



Национална програма
"Обучение за ИТ умения и кариера"
<https://it-kariera.mon.bg>

Министерството на
образованието и науката
<https://www.mon.bg>

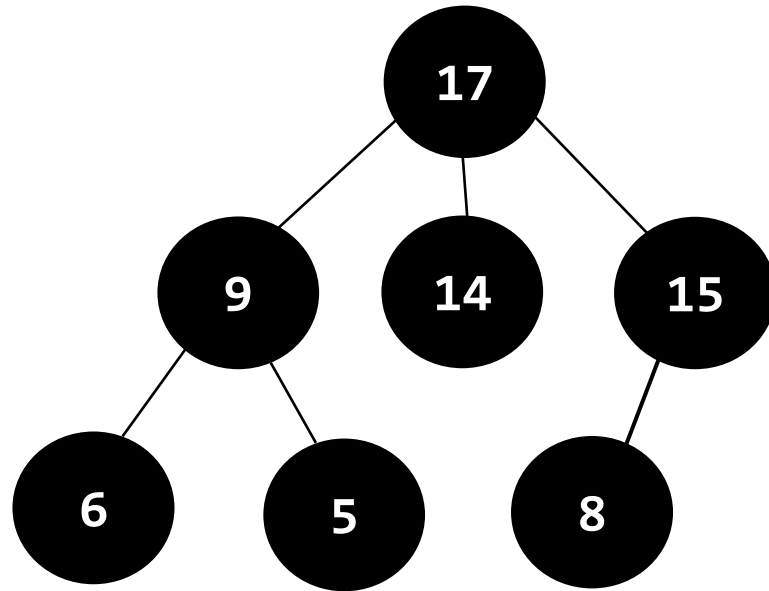


Дървовидни структури от данни и алгоритми върху тях

Алгоритми и структури от данни

Съдържание

- Дървета и дървовидни структури
- Подредени двоични дървета, балансирани дървета, В-дървета
- Упражнения: структура от данни “дърво”, използване на класове и библиотеки за дървовидни структури
- Обхождания в дълбочина и ширина (DFS и BFS)
- Упражнения: обхождане в дълбочина (DFS)
- Упражнения: обхождане в ширина (BFS)

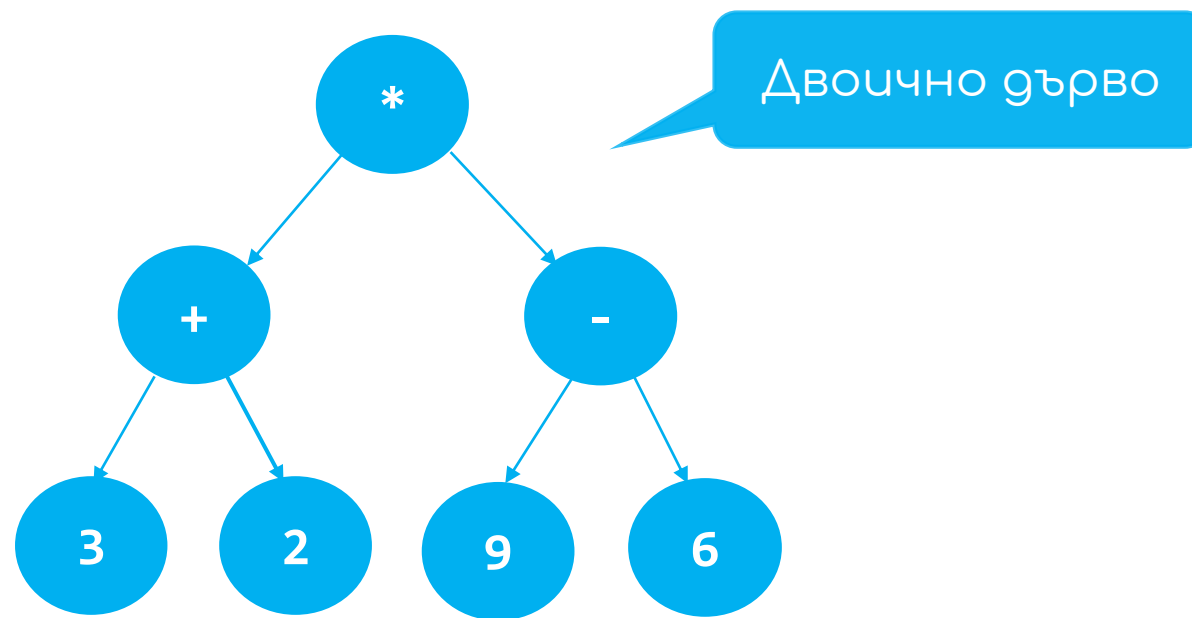
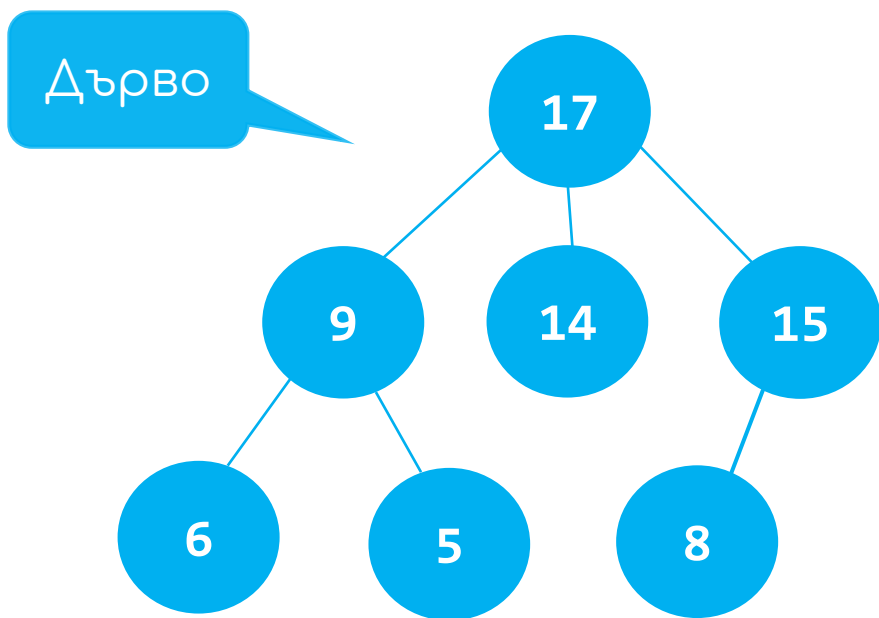


Дървовидни структури от данни

Терминология

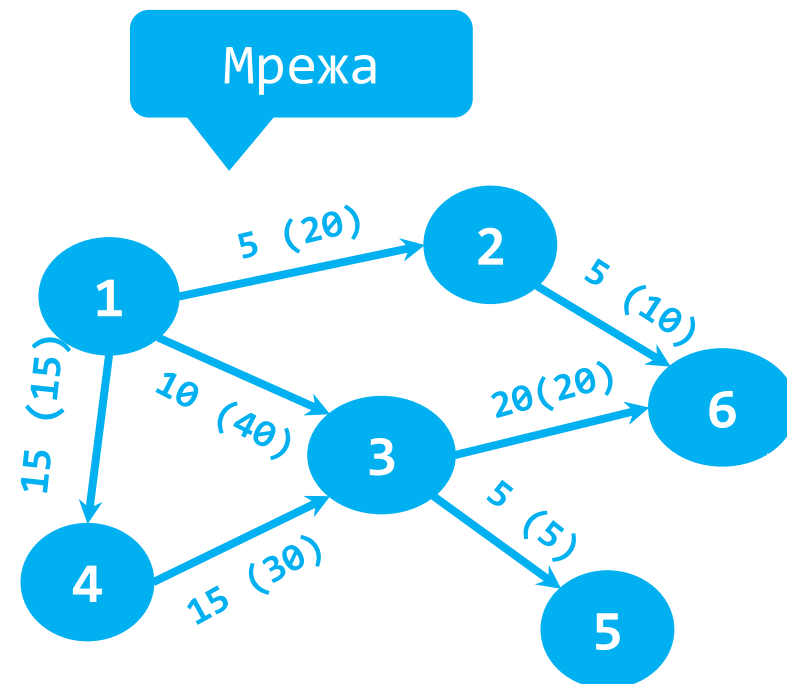
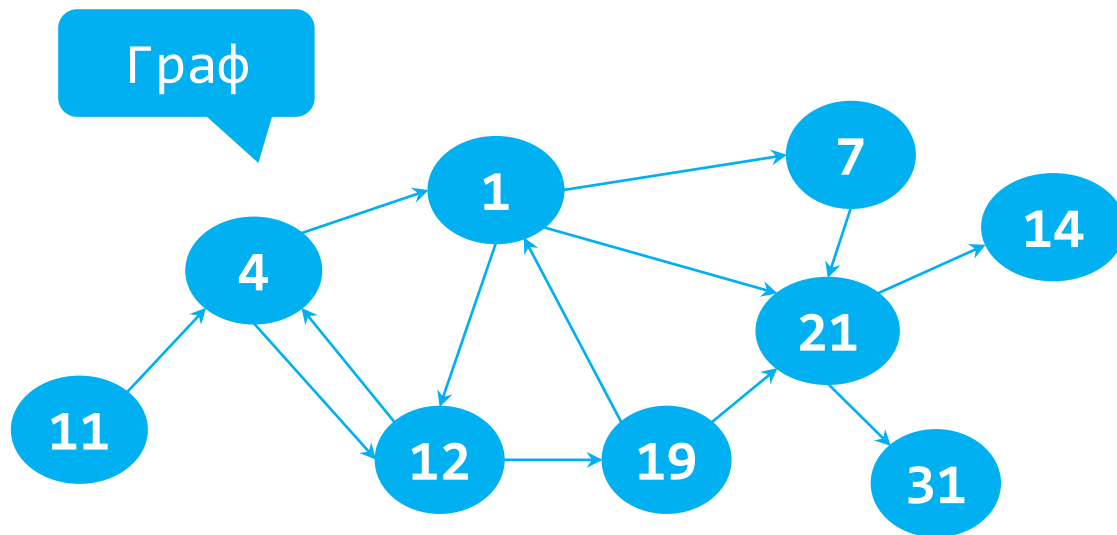
Дървовидни структури от данни [1/2]

- Дървовидните структури от данни са:
 - Разклонени йерархични структури от данни
 - Изградени от възли
 - Всеки възел е свързан с други възли (разклонения на дървото)



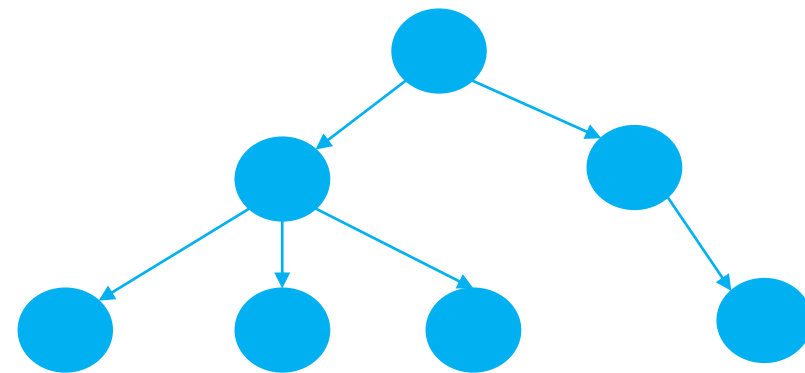
Дървовидни структури от данни [2/2]

- Дървовидните структури от данни:
 - **Дървета** - двоични, балансирани, подредени и др.
 - **Графи** - ориентирани, неориентирани, с тегла и др.
 - **Мрежи** - графи с особени свойства



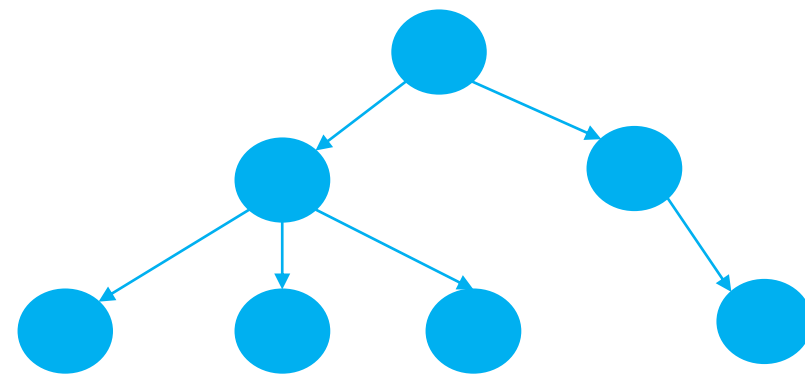
Дефиниция за дървета

- Нека $D = \{V, E\}$ е кореново дърво
 - Всяко дърво се образува от възли и дъги, които ги свързват
 - Формално върховете могат да бъдат от два вида:
 - Родител
 - Наследник
 - Върхът без родител се нарича **корен**
 - Всяко дърво има само корен
 - Върх без наследници се нарича **листо**



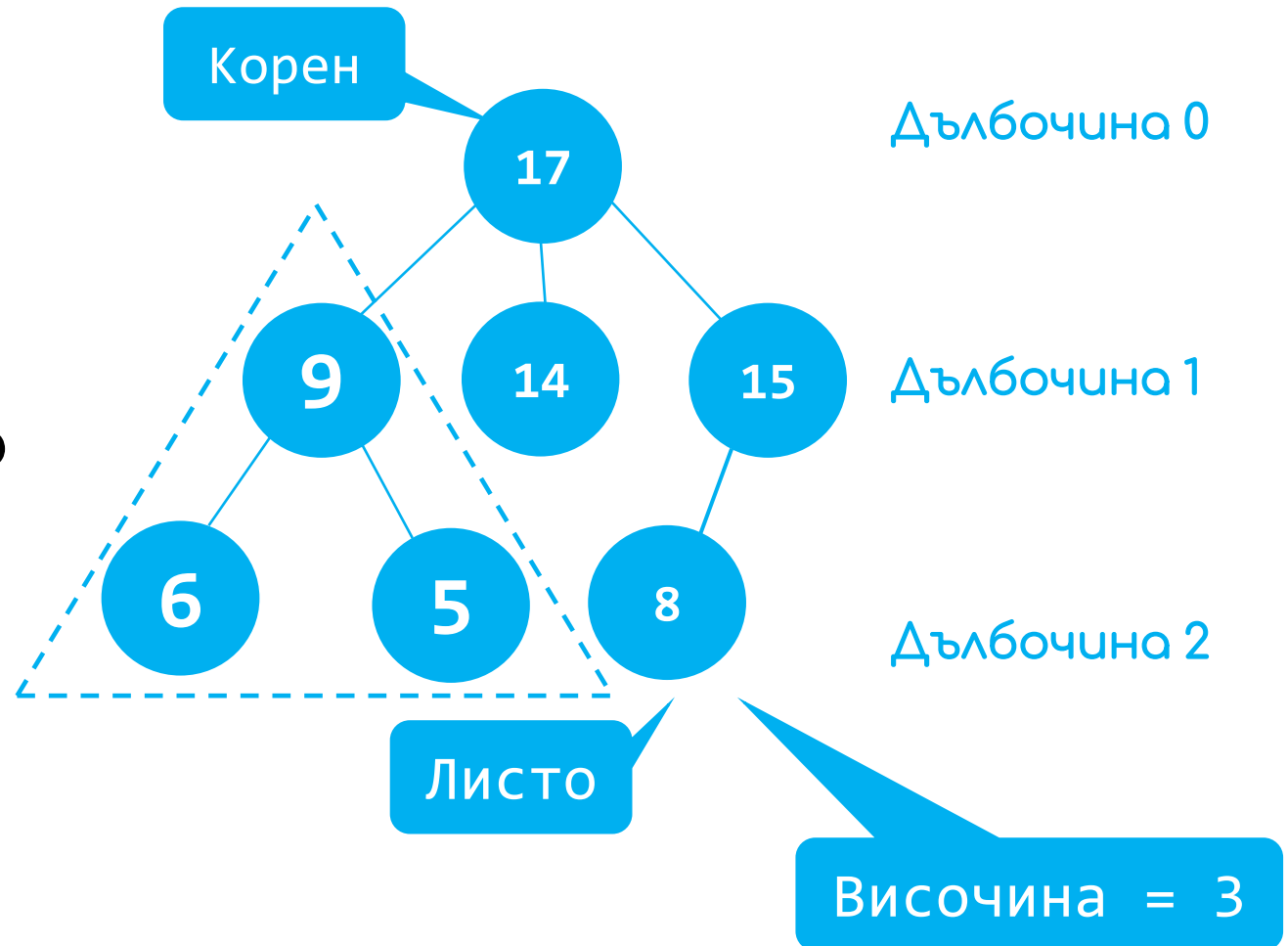
Обща дефиниция за дърво

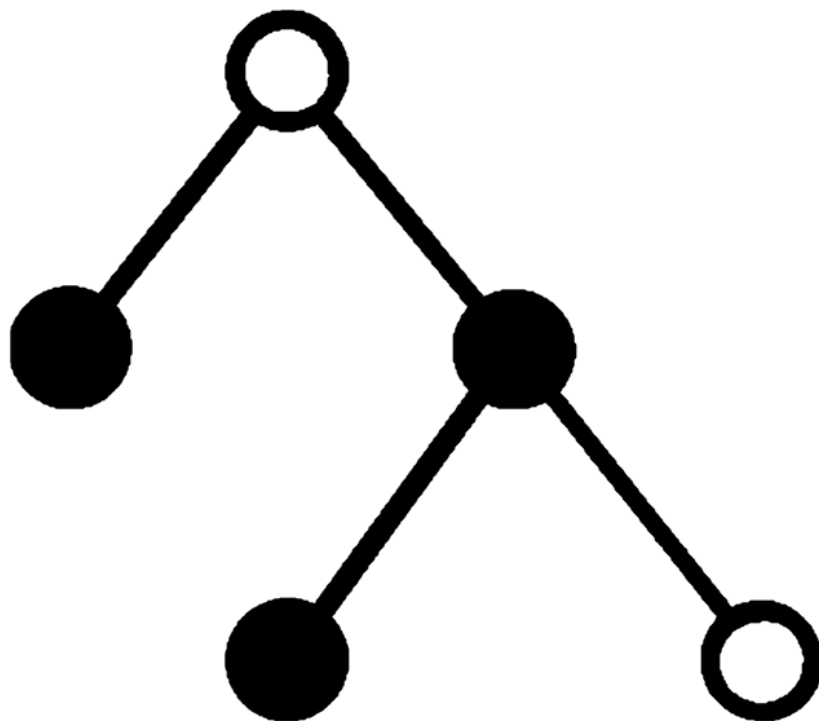
- Дърво от тип T е структура, образувана от:
 - Елемент от тип T , наречен **корен**
 - Крайно множество елементи от тип T , наречени **поддървета**
- Дървото се бележи с $T = \{V, E\}$, където:
 - V е множеството от възли в структурата
 - E е множеството от ребра в структурата
- Дървета, в които T има k на брой разклонения наричаме k -ични дървета



Терминология за дървовидни структури от данни

- Възел, Ребро
- Корен, Родител, Дете, Брат
- Дълбочина, Височина
- Под-дърво
- Вътрешен възел, листо
- Предшественик, Наследник





Двоични дървета

Терминология

Двоични дървета

Възлите на двоичните дървета имат по не повече от две разклонения

- Двоични дървета
 - Няма правила за подредба на елементите
- Наредени (сортирани) двоични дървета (правила за подредба на възлите)
- Двоични дървета за търсене (частен случай на сортирани дървета):
 - Лявото разклонение на всеки възел има по-малка стойност от стойността на възела
 - Дясното разклонение на всеки възел има по-голяма стойност от стойността на възела.

Рекурсивна дефиниция на дървета

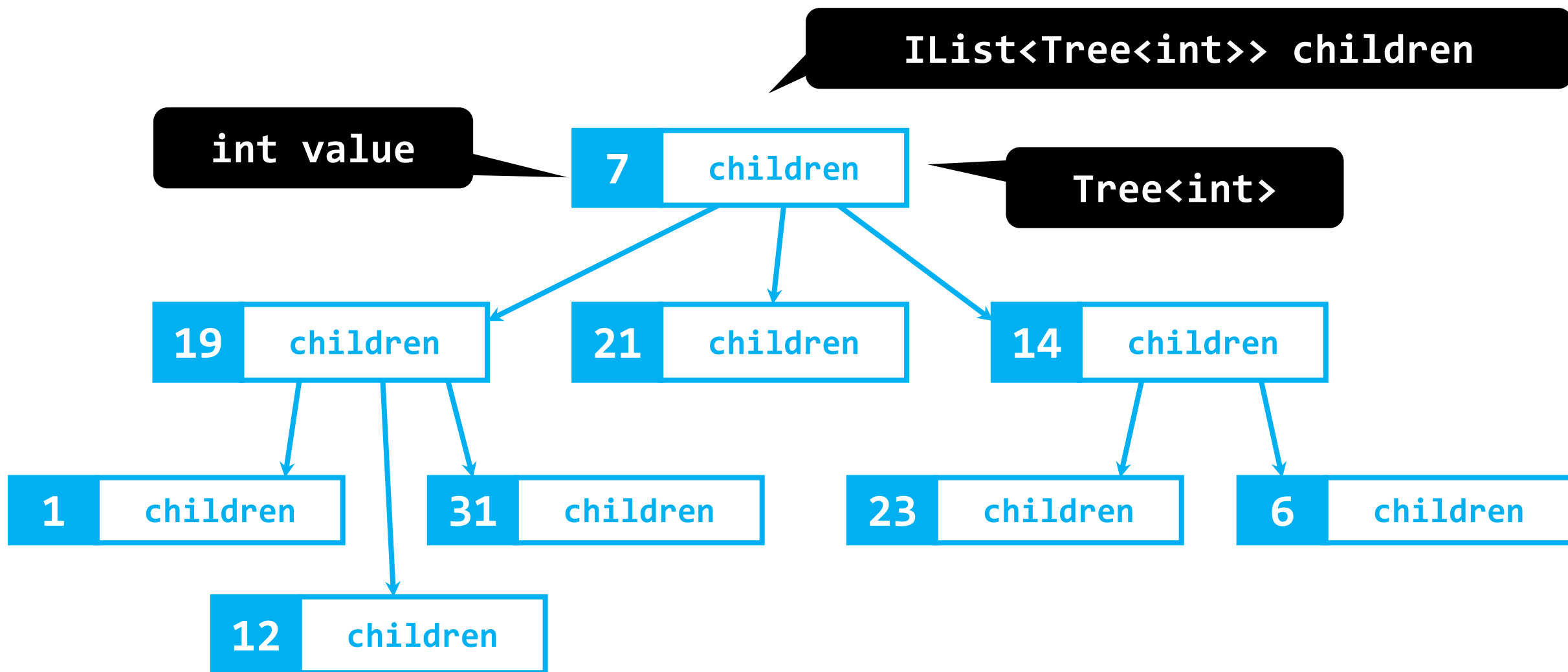
- Рекурсивна дефиниция на дървета:
 - Всеки възел е дърво
 - Възлите имат 0 или много деца, които също са дървета

```
public class Tree<T>
{
    private T value;
    private IList<Tree<T>> children;
    ...
}
```

Стойността
на възела

Списък с възли -
деца (поддървета)

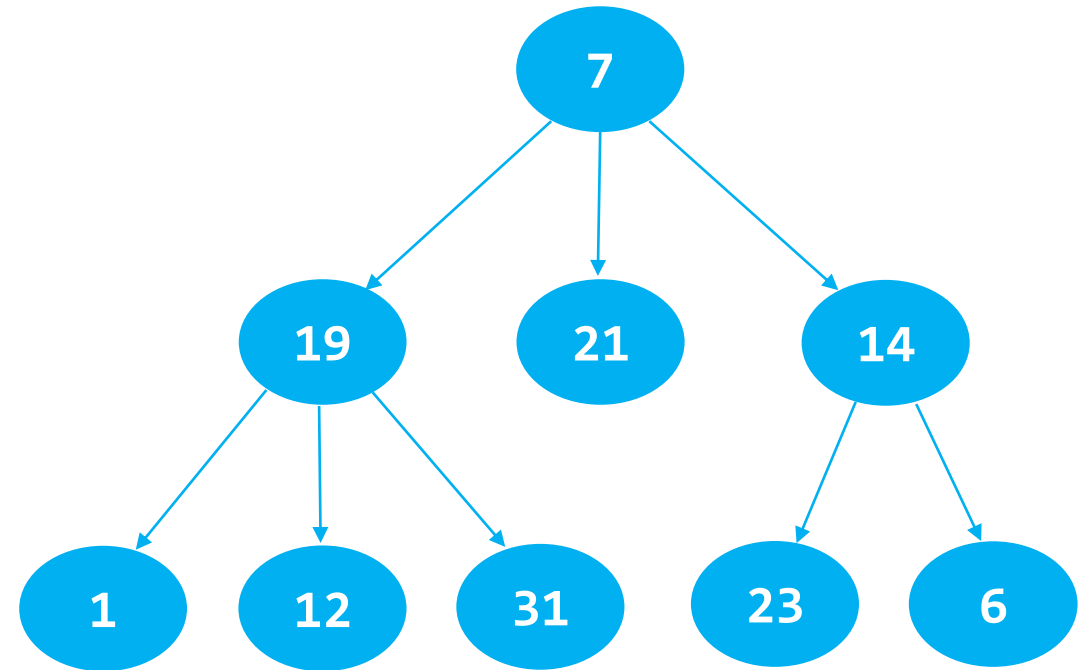
Пример за структурата Tree<T>



Задача: Реализирайте възел на дърво

Създайте рекурсивно дефинирана структура описваща дърво

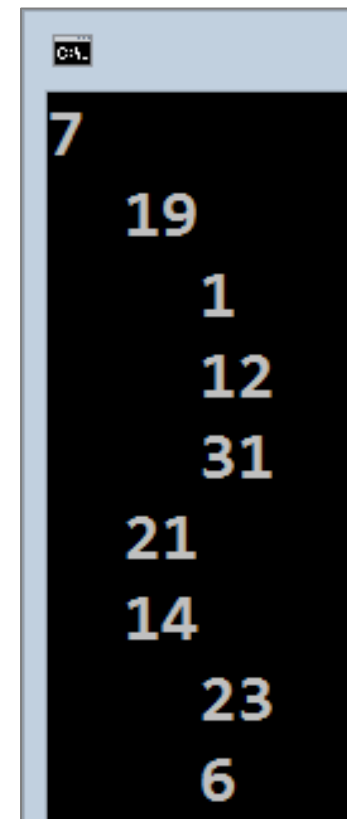
```
Tree<int> tree =  
    new Tree<int>(7,  
        new Tree<int>(19,  
            new Tree<int>(1),  
            new Tree<int>(12),  
            new Tree<int>(31)),  
        new Tree<int>(21),  
        new Tree<int>(14,  
            new Tree<int>(23),  
            new Tree<int>(6))  
    );
```



Задача: Отпечатайте елементите на дърво

Отпечатайте на конзолата елементите на дърво с 2 интервала отместване за всяко следващо ниво

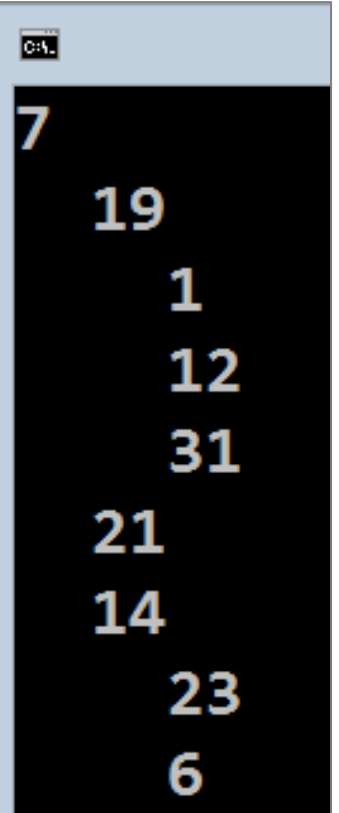
```
Tree<int> tree =  
    new Tree<int>(7,  
        new Tree<int>(19,  
            new Tree<int>(1),  
            new Tree<int>(12),  
            new Tree<int>(31)),  
        new Tree<int>(21),  
        new Tree<int>(14,  
            new Tree<int>(23),  
            new Tree<int>(6))  
    );
```



