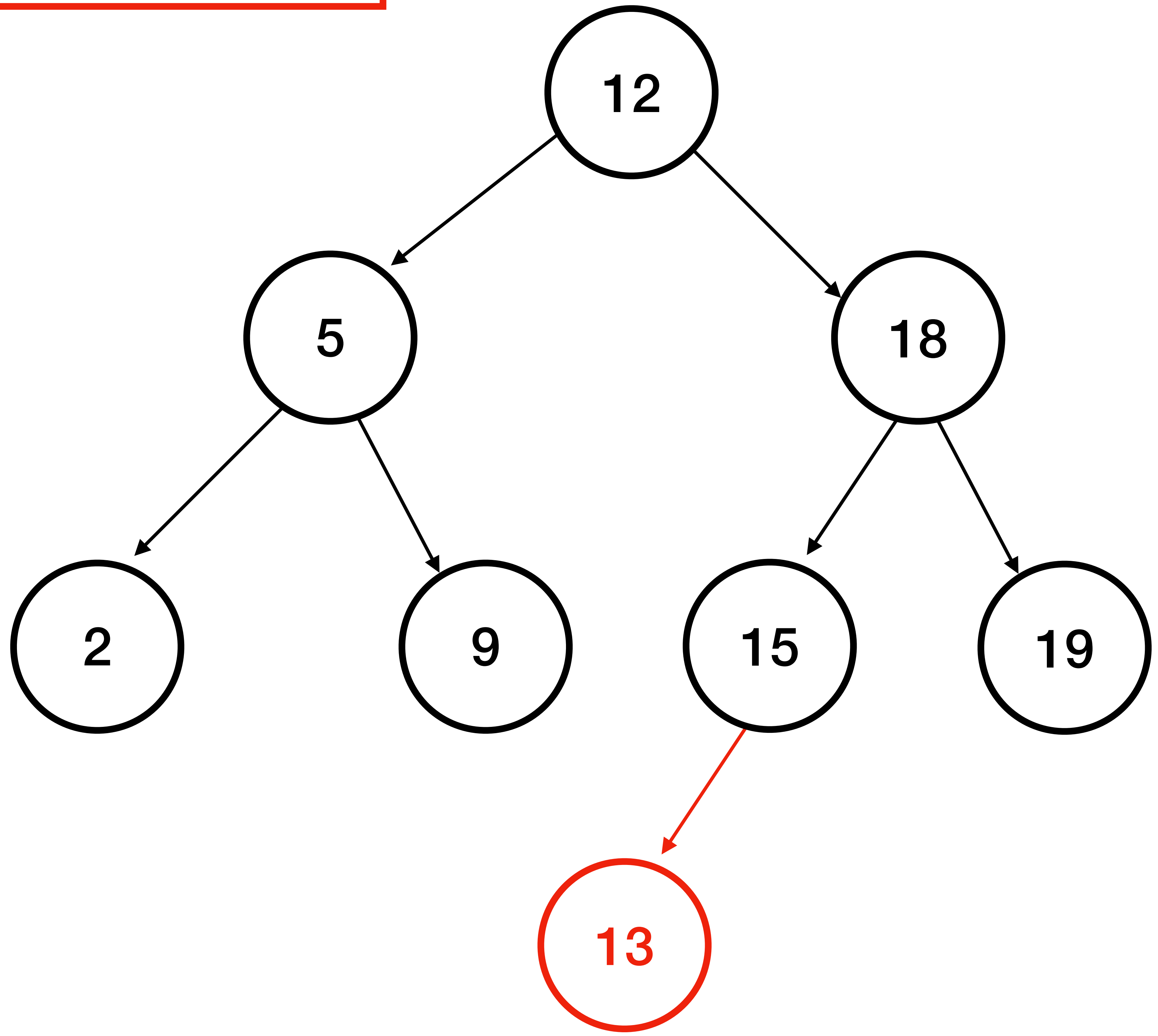
Insertion at the leave

```
Insert(tree, z)
y := NULL
x := T.root
while x ≠ NULL
  y := x
  if z.key < x.key
    x := x.left
  else
    x := x.right
  end if
end while
z.parent := y
if y = NULL
    T.root := z
  else
    if z.key < y.key
        y.left := z
      else
        y.right := z
    end if
end if
```

Search where to insert

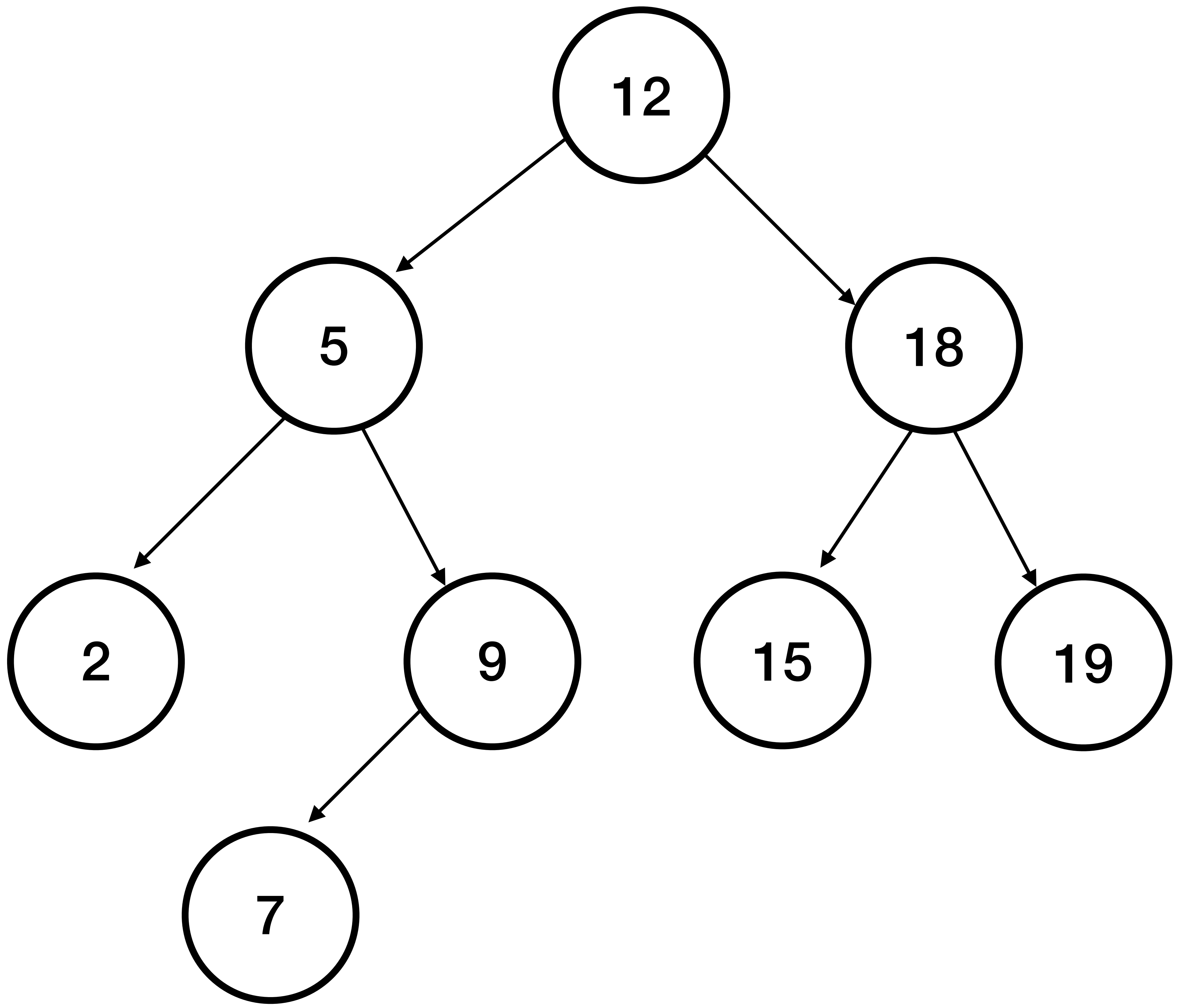
Insert node

Insert(tree, 13)

