4. Деревья

Шустова Л.И.

Деревья

Дерево – множество вершин:

Деревья

Дерево – множество вершин:

- пустое,

Деревья

Дерево – множество вершин:

- пустое,
- состоит из корневой вершины и совокупности связанных с ней поддеревьев.

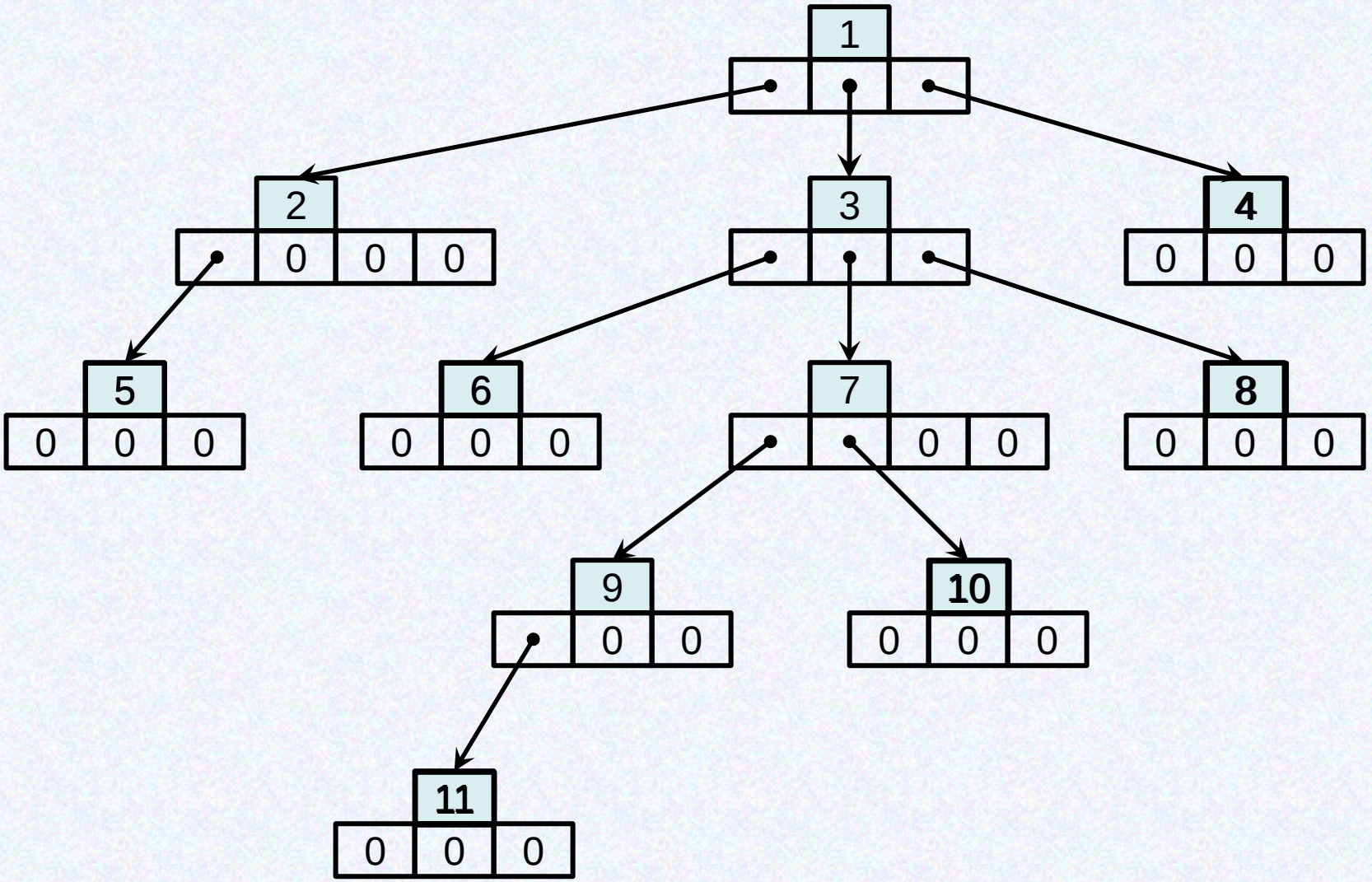
Деревья

Дерево – множество вершин:

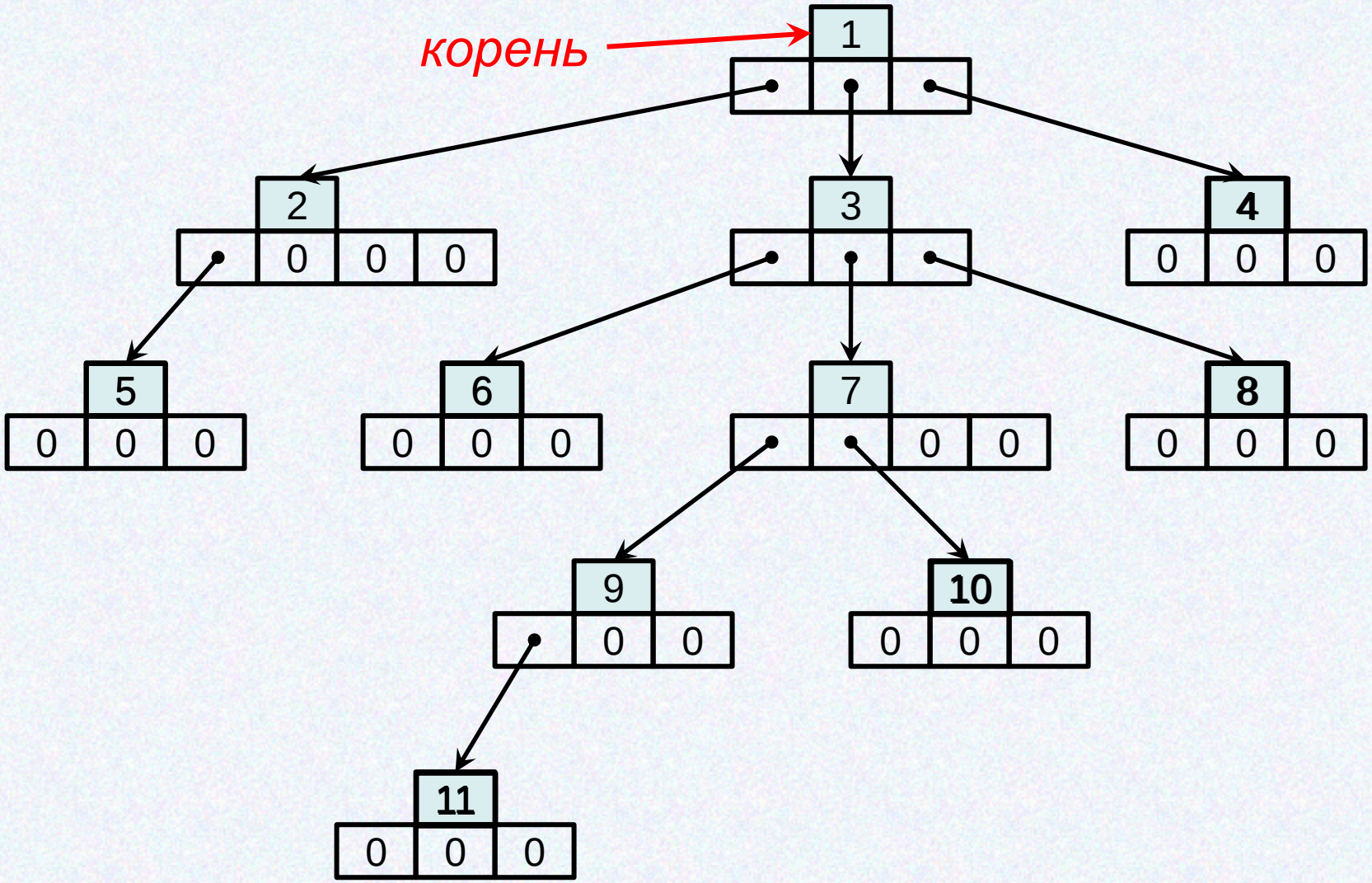
- пустое,
- состоит из корневой вершины и совокупности связанных с ней поддеревьев.