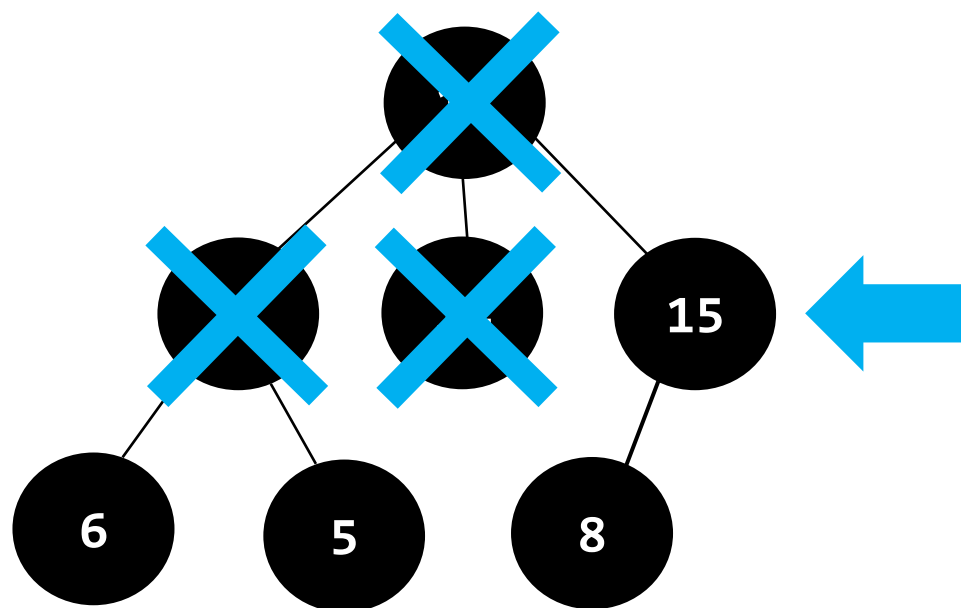
Решение: Отпечатайте елементите на дърво

Рекурсивен алгоритъм за обхождане на елементите на дърво

```
public class Tree<T>
{
    ...
    public void Print(int indent = 0)
    {
        Console.Write(new string(' ', 2 * indent));
        Console.WriteLine(this.Value);
        foreach (var child in this.Children)
            child.Print(indent + 1);
    }
}
```



```
7
  19
    1
    12
      31
  21
    14
    23
      6
```



Обхождане на дървовидни структури
Обхождане в ширина (BFS) и дълбочина (DFS)

Обхождане на дървовидни структури

- **Обхождане на дърво** представлява посещаването на всеки негов възел точно по веднъж
- **Последователността на обхождането** може да варира, в зависимост от алгоритъма за обхождане:

Обхождане в дълбочина (DFS):

- Първо се посещават наследниците на възела
- Стандартна реализация - чрез рекурсия

Обхождане в ширина (BFS):

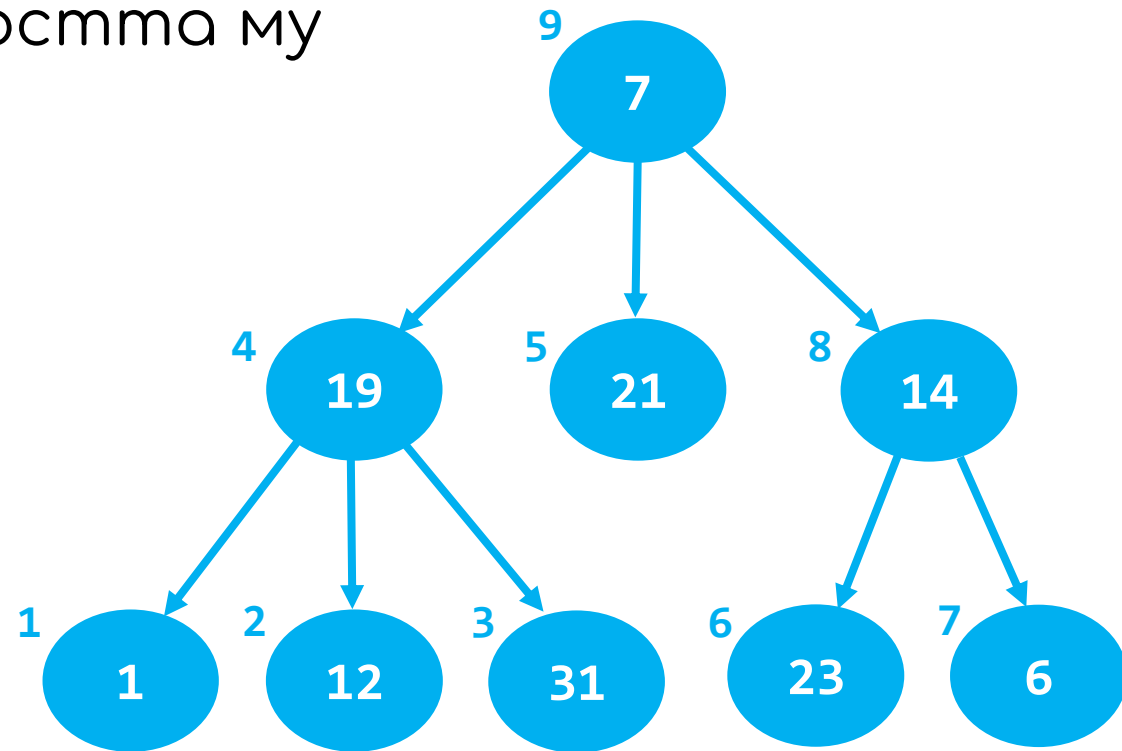
- Първо се посещава най-близкия възел
- Стандартна реализация - чрез опашка

Обхождане в дълбочина (DFS)

Обхождане в дълбочина (DFS) - за всеки възел:

- Посещават се всички негови деца
- Ако възела няма деца или всички негови деца са вече обходени се обработва стойността му

```
DFS (node)
{
  for each child c of node
    DFS(c);
  print node;
}
```



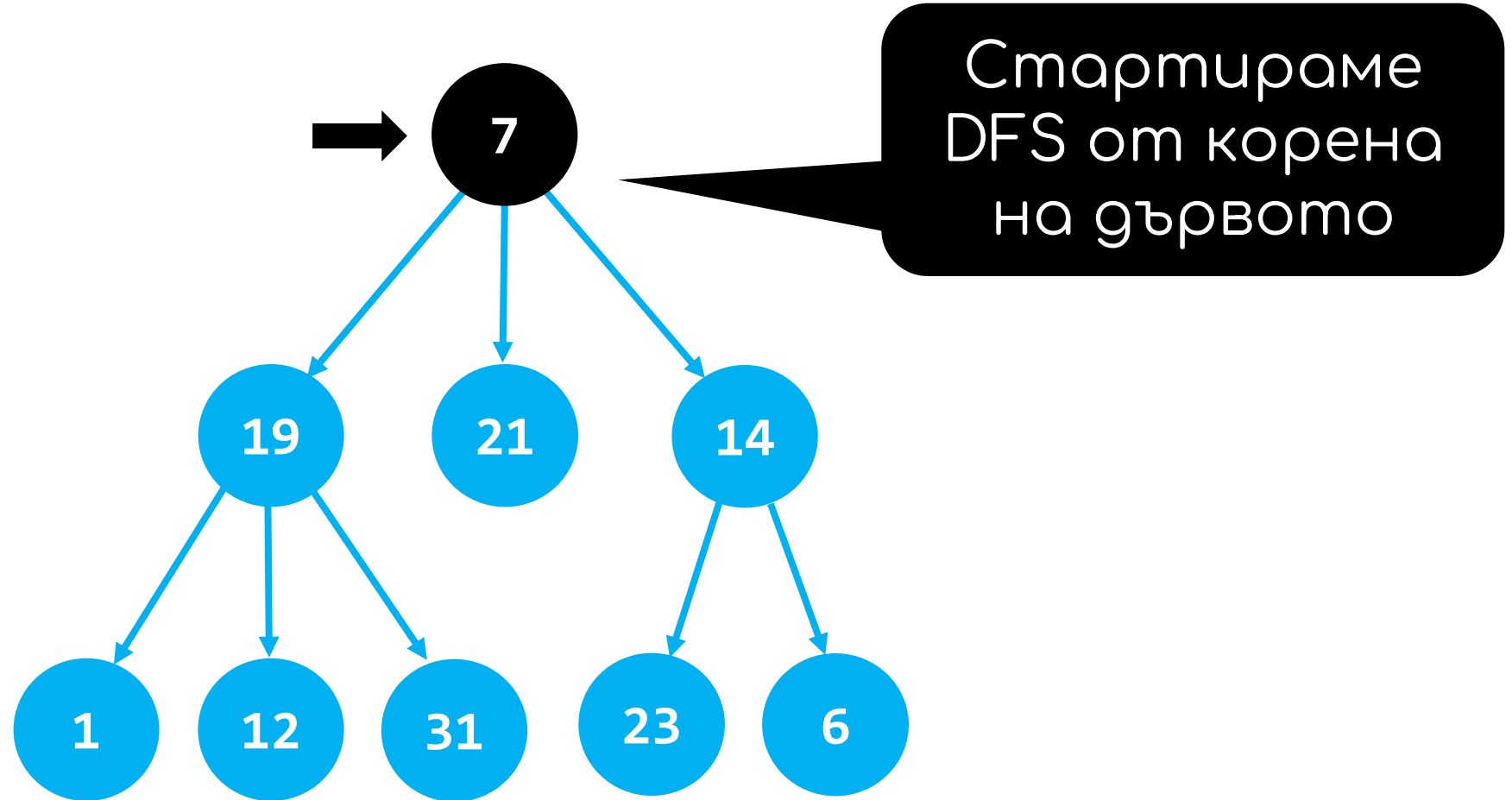
DFS в действие [1/18]

Стек:

7

Изход:

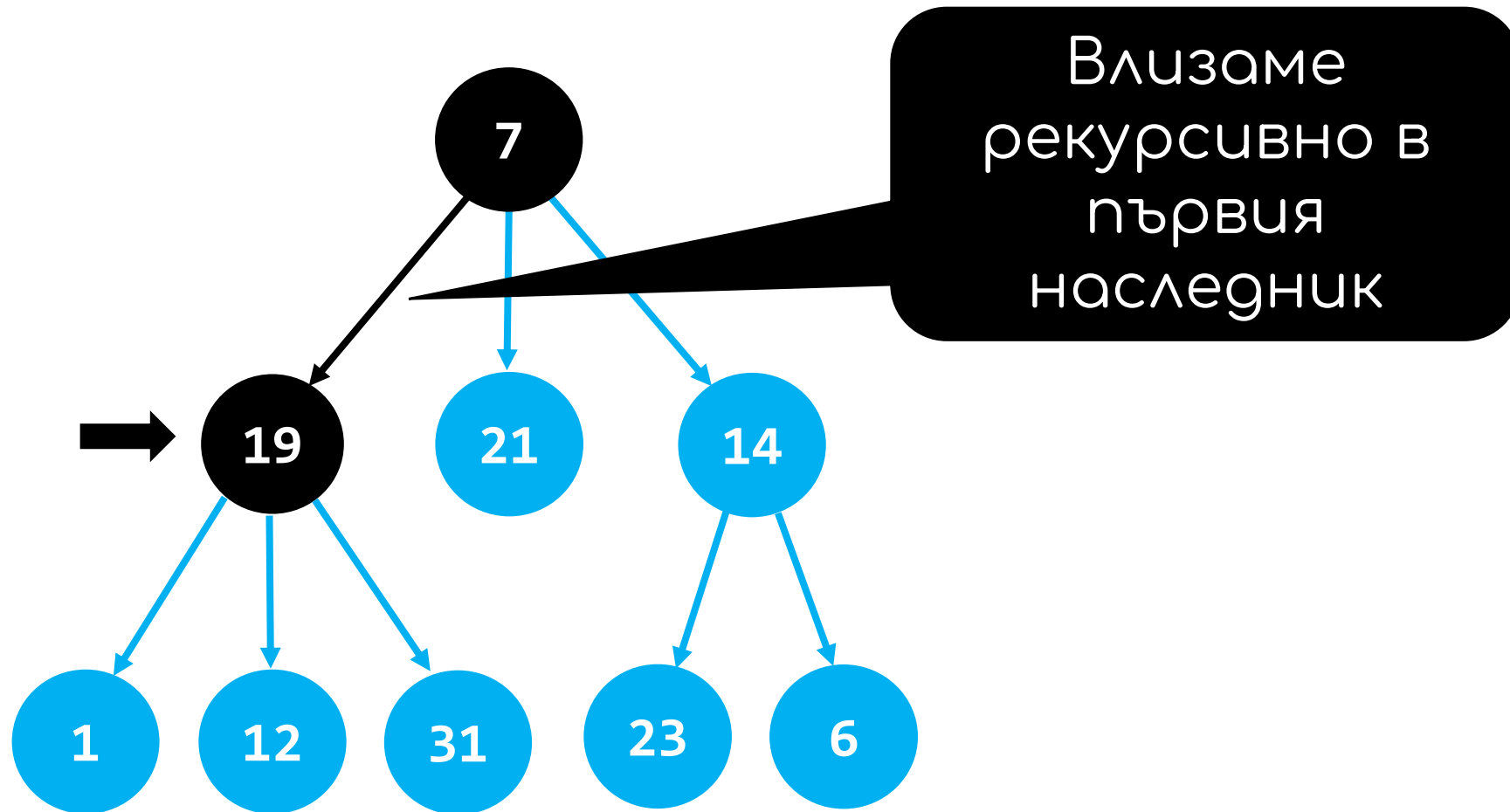
(празен)



DFS в действие [2/18]

Стек:
7, 19

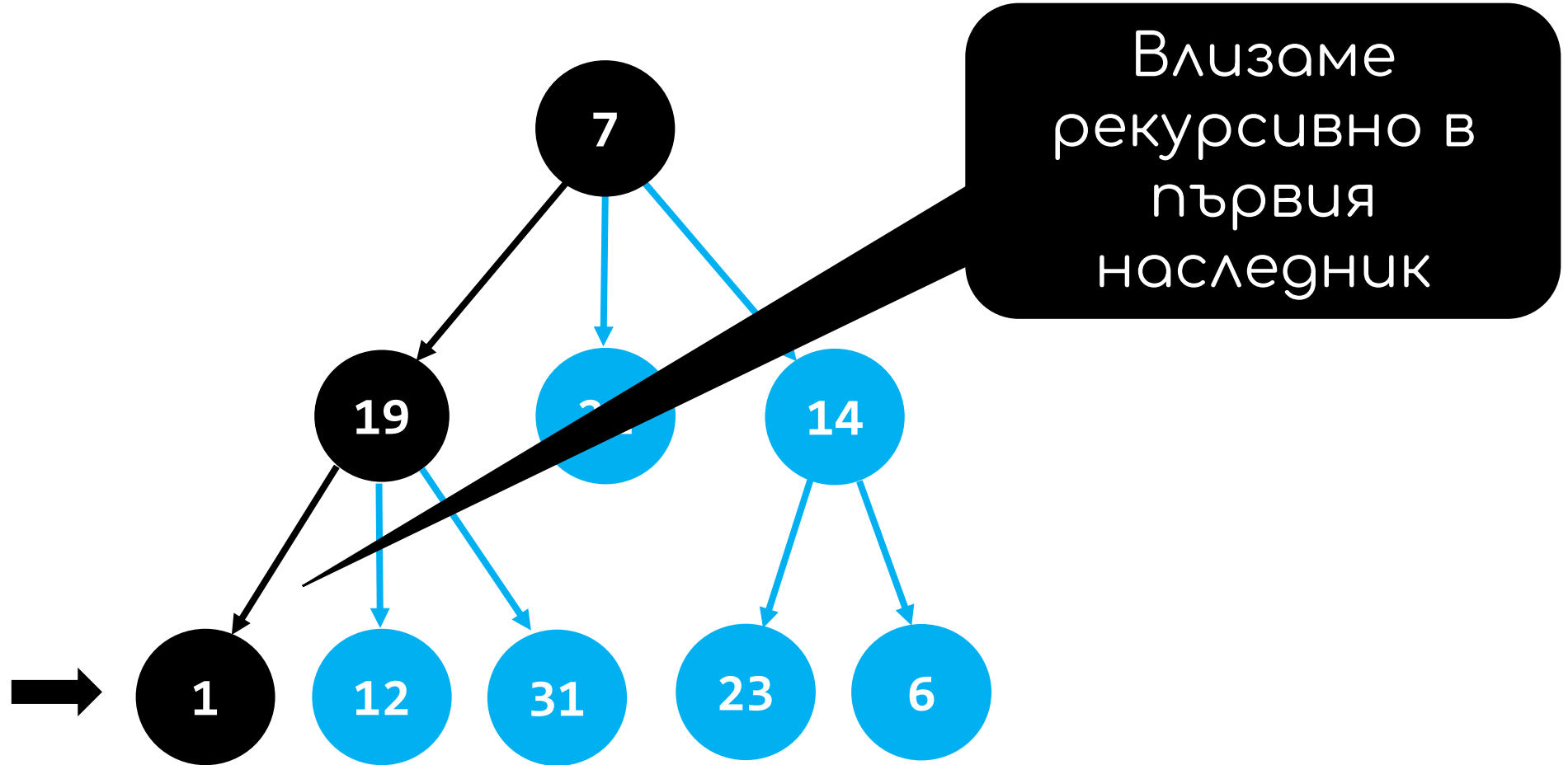
Изход:
(празен)



DFS в действие [3/18]

Стек:
7, 19, 1

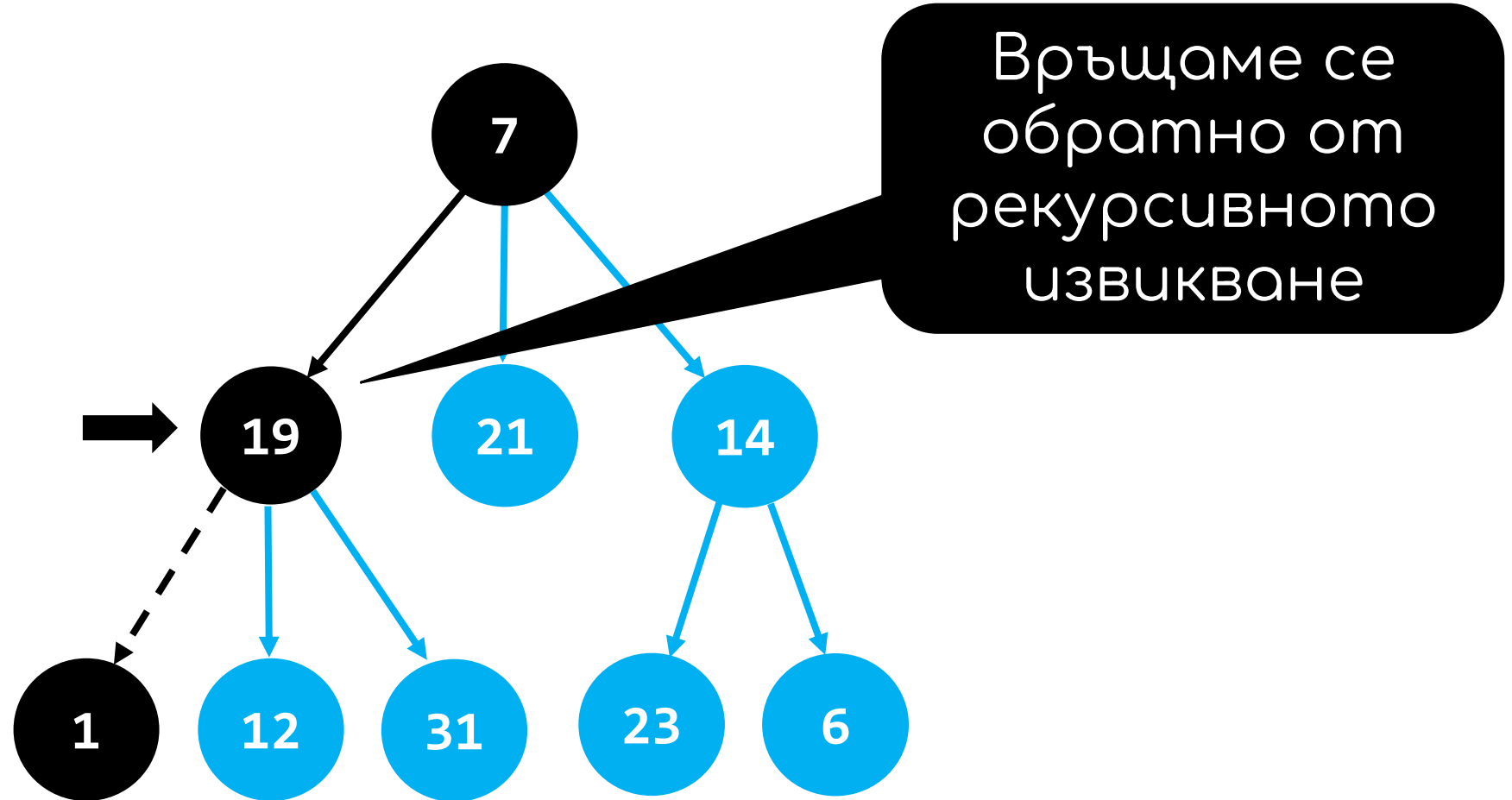
Изход:
(празен)



DFS в действие [4/18]

Стек:
7, 19

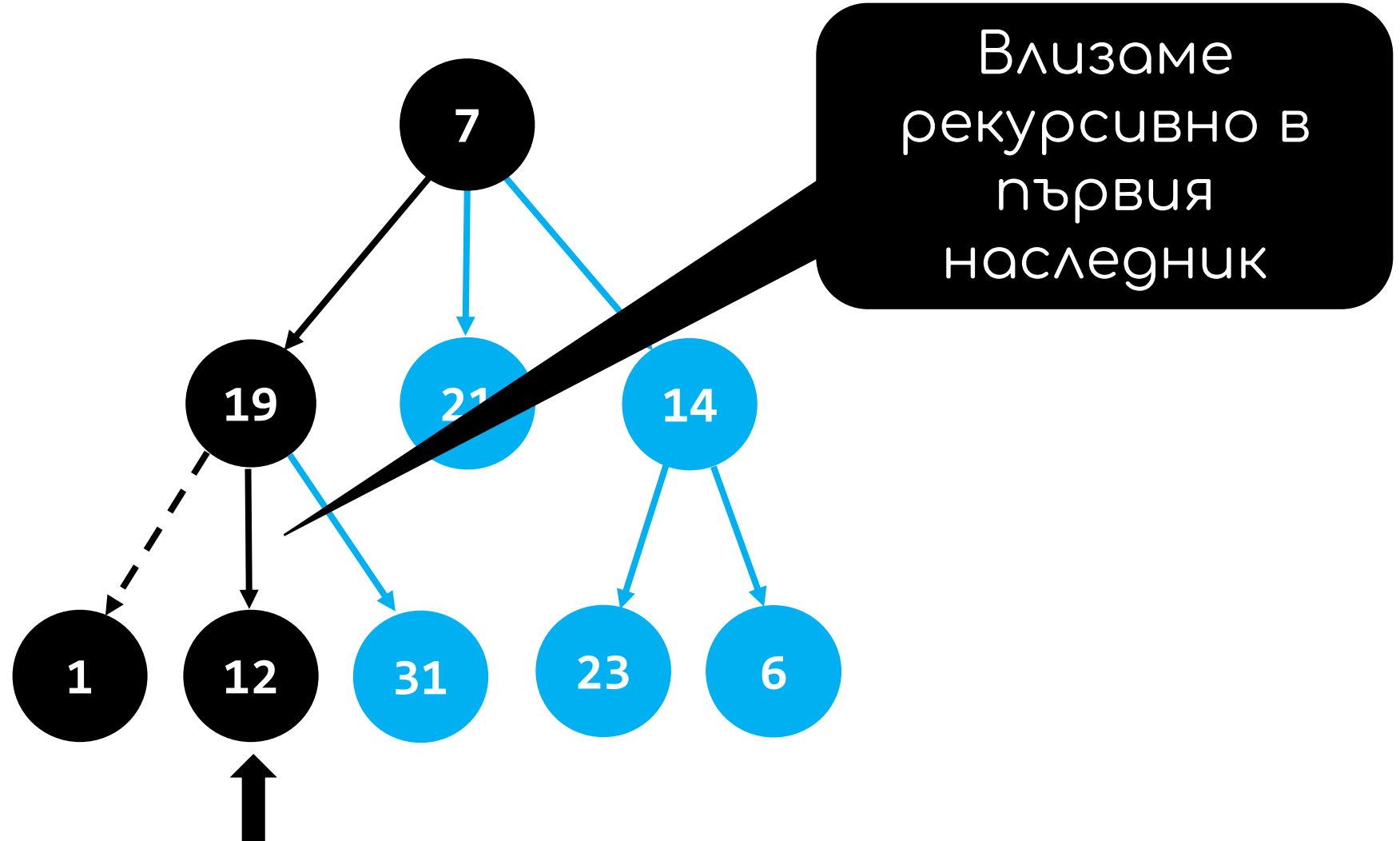
Изход:
1



DFS в действие [5/18]

Стек:
7, 19, 12

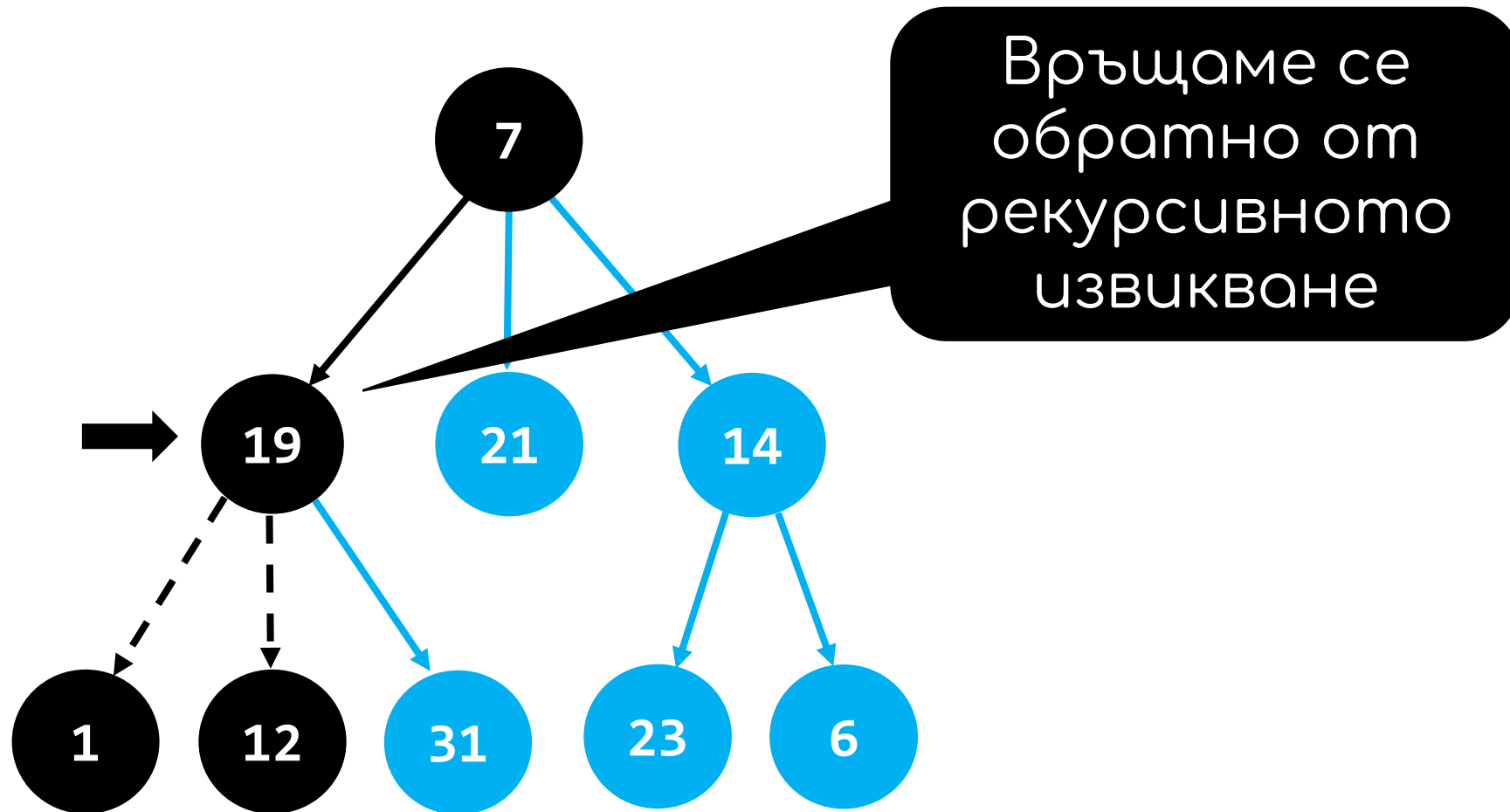
Изход:
1



DFS в действие [6/18]