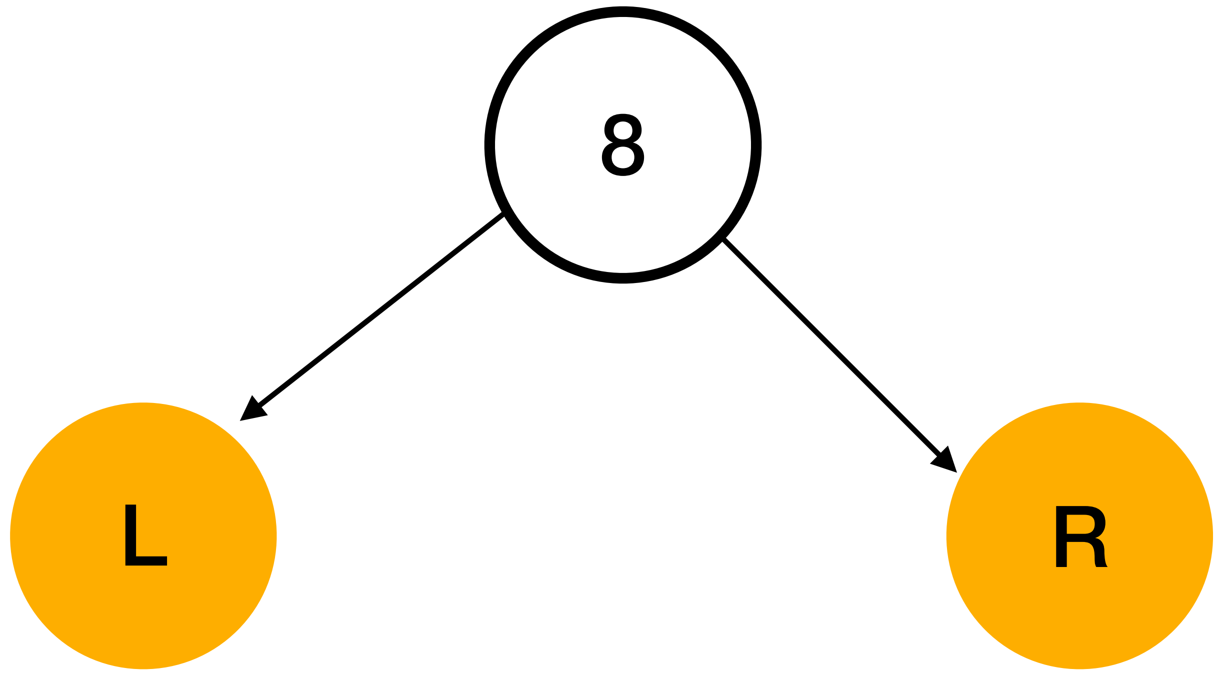


Insertion at the root

```
Insert_root(tree, 8)
```

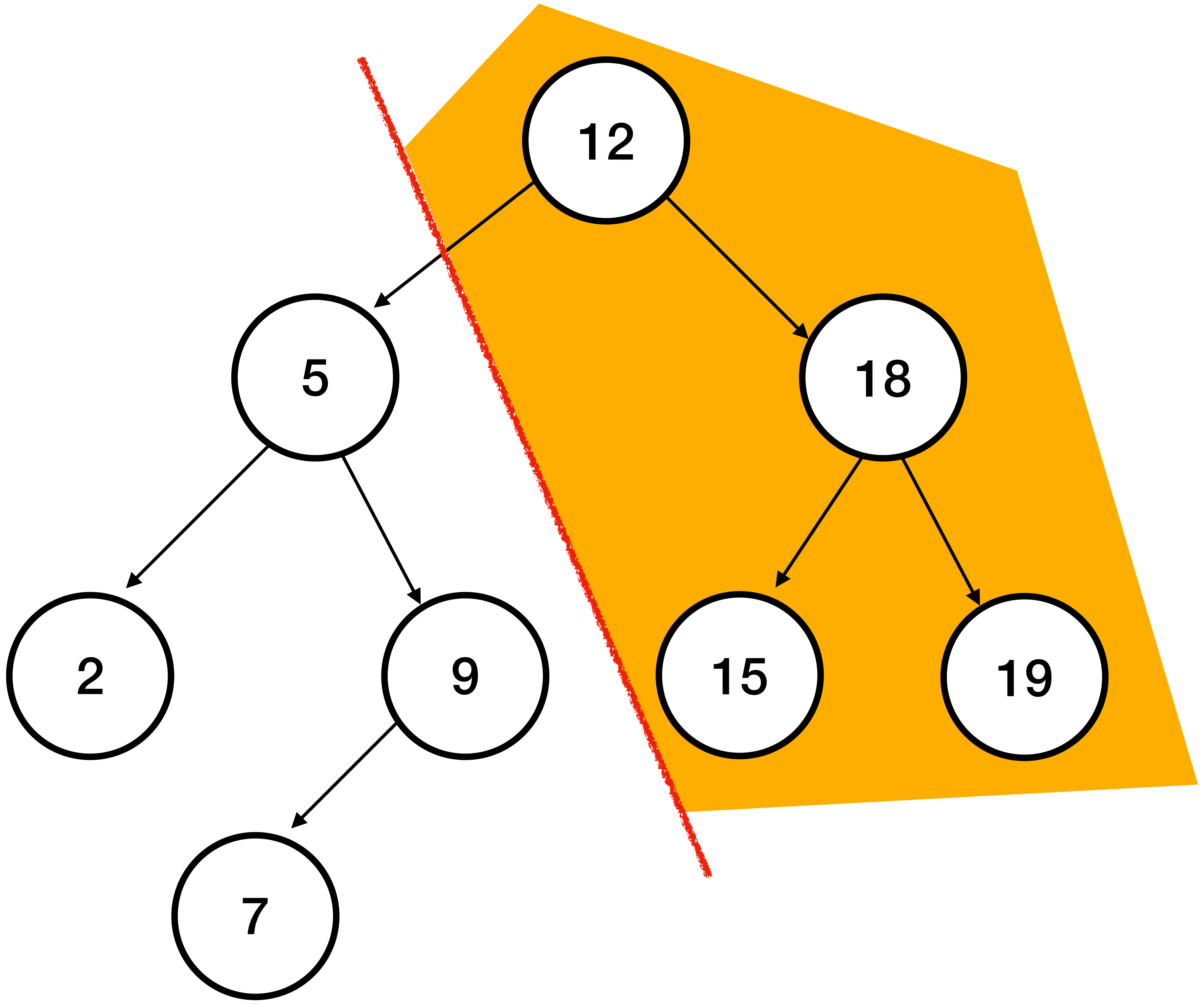


Insertion at the root

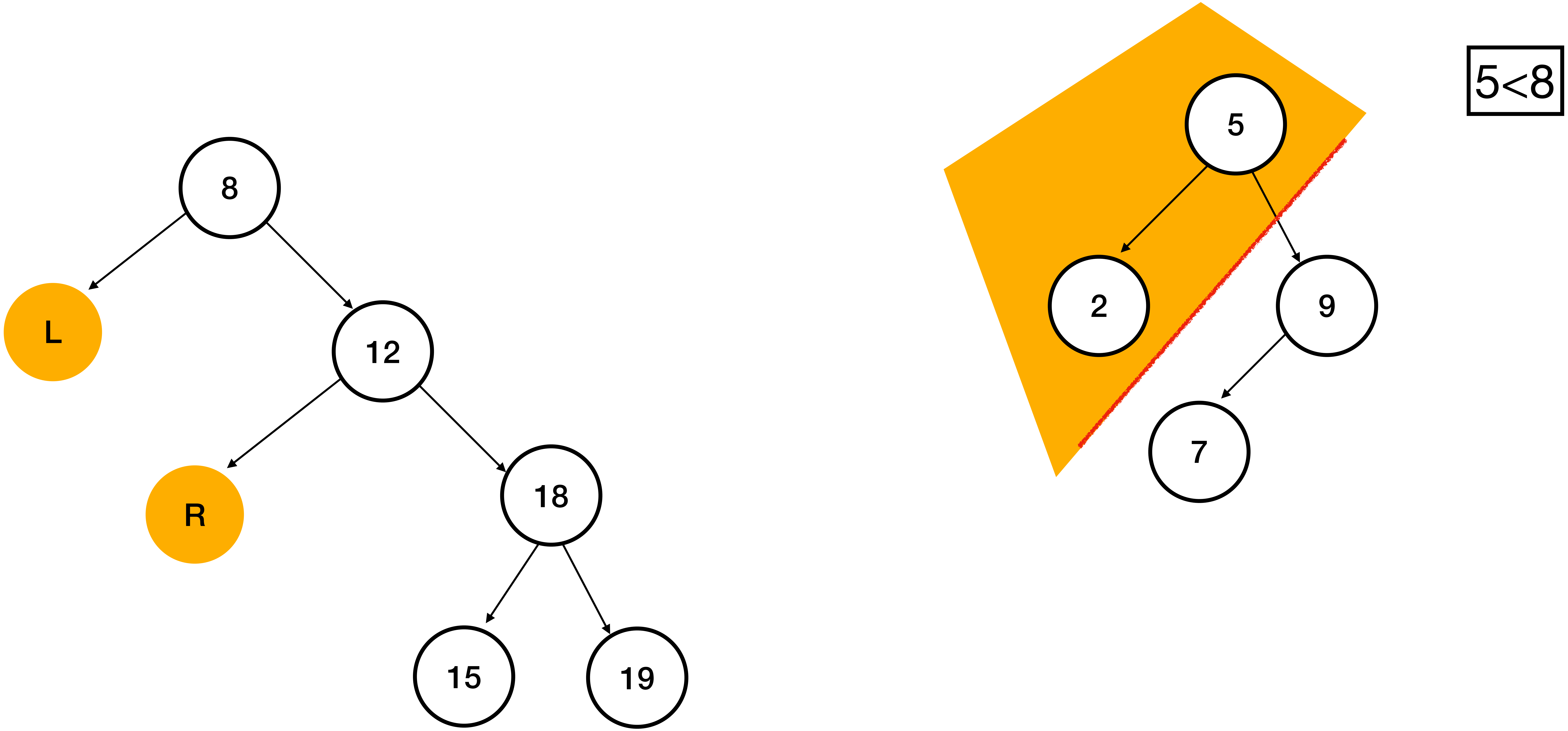


`Insert_root(tree, 8)`

12>8

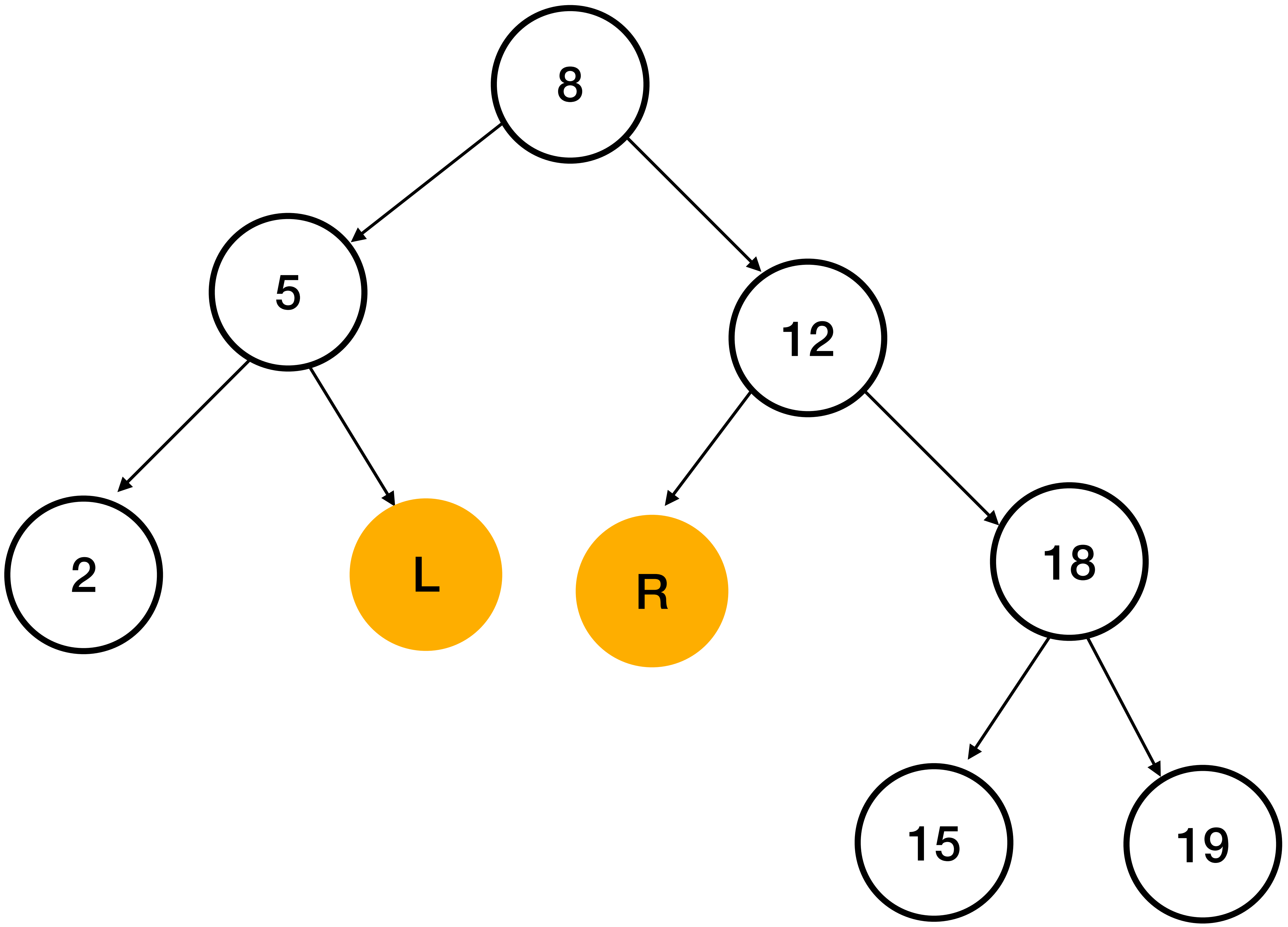


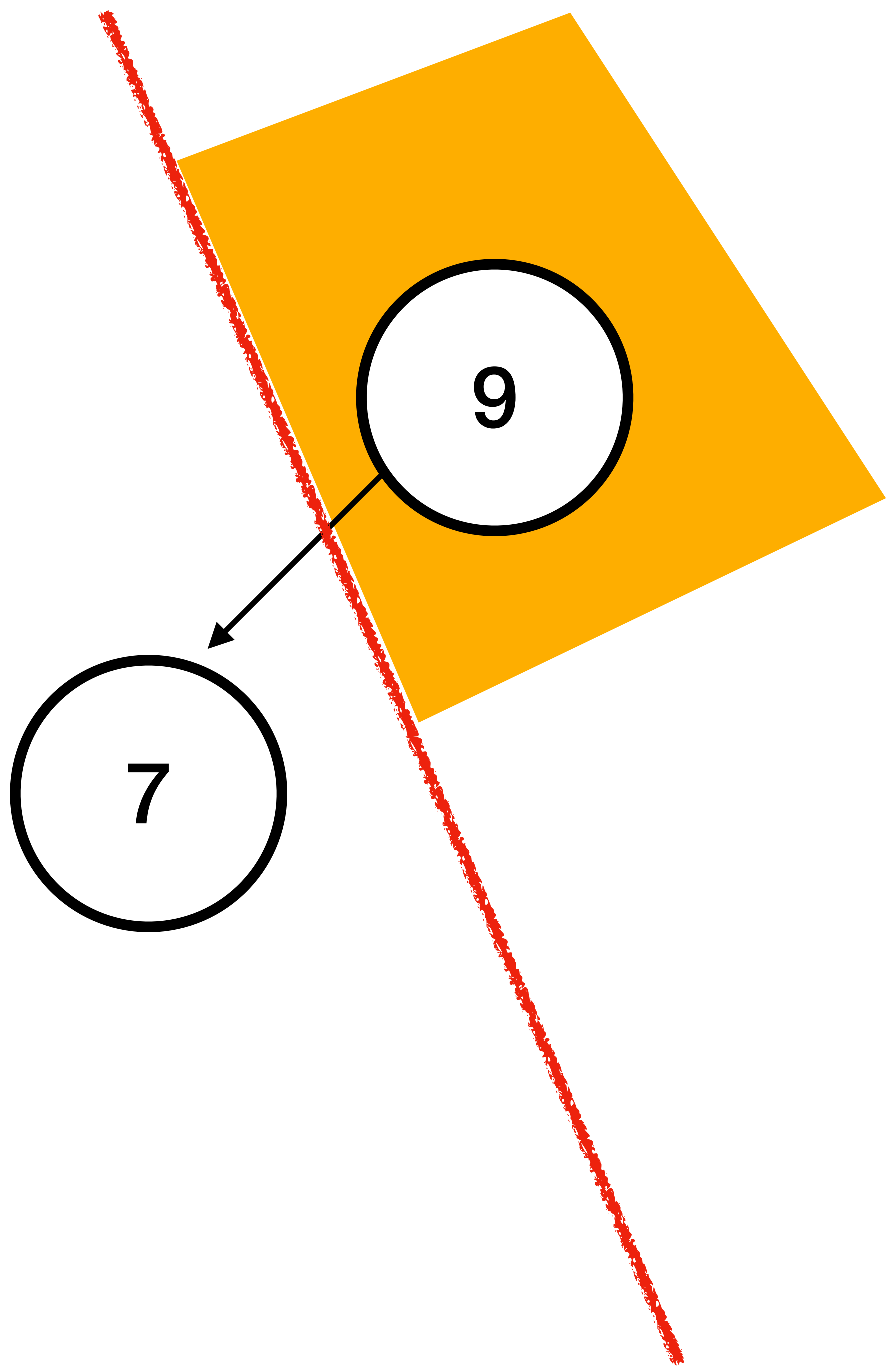
Insertion at the root



