Algoritmi in podatkovne strukture 1

8. laboratorijske vaje

Programerske izpitne naloge

NALOGA 1

Dan je enosmerni seznam s kazalci. Sestavi algoritem, ki za dani seznam poišče srednji element, če je število elementov liho, oziroma element, ki se nahaja tik pred sredino seznama, če je število elementov sodo.

Izberi ustrezne parametre in oceni časovno zahtevnost svojega algoritma.

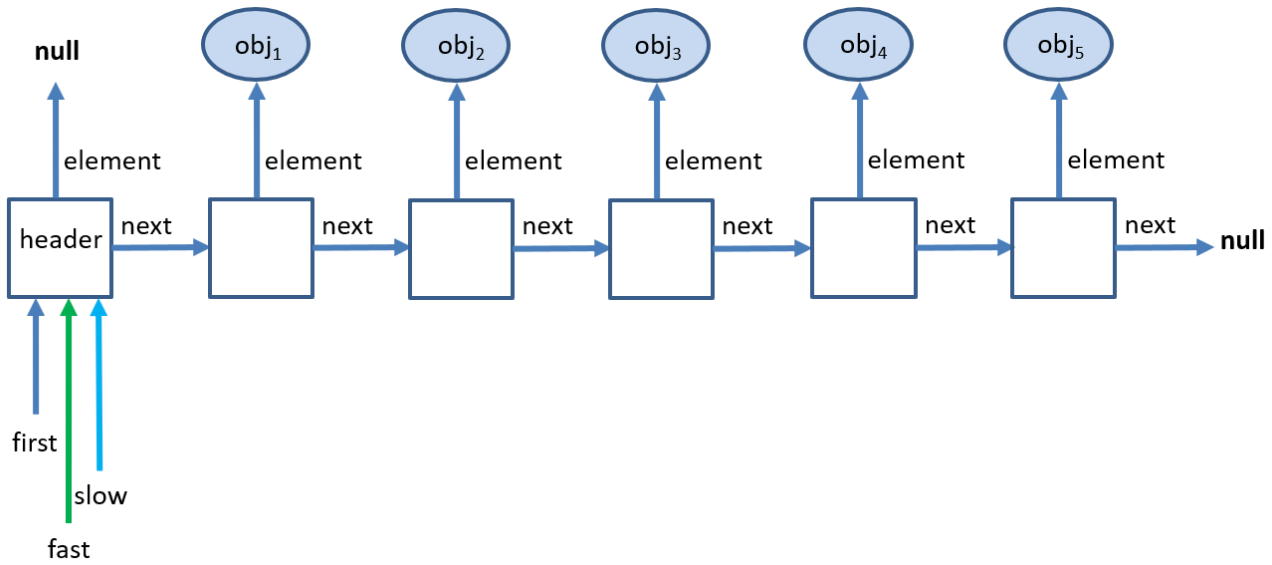
REŠITEV

```
// Metoda razreda LinkedList

public Object median() {
    ListNode slow = first();
    ListNode fast = first();

    while (!overEnd(fast)) {
        fast = next(fast);

        if (!overEnd(fast))
            fast = next(fast);

        if (!overEnd(fast))
            slow = next(slow);
    }

    if (!overEnd(slow))
        return retrieve(slow);
    else
        return null;
}
```

Časovna kompleksnost: O(n), n je število elementov v seznamu.

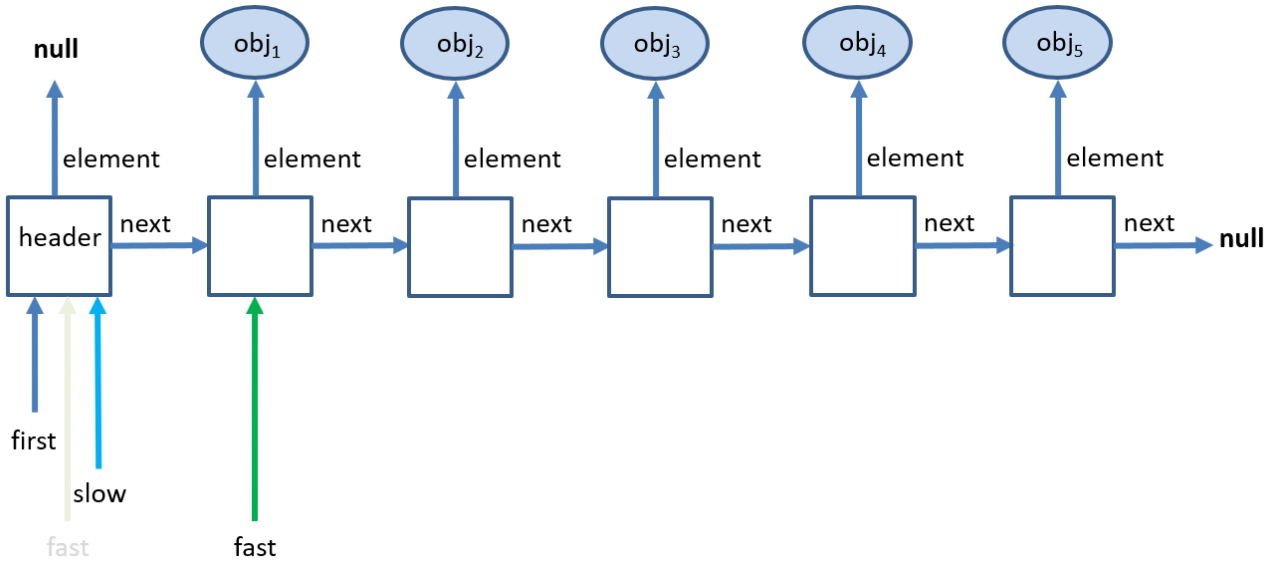
REŠITEV

```
// Metoda razreda LinkedList

public Object median() {
    ListNode slow = first();
    ListNode fast = first();

    while (!overEnd(fast)) {
        fast = next(fast);

        if (!overEnd(fast))
            fast = next(fast);

        if (!overEnd(fast))
            slow = next(slow);
    }

    if (!overEnd(slow))
        return retrieve(slow);
    else
        return null;
}
```



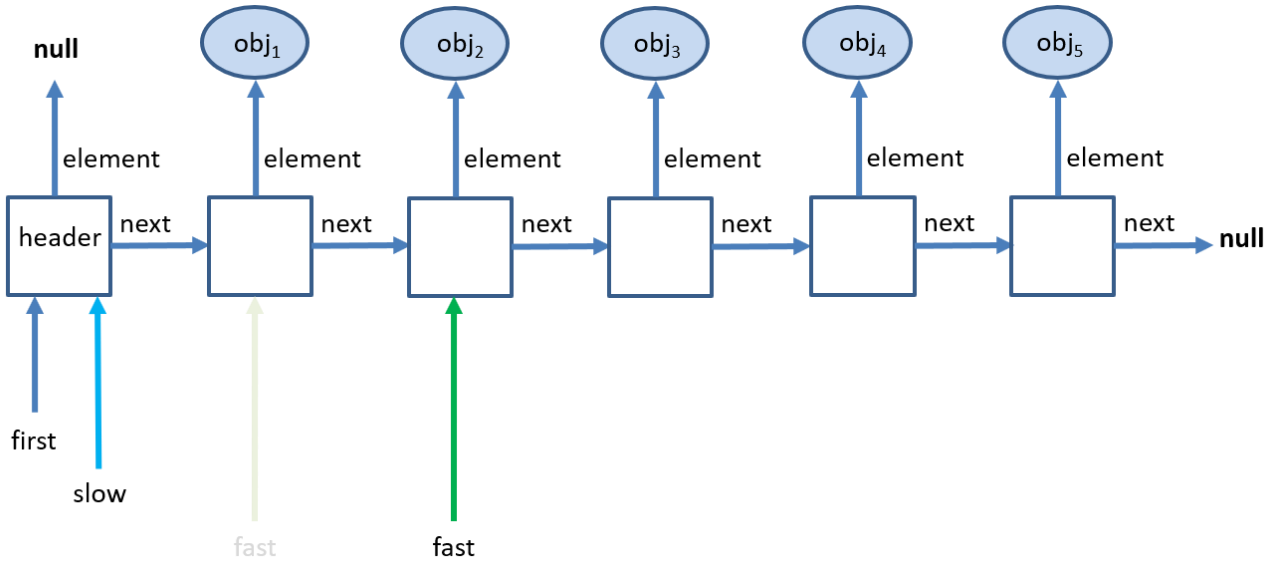
REŠITEV

```
// Metoda razreda LinkedList

public Object median() {
    ListNode slow = first();
    ListNode fast = first();

    while (!overEnd(fast)) {
        fast = next(fast);

        if (!overEnd(fast))
            fast = next(fast);

        if (!overEnd(fast))
            slow = next(slow);
    }

    if (!overEnd(slow))
        return retrieve(slow);
    else
        return null;
}
```



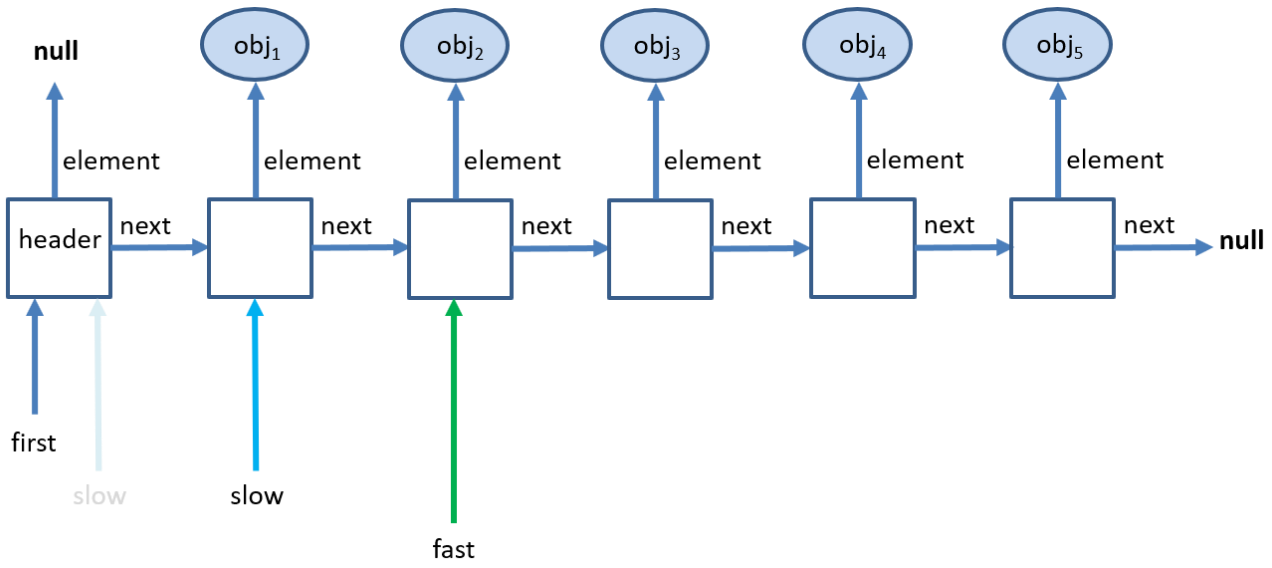
REŠITEV

```
// Metoda razreda LinkedList

public Object median() {
    ListNode slow = first();
    ListNode fast = first();

    while (!overEnd(fast)) {
        fast = next(fast);

        if (!overEnd(fast))
            fast = next(fast);

        if (!overEnd(fast))
            slow = next(slow);
    }

    if (!overEnd(slow))
        return retrieve(slow);
    else
        return null;
}
```



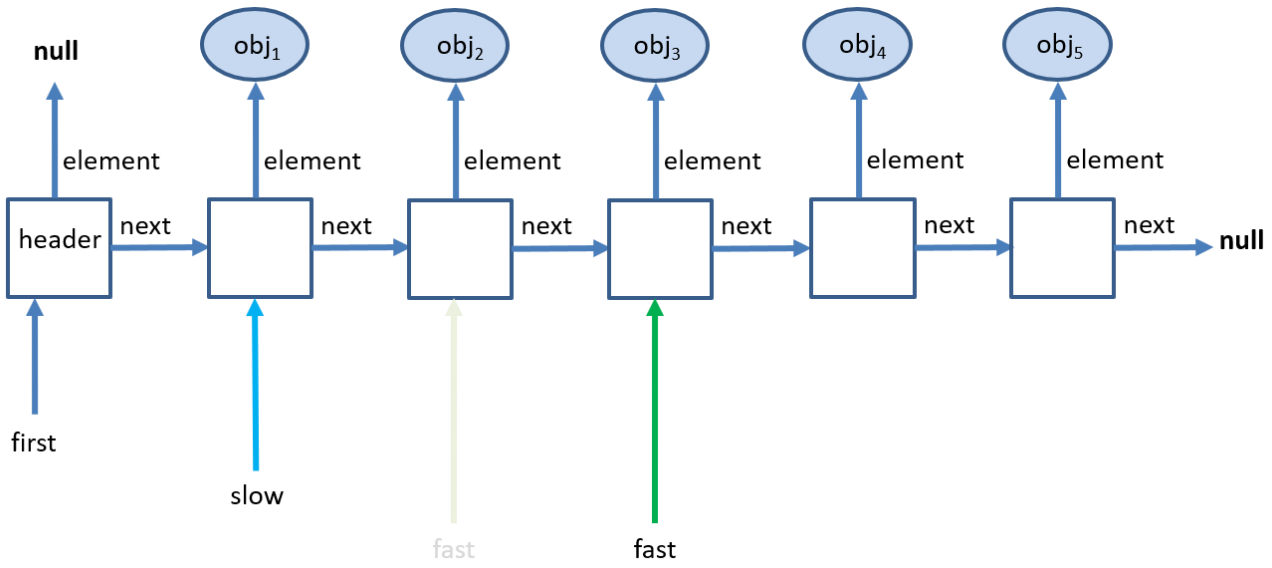
REŠITEV

```
// Metoda razreda LinkedList

public Object median() {
    ListNode slow = first();
    ListNode fast = first();

    while (!overEnd(fast)) {
        fast = next(fast);

        if (!overEnd(fast))
            fast = next(fast);

        if (!overEnd(fast))
            slow = next(slow);
    }

    if (!overEnd(slow))
        return retrieve(slow);
    else
        return null;
}
```



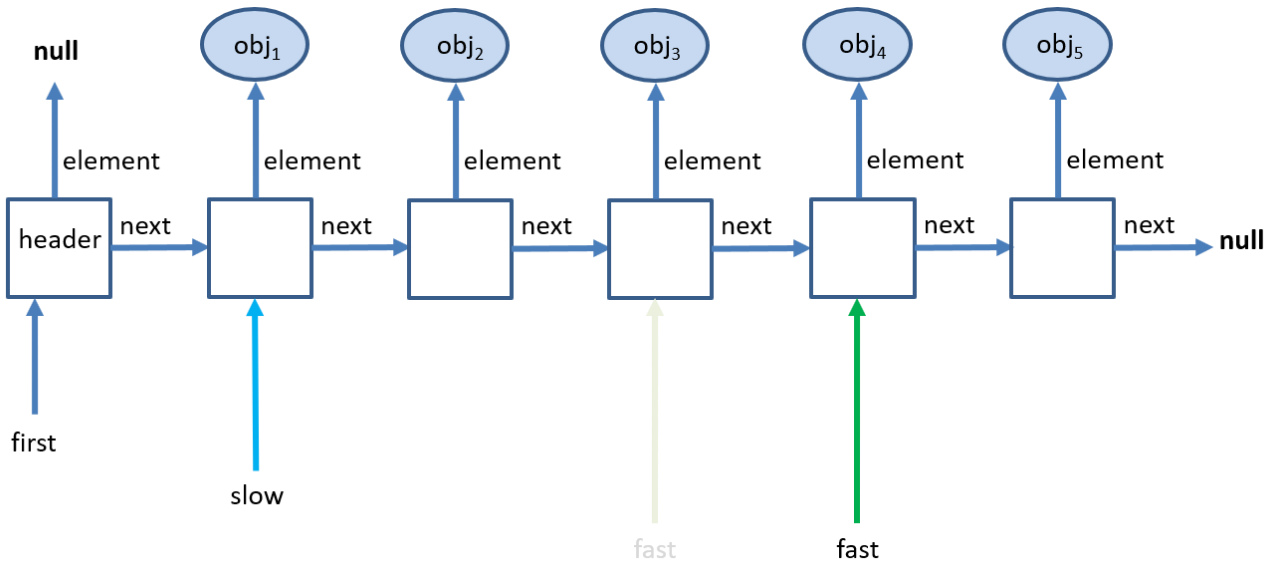
REŠITEV

```
// Metoda razreda LinkedList

public Object median() {
    ListNode slow = first();
    ListNode fast = first();

    while (!overEnd(fast)) {
        fast = next(fast);

        if (!overEnd(fast))
            fast = next(fast);

        if (!overEnd(fast))
            slow = next(slow);
    }

    if (!overEnd(slow))
        return retrieve(slow);
    else
        return null;
}
```



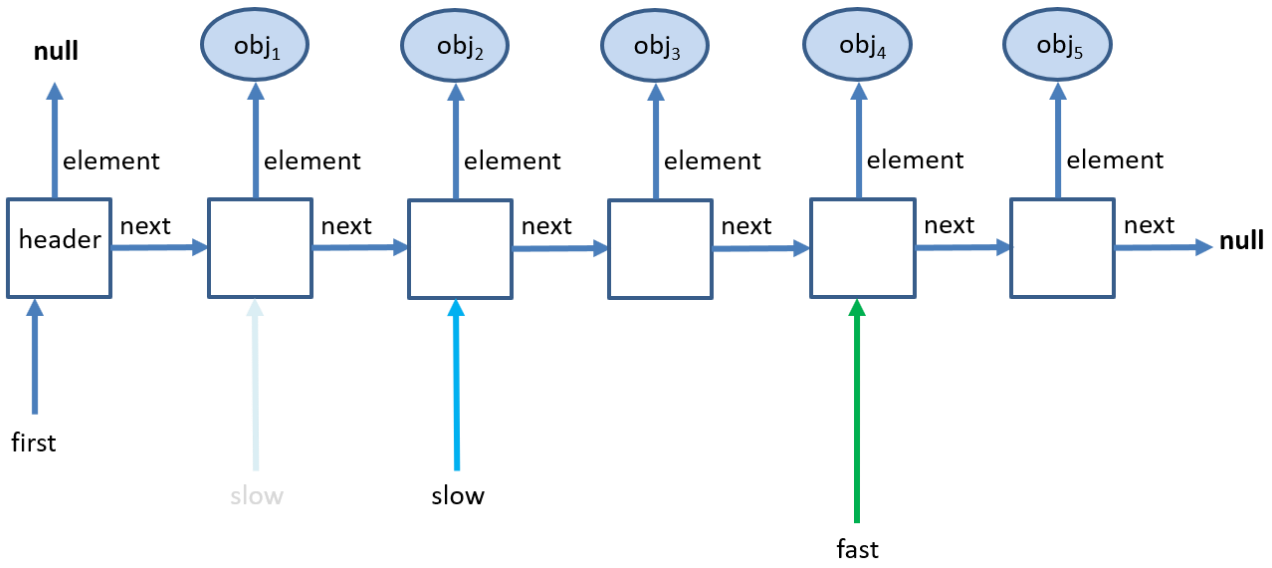
REŠITEV

```
// Metoda razreda LinkedList

public Object median() {
    ListNode slow = first();
    ListNode fast = first();

    while (!overEnd(fast)) {
        fast = next(fast);

        if (!overEnd(fast))
            fast = next(fast);

        if (!overEnd(fast))
            slow = next(slow);
    }

    if (!overEnd(slow))
        return retrieve(slow);
    else
        return null;
}
```



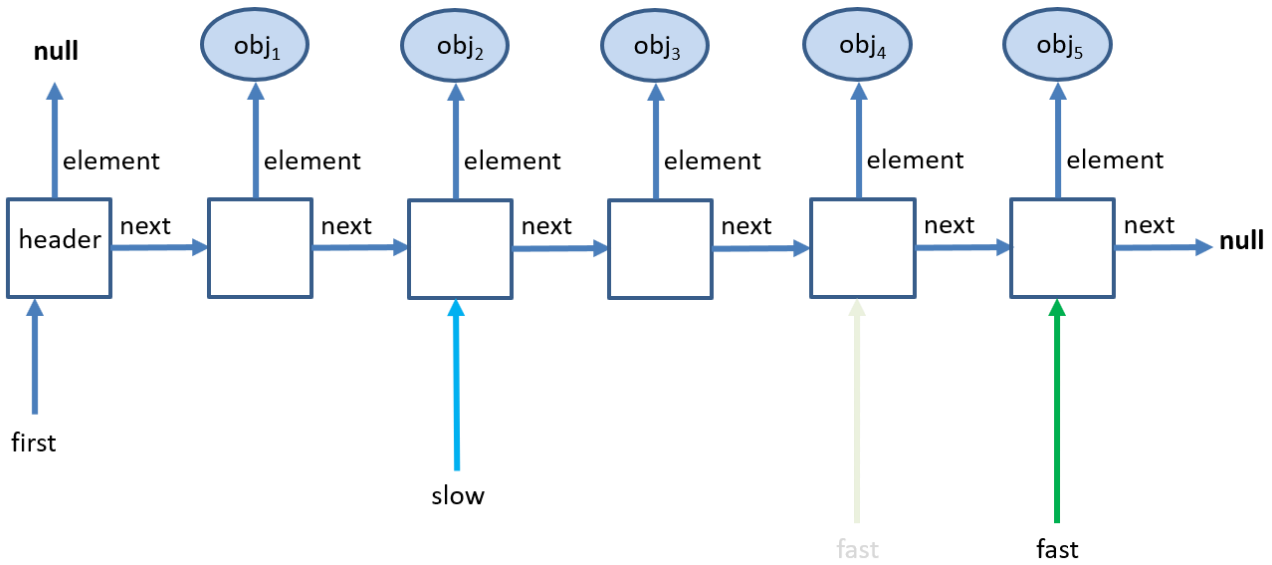
REŠITEV

```
// Metoda razreda LinkedList

public Object median() {
    ListNode slow = first();
    ListNode fast = first();

    while (!overEnd(fast)) {
        fast = next(fast);

        if (!overEnd(fast))
            fast = next(fast);

        if (!overEnd(fast))
            slow = next(slow);
    }

    if (!overEnd(slow))
        return retrieve(slow);
    else
        return null;
}
```



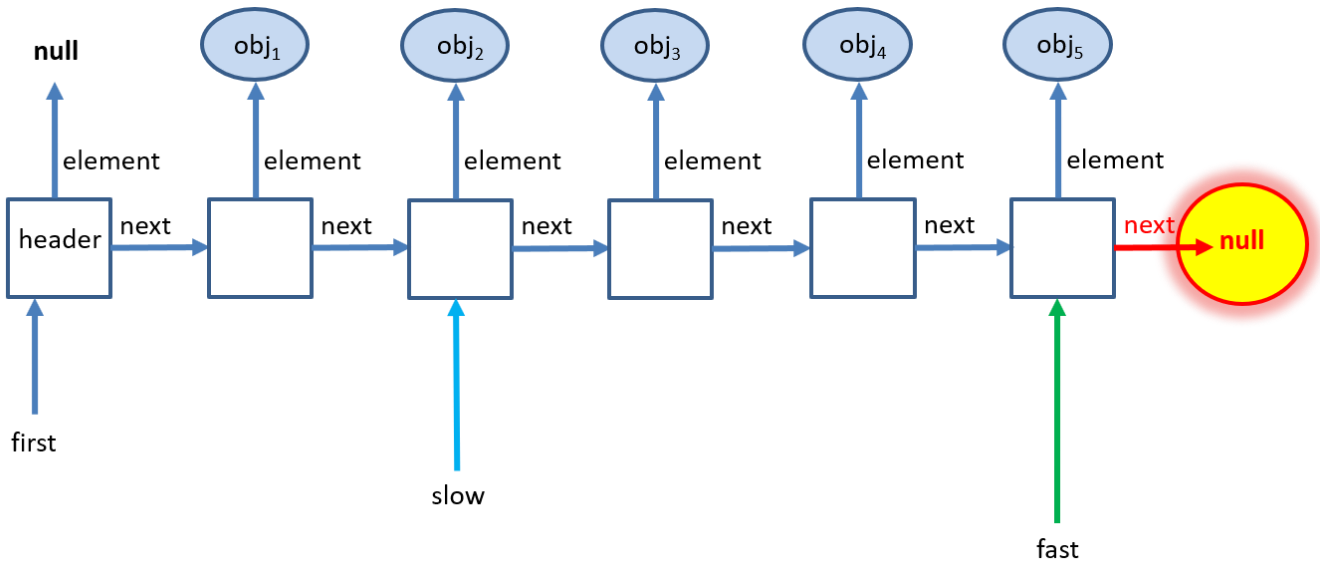
REŠITEV

```
// Metoda razreda LinkedList

public Object median() {
    ListNode slow = first();
    ListNode fast = first();

    while (!overEnd(fast)) {
        fast = next(fast);

        if (!overEnd(fast))
            fast = next(fast);

        if (!overEnd(fast))
            slow = next(slow);
    }

    if (!overEnd(slow))
        return retrieve(slow);
    else
        return null;
}
```



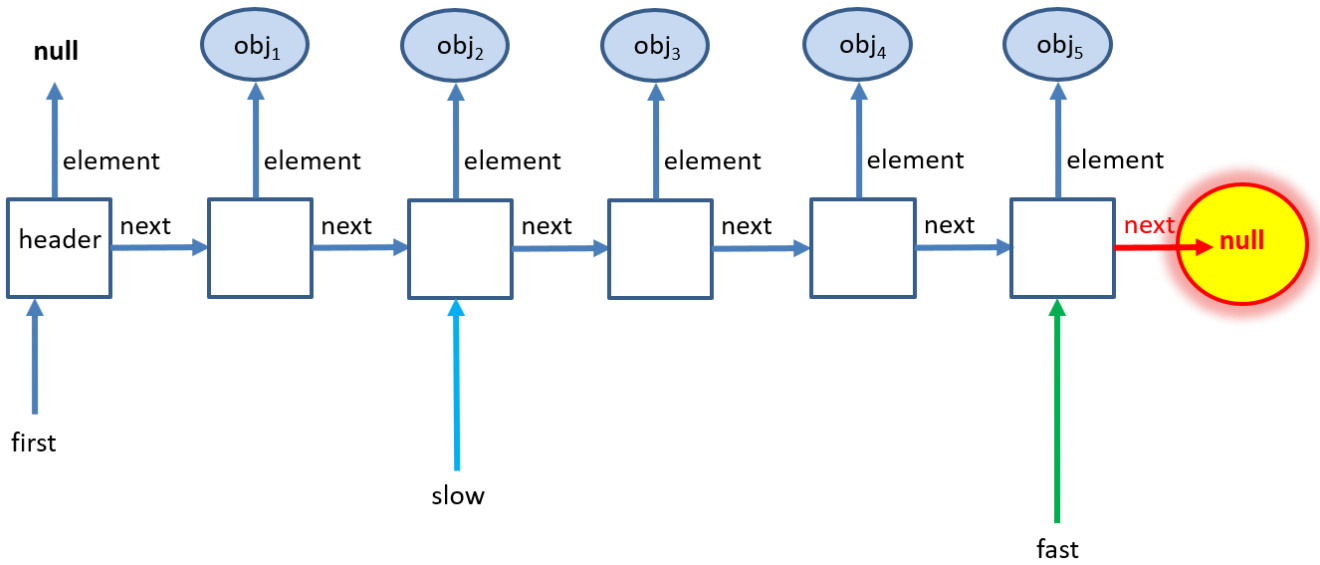
REŠITEV

```
// Metoda razreda LinkedList

public Object median() {
    ListNode slow = first();
    ListNode fast = first();

    while (!overEnd(fast)) {
        fast = next(fast);

        if (!overEnd(fast))
            fast = next(fast);

        if (!overEnd(fast))
            slow = next(slow);
    }

    if (!overEnd(slow))
        return retrieve(slow);
    else
        return null;
}
```



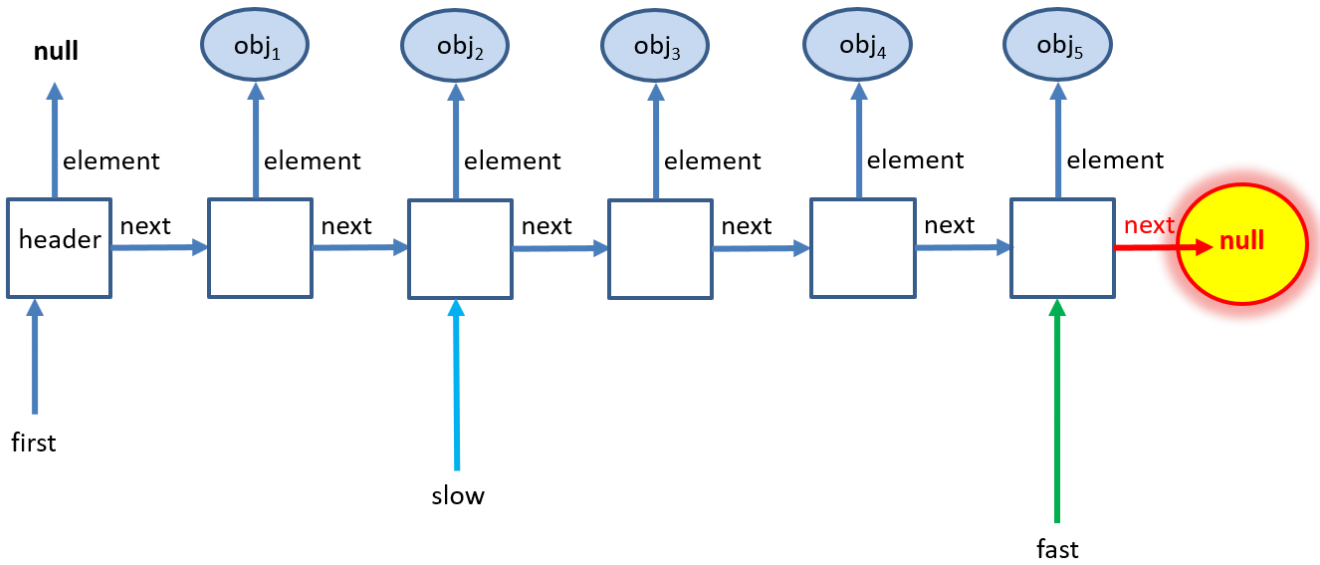
REŠITEV

```
// Metoda razreda LinkedList

public Object median() {
    ListNode slow = first();
    ListNode fast = first();

    while (!overEnd(fast)) {
        fast = next(fast);

        if (!overEnd(fast))
            fast = next(fast);

        if (!overEnd(fast))
            slow = next(slow);
    }

    if (!overEnd(slow))
        return retrieve(slow);
    else
        return null;
}
```



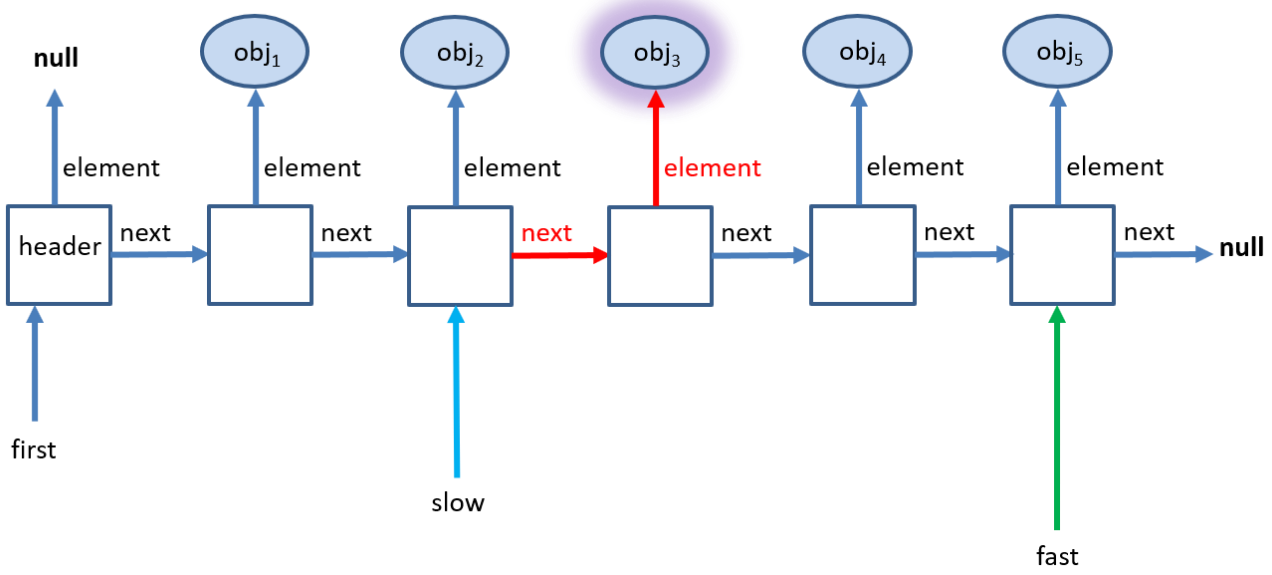
REŠITEV

```
// Metoda razreda LinkedList

public Object median() {
    ListNode slow = first();
    ListNode fast = first();

    while (!overEnd(fast)) {
        fast = next(fast);

        if (!overEnd(fast))
            fast = next(fast);

        if (!overEnd(fast))
            slow = next(slow);
    }

    if (!overEnd(slow))
        return retrieve(slow);
    else
        return null;
}
```



REŠITEV

```
// Metoda razreda LinkedList

public Object median() {
    ListNode slow = first();
    ListNode fast = first();

    while (!overEnd(fast)) {
        fast = next(fast);

        if (!overEnd(fast))
            fast = next(fast);

        if (!overEnd(fast))
            slow = next(slow);
    }

    if (!overEnd(slow))
        return retrieve(slow);
    else
        return null;
}
```



NALOGA 2

Dan je enosmerni seznam s kazalci.

Sestavi algoritem, ki bo izpisoval elemente izmenoma z začetka in konca seznama, torej tako, da se najprej izpiše prvi element, zatem zadnji, zatem drugi, zatem predzadnji itd.

Izberi ustrezne parametre in oceni časovno zahtevnost svojega algoritma.

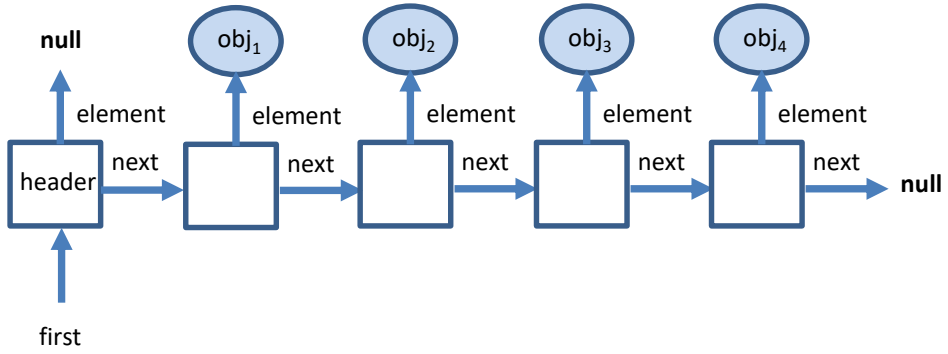
REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```

Časovna kompleksnost: $O(n^2)$, n je število elementov v seznamu.

REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```

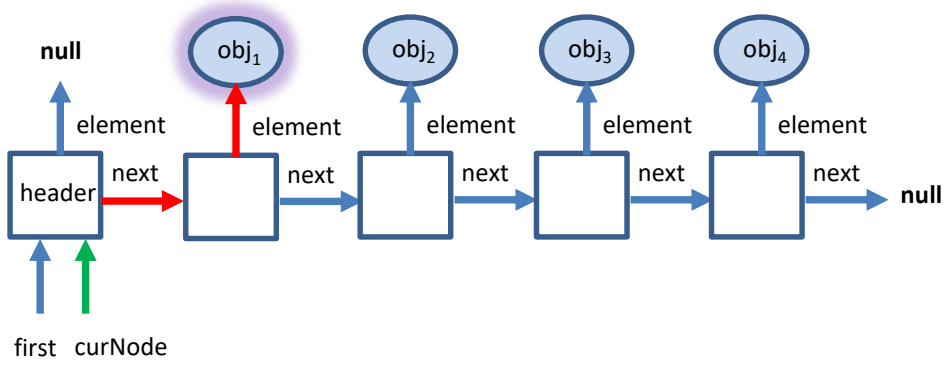


n = 4

Izpis:

REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```

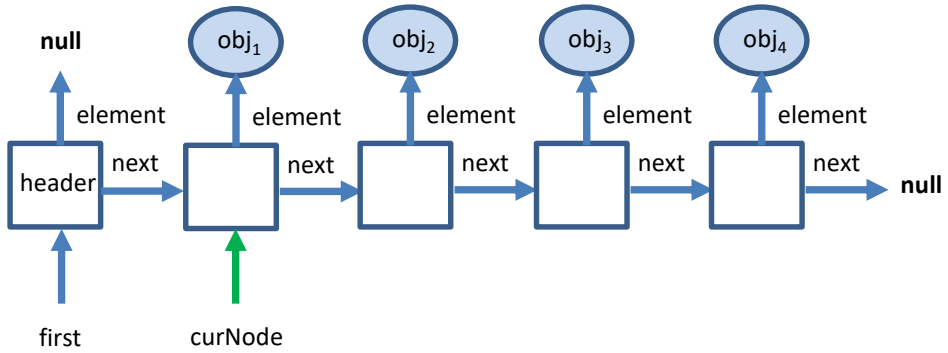


n = 4

Izpis: obj_1,

REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```

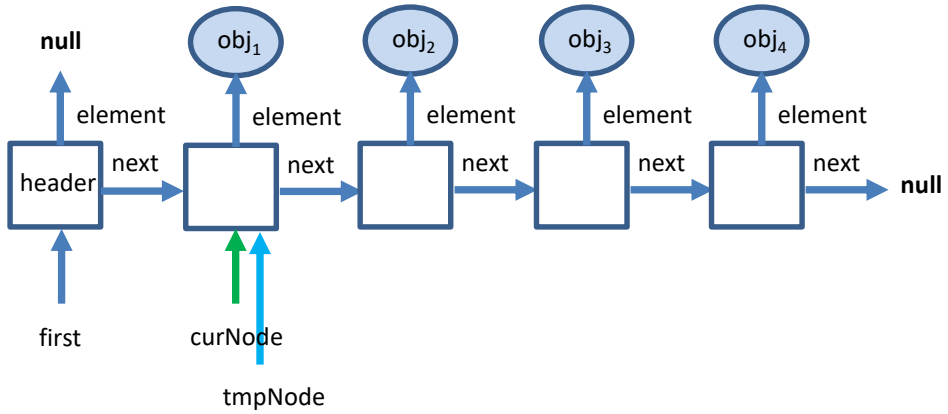


n = 4 → 3

Izpis: obj_1,

REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```

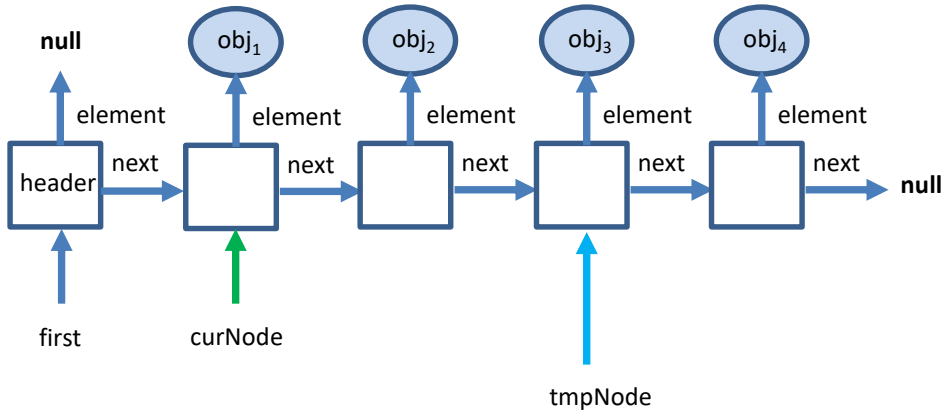


n = 3

Izpis: obj_1,

REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```

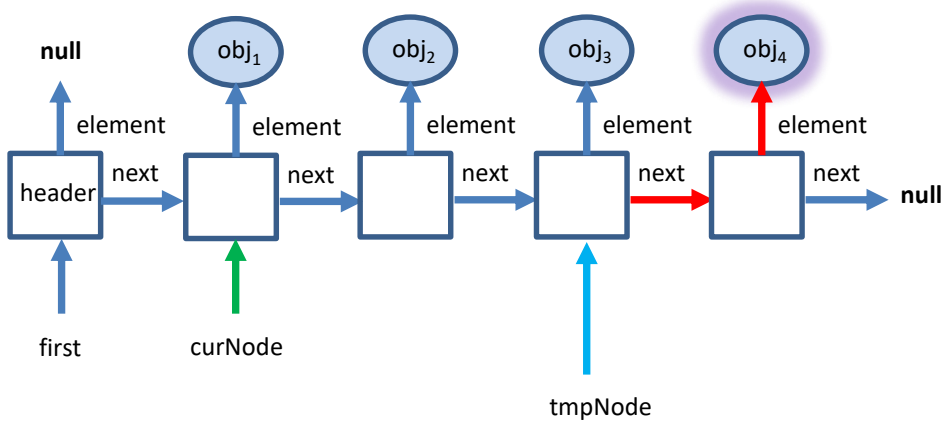


n = 3

Izpis: obj_1,

REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```

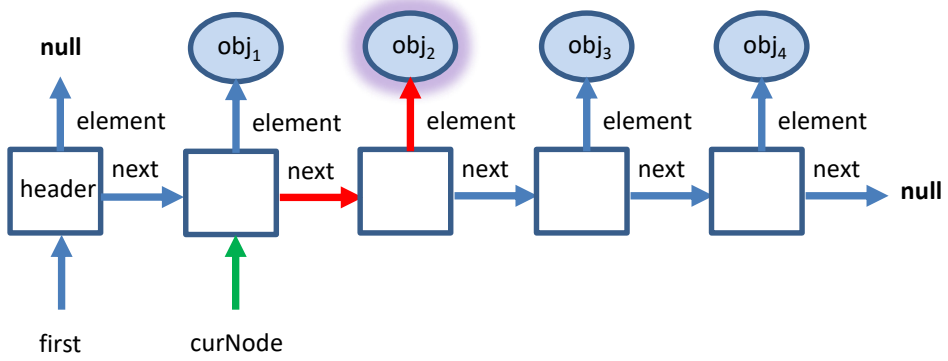


n = 3 → 2

Izpis: obj_1, obj_4,

REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```

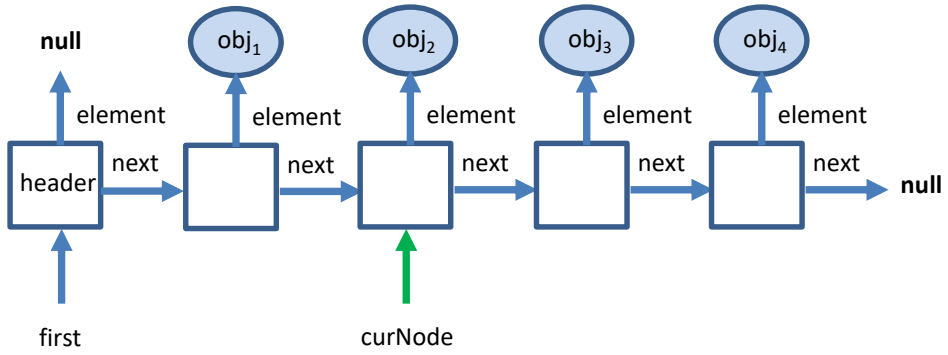


n = 2

Izpis: obj_1, obj_4, obj_2,

REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```

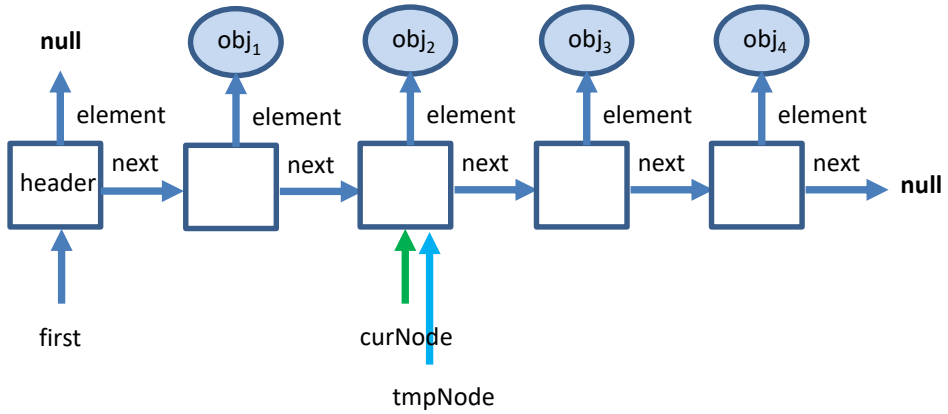


n = 2 → 1

Izpis: obj_1, obj_4, obj_2,

REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```

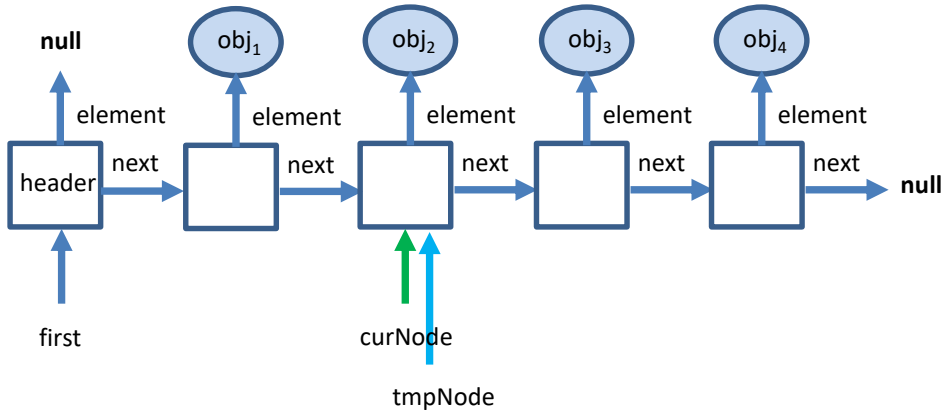


n = 1

Izpis: obj_1, obj_4, obj_2,

REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```

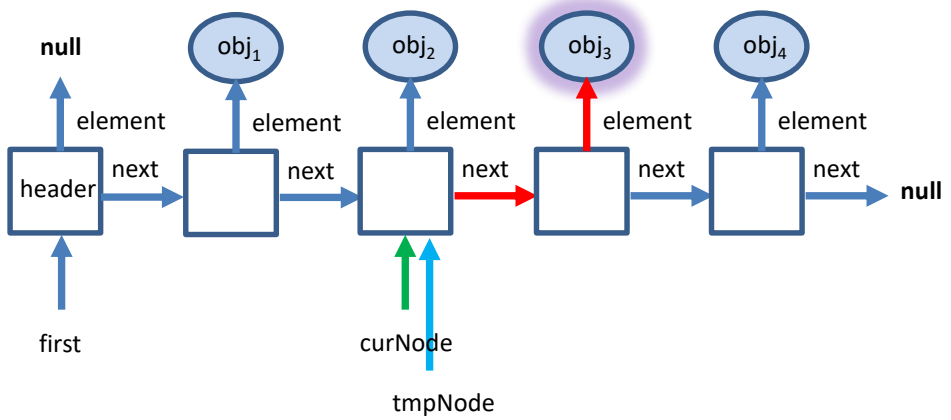


n = 1

Izpis: obj_1, obj_4, obj_2,

REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```

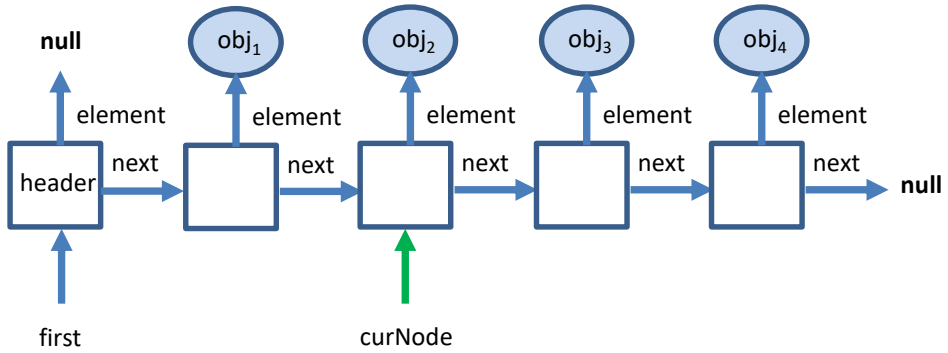


n = 1 → 0

Izpis: obj_1, obj_4, obj_2, obj_3,

REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```

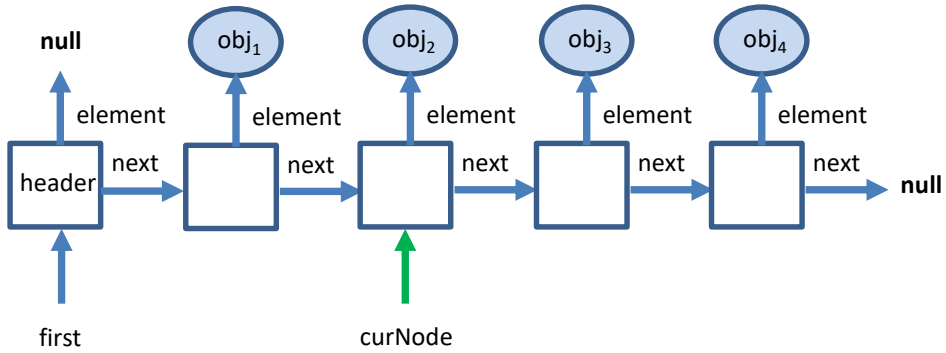


n = 0

Izpis: obj_1, obj_4, obj_2, obj_3,

REŠITEV (NEOPTIMALNA)

```
// Metoda razreda LinkedList
public void printAltKvadraticno() {
    ListNode curNode;
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode))
        n++;

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            ListNode tmpNode = curNode;

            for (int i = 1; i < n; i++)
                tmpNode = next(tmpNode);

            System.out.print(retrieve(tmpNode) + ",");
            n--;
        }
    }

    System.out.println();
}
```



n = 0

Izpis: obj_1, obj_4, obj_2, obj_3,

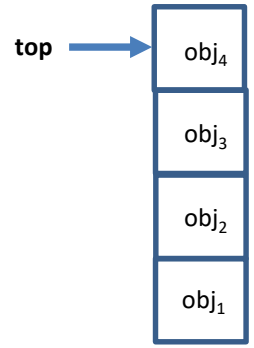
REŠITEV (LINEARNA KOMPLEKSNOST)

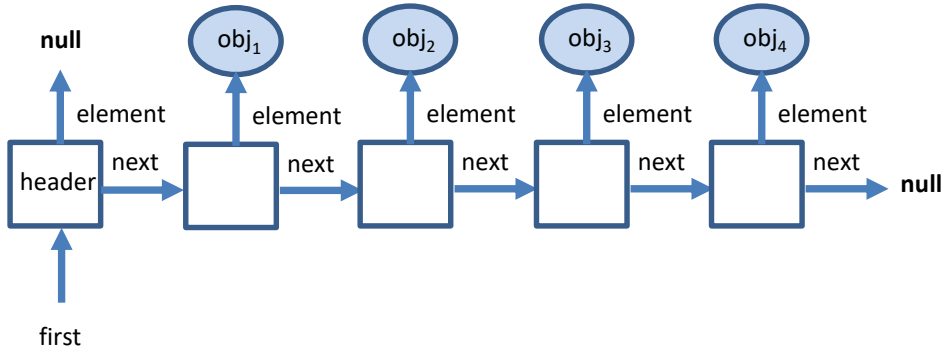
```
// Metoda razreda LinkedList
public void printAlt() {
    ListNode curNode;
    Stack stack = new Stack();
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode)) {
        stack.push(retrieve(curNode));
        n++;
    }

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            System.out.print(stack.top() + ",");
            stack.pop();
            n--;
        }
    }

    System.out.println();
}
```

Časovna kompleksnost: O(n), n je število elementov v seznamu.