Стек:
7, 19

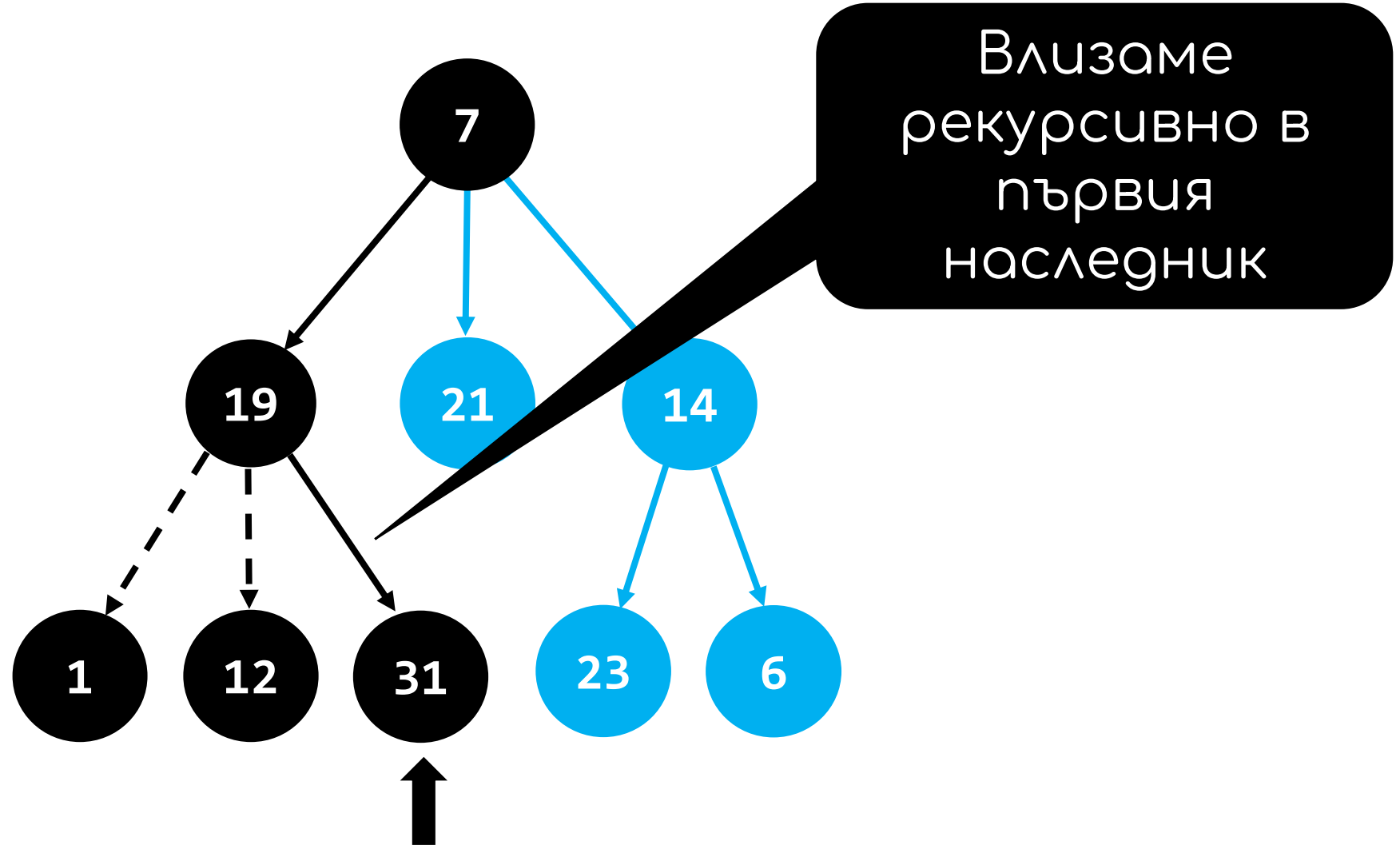
Изход:
1, 12



DFS в действии [7/18]

Стек:
7, 19, 31

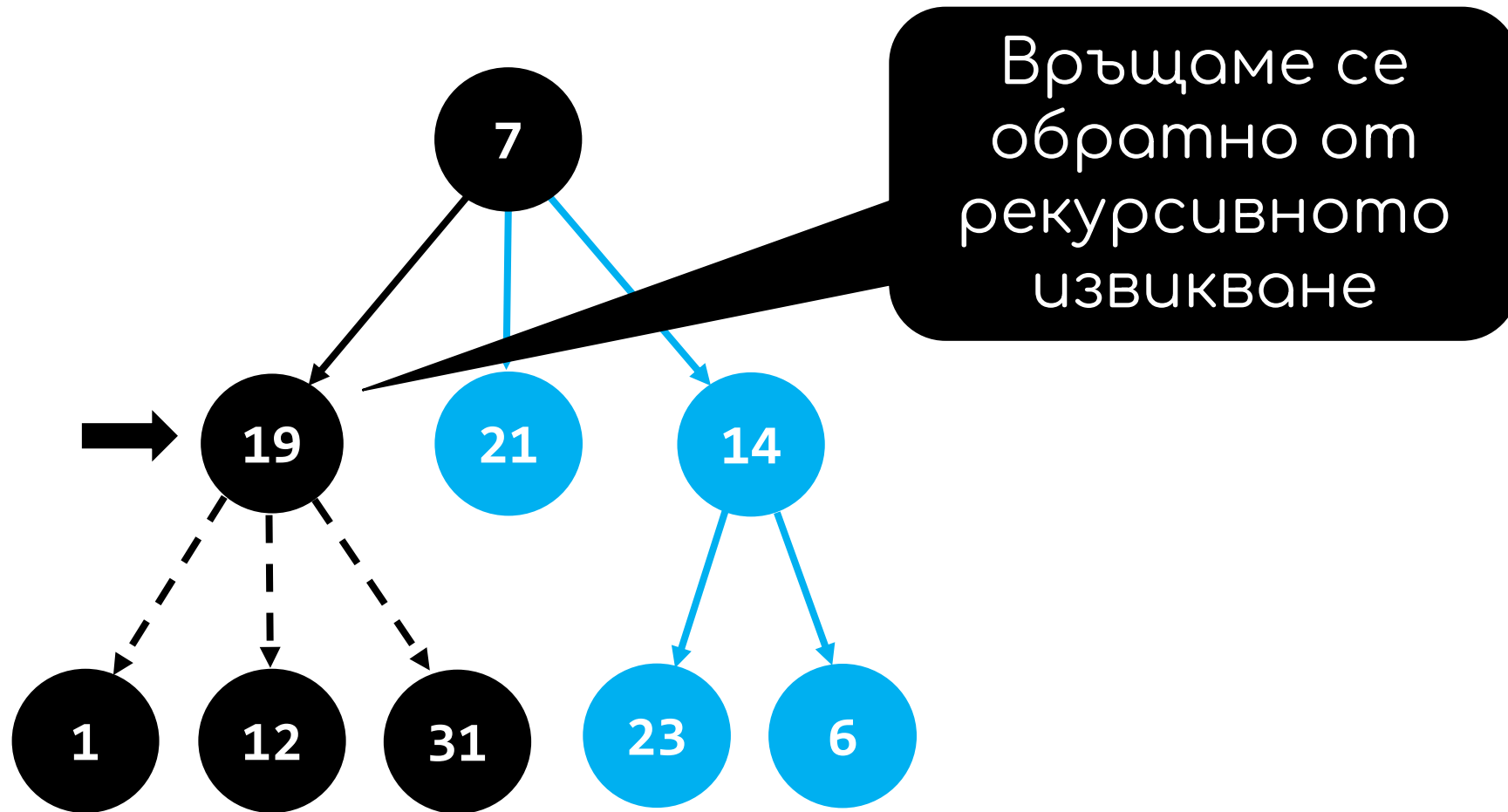
Исход:
1, 12



DFS в действие [8/18]

Стек:
7, 19

Изход:
1, 12, 31



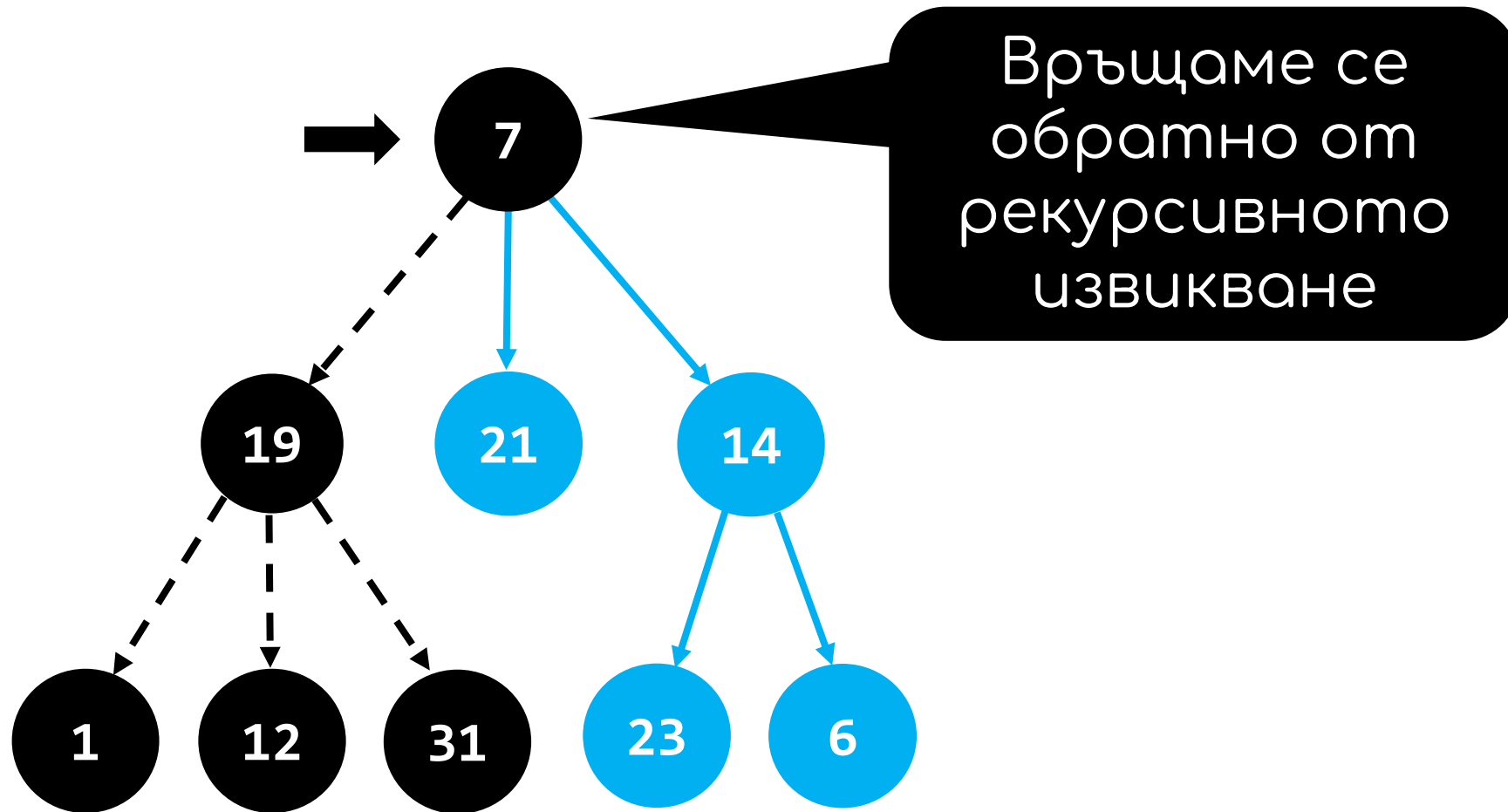
DFS в действие [9/18]

Стек:

7

Изход:

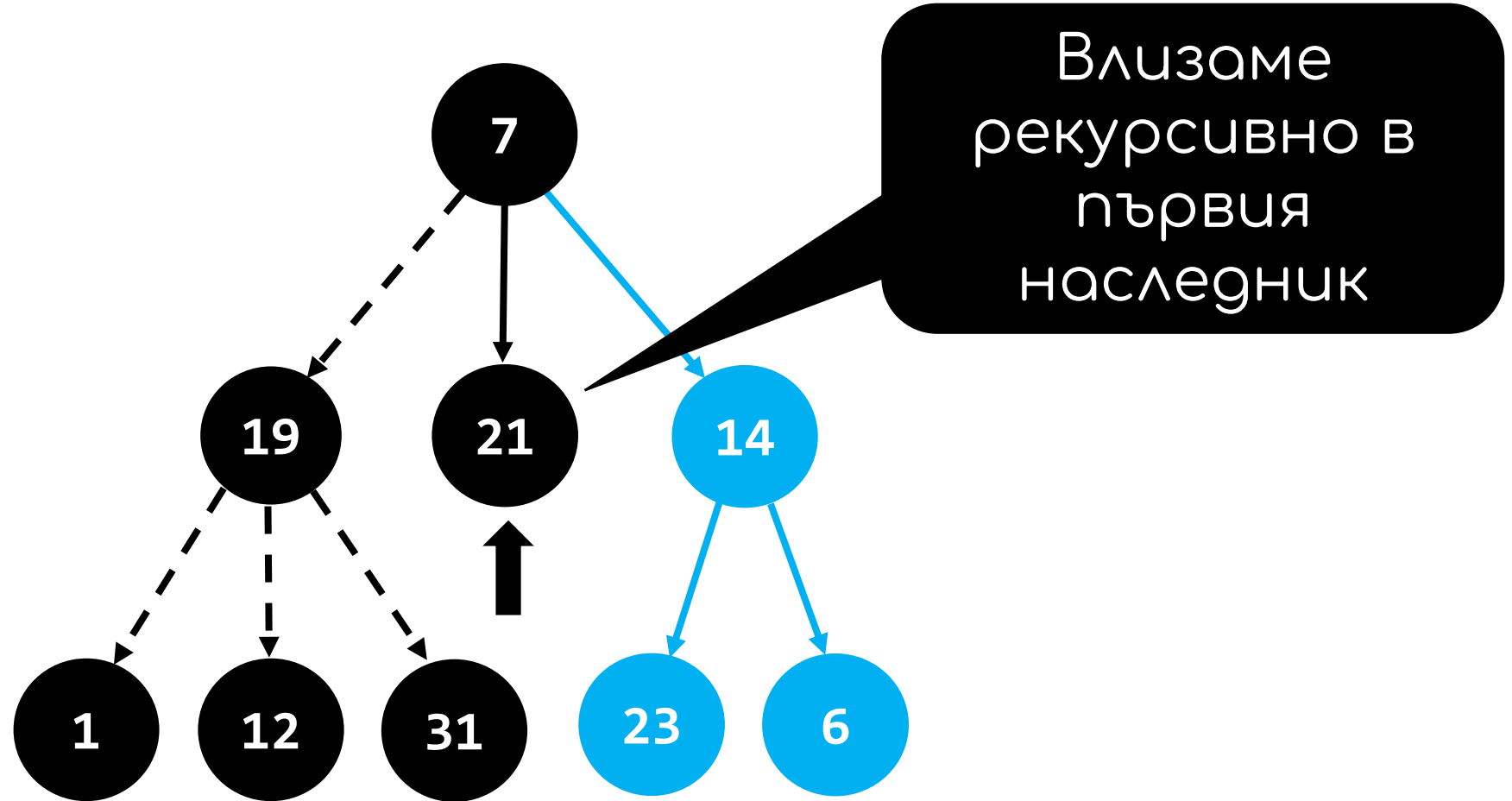
1, 12, 31, 19



DFS в действие [10/18]

Стек:
7, 21

Изход:
1, 12, 31, 19



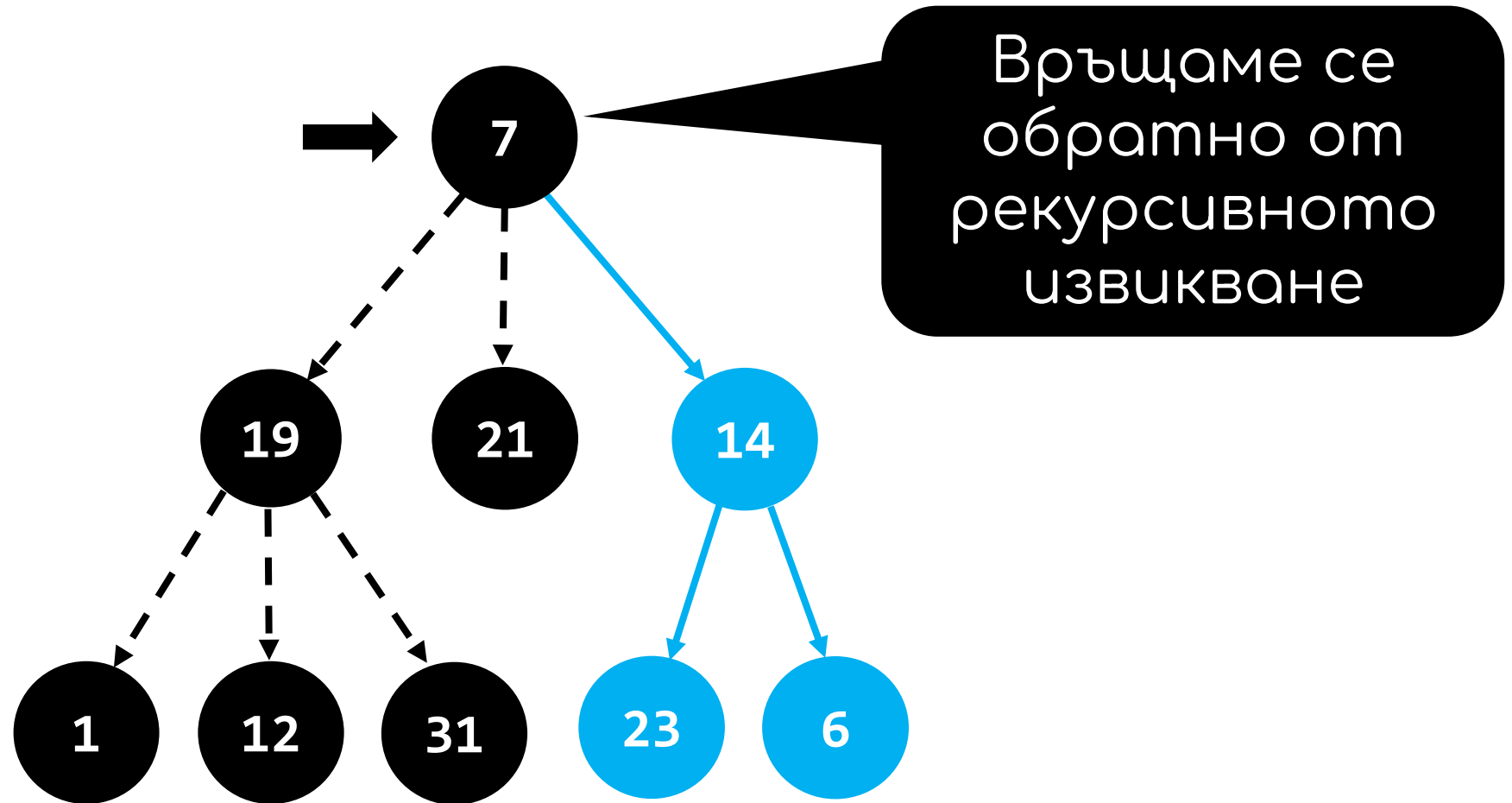
DFS в действие [11/18]

Стек:

7

Изход:

1, 12, 31, 19, 21



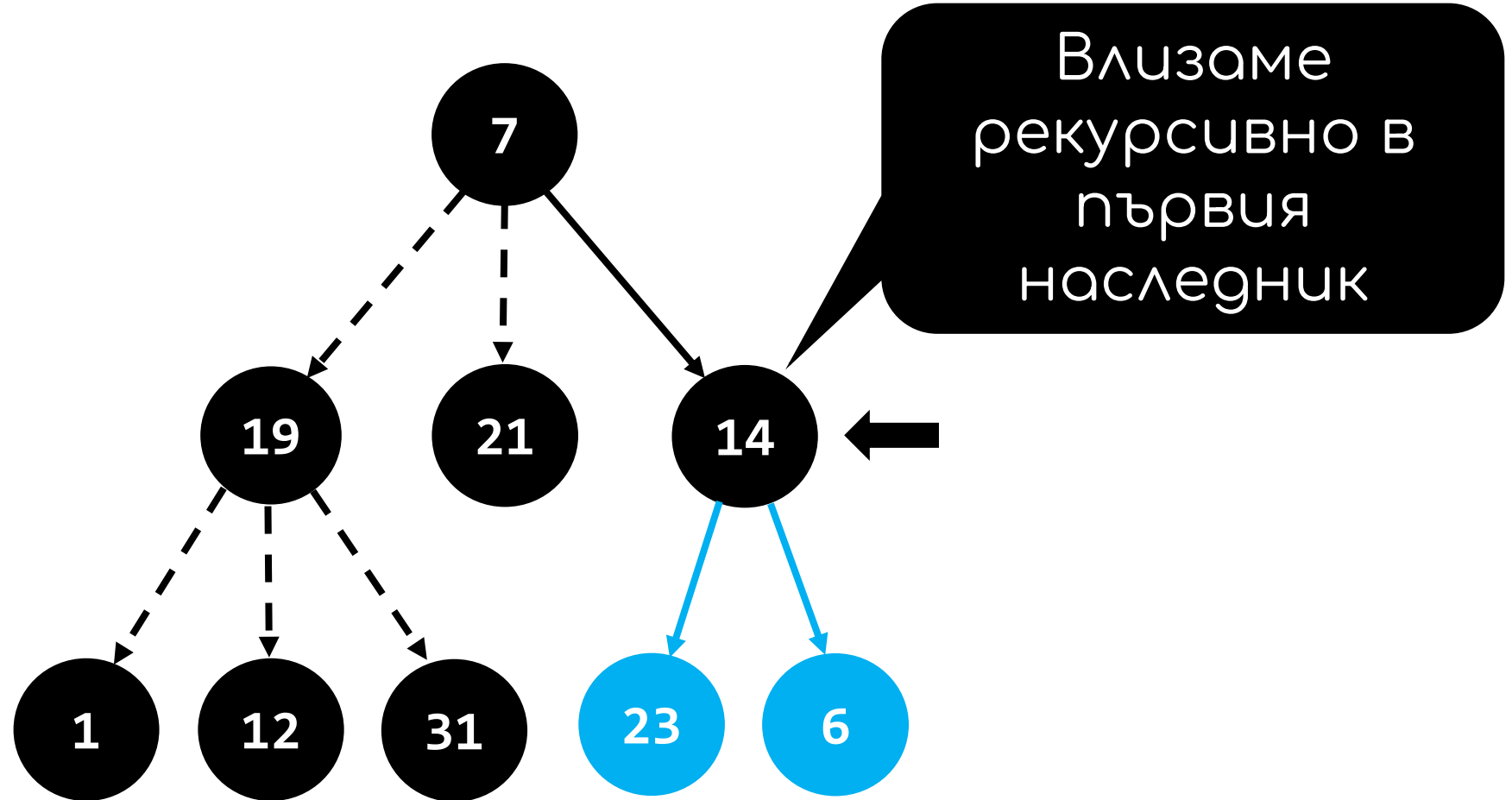
DFS в действие [12/18]

Стек:

7, 14

Изход:

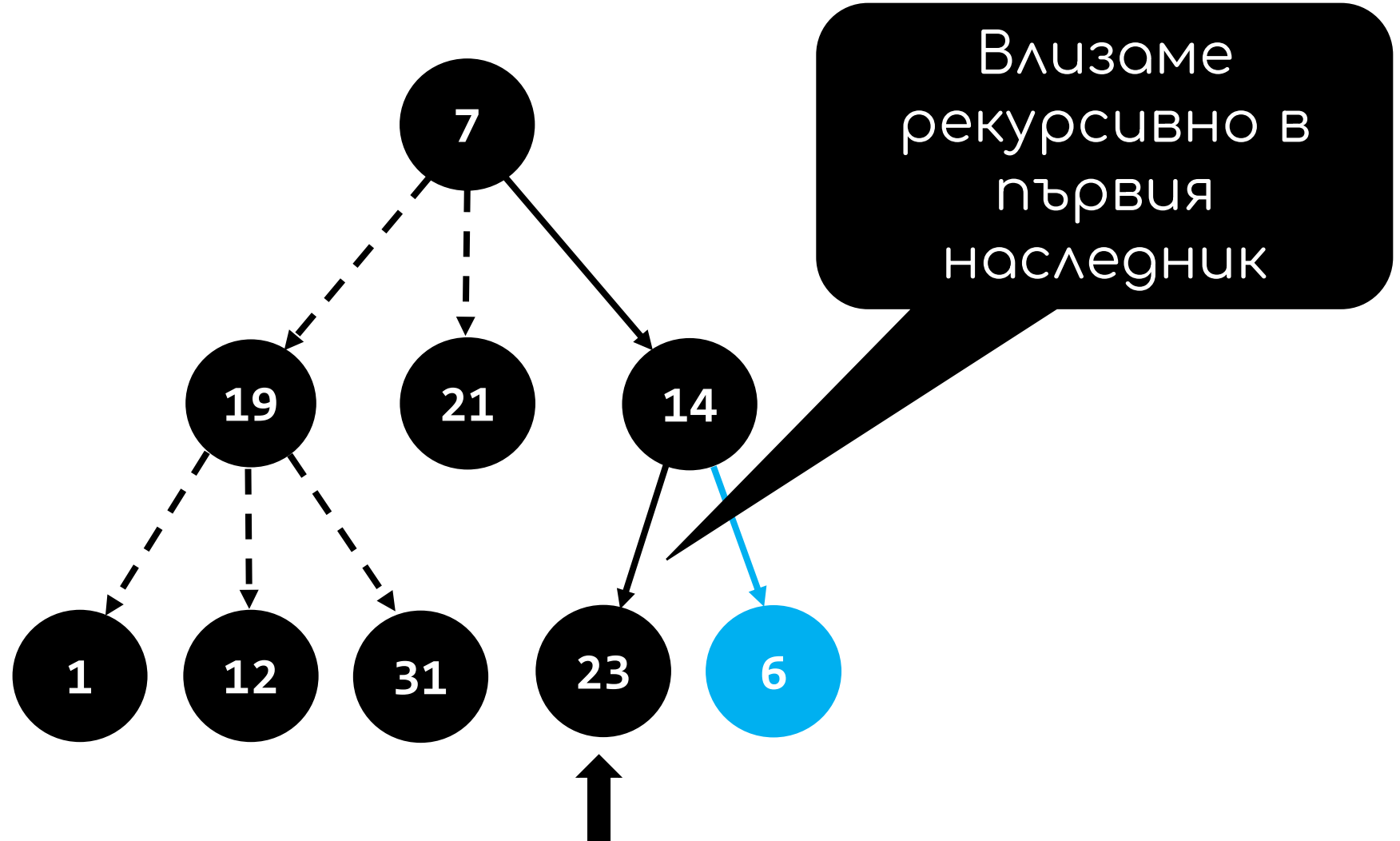
1, 12, 31, 19, 21



DFS в действие [13/18]

Стек:
7, 14, 23

Изход:
1, 12, 31, 19, 21



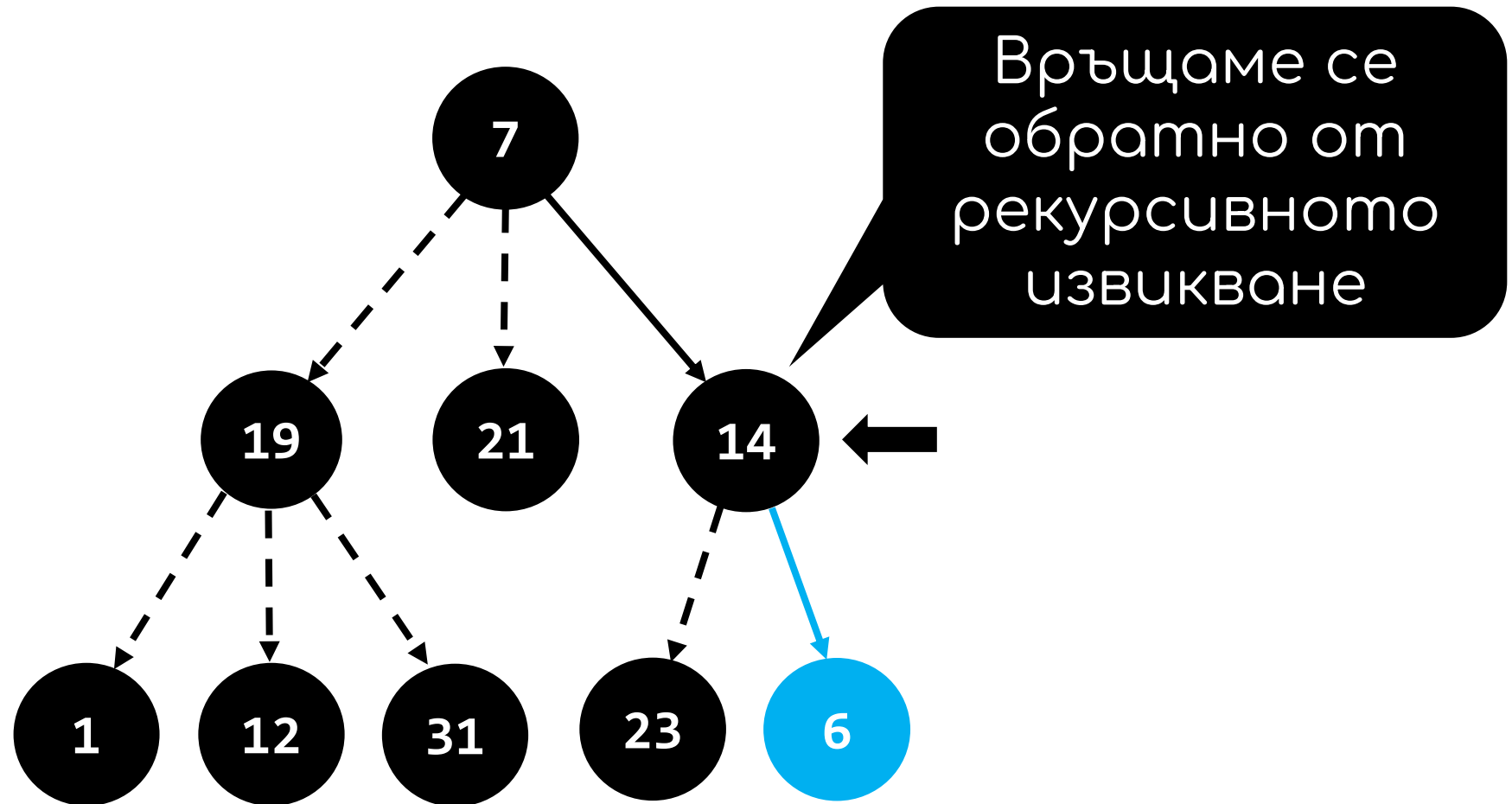
DFS в действие [14/18]