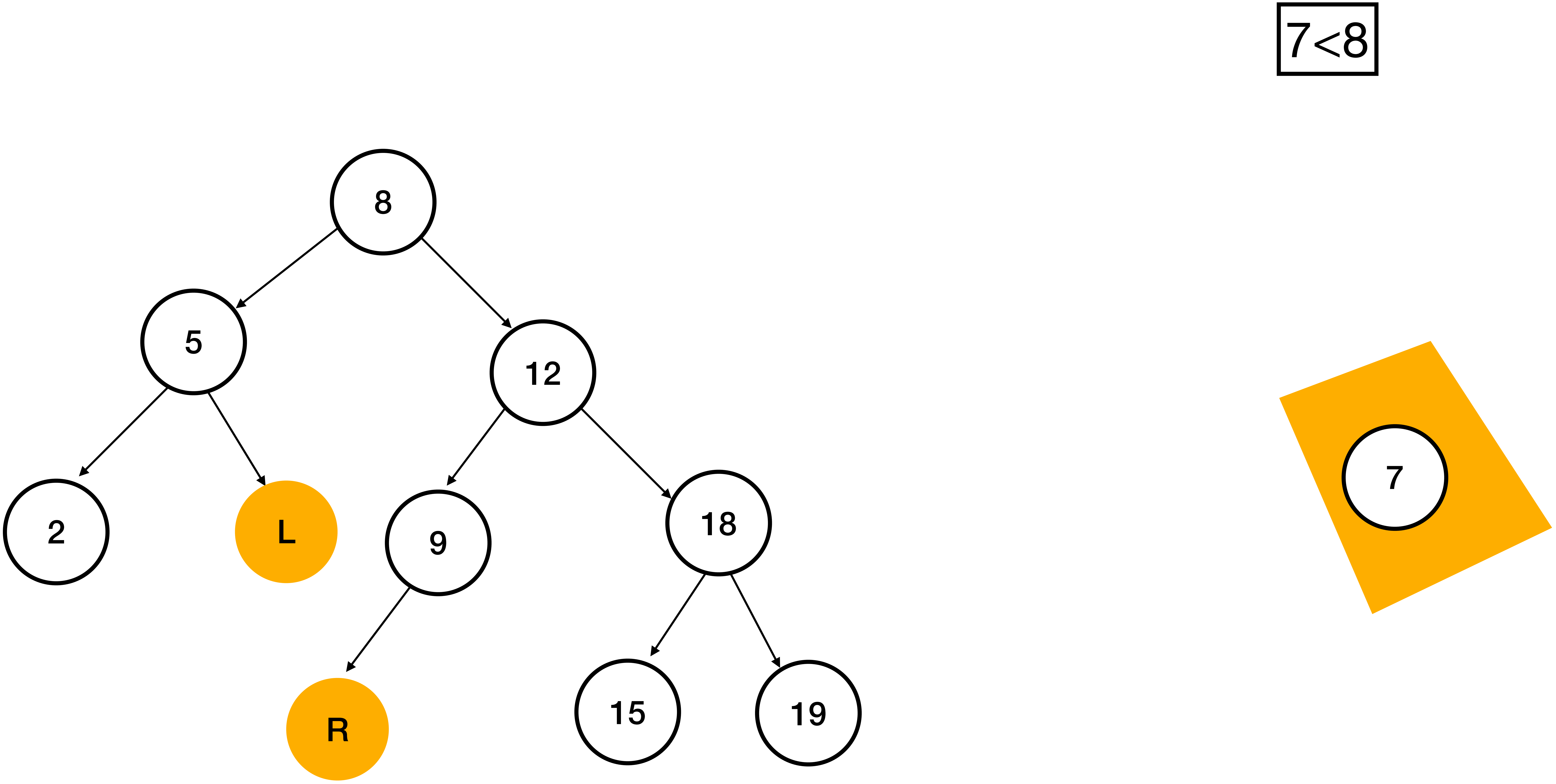
Insertion at the root

$\boxed{9>8}$



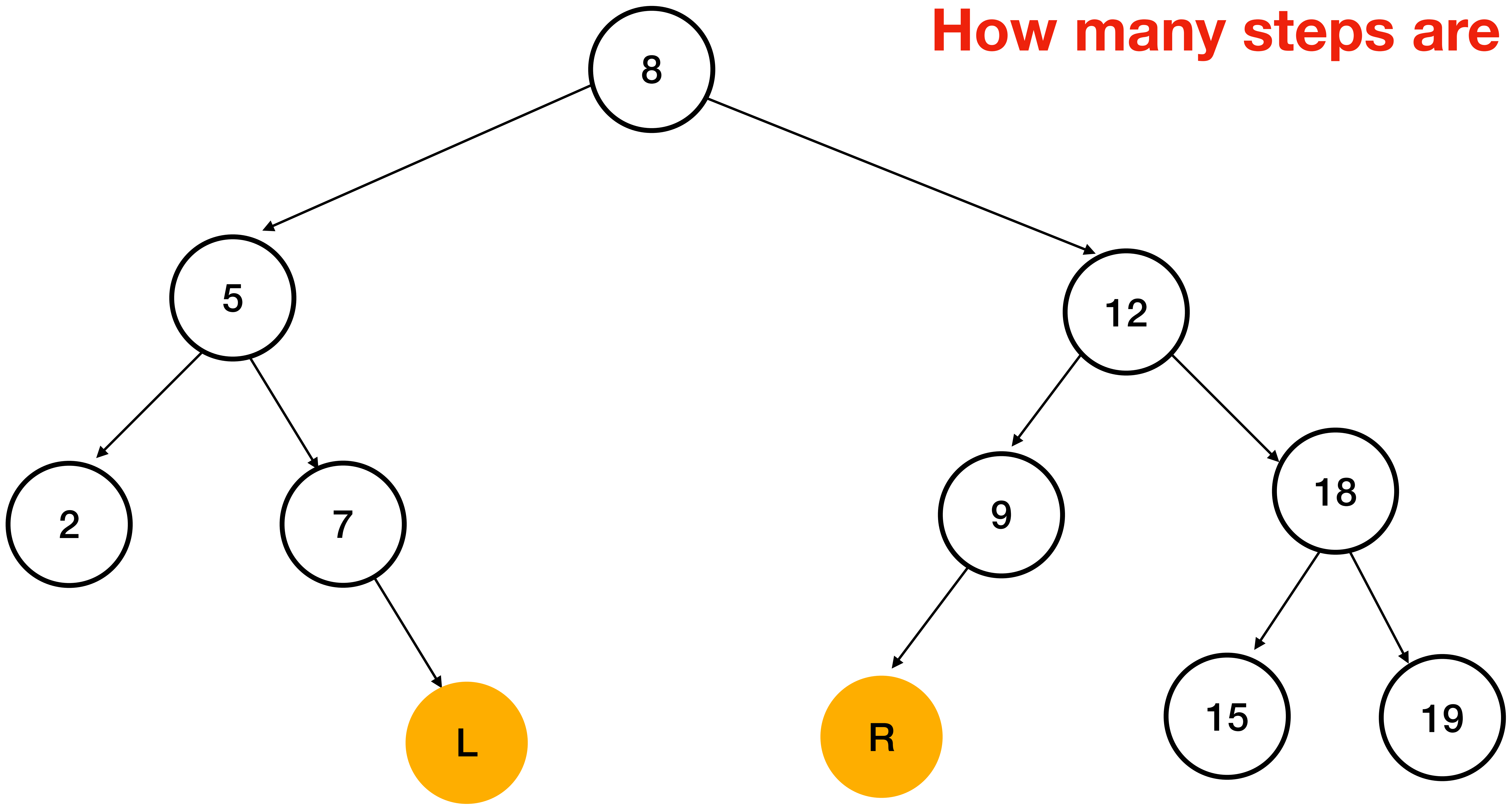


Insertion at the root



Insertion at the root

How many steps are needed?



Insertion at the root

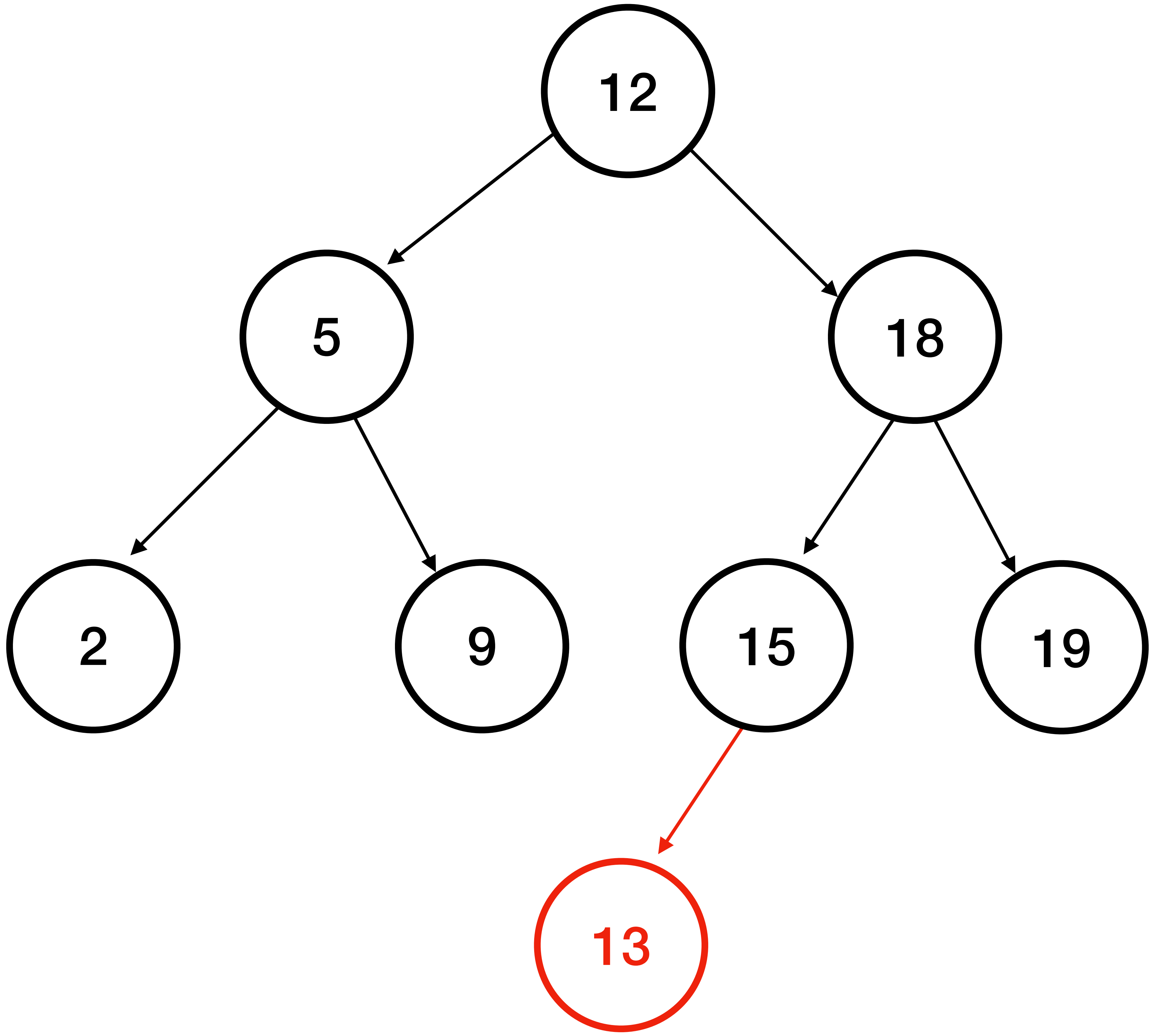
Pseudo code

```
Insert_root(tree, x)
  node root;
  node L,R;
  root.key := x
  cut(x, tree, L, R);
  root.left := L;
  root.right := R;
  return root;
```

```
cut(x, tree, L, R)
  if tree = NULL
     L := NULL, R := NULL;
   else
     if x < tree.key
        R := tree;
        cut(x, tree.left, L, R.left)
      else
        L := tree;
        cut(x, tree.right, L.right, R)
     end if
  end if
```