REŠITEV (LINEARNA KOMPLEKSNOST)

```
// Metoda razreda LinkedList
public void printAlt() {
    ListNode curNode;
    Stack stack = new Stack();
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode)) {
        stack.push(retrieve(curNode));
        n++;
    }

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            System.out.print(stack.top() + ",");
            stack.pop();
            n--;
        }
    }

    System.out.println();
}
```



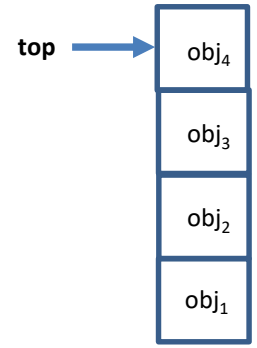


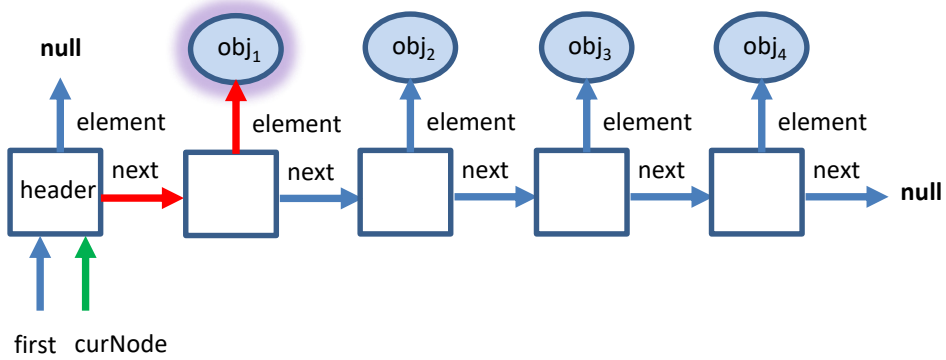
n = 4

Izpis:

REŠITEV (LINEARNA KOMPLEKSNOST)

```
// Metoda razreda LinkedList
public void printAlt() {
    ListNode curNode;
    Stack stack = new Stack();
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode)) {
        stack.push(retrieve(curNode));
        n++;
    }

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            System.out.print(stack.top() + ",");
            stack.pop();
            n--;
        }
    }

    System.out.println();
}
```



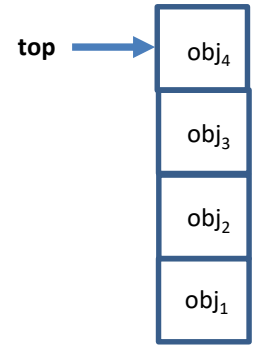


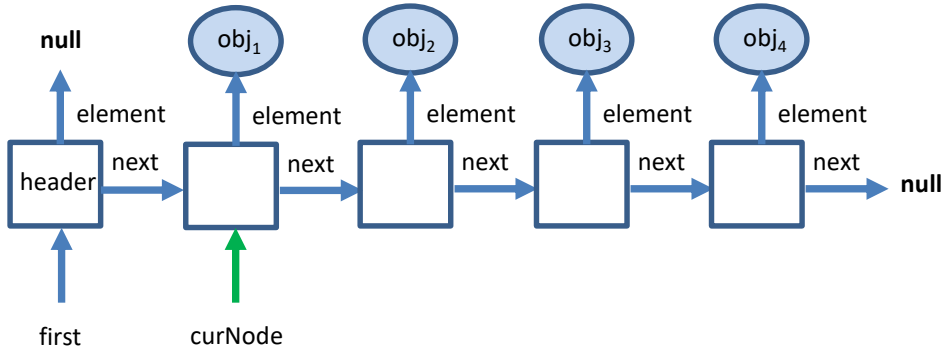
n = 4

Izpis: obj_1,

REŠITEV (LINEARNA KOMPLEKSNOST)

```
// Metoda razreda LinkedList
public void printAlt() {
    ListNode curNode;
    Stack stack = new Stack();
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode)) {
        stack.push(retrieve(curNode));
        n++;
    }

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            System.out.print(stack.top() + ",");
            stack.pop();
            n--;
        }
    }

    System.out.println();
}
```



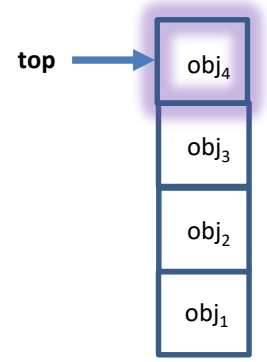


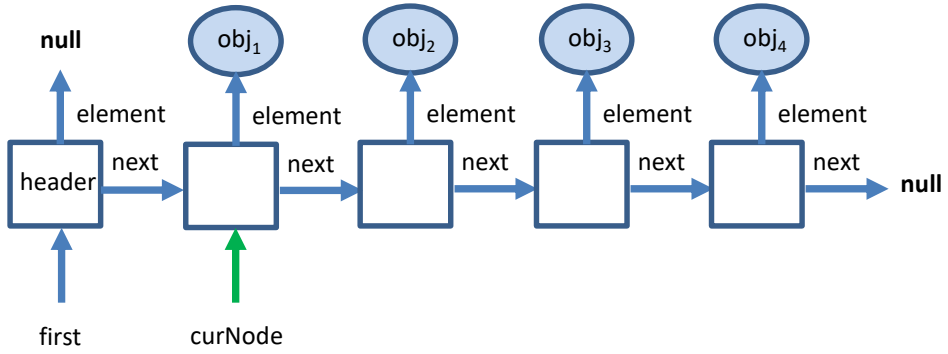
n = 4 → 3

Izpis: obj_1,

REŠITEV (LINEARNA KOMPLEKSNOST)

```
// Metoda razreda LinkedList
public void printAlt() {
    ListNode curNode;
    Stack stack = new Stack();
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode)) {
        stack.push(retrieve(curNode));
        n++;
    }

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            System.out.print(stack.top() + ",");
            stack.pop();
            n--;
        }
    }

    System.out.println();
}
```



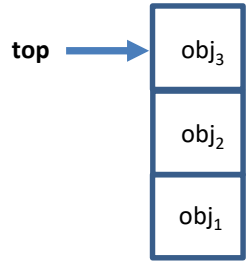


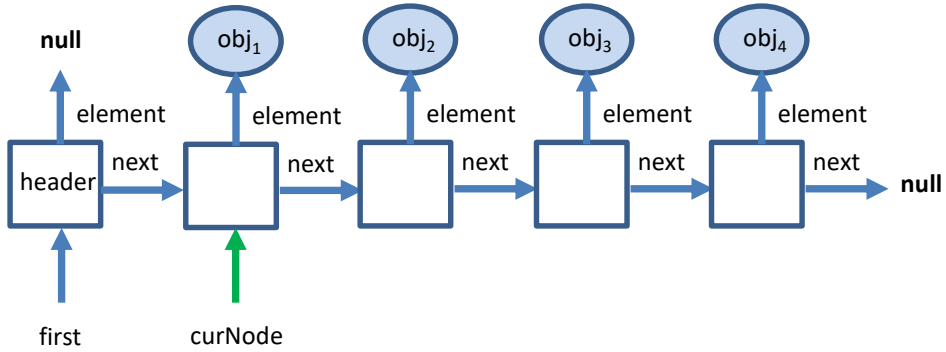
n = 3

Izpis: obj_1, obj_4,

REŠITEV (LINEARNA KOMPLEKSNOST)

```
// Metoda razreda LinkedList
public void printAlt() {
    ListNode curNode;
    Stack stack = new Stack();
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode)) {
        stack.push(retrieve(curNode));
        n++;
    }

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            System.out.print(stack.top() + ",");
            stack.pop();
            n--;
        }
    }

    System.out.println();
}
```



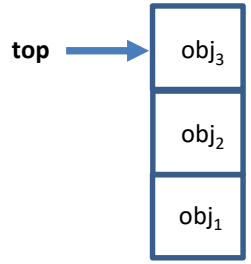


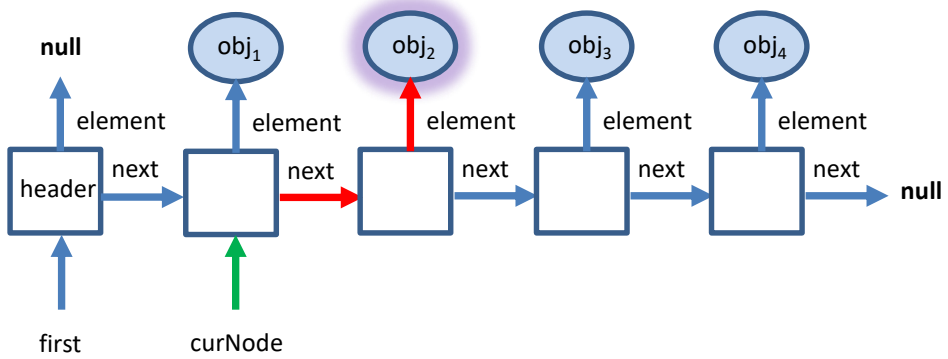
n = 3 → 2

Izpis: obj_1, obj_4,

REŠITEV (LINEARNA KOMPLEKSNOST)

```
// Metoda razreda LinkedList
public void printAlt() {
    ListNode curNode;
    Stack stack = new Stack();
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode)) {
        stack.push(retrieve(curNode));
        n++;
    }

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            System.out.print(stack.top() + ",");
            stack.pop();
            n--;
        }
    }

    System.out.println();
}
```



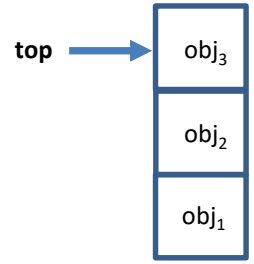


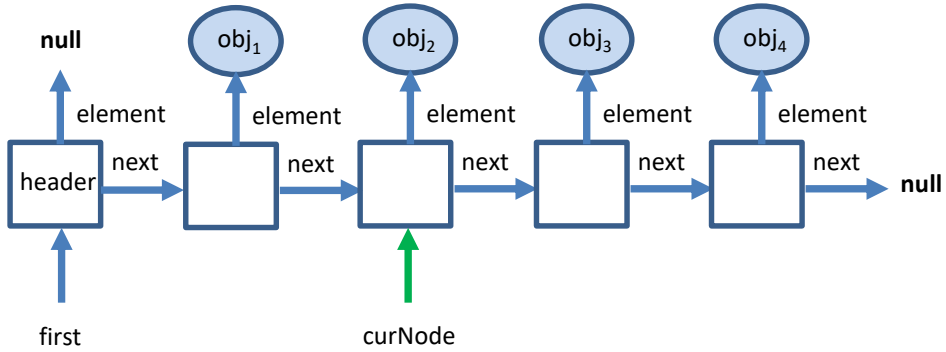
n = 2

Izpis: obj_1, obj_4, obj_2,

REŠITEV (LINEARNA KOMPLEKSNOST)

```
// Metoda razreda LinkedList
public void printAlt() {
    ListNode curNode;
    Stack stack = new Stack();
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode)) {
        stack.push(retrieve(curNode));
        n++;
    }

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            System.out.print(stack.top() + ",");
            stack.pop();
            n--;
        }
    }

    System.out.println();
}
```



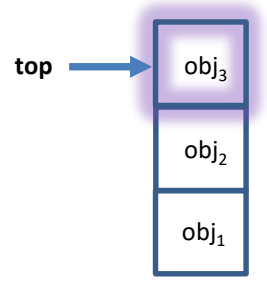


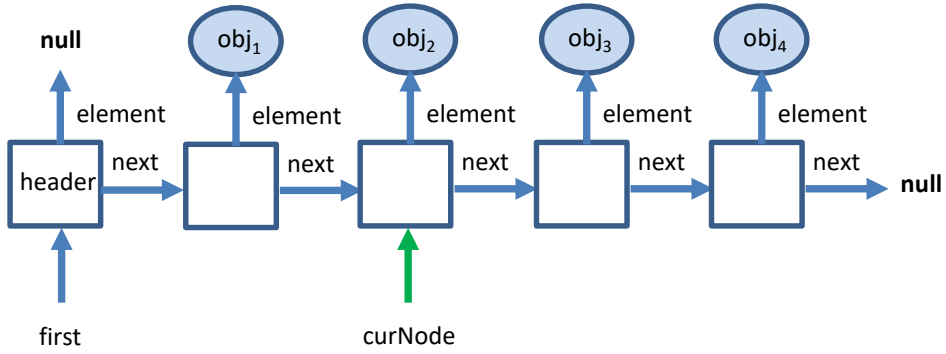
n = 2 → 1

Izpis: obj_1, obj_4, obj_2,

REŠITEV (LINEARNA KOMPLEKSNOST)

```
// Metoda razreda LinkedList
public void printAlt() {
    ListNode curNode;
    Stack stack = new Stack();
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode)) {
        stack.push(retrieve(curNode));
        n++;
    }

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            System.out.print(stack.top() + ",");
            stack.pop();
            n--;
        }
    }

    System.out.println();
}
```



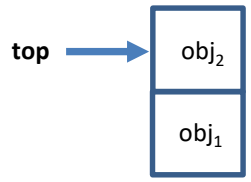


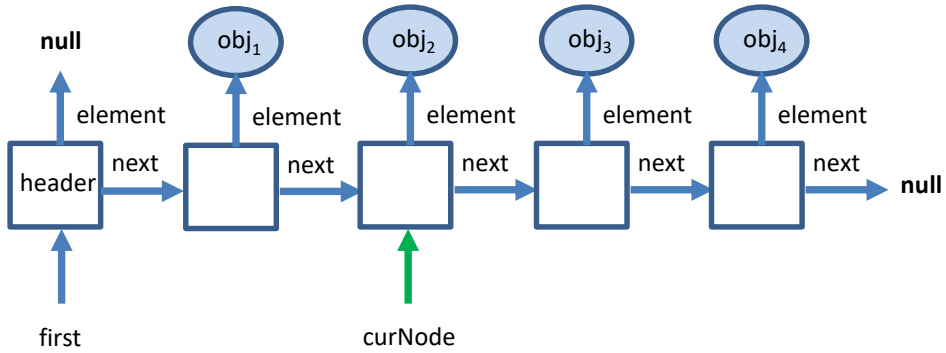
n = 1

Izpis: obj_1, obj_4, obj_2, obj_3,

REŠITEV (LINEARNA KOMPLEKSNOST)

```
// Metoda razreda LinkedList
public void printAlt() {
    ListNode curNode;
    Stack stack = new Stack();
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode)) {
        stack.push(retrieve(curNode));
        n++;
    }

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            System.out.print(stack.top() + ",");
            stack.pop();
            n--;
        }
    }

    System.out.println();
}
```



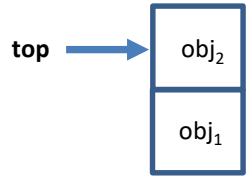


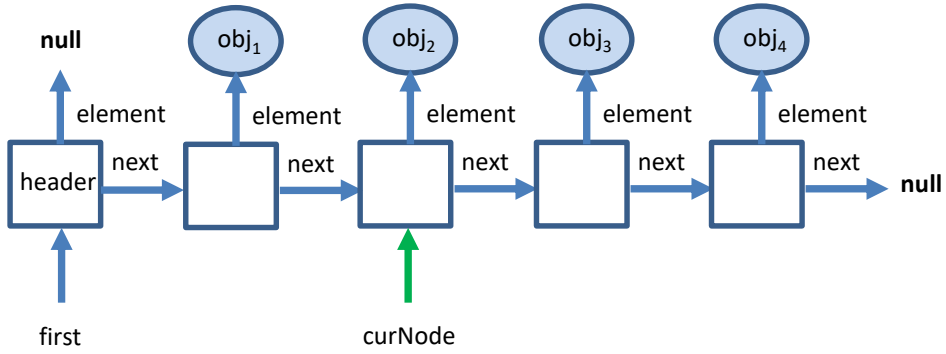
n = 1 → 0

Izpis: obj_1, obj_4, obj_2, obj_3,

REŠITEV (LINEARNA KOMPLEKSNOST)

```
// Metoda razreda LinkedList
public void printAlt() {
    ListNode curNode;
    Stack stack = new Stack();
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode)) {
        stack.push(retrieve(curNode));
        n++;
    }

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            System.out.print(stack.top() + ",");
            stack.pop();
            n--;
        }
    }

    System.out.println();
}
```



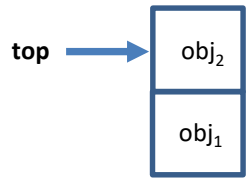


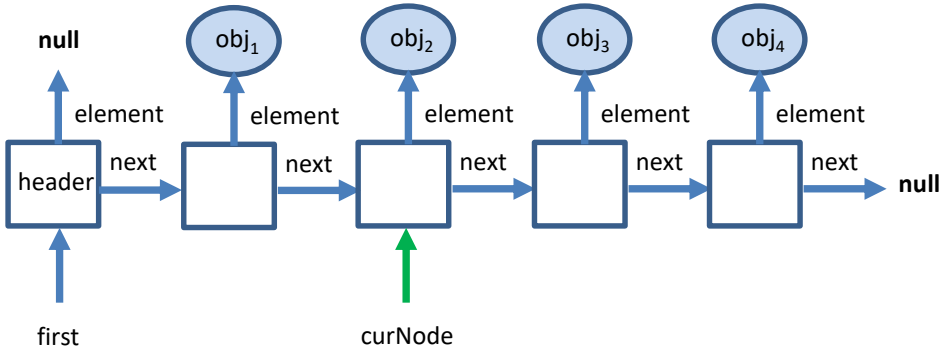
n = 0

Izpis: obj_1, obj_4, obj_2, obj_3,

REŠITEV (LINEARNA KOMPLEKSNOST)

```
// Metoda razreda LinkedList
public void printAlt() {
    ListNode curNode;
    Stack stack = new Stack();
    int n = 0;

    for (curNode = first(); !overEnd(curNode); curNode = next(curNode)) {
        stack.push(retrieve(curNode));
        n++;
    }

    curNode = first();
    while (n > 0) {
        System.out.print(retrieve(curNode) + ",");
        curNode = next(curNode);
        n--;

        if (n > 0) {
            System.out.print(stack.top() + ",");
            stack.pop();
            n--;
        }
    }

    System.out.println();
}
```





n = 0

Izpis: obj_1, obj_4, obj_2, obj_3,

NALOGA 3

Dano je binarno drevo (dostopno preko korena root), ki za vsako vozlišče vsebuje naslednje podatke:

```
class Node {
    int key, diff;
    Node left, right;
}
```

Sestavi algoritem `public int sum(Node root)`, ki bo izračunal vsoto vseh ključev v drevesu, hkrati pa bo v vsakem vozlišču izračunal in nastavil razliko (`diff`) med vsoto ključev desnega in vsoto ključev levega poddrevesa tega vozlišča.

Izberi ustrezne parametre in oceni časovno zahtevnost svojega algoritma.