Стек:

7, 14

Изход:

1, 12, 31, 19, 21, 23



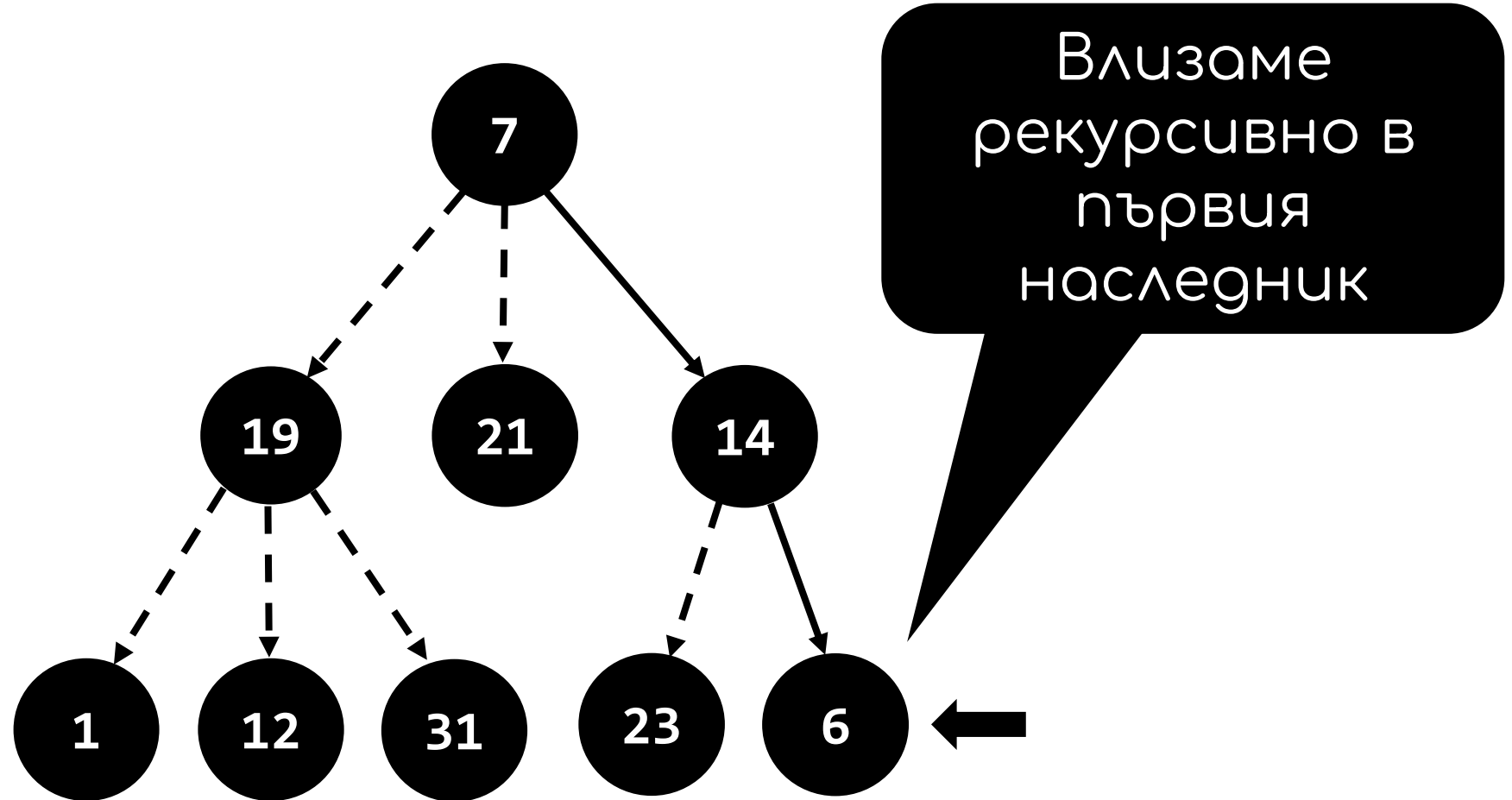
DFS в действие [15/18]

Стек:

7, 14, 6

Изход:

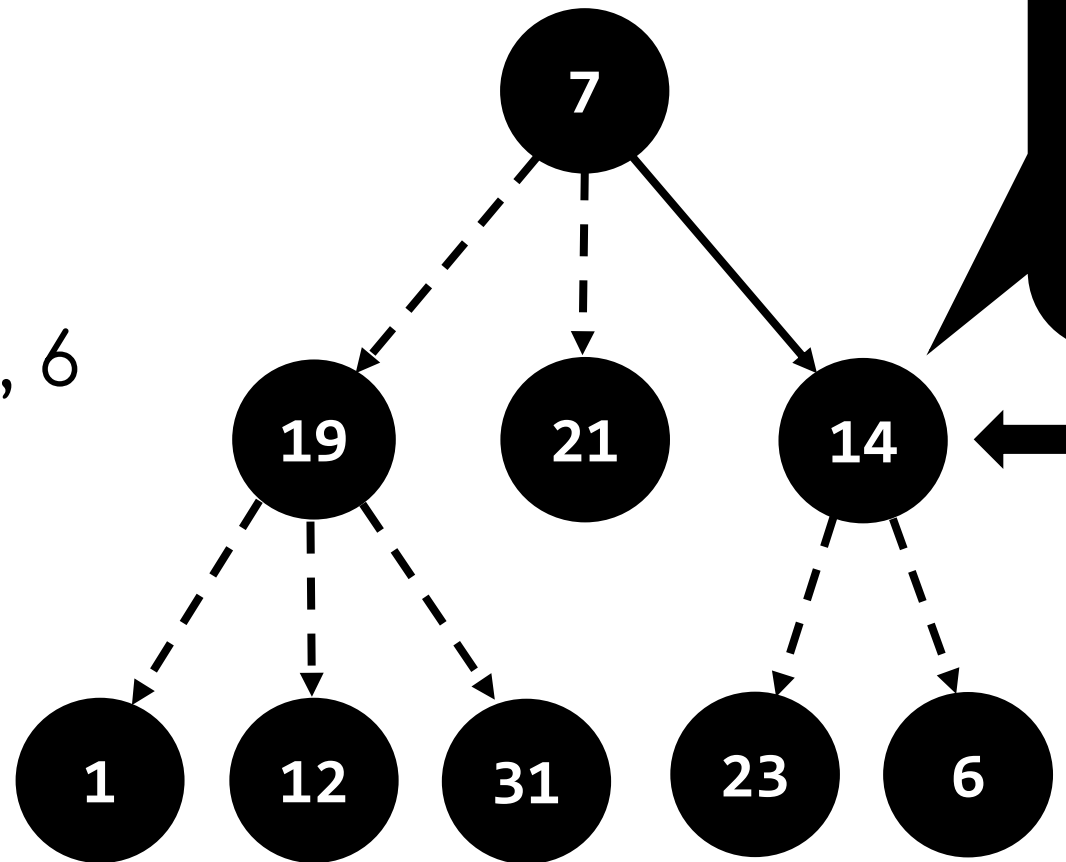
1, 12, 31, 19, 21, 23



DFS в действие [16/18]

Стек:
7, 14

Изход:
1, 12, 31, 19, 21, 23, 6



Връщаме се обратно
от рекурсивното
извикване и
принтираме
стойността на
последно посещения
възел

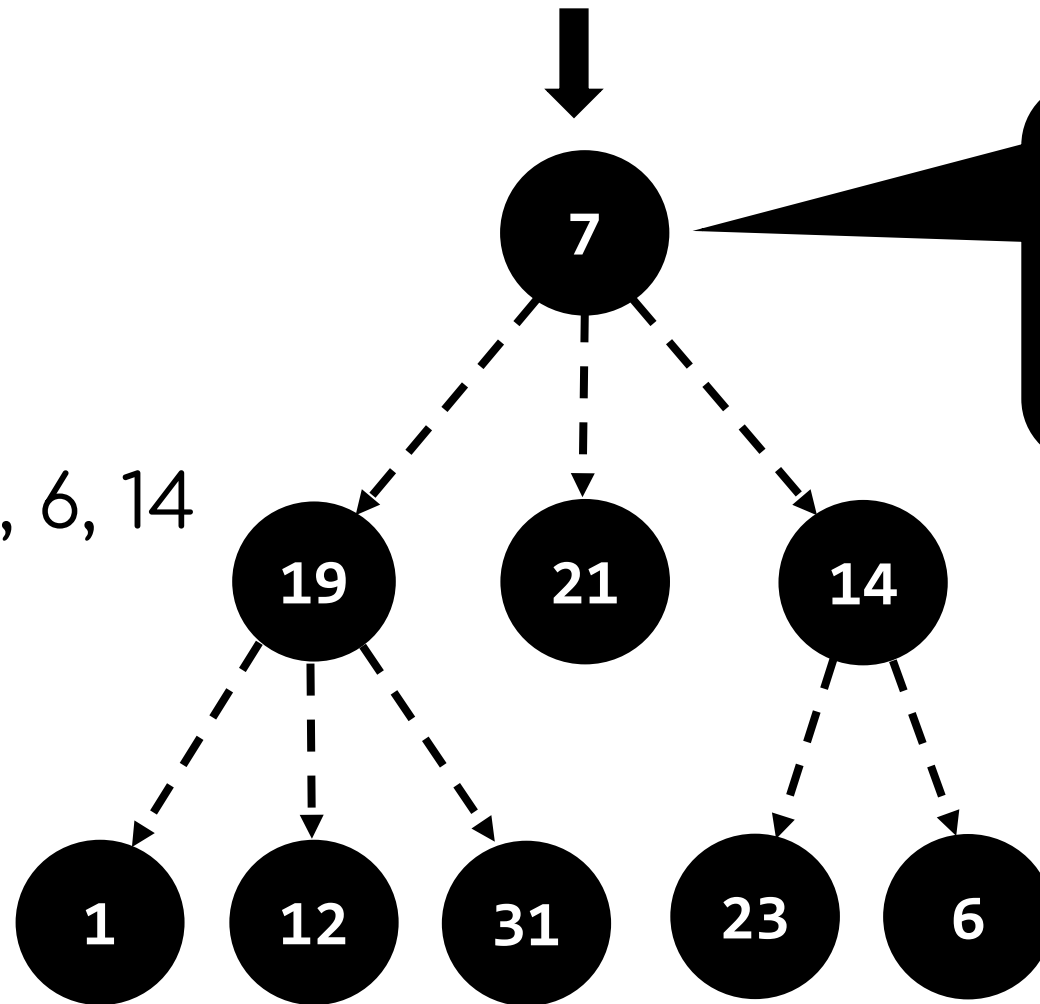
DFS в действие [17/18]

Стек:

7

Изход:

1, 12, 31, 19, 21, 23, 6, 14



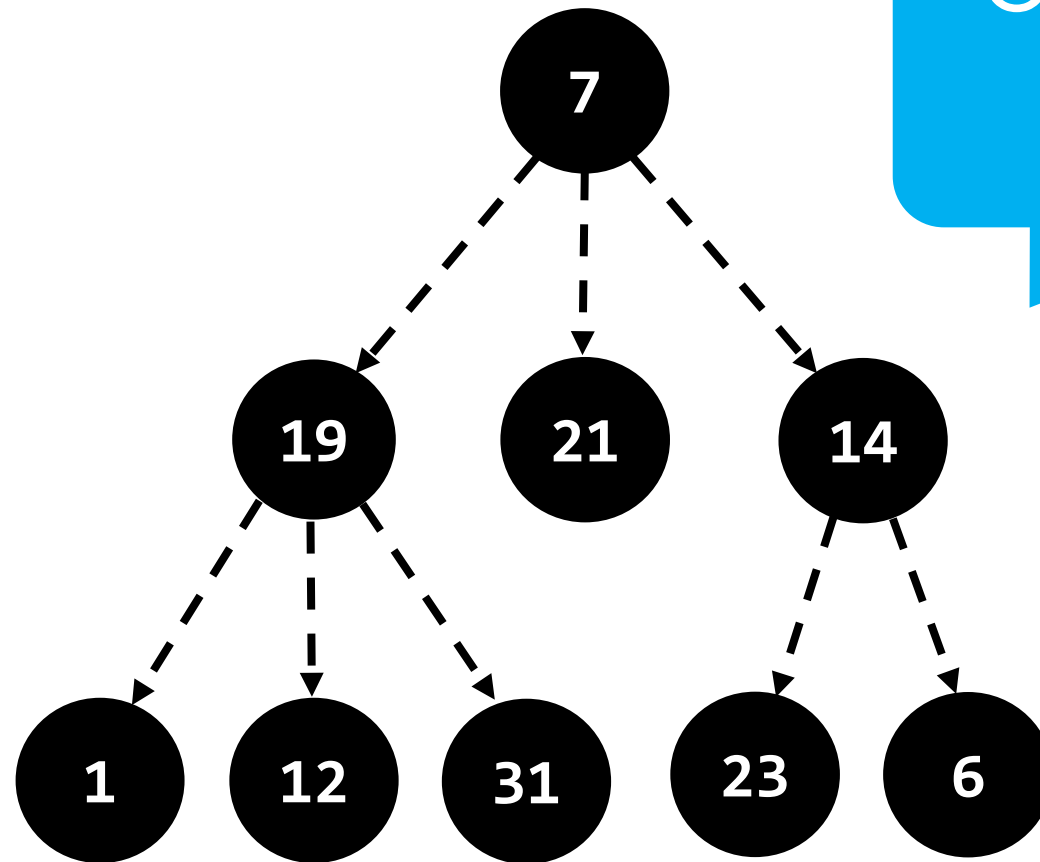
Връщаме се
обратно от
рекурсивното
извикване

DFS в действие [18/18]

Стек:
(празен)

Изход:

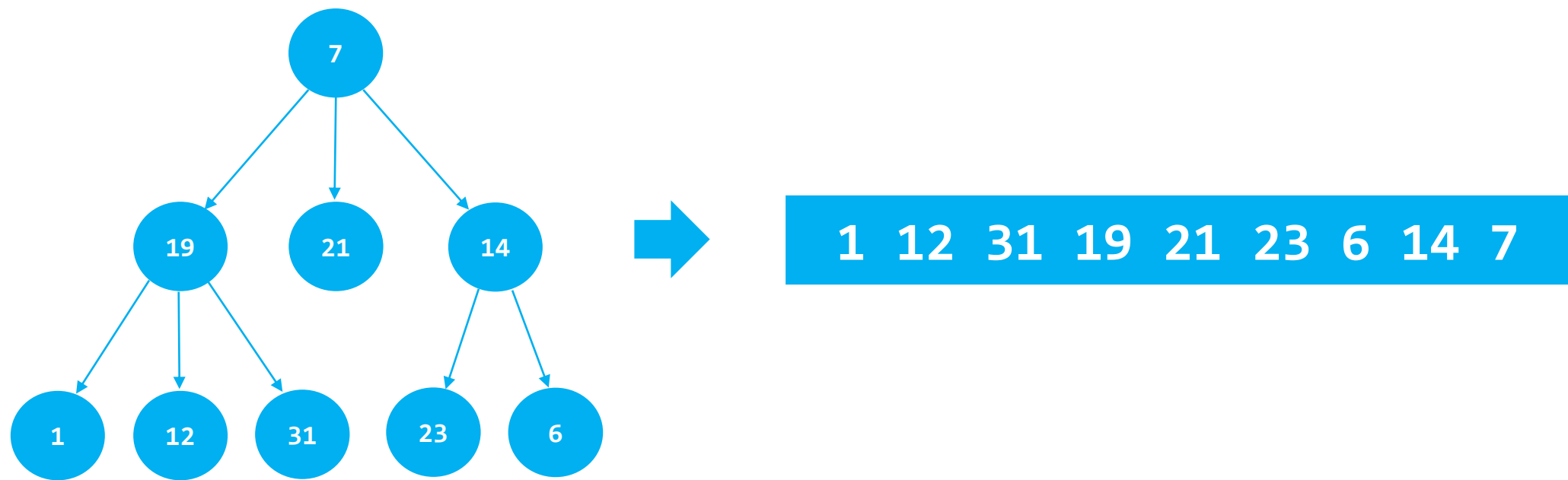
1, 12, 31, 19, 21,
23, 6, 14, 7



Обхождането в
дълбочина е
приключено

Задача: Извличане на елементи от дърво (DFS)

- Обходете дърво от тип `Tree<T>`, като дефинирате:
 - `IEnumerable<T> OrderDFS()`, който връща елементите на дървото по поредността на обхождане с DFS



Задача: Извличане на елементи от дърво (DFS)

```
public IEnumerable<T> OrderDFS()
{
    List<T> order = new List<T>();
    this.DFS(this, order);
    return order;
}

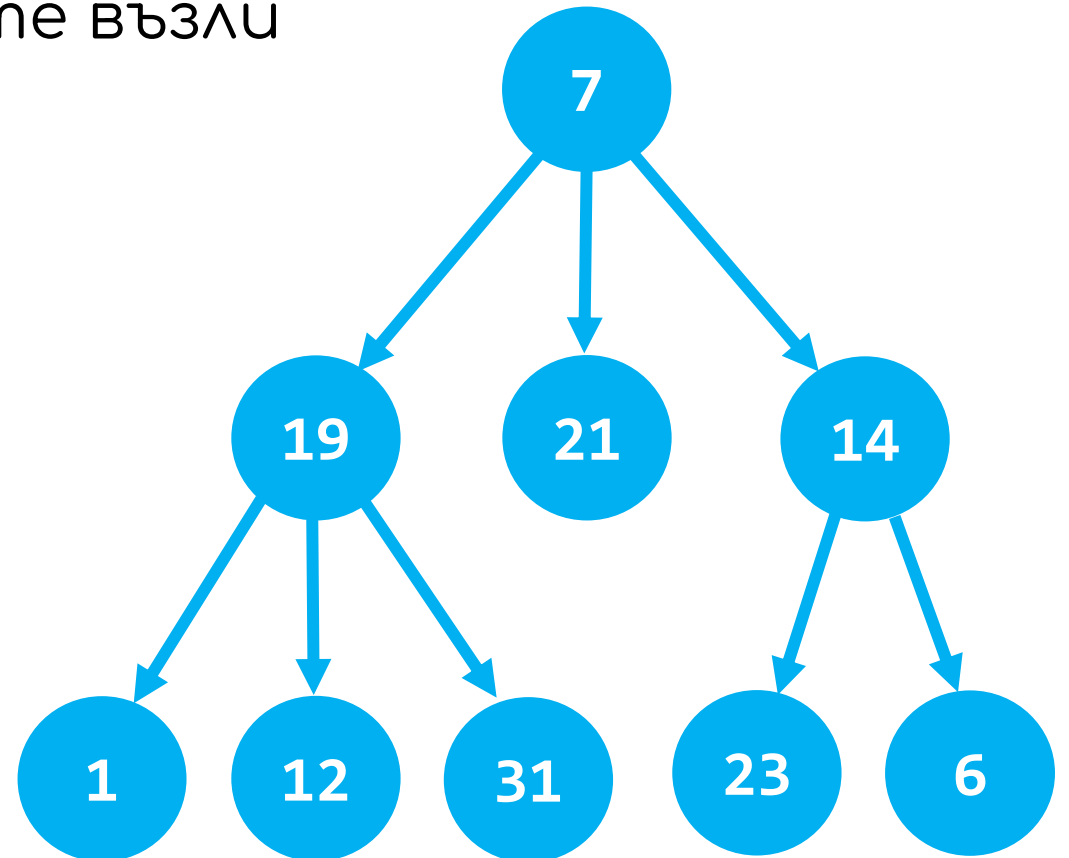
private void DFS(Tree<T> tree, List<T> order)
{
    foreach (var child in tree.Children)
        this.DFS(child, order);

    order.Add(tree.Value);
}
```

Обхождане в ширина (BFS)

- Обхождане в ширина (BFS) -
 - Обработва се стойността на възела
 - Посещават се всички съседните възли

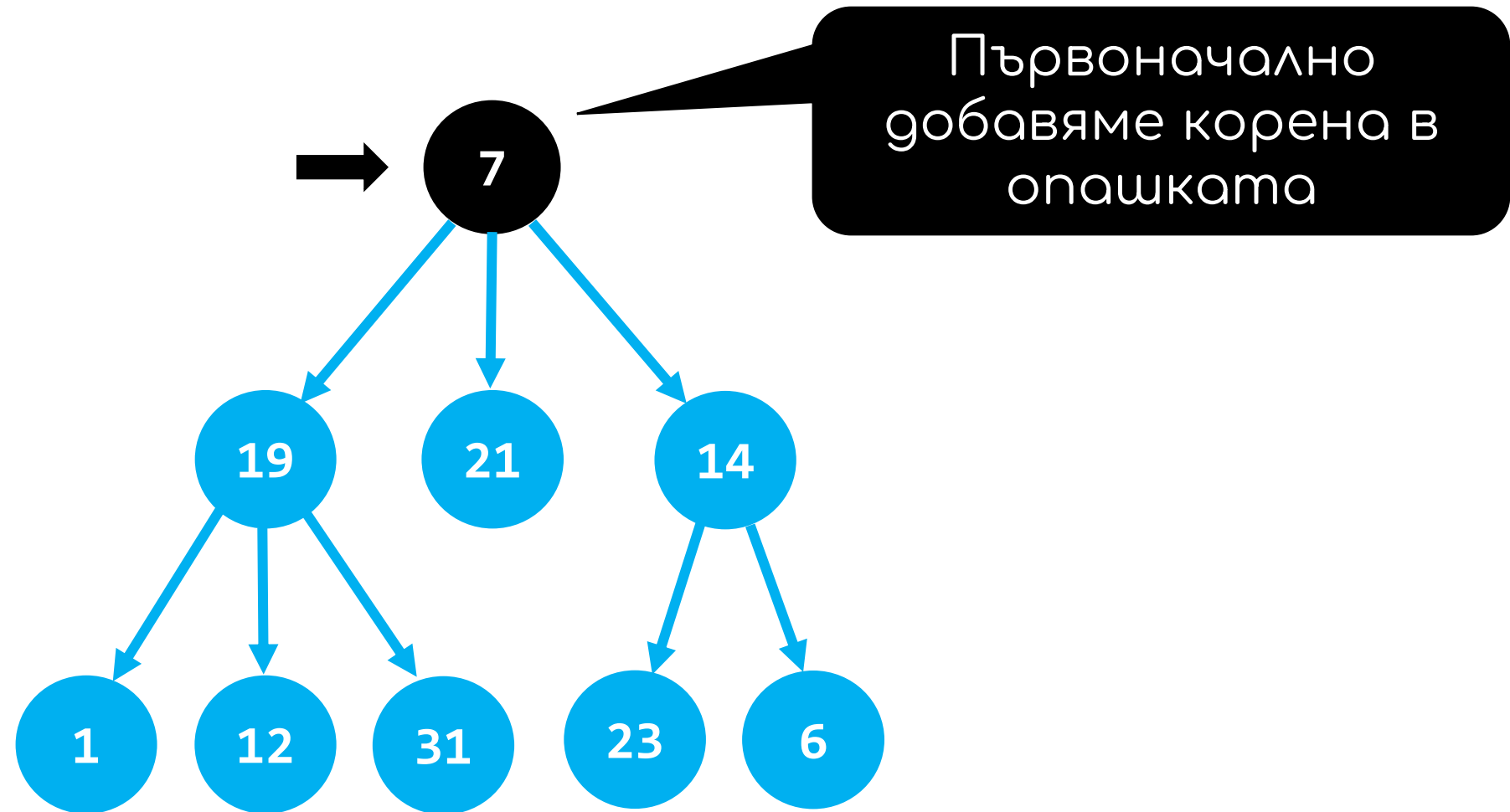
```
BFS (node)
{
    queue -> node
    while queue not empty
        v -> queue
        print v
        for each child c of v
            queue -> c
}
```



BFS в действие [1/19]

Опашка:
7

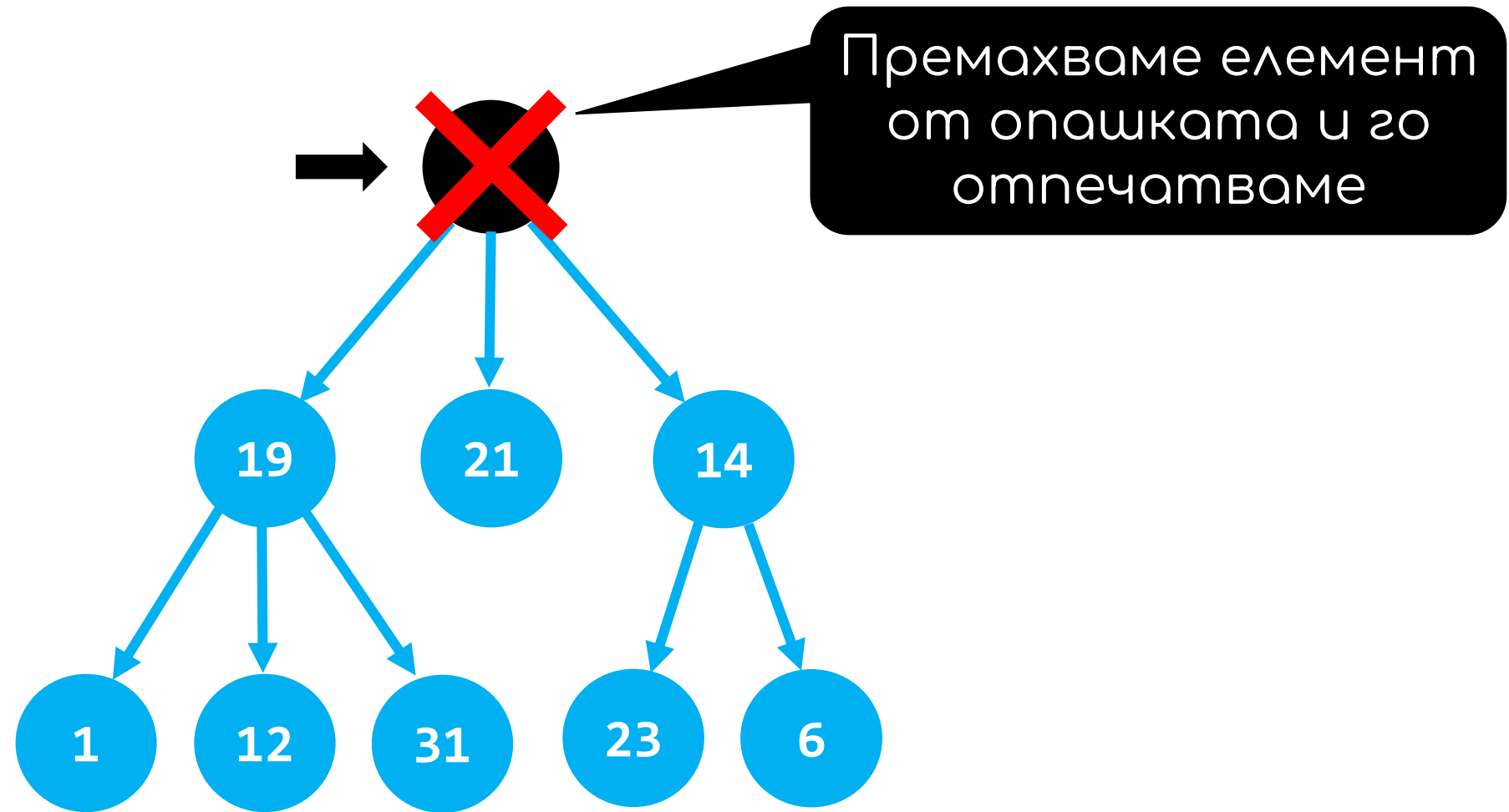
Изход:
(празен)



BFS в действие [2/19]

Опашка:
(празен)