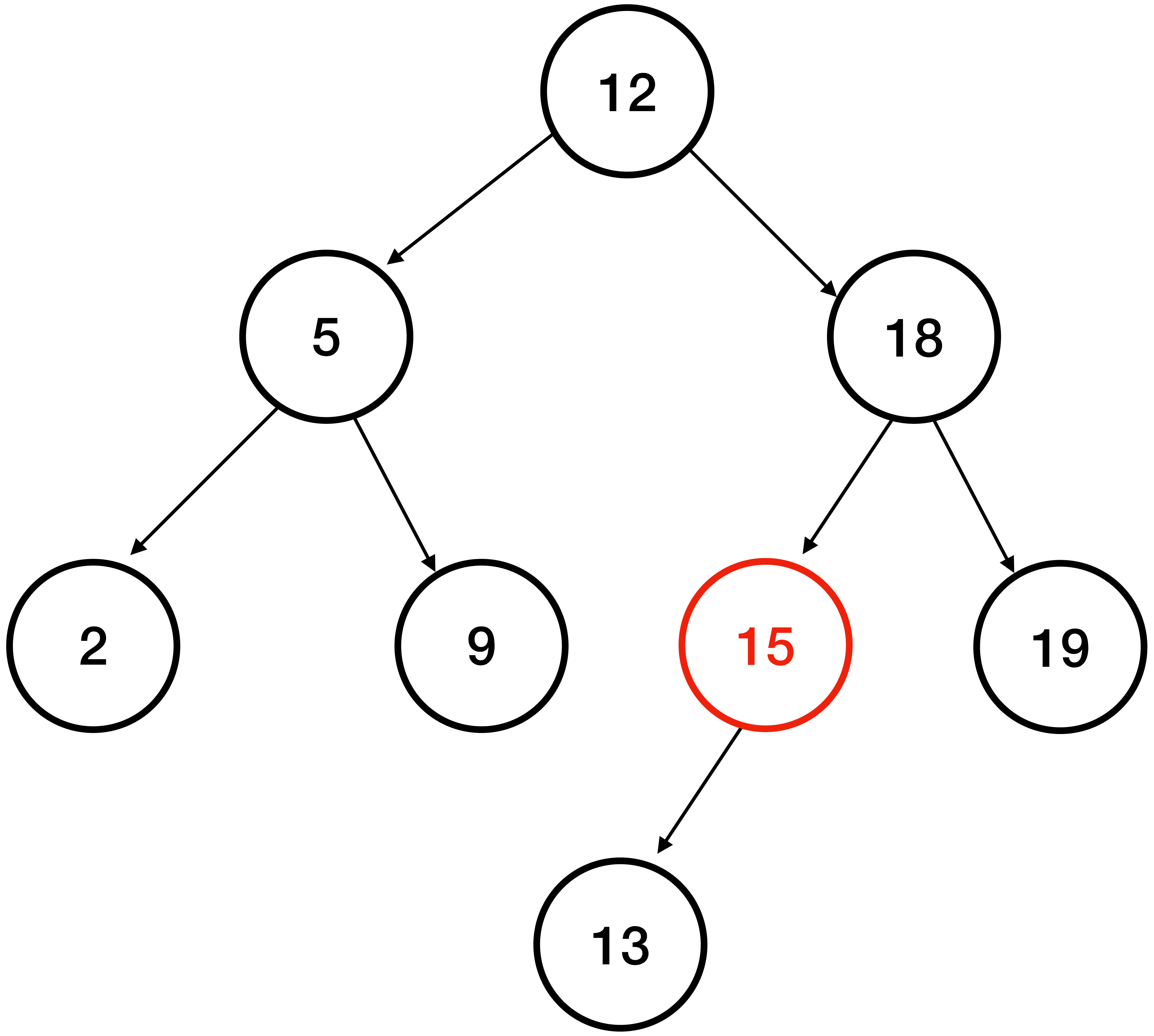
Deletion of a node z

- 1st case: z is a leaf node
  - delete directly



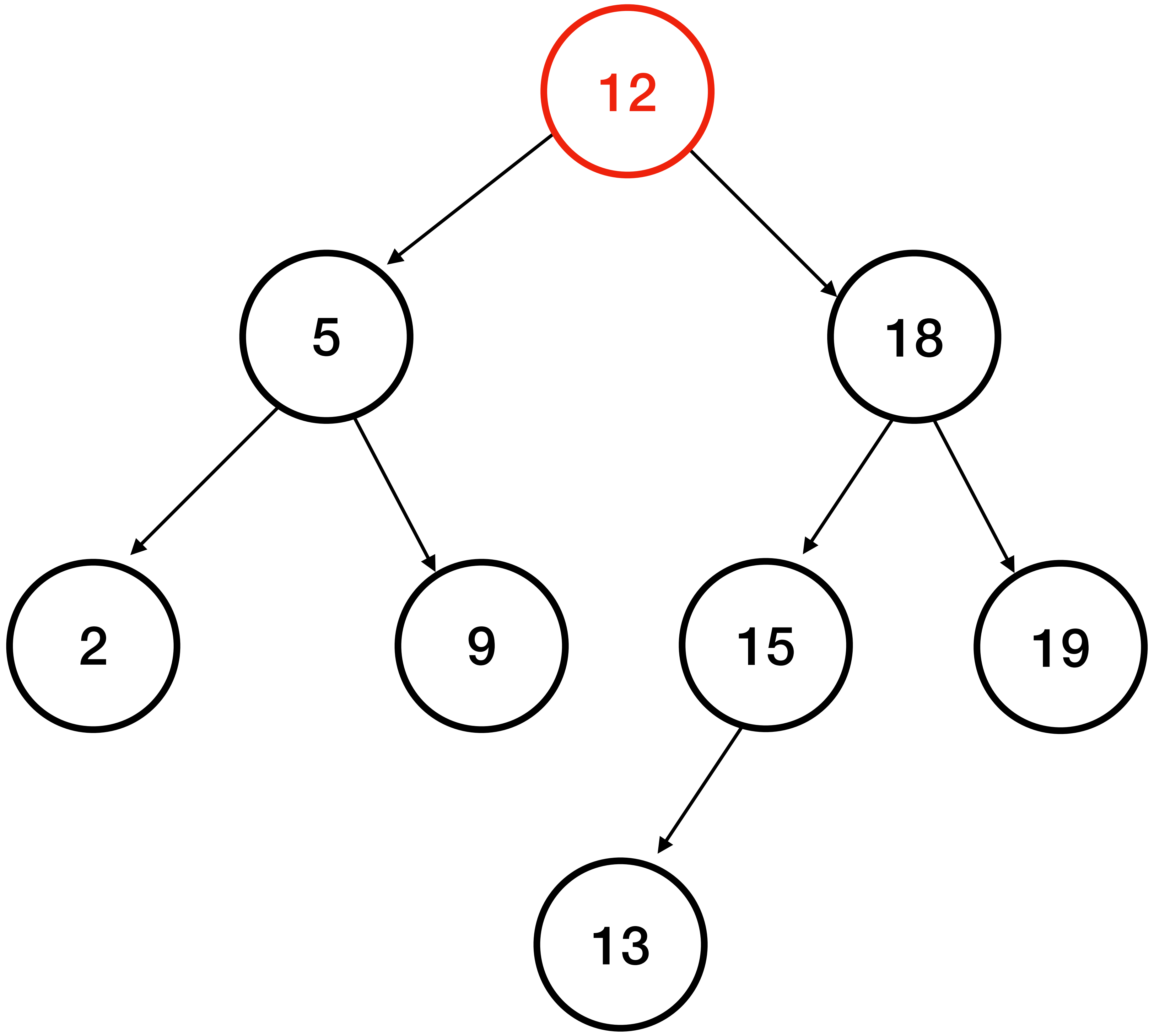
Deletion of a node z

- 1st case: z is a leaf node
  - delete directly
- 2nd case: z only has one child
  - replace the node by its child



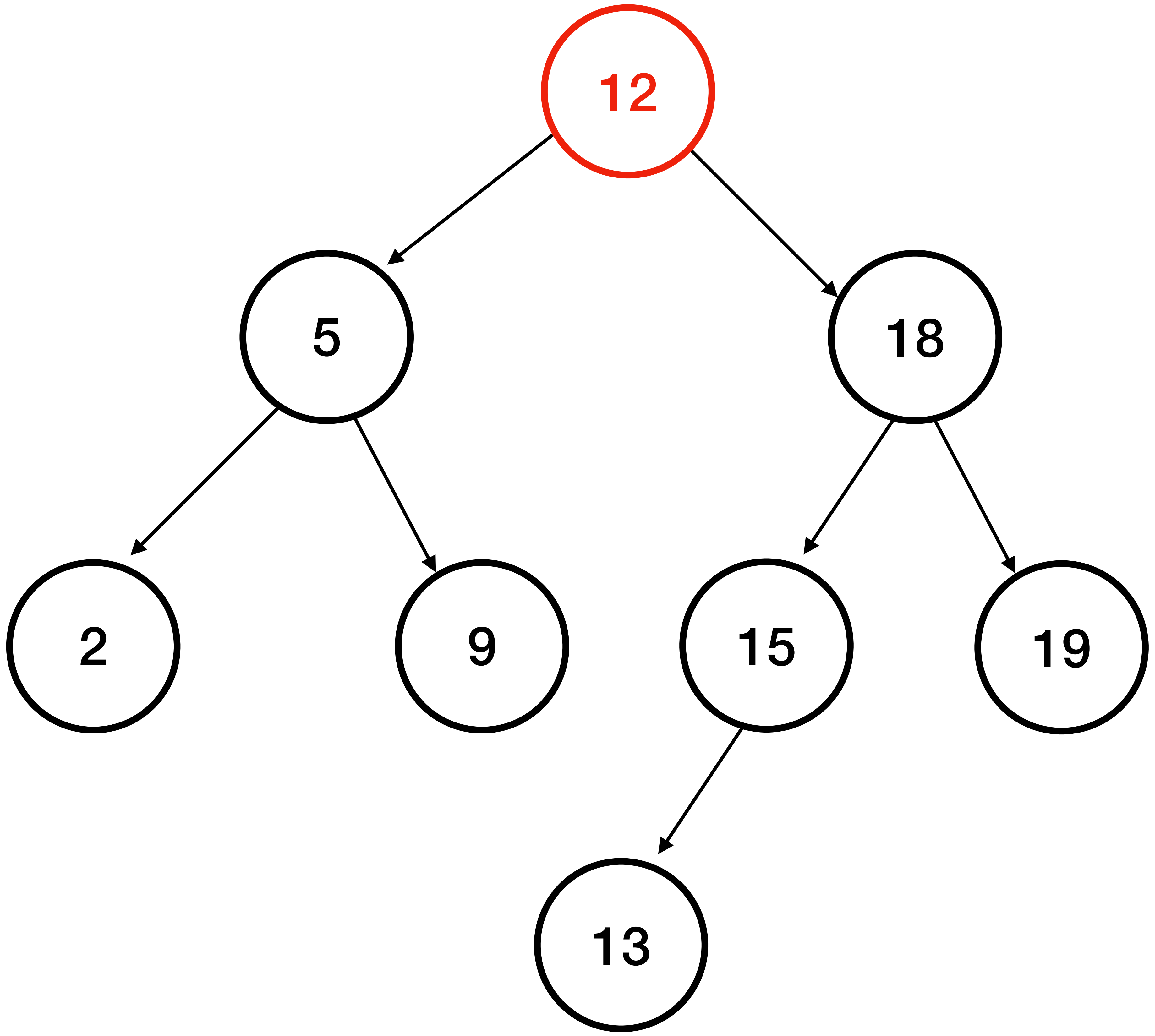
Deletion of a node z

- 1st case: z is a leaf node
    - delete directly
- 2nd case: z only has one child
    - replace the node by its child
- 3rd case: z has 2 children
    - find successor of z
    - remove successor of z from the tree
    - replace z by its successor



Deletion of a node z

- 1st case: z is a leaf node
  - delete directly
- 2nd case: z only has one child
  - replace the node by its child
- 3rd case: z has 2 children
  - find successor of z
  - remove successor of z from the tree
  - replace z by its successor



Successor of z has at most 1 child

Deletion of a node z

pseudo-code

```
Delete(T, k):
    z:=search(T, k)
    if z ≠ NULL
        if z.left = NULL
            Subtree-Shift(T, z, z.right)
          else
            if z.right = NIL
                Subtree-Shift(T, z, z.left)
          else
            y := Tree-Successor(z)
            if y.parent ≠ z
              Subtree-Shift(T, y, y.right)
              y.right := z.right
              y.right.parent := y
            end if
            Subtree-Shift(T, z, y)
            y.left := z.left
            y.left.parent := y
        end if
    end if
```

```
Subtree-Shift(T, u, v):
    if u.parent = NULL
        T.root := v
      else if u = u.parent.left
        u.parent.left := v
      else
        u.parent.right := v
    end if
    if v ≠ NULL
        v.parent := u.parent
    end if
```

Summary of Binary Search Tree

Comparison with others

algorithm (data structure)	worst-case cost (after n inserts)		average-case cost (after n random inserts)		efficiently support ordered operations?
	search	insert	search hit	insert	
sequential search (unordered linked list)	n	n	n/2	n	no
binary search (ordered array)	log(n)	n	log(n)	n/2	yes
binary tree search (BST)	n	n	1.39 log(n)	1.39 log(n)	yes