REŠITEV

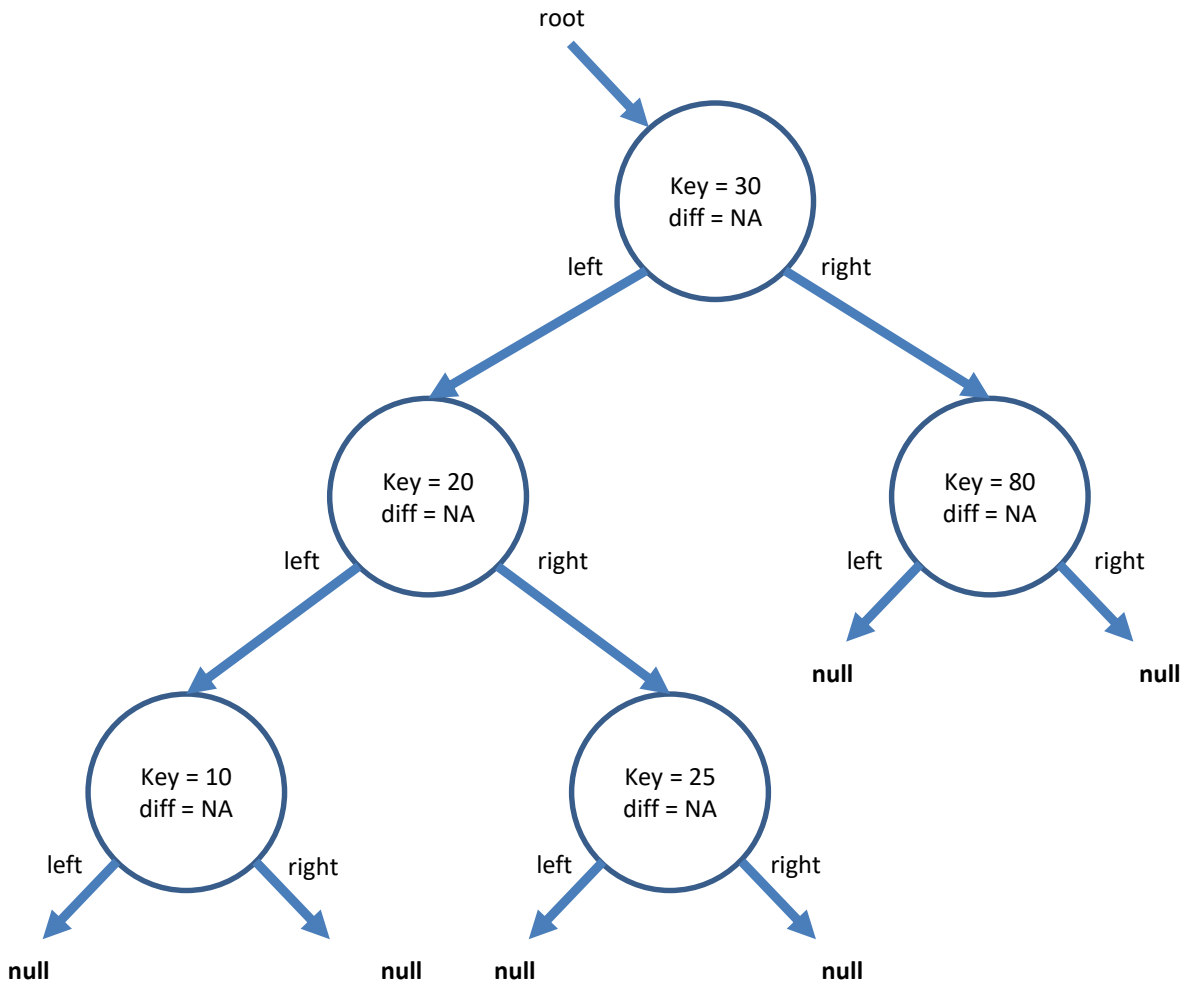
```
class Node {
    int key, diff;
    Node left, right;
}


public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

Časovna kompleksnost: O(n), n je število vozlišč v drevesu.

REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

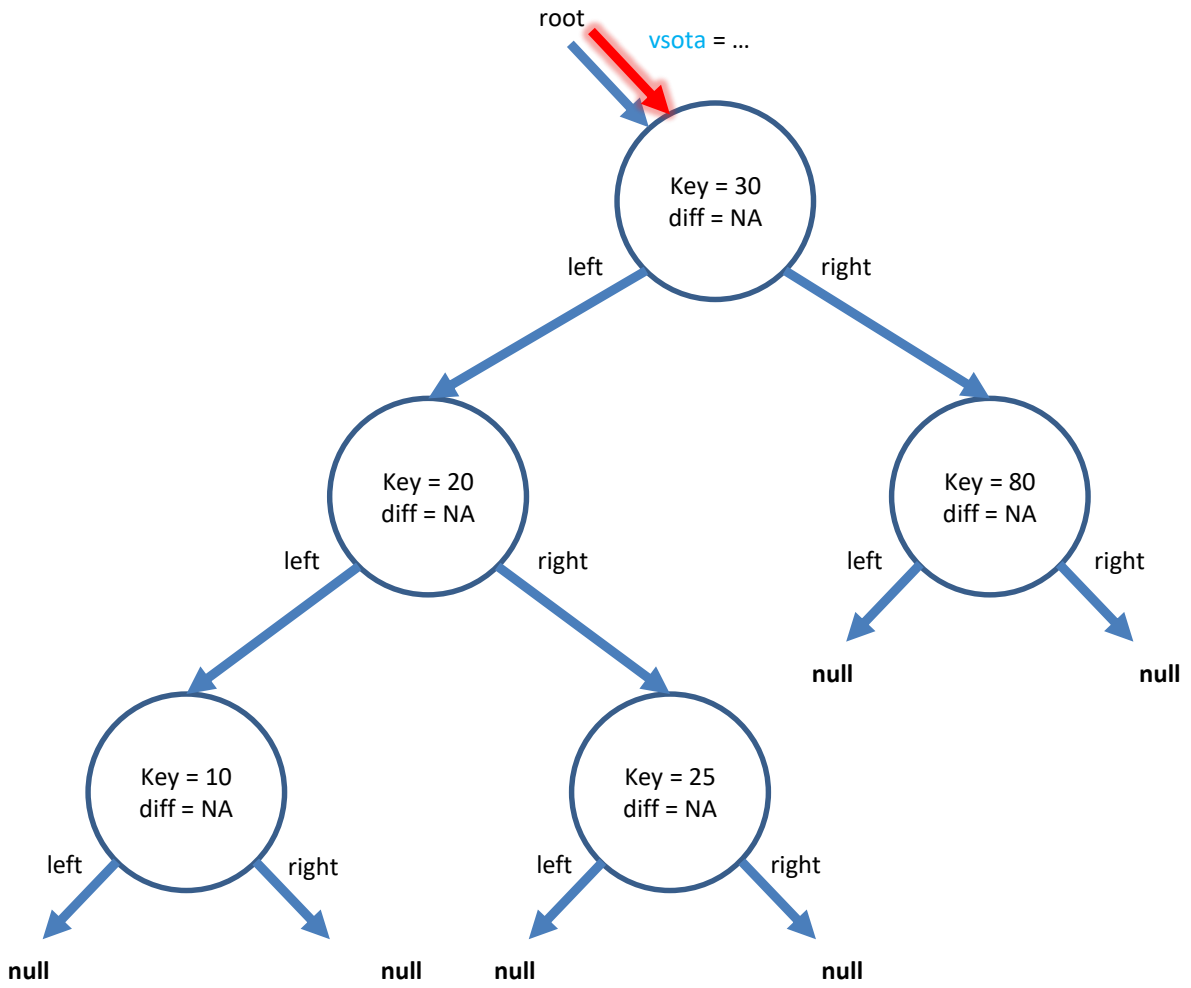
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

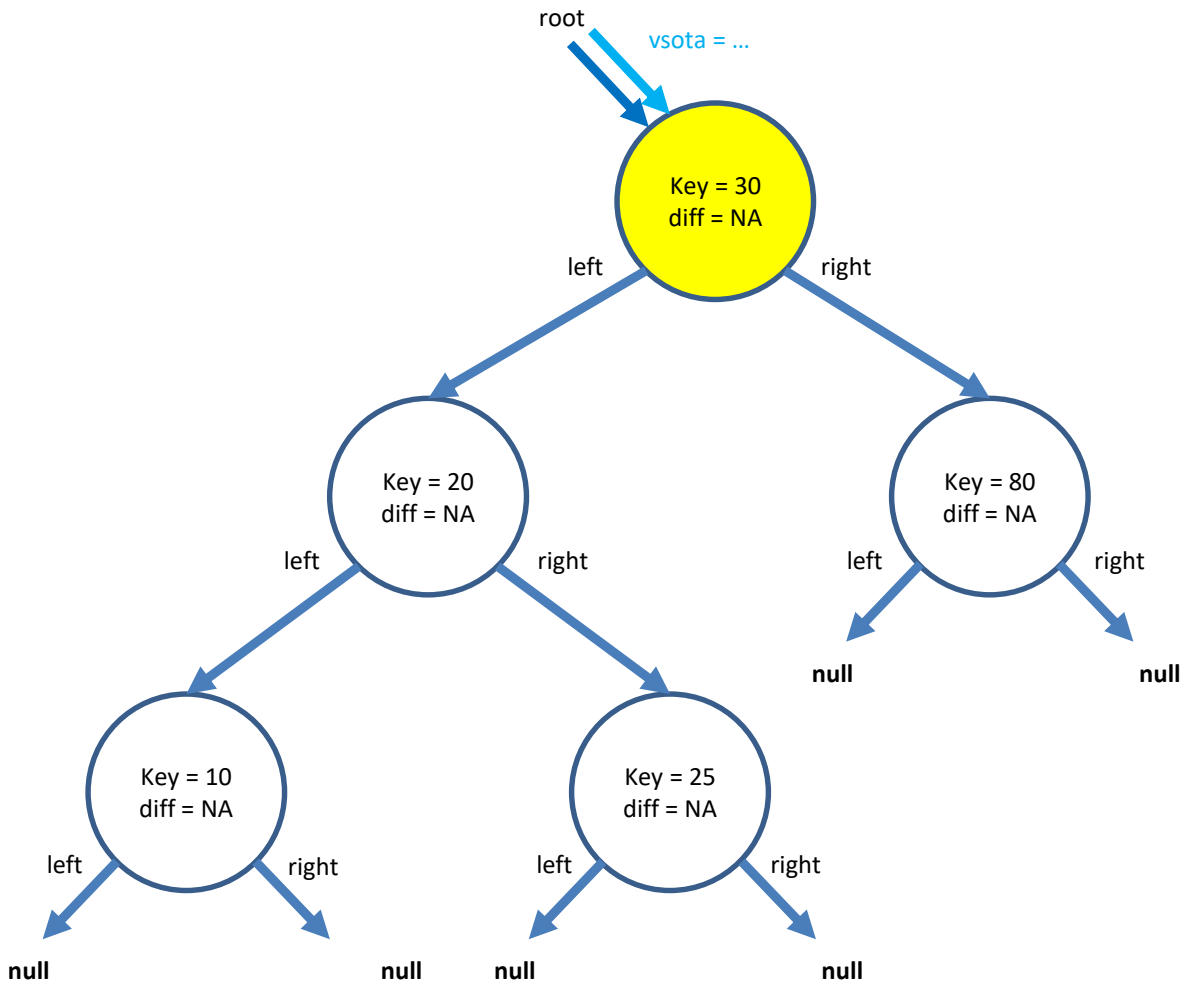
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

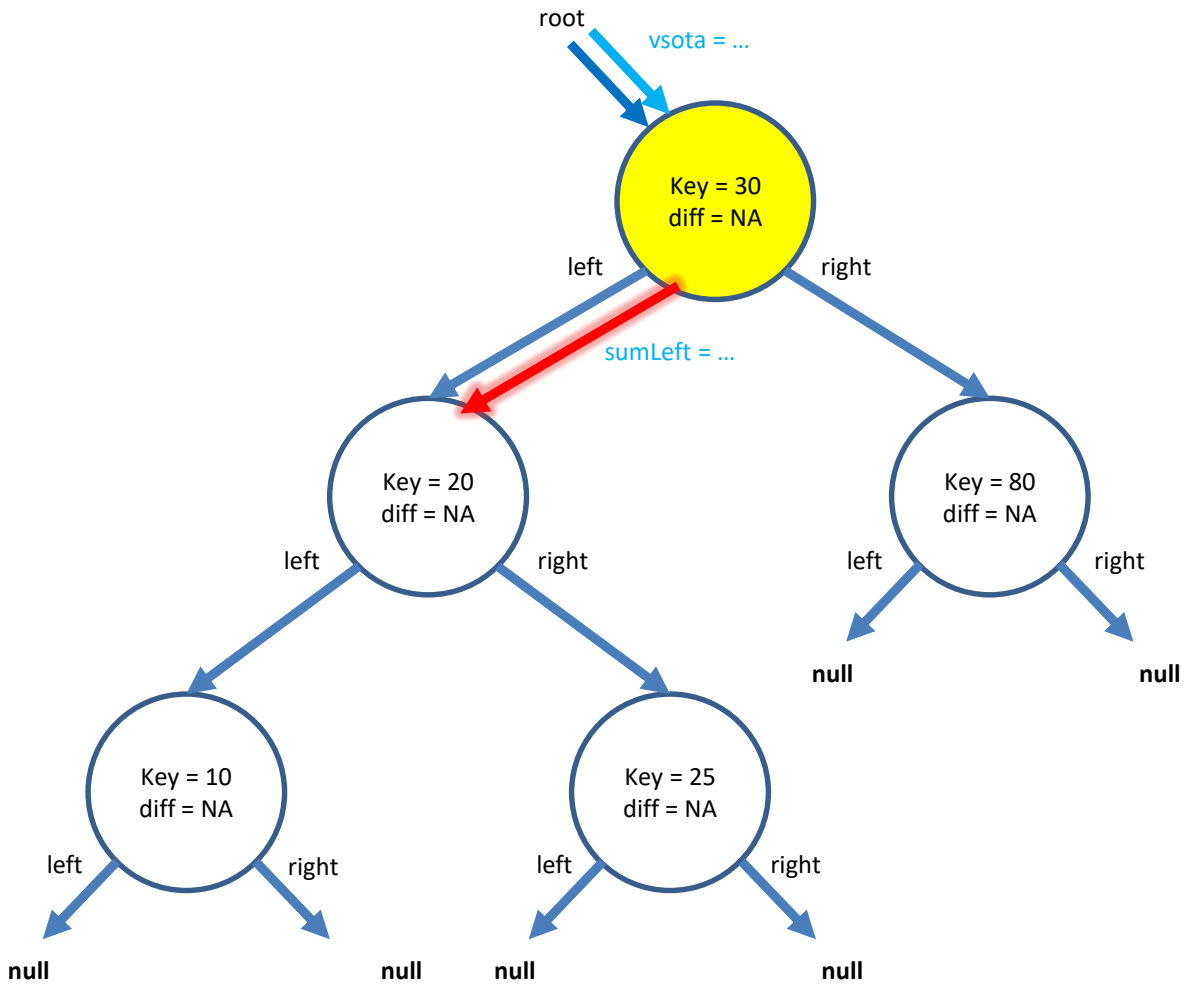
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

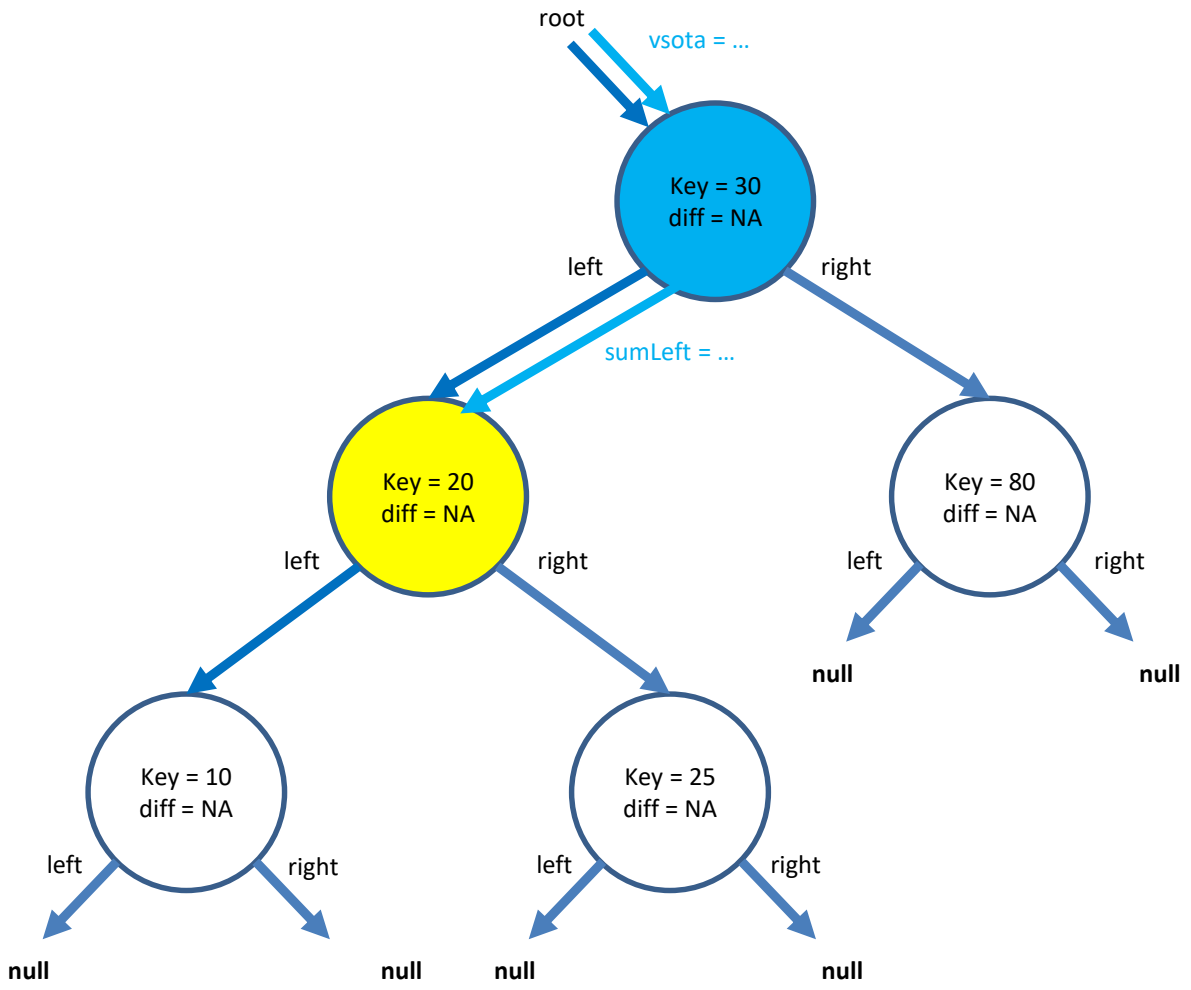
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

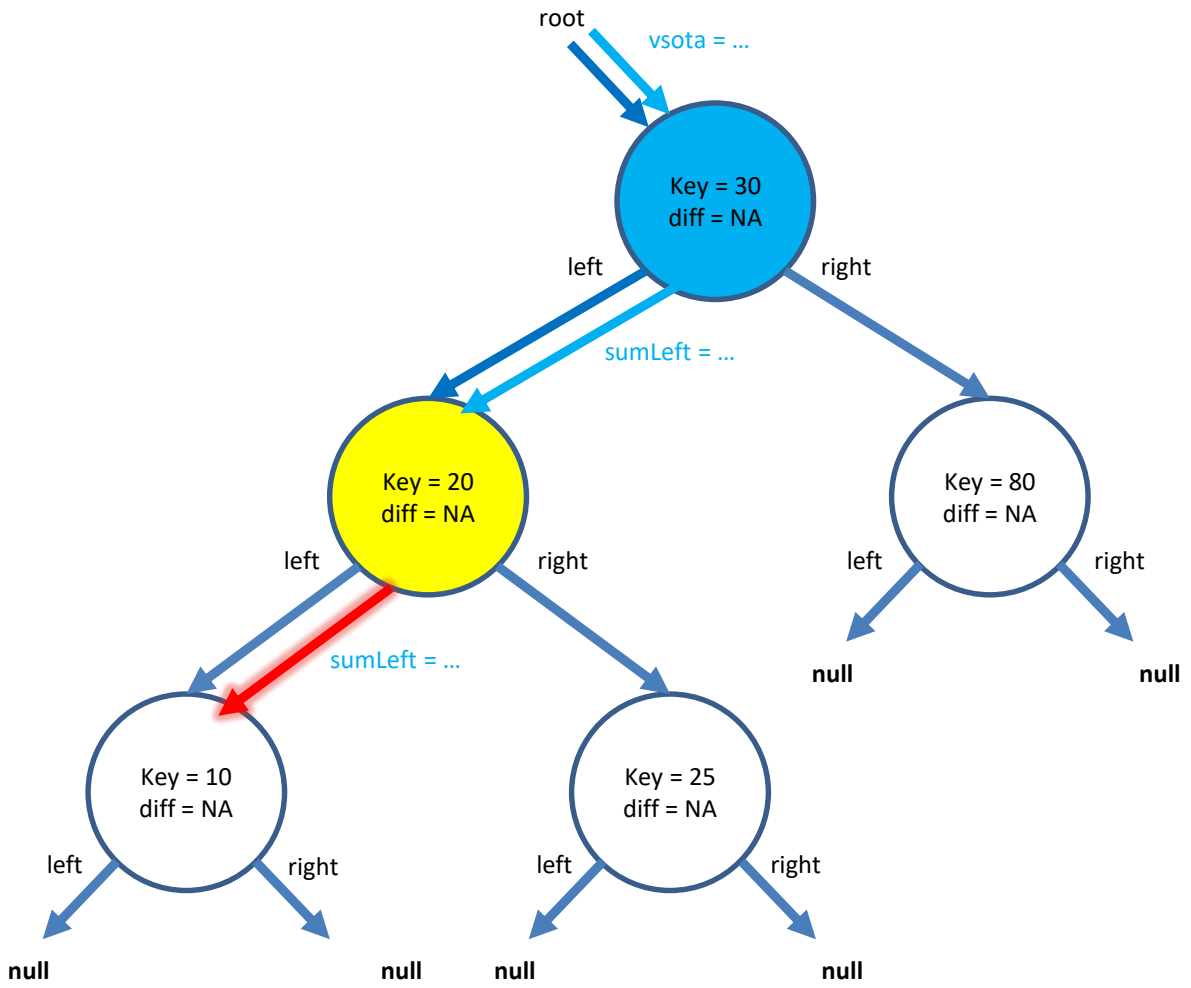
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

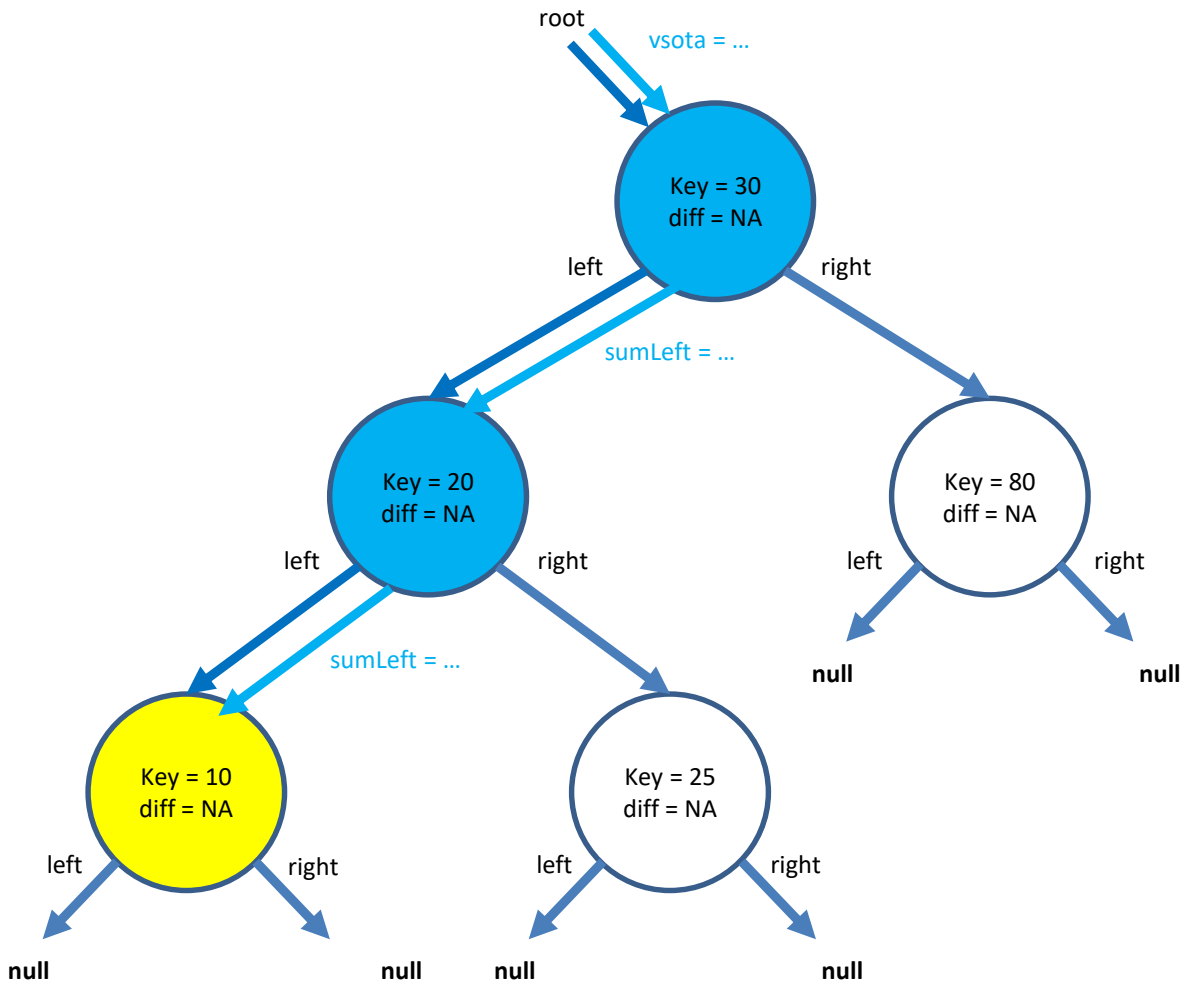
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

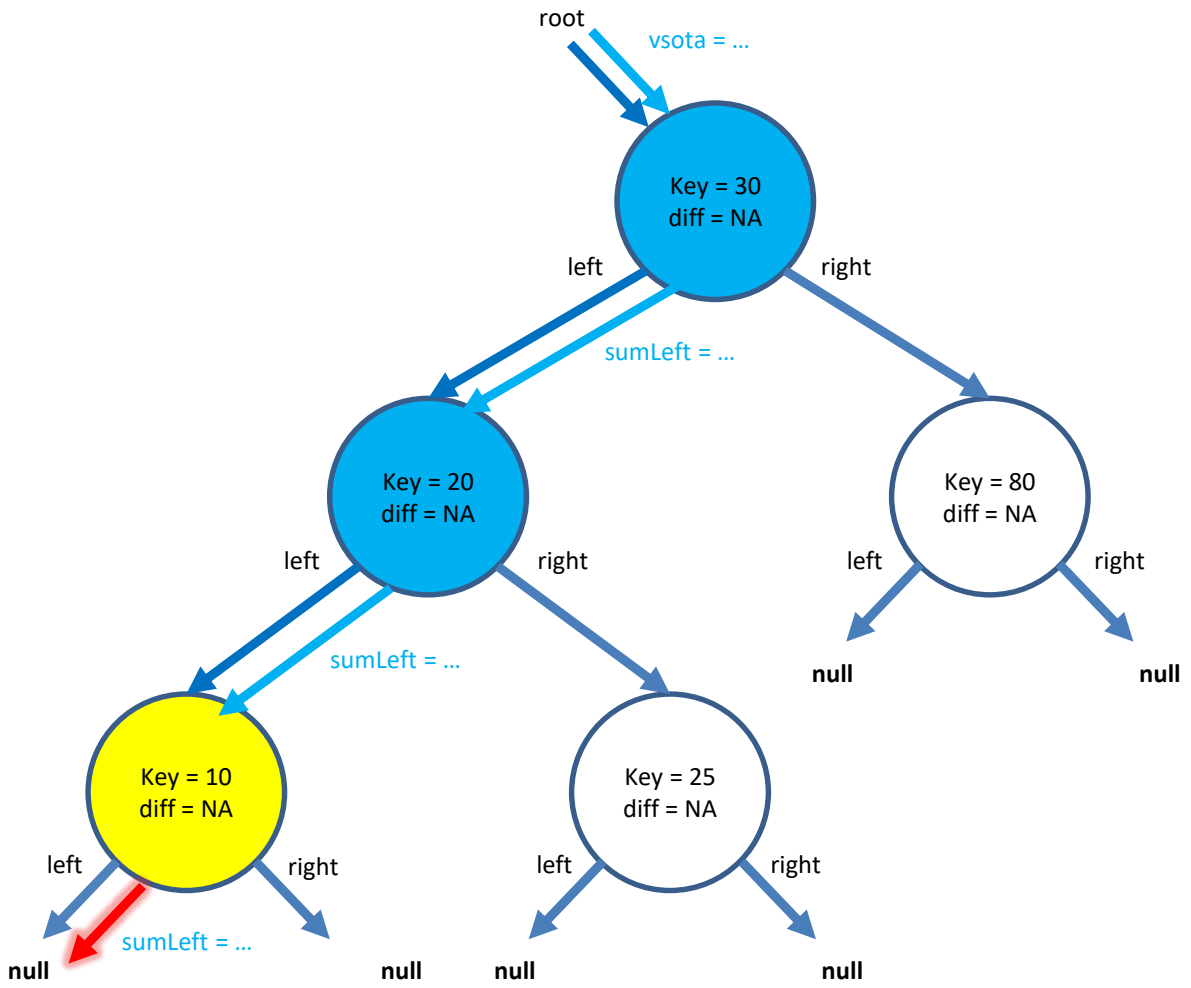
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

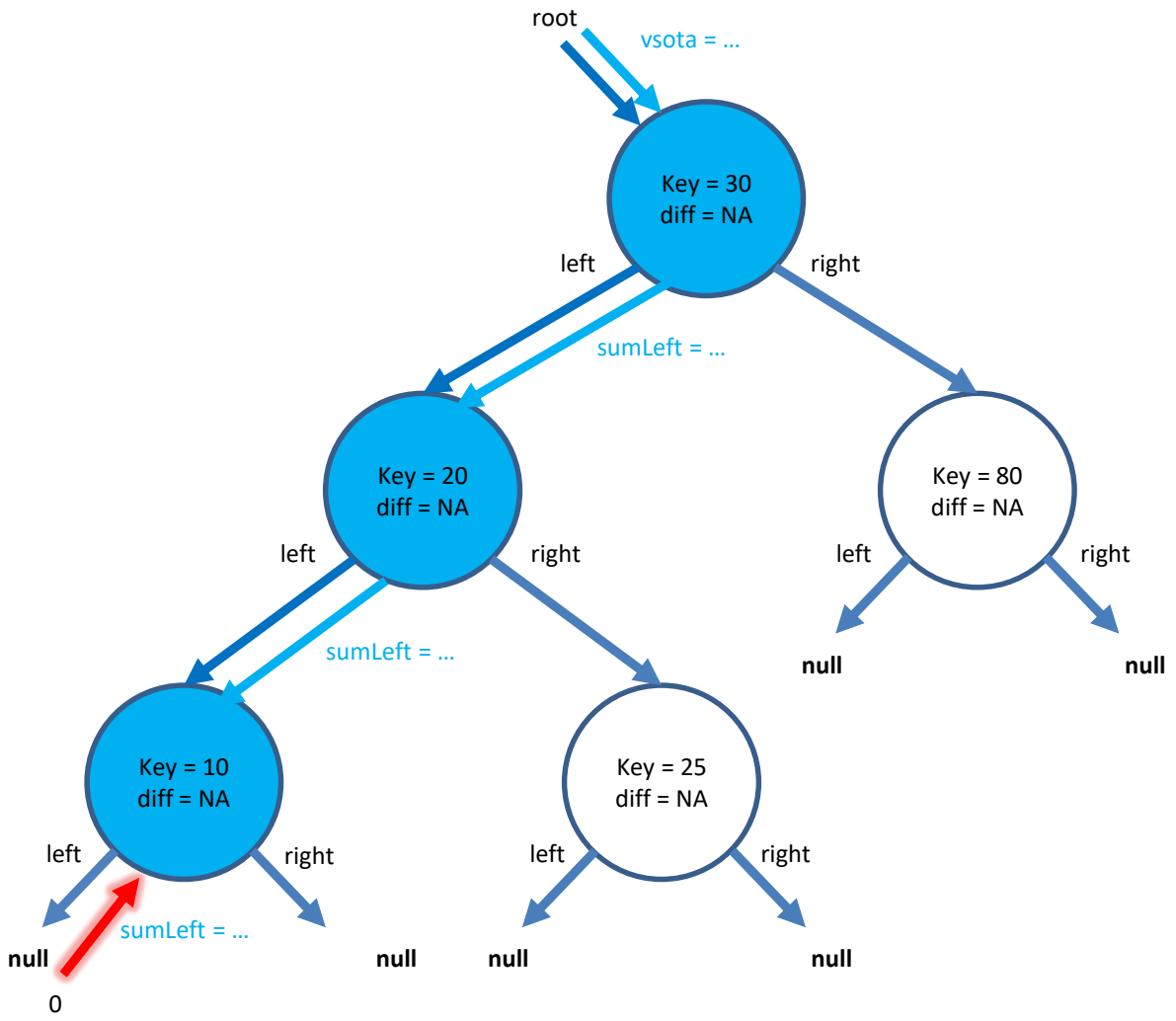
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

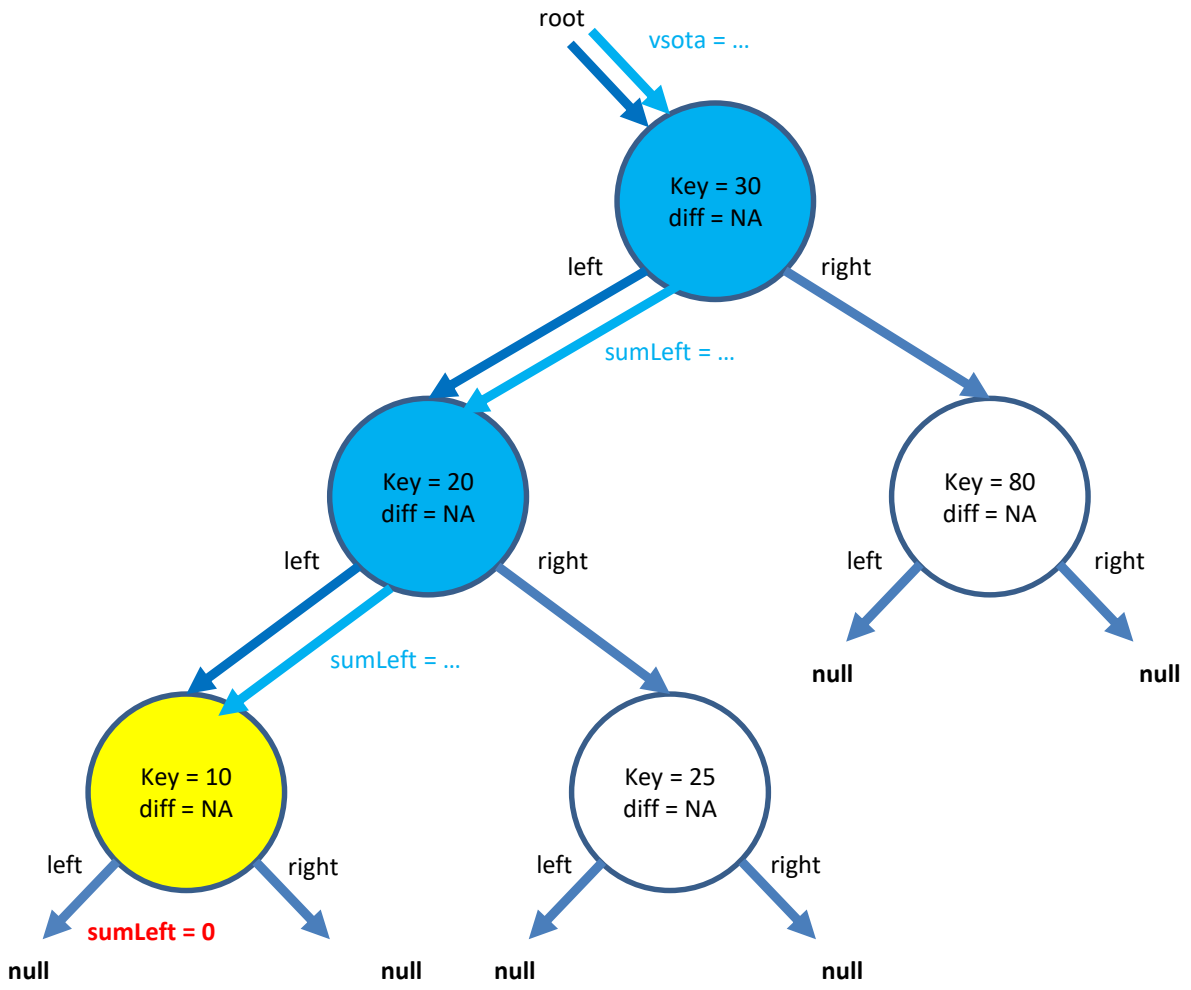
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

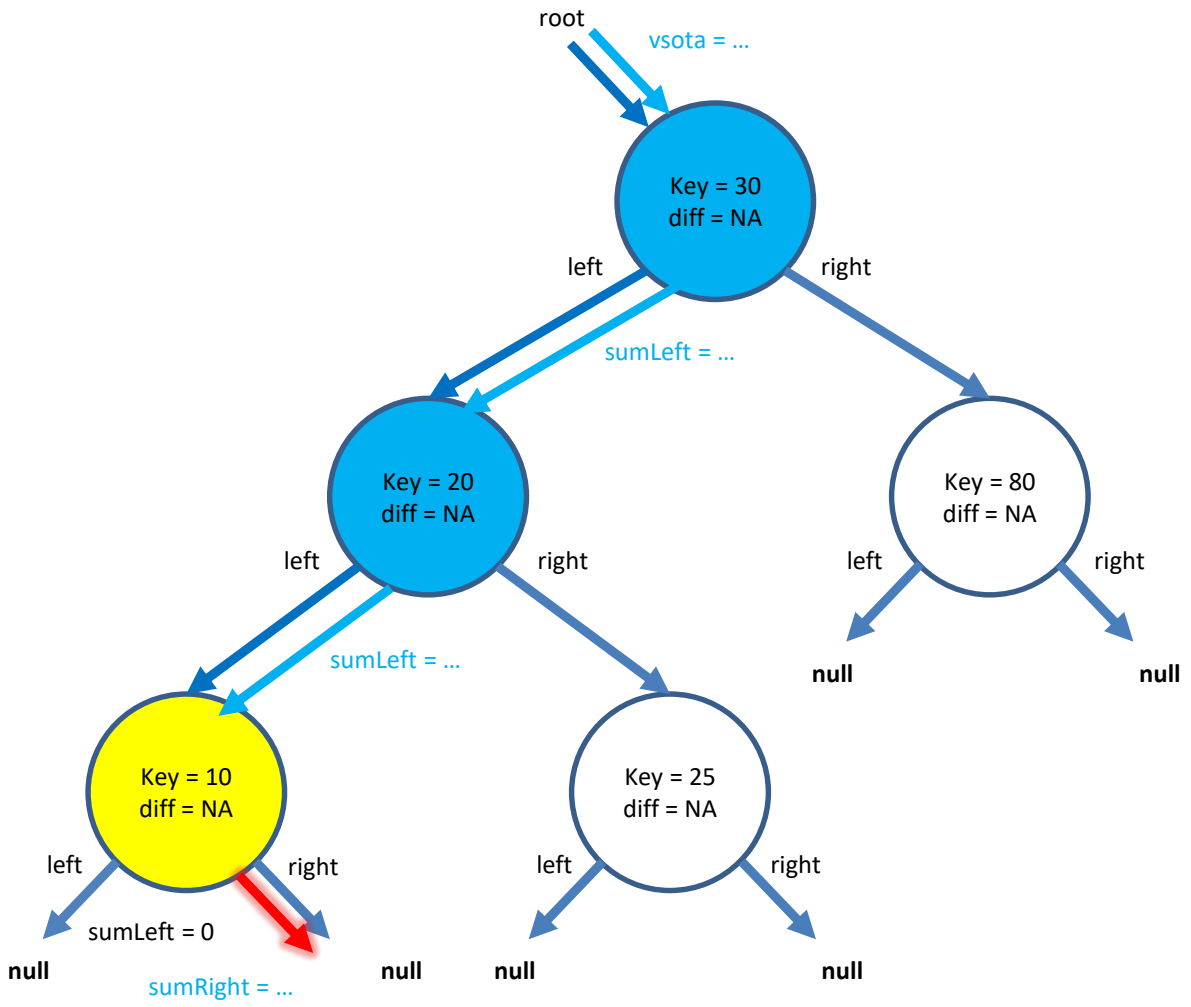
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

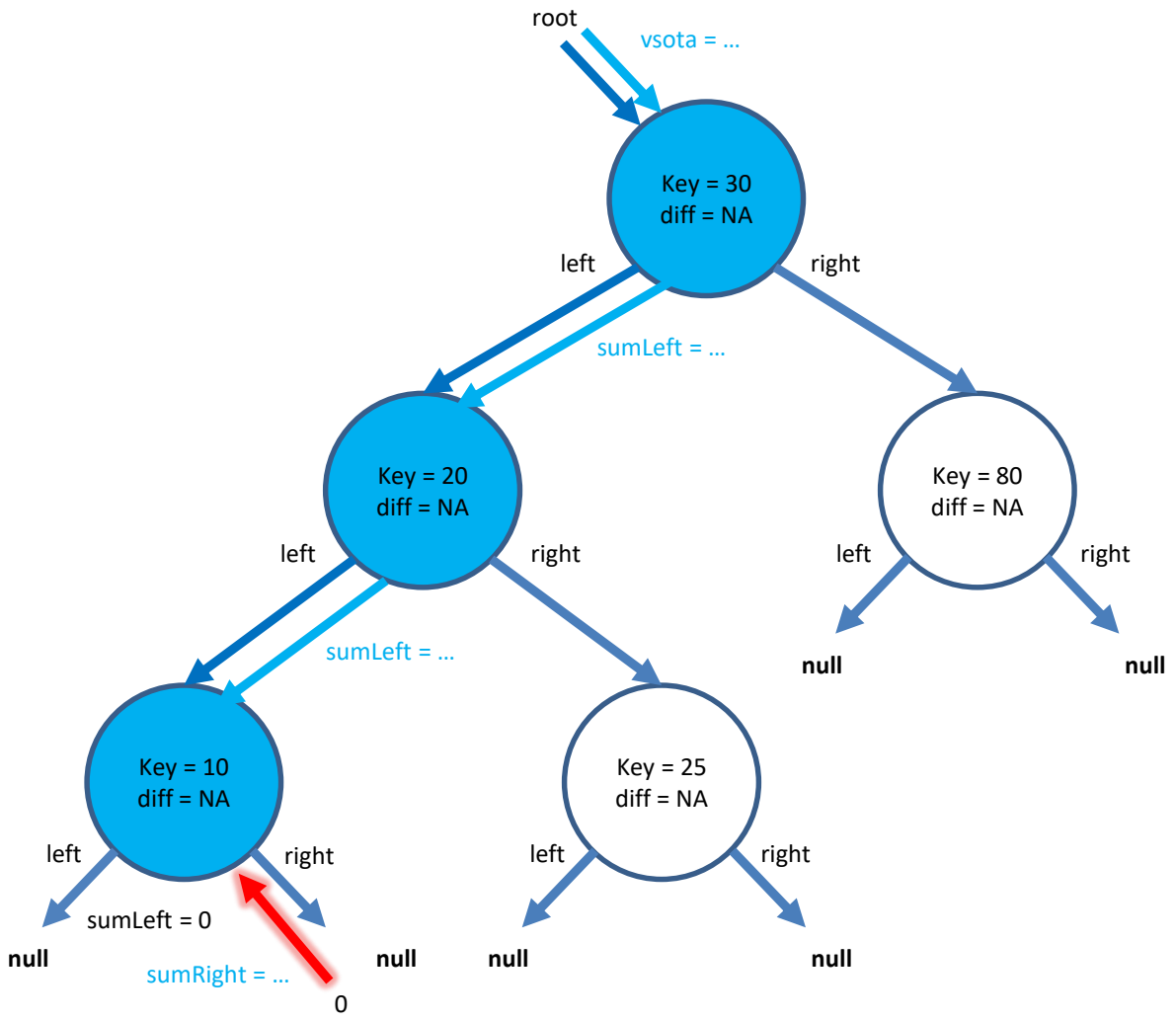
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

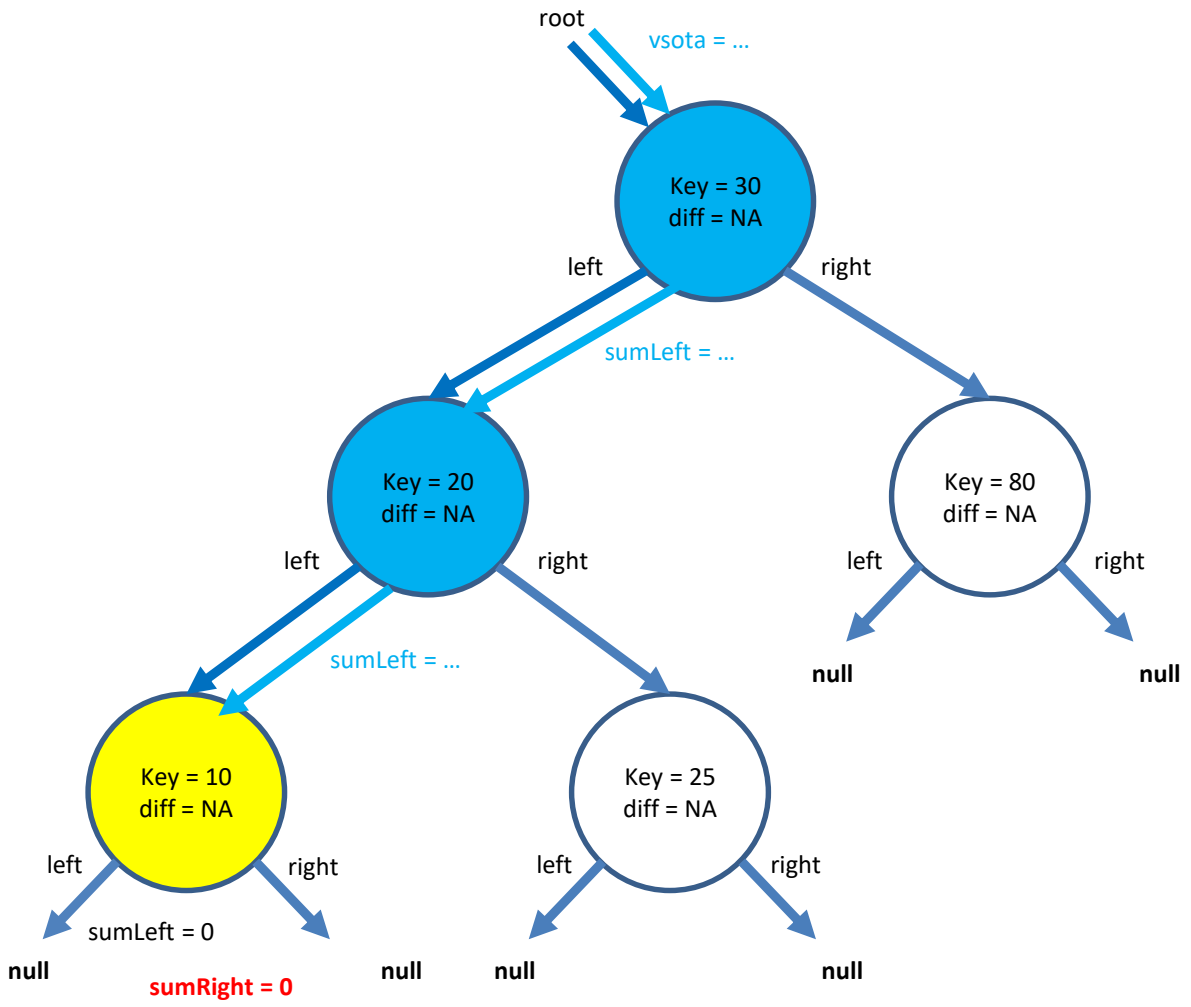
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

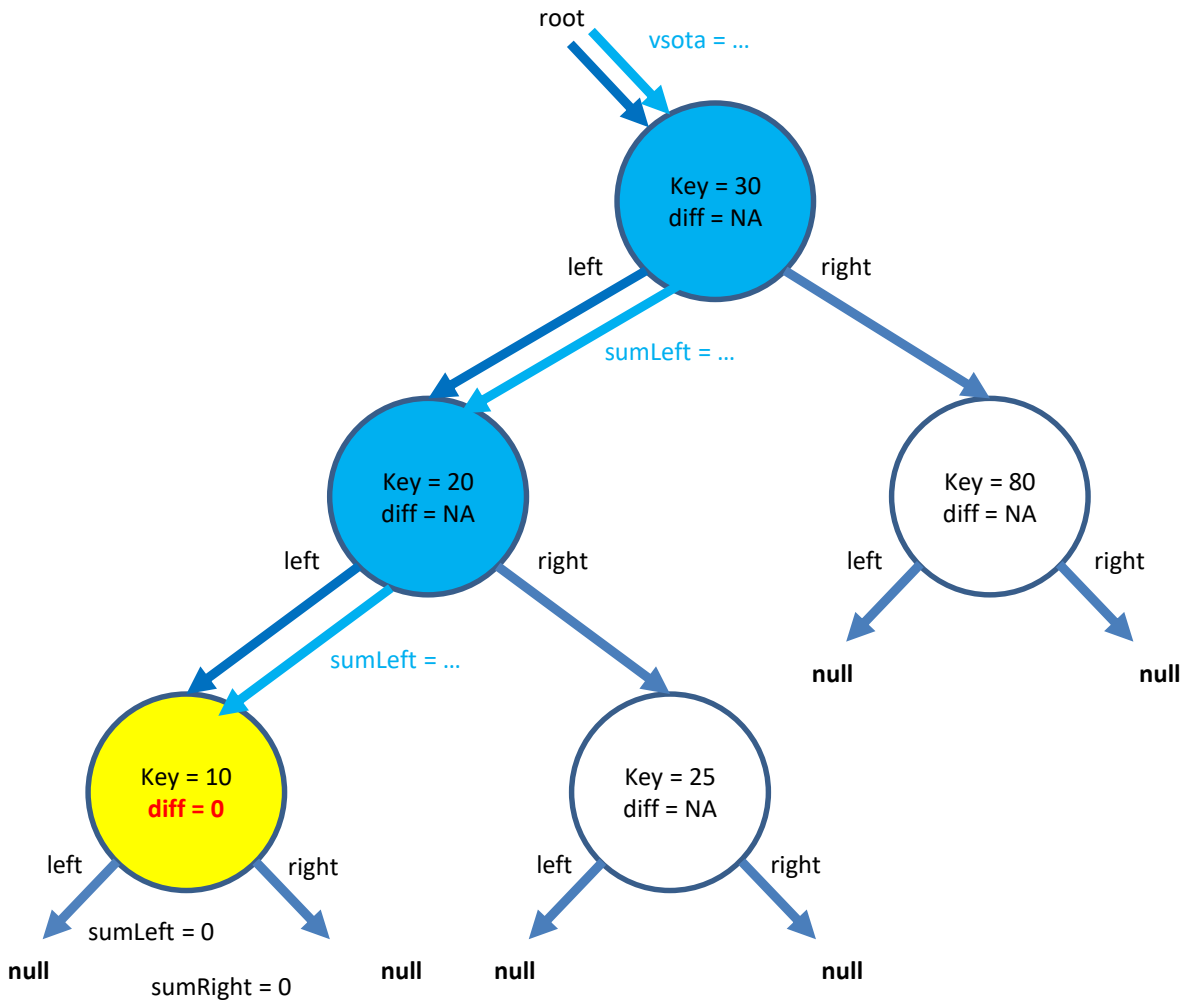
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

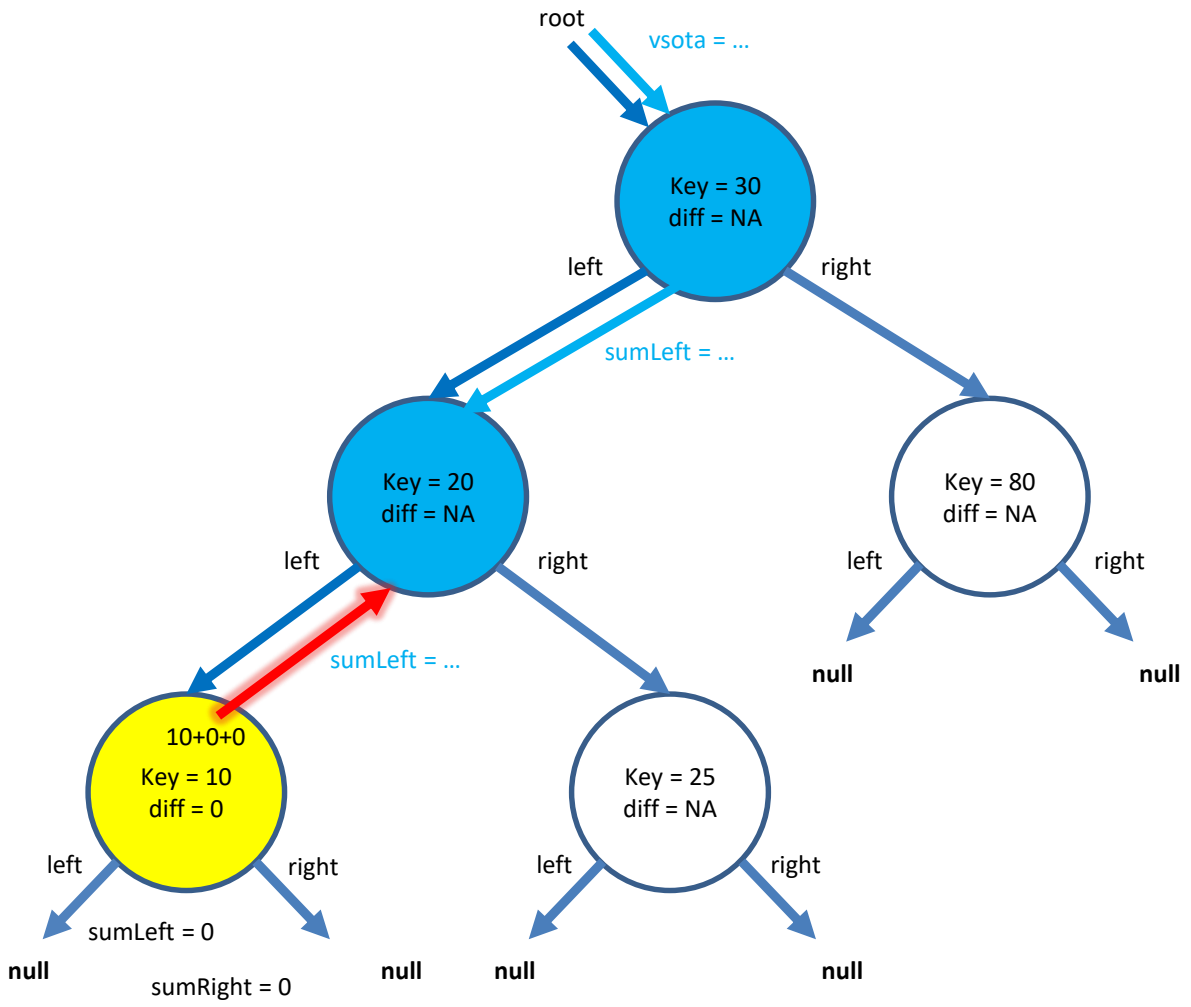
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

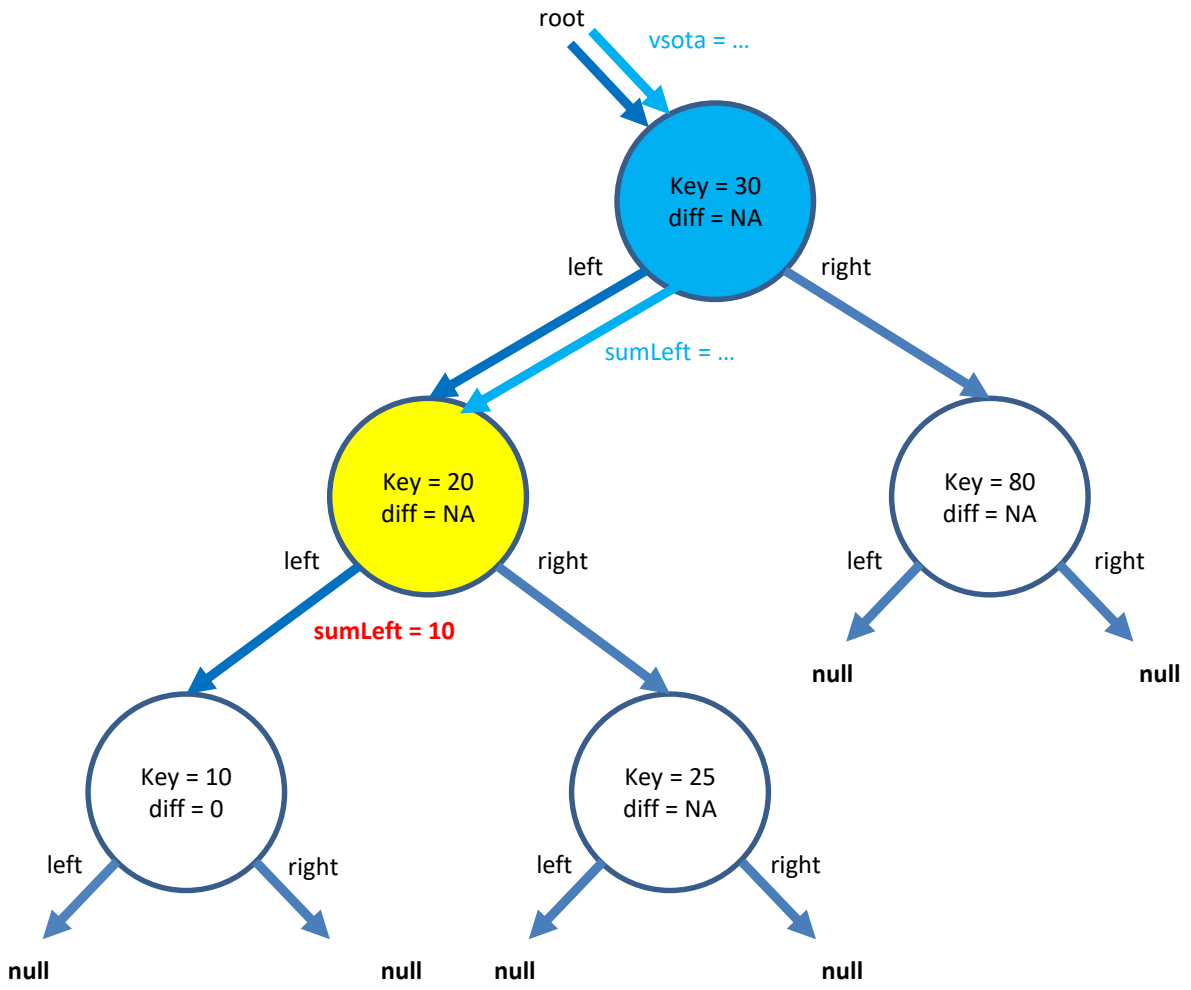
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

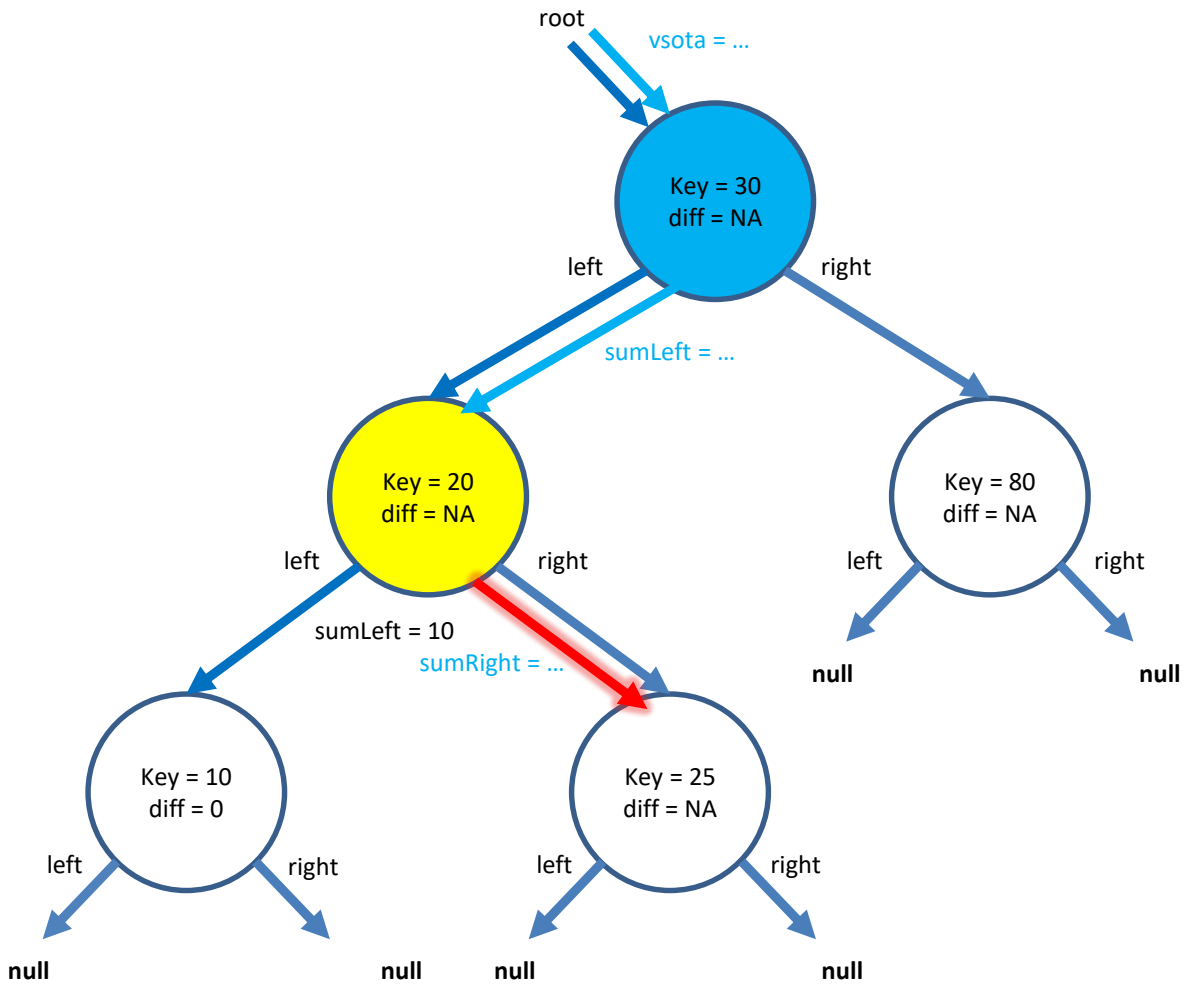
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

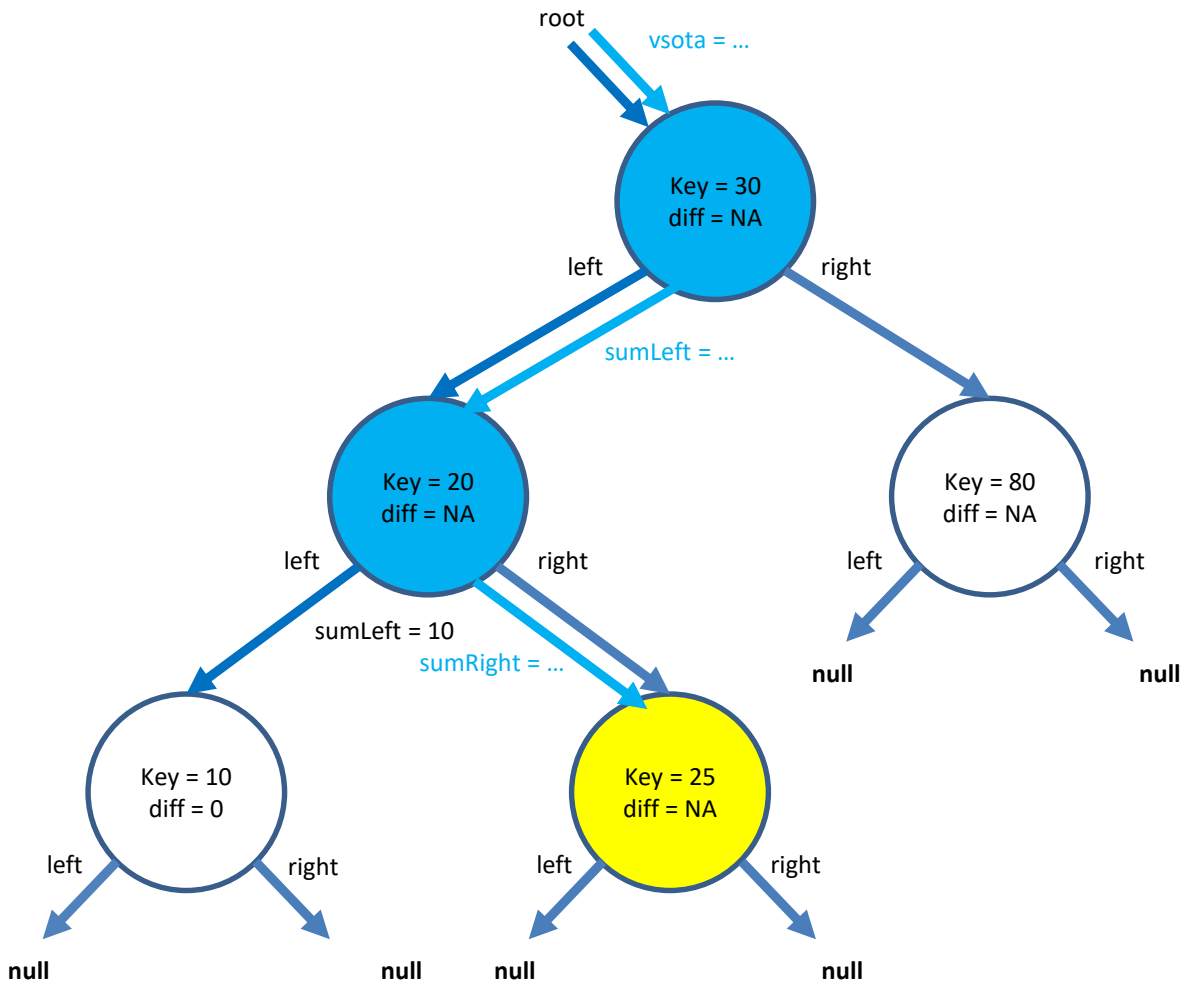
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

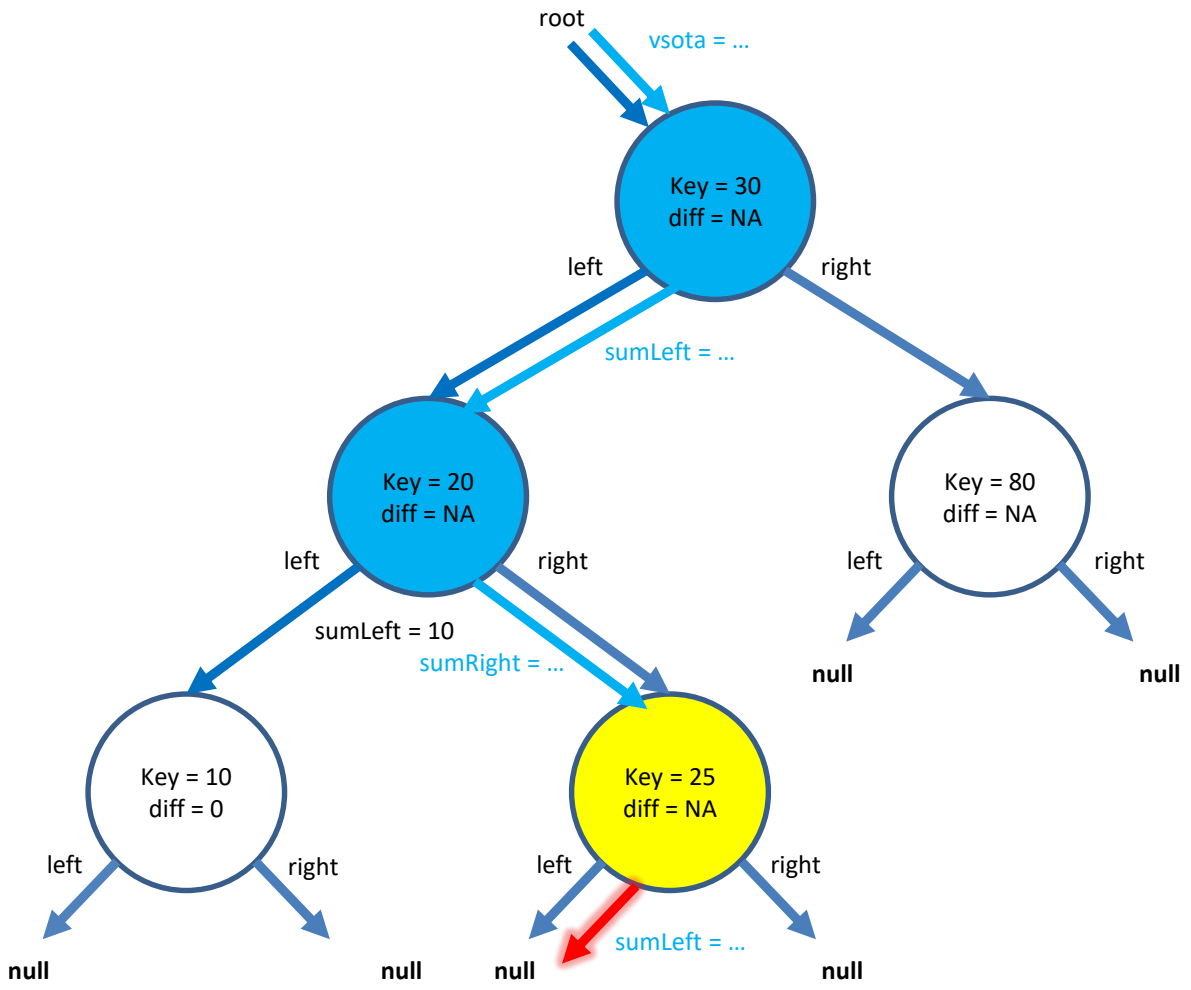
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

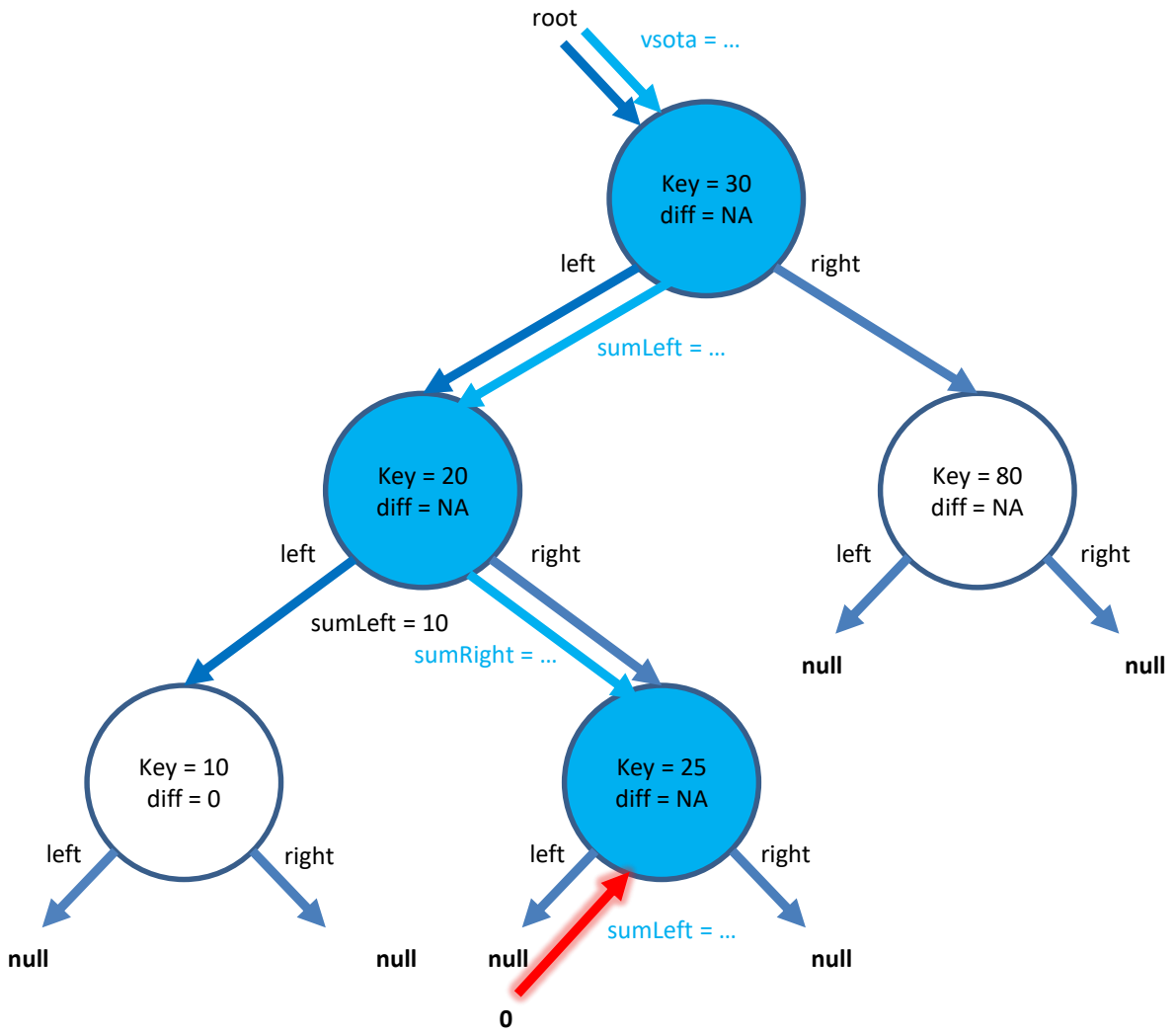
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

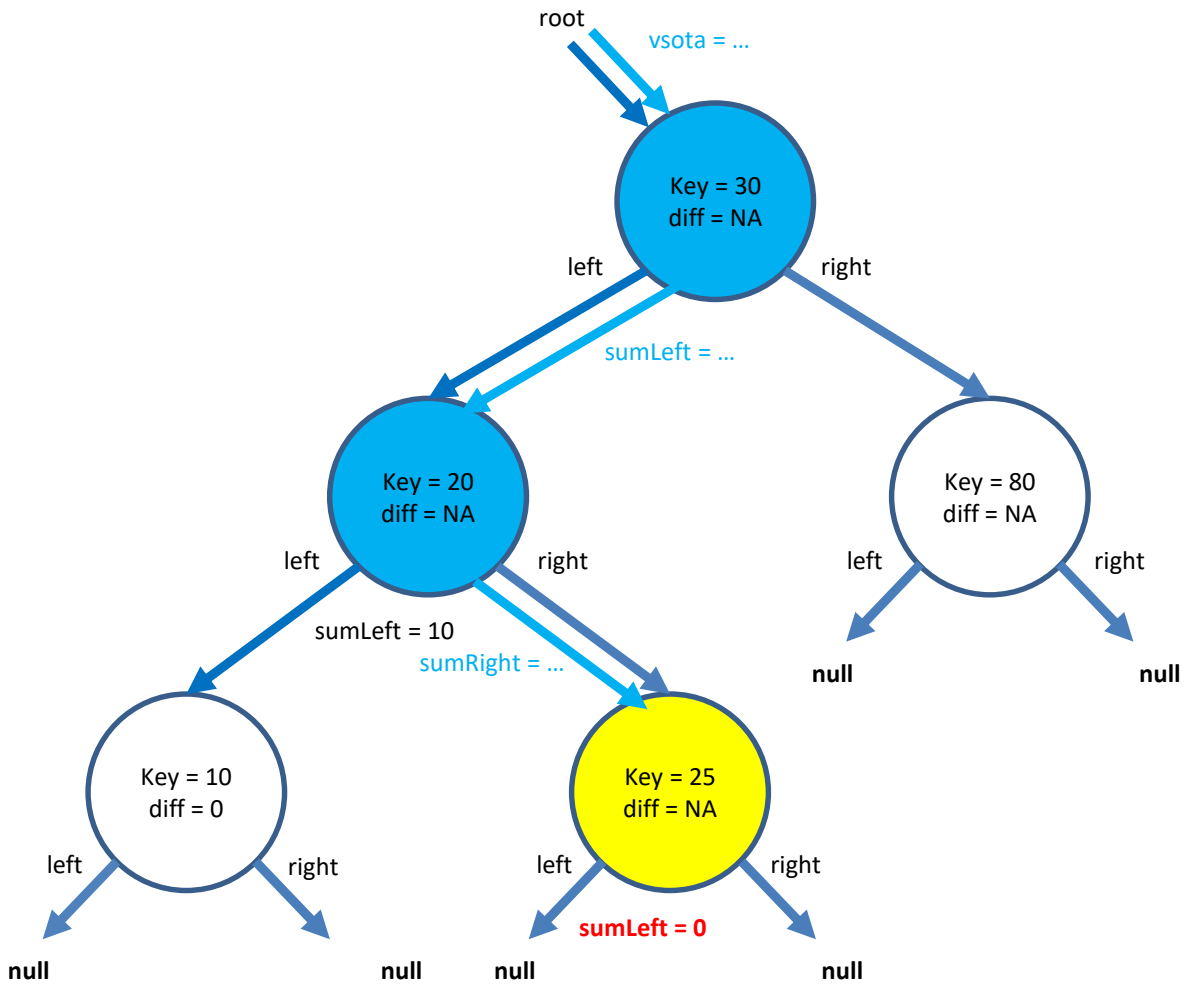
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

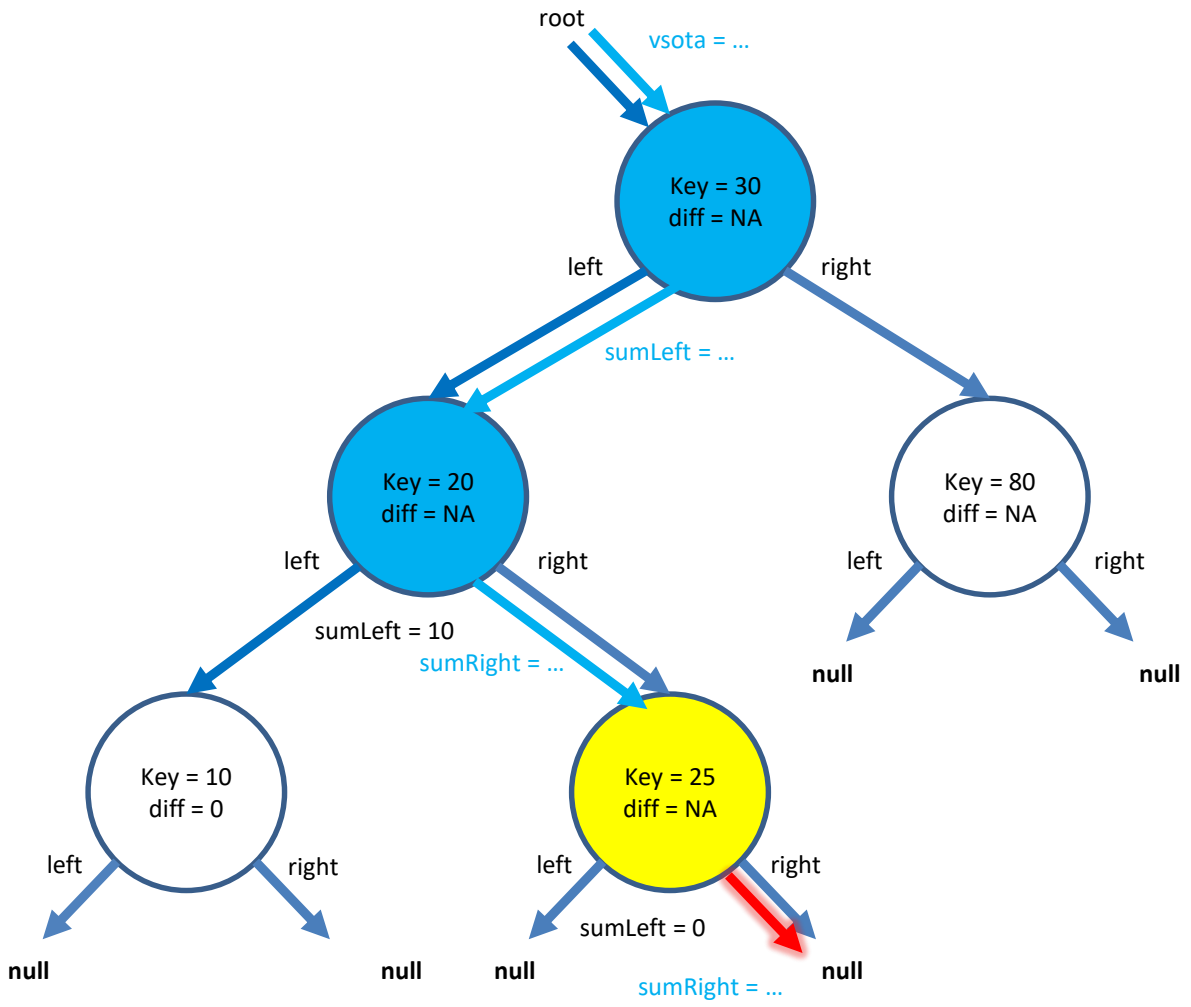
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

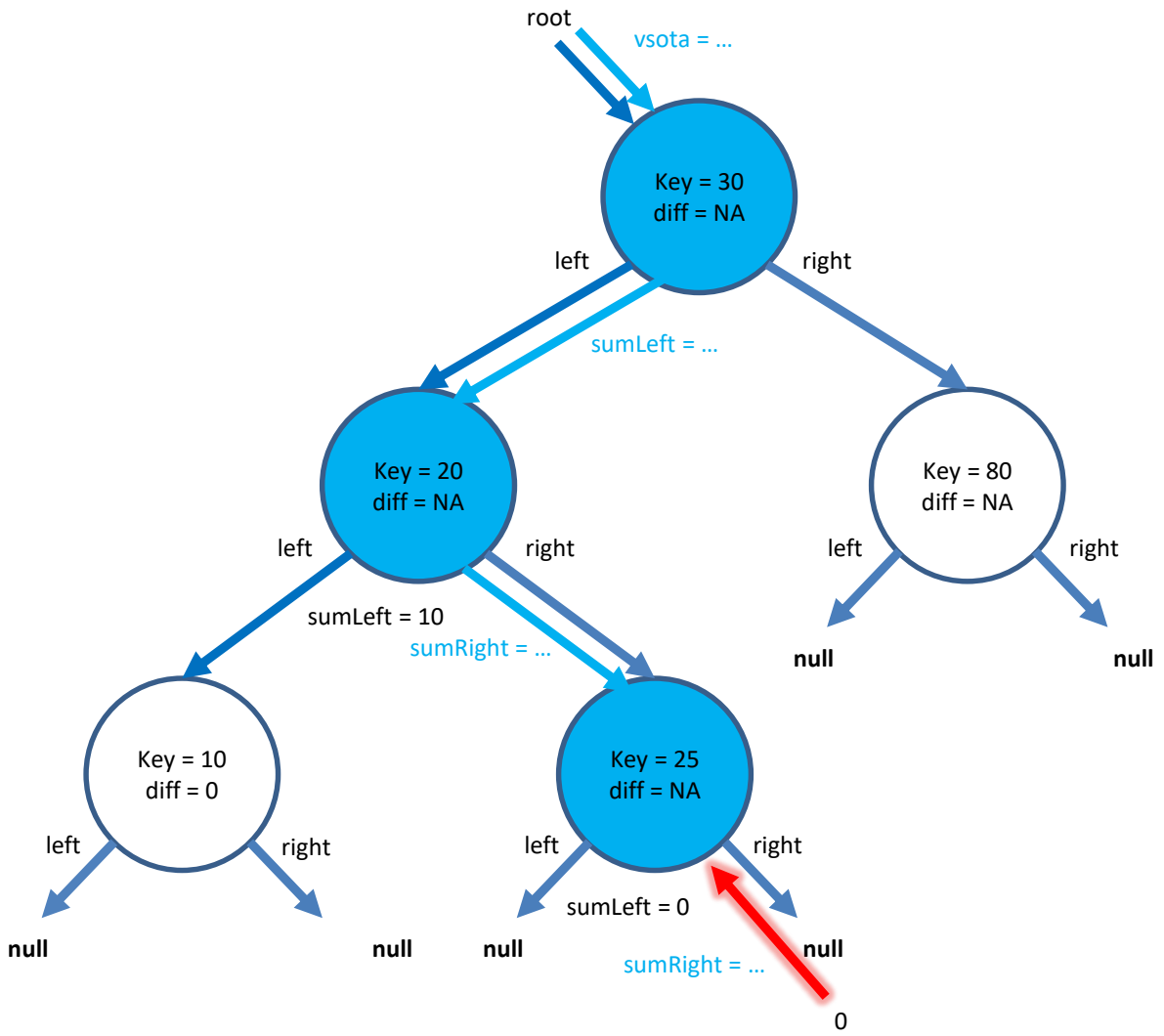
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

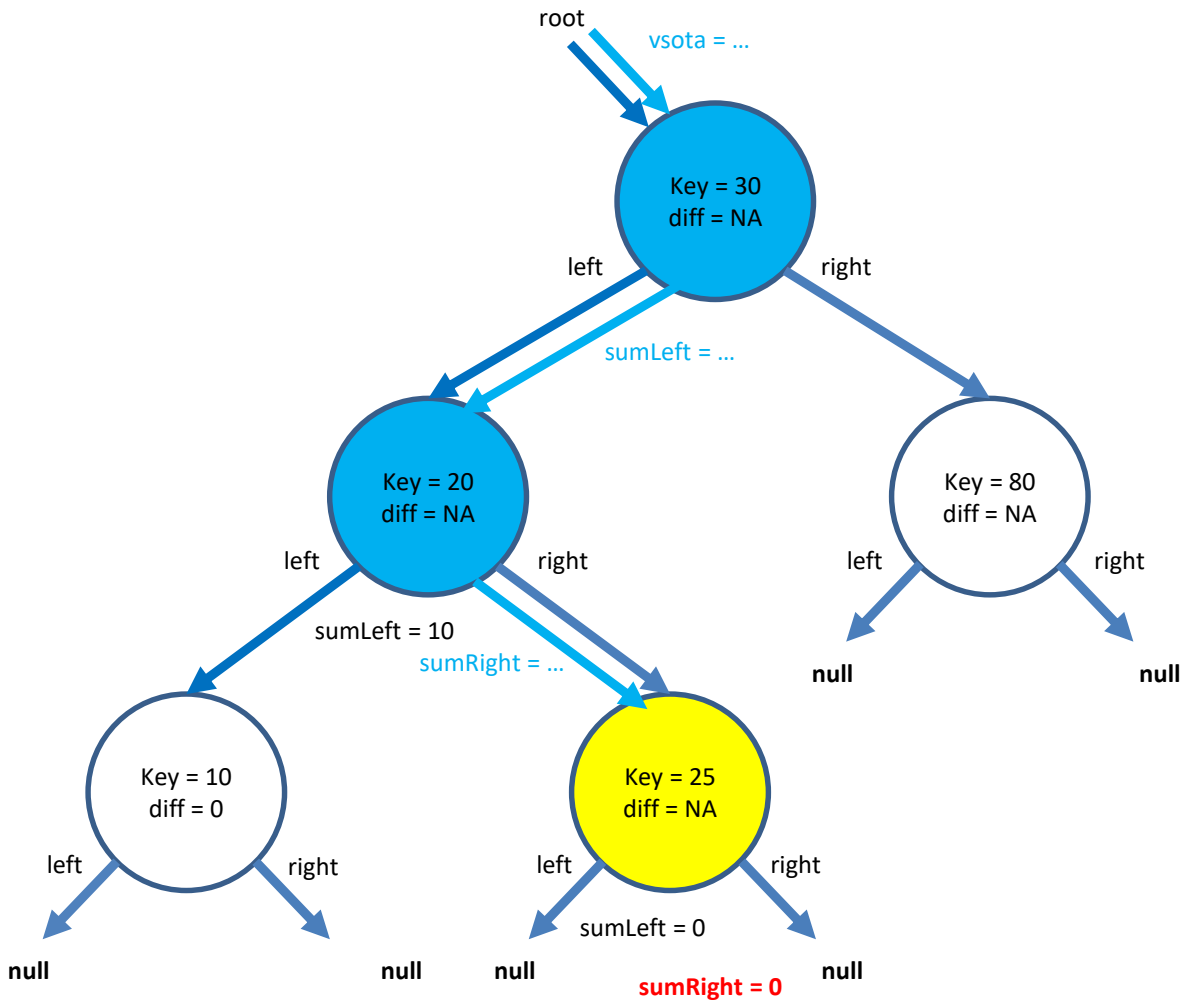
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

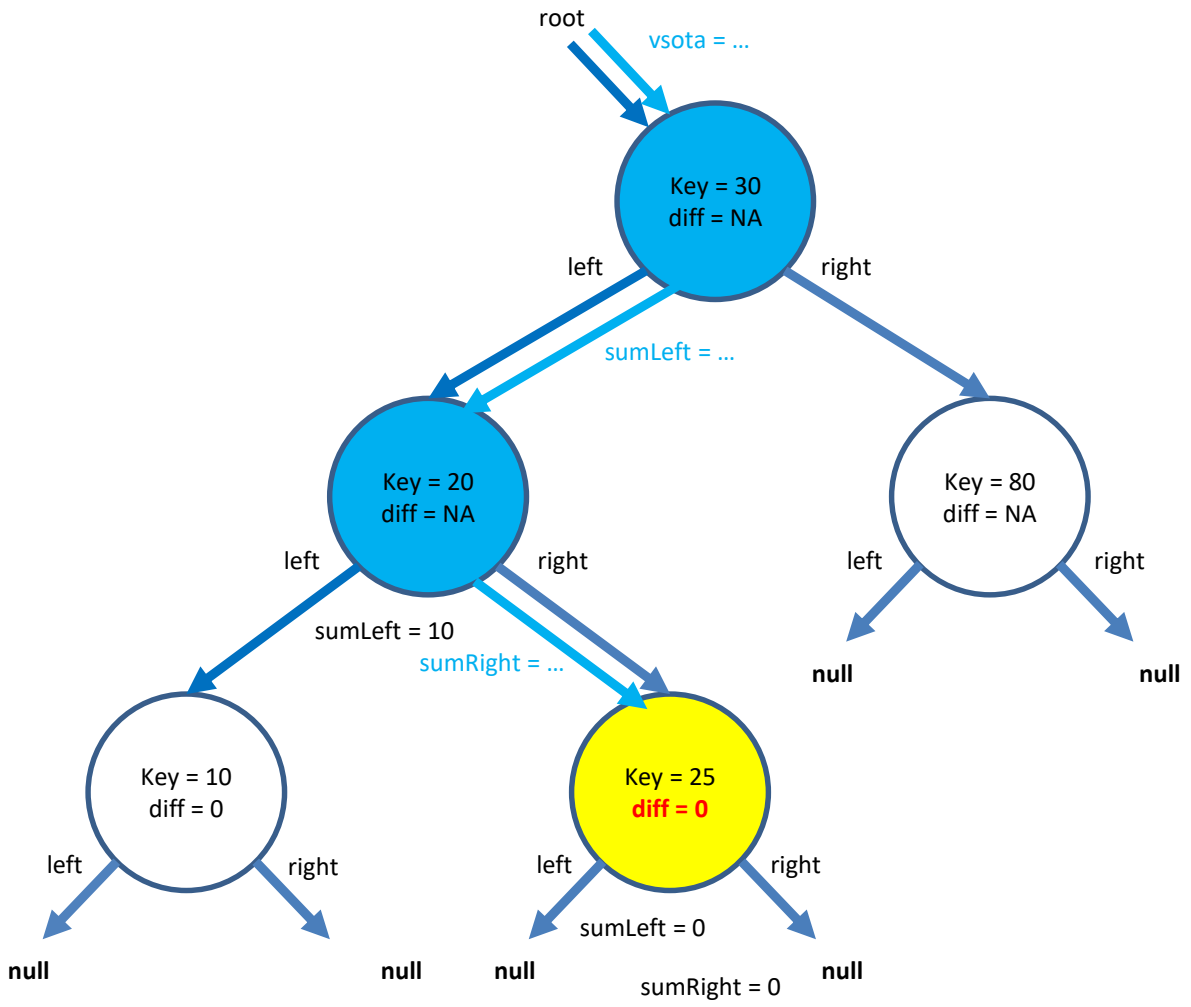
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

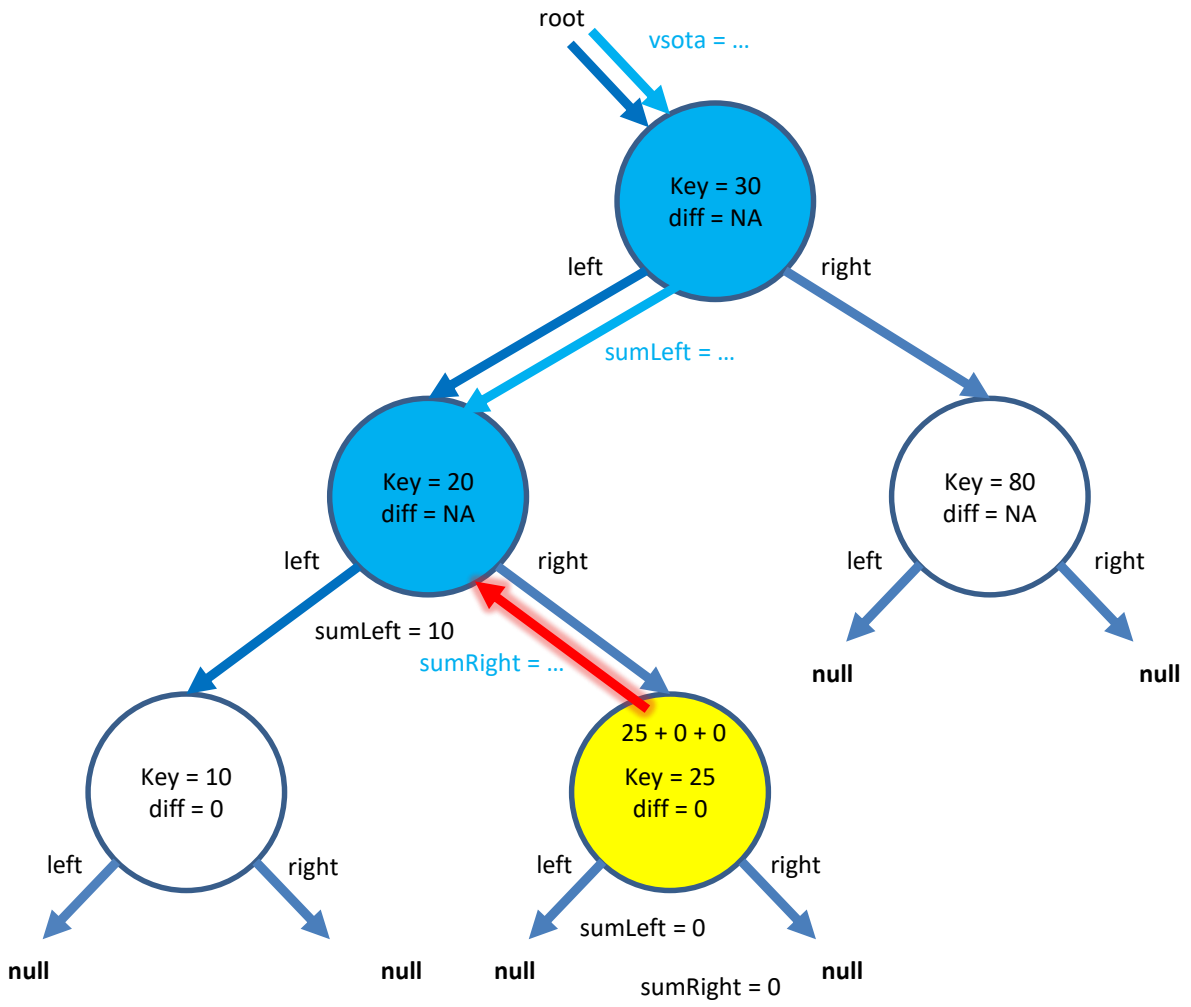
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

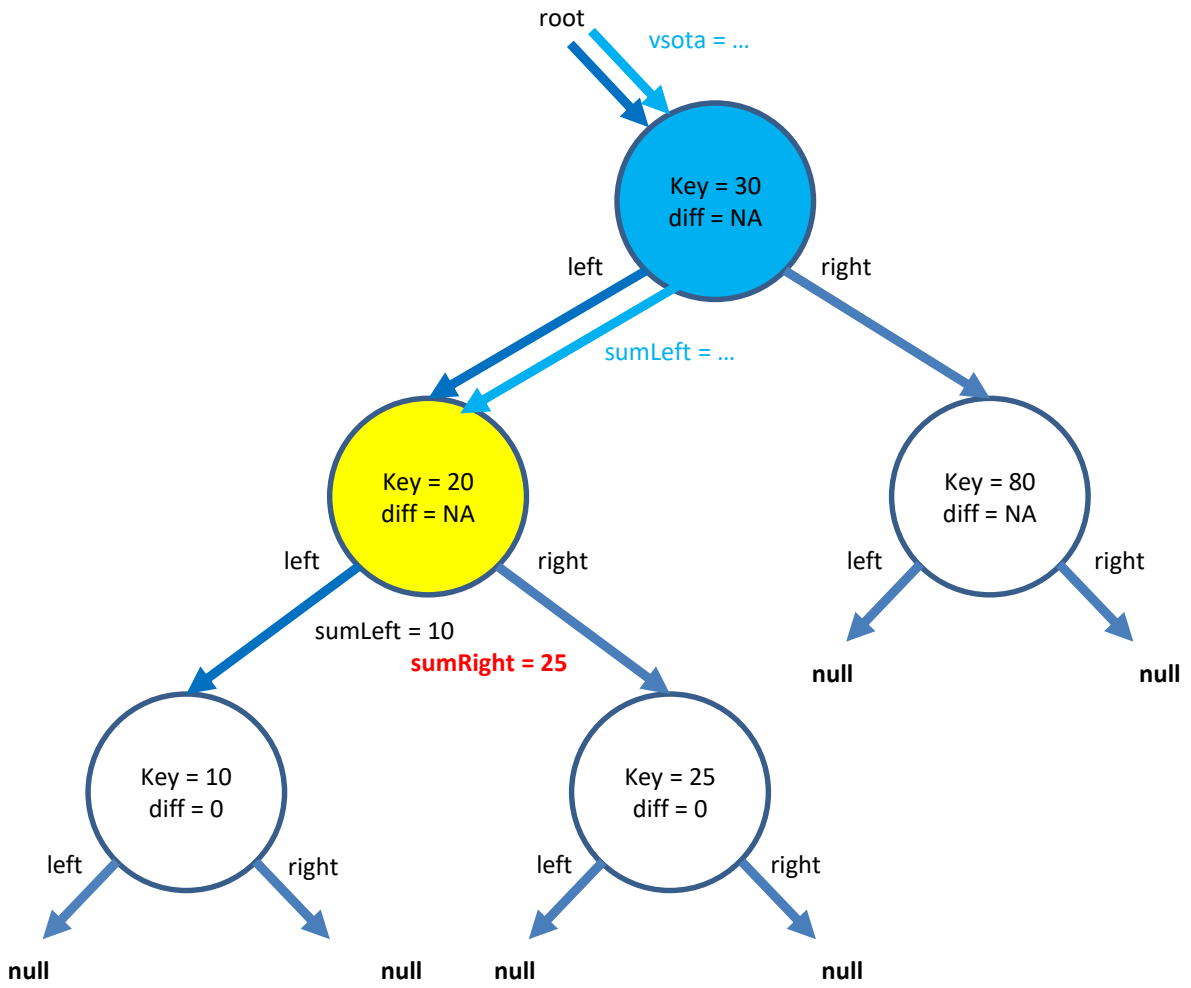
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

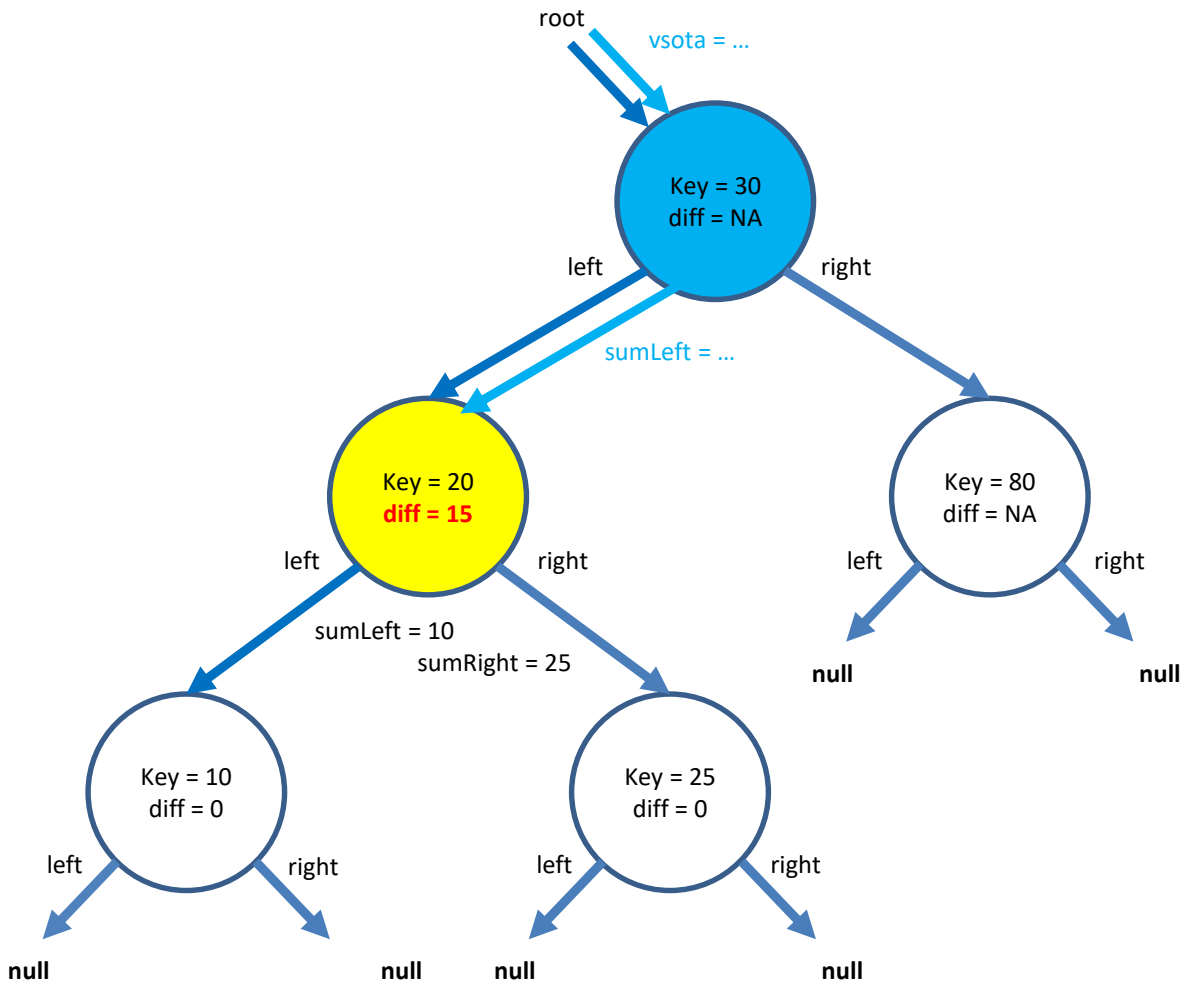
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

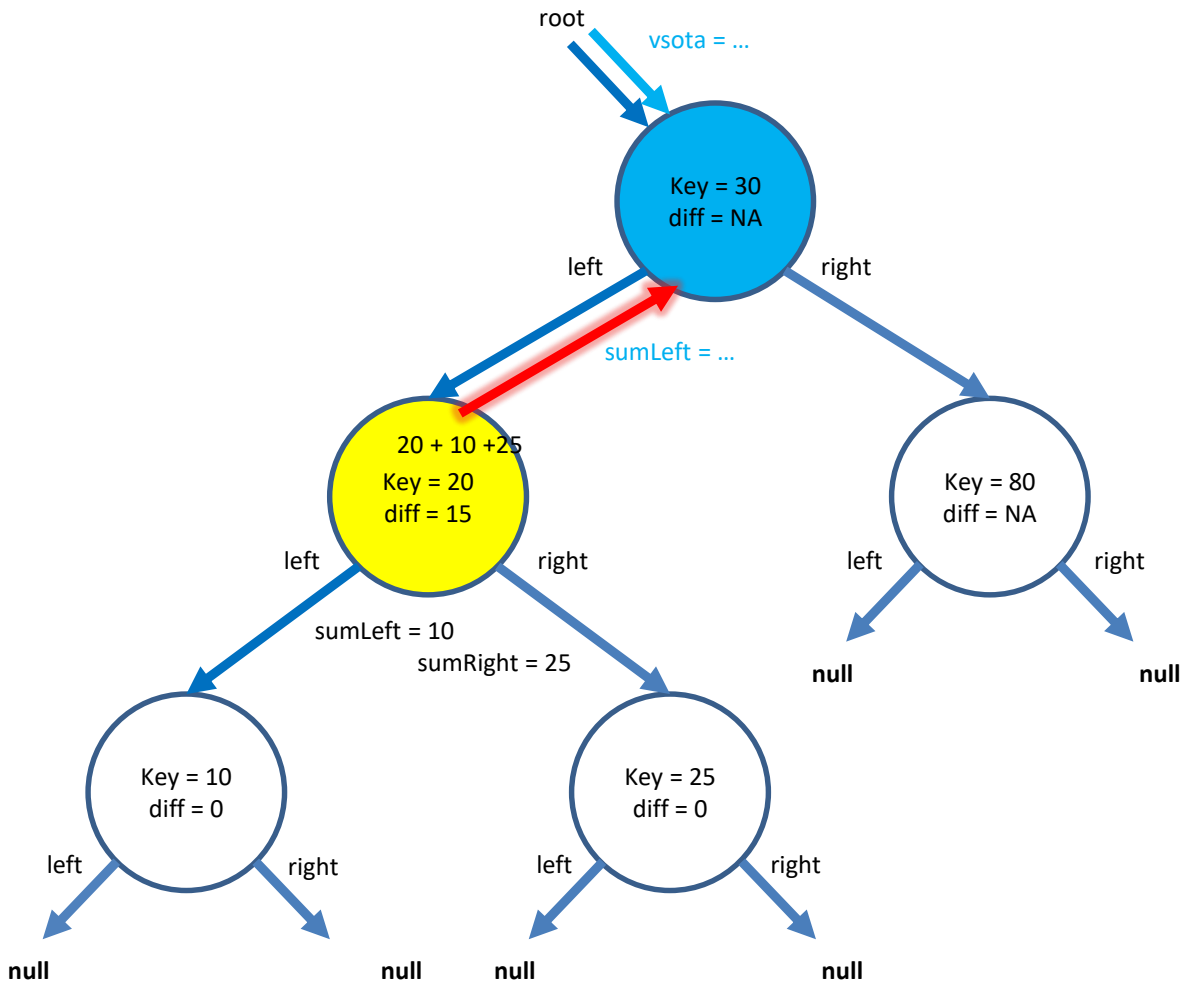
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

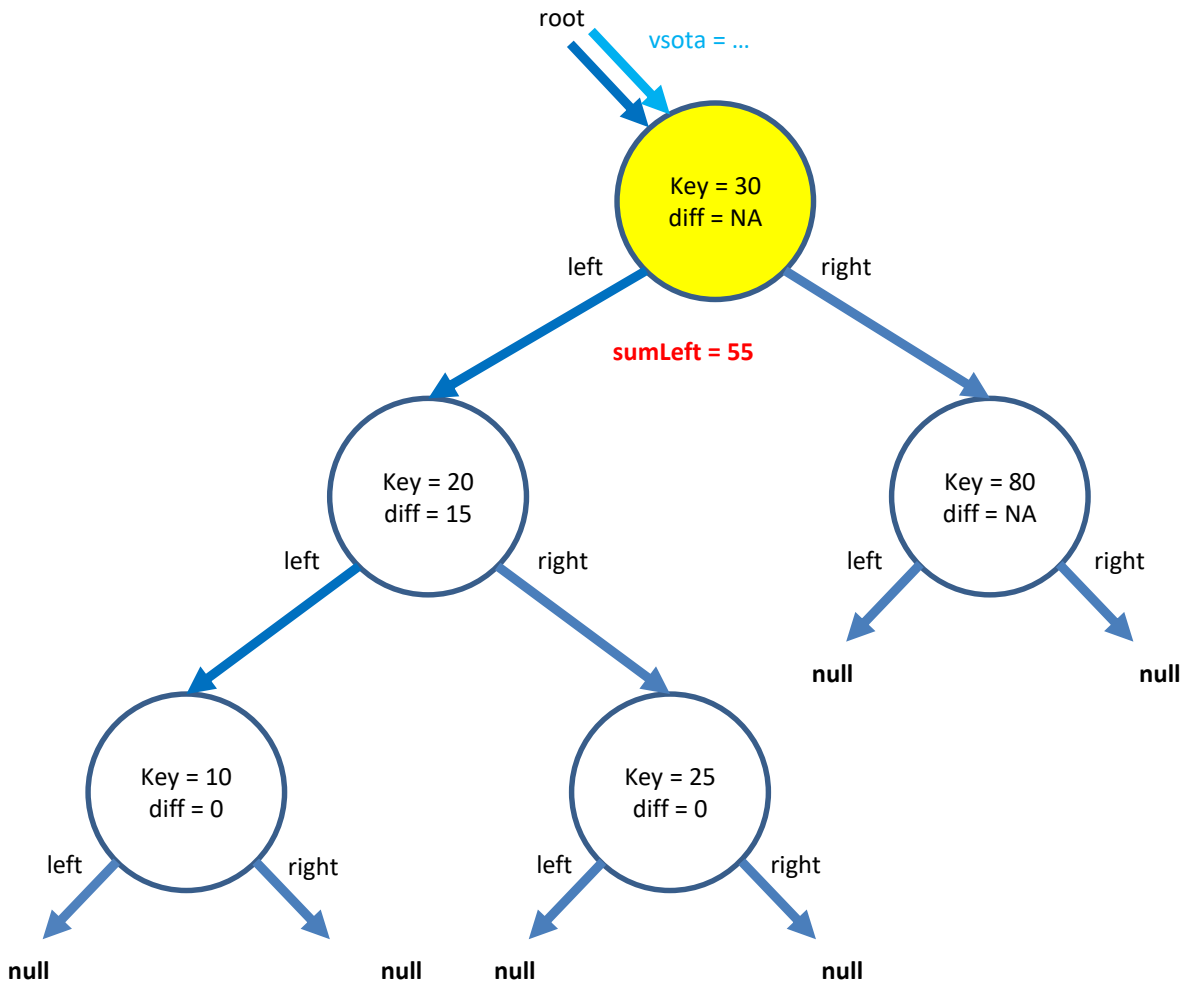
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

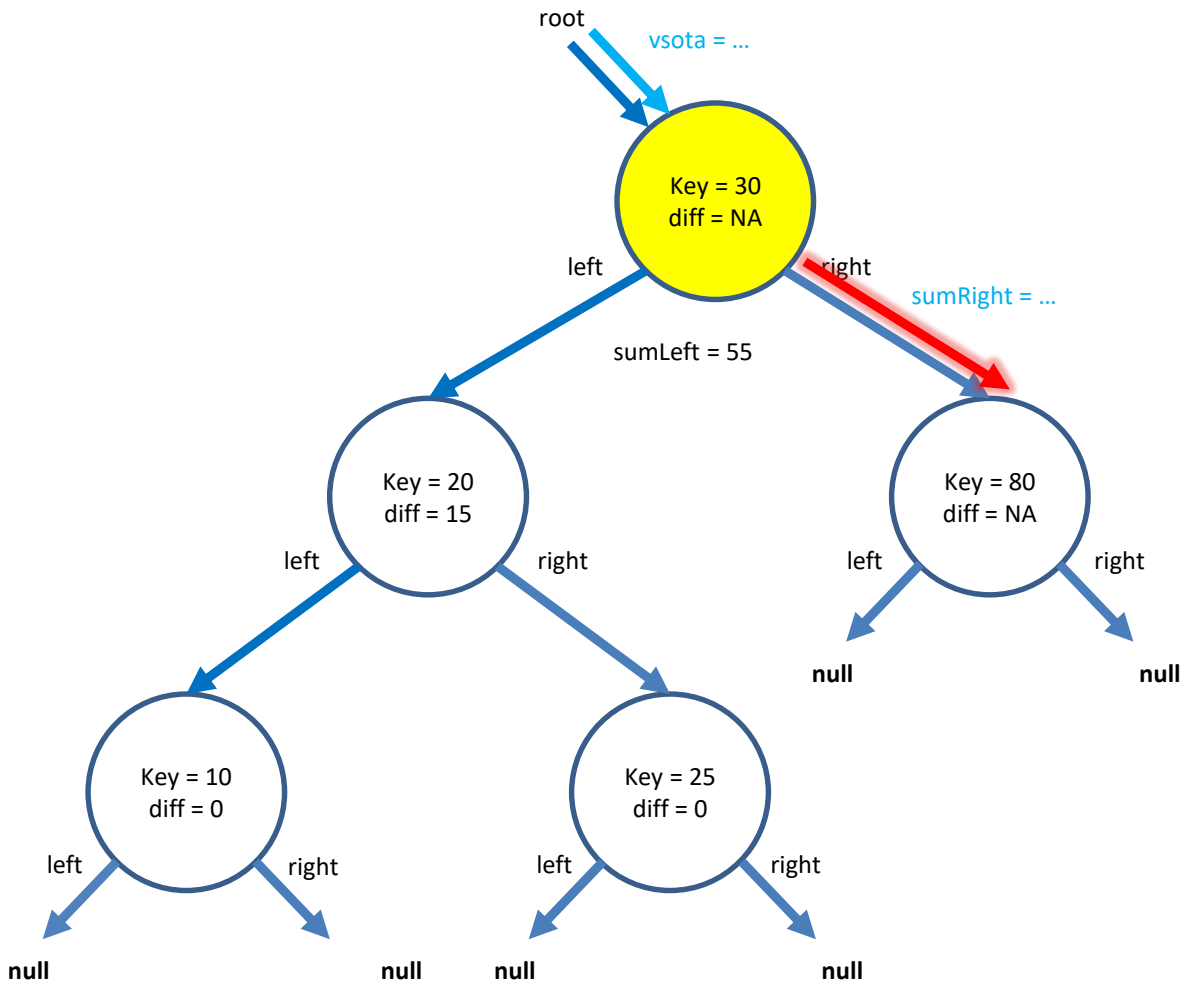
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

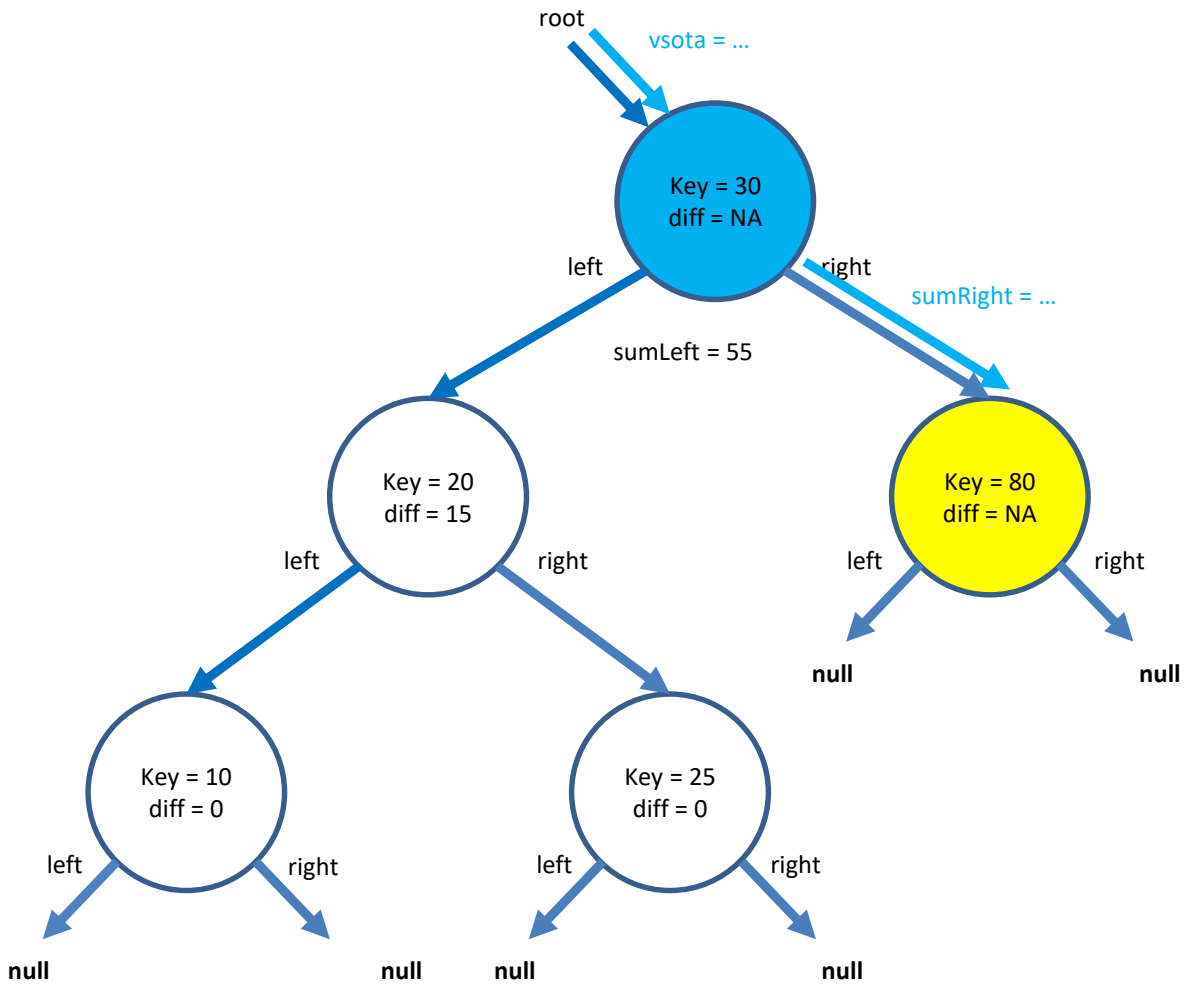
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

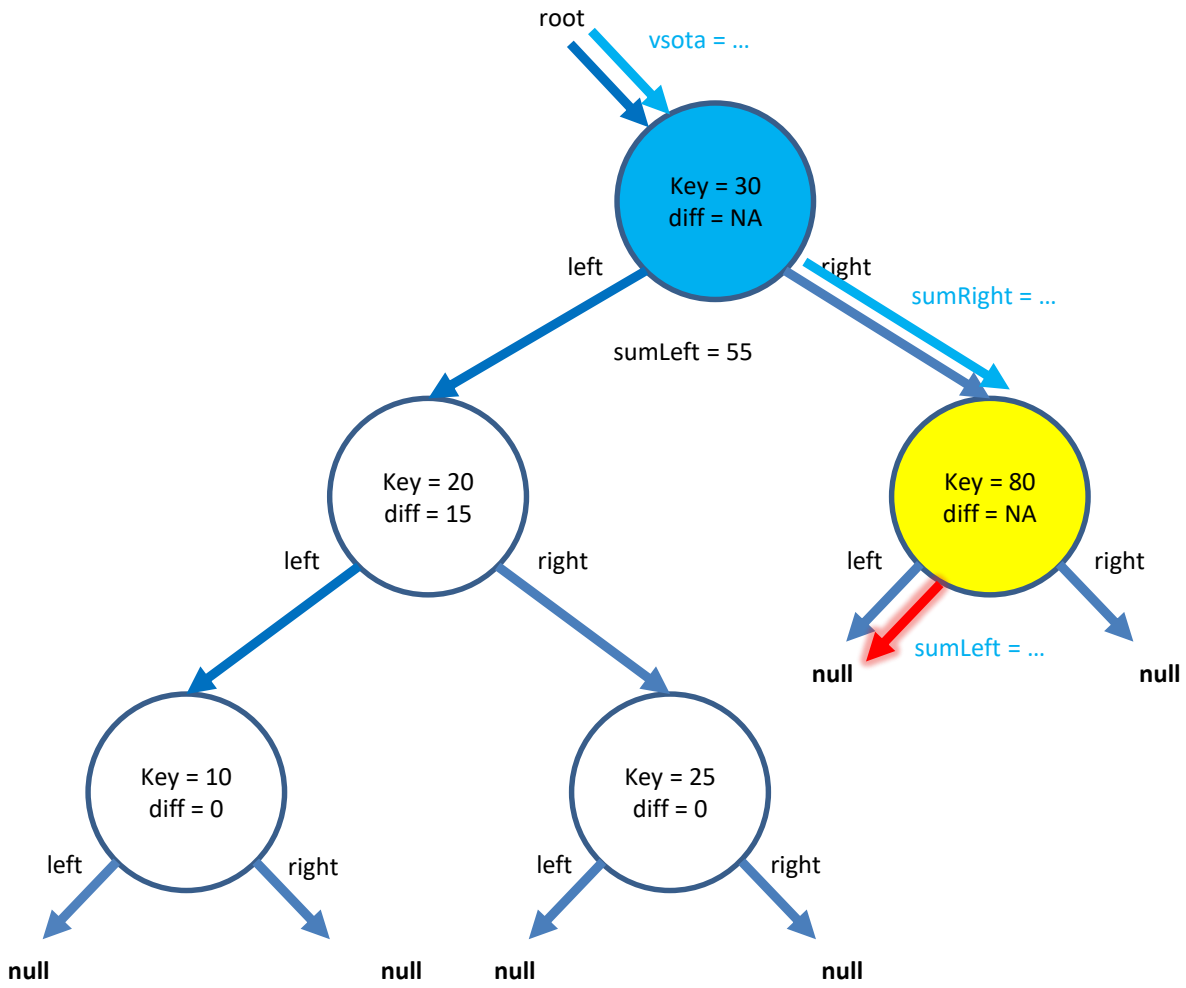
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

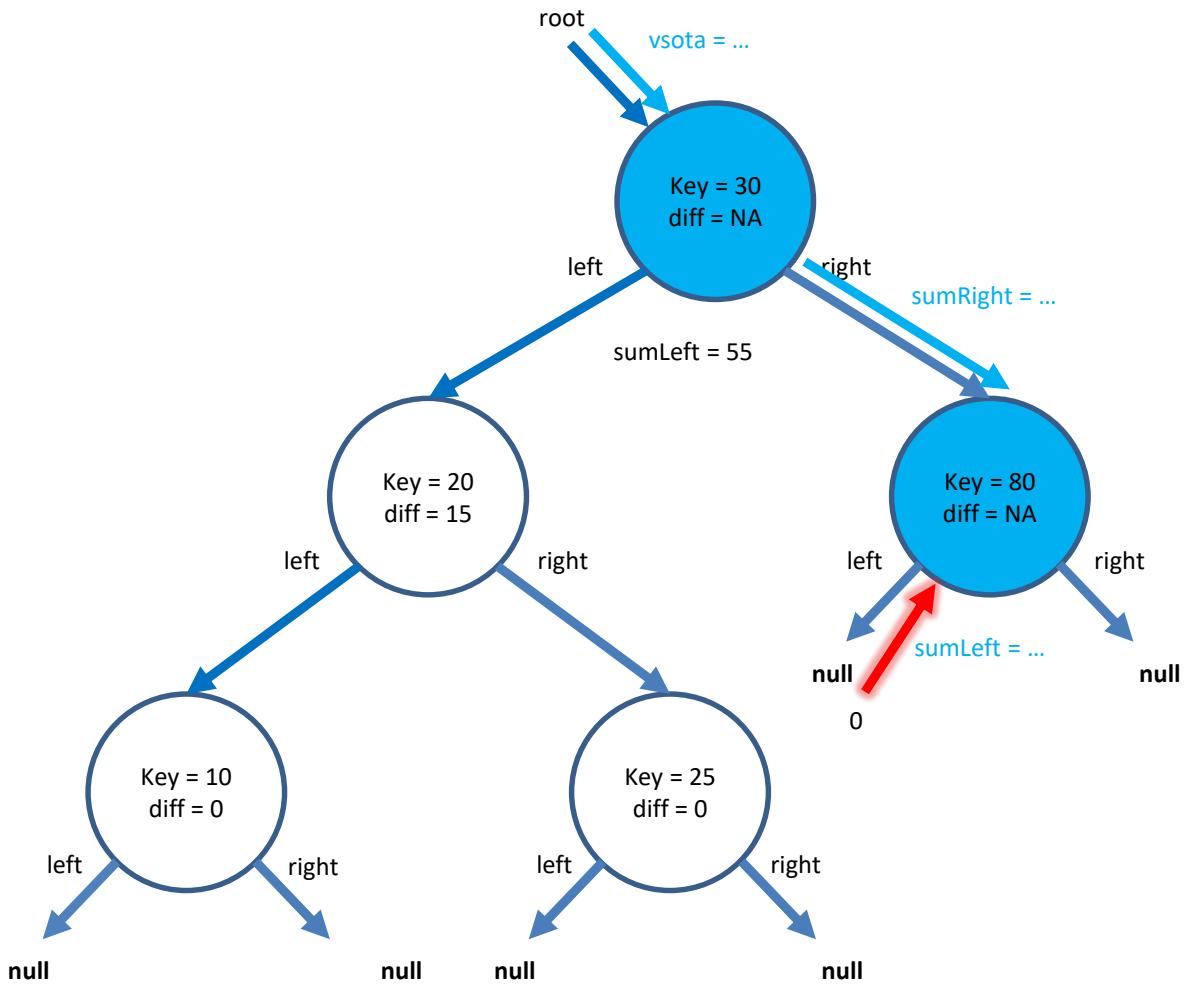
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

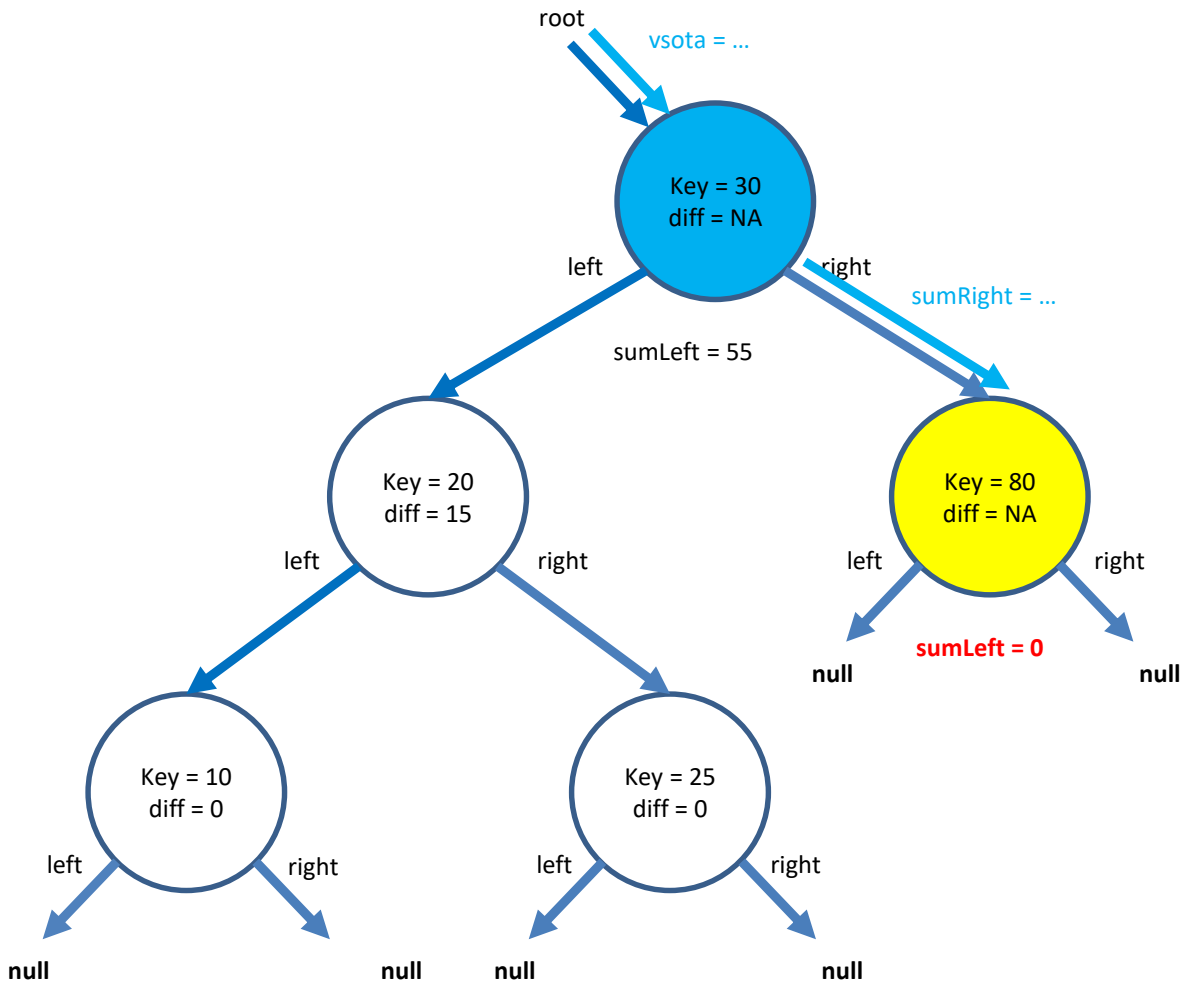
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

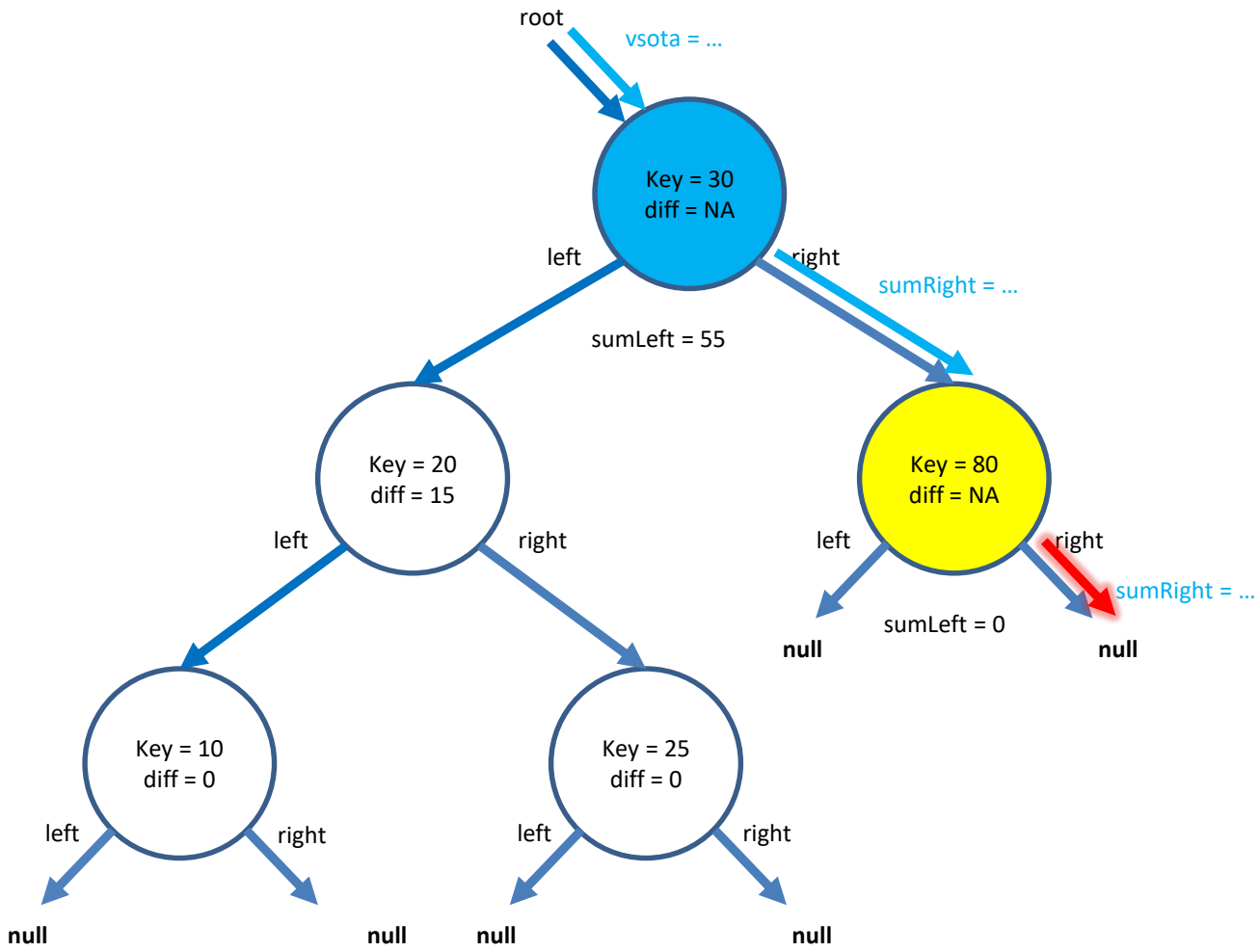
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

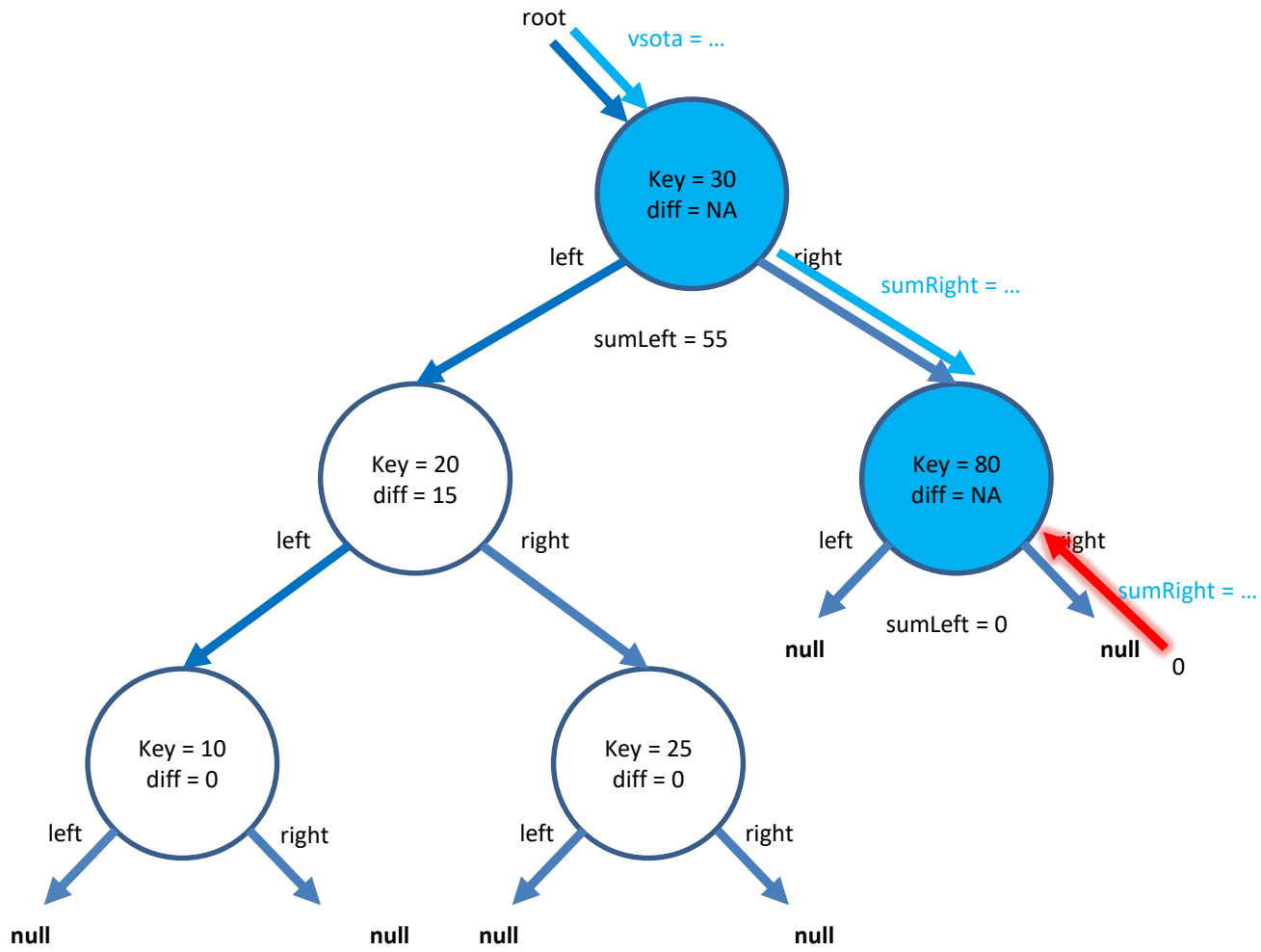
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

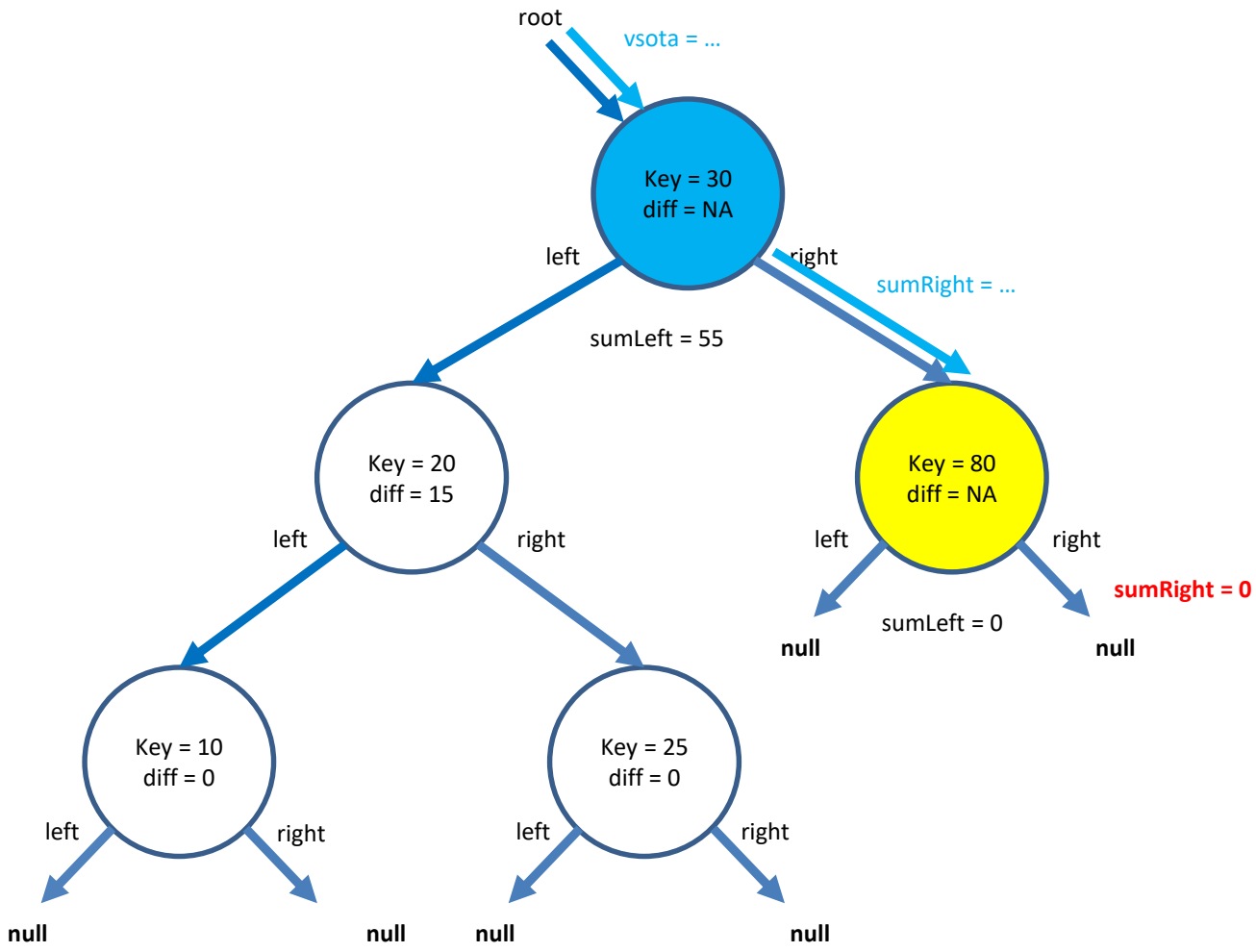
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

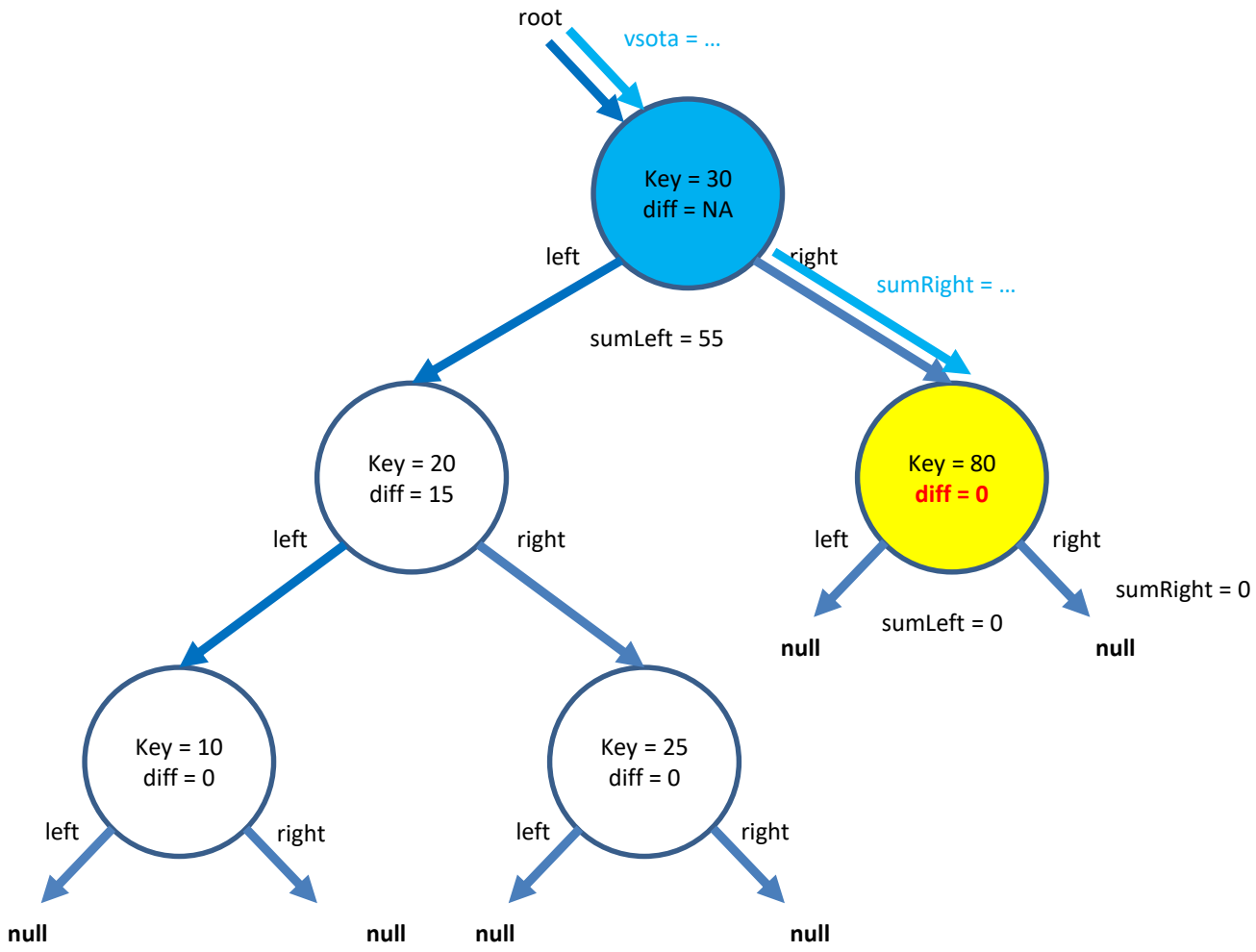
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

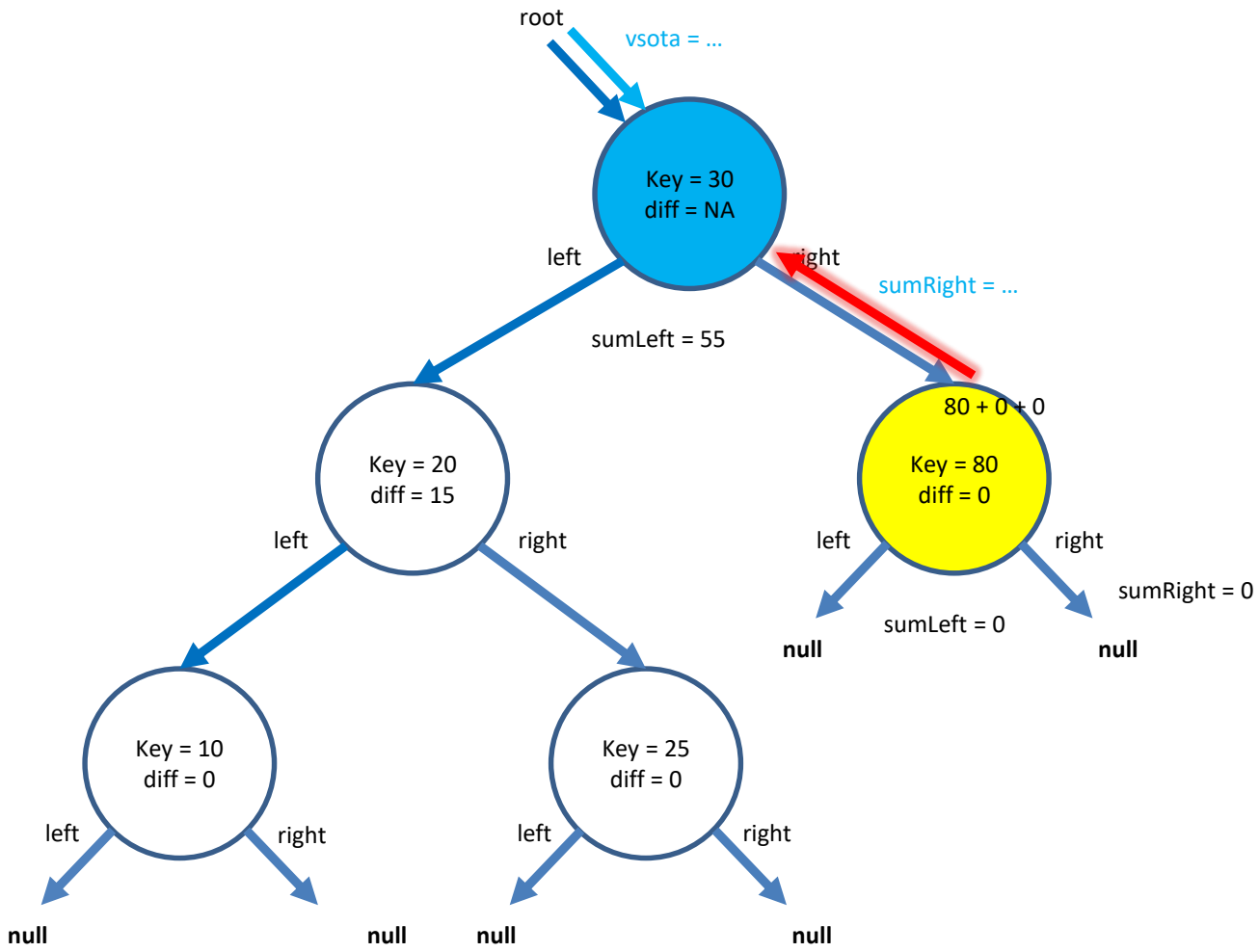
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

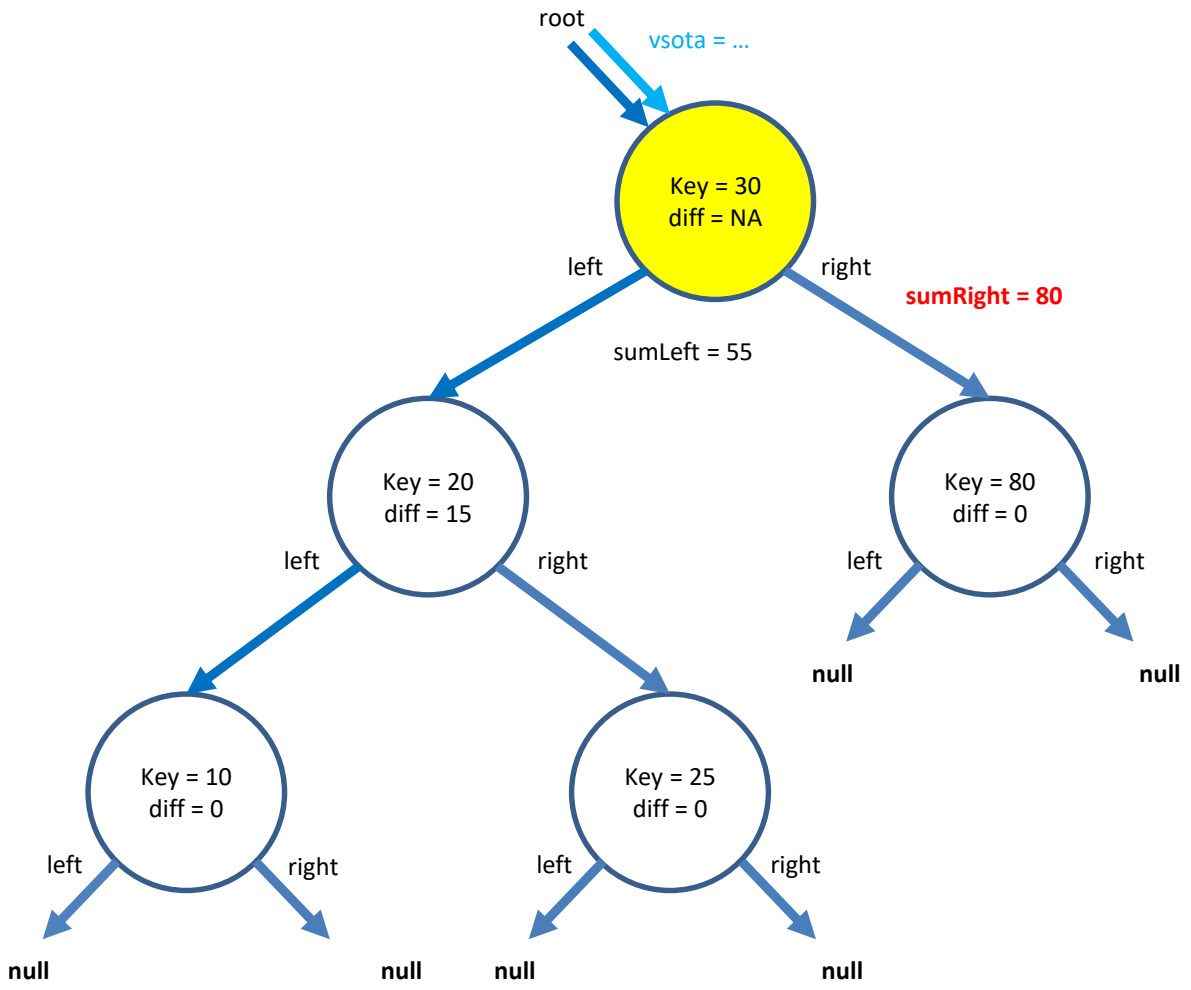
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

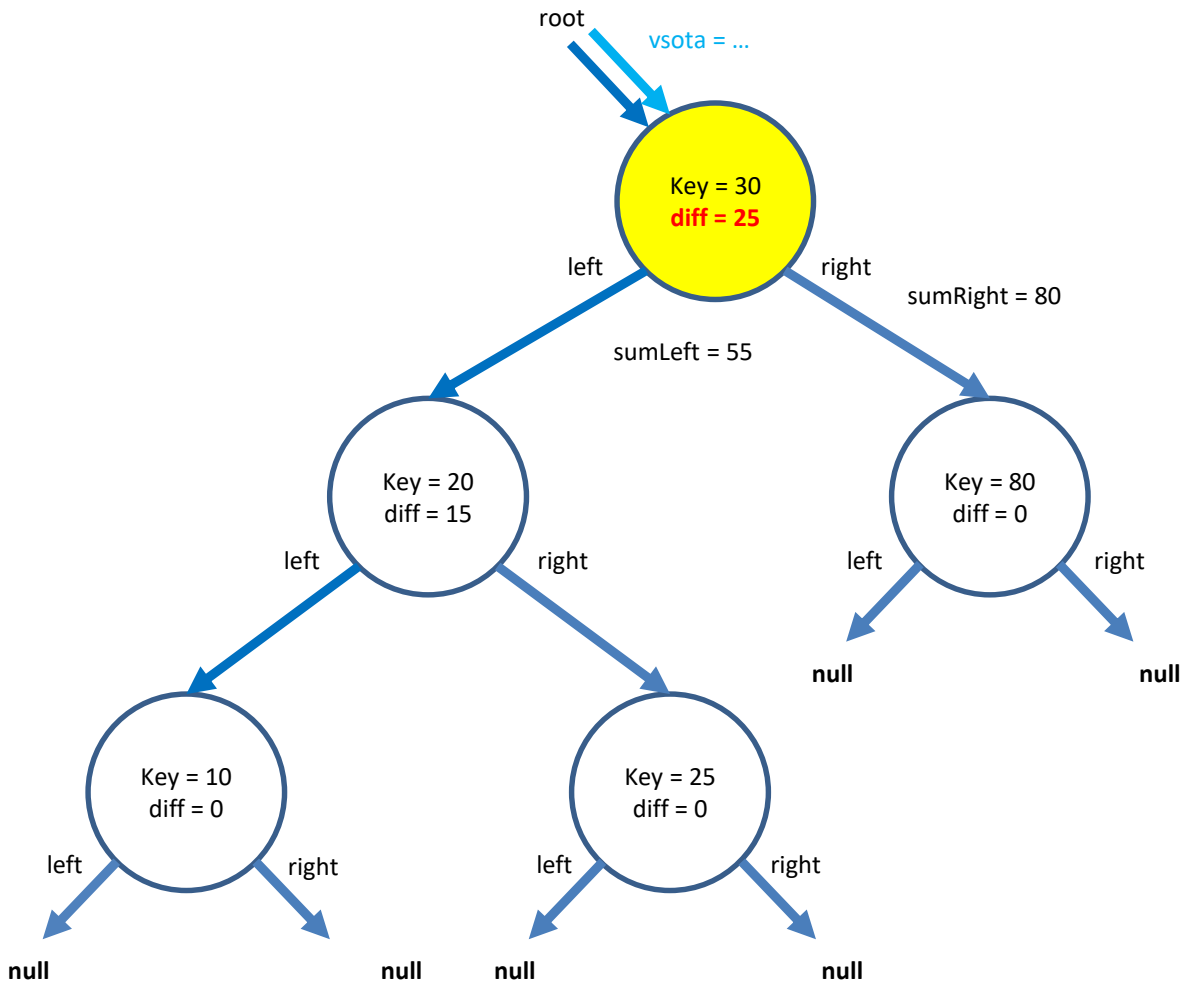
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

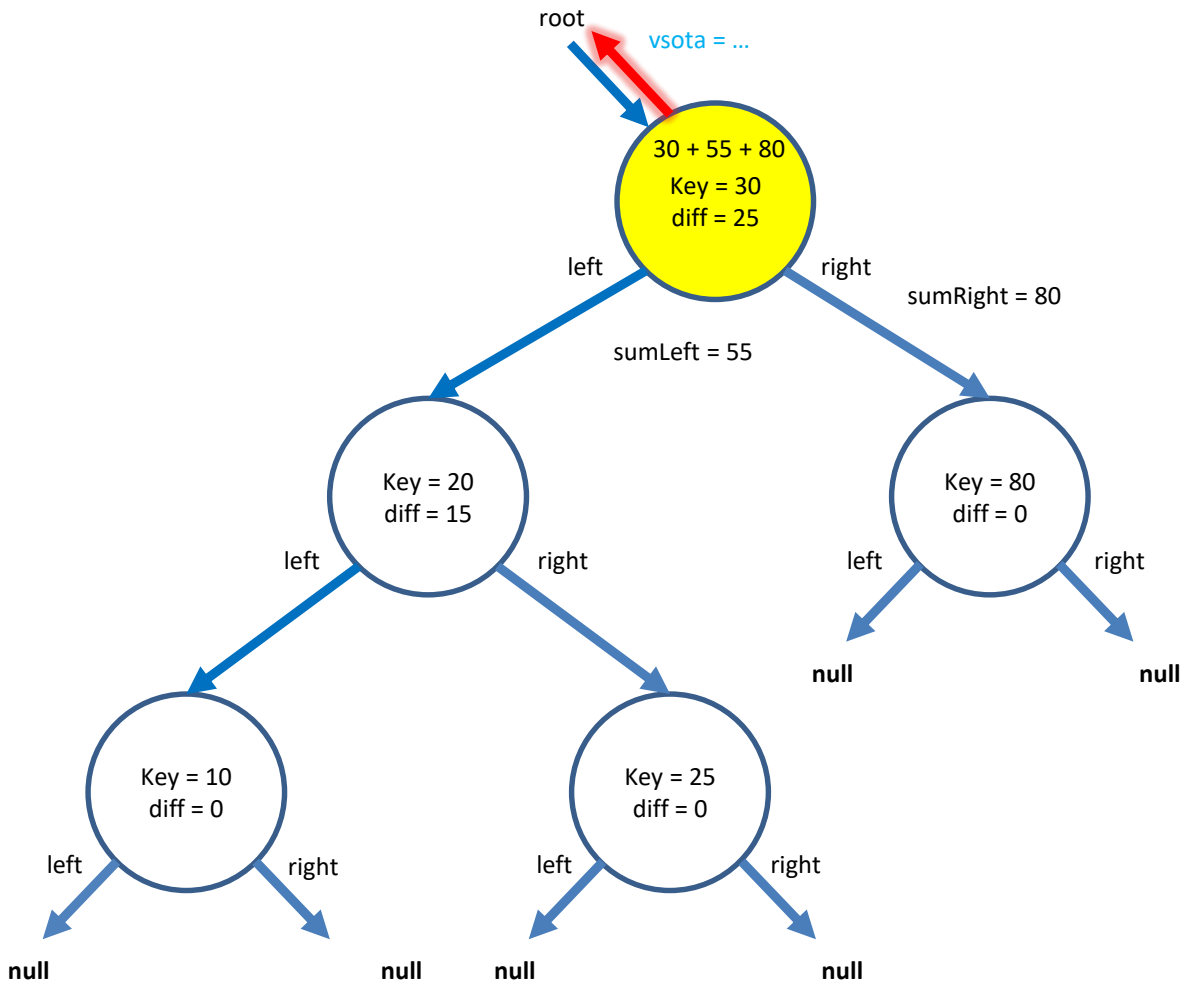
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

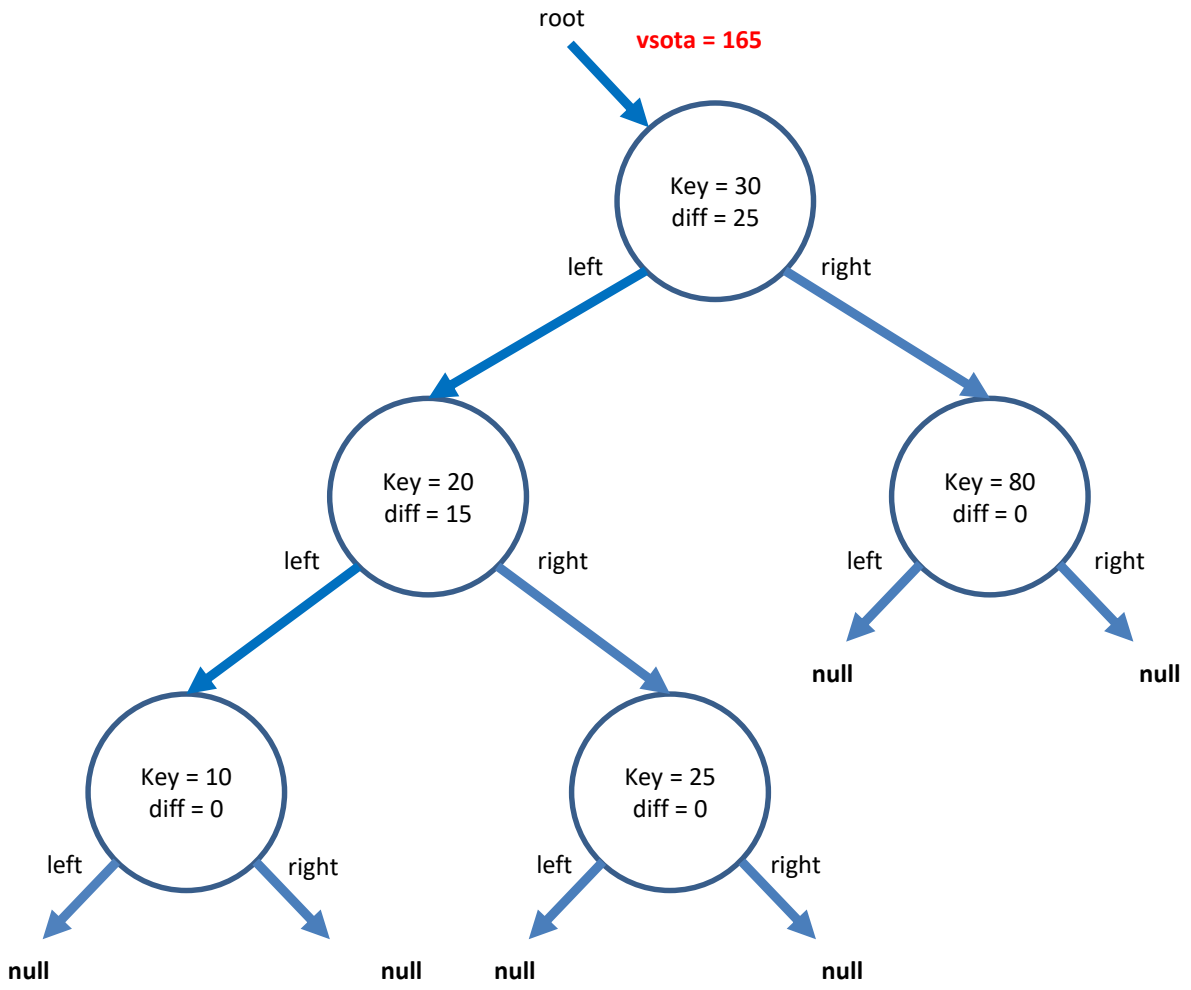
```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

```
public static void main(String[] args) {
    …
    int vsota = sum(root);
    …
}
```