Изход:
7



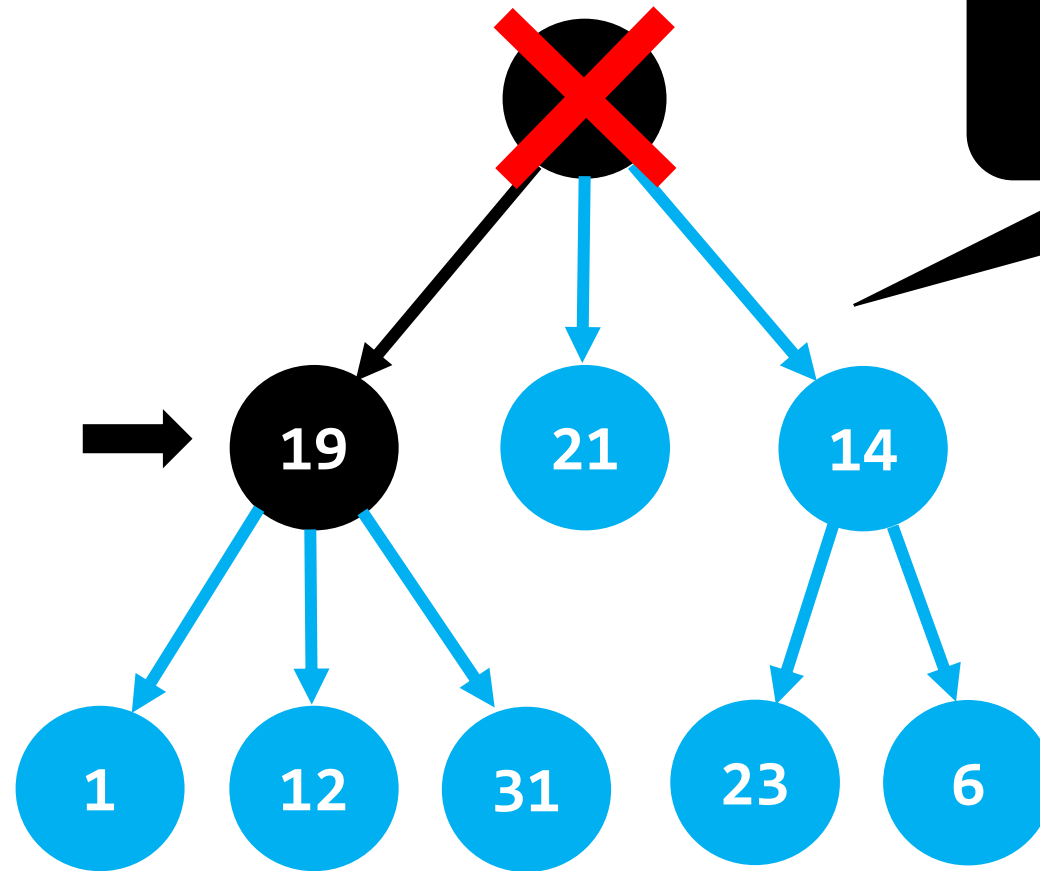
BFS в действие [3/19]

Опашка:

19

Изход:

7



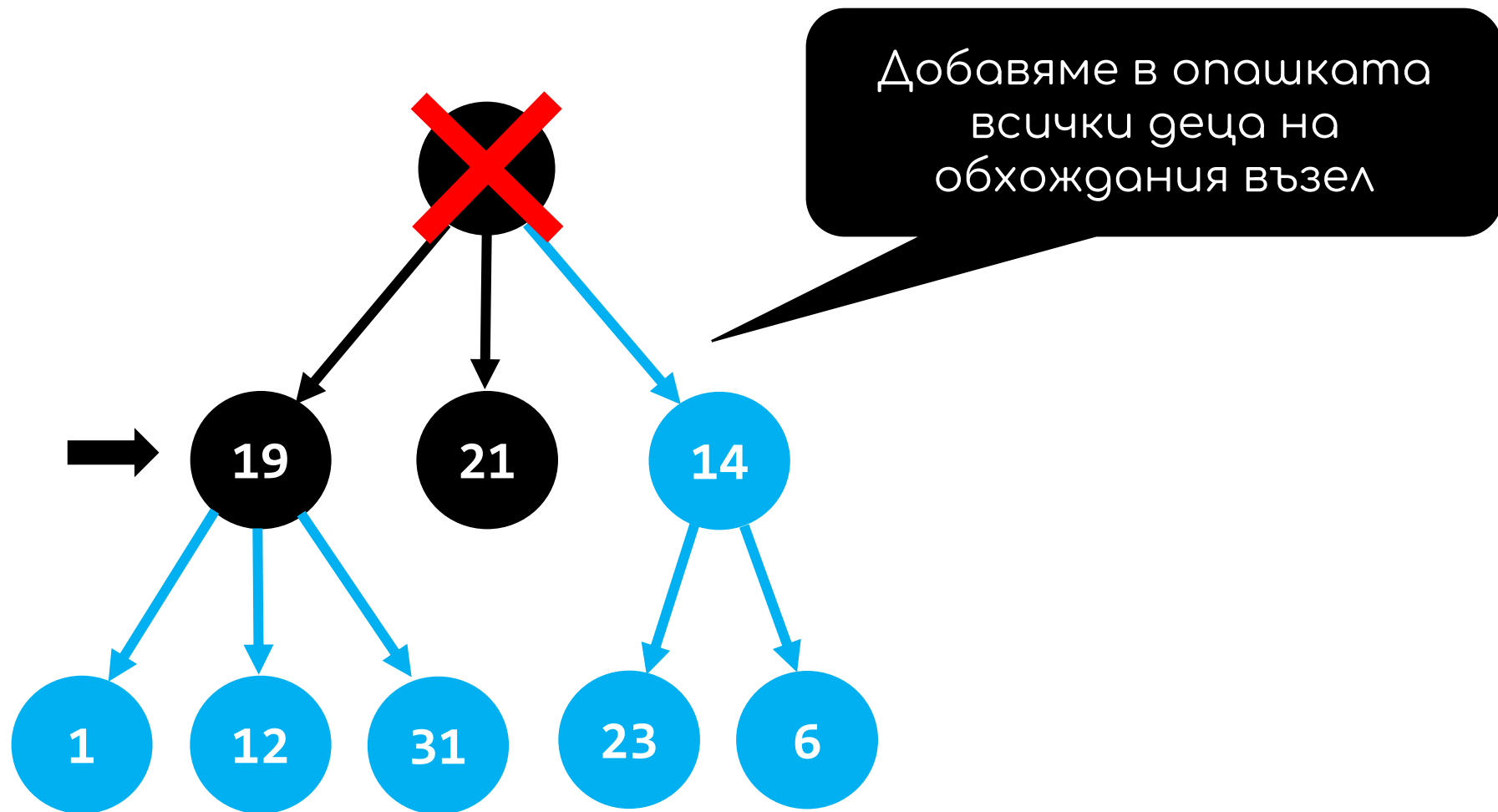
BFS в действие [4/19]

Опашка:

19, 21

Изход:

7



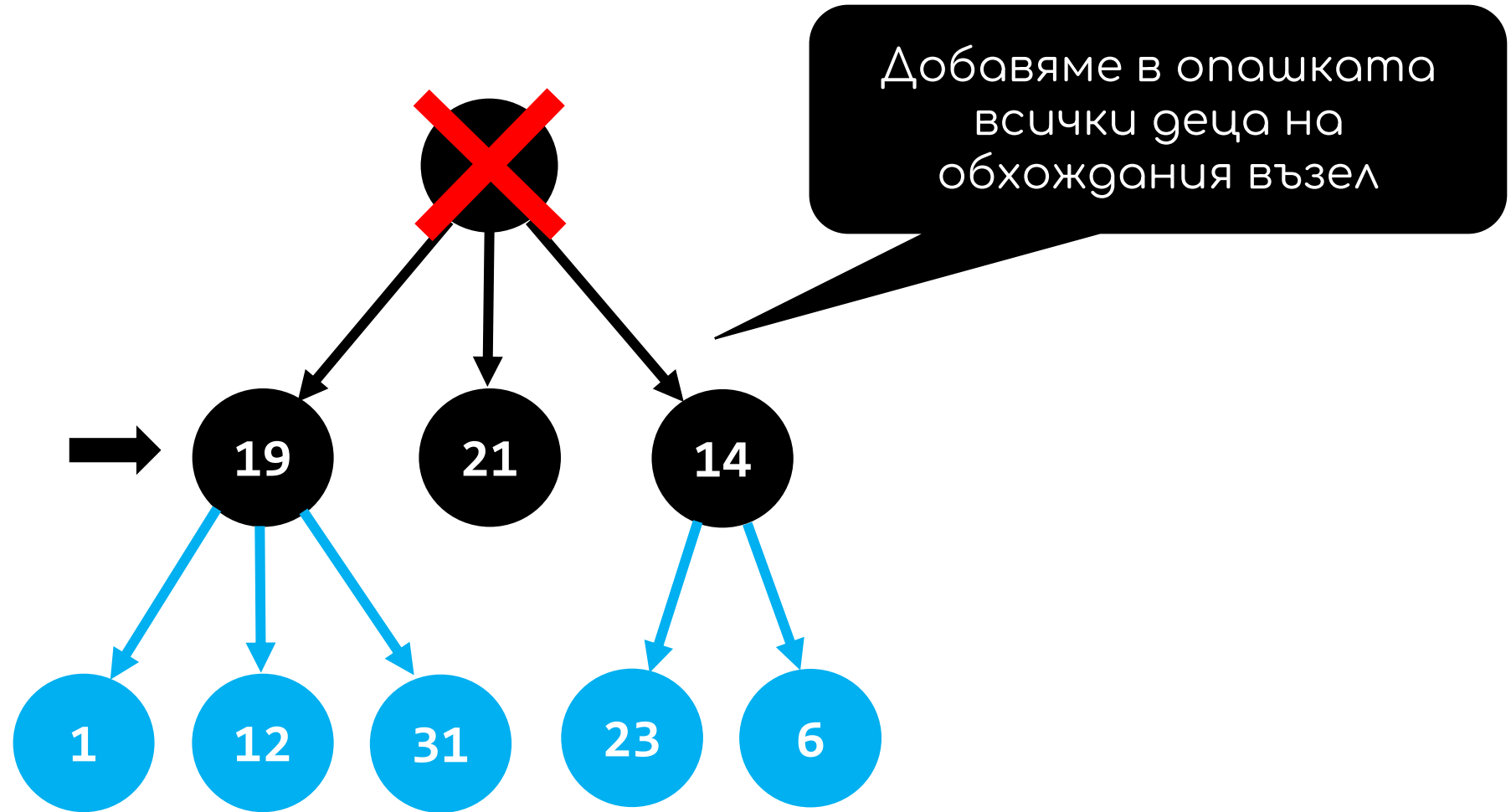
BFS в действие [5/19]

Опашка:

19, 21, 14

Изход:

7



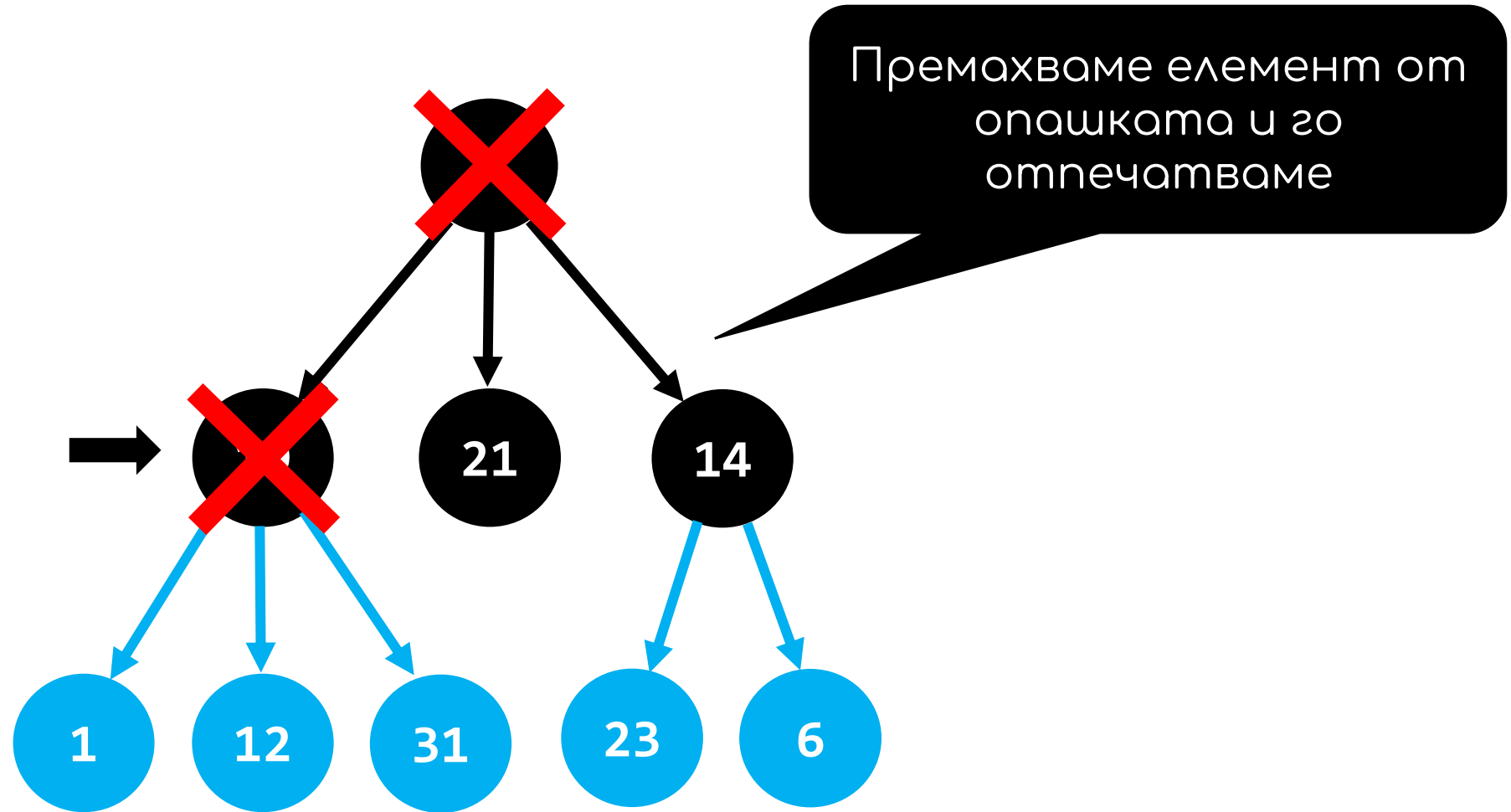
BFS в действие [6/19]

Опашка:

21, 14

Изход:

7, 19



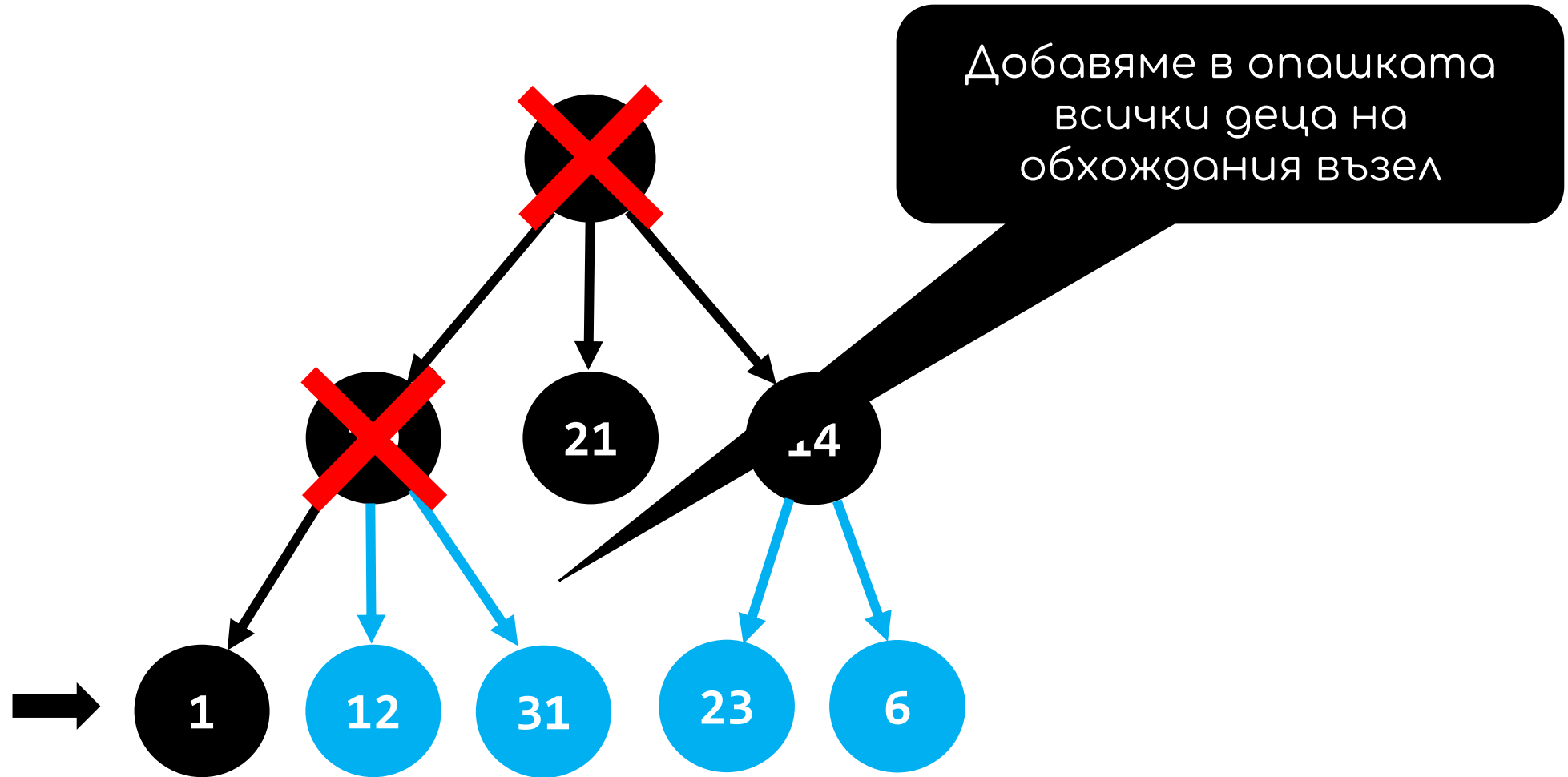
BFS в действие [7/19]

Опашка:

21, 14, 1

Изход:

7, 19



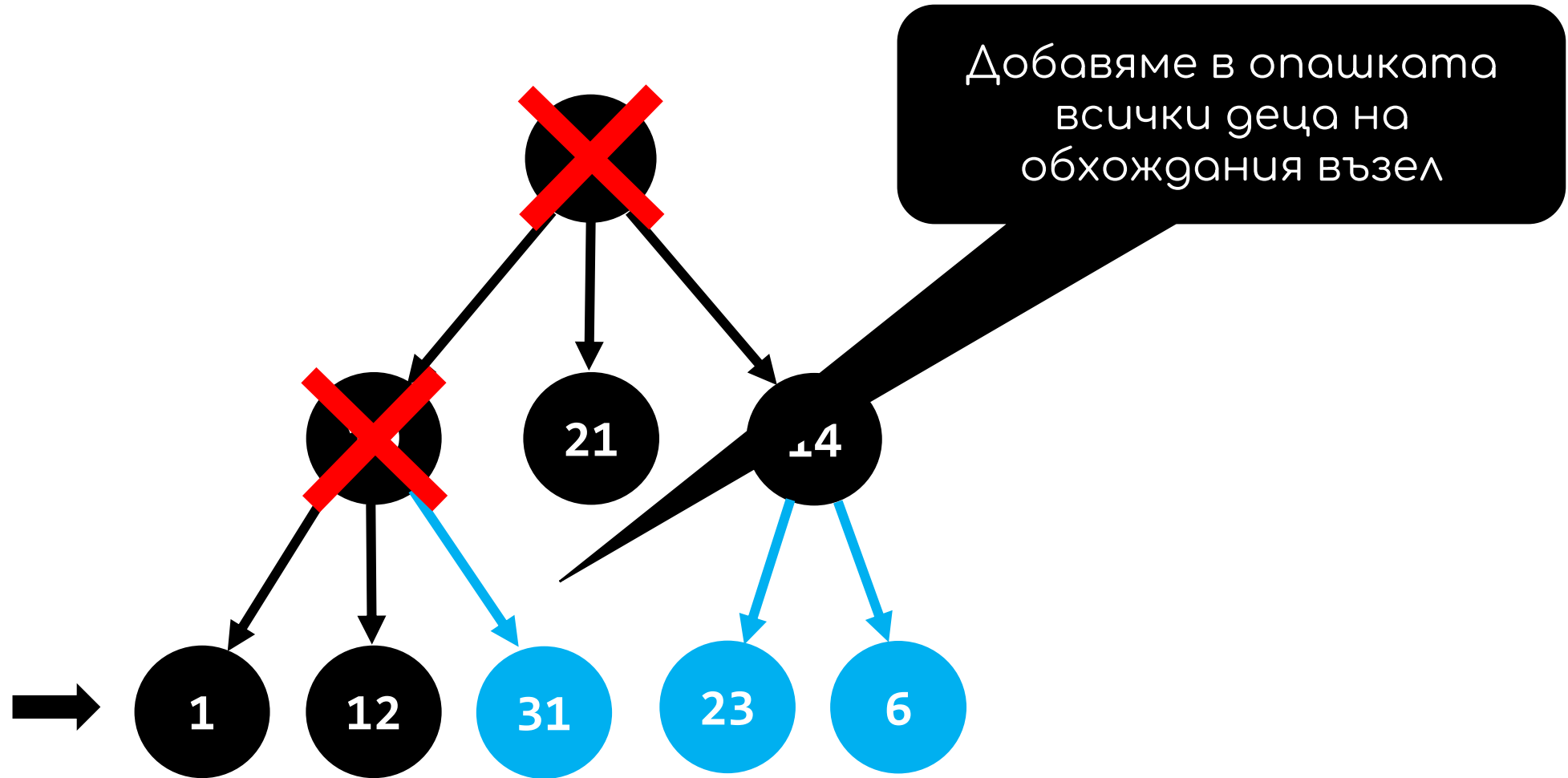
BFS в действие [8/19]

Опашка:

21, 14, 1, 12

Изход:

7, 19



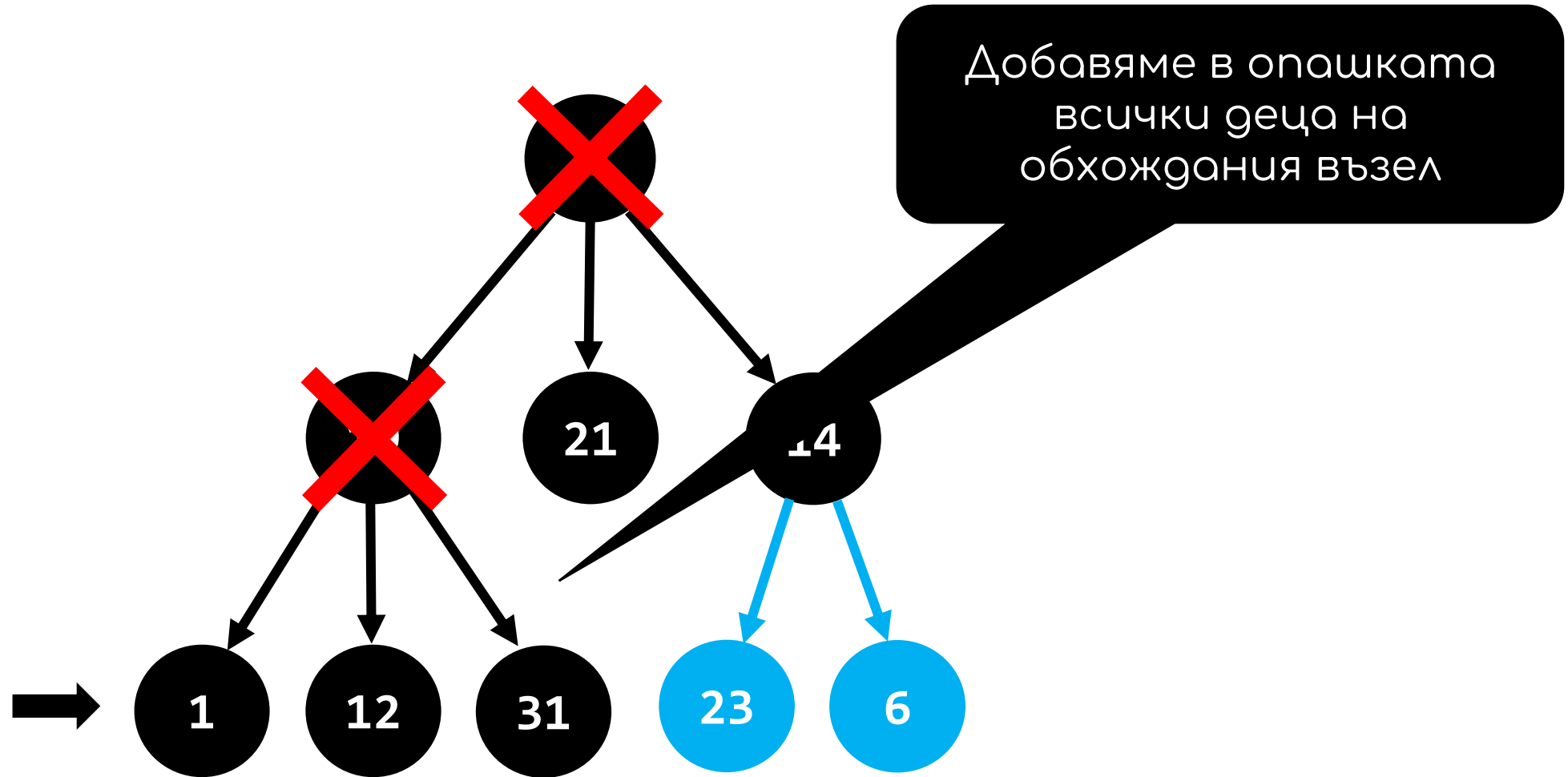
BFS в действие [9/19]

Опашка:

21, 14, 1, 12, 31

Изход:

7, 19



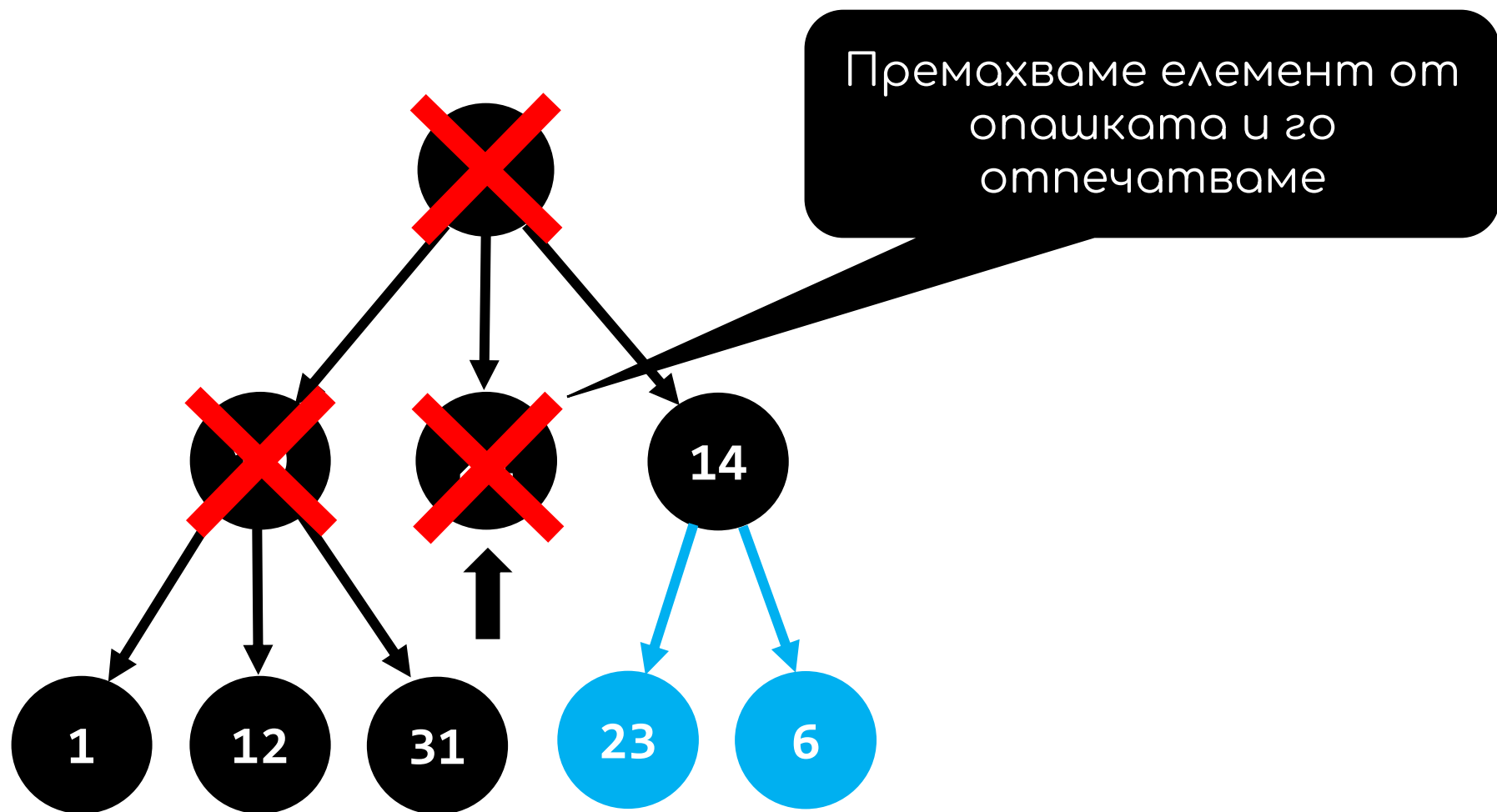
BFS в действие [10/19]