Каждое поддерево – дерево.

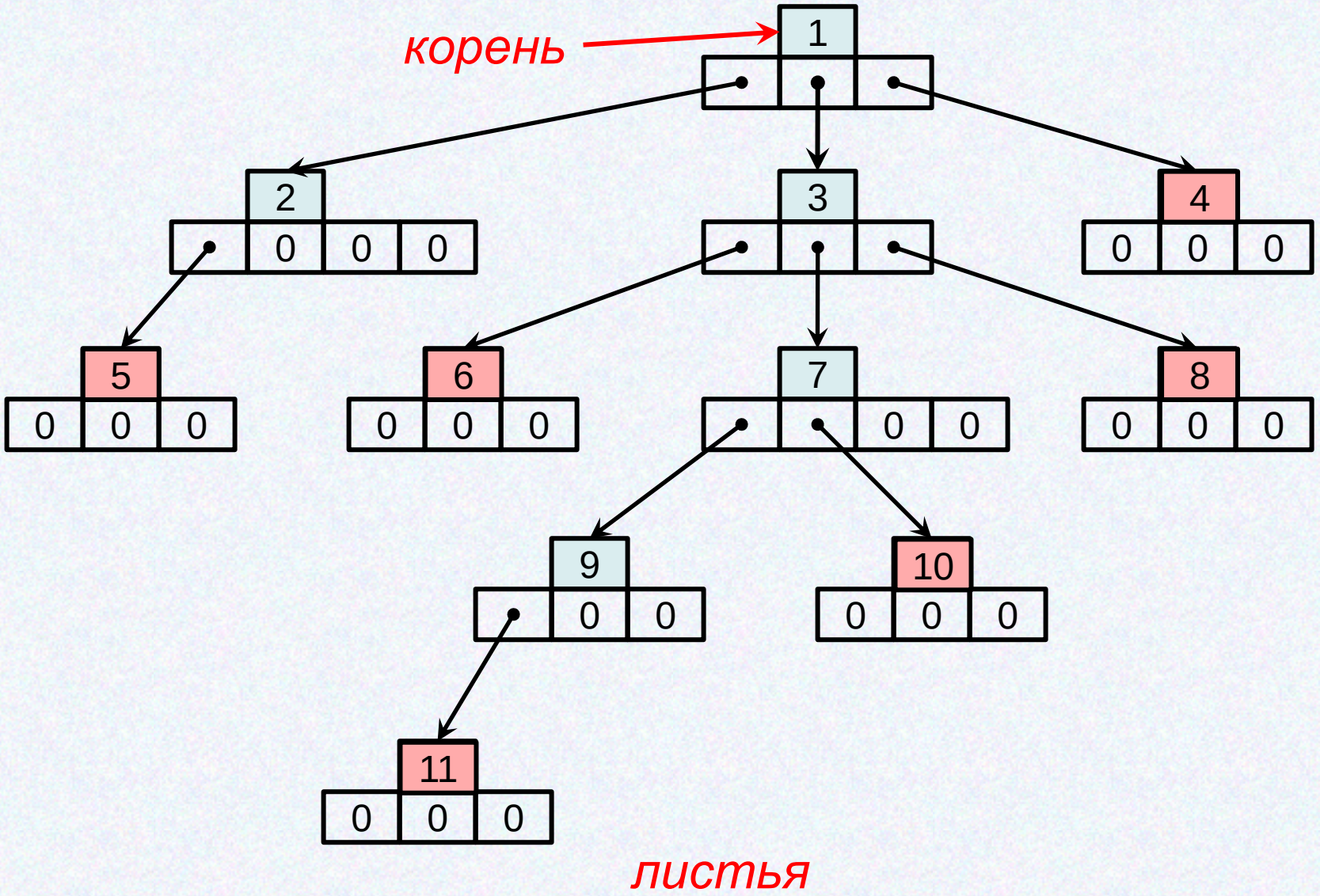
Деревья



Деревья



Деревья


корень
1
2
0 0 0
3
4
0 0 0
5
0 0 0
6
0 0 0
7
0 0
8
0 0 0
9
0 0
10
0 0 0
11
0 0 0
листья

Деревья

```
struct   Node {
   Info  info;          // информация
   int n;               // количество поддеревьев
   Node *ptrs[N];       // массив указателей на поддеревья
};
```

Деревья

```
struct   Node {
   Info  info;        // информация
   int n;             // количество поддеревьев
   Node *ptrs[N]; // массив указателей на поддеревья
};
```

Динамический массив указателей:

```
Node **ptrs;
```

Двоичное дерево

Двоичное дерево – множество вершин:

– пустое,

Двоичное дерево

Двоичное дерево – множество вершин:

- пустое,
- состоит из корневой вершины и двух связанных с ней поддеревьев – левого и правого.