NALOGA 4

Rekurzivno funkcijo iz prejšnje naloge spremenite v iterativno z uporabo sklada.

```
class Node {
    int key, diff;
    Node left, right;
}

public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;

    int sumLeft = sum(node.left);

    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

address0 (začetek funkcije)

address1 (povratek iz prvega rekurzivnega klica)

address2 (povratek iz drugega rekurzivnega klica)

REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;
    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

address0

address1

address2

```
class StackElement // <------------------------ ta struktura hrani navodilo za izvajanje programa
{
    Node node;     // <------------------------ argument rekurzivne funkcije
    int sumLeft;   // <------------------------ lokalna spremenljivka v rek. funkciji
    int sumRight;  // <------------------------ lokalna spremenljivka v rek. funkciji
    int address;   // <------------------------ povratni "naslov"

    public StackElement(){}

    public StackElement(StackElement el) {
        node = el.node;
        sumLeft = el.sumLeft;
        sumRight = el.sumRight;
        address = el.address;
    }
}
```

REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;
    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

address0

address1

address2

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        switch(el.address) {
        case 0:
            // <---------- tukaj pride del kode med oznakama address0 in address1
            break;

        case 1:
            // <---------- tukaj pride del kode med oznakama address1 in address2
            break;

        case 2:
            // <---------- tukaj pride del kode za oznako address2
            break;
        }

    } while (!s.empty());

    return result;
}
```

REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;
    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

address0

address1

address2

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        switch(el.address) {
        case 0:
            // <---------- tukaj pride del kode med oznakama address0 in address1
            break;

        case 1:
            // <---------- tukaj pride del kode med oznakama address1 in address2
            break;

        case 2:
            // <---------- tukaj pride del kode za oznako address2
            break;
        }

    } while (!s.empty());

    return result;
}
```

na sklad postavimo navodilo za začetek izvajanja funkcije

REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;
    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

address0

address1

address2

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        switch(el.address) {
        case 0:
            // <---------- tukaj pride del kode med oznakama address0 in address1
            break;

        case 1:
            // <---------- tukaj pride del kode med oznakama address1 in address2
            break;

        case 2:
            // <---------- tukaj pride del kode za oznako address2
            break;
        }

    } while (!s.empty());

    return result;
}
```

tukaj hranimo rezultat, ki ga vrne rekurzija

REŠITEV

```
public int sum(Node node) {
    if (node == null)                    address0
        return 0;
    int sumLeft = sum(node.left);        address1
    int sumRight = sum(node.right);      address2

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        switch(el.address) {
        case 0:
            // <---------- tukaj pride del kode med oznakama address0 in address1
            break;

        case 1:
            // <---------- tukaj pride del kode med oznakama address1 in address2
            break;

        case 2:
            // <---------- tukaj pride del kode za oznako address2
            break;
        }

    } while (!s.empty());

    return result;
}
```

v vsakem koraku poberemo s sklada navodilo za nadaljevanje izvajanja funkcije

REŠITEV

```
public int sum(Node node) {
    if (node == null)                    <-- address0
        return 0;
    int sumLeft = sum(node.left);        <-- address1
    int sumRight = sum(node.right);      <-- address2

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

```
...
        switch(el.address) {
        case 0:
            if (el.node == null)
                result = 0;
            else
            {
                el.address = 1;
                s.push(new StackElement(el));

                el.node = el.node.left;
                el.address = 0;
                s.push(new StackElement(el));
            }
            break;

        case 1:
            // <---------- tukaj pride del kode med oznakama address1 in address2
            break;

        case 2:
            // <---------- tukaj pride del kode za oznako address2
            break;
        }

...
```

po povratku iz rekurzije nadaljujemo na naslovu "address1" (z istim vhodnim argumentom)

rekurzivni klic pomeni, da začnemo na naslovu "address0" (z novim vhodnim argumentom)

REŠITEV

```
public int sum(Node node) {
    if (node == null)
        return 0;
    int sumLeft = sum(node.left);
    int sumRight = sum(node.right);

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

address0

address1

address2

```
...
        switch(el.address) {
        case 0:
            if (el.node == null)
                result = 0;
            else
            {
                el.address = 1;
                s.push(new StackElement(el));

                el.node = el.node.left;
                el.address = 0;
                s.push(new StackElement(el));
            }
            break;

        case 1:
            el.sumLeft = result;

            el.address = 2;
            s.push(new StackElement(el));

            el.node = el.node.right;
            el.address = 0;
            s.push(new StackElement(el));
            break;

        case 2:
            // <---------- tukaj pride del kode za oznako address2
            break;
        }

...
```

najprej shranimo rezultat prve rekurzije v lok. spr. trenutnega konteksta

po povratku iz druge rekurzije nadaljujemo na naslovu "address2" (z istim kontekstom)

ob klicu druge rekurzije ponovno začnemo na naslovu "address0" (z novim vhodnim argumentom)

REŠITEV

```
public int sum(Node node) {
    if (node == null)                    address0
        return 0;
    int sumLeft = sum(node.left);        address1
    int sumRight = sum(node.right);      address2

    node.diff = sumRight - sumLeft;
    return node.key + sumLeft + sumRight;
}
```

```
...
        switch(el.address) {
        case 0:
            if (el.node == null)
                result = 0;
            else
            {
                el.address = 1;
                s.push(new StackElement(el));

                el.node = el.node.left;
                el.address = 0;
                s.push(new StackElement(el));
            }
            break;

        case 1:
            el.sumLeft = result;

            el.address = 2;
            s.push(new StackElement(el));

            el.node = el.node.right;
            el.address = 0;
            s.push(new StackElement(el));
            break;

        case 2:
            el.sumRight = result;
            el.node.diff = el.sumRight - el.sumLeft;
            result = el.node.key + el.sumLeft + el.sumRight;
            break;
        }

...
```

shranimo rezultat druge rekurzije v lok. spr. trenutnega konteksta,
posodobimo lastnost "diff" v trenutnem vozlišču,
ter si zapomnimo rezultat, ki ga vrne funkcija