Опашка:

14, 1, 12, 31

Изход:

7, 19, 21



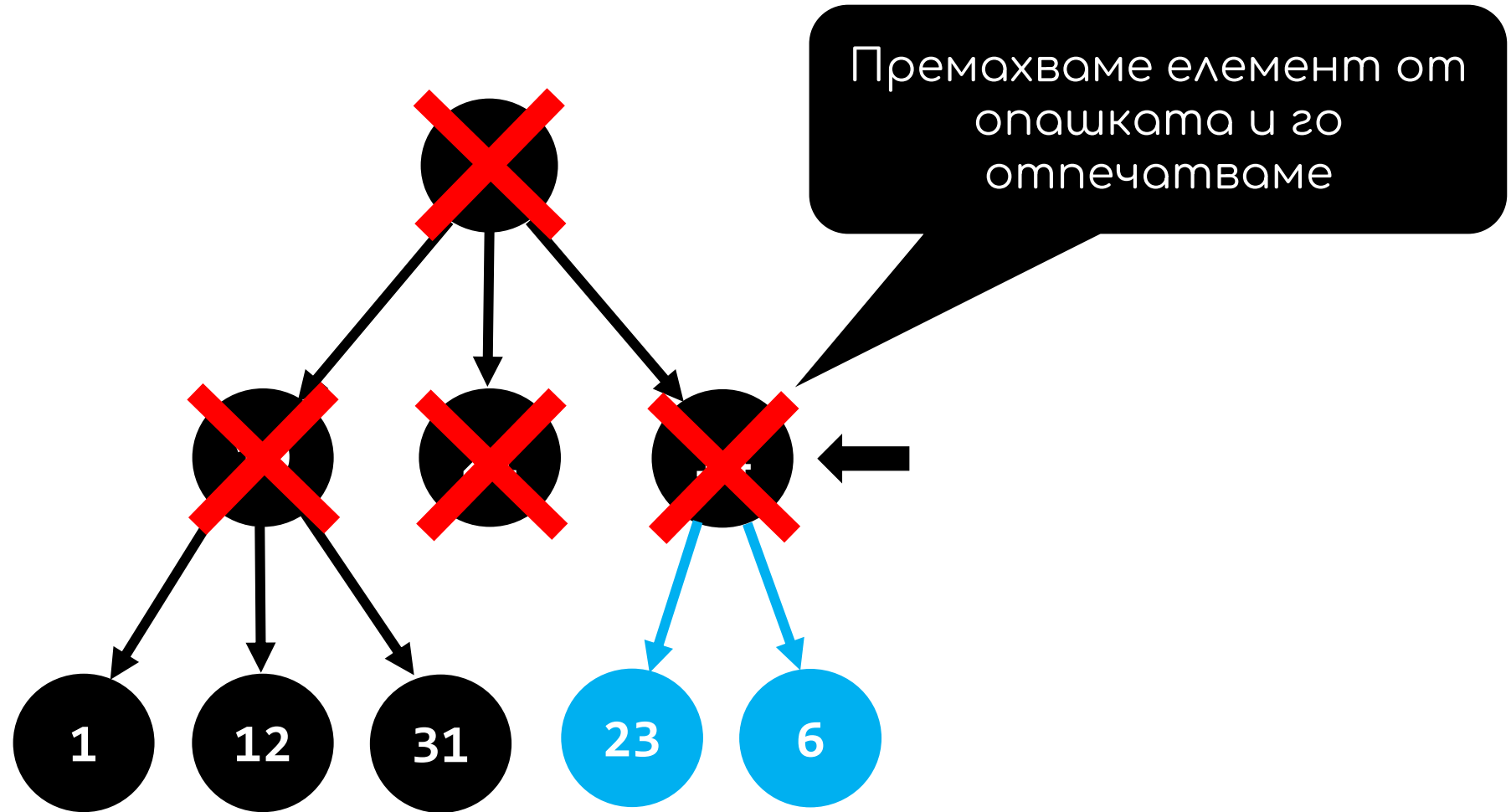
BFS в действие [11/19]

Опашка:

1, 12, 31

Изход:

7, 19, 21, 14



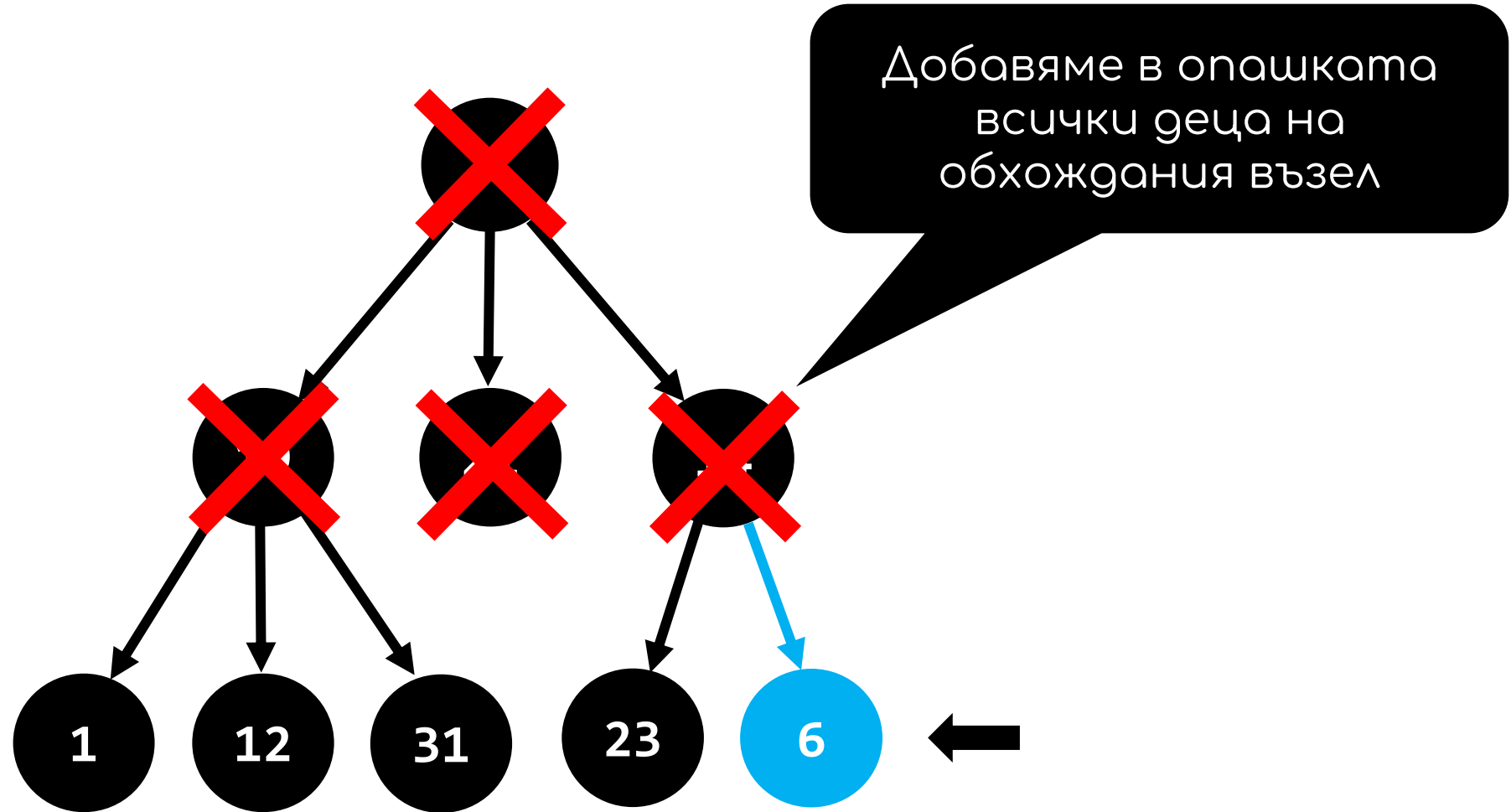
BFS в действие [12/19]

Опашка:

1, 12, 31, 23

Изход:

7, 19, 21, 14



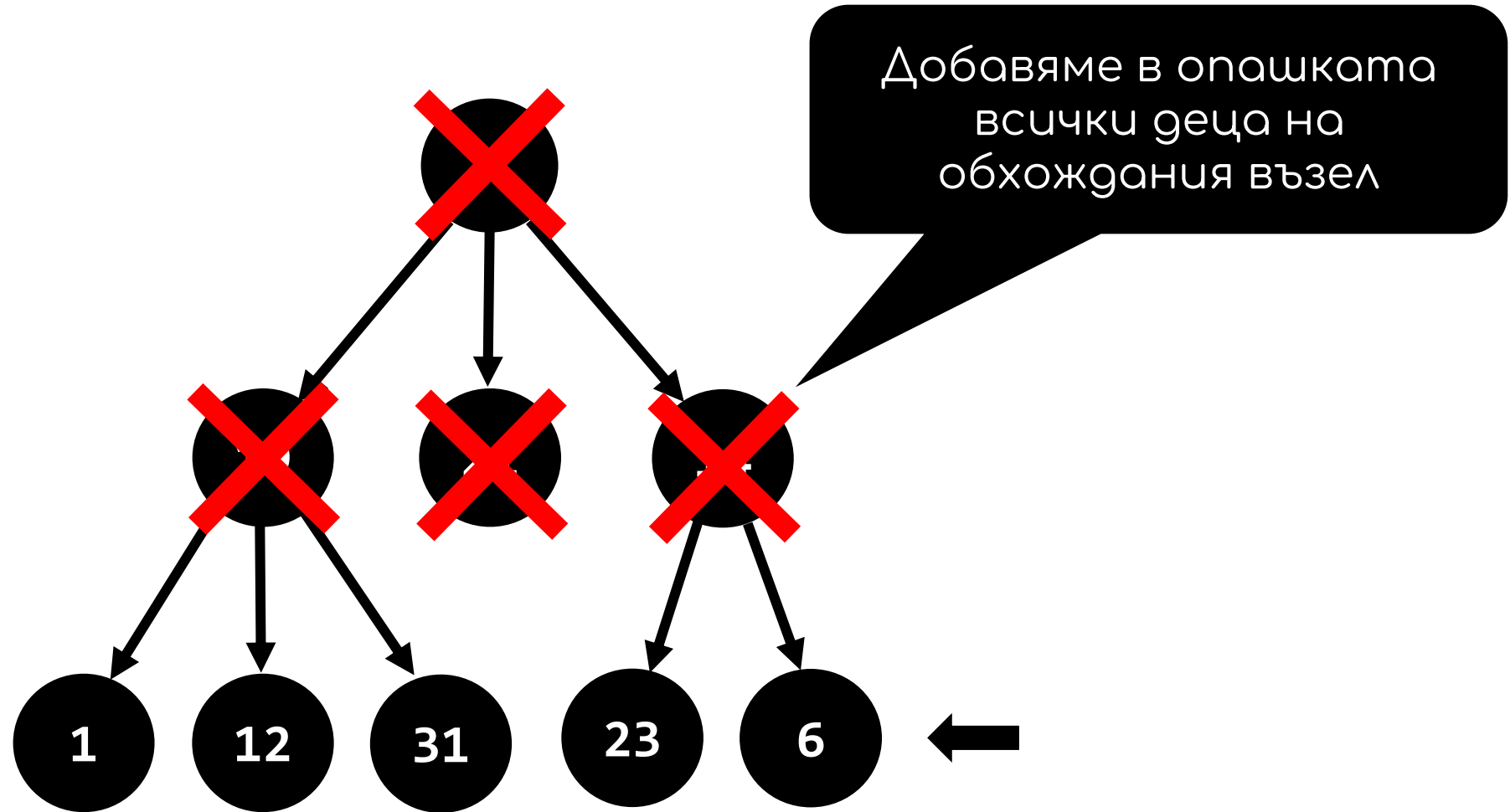
BFS в действие [13/19]

Опашка:

1, 12, 31, 23, 6

Изход:

7, 19, 21, 14



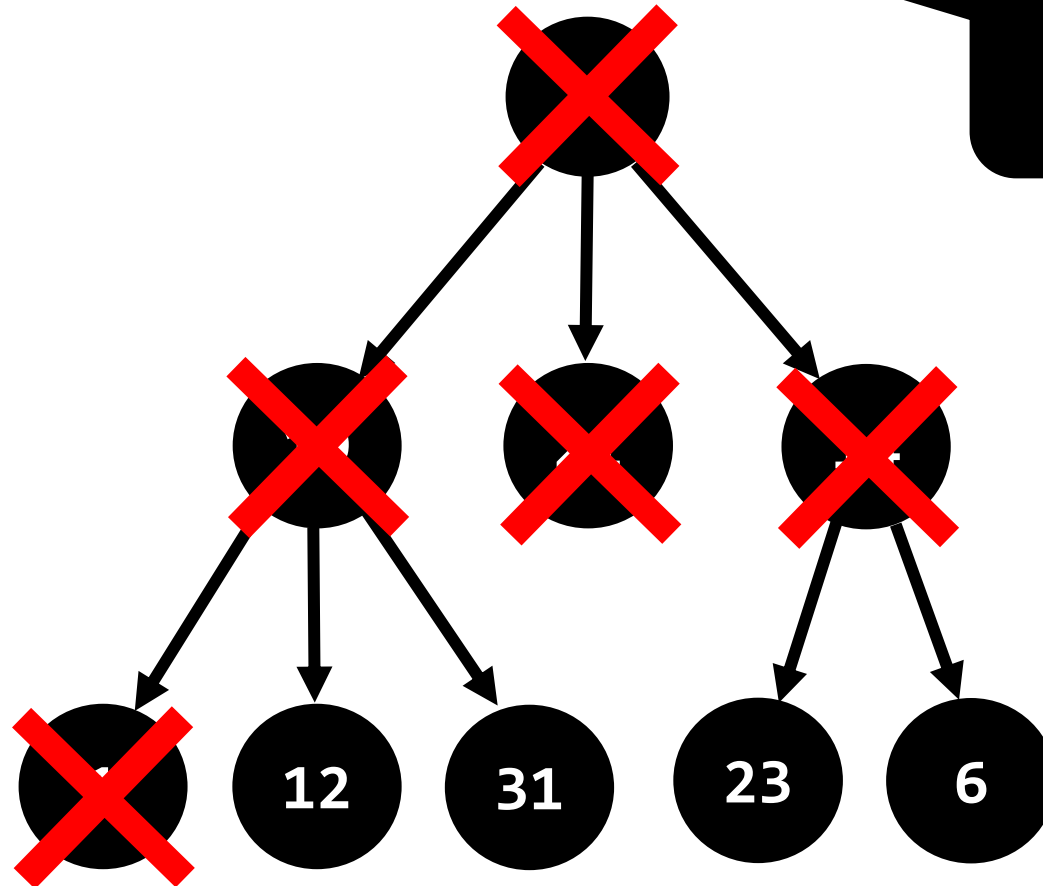
BFS в действие [14/19]

Опашка:

12, 31, 23, 6

Изход:

7, 19, 21, 14, 1



Премахваме елемент от
опашката и го
отпечатваме

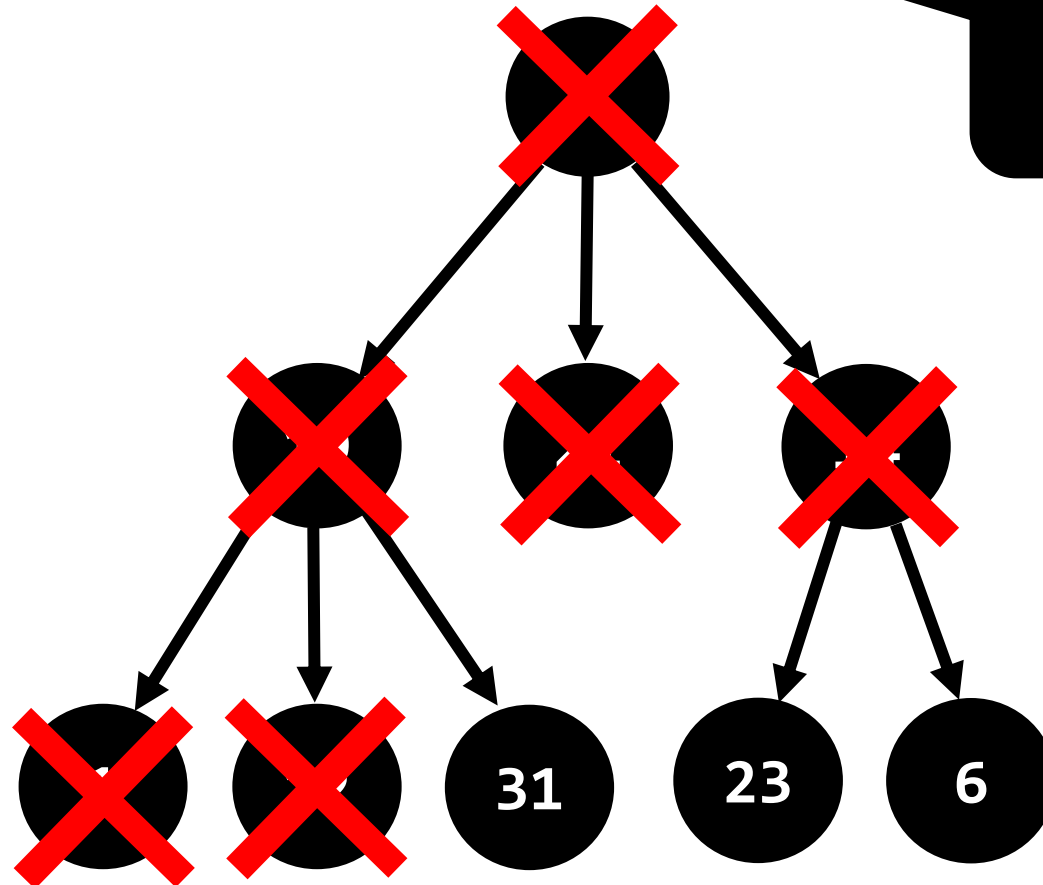
BFS в действие [15/19]

Опашка:

31, 23, 6

Изход:

7, 19, 21, 14, 1,
12



Премахваме елемент от
опашката и го
отпечатваме

BFS в действие [16/19]

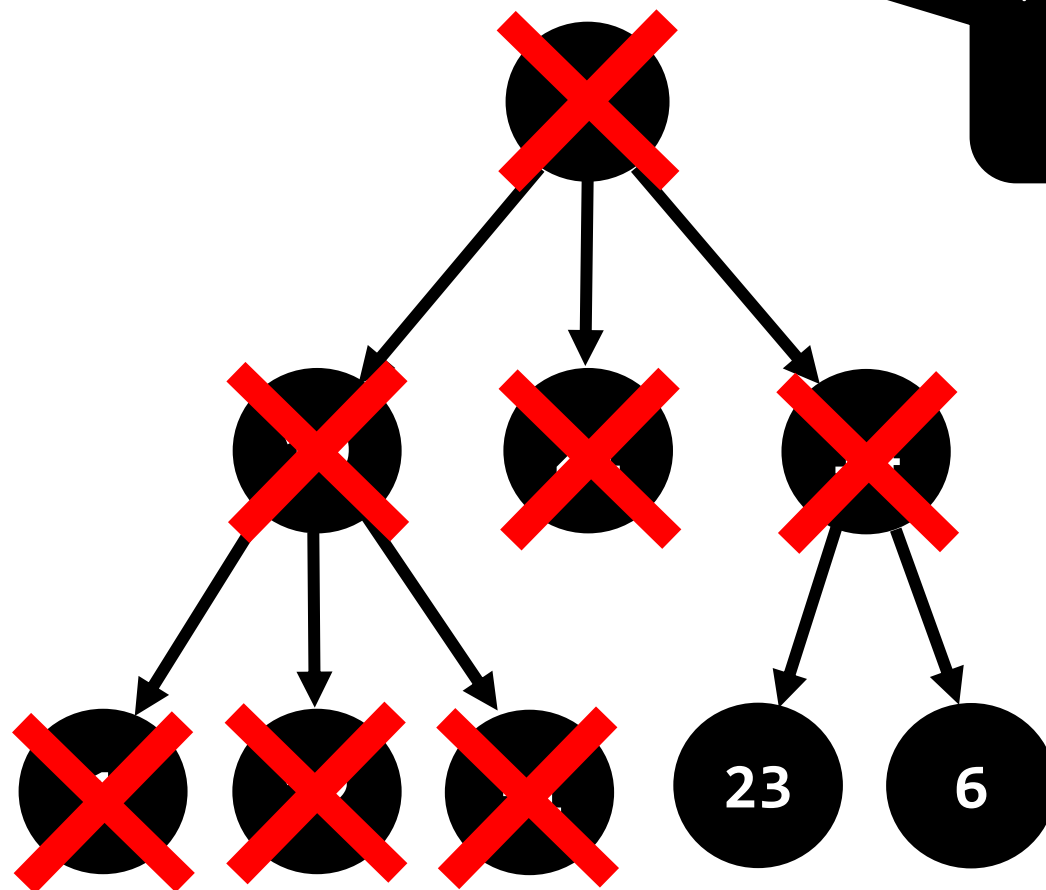
Опашка:

23, 6

Изход:

7, 19, 21, 14, 1,

12, 31



Премахваме елемент от
опашката и го
отпечатваме

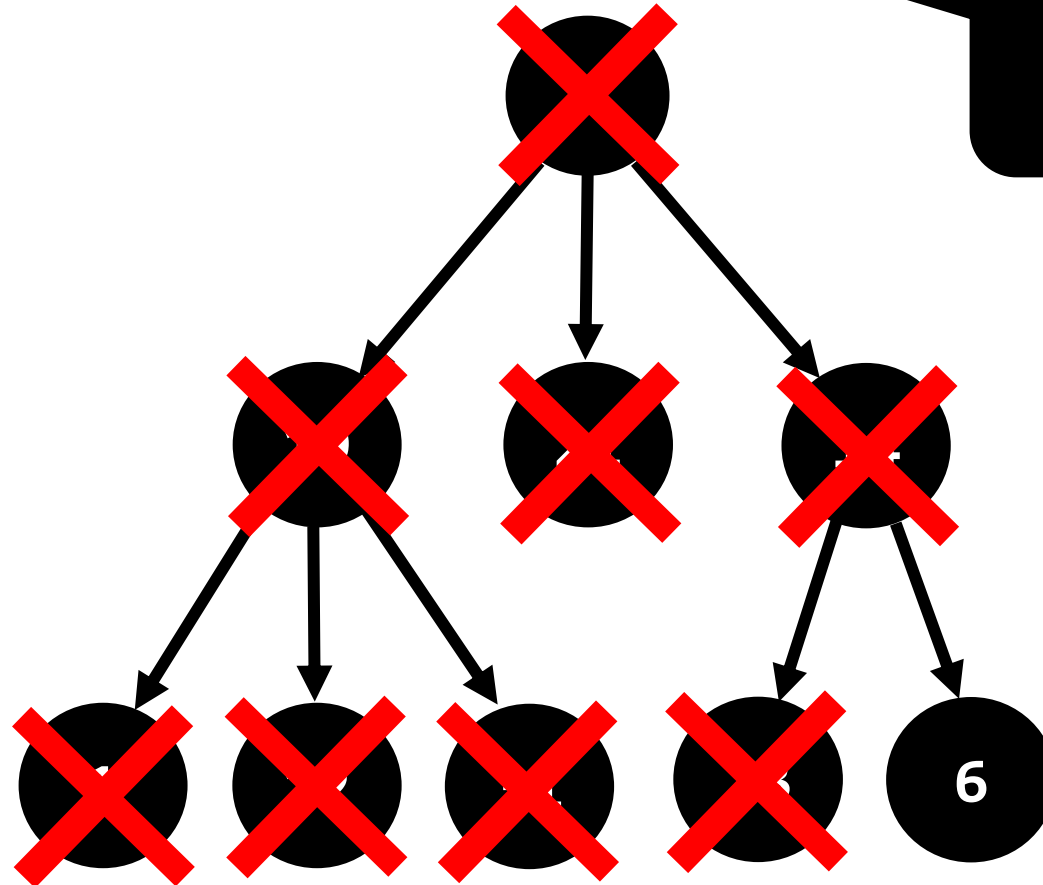
BFS в действие [17/19]

Опашка:

6

Изход:

7, 19, 21, 14, 1,
12, 31, 23

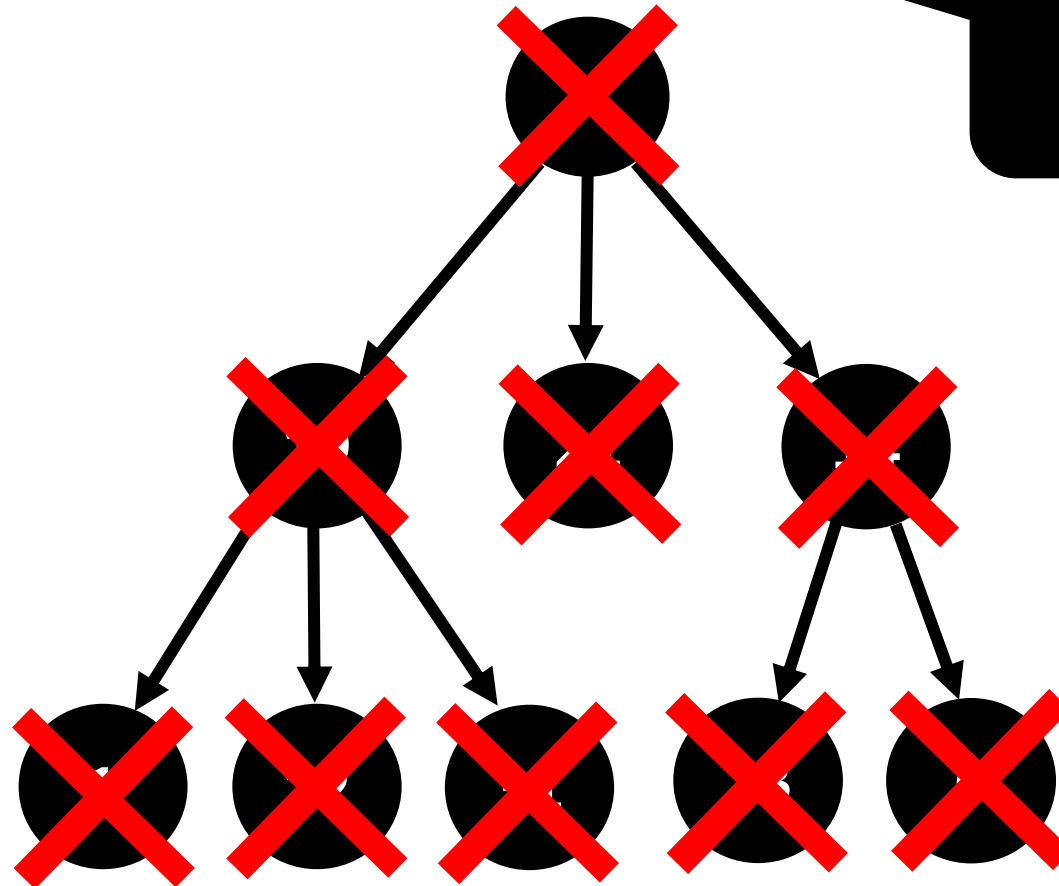


Премахваме елемент от
опашката и го
отпечатваме

BFS в действие [18/19]

Опашка:
(празна)