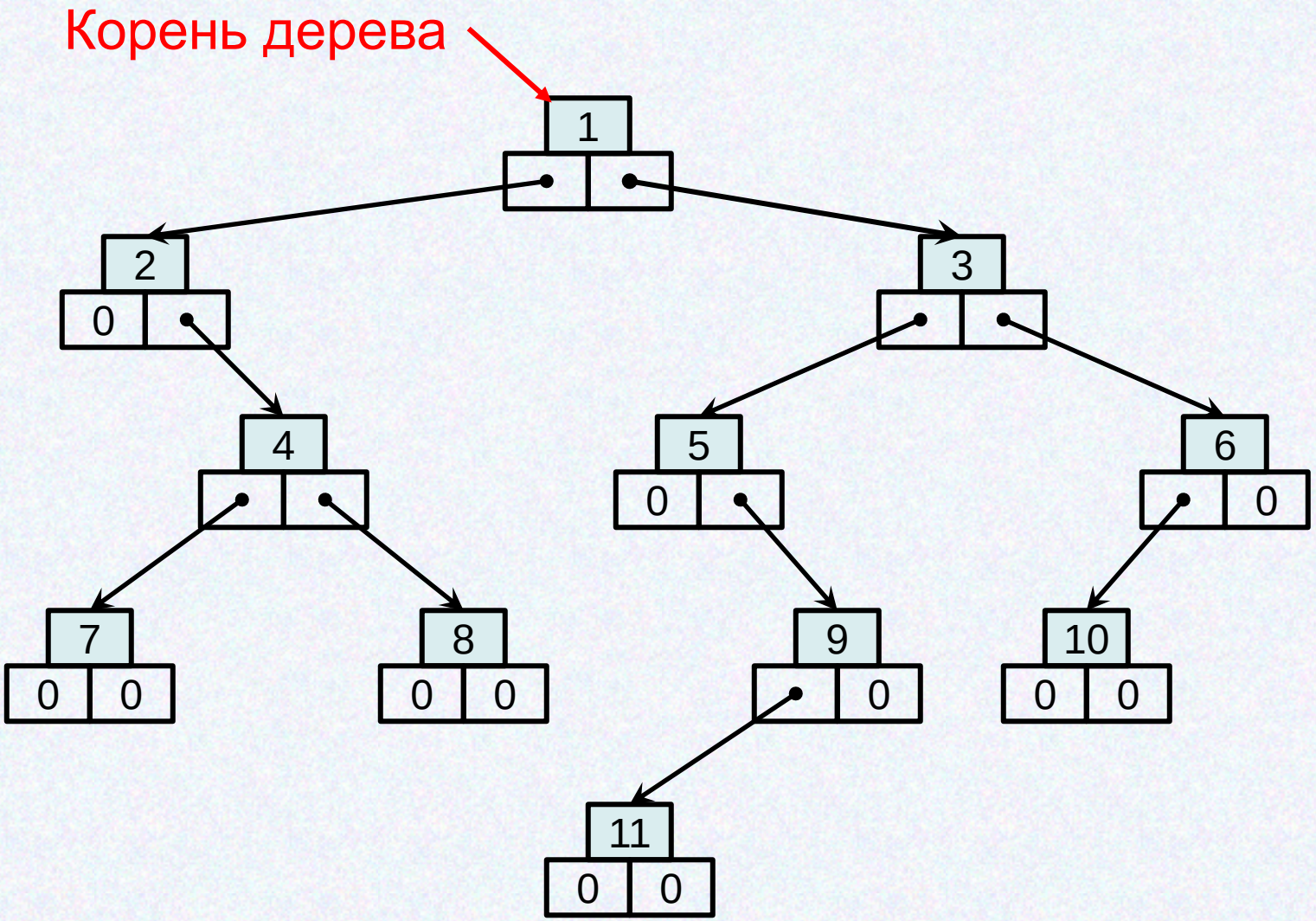
Двоичное дерево

Двоичное дерево – множество вершин:

- пустое,
- состоит из корневой вершины и двух связанных с ней поддеревьев – левого и правого.

Каждое поддерево – двоичное дерево.

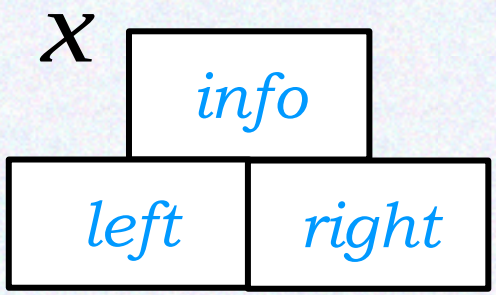
Двоичное дерево



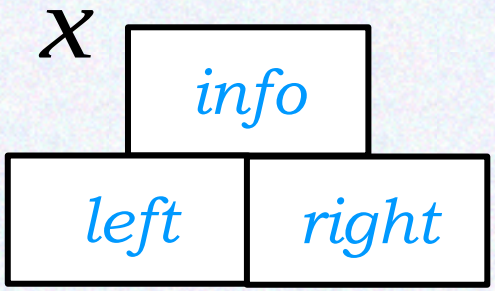
4.6 Представление двоичного дерева

```
struct   Node {
    Info  info;     // информация
    Node *left;   // левое поддерево
    Node *right; // правое поддерево
};
```