Branch & Bound

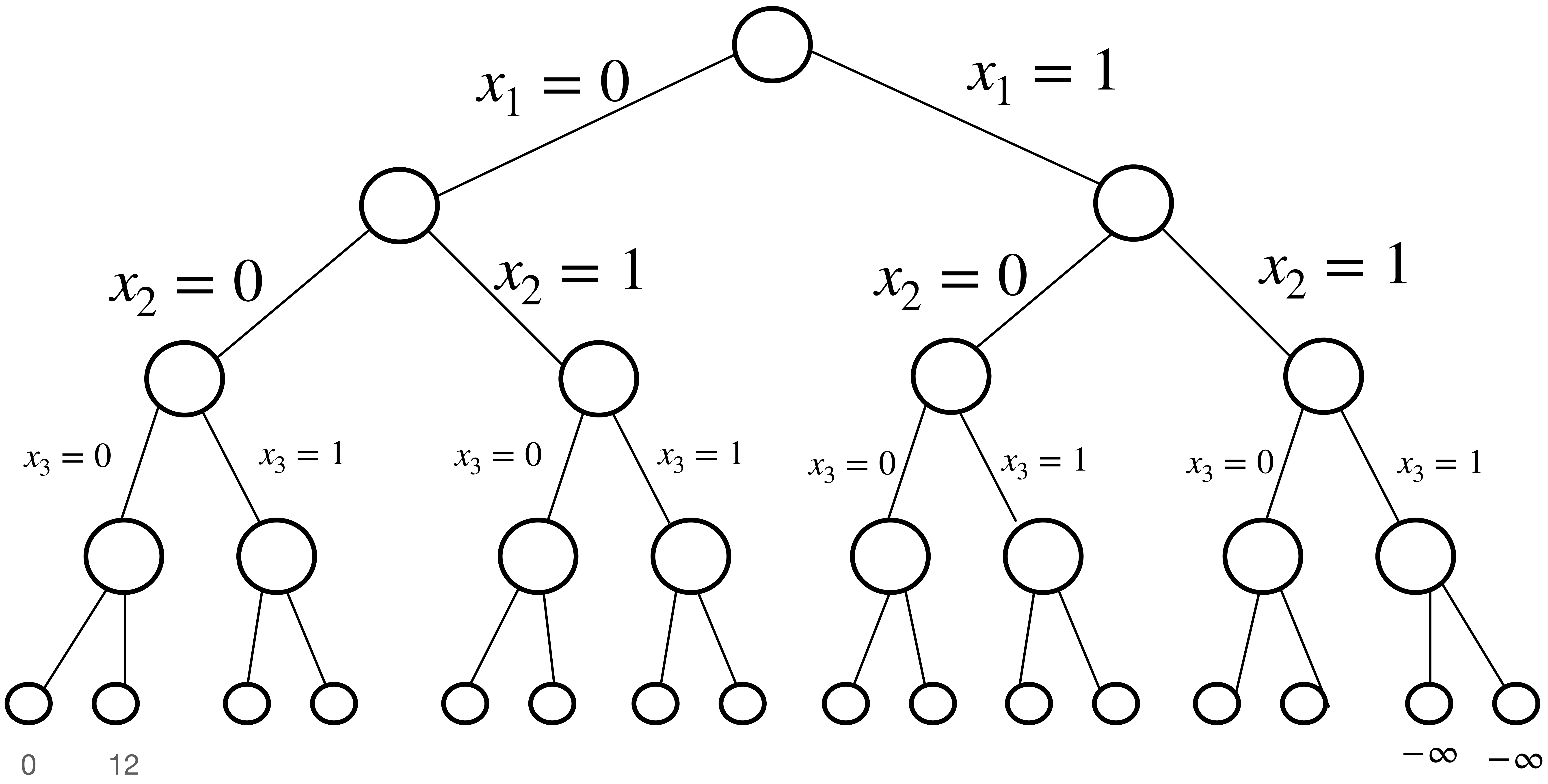
Branch & Bound

Minimization problem (maximization problem)

- Algorithm technique for discrete and combinatorial optimization problems to compute the optimal solution
- Enumeration of candidate solutions by means of state space search
- Construct a rooted tree
- Fix part of the decisions of a solution and evaluate lower (upper) bound
- If lower (upper) bound is too high (low), we do not need to explore this branch

Knapsack problem

$$x_i = \begin{cases} 1 \text{ if item i is chosen} \\ 0 \text{ otherwise} \end{cases}$$

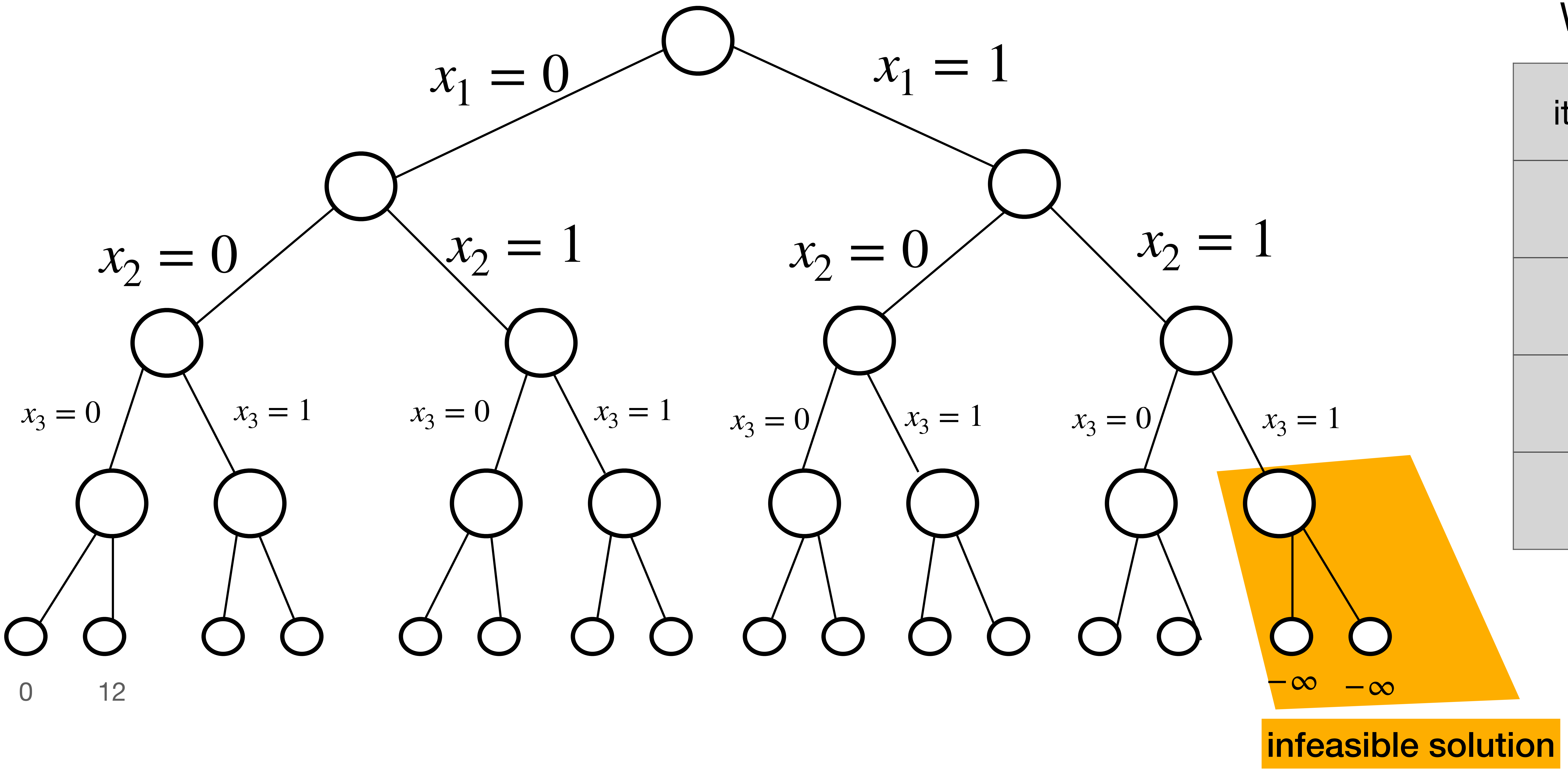


Weight capacity K=25

item i	weight w_i	value v_i
1	9	9
2	10	9
3	7	6
4	22	12

Knapsack problem

$$x_i = \begin{cases} 1 \text{ if item i is chosen} \\ 0 \text{ otherwise} \end{cases}$$

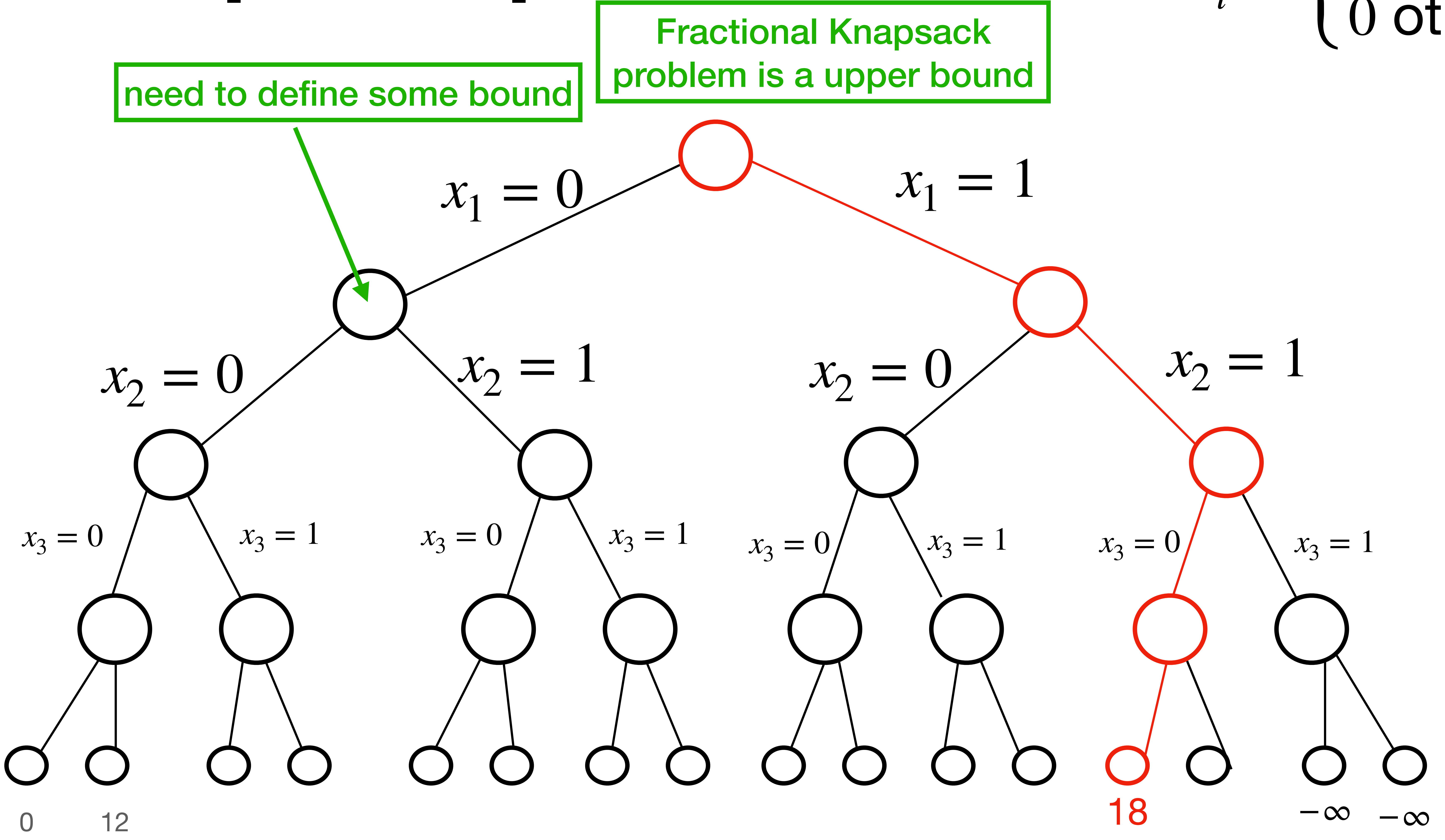


Weight capacity K=25

item i	weight w_i	value v_i
1	9	9
2	10	9
3	7	6
4	22	12

Knapsack problem

$$x_i = \begin{cases} 1 \text{ if item i is chosen} \\ 0 \text{ otherwise} \end{cases}$$