Изход:
7, 19, 21, 14, 1,
12, 31, 23, 6

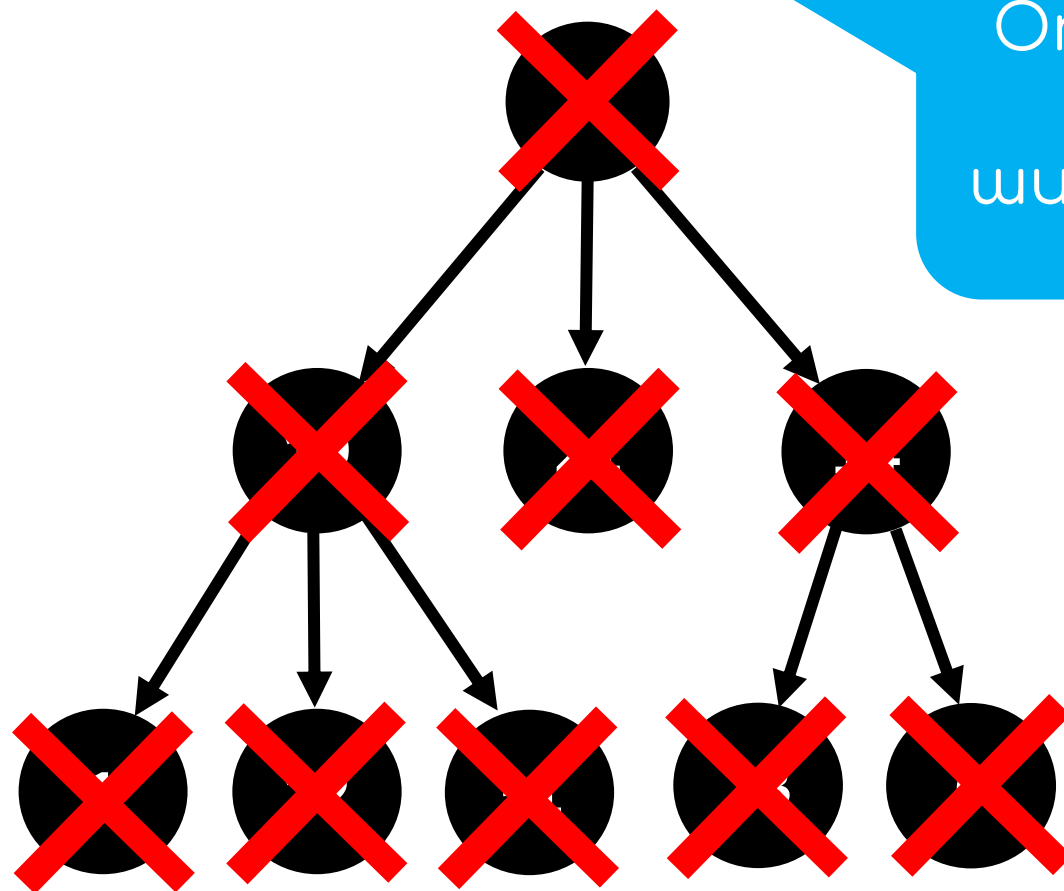


Премахваме елемент от
опашката и го
отпечатваме

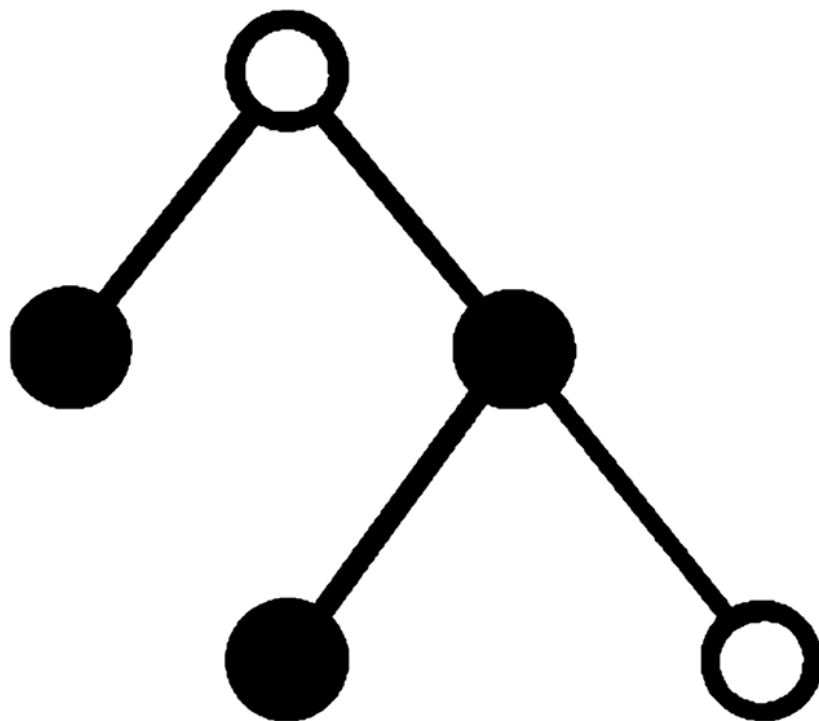
BFS в действие [18/19]

Опашка:
(празна)

Изход:
7, 19, 21, 14, 1,
12, 31, 23, 6



Опашката е празна!
Обхождането в
ширина е приключено



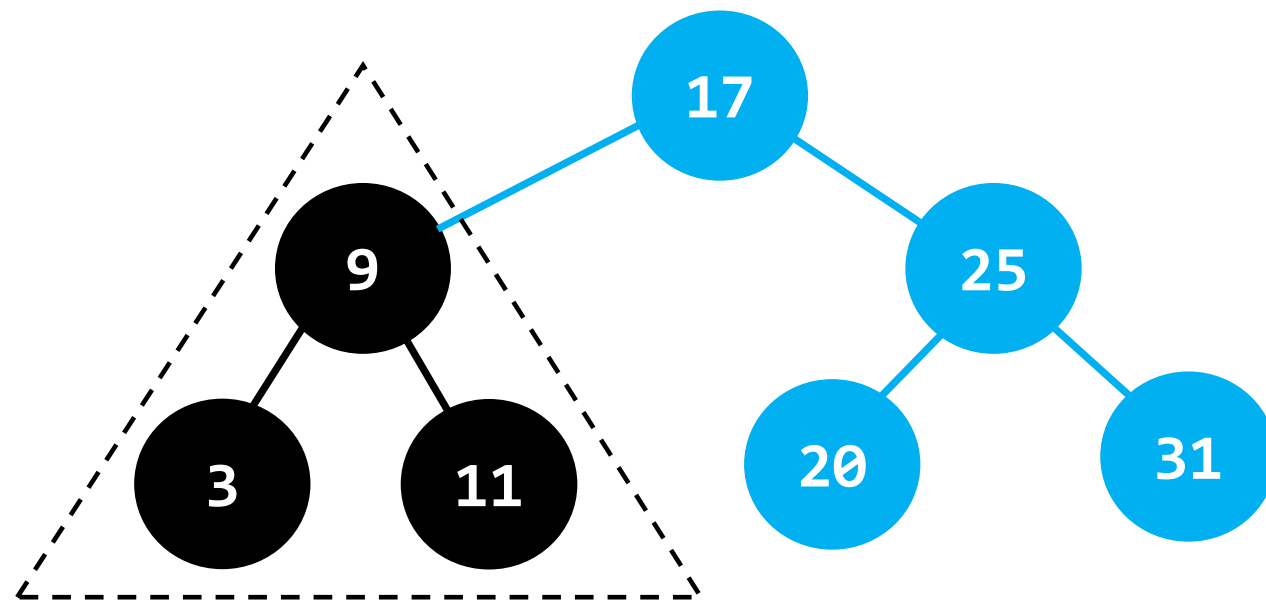
Двоични дървета за търсене
Добавяне, търсене, редакция изтриване

Двоични дървета за търсене

Двоичните дървета за търсене са **подредени**:

- За всеки възел x :
 - Елементите в лявото поддърво на x са по-малки от x
 - Елементите в дясното поддърво на x са по-големи от x

Какво става с елементите равни на x ?



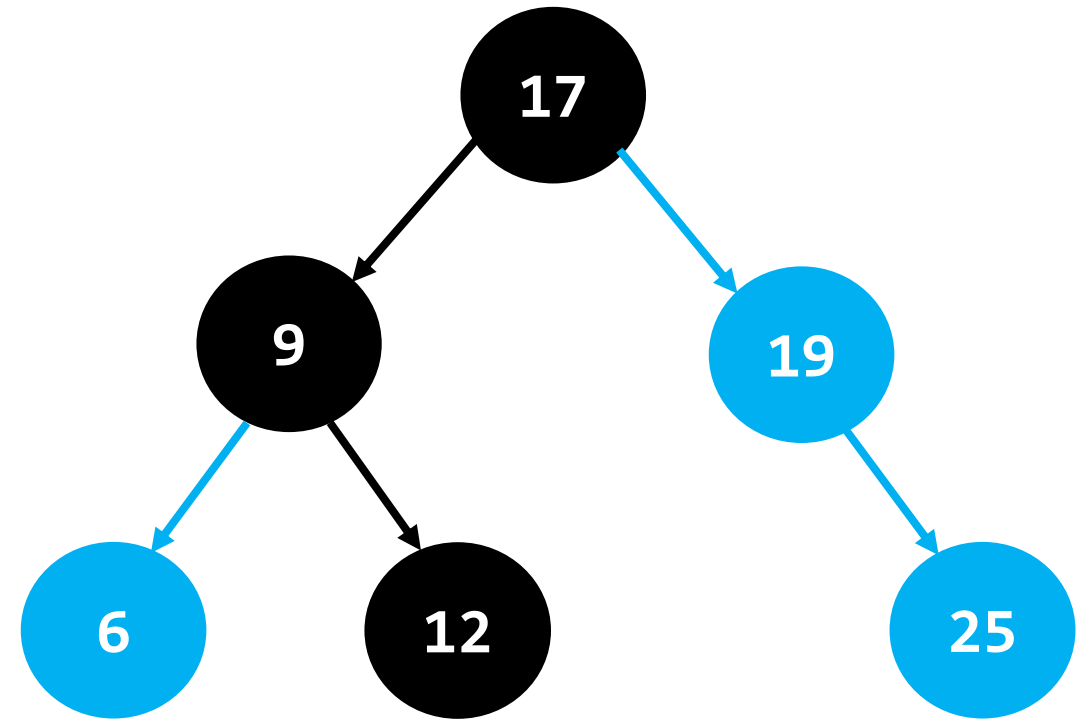
Възлите са < 17

Възел в двоично дърво за търсене

```
public class BinaryTree<T>
{
    private class Node
    {
        public Node Left { get; set; }
        public Node Right { get; set; }
        public T Item { get; set; }
    }
    private Node Root { get; set; }
    public int Count { get; private set; }
    public void Add(T item)...
    public void Remove(T item)...
    public bool Contains(T item)...
}
```

Търсене в двоично дърво за търсене [1/2]

- Търсене на елемент x в двоично дърво за търсене
 - if $node \neq null$
 - if $x < node.value \rightarrow$ левия клон
 - else if $x > node.value \rightarrow$ десния клон
 - else if $x == node.value \rightarrow$ върни възел



Търсим 12 -> 17 9 12

Търсим 27 -> 17 19 25 null

Търсене в двоично дърво за търсене [2/2]

```
public bool Contains(T item)
{
    if (Root == null) return false;

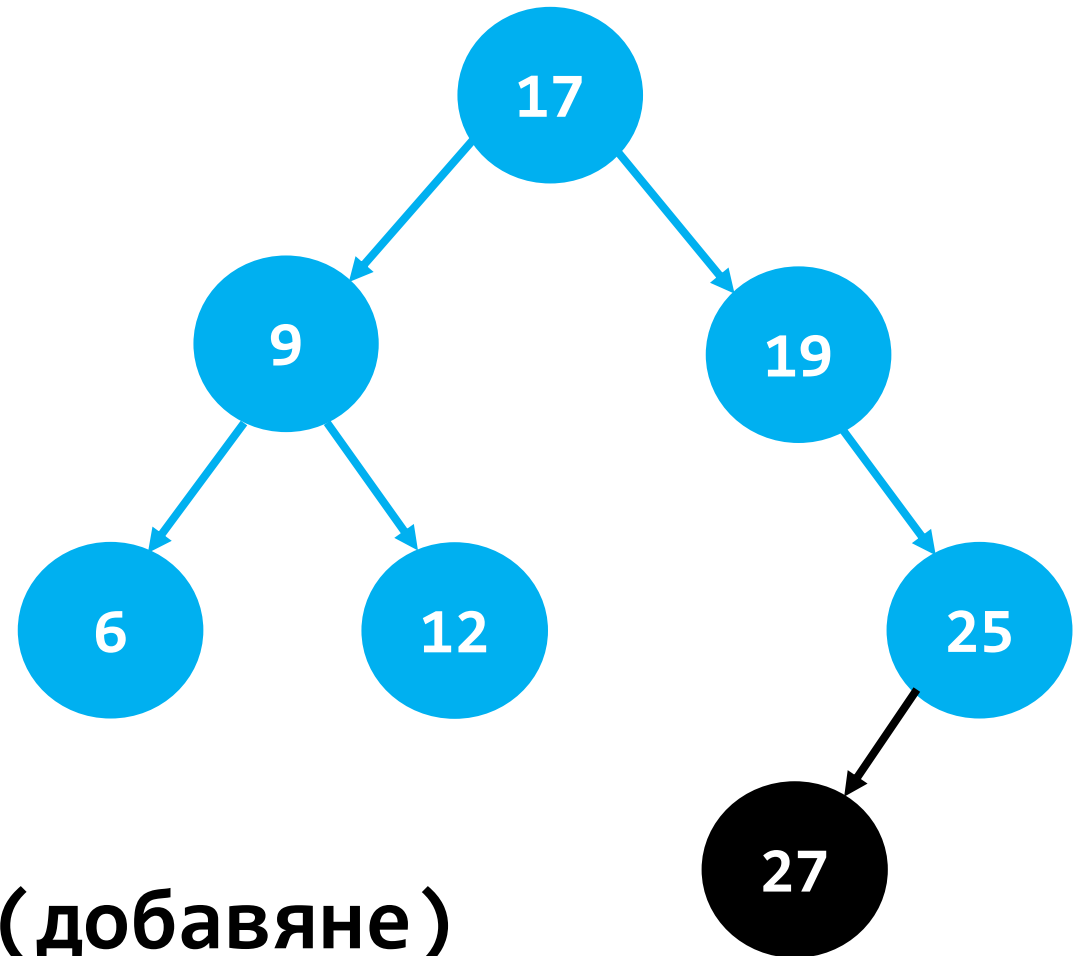
    Node iterator = Root;
    while (true)
    {
        if (iterator == null)
            return false;
        else if (iterator.Item.CompareTo(item) == 0)
            return true;
        else if (iterator.Item.CompareTo(item) > 0)
            iterator = iterator.Left;
        else if (iterator.Item.CompareTo(item) < 0)
            iterator = iterator.Right;
    }
}
```

Добавяне в двоично дърво за търсене [1/2]

- Добавяне на елемент x в двоично дърво за търсене
 - if $node == null \rightarrow$ добави x
 - else if $x < node.value \rightarrow$ ляв клон
 - else if $x > node.value \rightarrow$ десен клон
- else \rightarrow възела съществува

Добавяне 12 \rightarrow 17 9 12 return

Добавяне 27 \rightarrow 17 19 25 null (добавяне)



Добавяне в двоично дърво за търсене [2/2]

```
public void Add(T item)
{
    Node node = new Node();
    node.Item = item;

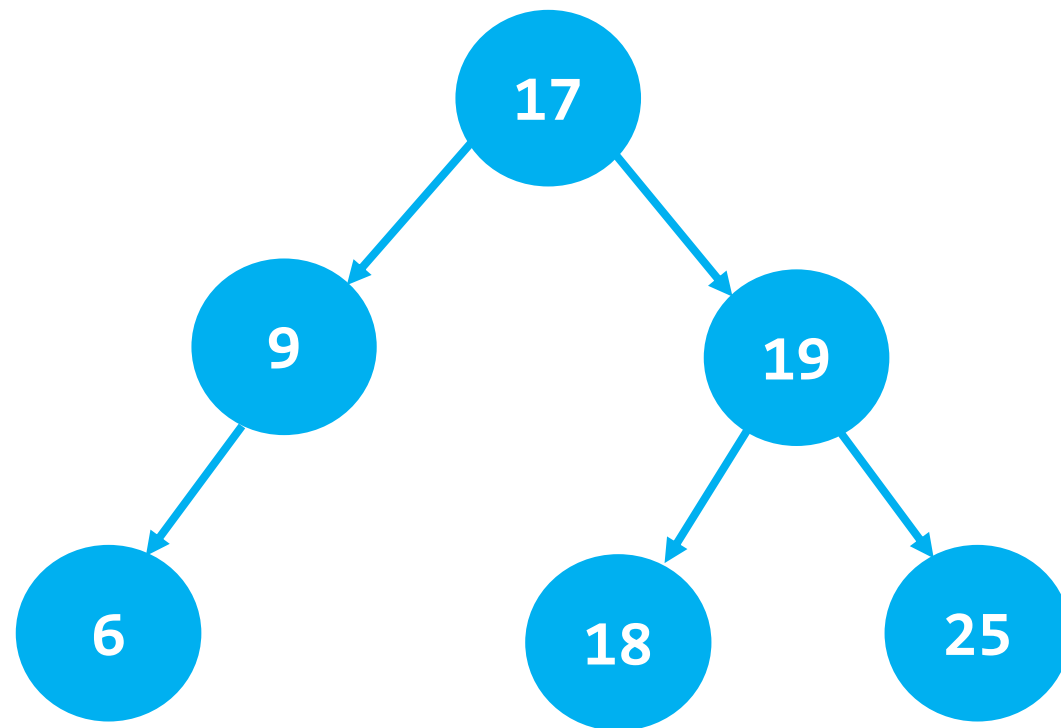
    if (Root == null)
    {
        Root = node;
        return;
    }

    Node iterator = Root;
    while (true)
    {
        if (iterator.Left != null && iterator.Item.CompareTo(item) >= 0)
            iterator = iterator.Left;
        else if (iterator.Right != null && iterator.Item.CompareTo(item) < 0)
            iterator = iterator.Right;
        else
            break;
    }

    if (iterator.Item.CompareTo(item) >= 0)
        iterator.Left = node;
    else if (iterator.Item.CompareTo(item) < 0)
        iterator.Right = node;
}
```

Премахване от двоично дърво за търсене [1/4]

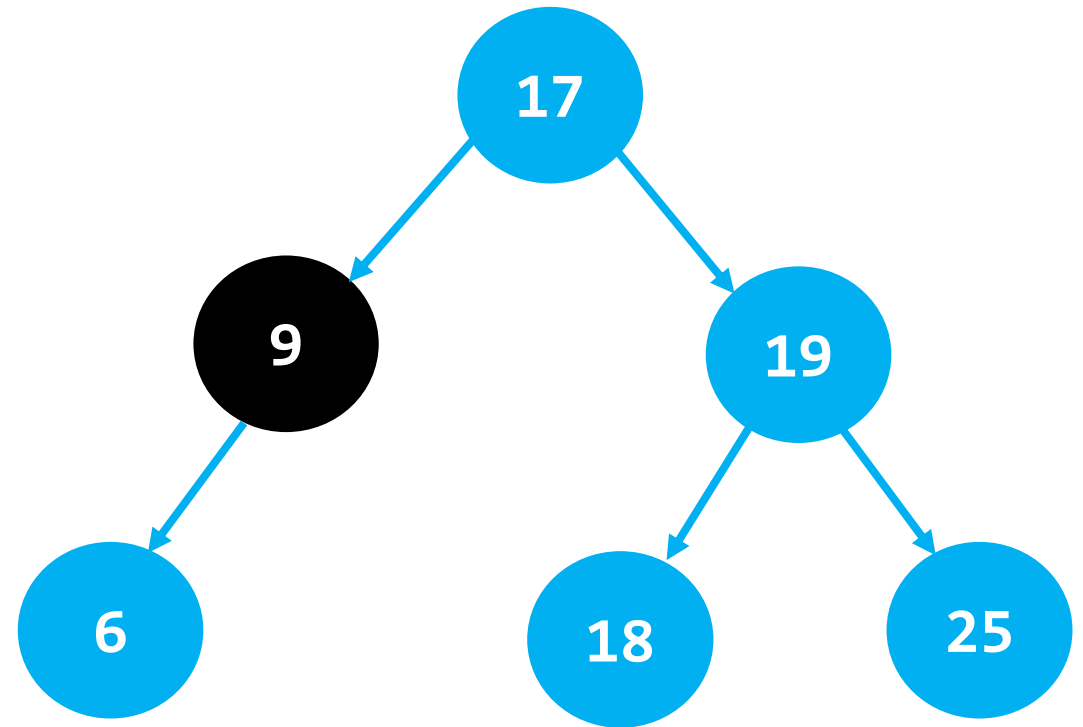
- Премахване на елемент x в двоично дърво за търсене
 - if node == null -> изход
 - else if x is leaf -> премахни
 - else if x is not leaf -> подмени
 - (3 случая при подмяна на възел)



Премахване от двоично дърво за търсене [2/4]

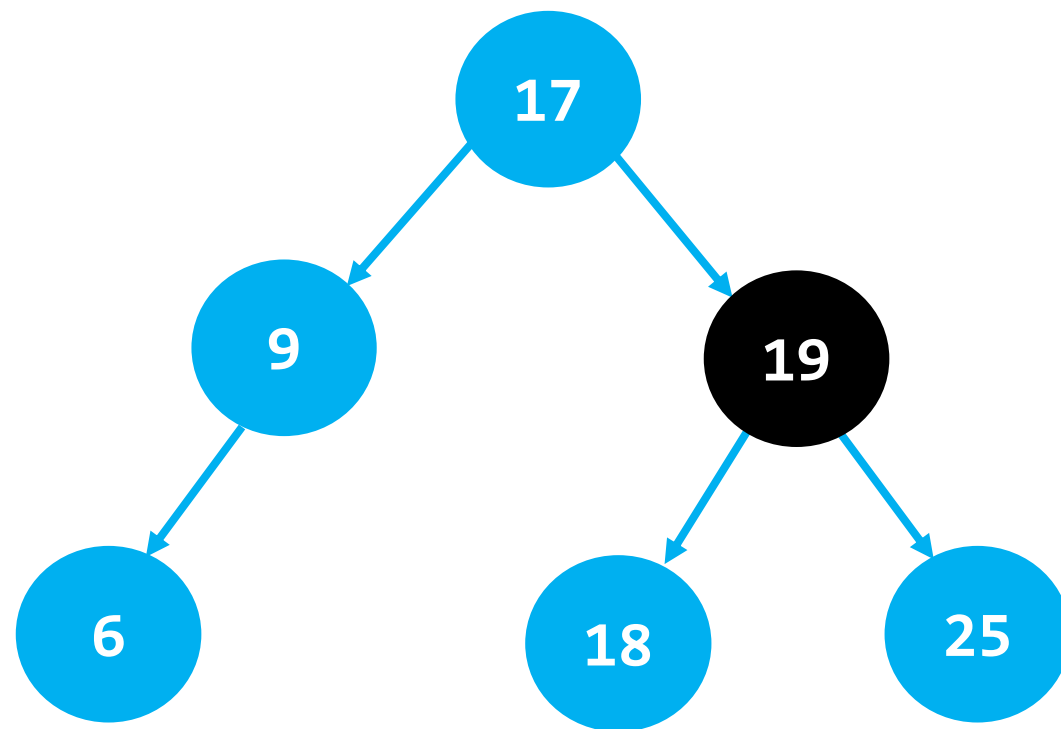
- Премахване на елемент, който няма дясно поддърво
 - Намираме елемента за премахване
 - Корена на лявото поддърво заема мястото на премахнатия елемент

Example: Delete 9



Премахване от двоично дърво за търсене [3/4]

- Премахване на елемент, чието дясно поддърво няма ляво поддърво
 - Намираме елемента за премахване
 - Корена на дясното поддърво заема мястото на премахнатия елемент

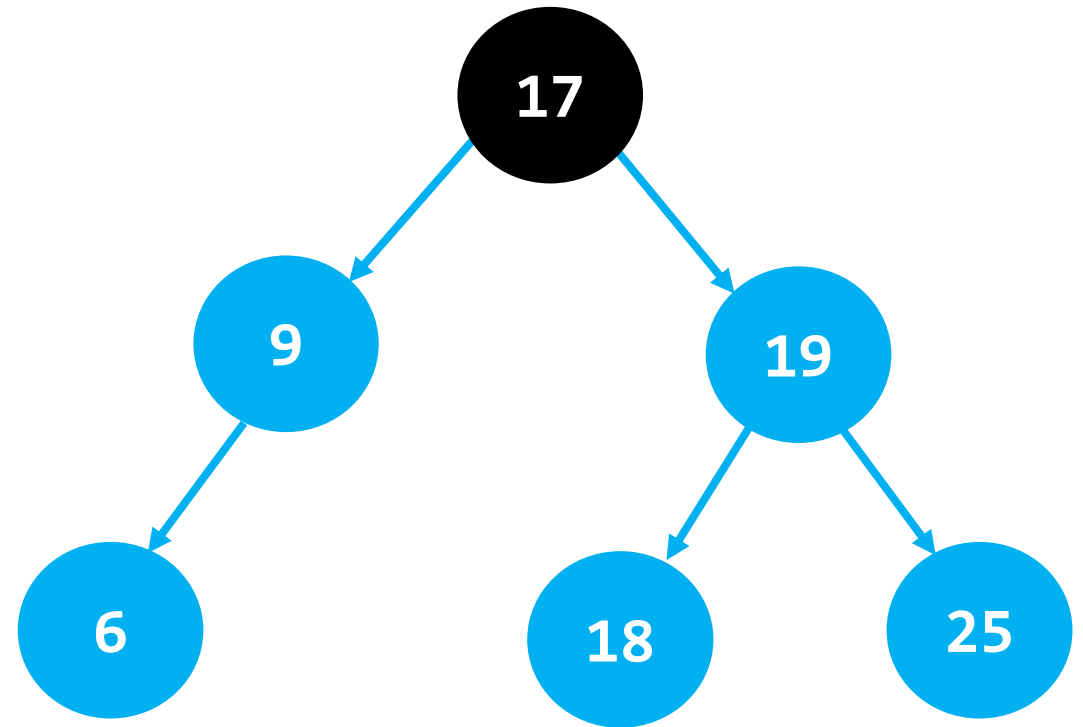


Example: Delete 19

Премахване от двоично дърво за търсене [4/4]

- Премахване на елемент, който има и ляво и дясно поддърво
 - Намираме елемента за премахване
 - Намираме най-малкия елемент в лявото разклонение на дясното му поддърво
 - Разменяме двата елемента и извършваме премахването

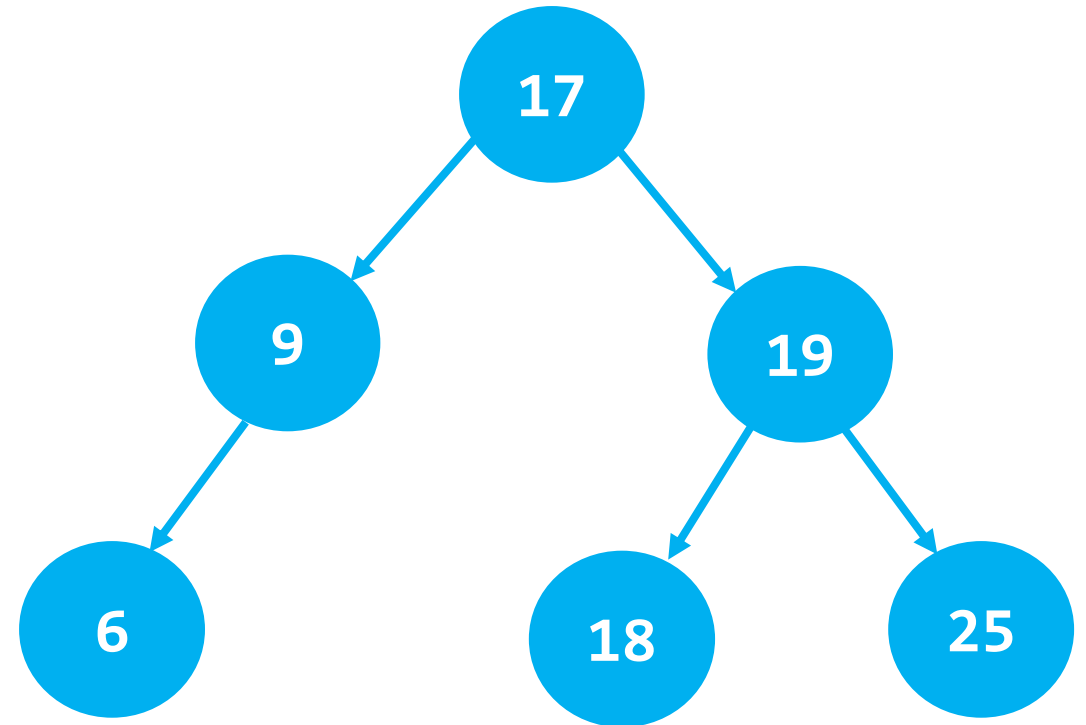
Example: Delete 17



Сложност при двоични дървета за търсене

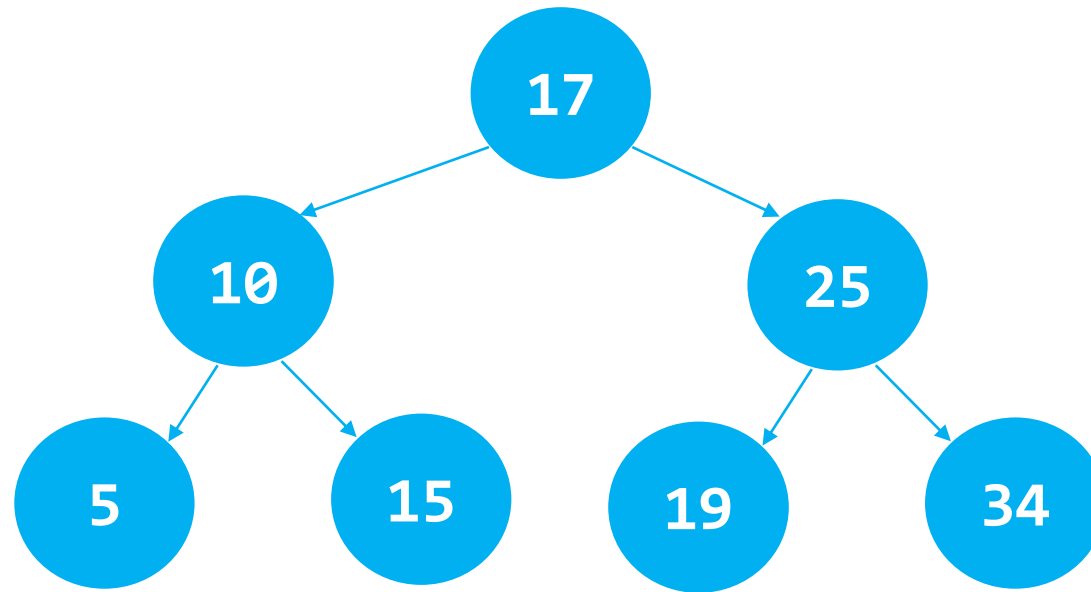
$O(n)$

- **Добавяне** – височината на дървото
- **Търсене** – височината на дървото
- **Премахване** – височината на дървото