REŠITEV

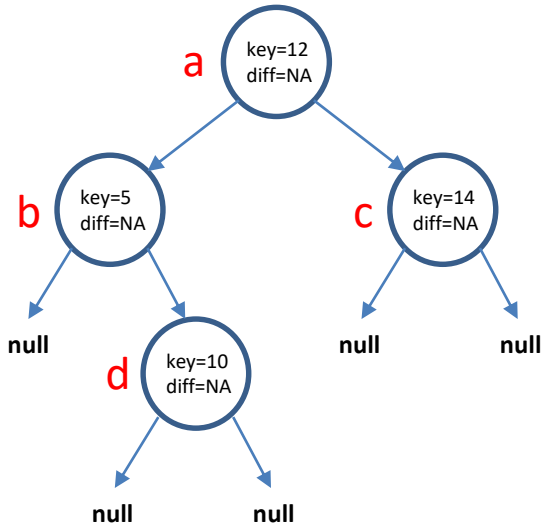
```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

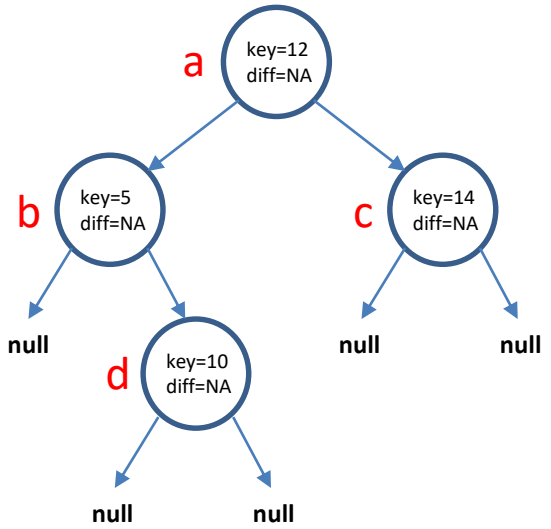


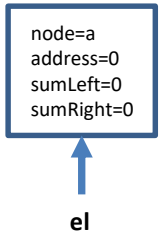
Poglejmo primer izvajanja ob klicu: `sumIterStack(a)`

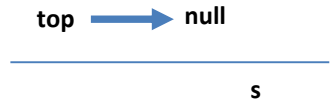
REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```





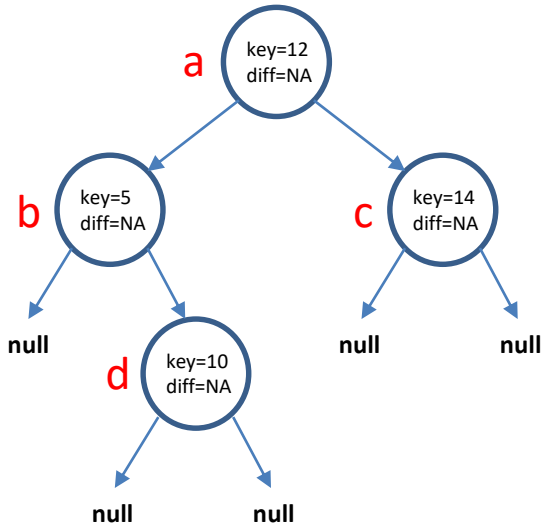


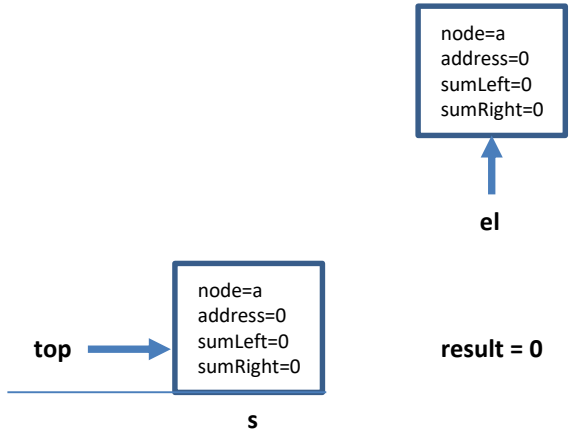
result = 0

REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

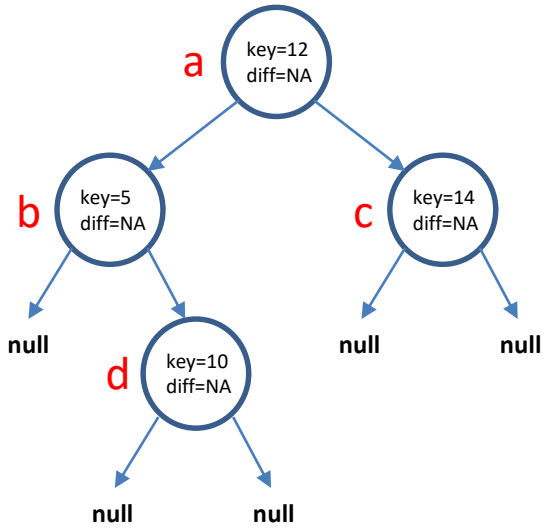


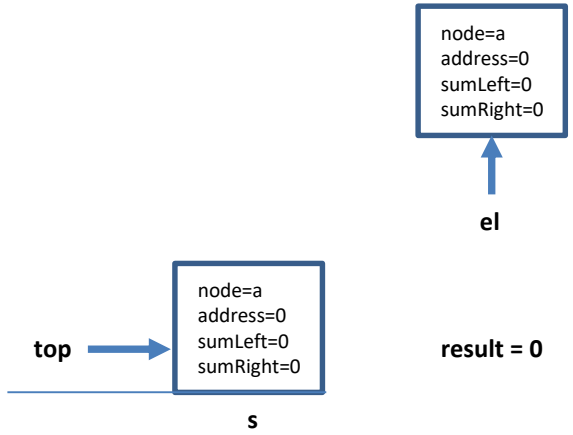


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

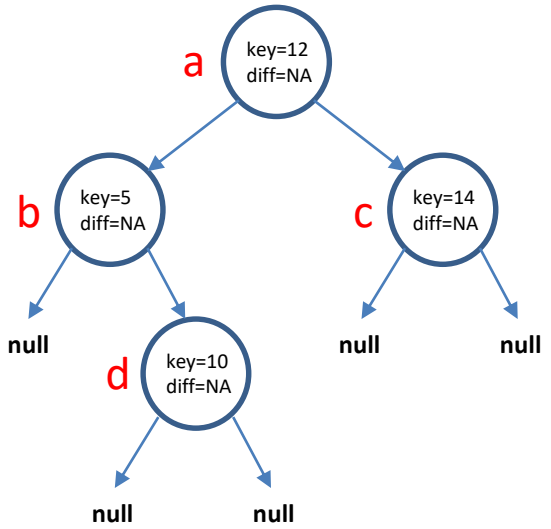


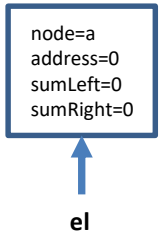


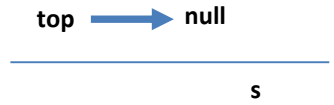
REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```





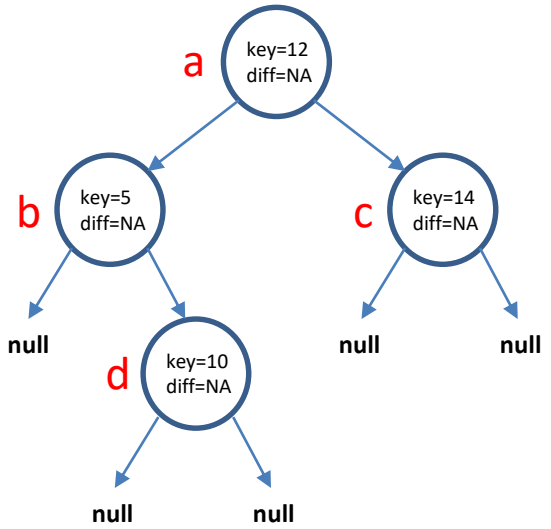


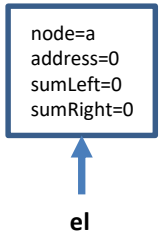
result = 0

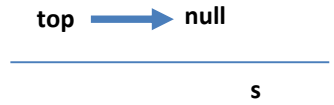
REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```





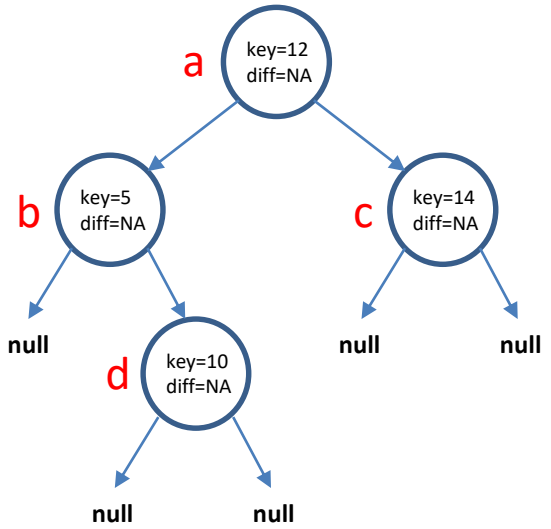


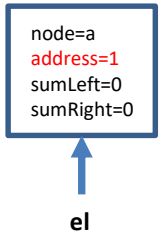
result = 0

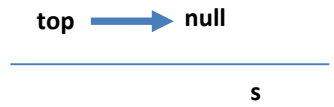
REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```





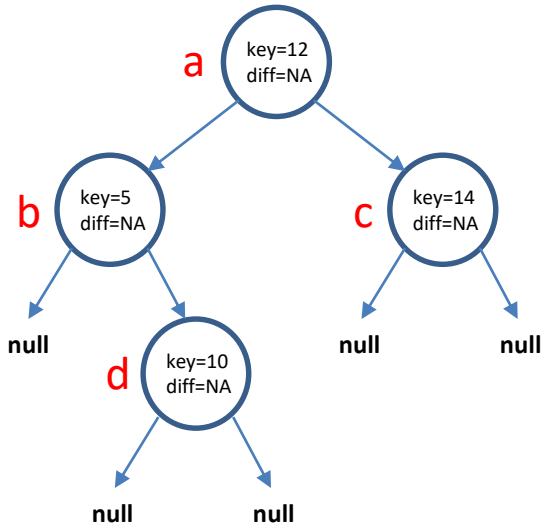


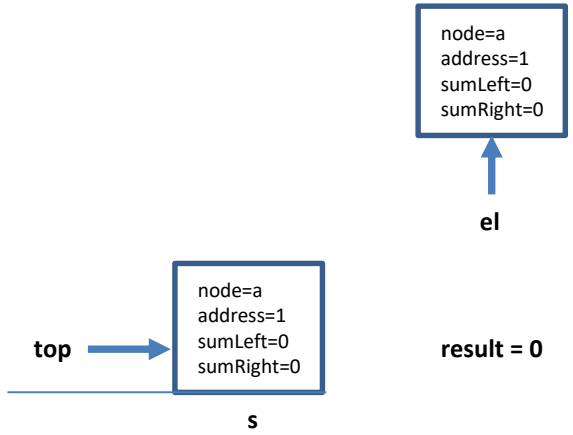
result = 0

REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

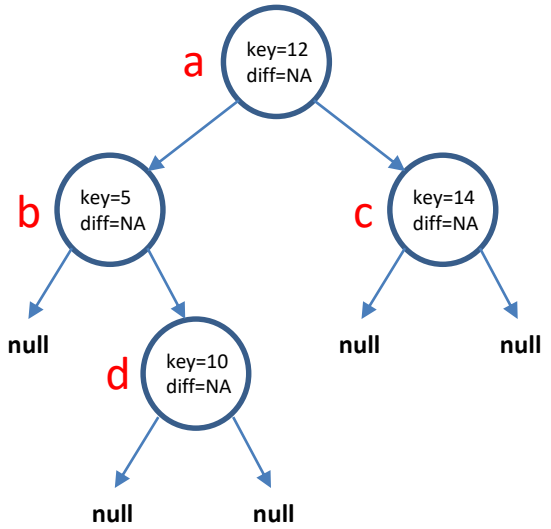


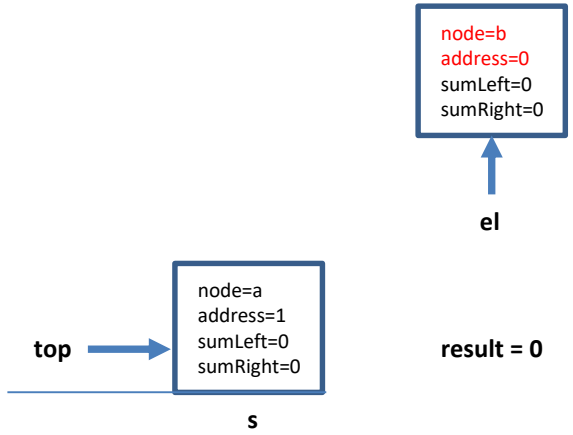


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

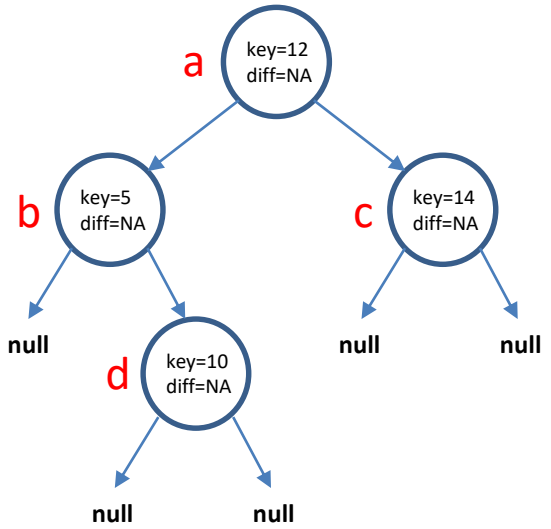


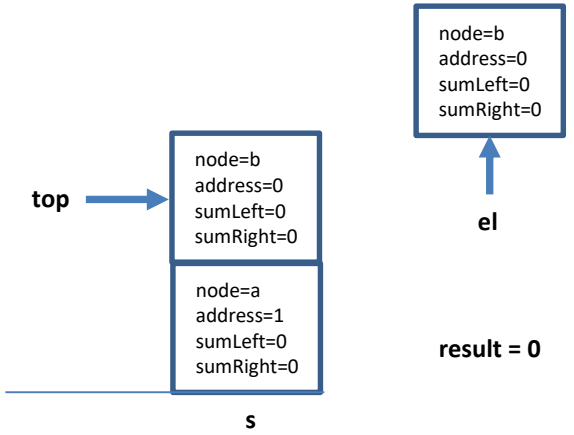


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

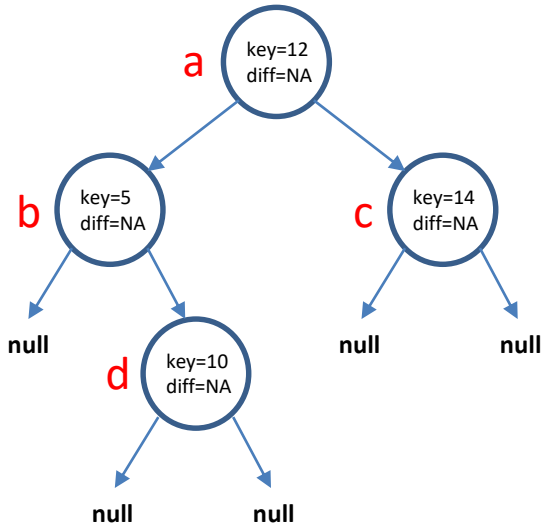


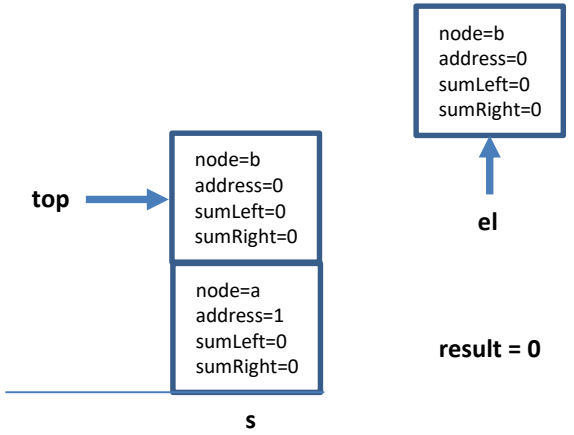


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

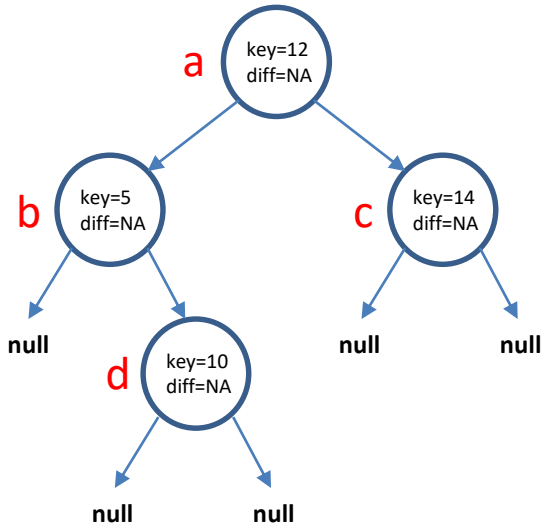


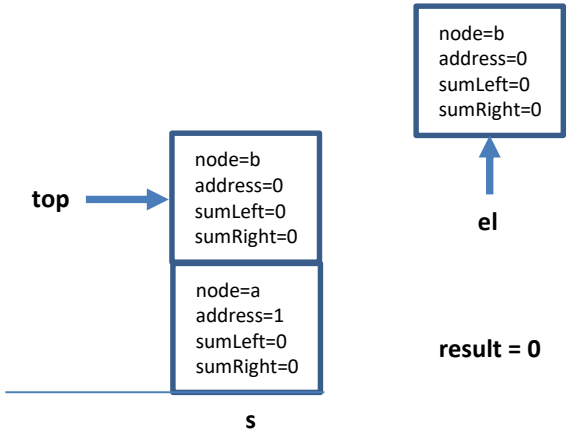


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

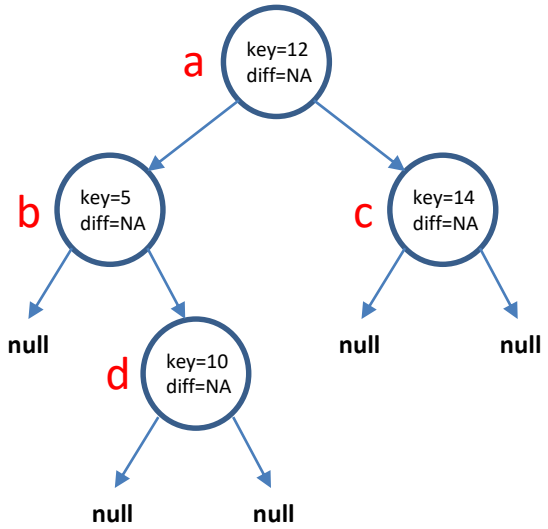


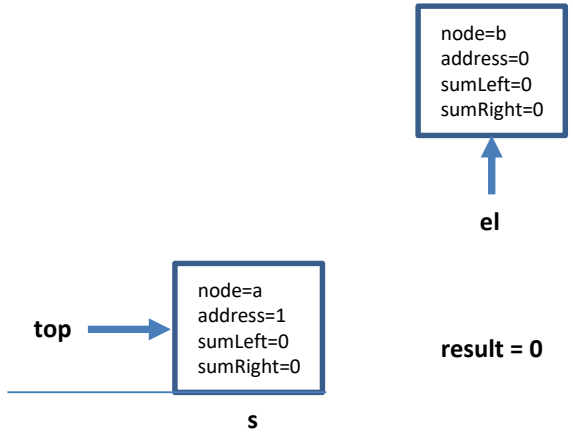


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

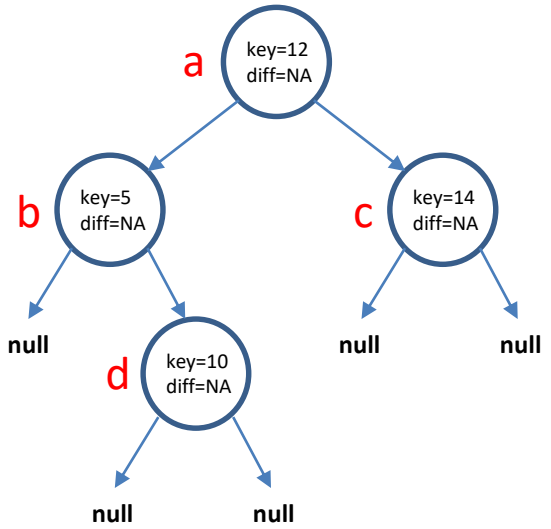


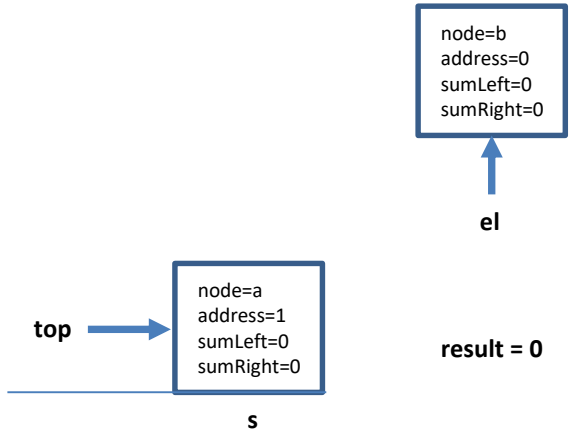


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

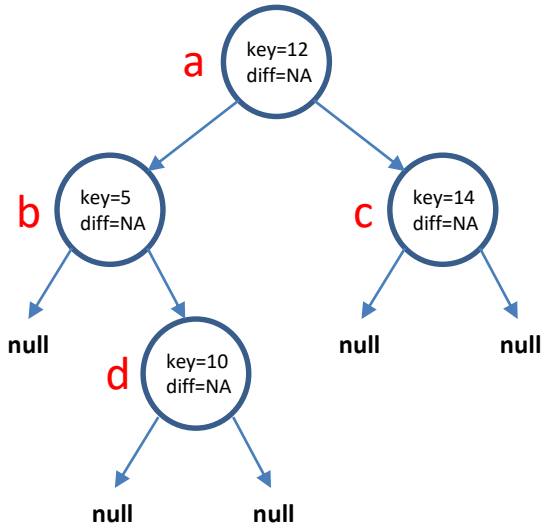


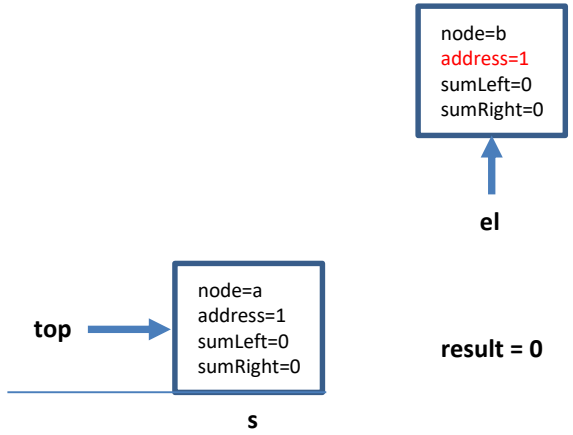


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

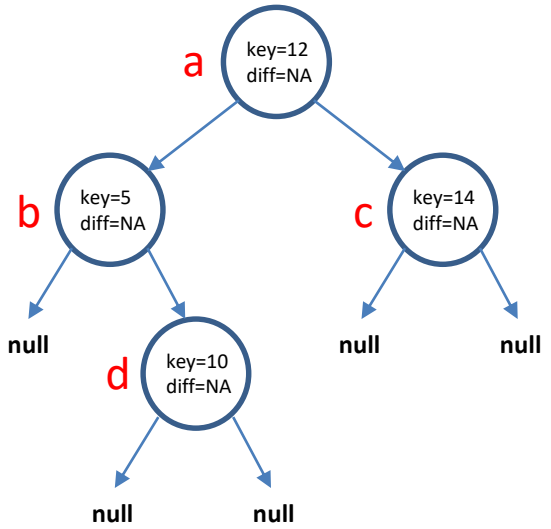


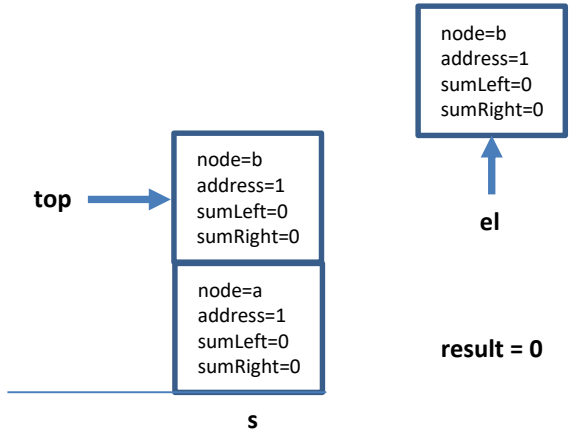


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

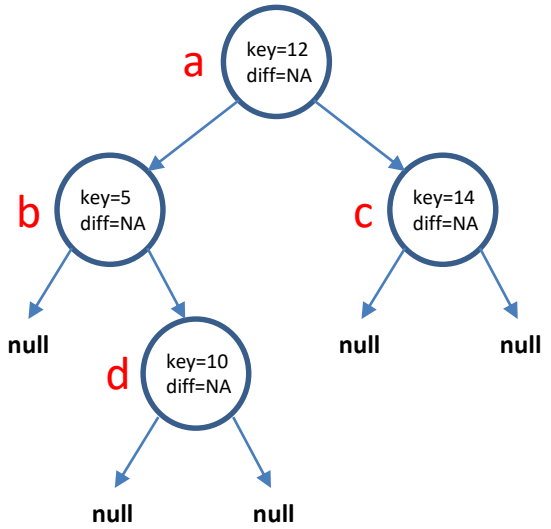


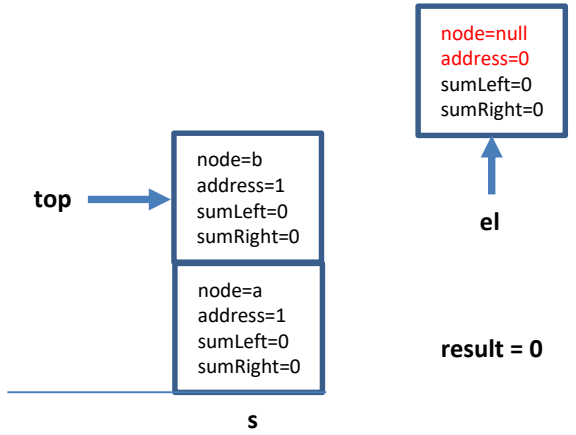


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

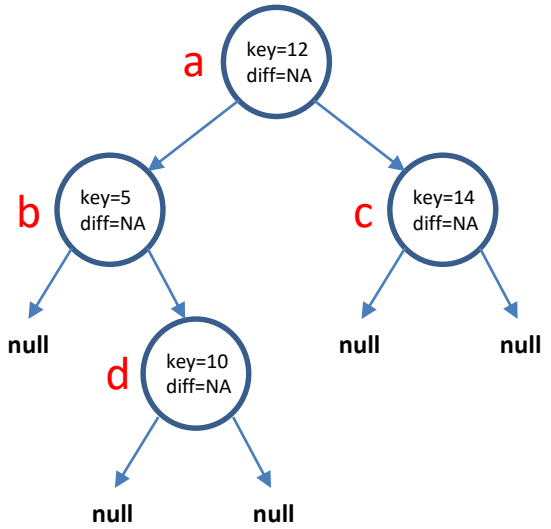


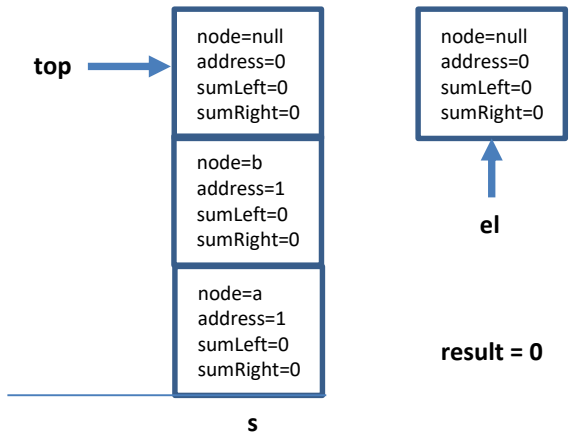


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

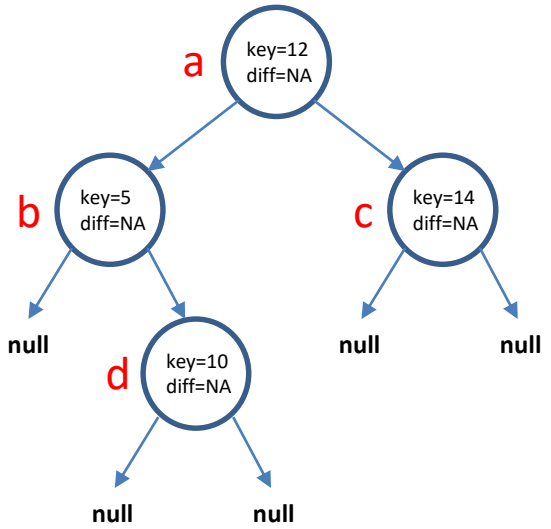


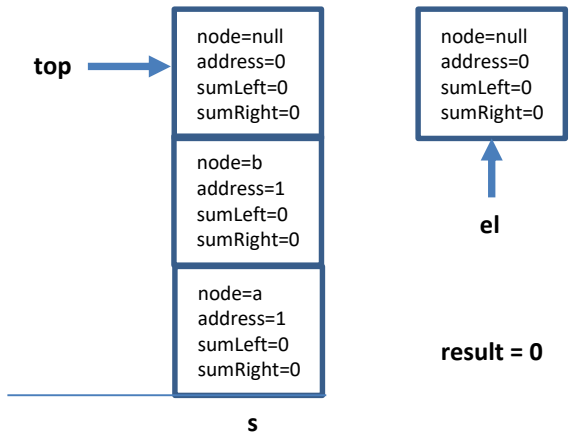


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

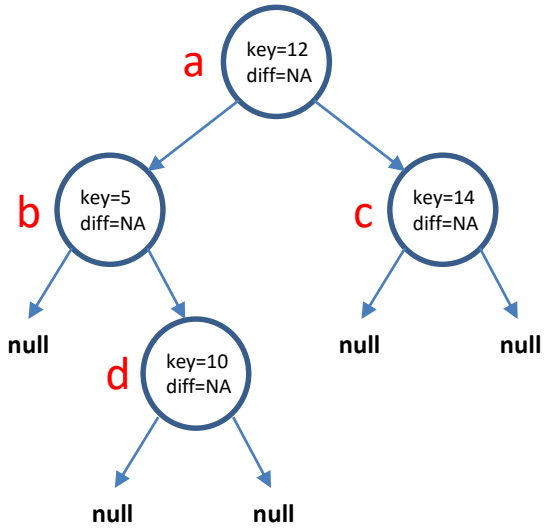


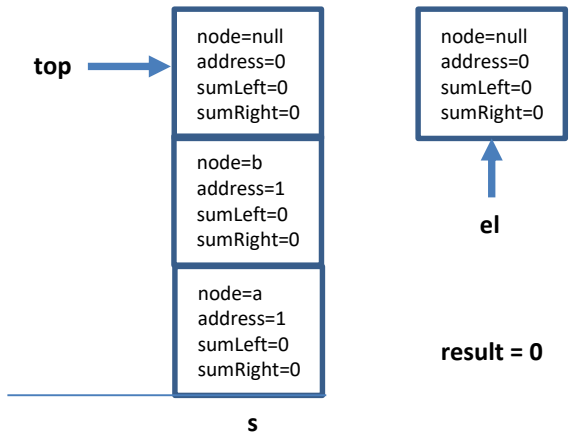


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

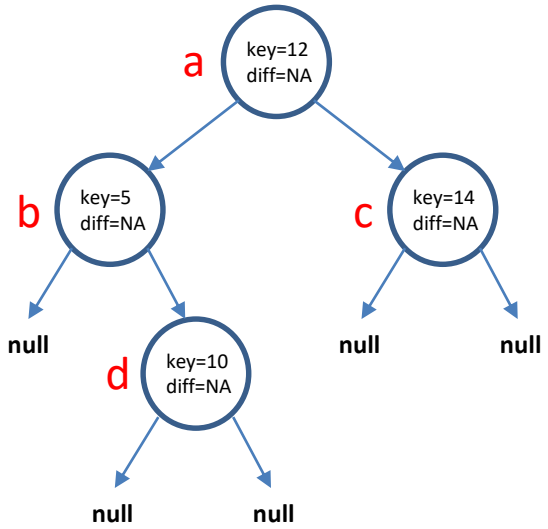


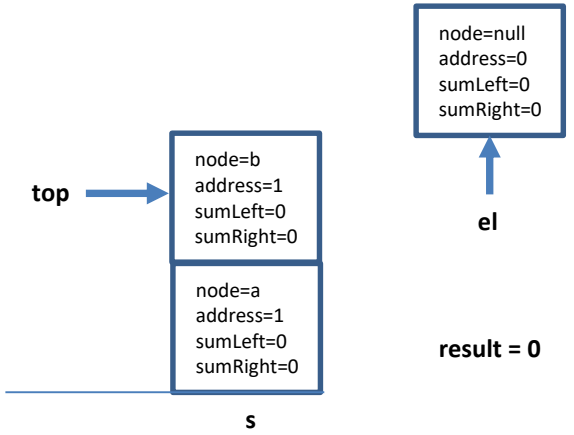


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

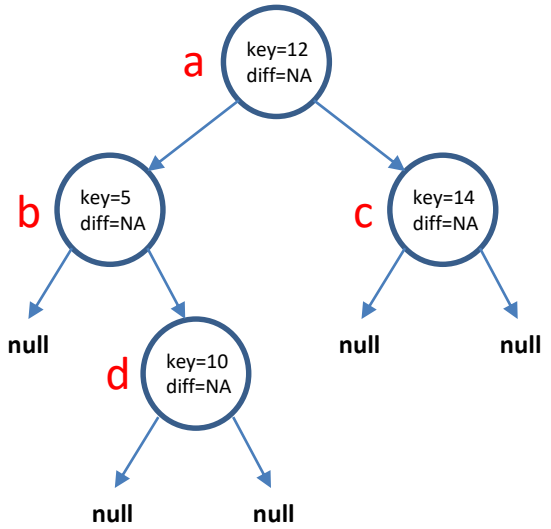


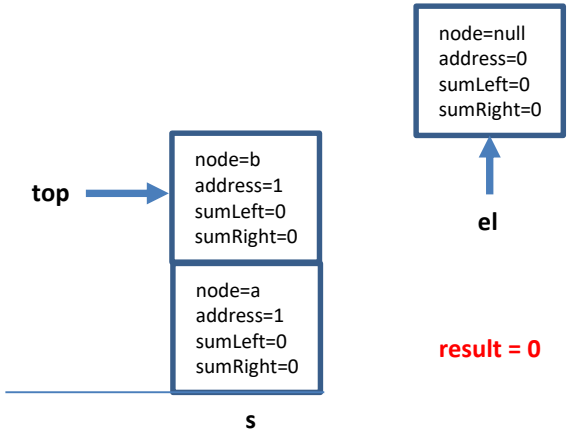


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

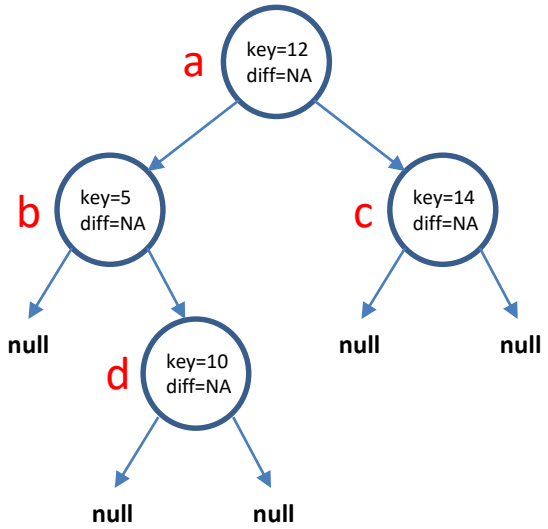


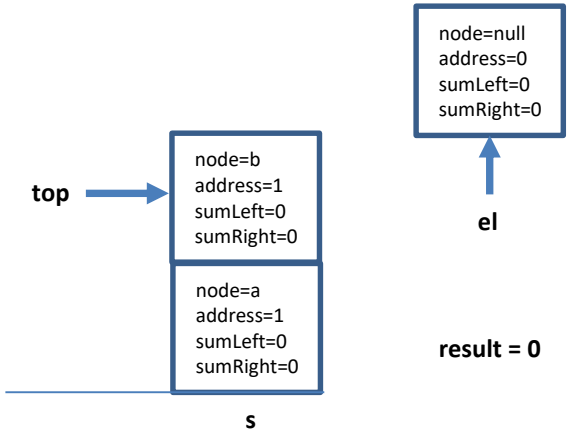


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

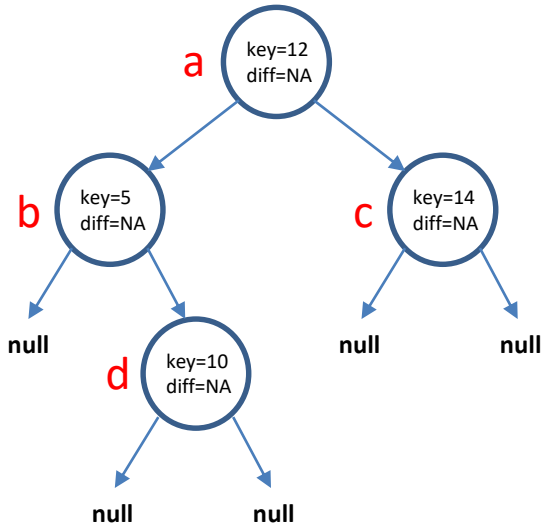


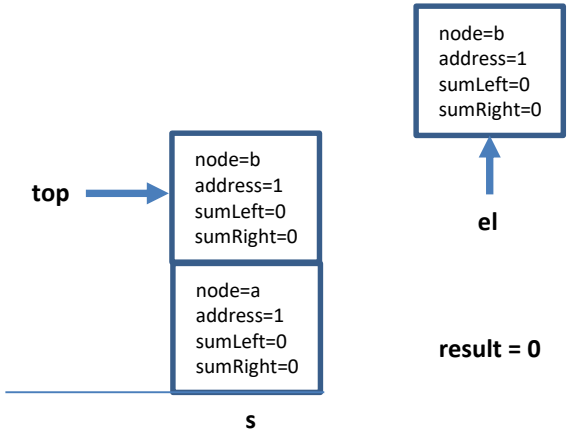


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

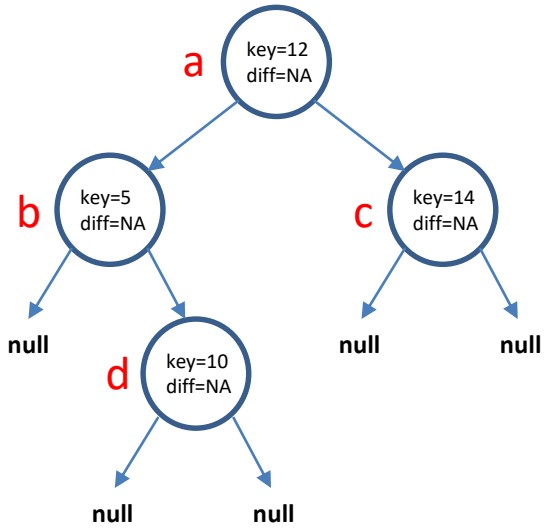


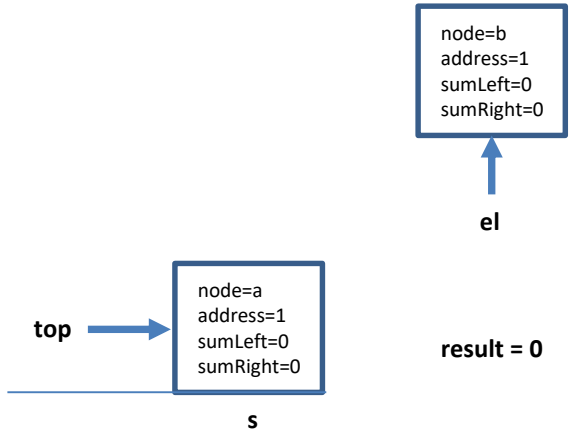


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

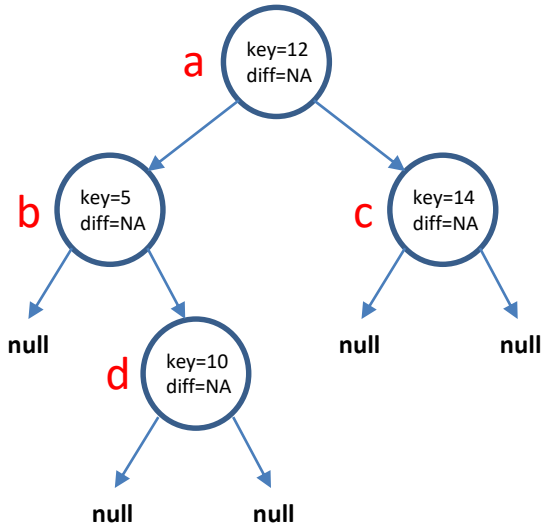


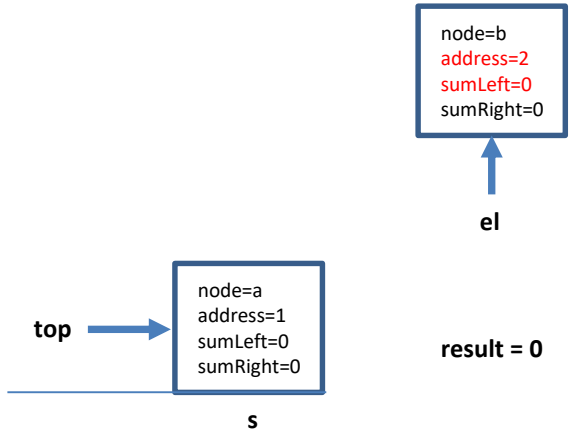


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

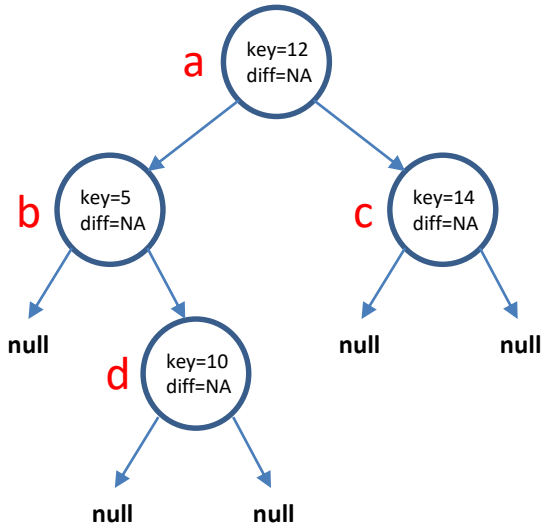


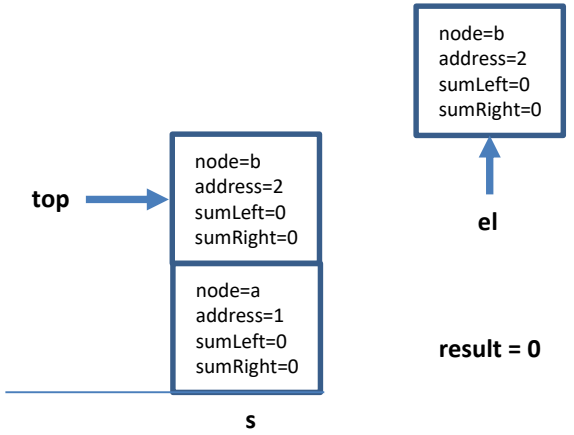


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

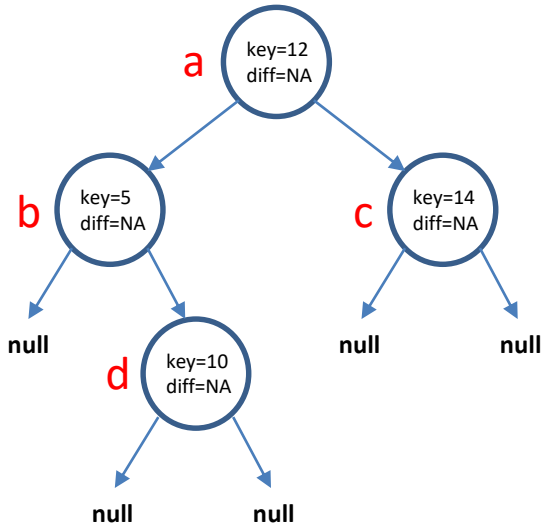


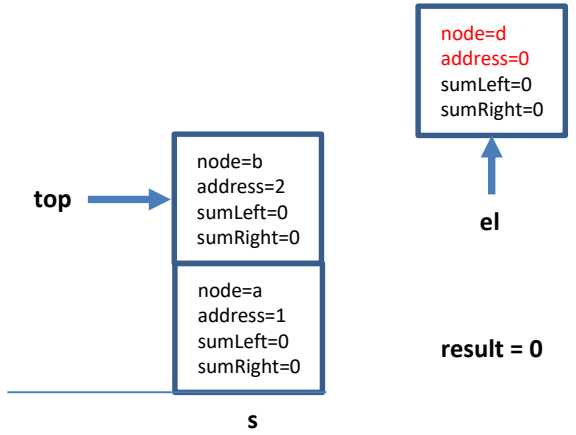


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

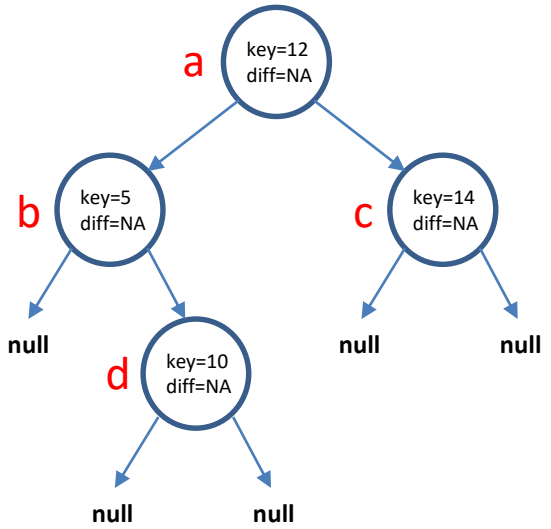


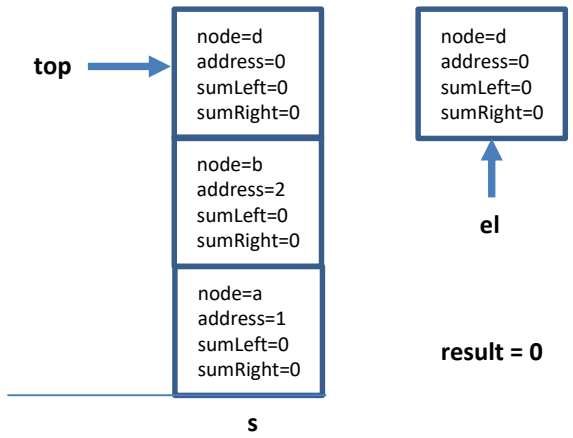


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

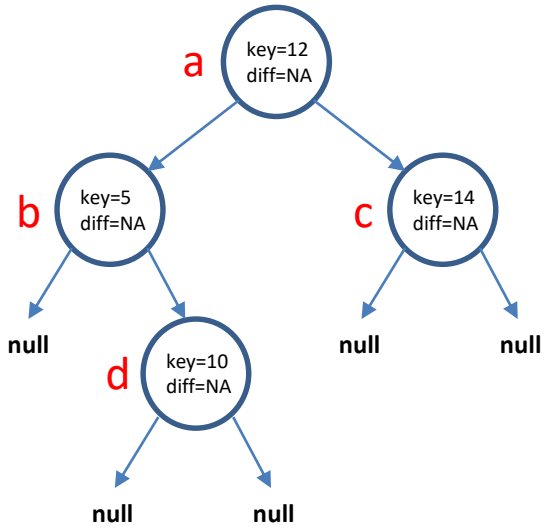


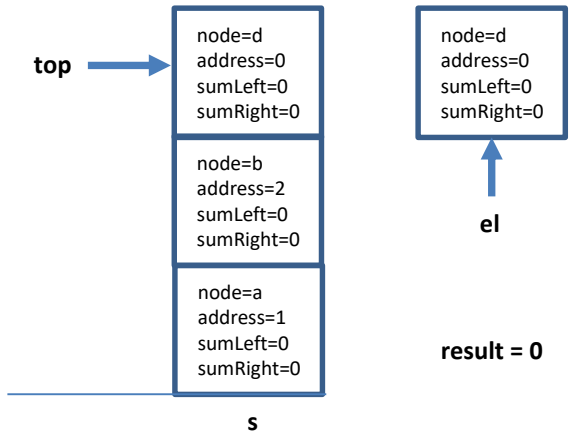


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

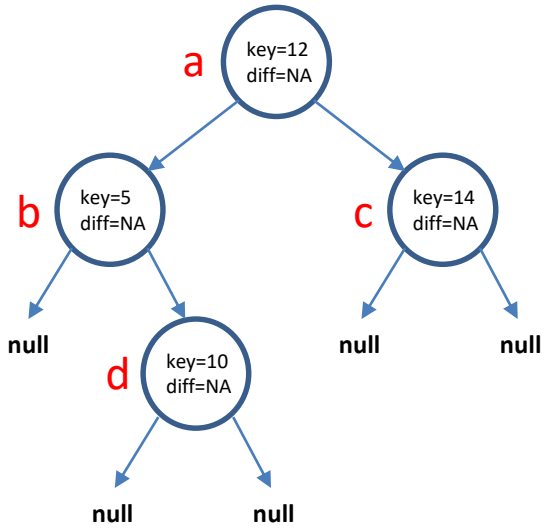


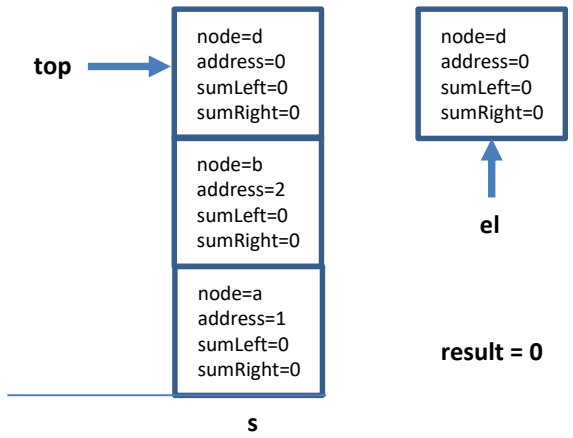


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

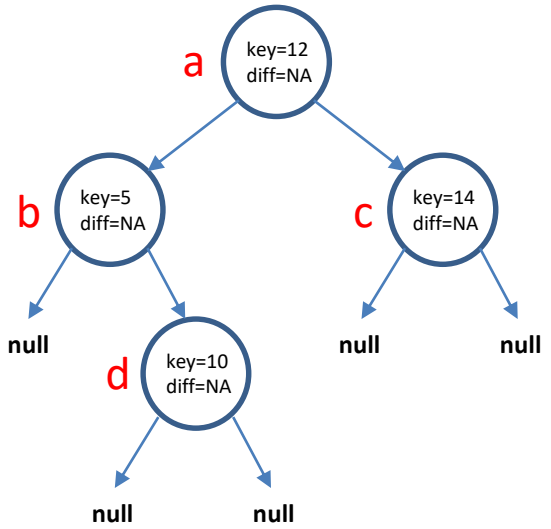


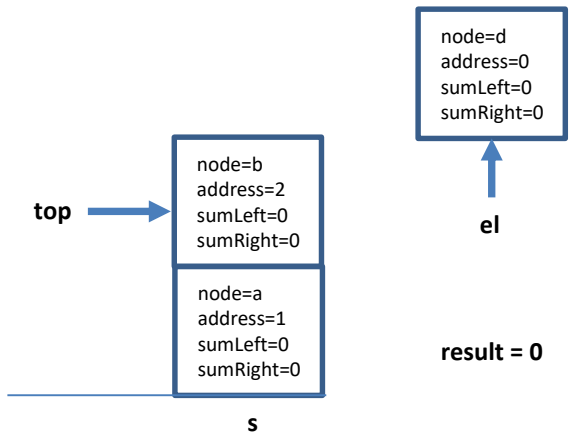


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

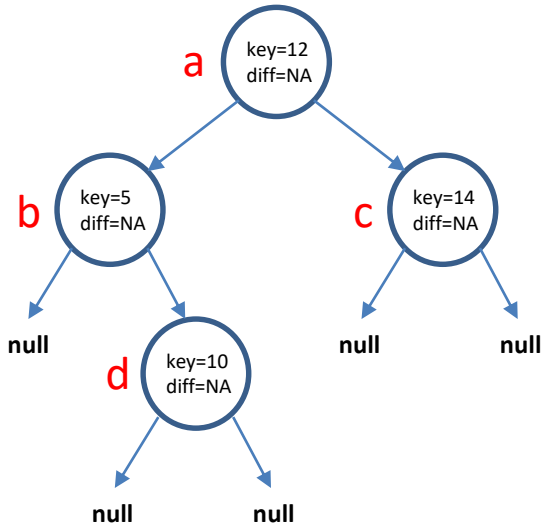


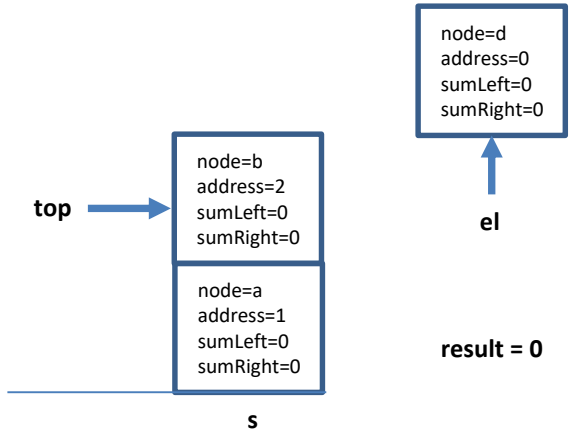


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

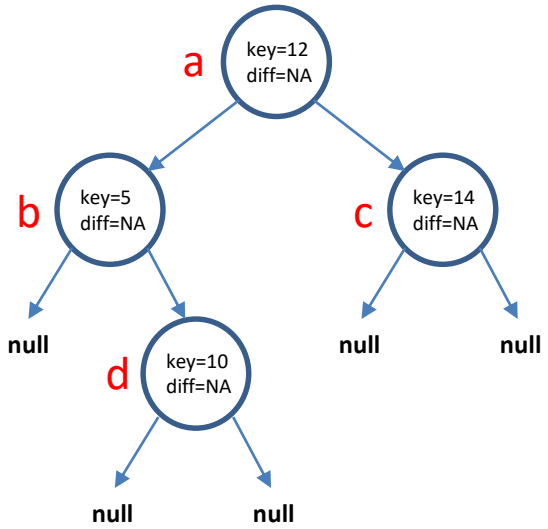


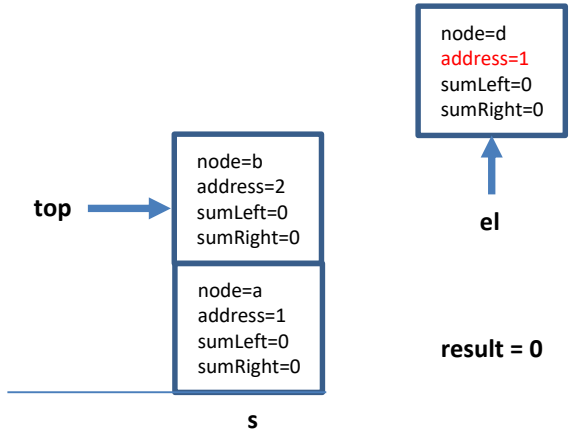


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

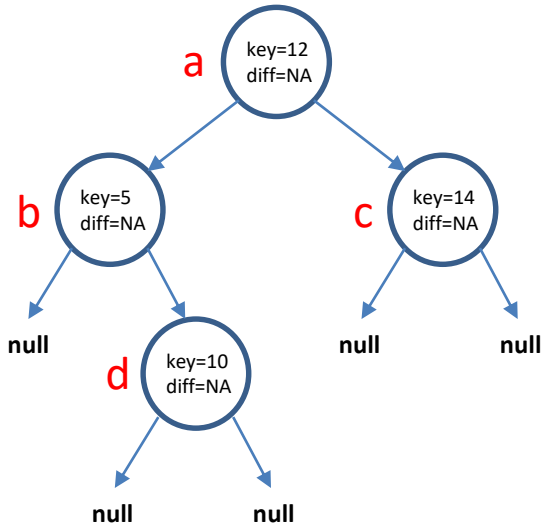


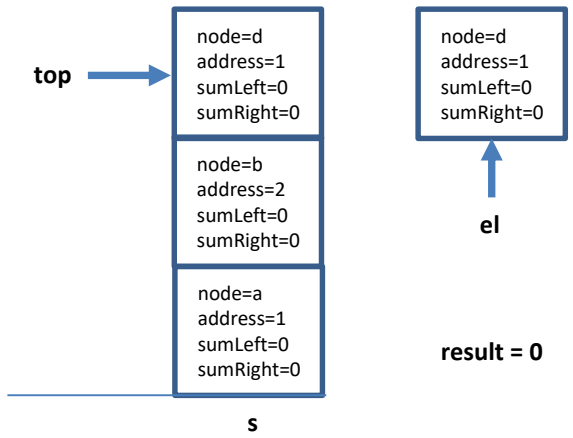


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

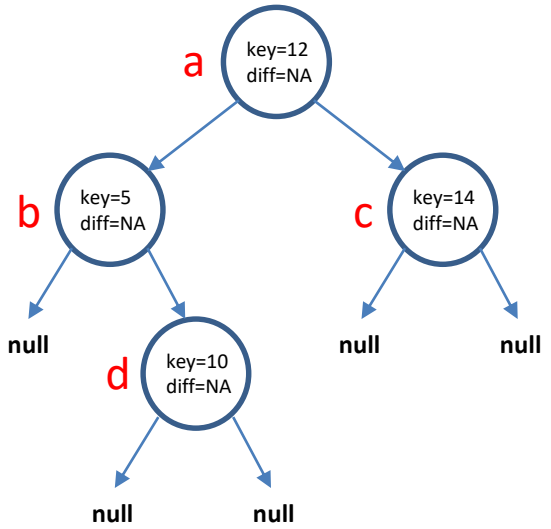


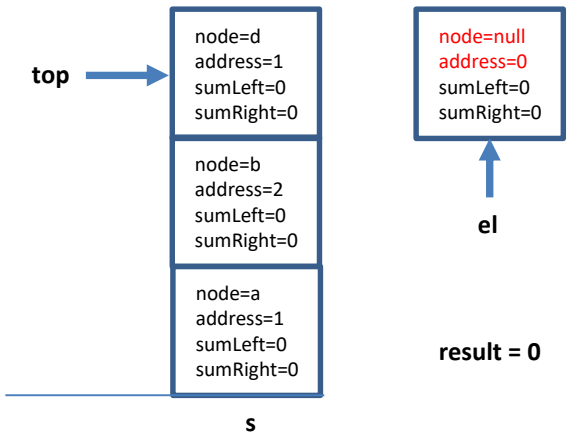


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

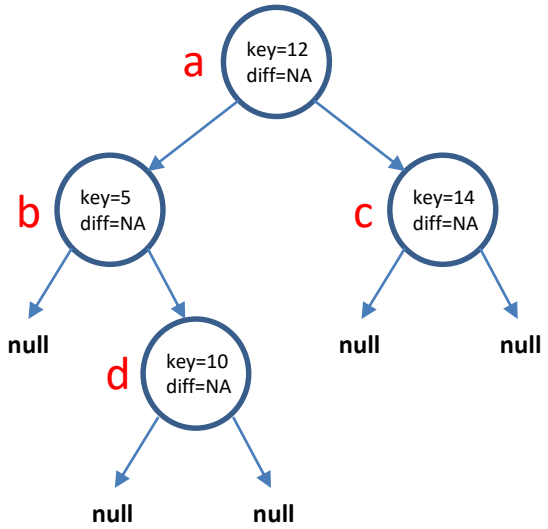


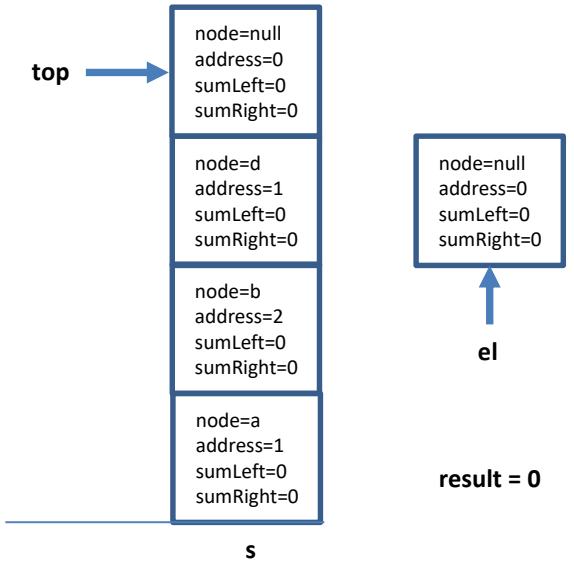


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

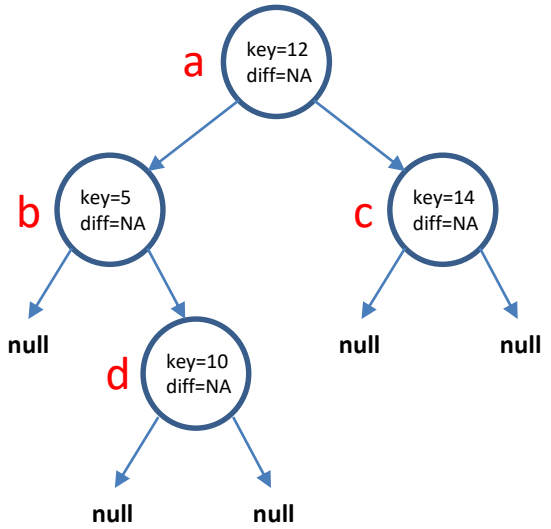


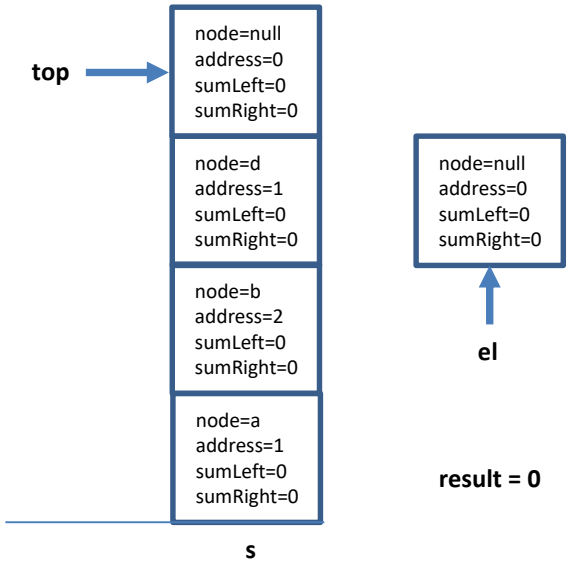


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

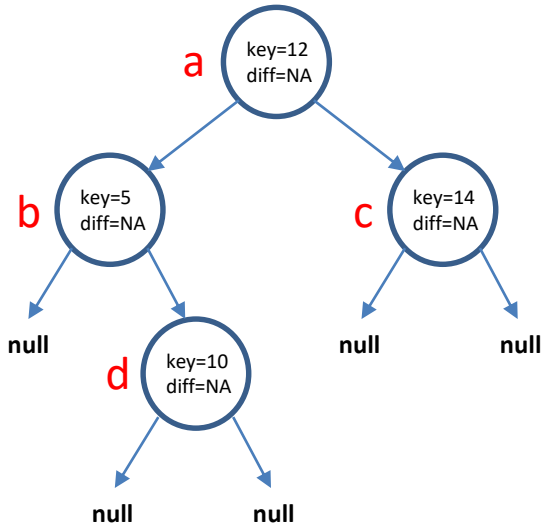


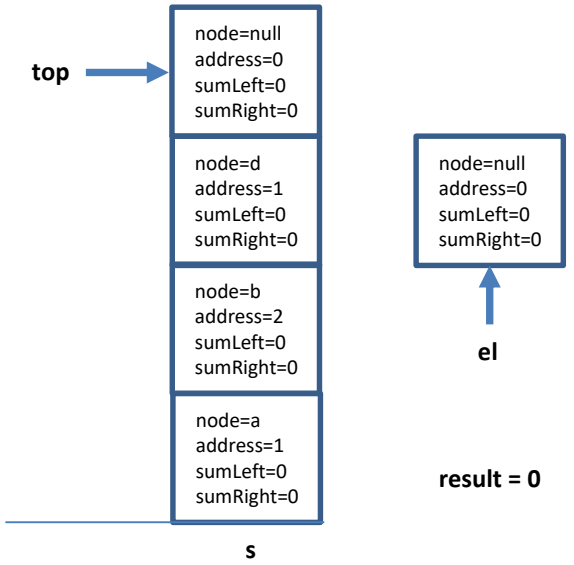


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

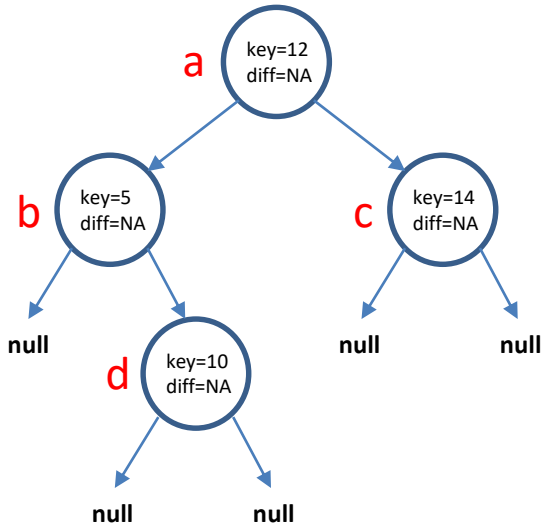


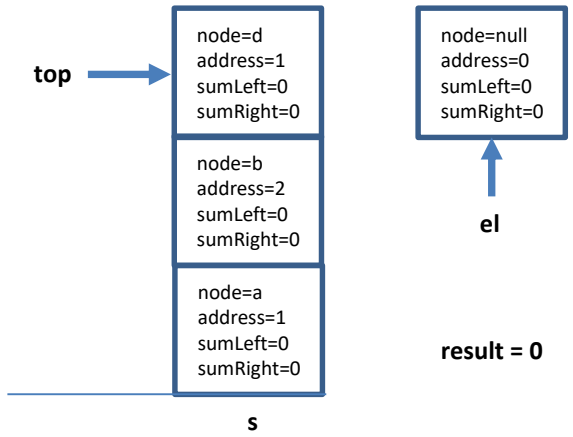


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

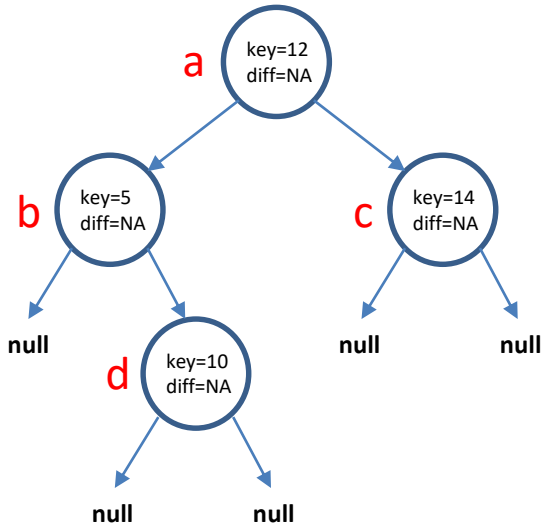


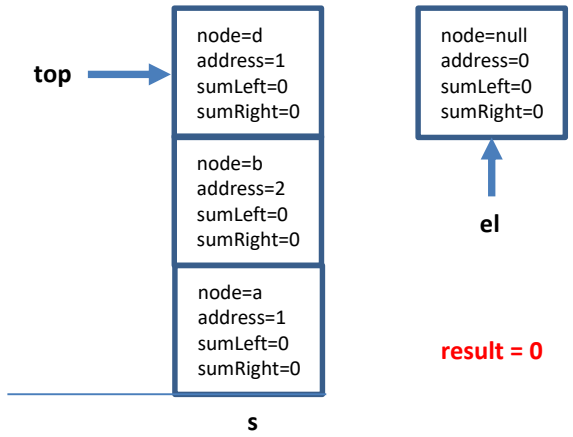


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

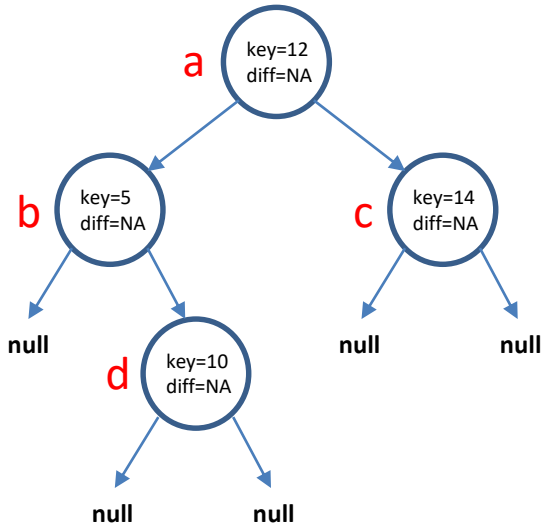


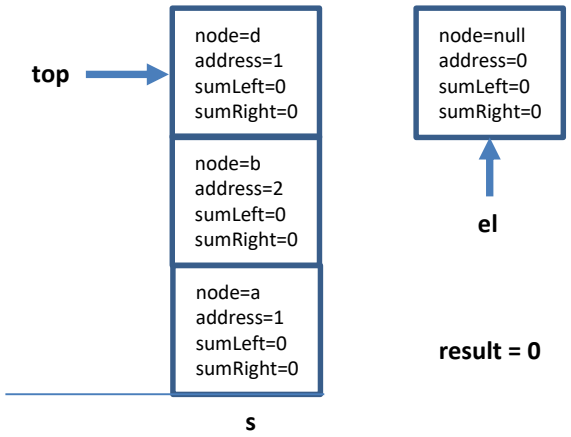


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

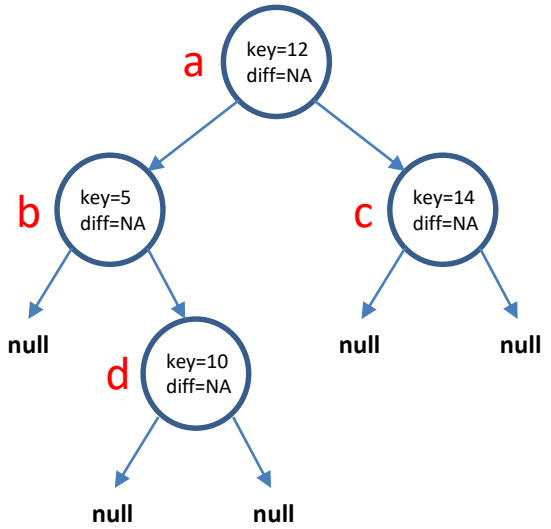


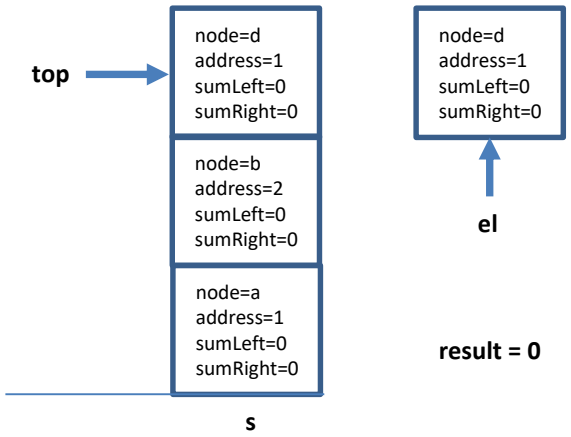


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

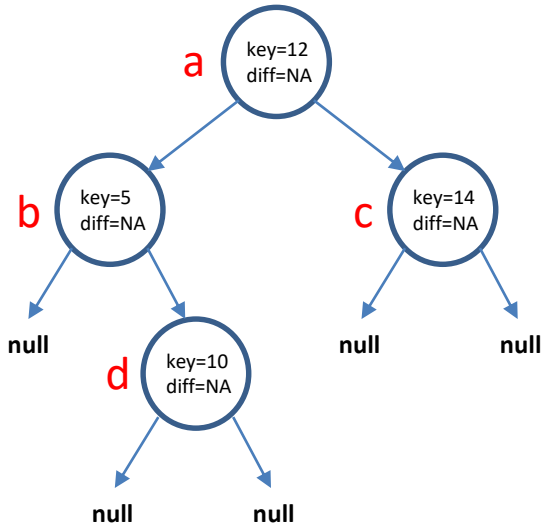


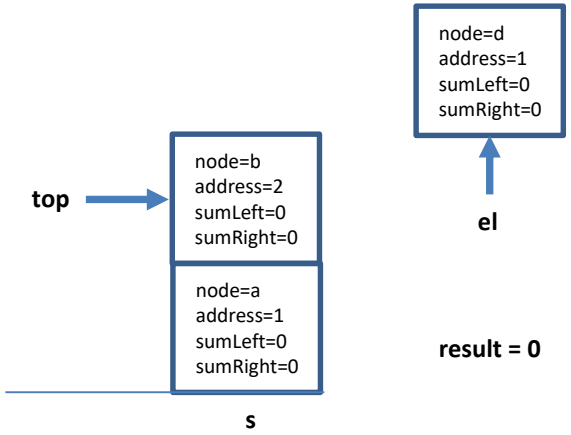


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

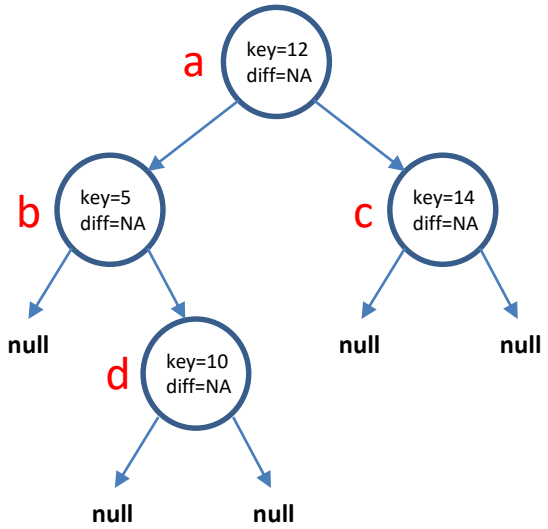


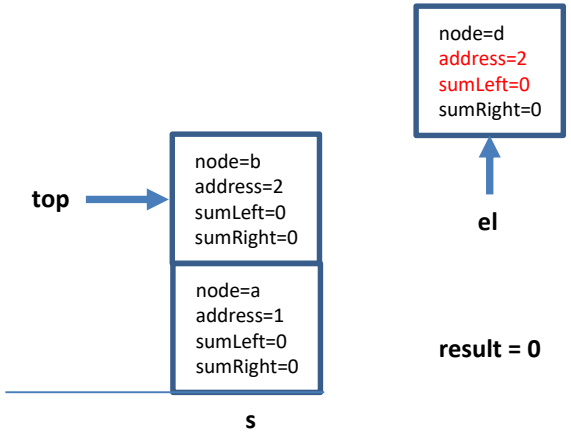


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

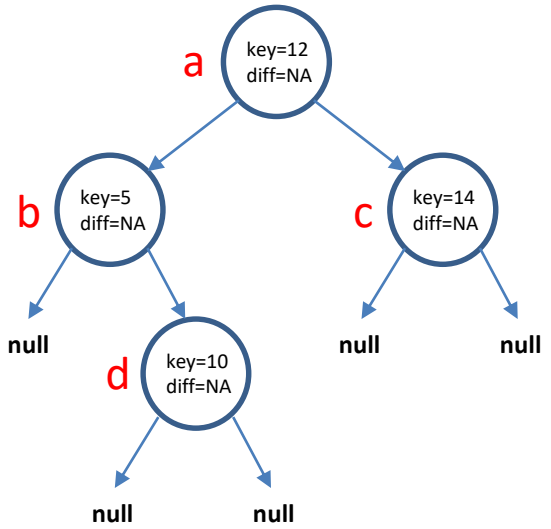


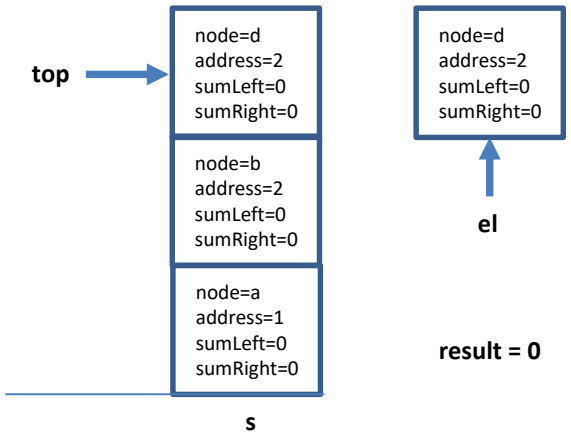


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

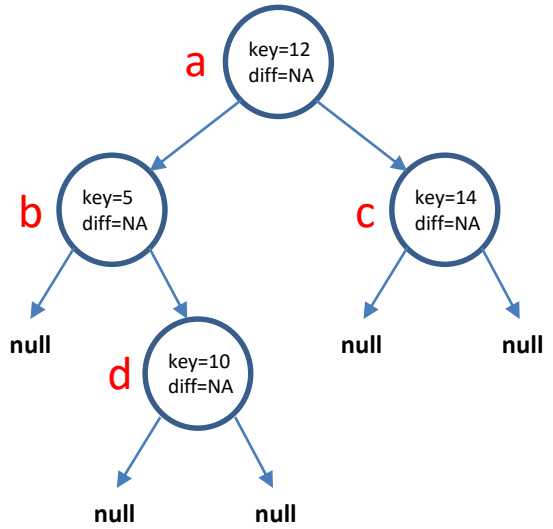


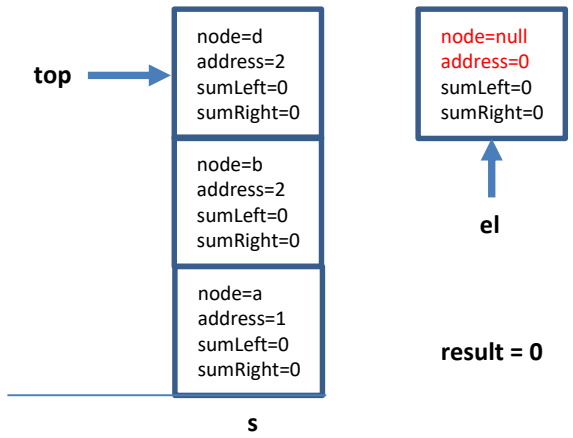


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

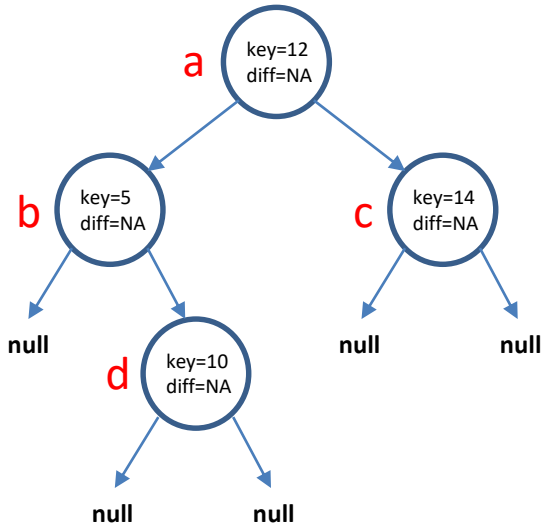


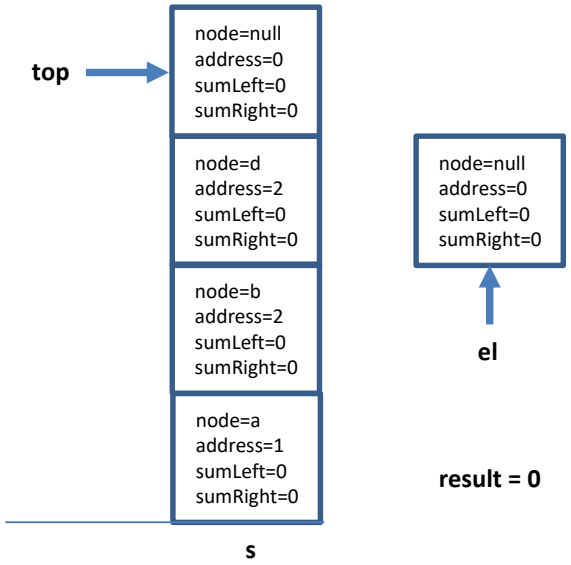


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

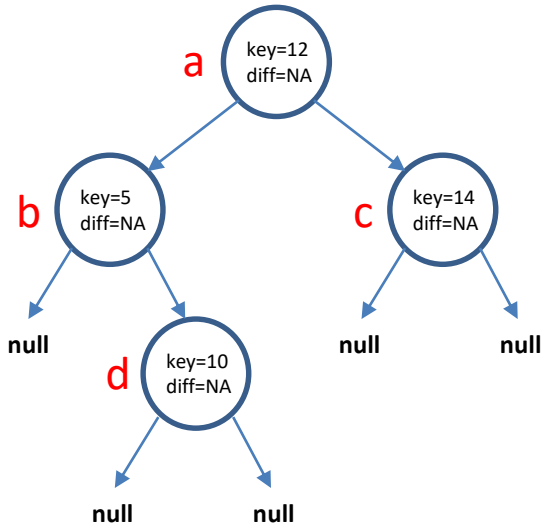


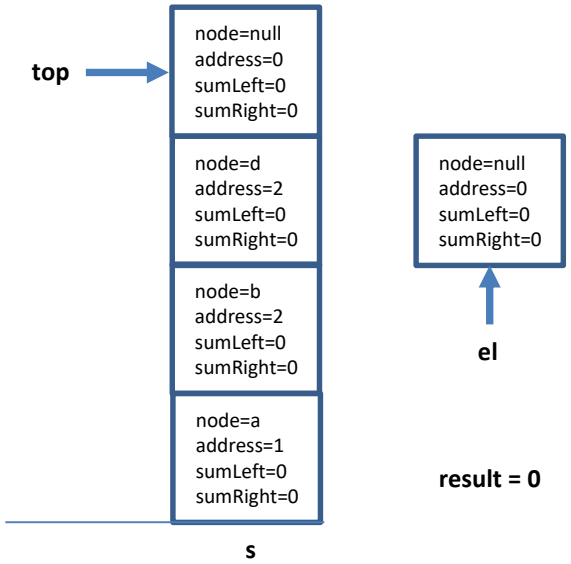


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

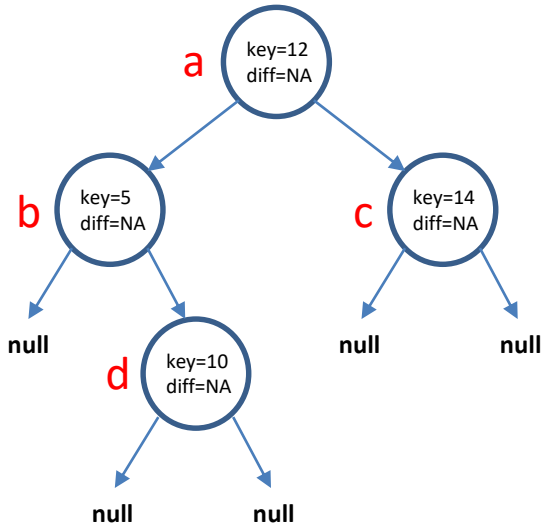


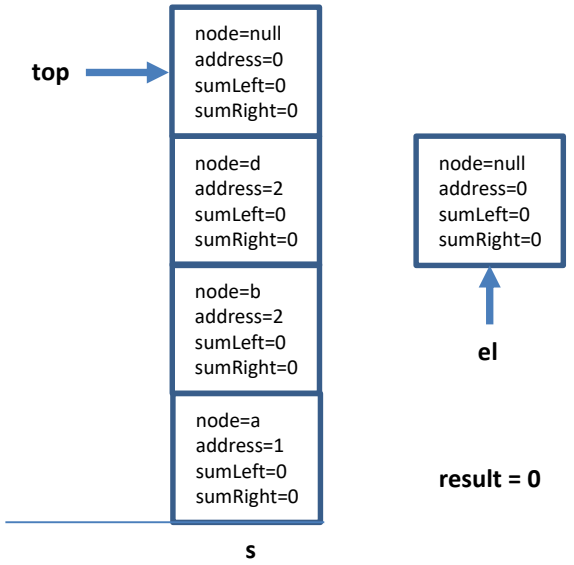


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

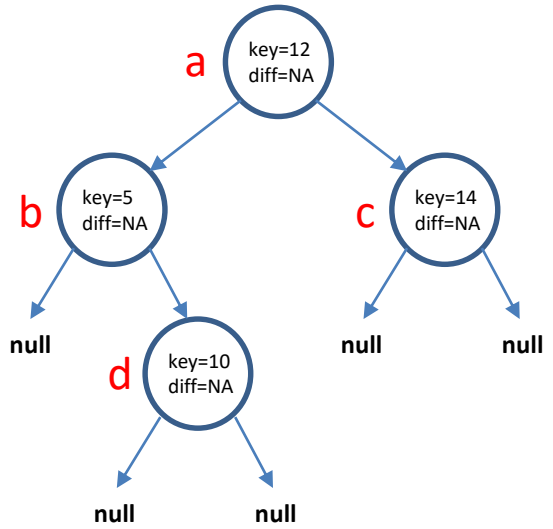


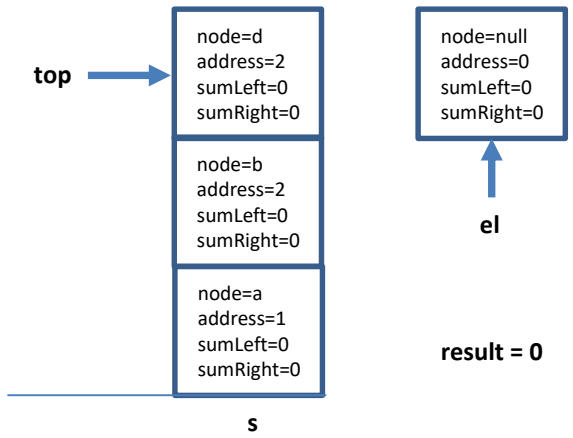


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

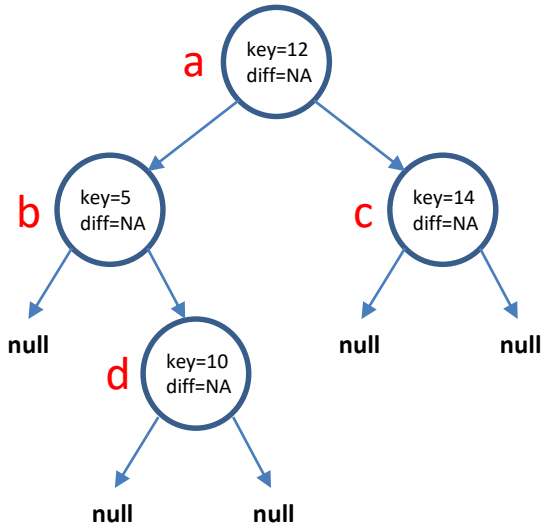


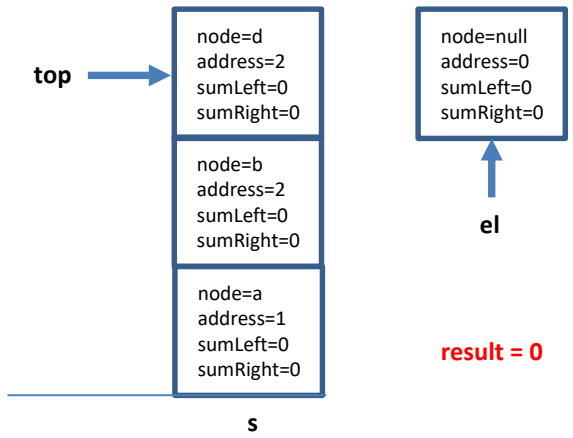


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

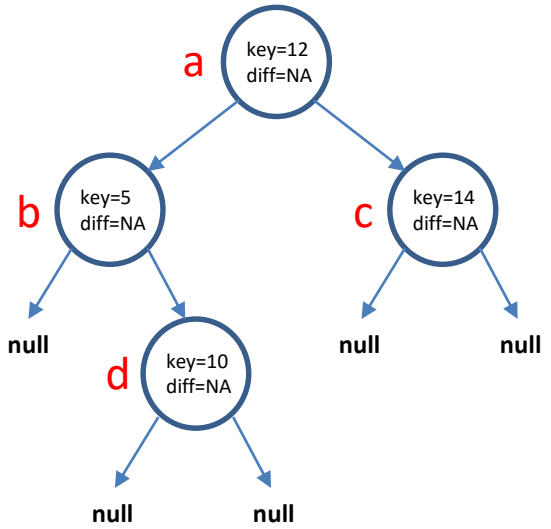


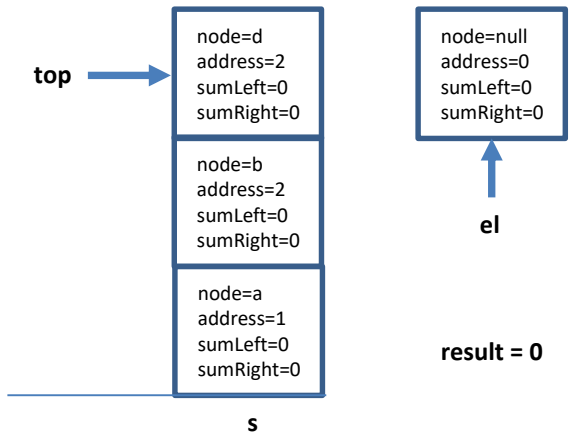


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