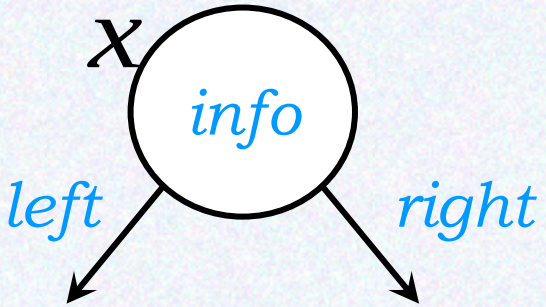
Представление двоичного дерева



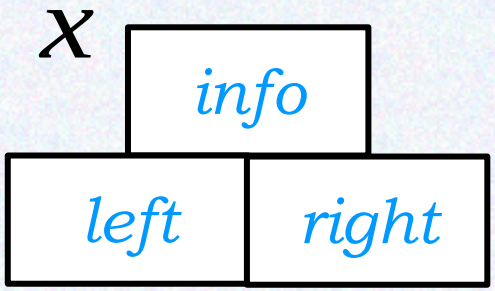
Представление двоичного дерева



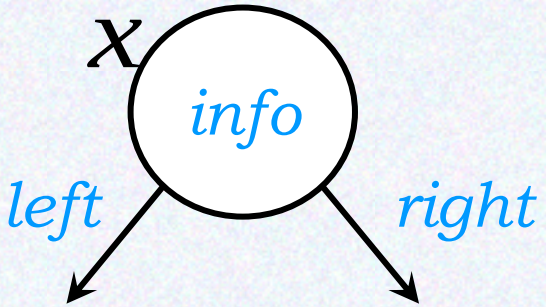
или



Представление двоичного дерева

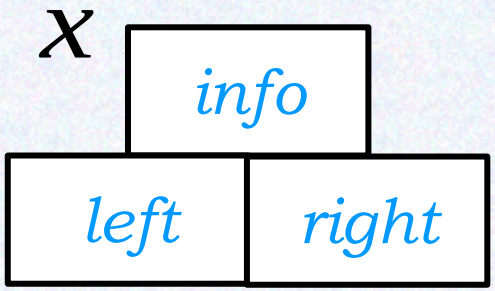


или

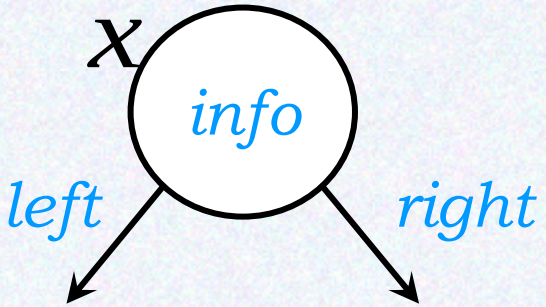


root – указатель на корень дерева

Представление двоичного дерева



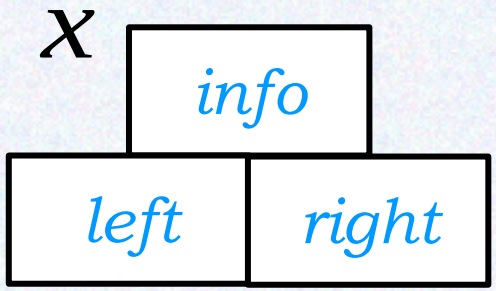
или



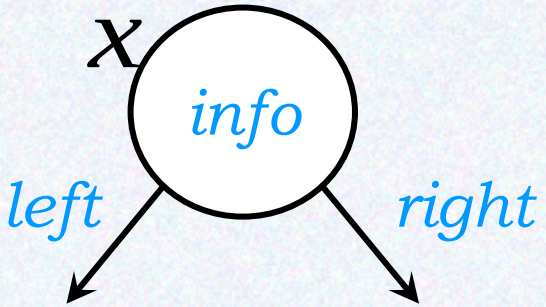
root – указатель на корень дерева

ptr – указатель на любую вершину дерева

Представление двоичного дерева



или



root – указатель на корень дерева

ptr – указатель на любую вершину дерева

Доступ к полям вершины:
ptr->info, ptr->left, ptr->right

4.8

Представление обычного дерева с помощью двоичного

4.8

Представление обычного дерева с помощью двоичного

- Каждый узел x содержит два указателя

4.8

Представление обычного дерева с помощью двоичного

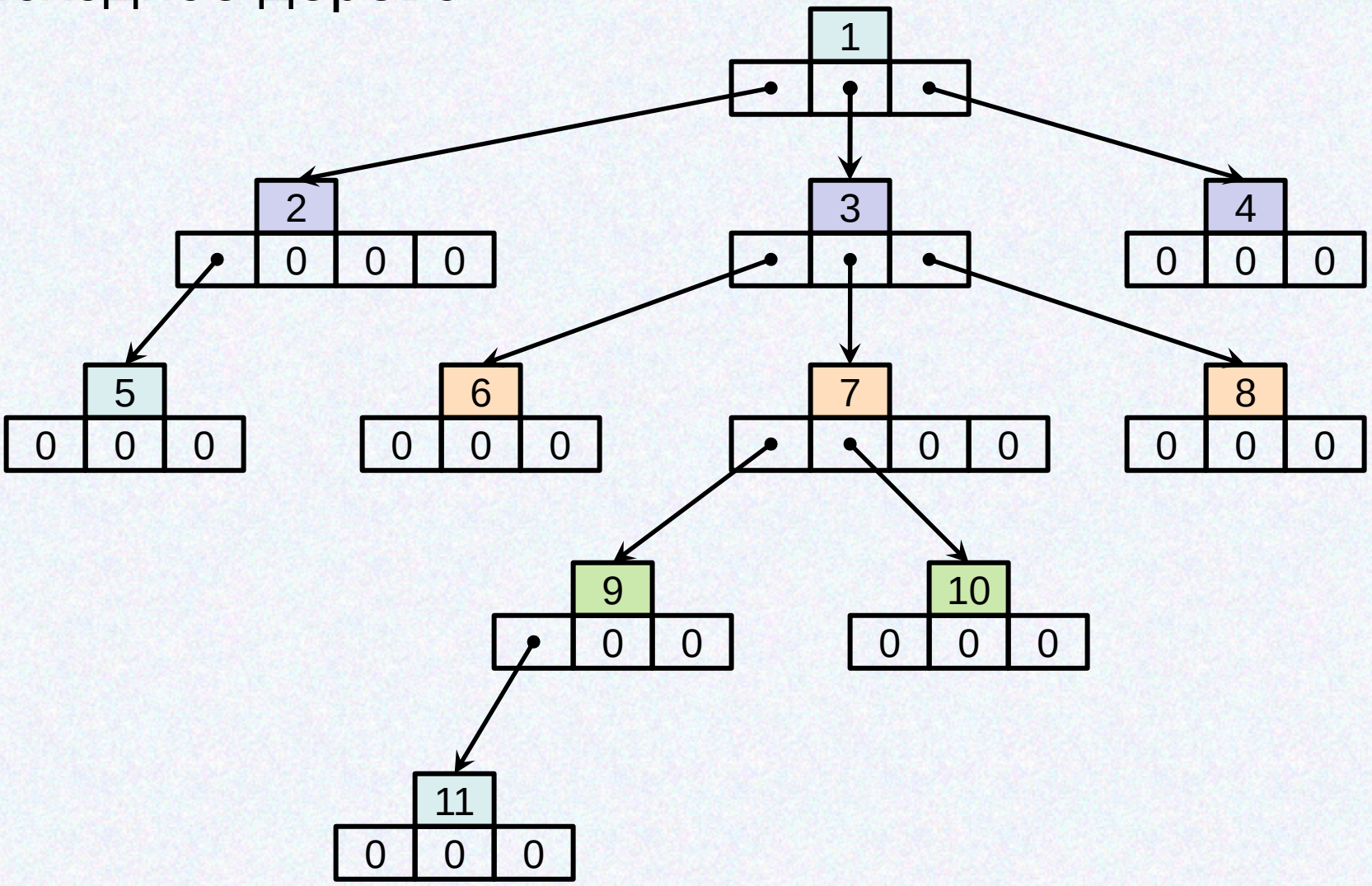
- Каждый узел x содержит два указателя
- Поле $left$ узла x содержит указатель на самый левый дочерний узел для данного узла

4.8 Представление обычного дерева с помощью двоичного

- Каждый узел x содержит два указателя
- Поле $left$ узла x содержит указатель на самый левый дочерний узел для данного узла
- Поле $right$ узла x содержит указатель на соседний дочерний узел для его родительского узла