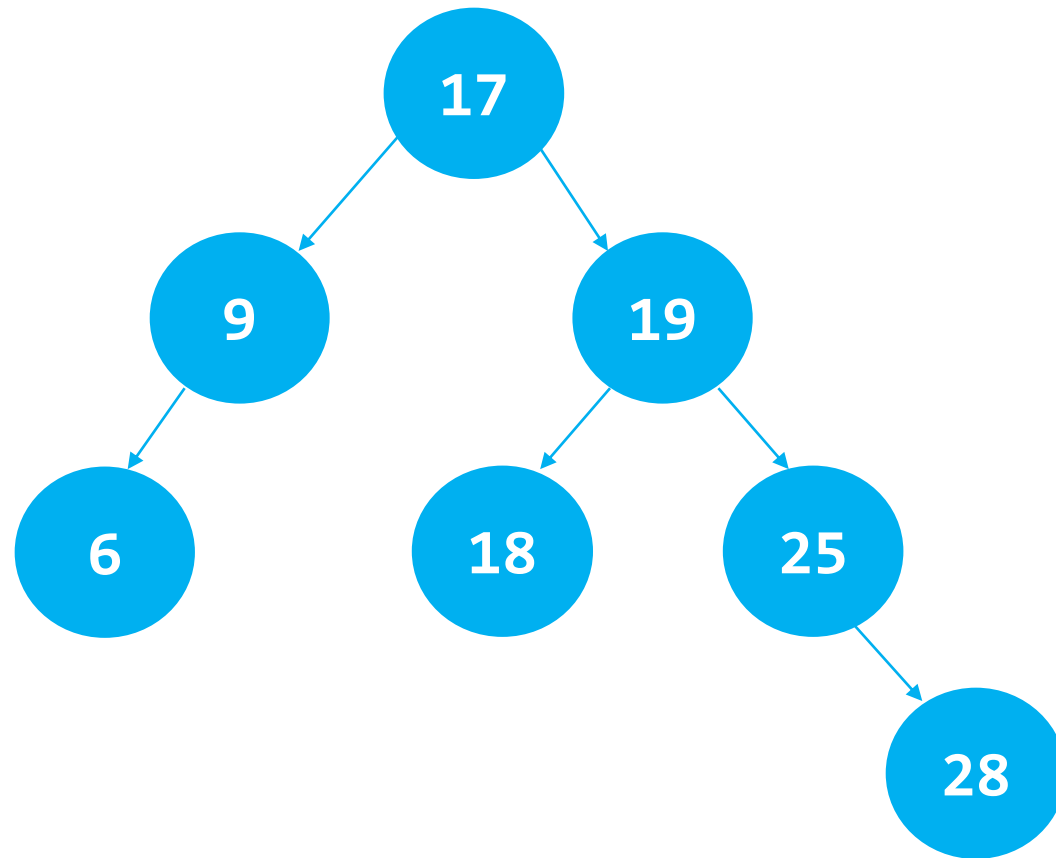
Най-добър случай

Пример: Добавяме: 17, 10, 25, 5, 15, 19, 34



Стандартен случай

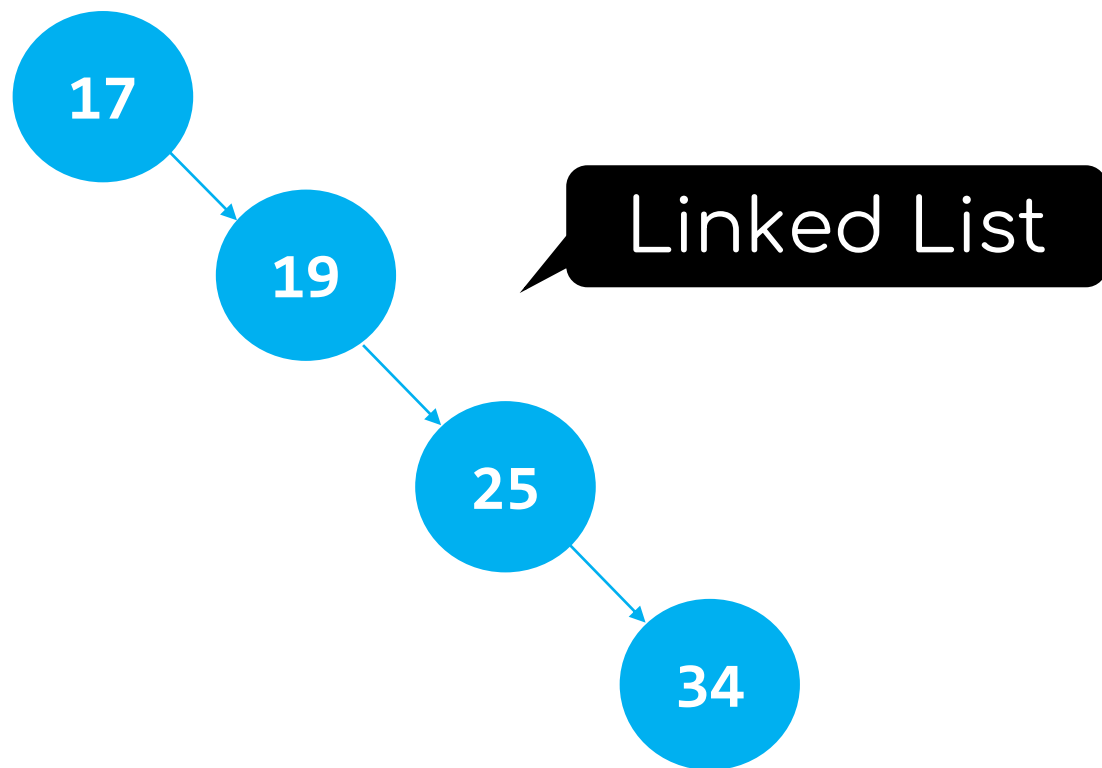
- Добавяне на стойности в произволна последователност
- Пример: Добавяме 17, 19, 9, 6, 25, 28, 18

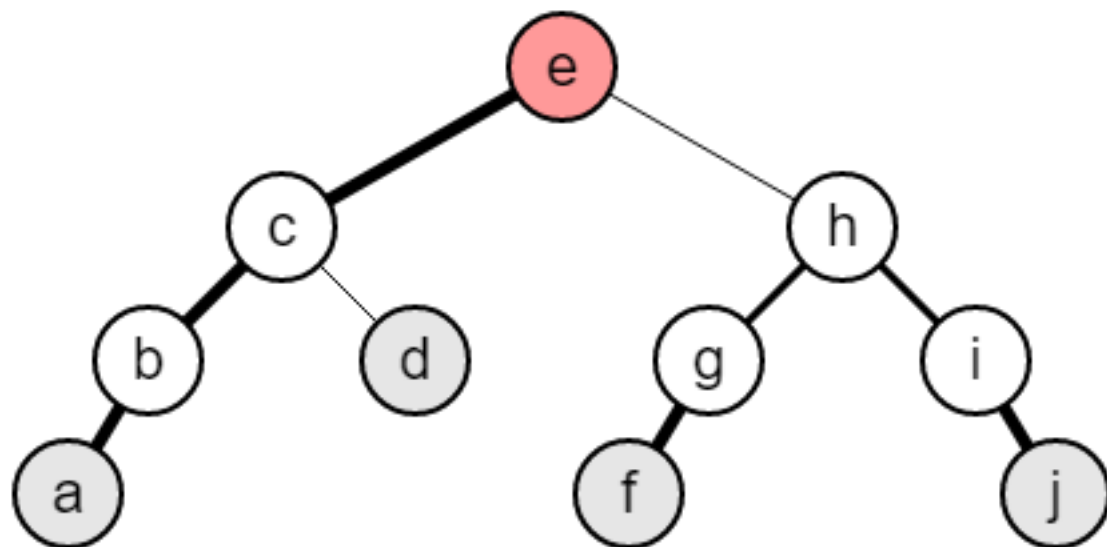


Най-лош случай

Добавяне на стойности в нарастваща/намаляваща последователност

Пример: Добавяме 17, 19, 25, 34

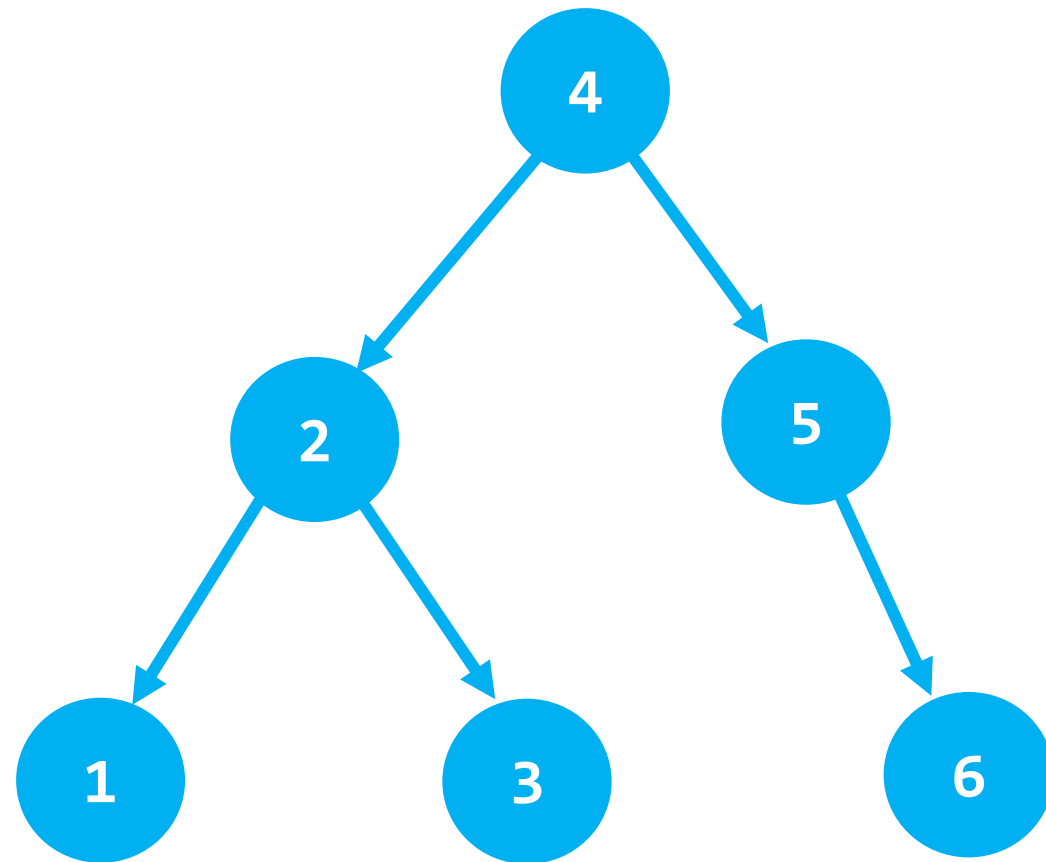




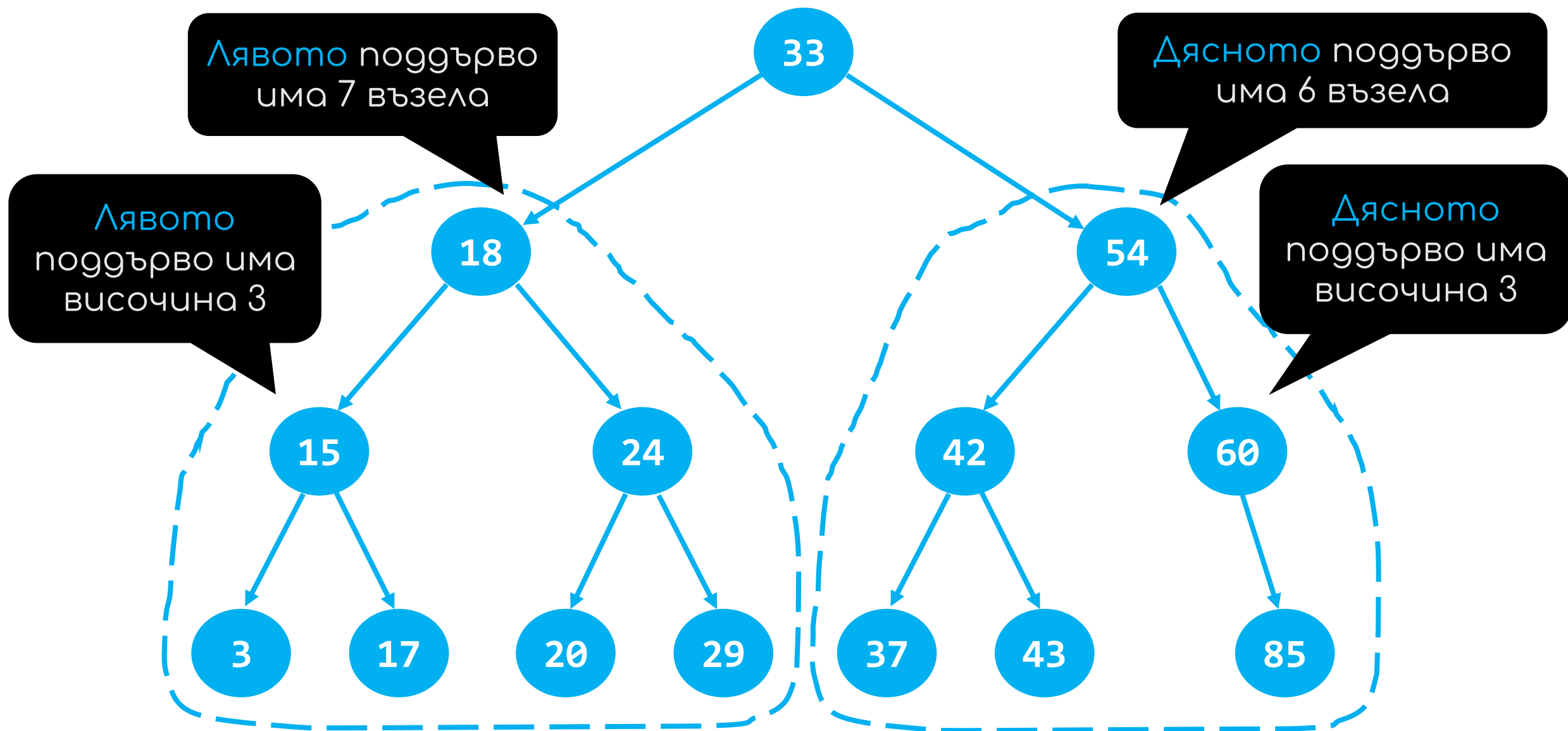
Балансирани дървета
Предназначение

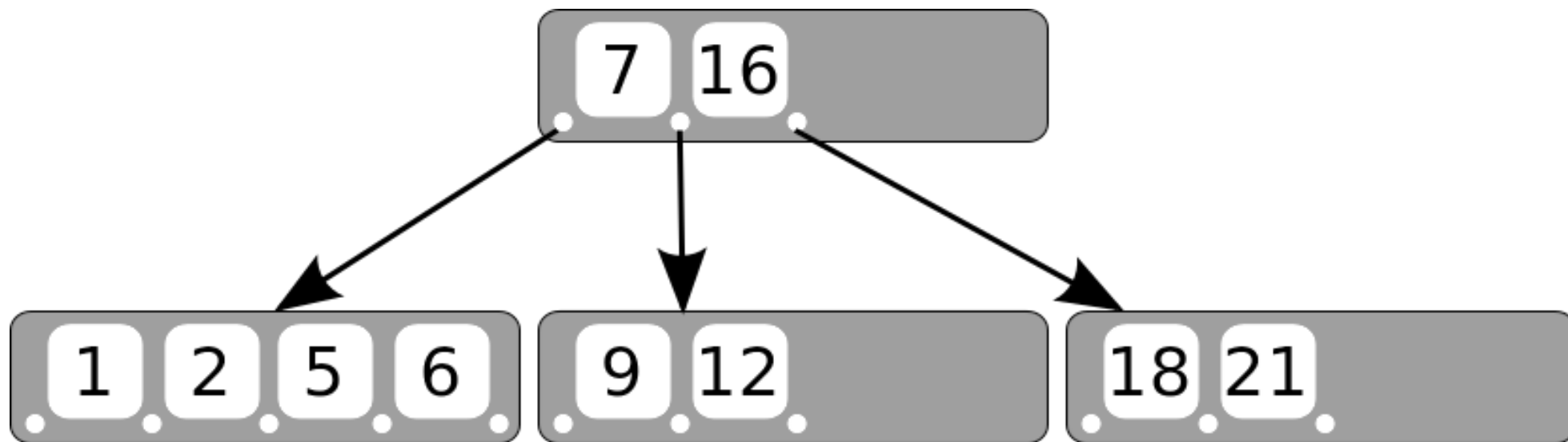
Балансирани двоични дървета за търсене

- Двоичните дървета за търсене могат да бъдат **балансирани**
 - В балансираните дървета всеки възел има почти еднакъв брой възли във своите поддървета
 - Балансираните дървета имат **височина** приблизително равна на $\log(n)$



Балансирани двоични дървета за търсене





Б-Дървета (B-Trees)
Предназначение

Какво са B-Trees?

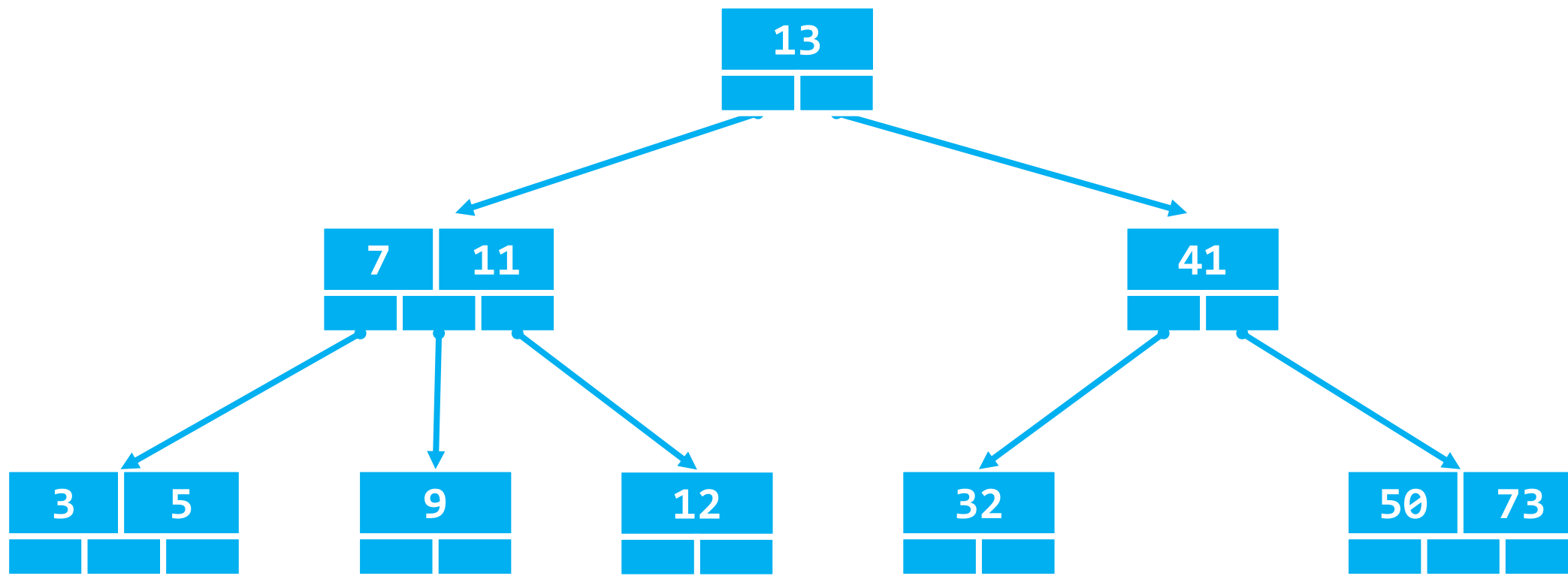
B-trees са генерализация на концепцията за подредени двоични дървета за търсене (визуализация)

- Всеки възел в B-tree от ред b съхранява между b и $2*b$ ключове и има между $b+1$ и $2*b+1$ наследника
- Ключовете във всеки възел са подредени нарастващо
- Всички ключове в наследниците има стойности, ограничени в диапазона на техните леви и десни родителски ключове

B-trees могат ефективно да се съхраняват на твърди дискове

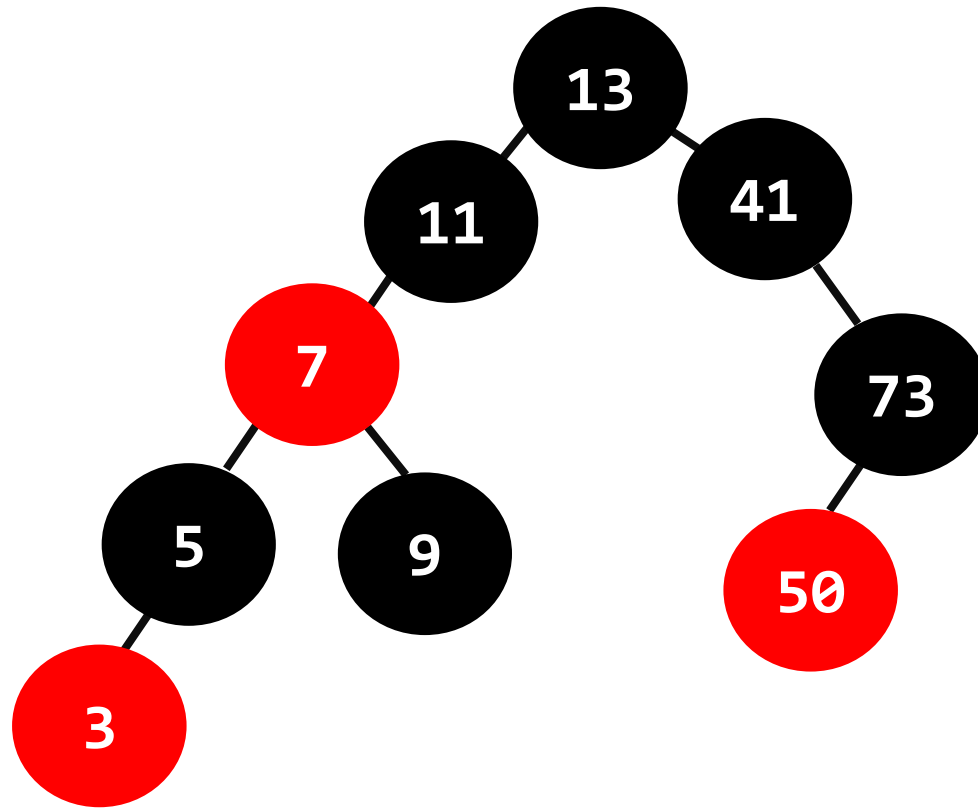
Пример на B-Tree

B-Tree от ред **3**, познати още като 2-3 дървета



B-Trees и други балансирани дървета за търсене

- Възлите в B-Trees могат да имат **много наследници**
 - B-trees нямат нужда от често пребалансиране
- B-Trees са добри за **индексиране в бази от данни**:
 - Защото всеки възел може да се съхрани в отделен клъстер на твърдия диск
 - Минимизация на дисковите операции (които са много бавни)
- B-Trees са почти перфектно балансирани
 - Броя на възлите от корена до кой да е **null** възел са едни и същи

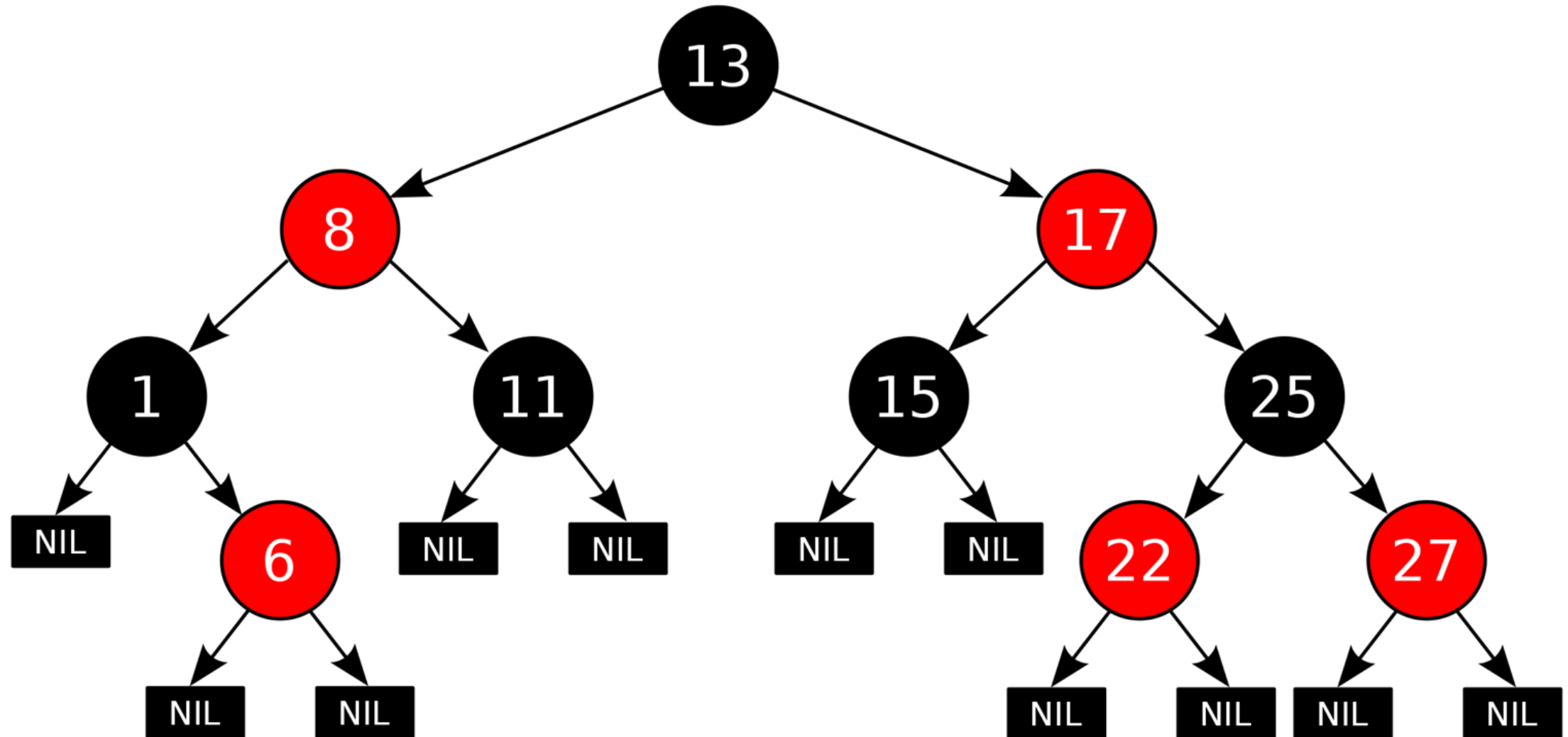


Червено-черни дървета
Семпла репрезентация на няколко дървета

Свойства на червено-черни дървета

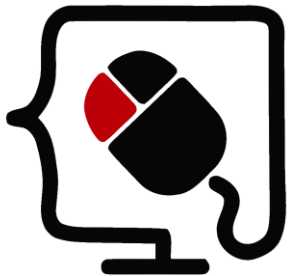
1. Всички листа са черни
2. Корена е черен
3. Няма възел, който да има две червени връзки към него
4. Всеки път от даден възел до листо в негово поддърво има еднакъв брой черни възли
5. Червените възли са винаги от ляво

Пример за червено-черно дърво



Обобщение

- Дървета и дървовидни структури
- Подредени двоични дървета, балансирани дървета, B-дървета
- Упражнения: структура от данни “дърво”, използване на класове и библиотеки за дървовидни структури
- Обхождания в дълбочина и ширина (DFS и BFS)
- Упражнения: обхождане в дълбочина (DFS)
- Упражнения: обхождане в ширина (BFS)



Национална програма
"Обучение за ИТ умения и кариера"
<https://it-kariera.mon.bg>

Министерството на
образованието и науката
<https://www.mon.bg>



Документът е разработен за нуждите на Национална програма "Обучение за ИТ умения и кариера" на Министерството на образованието и науката (МОН) и се разпространява под свободен лиценз CC-BY-NC-SA (Creative Commons Attribution-Non-Commercial-Share-Alike 4.0 International).