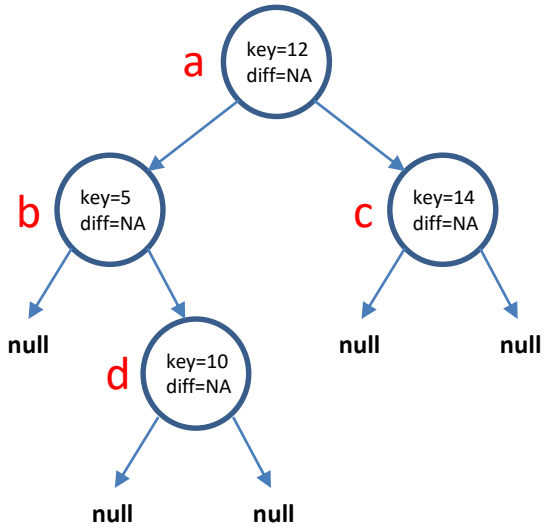


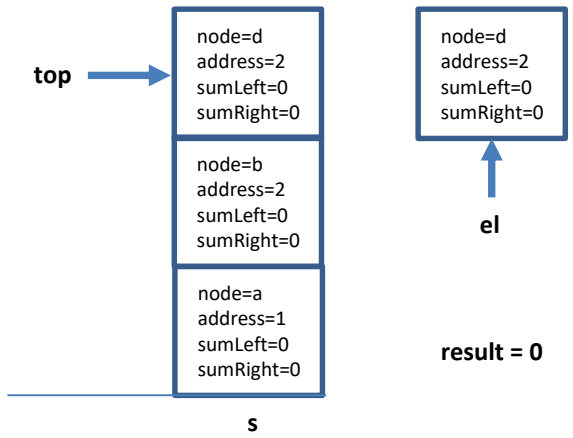


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

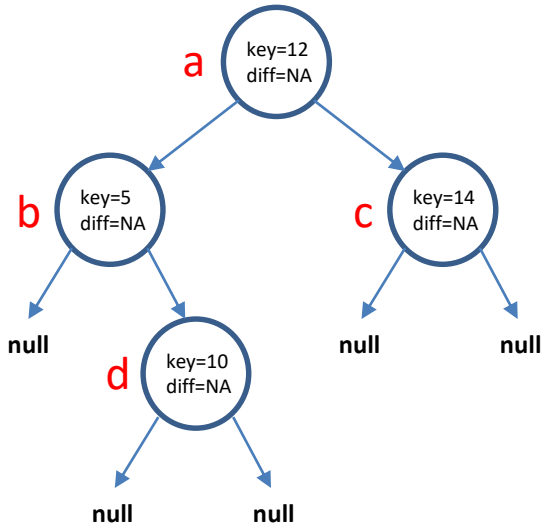


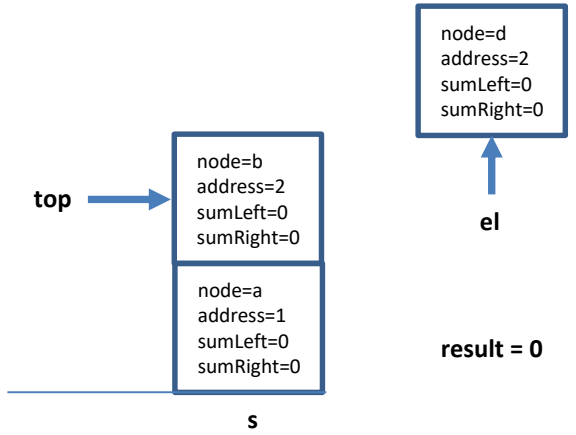


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

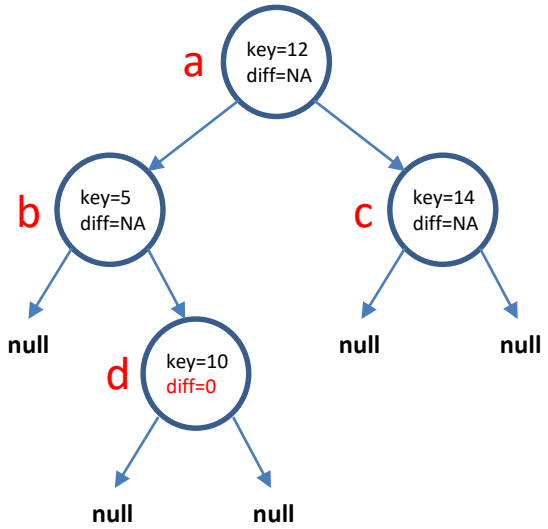


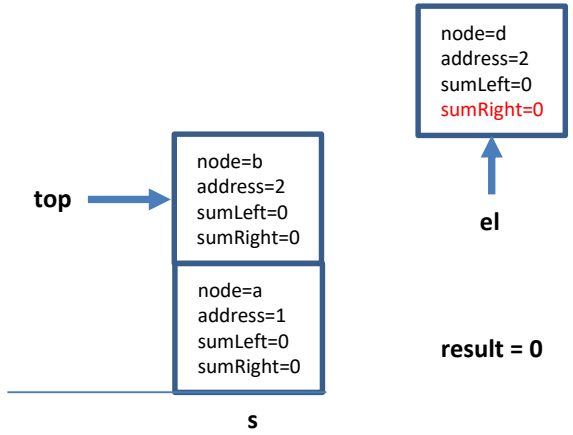


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

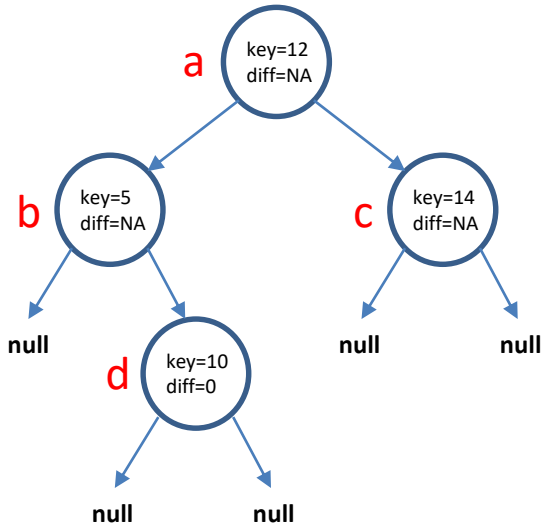


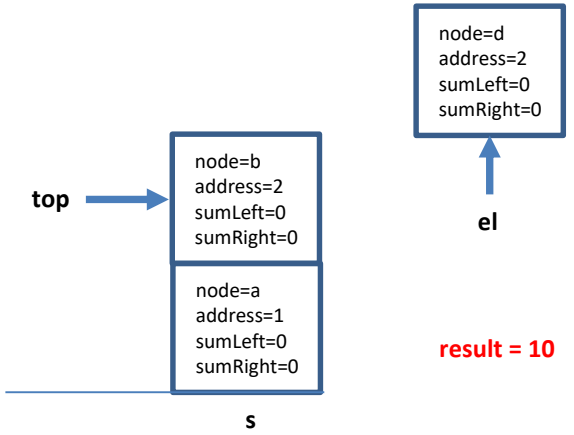


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```



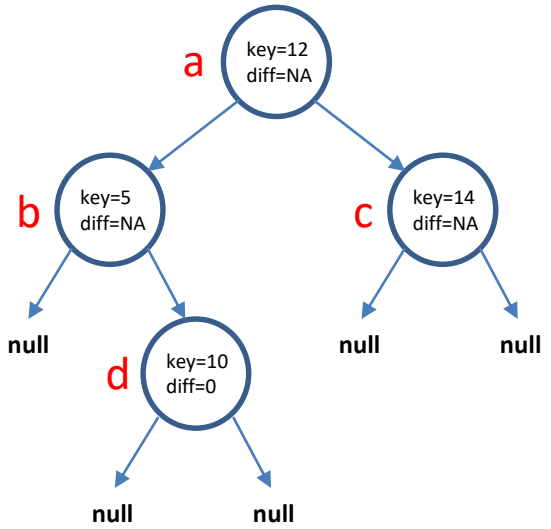


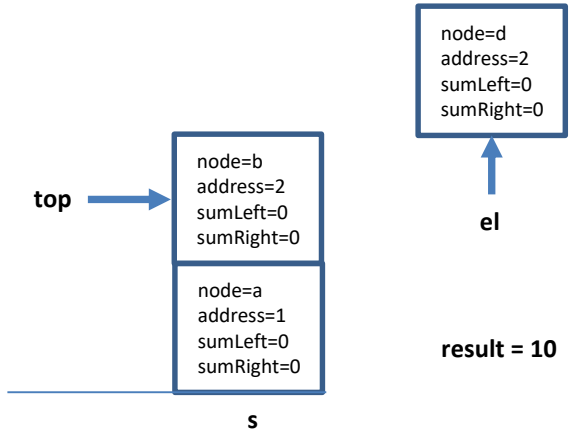
result = 10

REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

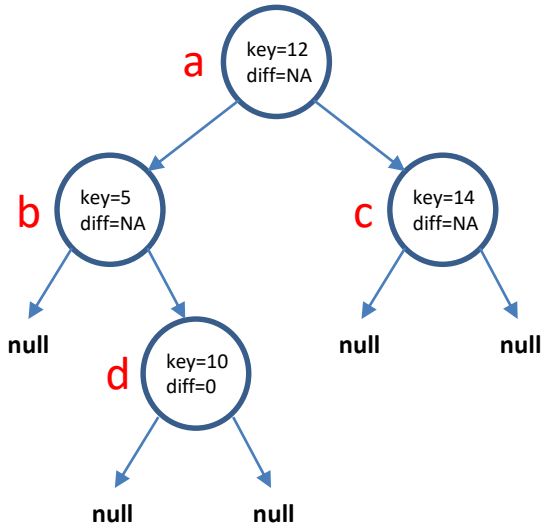


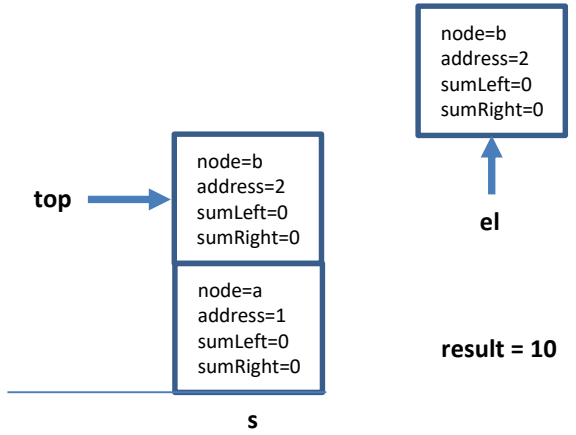


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

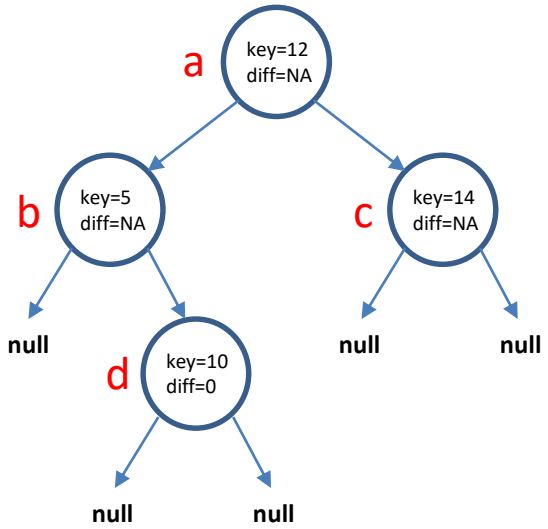


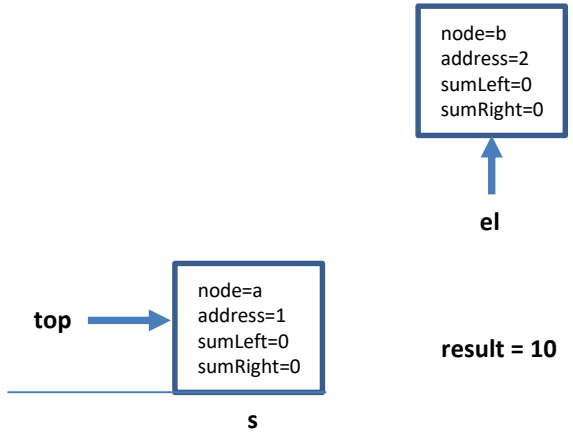


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

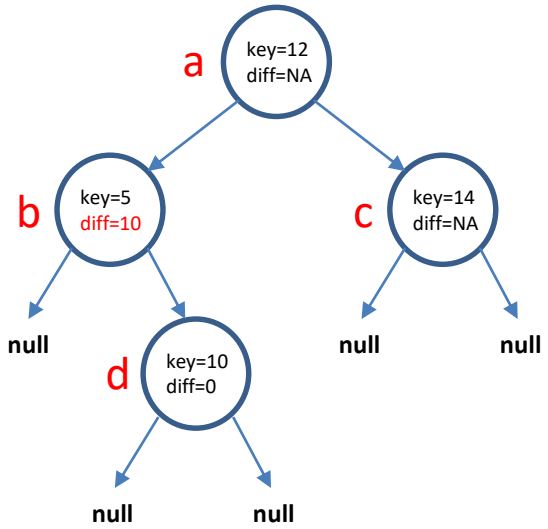


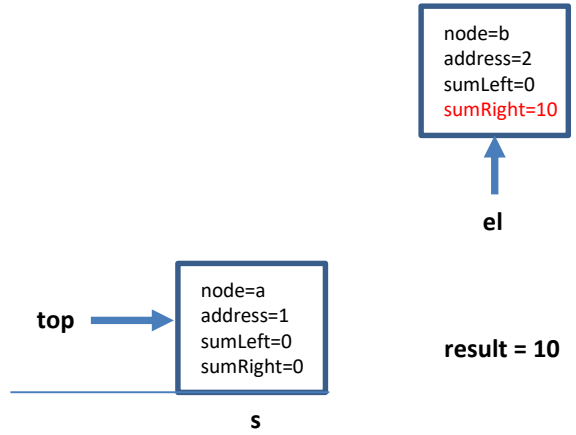


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

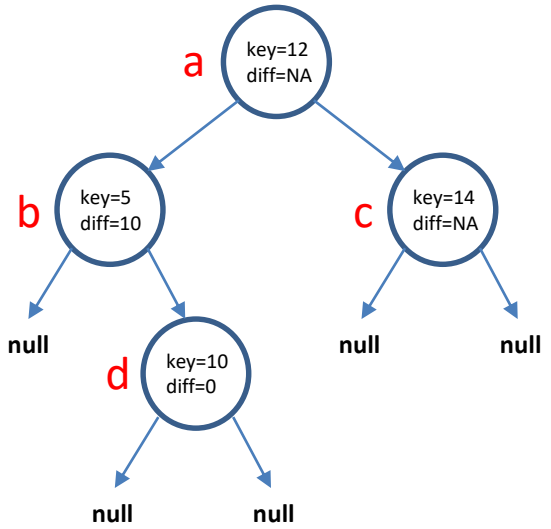


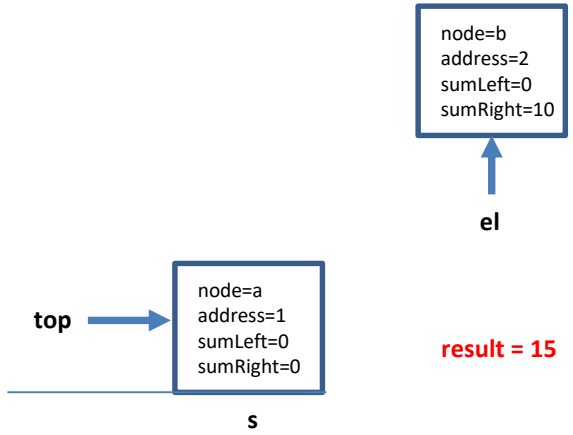


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```



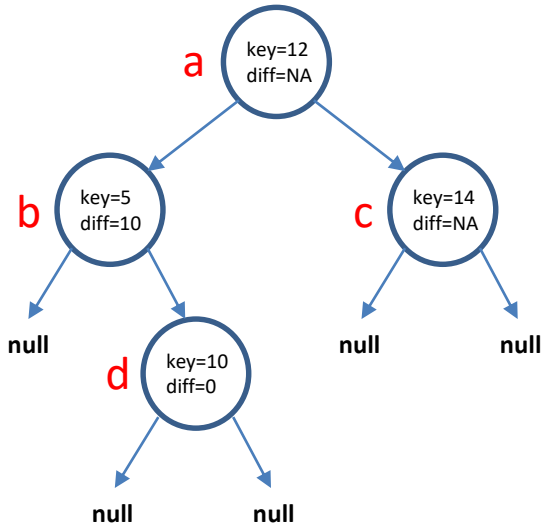


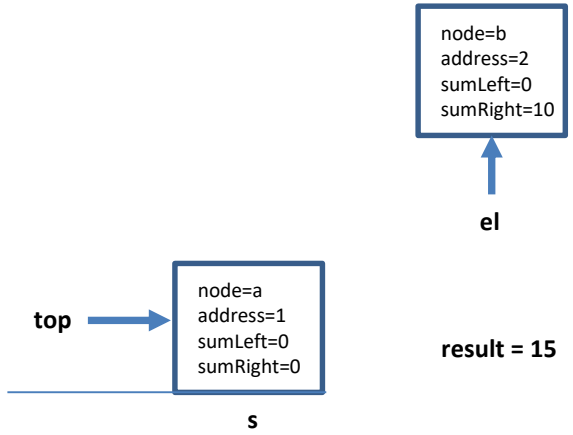
result = 15

REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

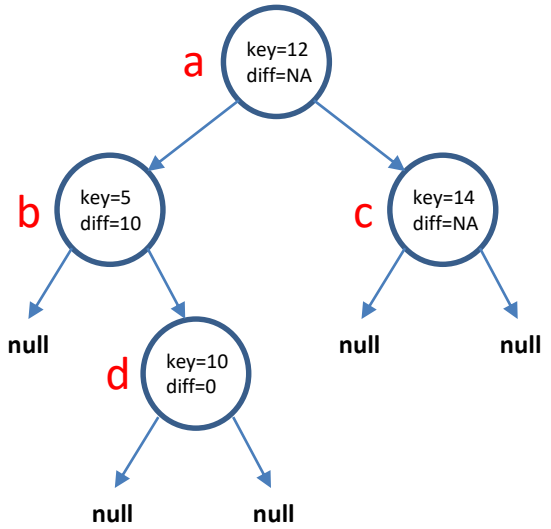


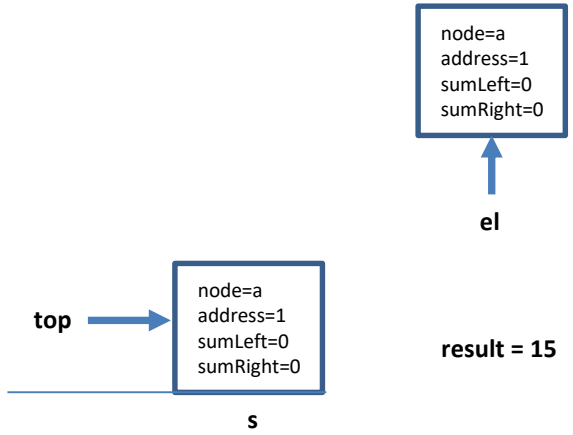


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

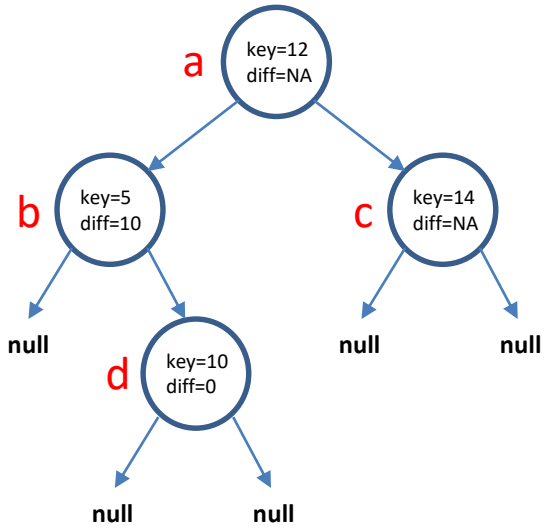


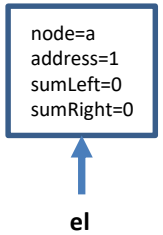


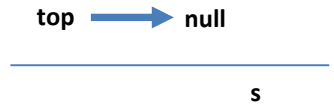
REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```





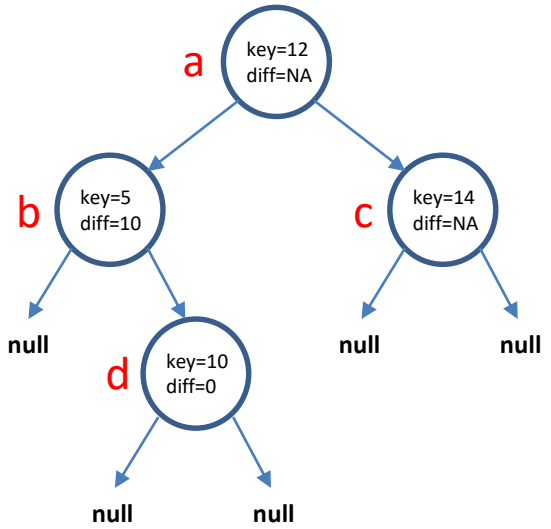


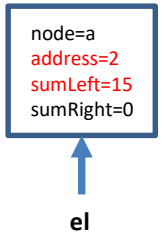
result = 15

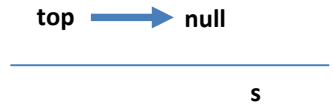
REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```





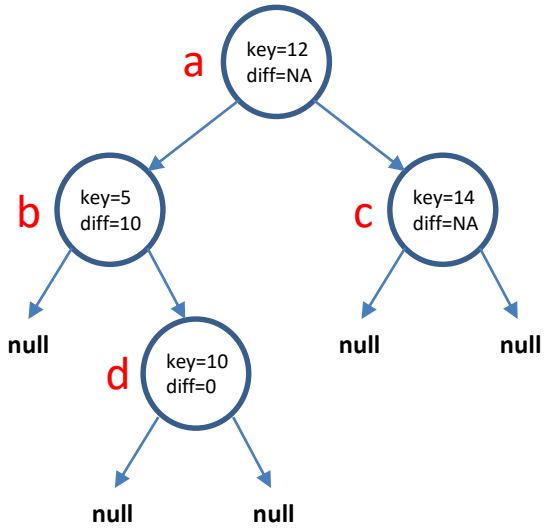


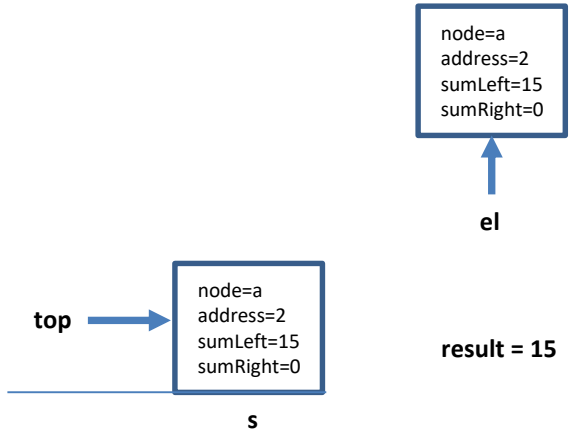
result = 15

REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

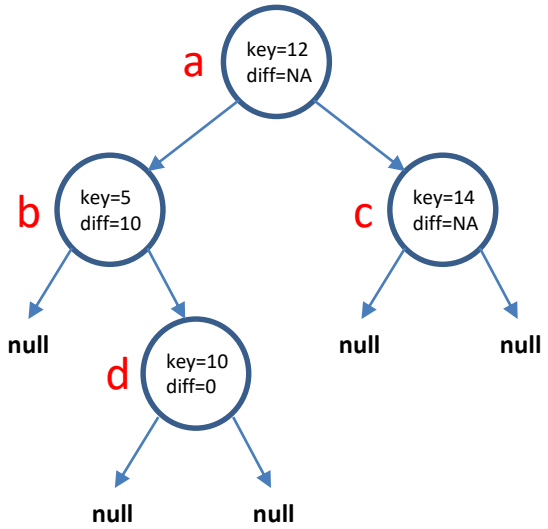


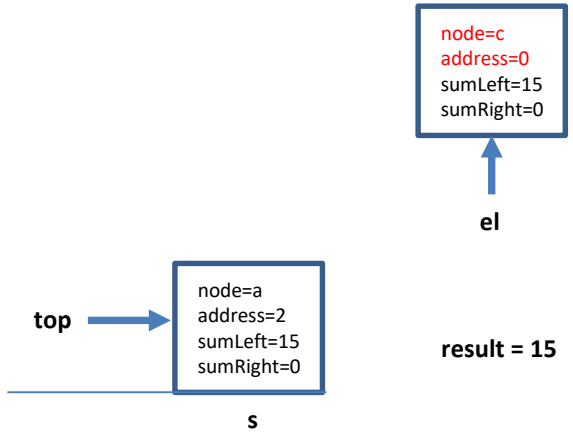


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

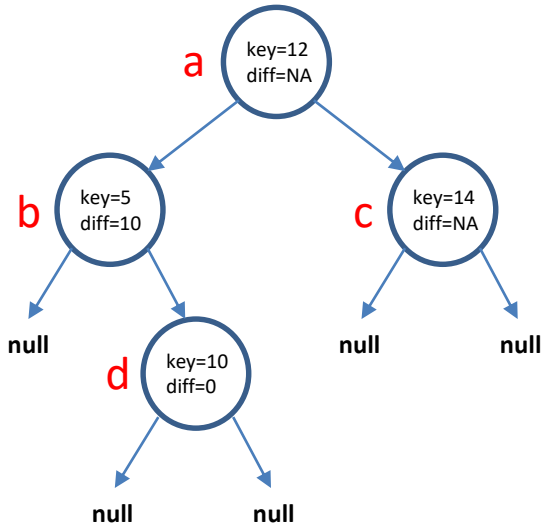


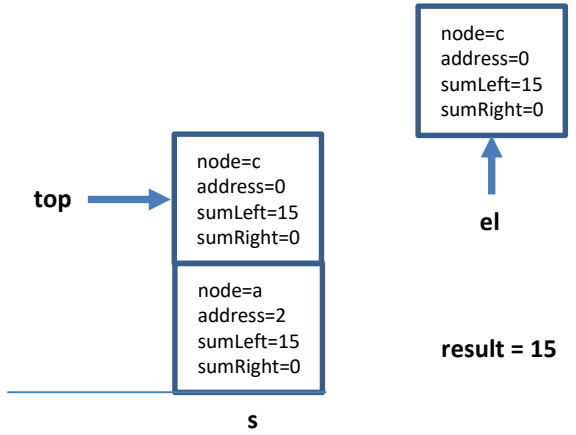


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

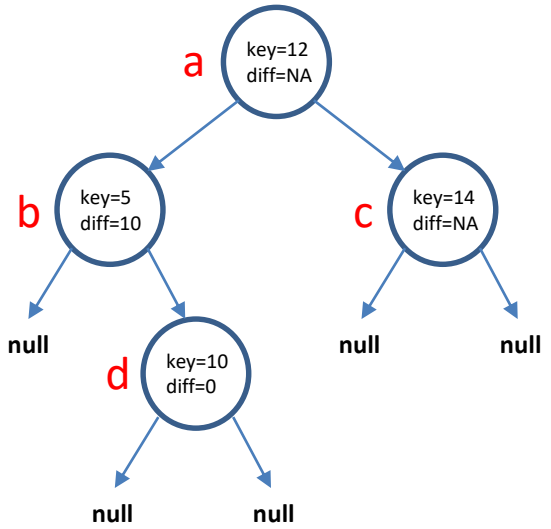


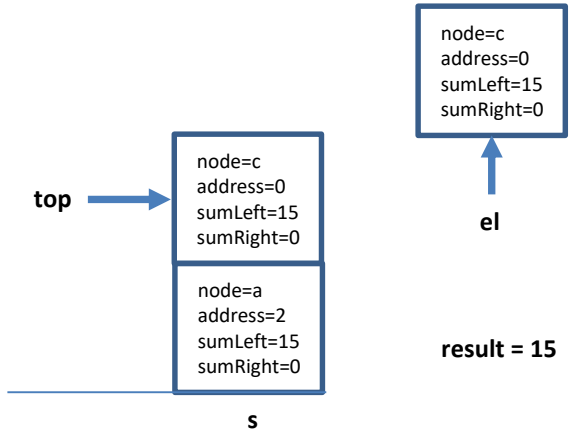


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

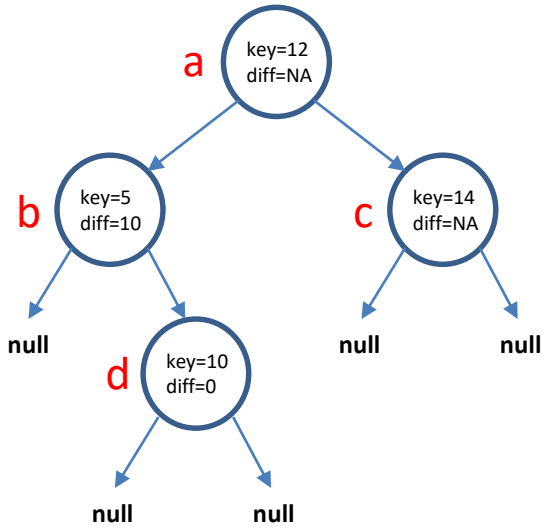


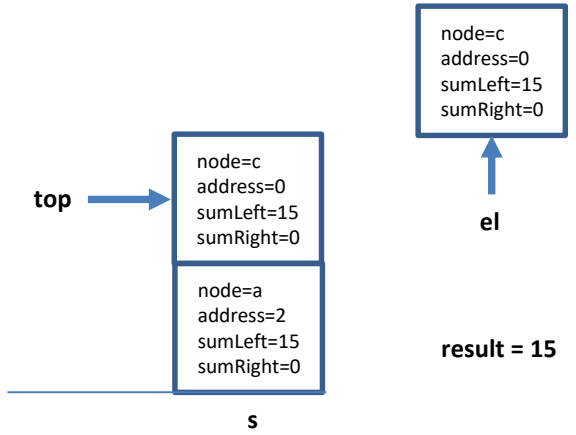


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

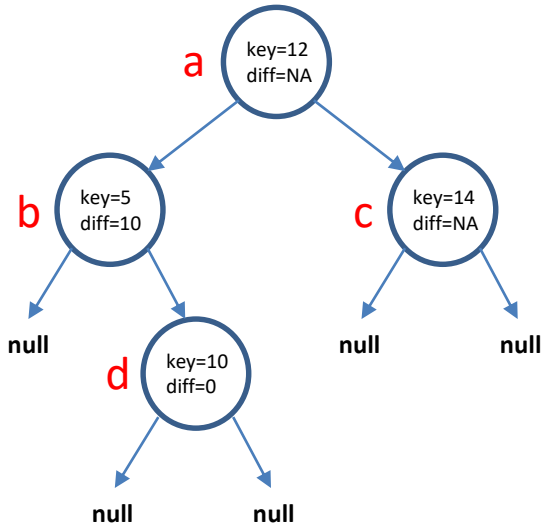


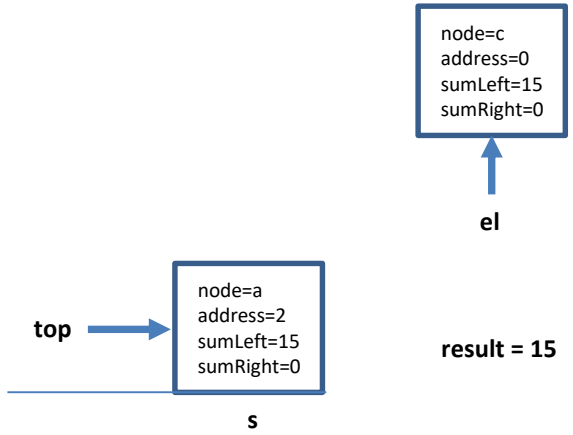


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

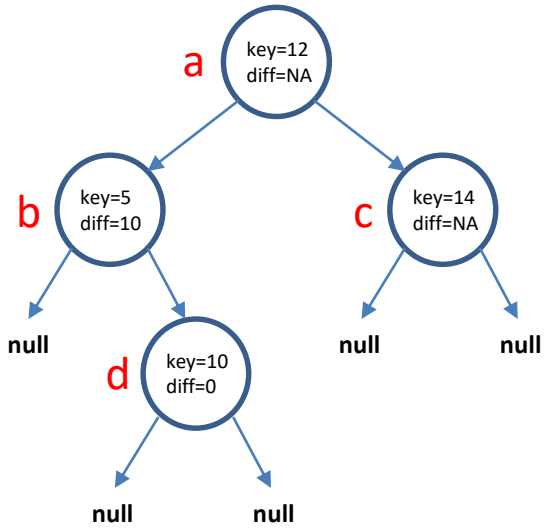


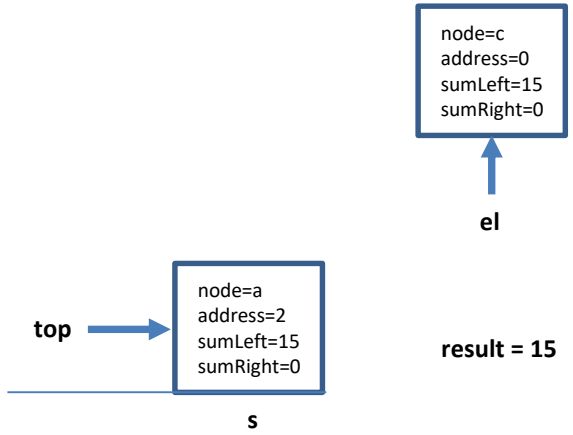


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

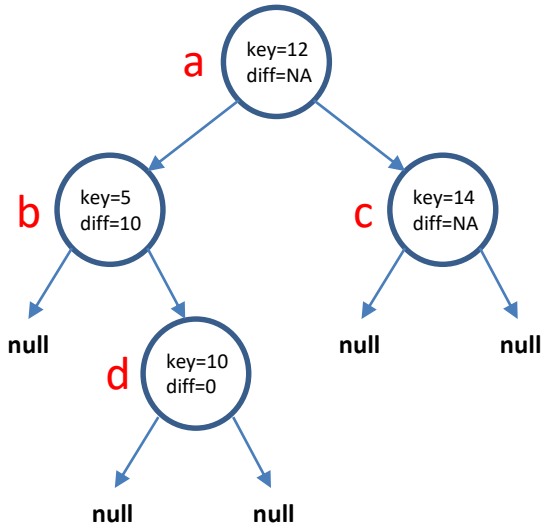


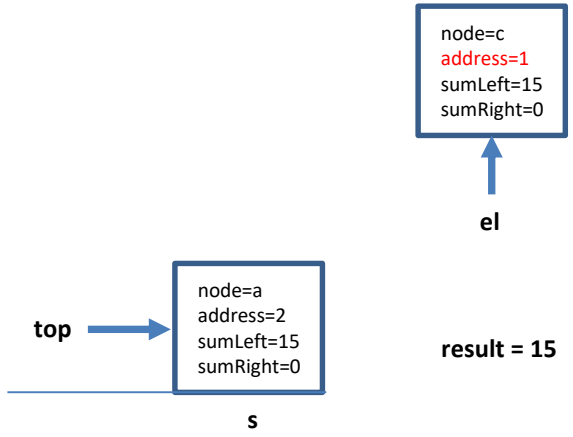


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

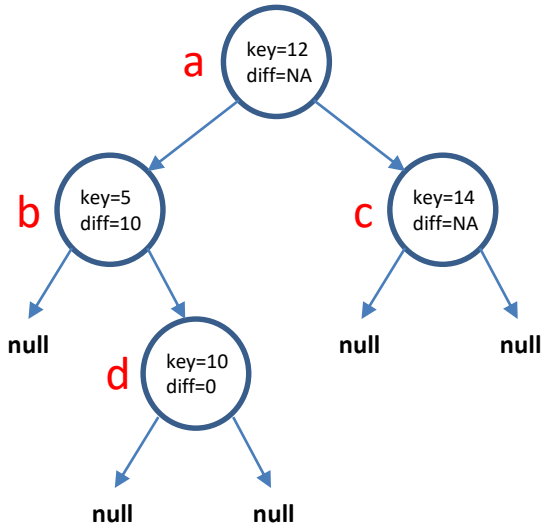


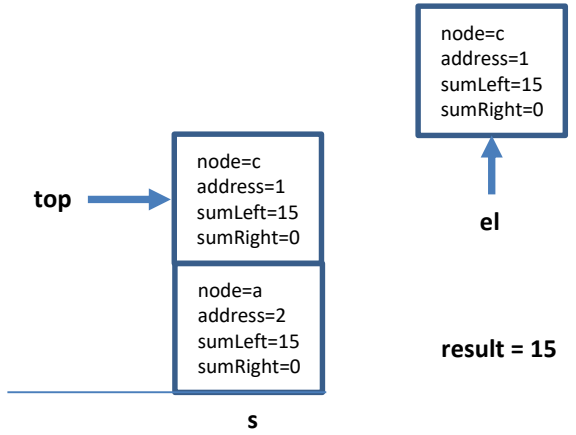


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

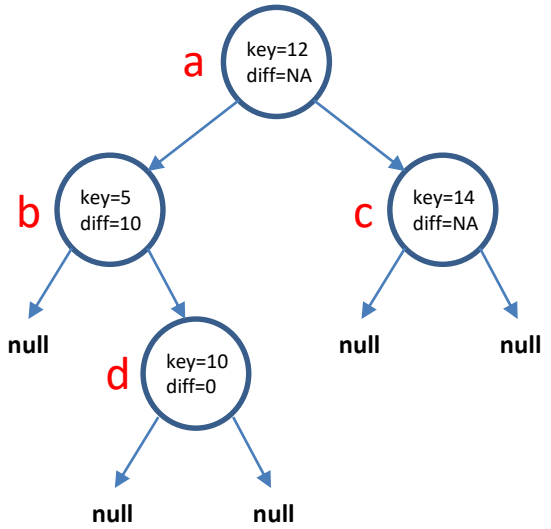


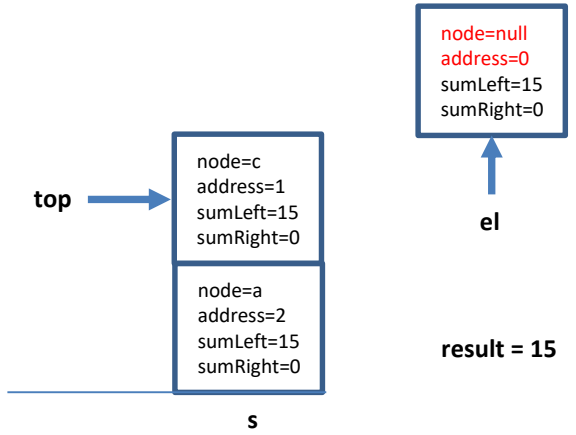


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

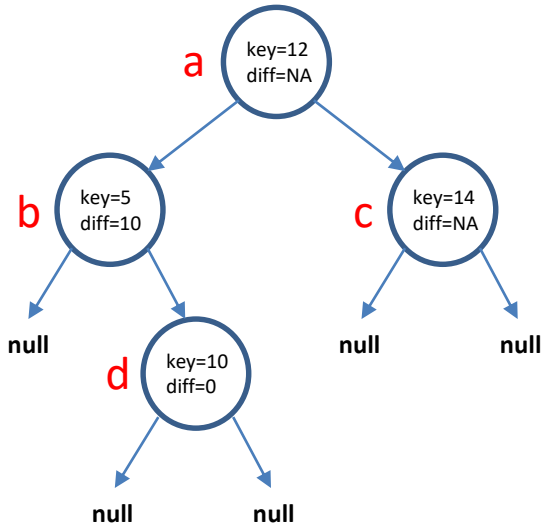


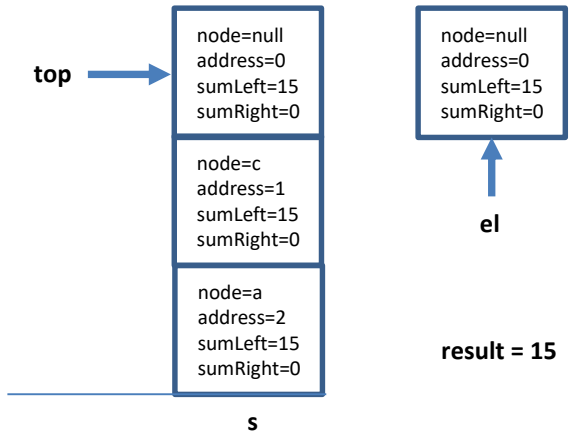


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

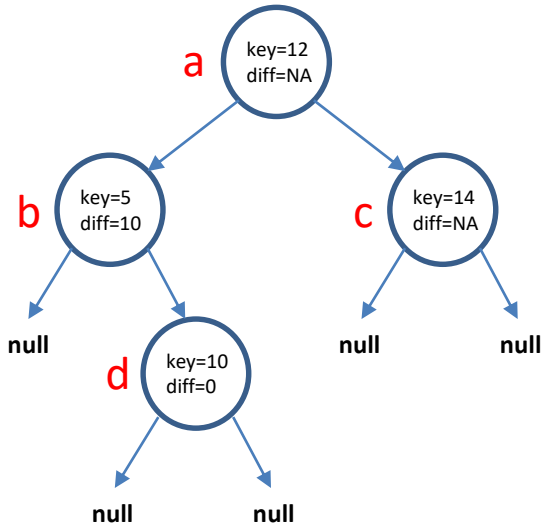


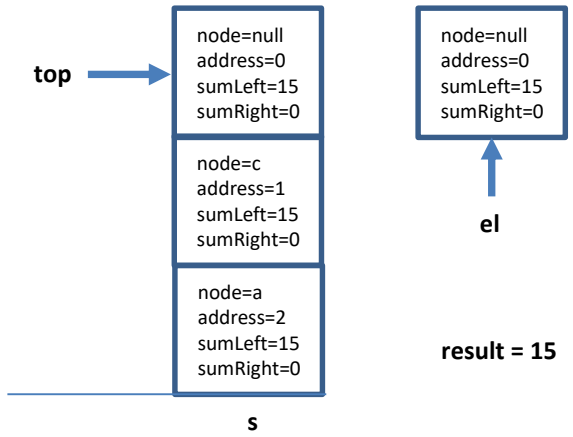


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

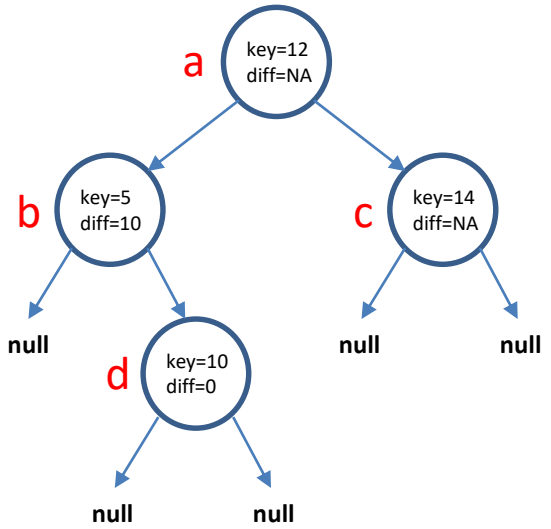


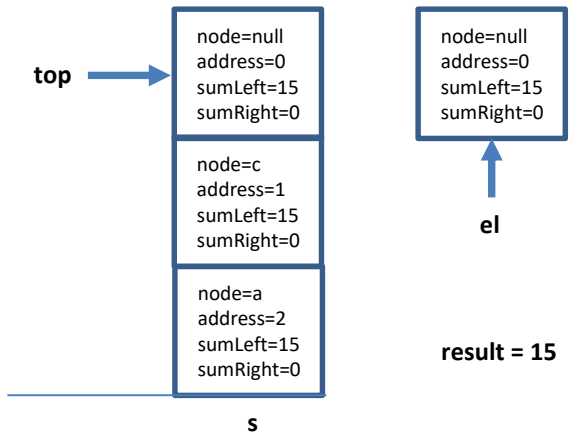


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

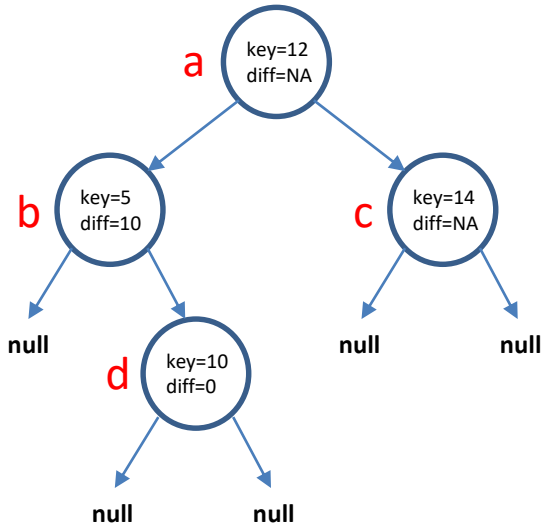


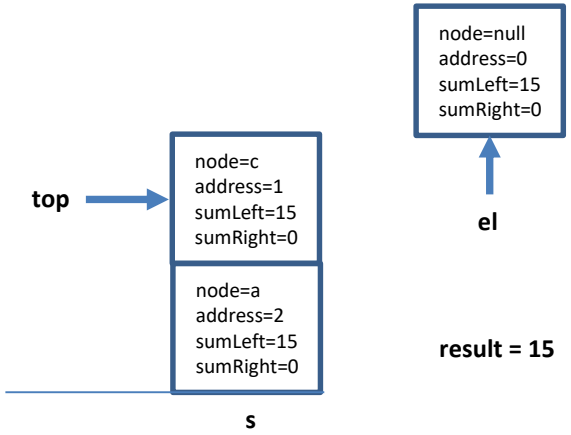


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

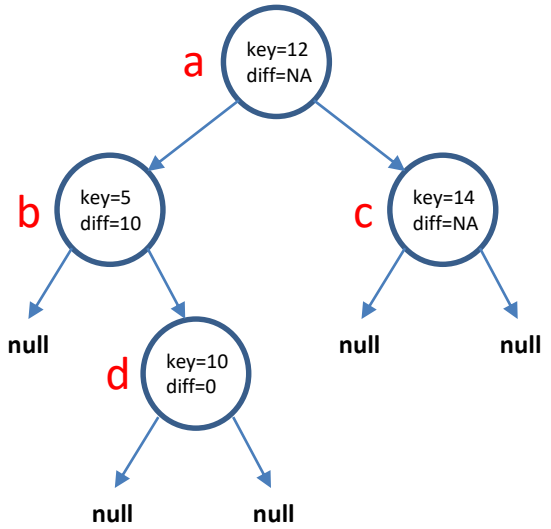


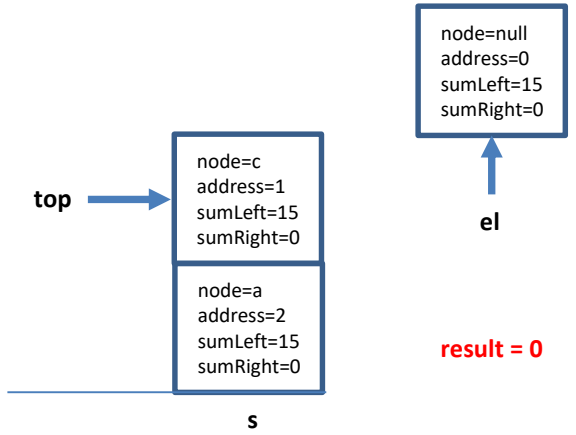


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

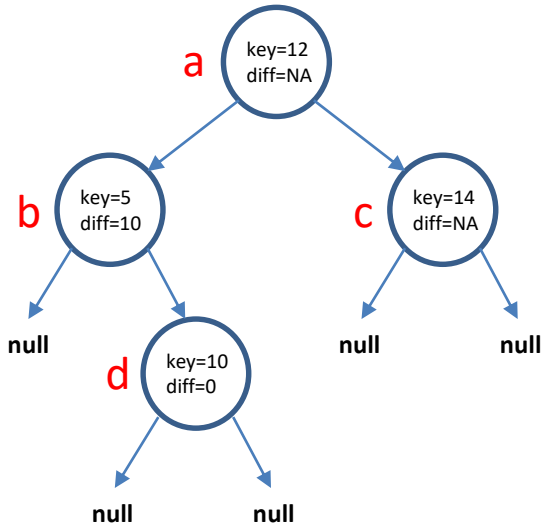


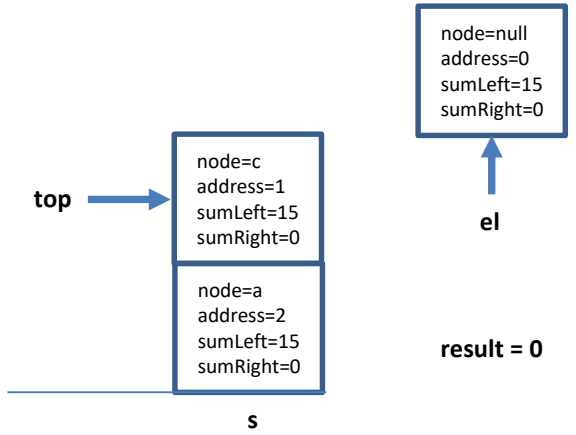


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

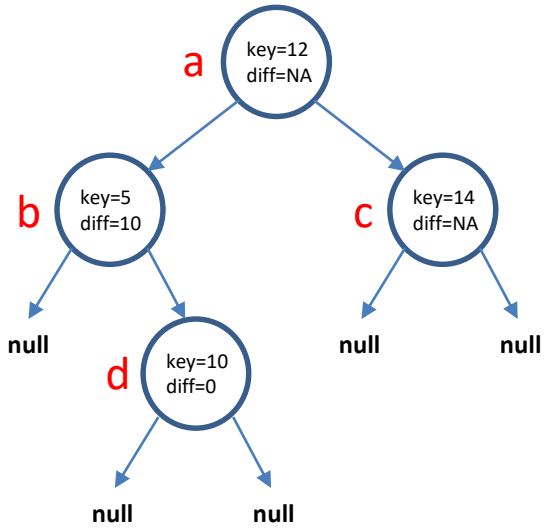


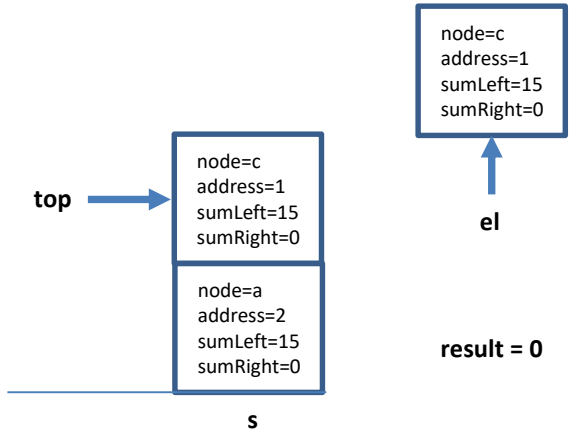


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

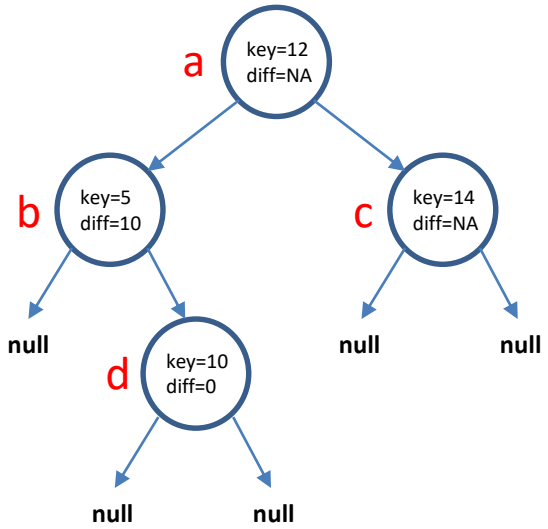


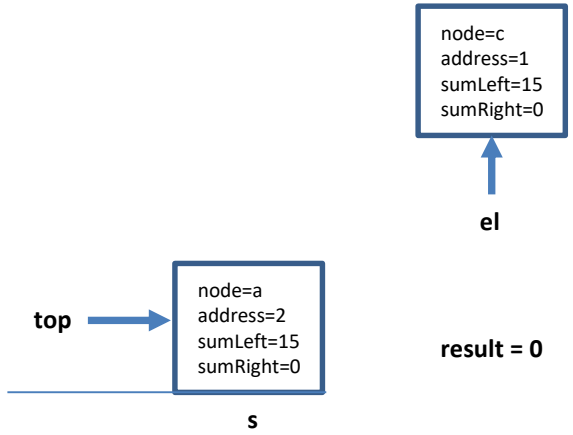


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

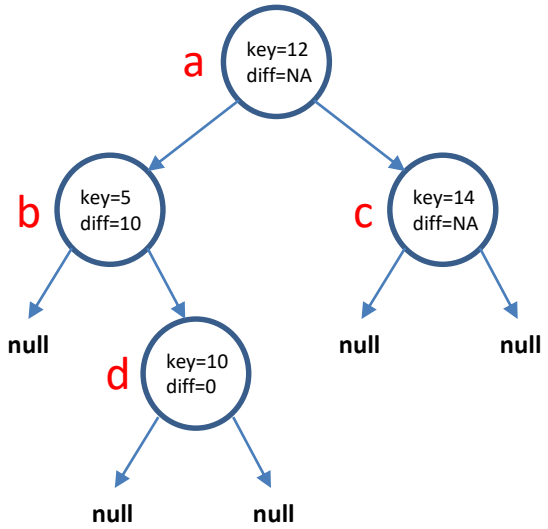


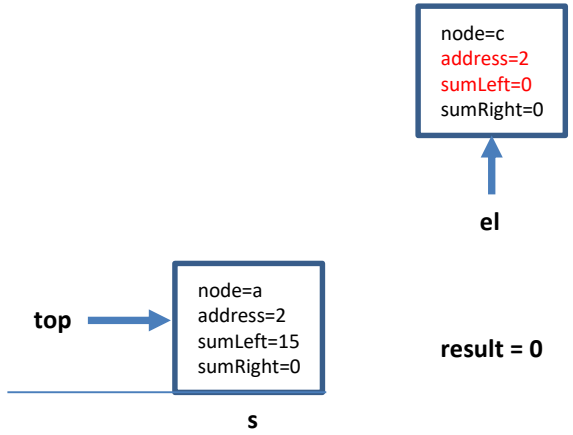


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

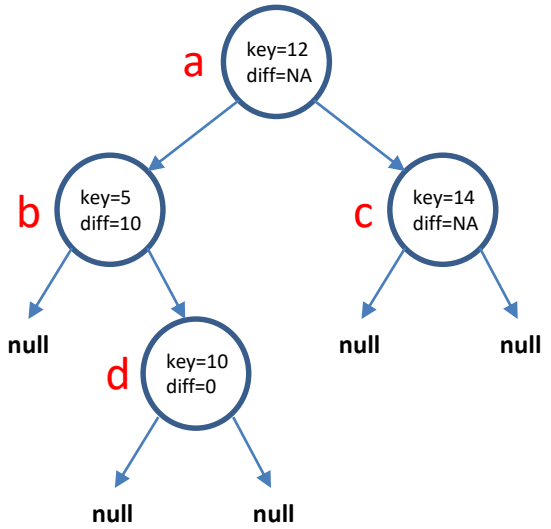


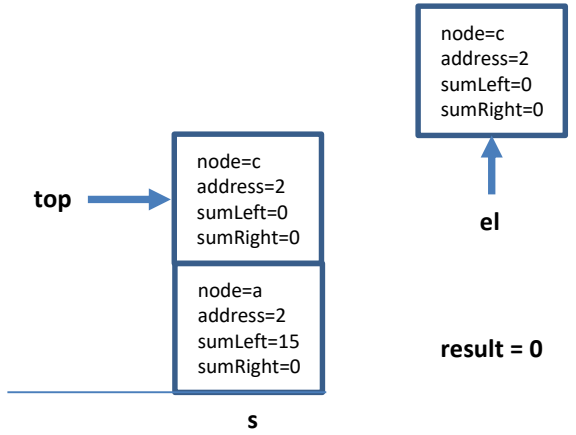


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

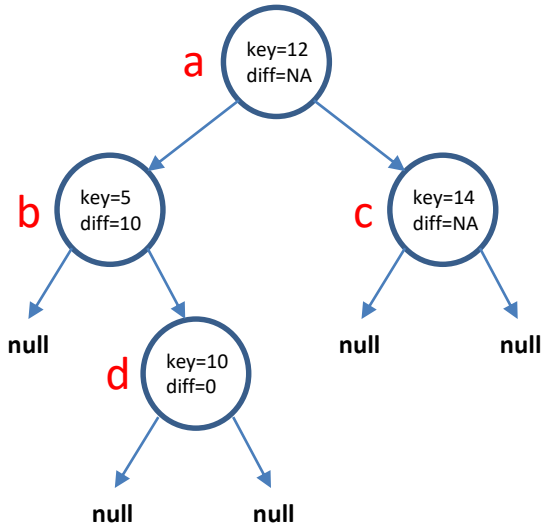


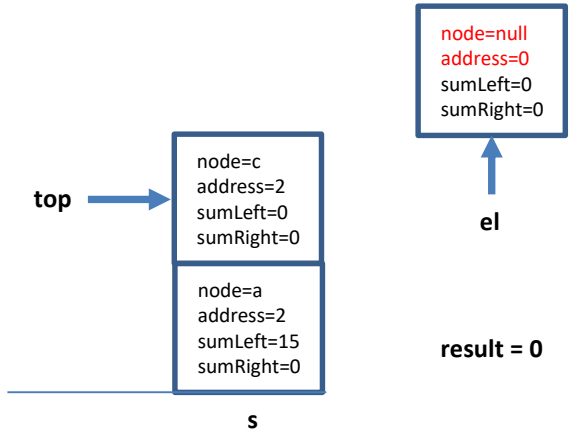


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

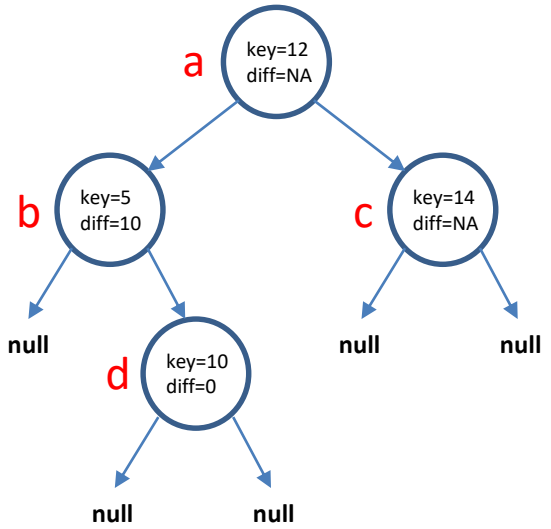


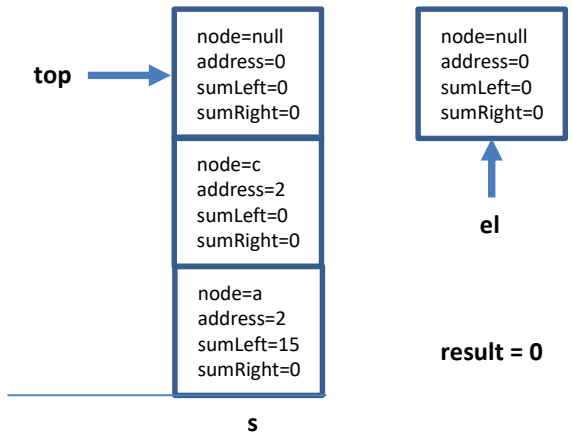


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

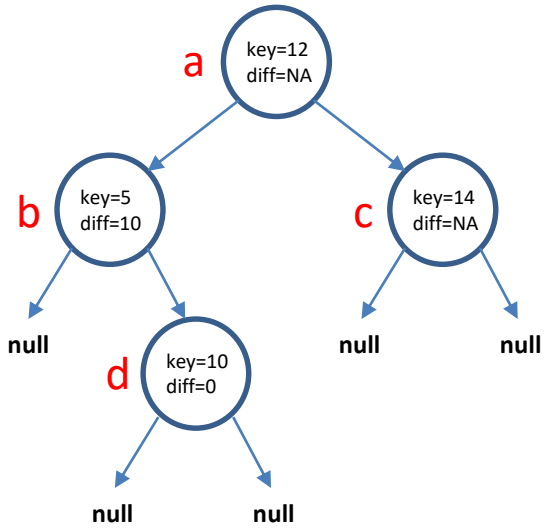


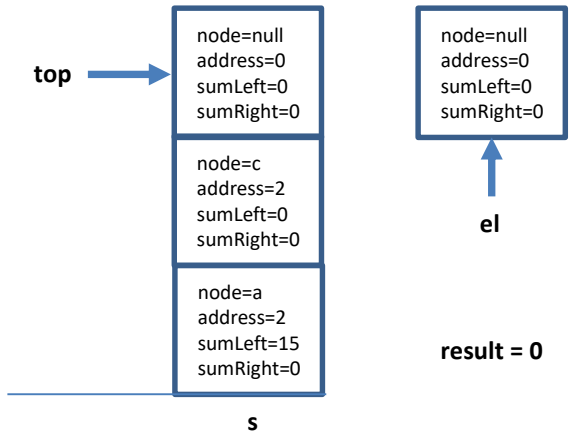


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

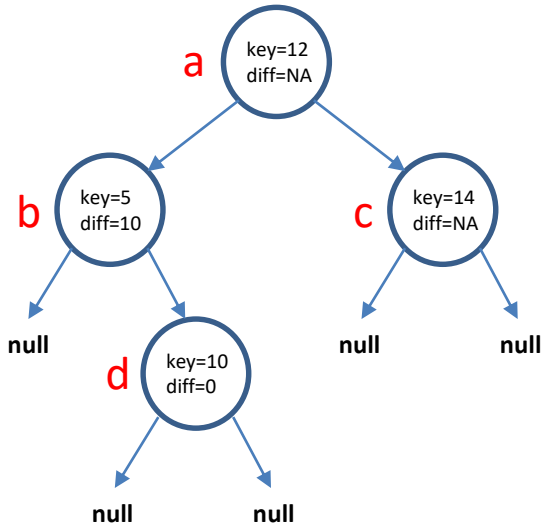


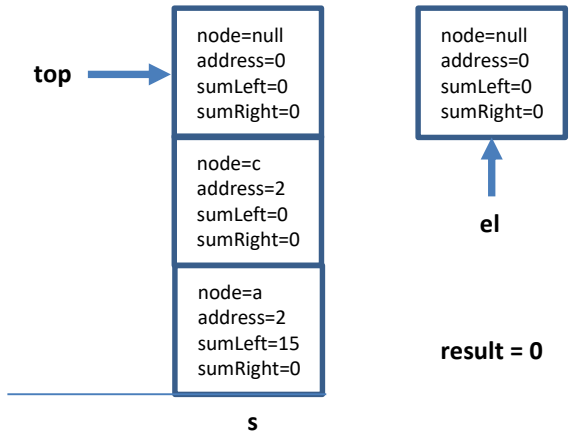


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

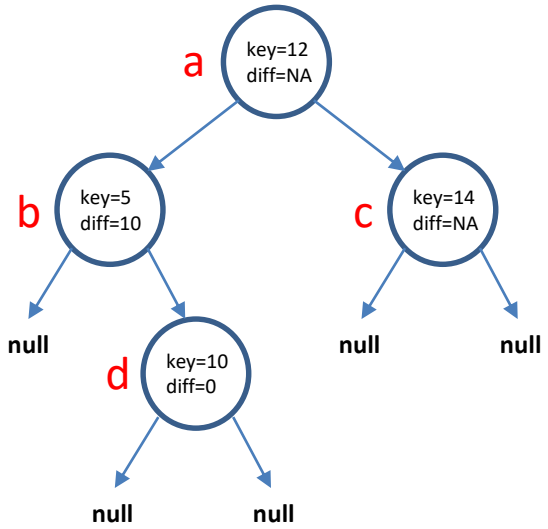


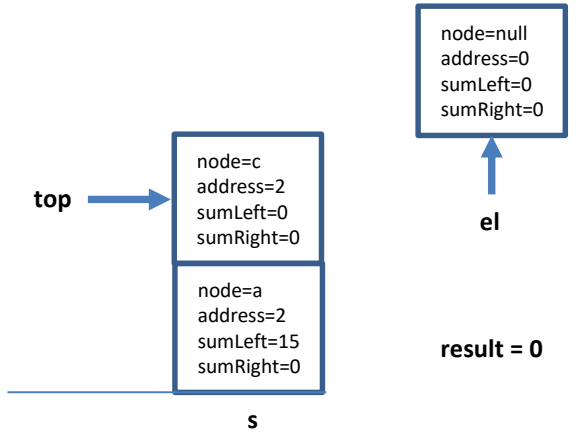


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

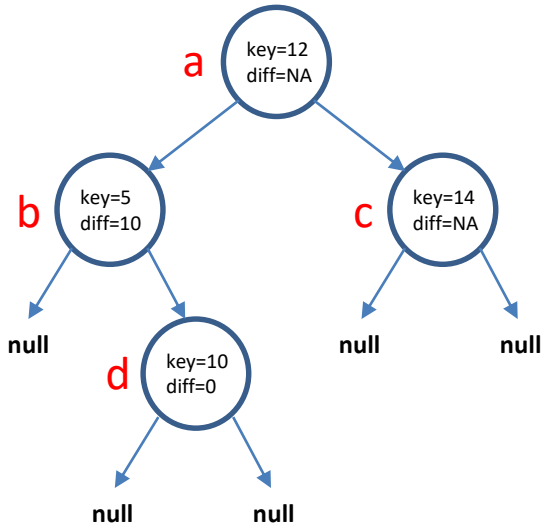


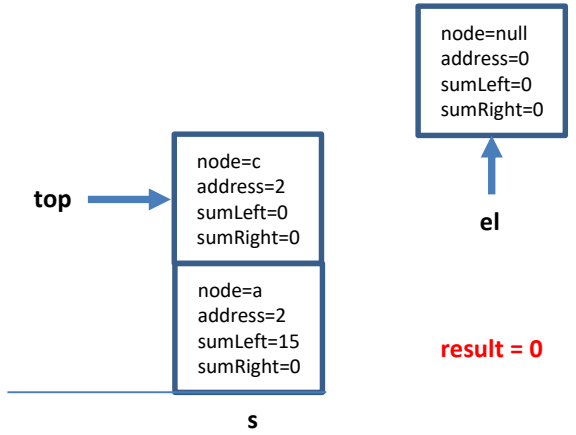


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

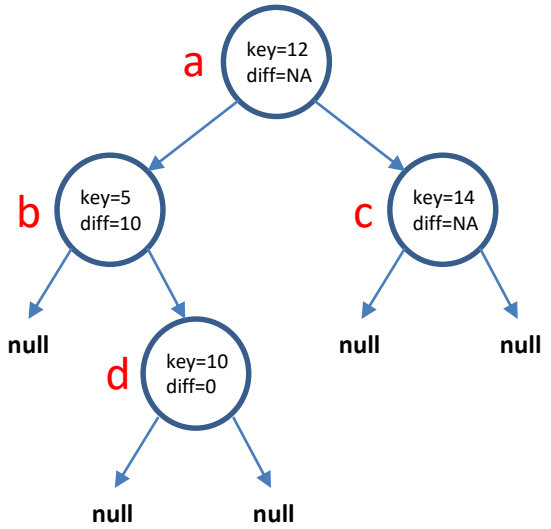


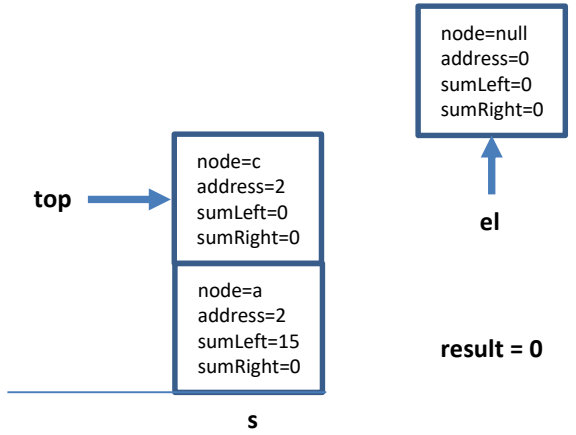


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

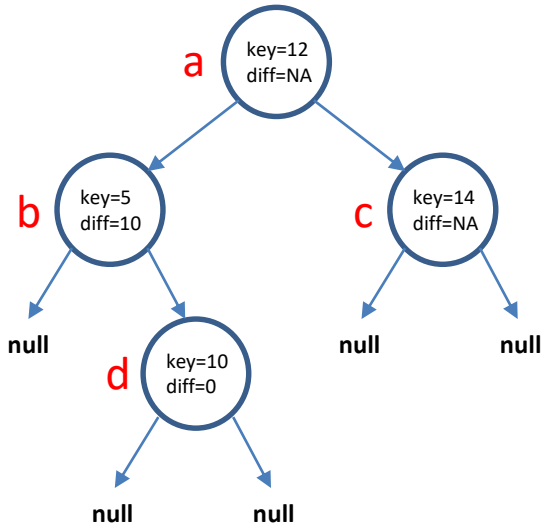


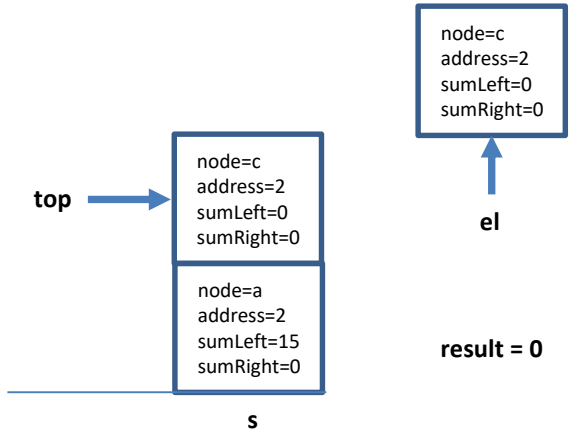


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

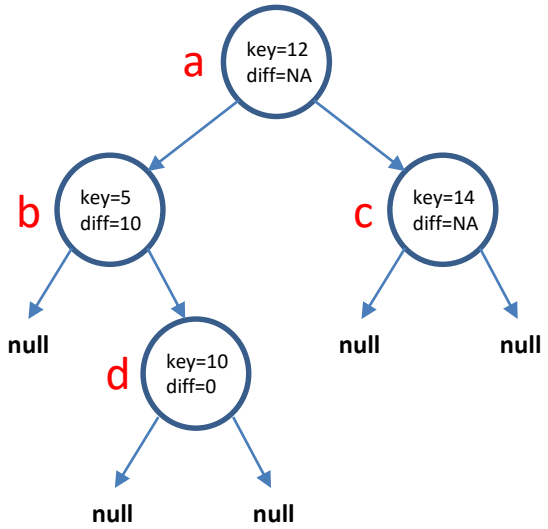


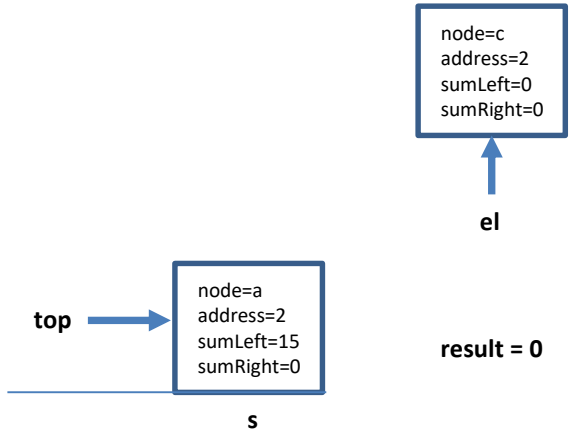


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

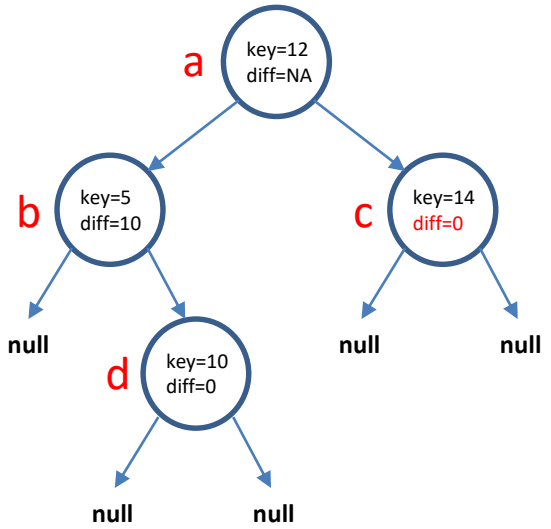


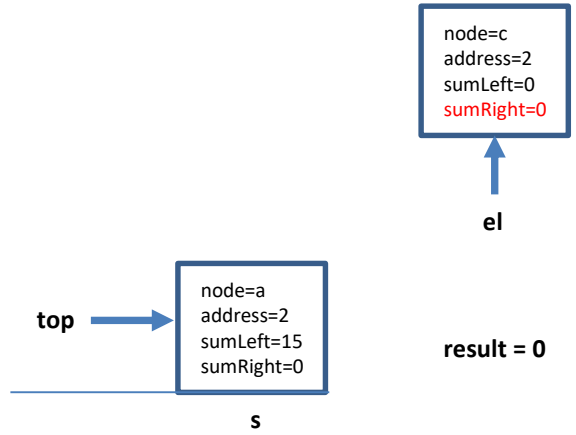


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

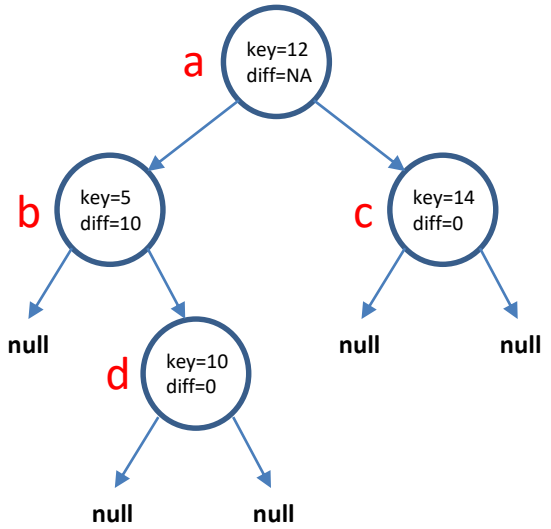


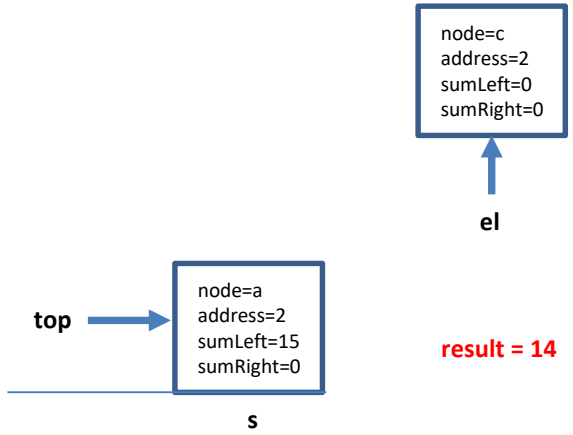


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```



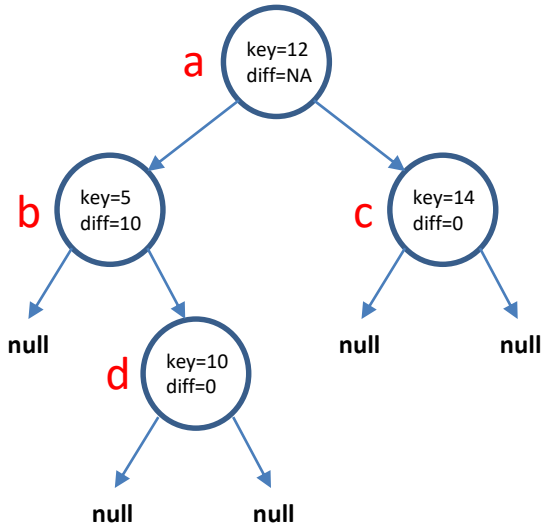


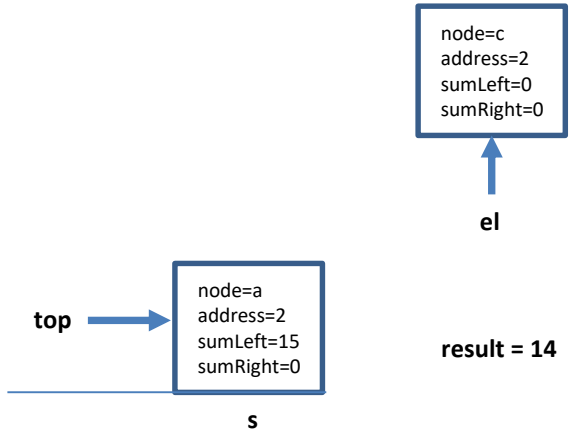
result = 14

REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

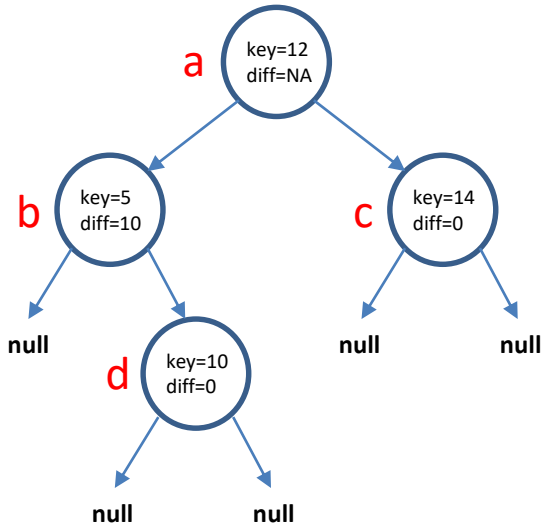


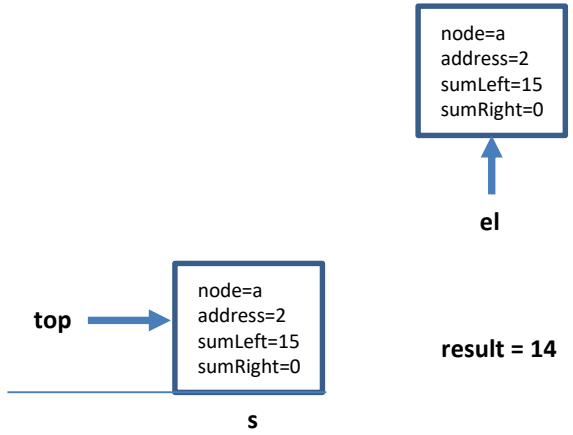


REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```

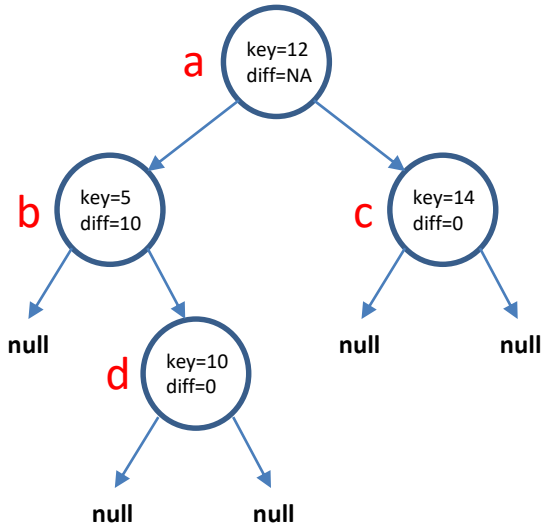


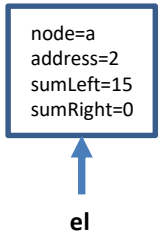


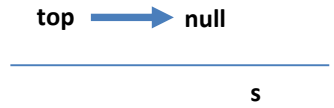
REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```





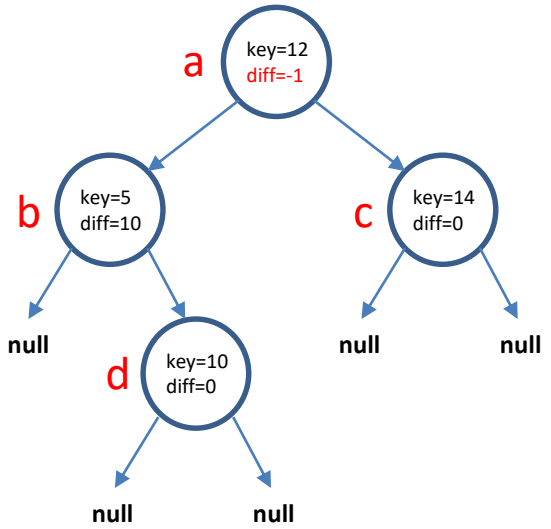


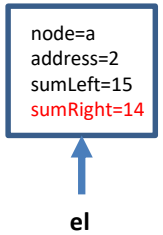
result = 14

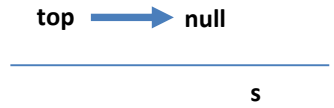
REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```





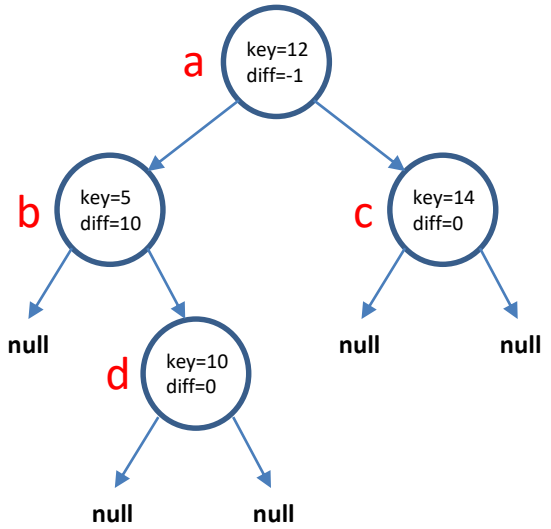


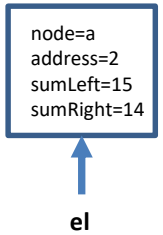
result = 14

REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```





top → null

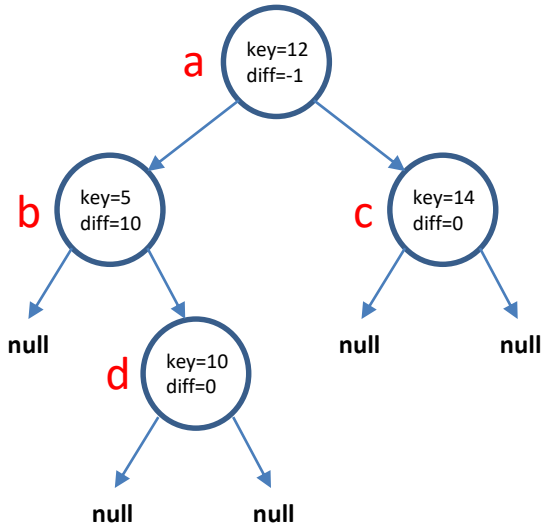
s

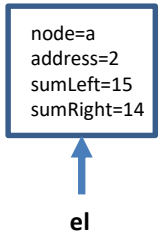
result = 41

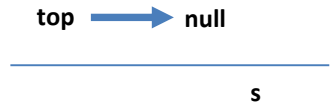
REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```





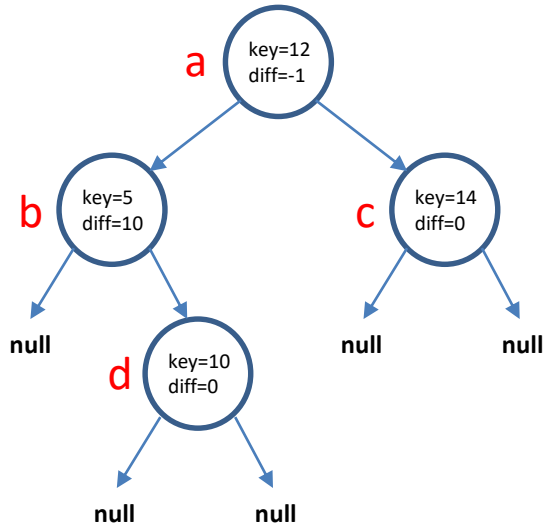


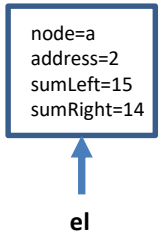
result = 41

REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```





top → null

s

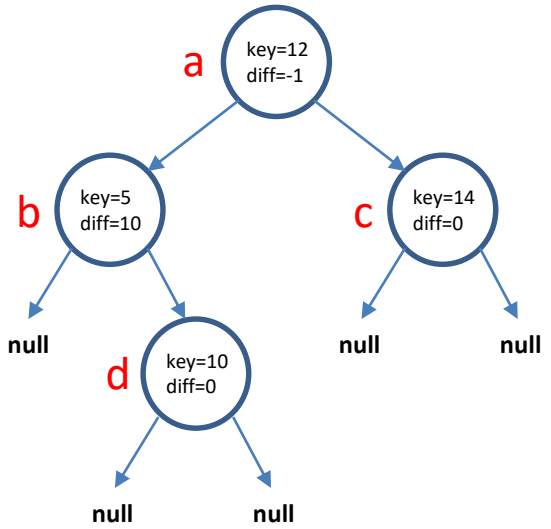
result = 41

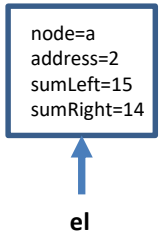
REŠITEV