Weight capacity K=25

item i	weight w_i	value v_i
1	9	9
2	10	9
3	7	6
4	22	12

Upper bound when $x_1 = 0$

- Use the greedy algorithm for the Fractional Knapsack Problem on items 2, 3, 4

item i	use weight	get value	quantity
1	0 (9)	0 (9)	0
2	10 (10)	9 (9)	1
3	7 (7)	6 (6)	1
4	8 (22)	~4.36 (12)	(25-(10+7))/22 ~0.36

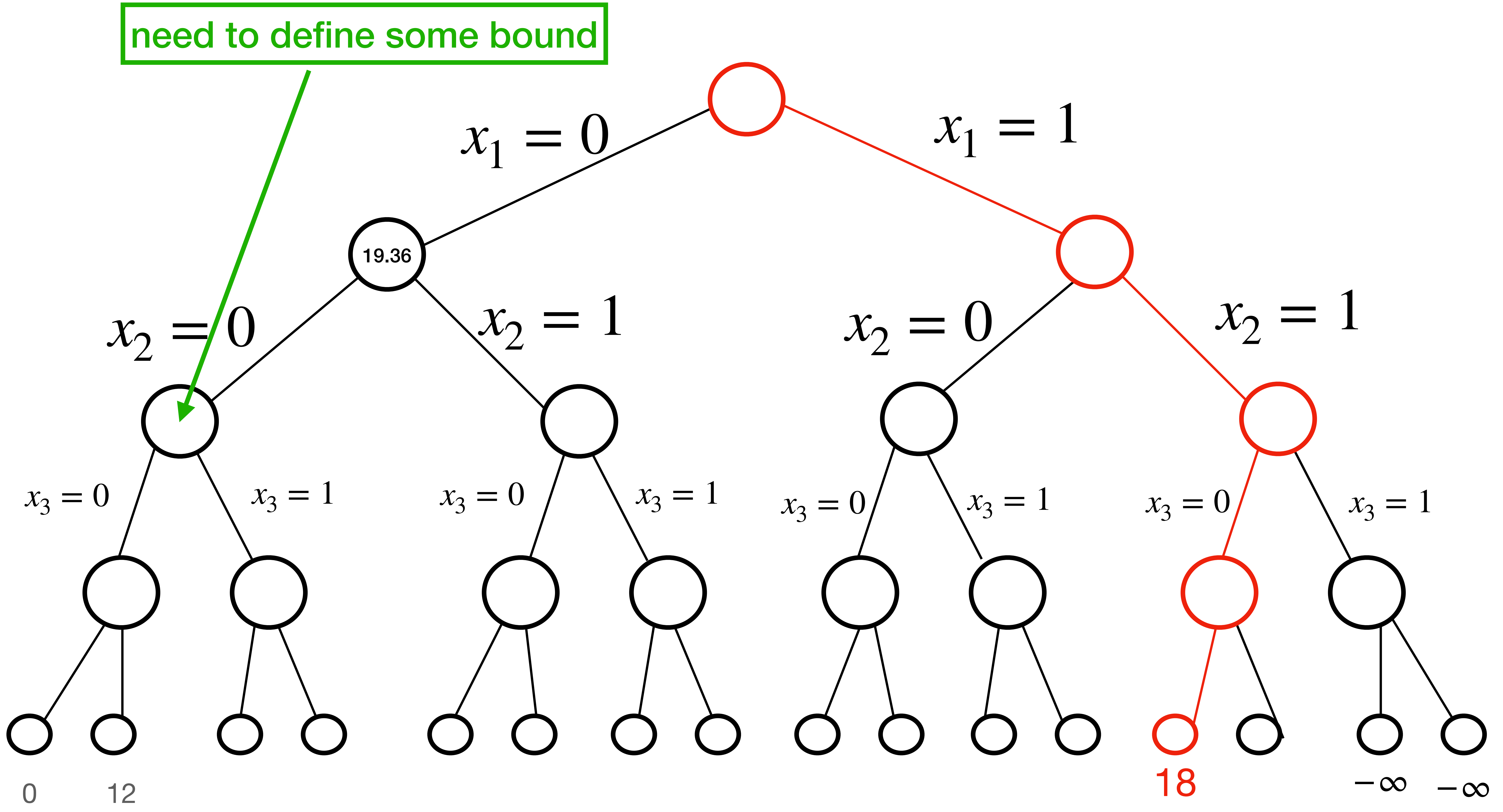
Weight capacity K=25

item i	weight w_i	value v_i
1	9	9
2	10	9
3	7	6
4	22	12

Best solution in this branch may be as good as 9+6+4.36 = 19.36

Knapsack problem

$$x_i = \begin{cases} 1 \text{ if item i is chosen} \\ 0 \text{ otherwise} \end{cases}$$



Weight capacity K=25

item i	weight w_i	value v_i
1	9	9
2	10	9
3	7	6
4	22	12

Upper bound when $x_1 = 0,\ x_2 = 0$

Weight capacity K=25

item i	weight w_i	value v_i
1	9	9
2	10	9
3	7	6
4	22	12

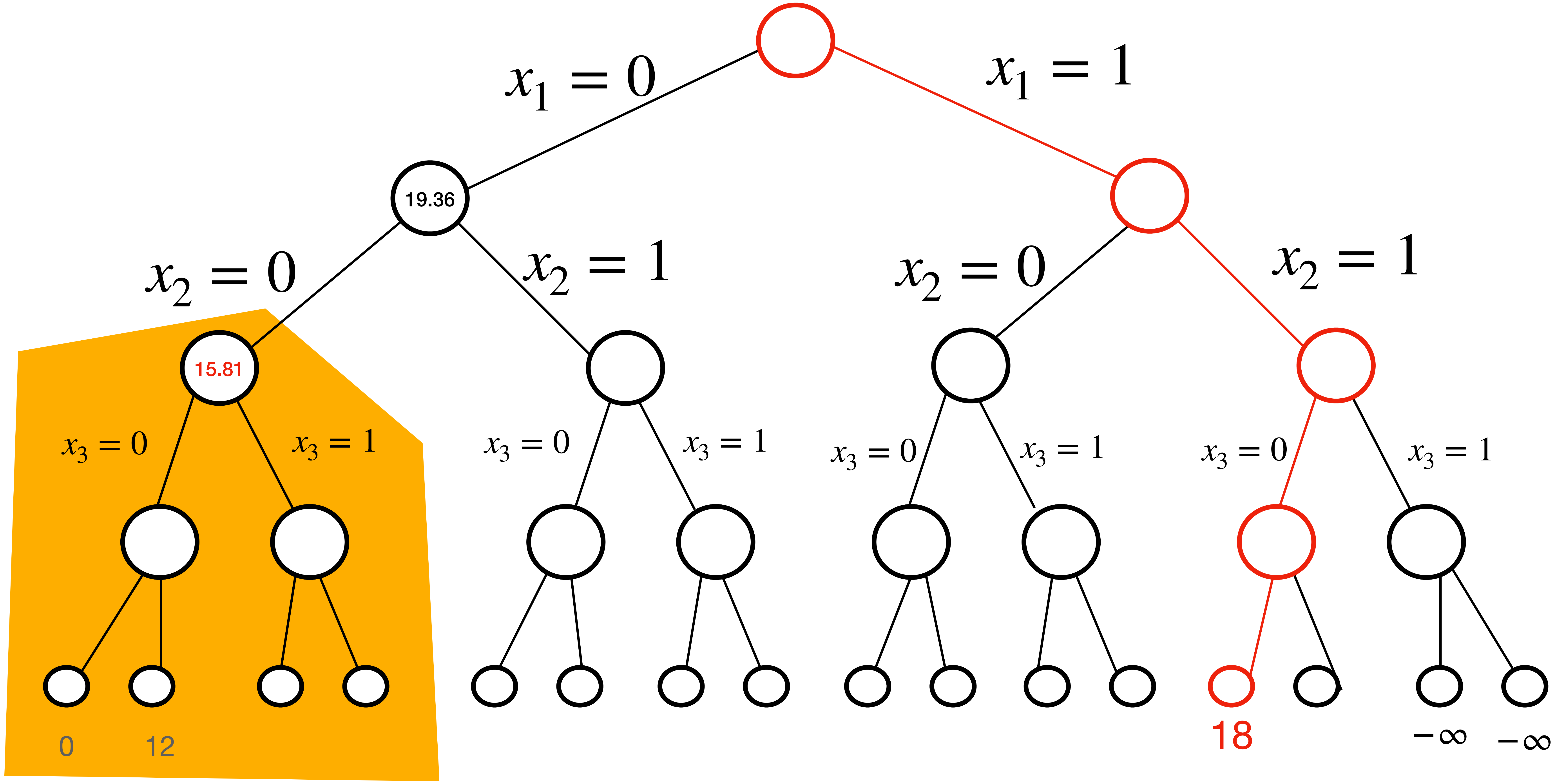
- Use the greedy algorithm for the Fractional Knapsack Problem on items 2, 3, 4

item i	use weight	get value	quantity
1	0 (9)	0 (9)	0
2	0 (10)	0 (9)	1
3	7 (7)	6 (6)	1
4	18 (22)	~9.81 (12)	(25-7)/22 ~0.81

Best solution in this branch may be as good as 6+9.81 = 15.81

Knapsack problem

$$x_i = \begin{cases} 1 \text{ if item i is chosen} \\ 0 \text{ otherwise} \end{cases}$$