```
public int sumIterStack(Node root) {
    Stack s = new Stack(100);
    StackElement el = new StackElement();
    el.node = root;
    el.address = 0;
    s.push(el);

    int result = 0;

    do {
        el = (StackElement)s.top();
        s.pop();

        ...

    } while (!s.empty());

    return result;
}
```

Funkcija vrne vrednost 41

```
switch(el.address) {
case 0:
    if (el.node == null)
        result = 0;
    else
    {
        el.address = 1;
        s.push(new StackElement(el));

        el.node = el.node.left;
        el.address = 0;
        s.push(new StackElement(el));
    }
    break;

case 1:
    el.sumLeft = result;

    el.address = 2;
    s.push(new StackElement(el));

    el.node = el.node.right;
    el.address = 0;
    s.push(new StackElement(el));
    break;

case 2:
    el.sumRight = result;
    el.node.diff = el.sumRight - el.sumLeft;
    result = el.node.key + el.sumLeft + el.sumRight;
    break;
}
```





top → null

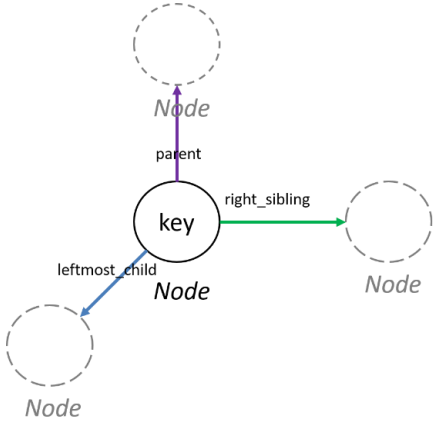
s

result = 41

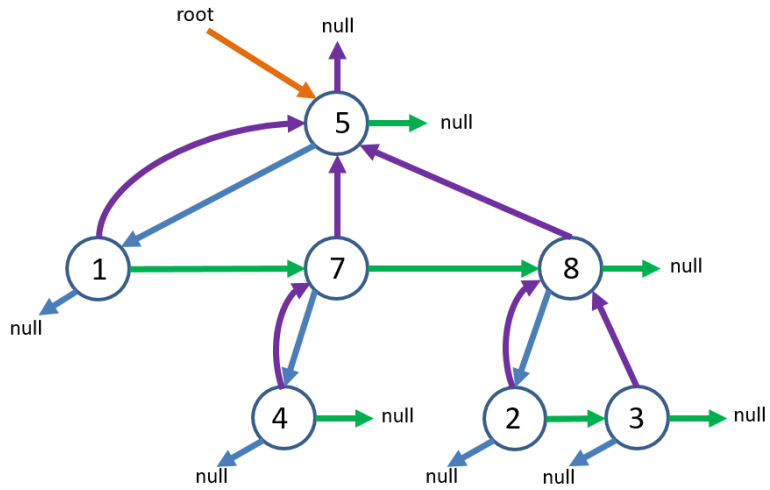
NALOGA 5

Dano je drevo (dostopno preko korena root), ki za vsako vozlišče vsebuje naslednje podatke:

```
class Node {
   int key;
   Node parent;
   Node leftmost_child;
   Node right_sibling;
}
```



Sestavi algoritem, ki bo izpisal elemente binarnega drevesa po nivojih (torej, najprej izpiše koren, nato izpiše njegove sinove od leve proti desni, zatem sledijo vnuki od leve proti desni itd.)



Izberi ustrezne parametre in oceni časovno zahtevnost svojega algoritma.

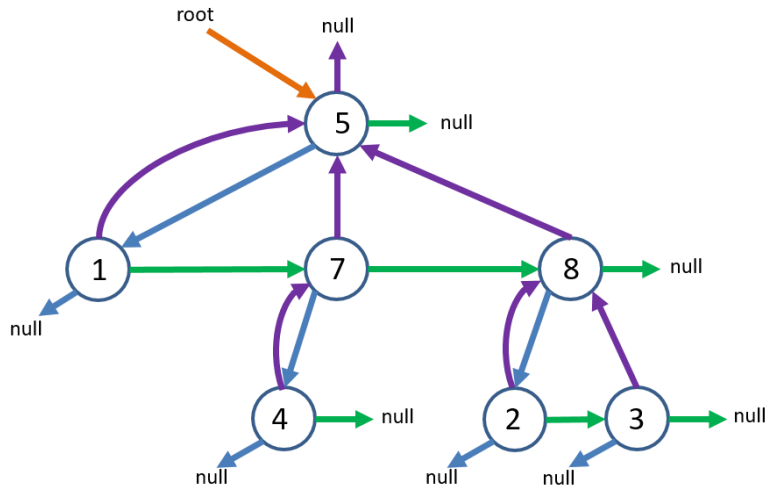
REŠITEV

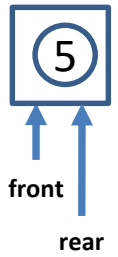
```
class Node {
    int key;
    Node parent, leftmost_child, right_sibling;
}


public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

Časovna kompleksnost: O(n), n je število vozlišč v drevesu.