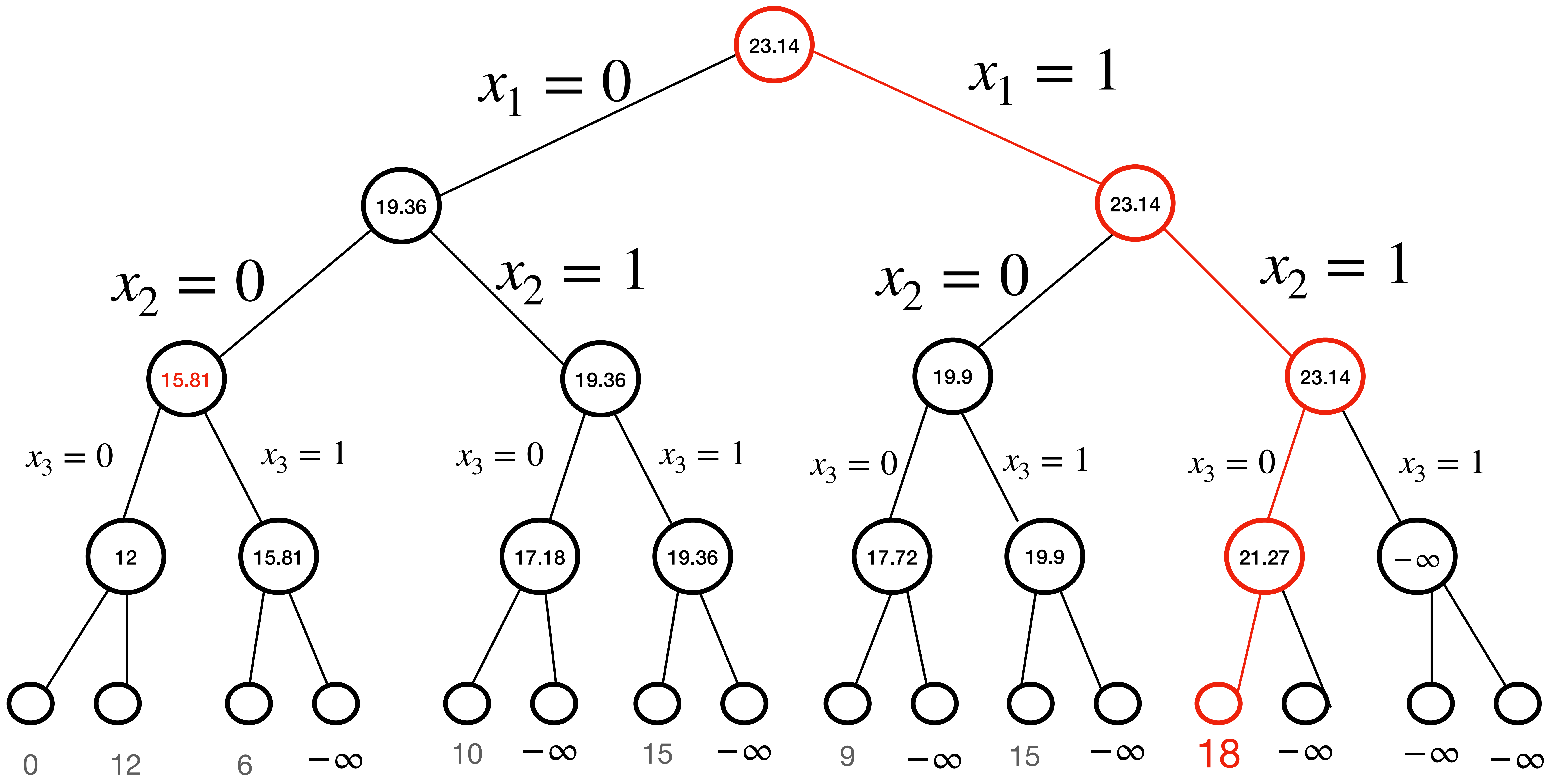
Weight capacity K=25

item i	weight w_i	value v_i
1	9	9
2	10	9
3	7	6
4	22	12

Any feasible solution has profit at most 15.81 < 18

Knapsack problem

$$x_i = \begin{cases} 1 \text{ if item i is chosen} \\ 0 \text{ otherwise} \end{cases}$$



Weight capacity K=25

item i	weight w_i	value v_i
1	9	9
2	10	9
3	7	6
4	22	12

Branch & Bound

- Can define different order of decision
- Can design other bounds according to the problem

- In the worst case, we enumerate all solutions
- It depends on the current feasible solution

- Can be used for problems with more than 2 choices

Exercises

Binary Search Tree

- Insert the following elements in a Binary Search Tree in the given order and draw the tree

12, 80, 65, 23, 72, 70, insert_root(47), 11, 87, 61, 54, 19, 44, 3, 27

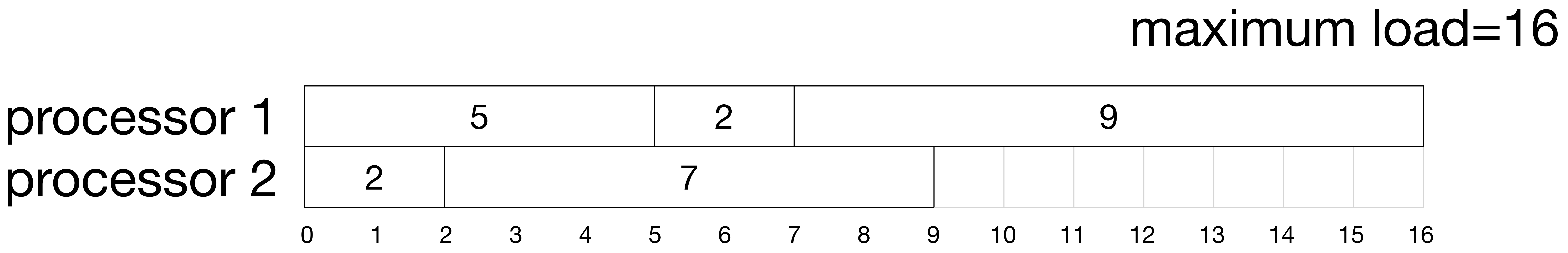
- What is the height of the constructed tree?
- What is the average time (average path from the root) for accessing each element of the tree?

Minimize maximum load on two processors

We consider the following scheduling problem. Given a set of n jobs. Each job i has processing time p_i, all available at the beginning.

The goal is to assign jobs on two processors such that the maximum load is minimized.

Example of 5 jobs with processing times $\{5,2,7,2,9\}$

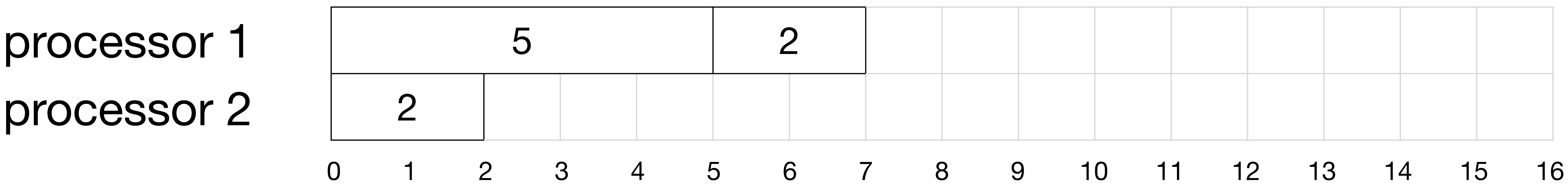


Design a Branch & Bound algorithm to solve this scheduling problem

Lower bound 1

Two lower bounds exist for this problem. The load of any solution should have at least the maximum processing time $\max_i p_i$

Example of 5 jobs with processing times $\{5,2,7,2,9\}$



Jobs $\{5,2,2\}$ are already assigned to processors

The maximum processing time of the remaining jobs is 9. So any feasible solution should have a load at least

load(processor 1) + 9 = 16

load(processor 2) + 9 = 11

Take the minimum

Any feasible solution should have at least the same cost as current solution

$$\max_i \{\text{load}(\text{processor } i)\} = 7$$

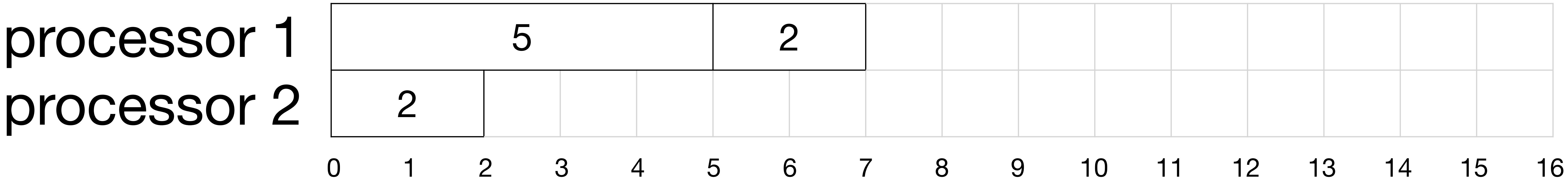
Take the maximum = 11

Lower bound 2

All processors has the same load (ideal situation)

$\frac{1}{m}\sum_{i=1}^{n} p_i$ where m is the number of machines

Example of 5 jobs with processing times $\{5,2,7,2,9\}$



$$LB2 = \max\left\{\frac{5+2+7+2+9}{2} = 12.5\ ,\ \max_i\{\text{load(processor } i)\}\right\}$$

Exercise Branch & Bound

Formal Lower bounds

$$LB1 = \max\left\{ \min_i \left\{ \text{load(processor } i) + \max_i p_i \right\} \ , \ \max_i \{\text{load(processor } i)\} \right\}$$

$$LB2 = \max\left\{ \frac{1}{m}\sum_{i=1}^{n} p_i \ , \ \max_i \{\text{load(processor } i)\} \right\}$$

where $\max_i \{\text{load(processor } i)\}$ is the load of the current solution

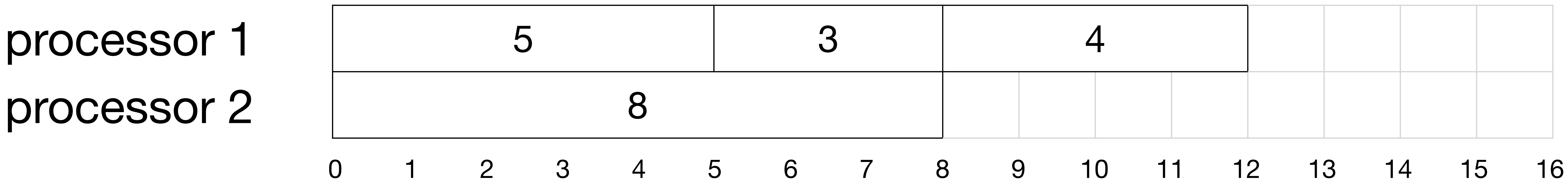
Exercise Branch & Bound

Formal Lower bounds

Use $LB = \max\{LB1,\ LB2\}$ for the design of a Branch & Bound algorithm

$E = \{5,3,8,4\}$ and $m = 2$

Start from the following feasible solution: $x_1 = 1,\ x_2 = 1,\ x_3 = 0,\ x_4 = 1$



1. Complete the tree of the Branch & Bound, and find the optimal solution
2. Give the order of the nodes you explored

$$E = \{5,3,8,4\}$$

$$x_i = \begin{cases} 1 \text{ if job i is assigned to processor 1} \\ 0 \text{ otherwise (if job i is assigned to processor 2)} \end{cases}$$