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

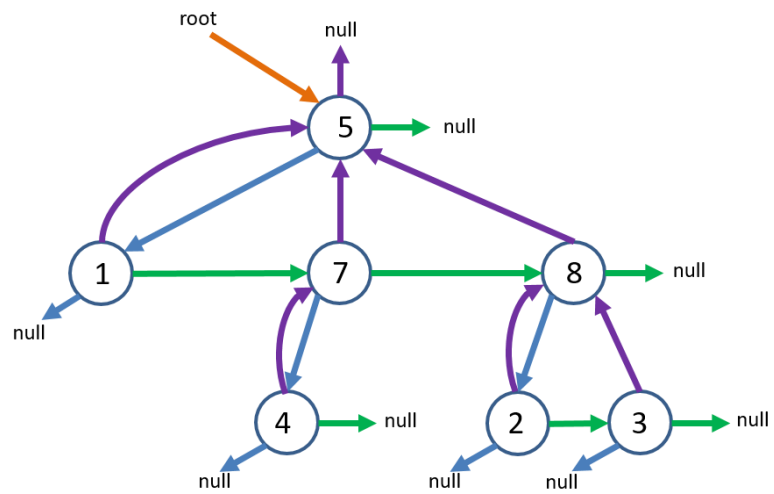


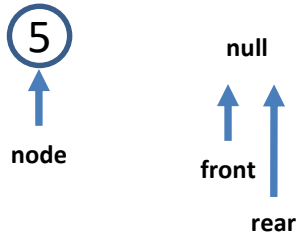


Izpis:

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

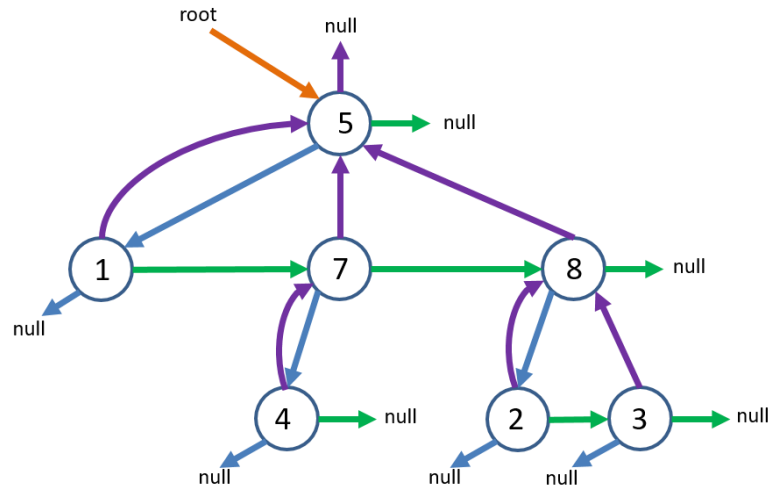


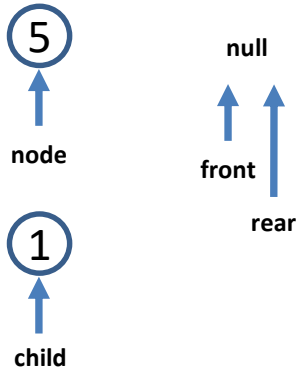


Izpis:

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

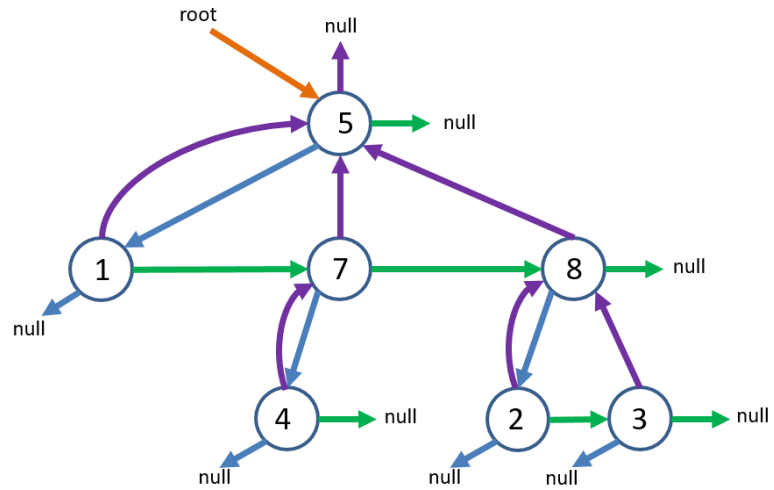


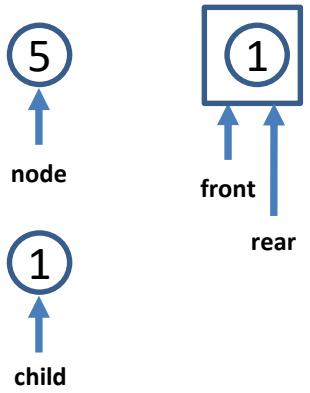


Izpis: 5,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

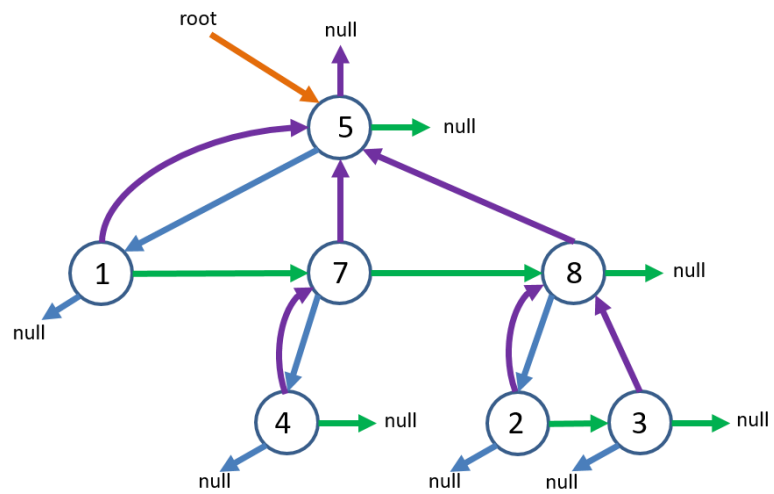


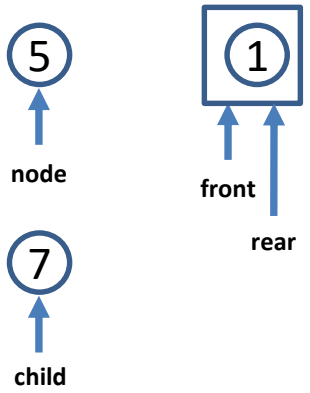


Izpis: 5,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

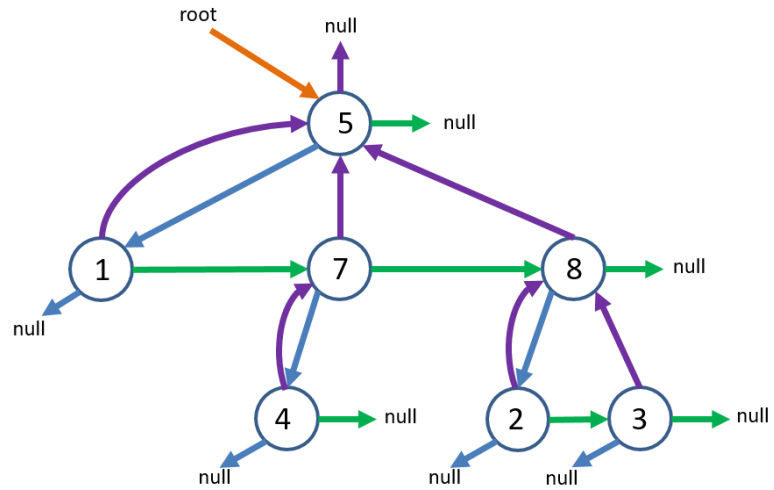


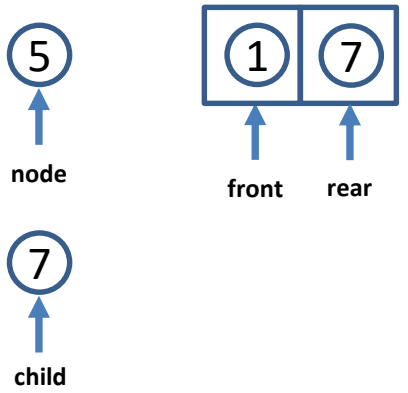


Izpis: 5,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

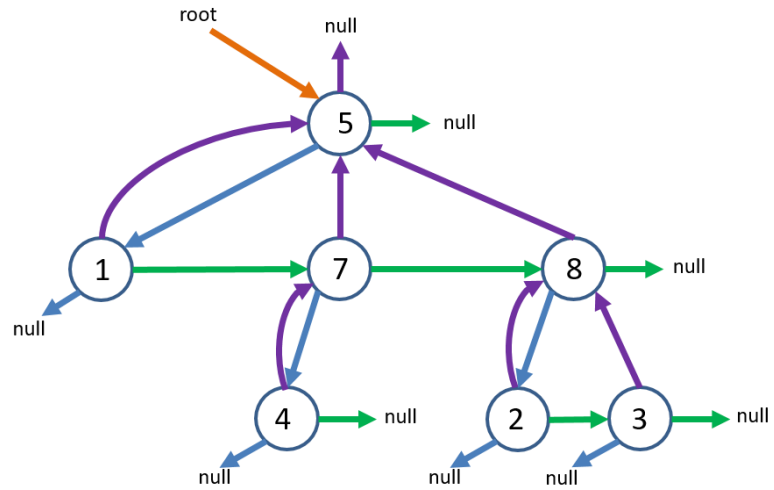


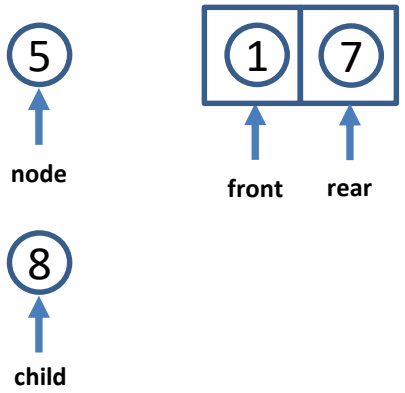


Izpis: 5,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

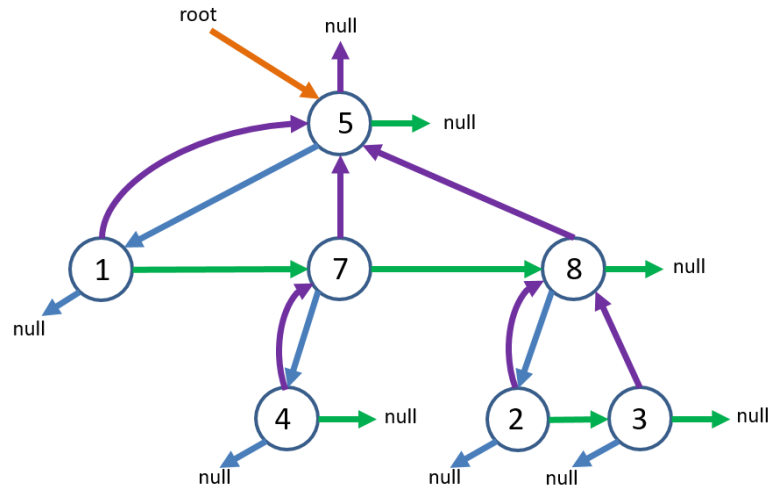


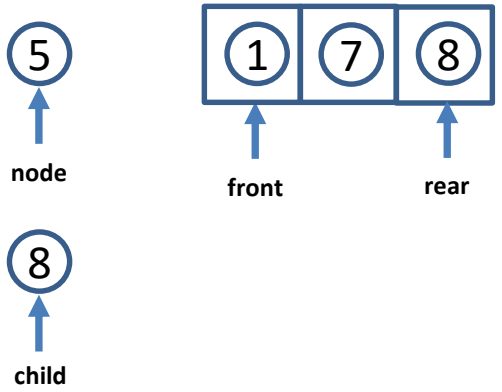


Izpis: 5,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

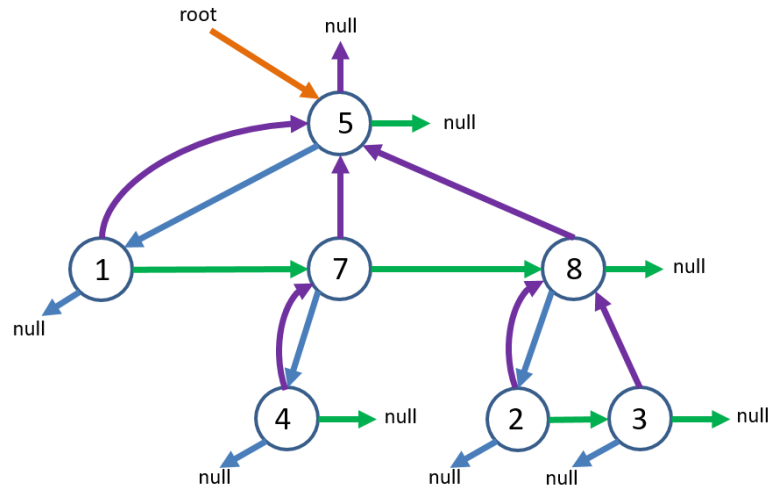


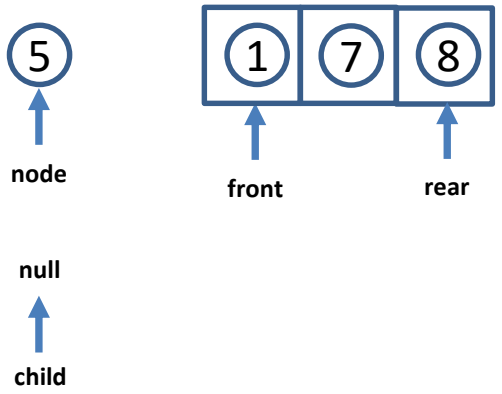


Izpis: 5,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

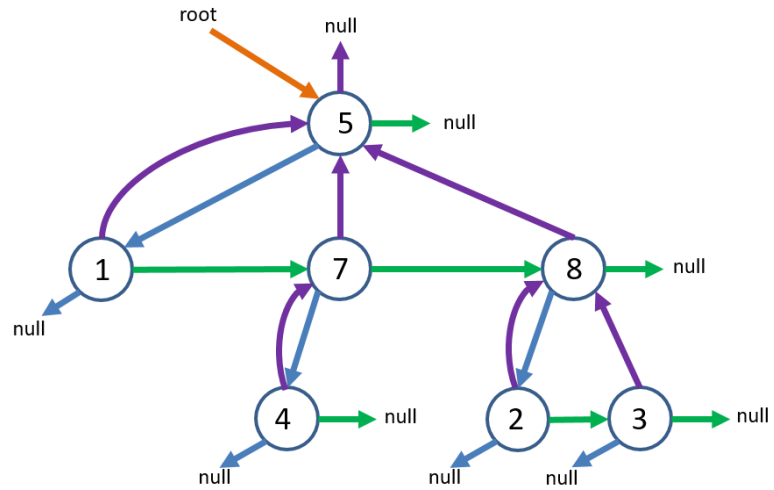


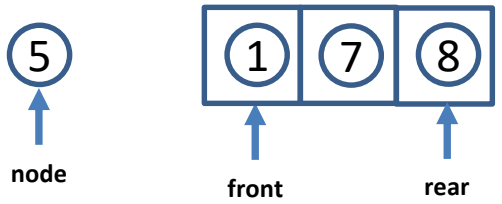


Izpis: 5,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

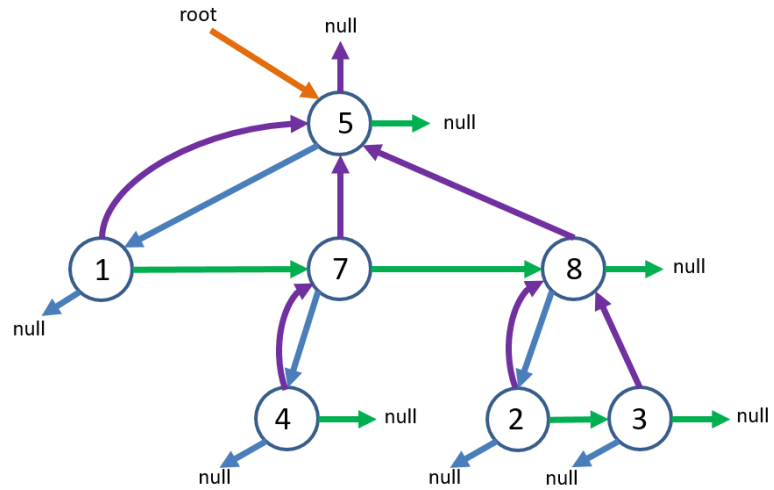


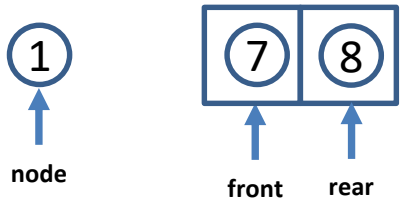


Izpis: 5,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

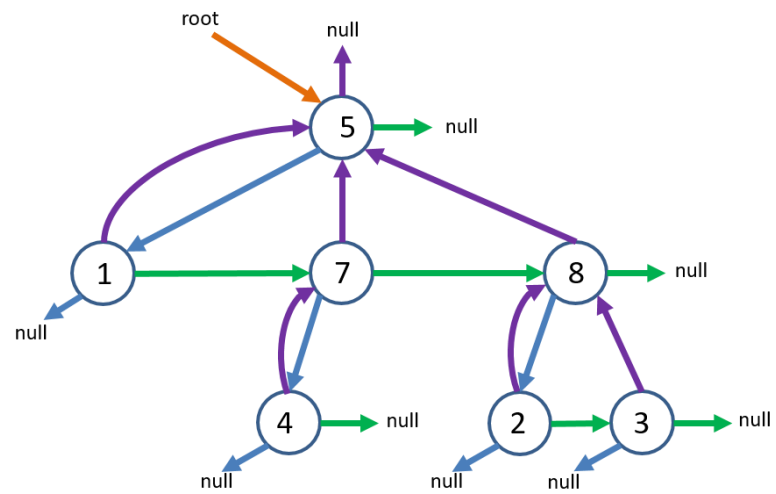


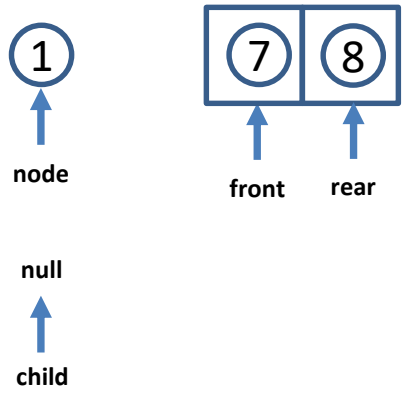


Izpis: 5,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

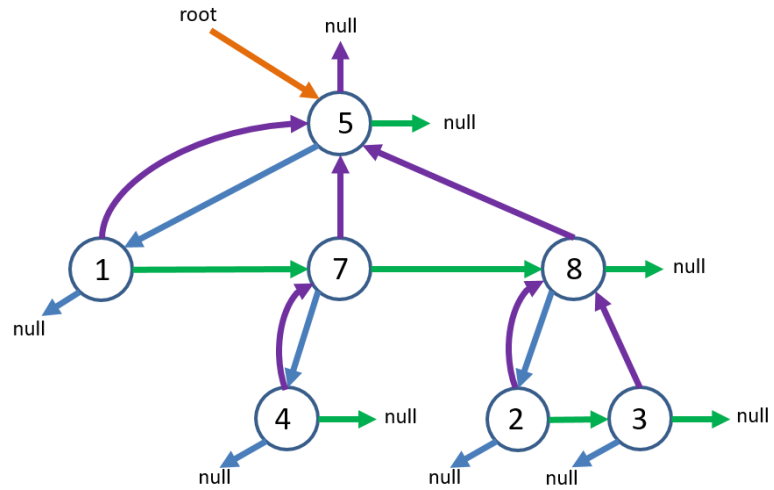


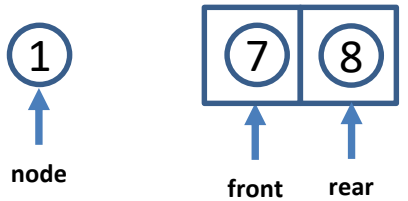


Izpis: 5, 1,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

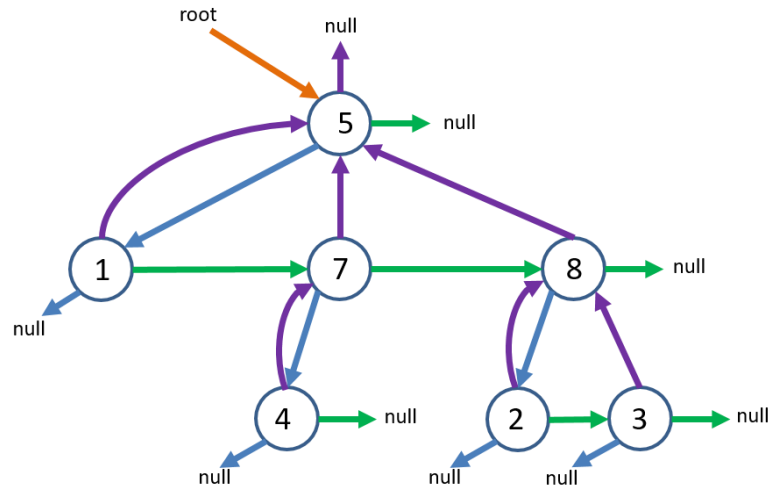


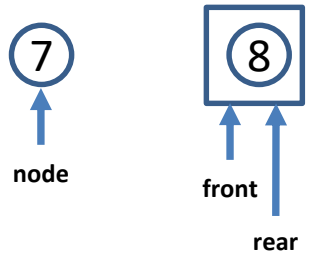


Izpis: 5, 1,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

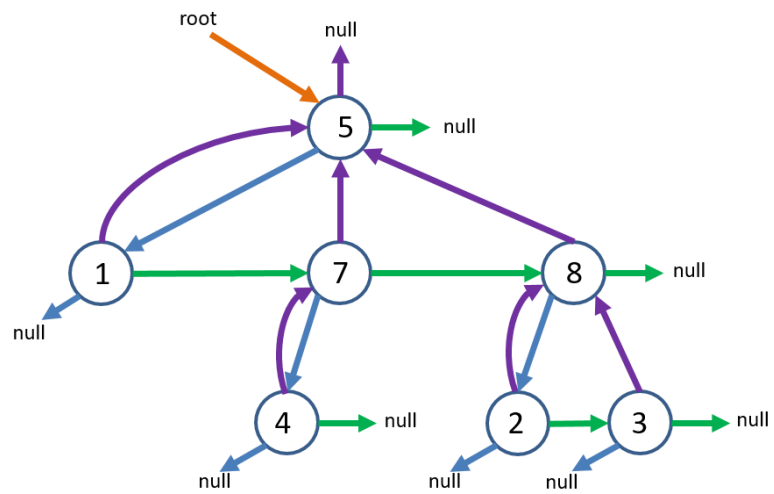


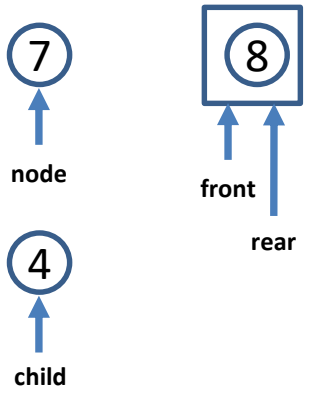


Izpis: 5, 1,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

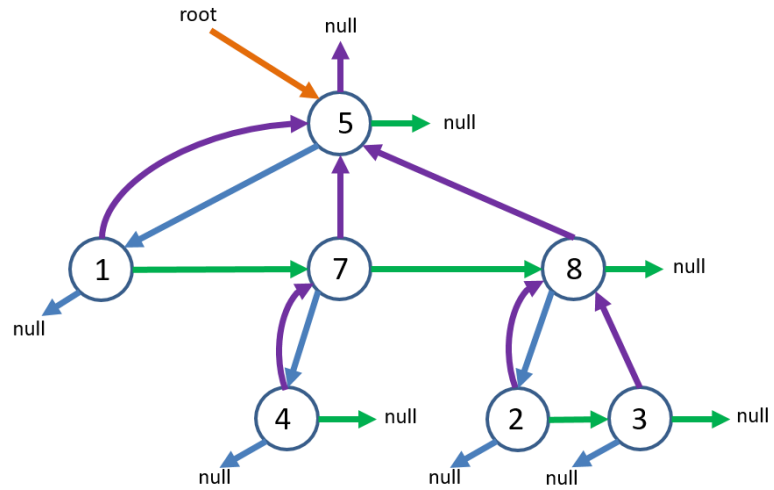


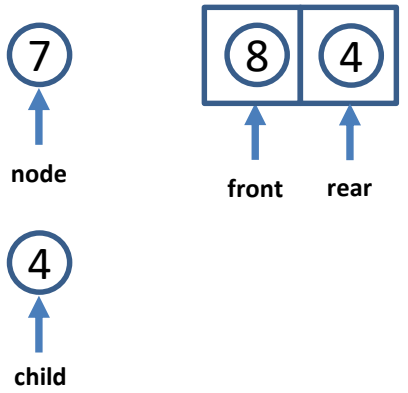


Izpis: 5, 1, 7,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

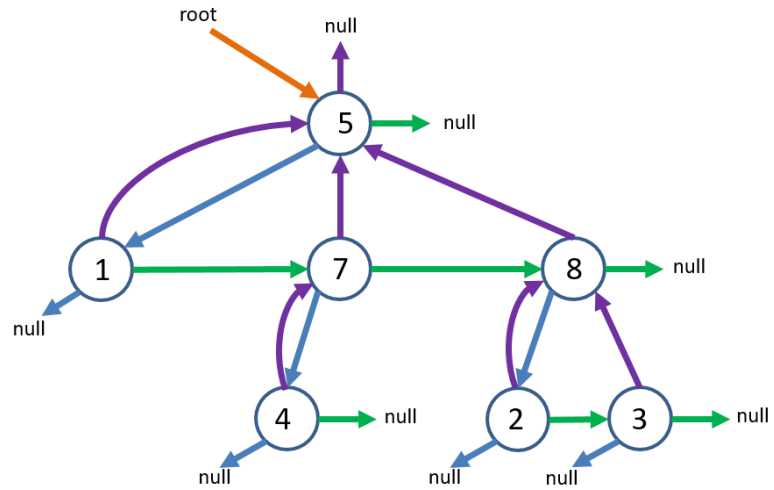


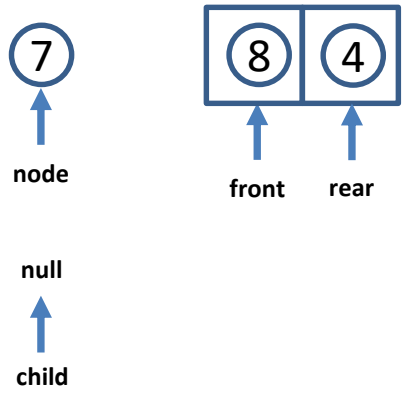


Izpis: 5, 1, 7,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

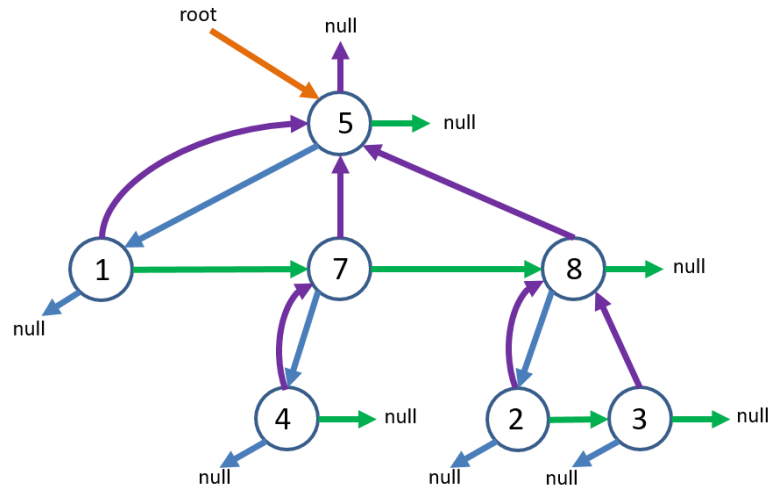


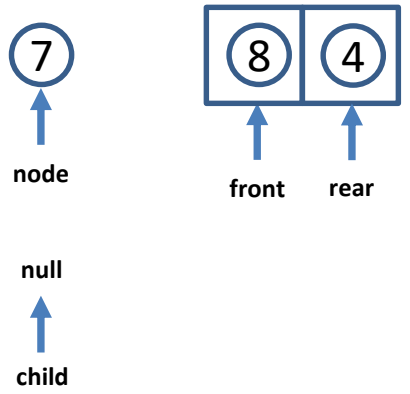


Izpis: 5, 1, 7,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

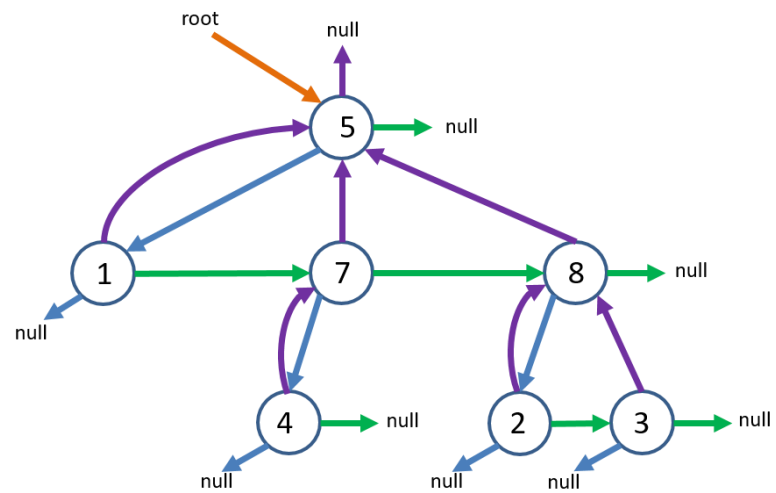


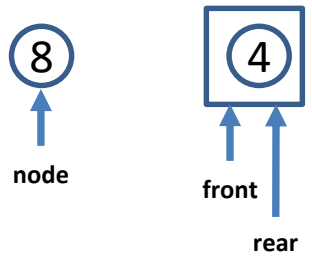


Izpis: 5, 1, 7,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

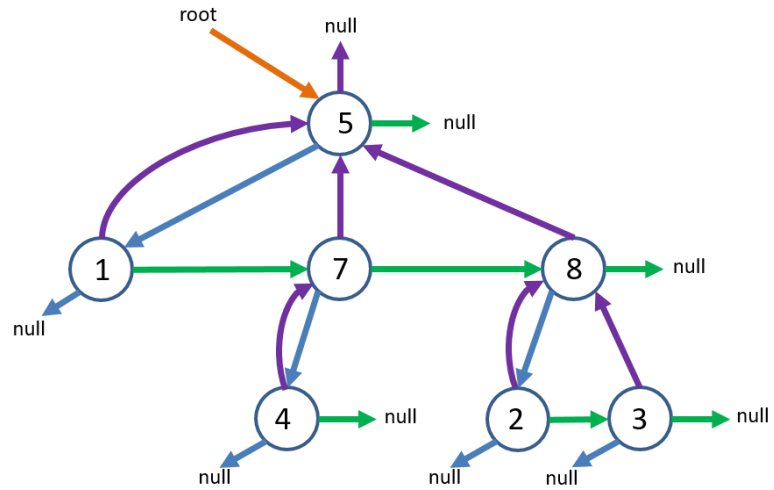


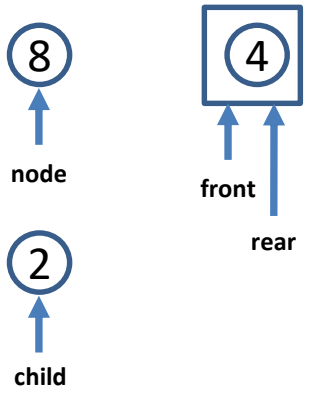


Izpis: 5, 1, 7,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

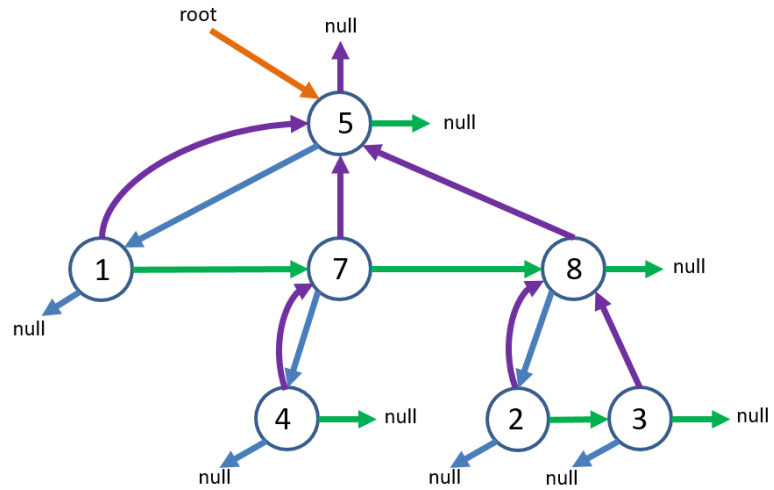


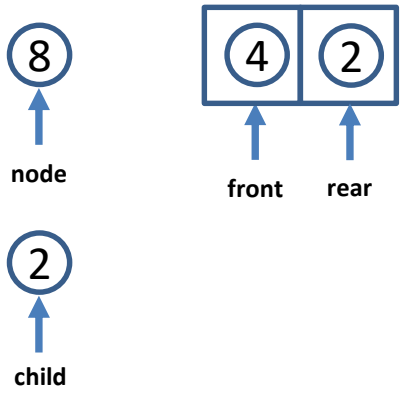


Izpis: 5, 1, 7, 8,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

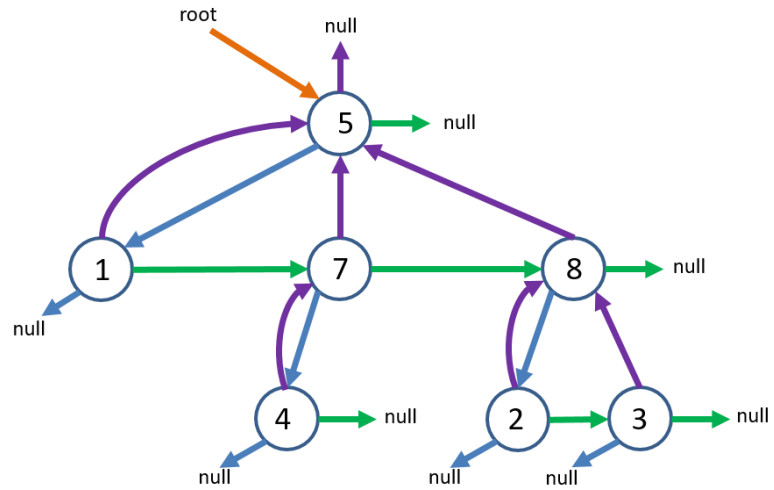


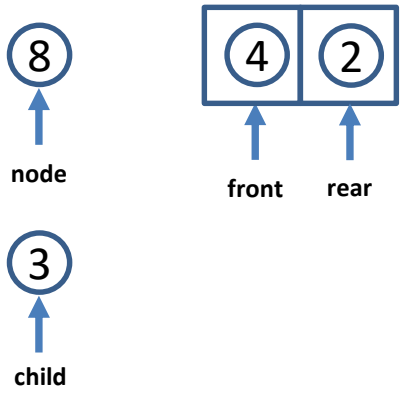


Izpis: 5, 1, 7, 8,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

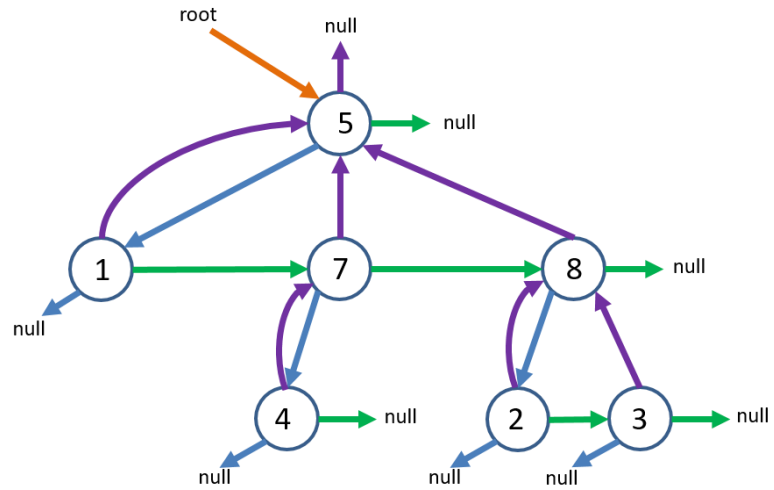


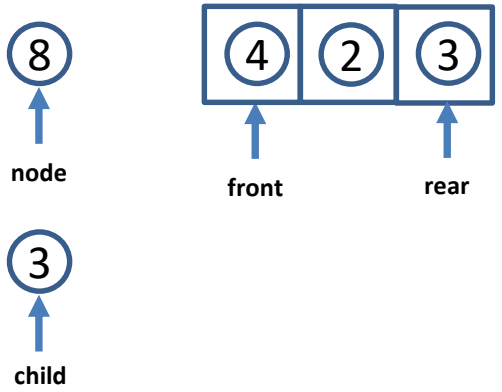


Izpis: 5, 1, 7, 8,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

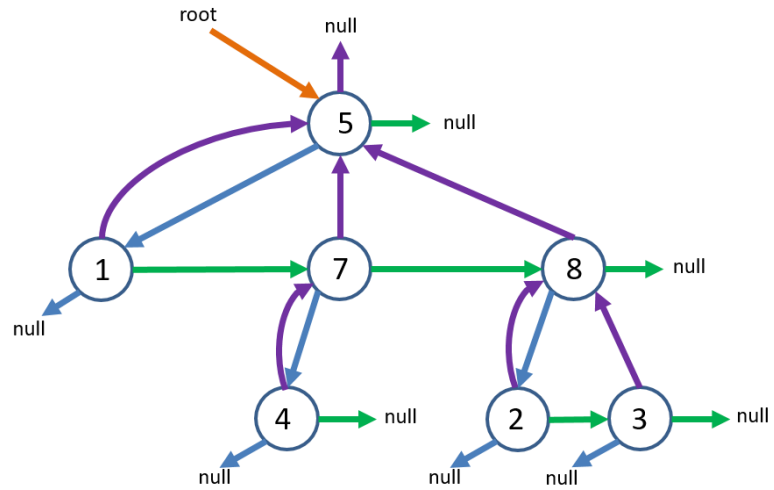


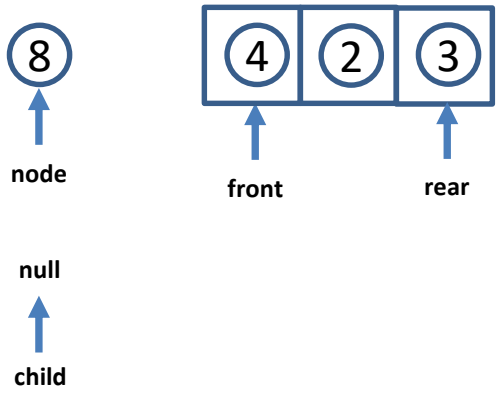


Izpis: 5, 1, 7, 8,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

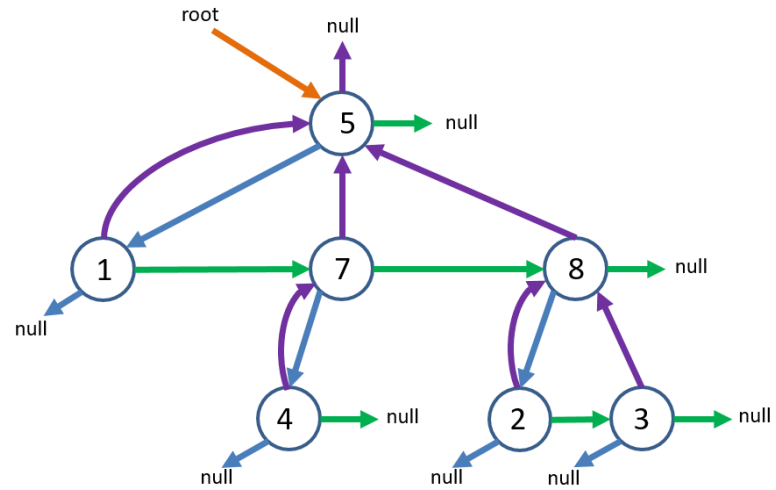


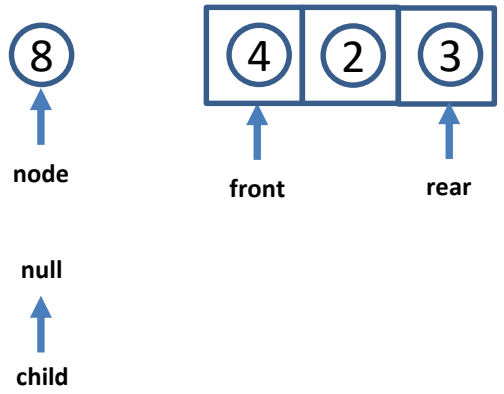


Izpis: 5, 1, 7, 8,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

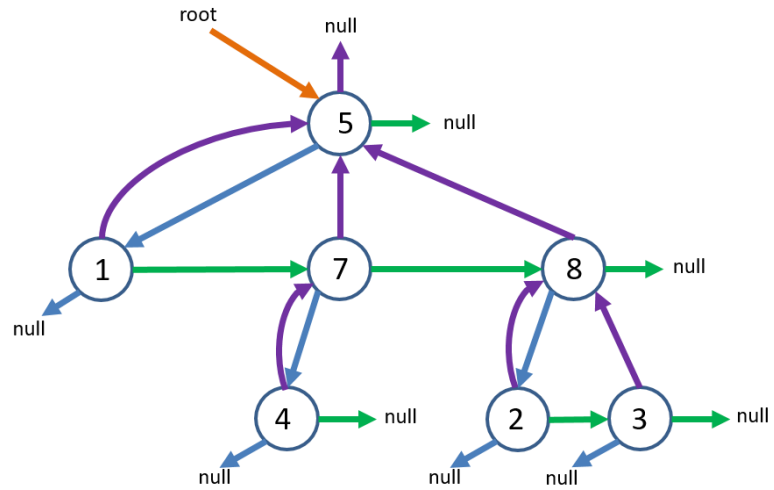


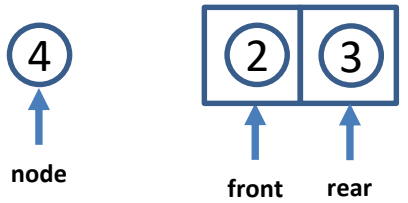


Izpis: 5, 1, 7, 8,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

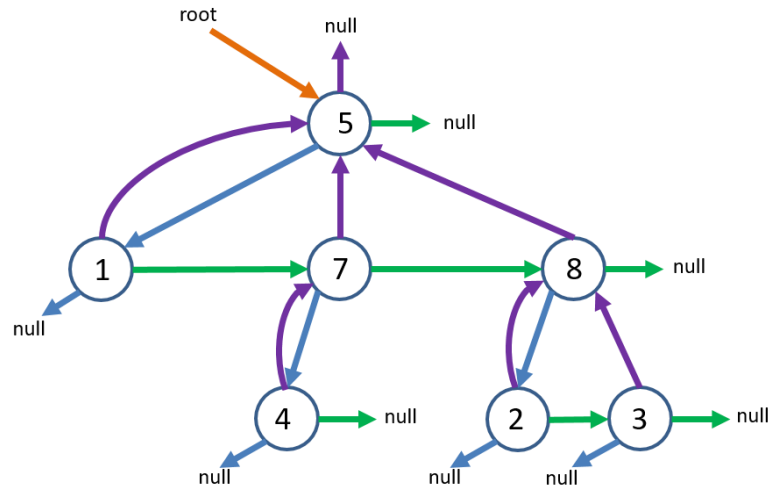


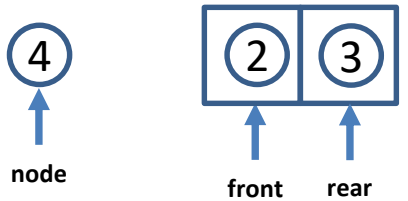


Izpis: 5, 1, 7, 8,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

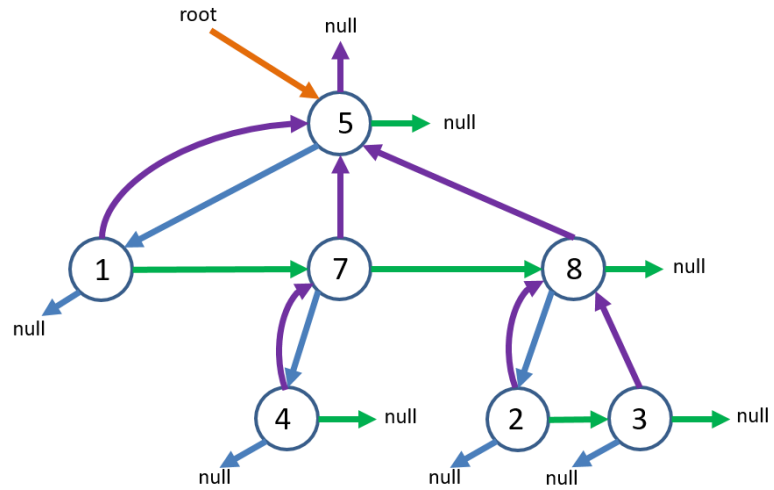


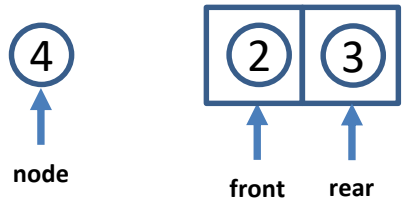


Izpis: 5, 1, 7, 8, 4,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

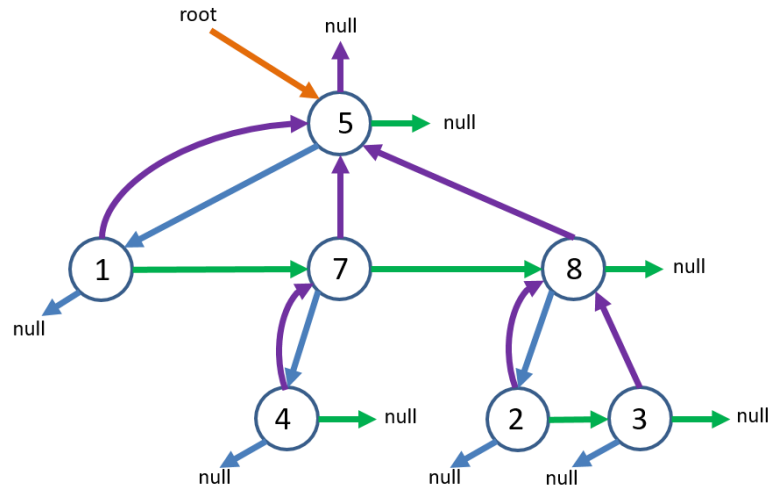


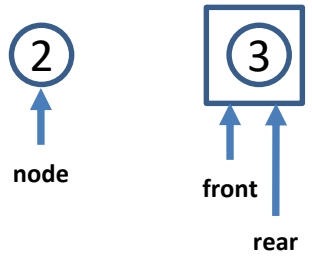


Izpis: 5, 1, 7, 8, 4,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

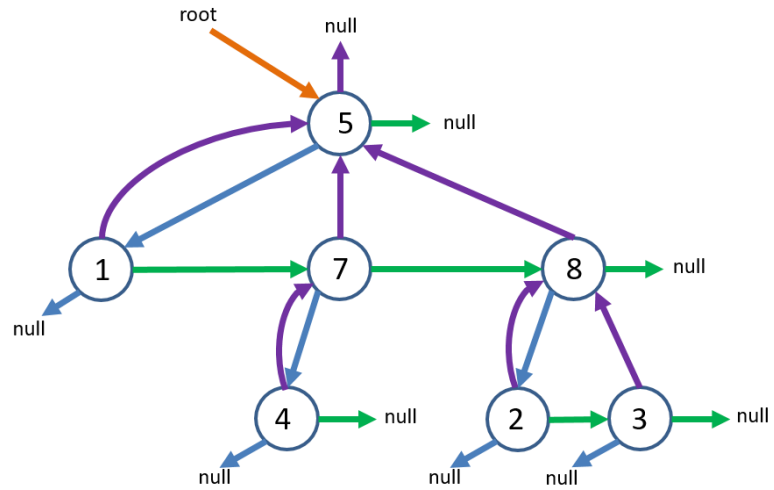


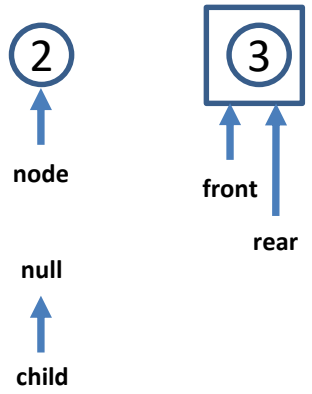


Izpis: 5, 1, 7, 8, 4,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

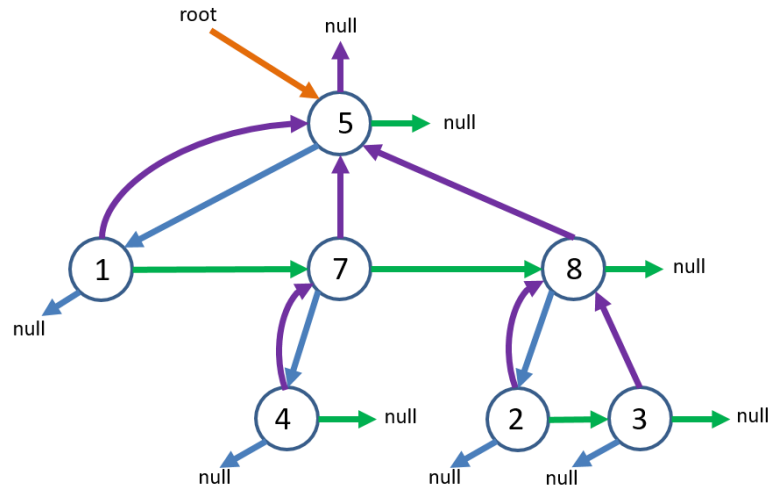


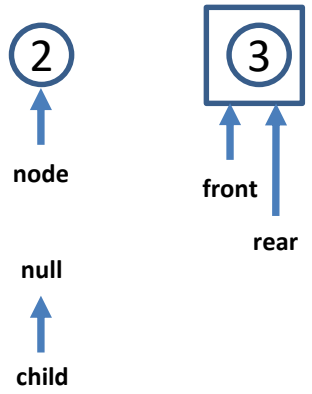


Izpis: 5, 1, 7, 8, 4, 2,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

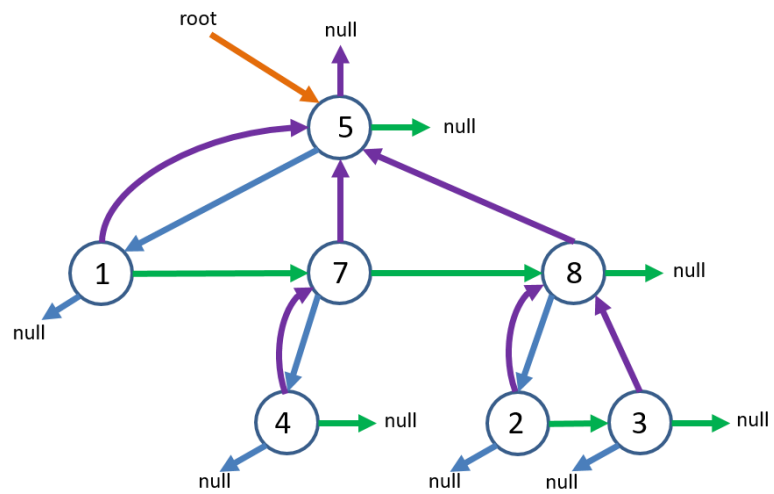


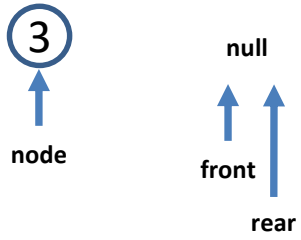


Izpis: 5, 1, 7, 8, 4, 2,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

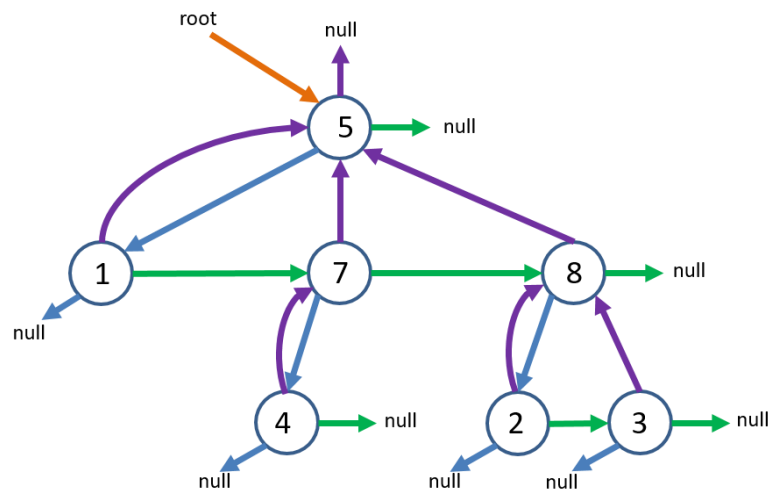


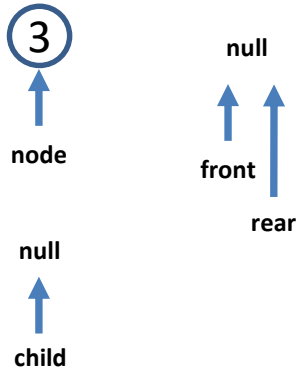


Izpis: 5, 1, 7, 8, 4, 2,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

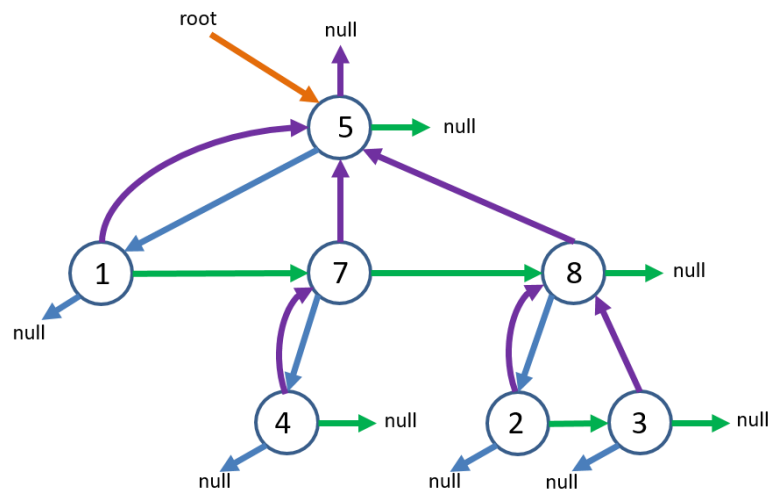


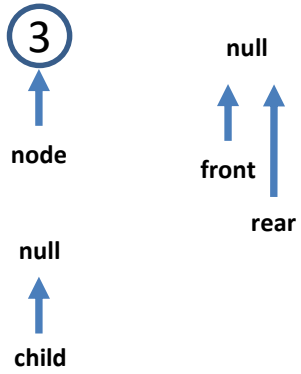


Izpis: 5, 1, 7, 8, 4, 2, 3,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

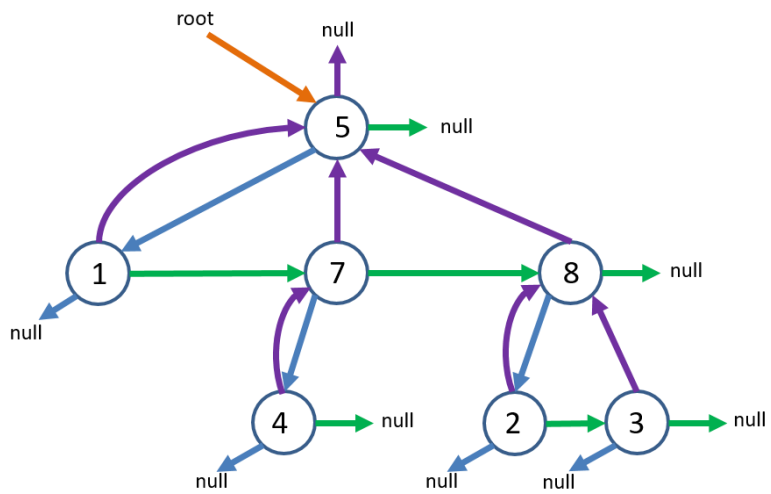


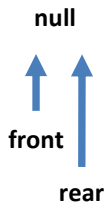


Izpis: 5, 1, 7, 8, 4, 2, 3,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```

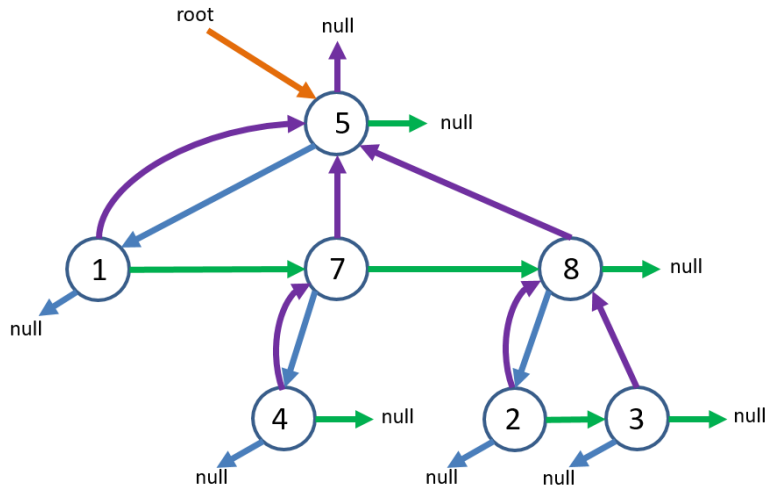


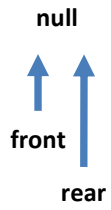


Izpis: 5, 1, 7, 8, 4, 2, 3,

REŠITEV

```
public static void printByLevels(Node root) {
    Queue queue = new Queue();

    if (root != null)
        queue.enqueue(root);

    while (!queue.empty()) {
        Node node = (Node)queue.front();
        queue.dequeue();

        System.out.print(node.key + ", ");

        Node child = node.leftmost_child;
        while (child != null) {
            queue.enqueue(child);
            child = child.right_sibling;
        }
    }

    System.out.println();
}
```





Izpis: 5, 1, 7, 8, 4, 2, 3,