Пример

Исходное дерево

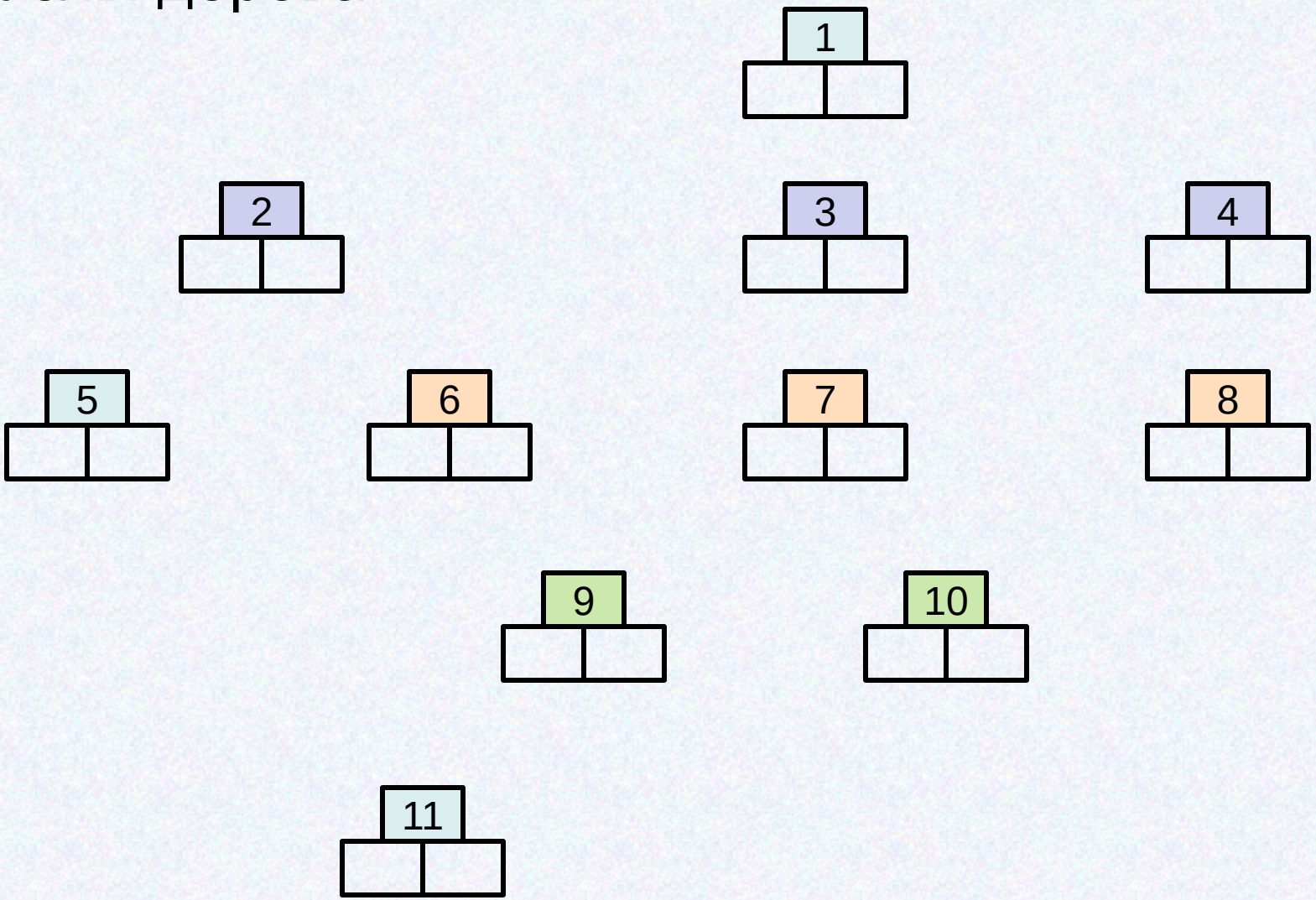


Пример

1. Узлы дерева

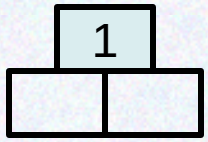
Пример

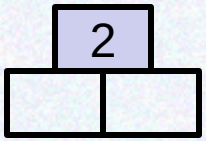
1. Узлы дерева

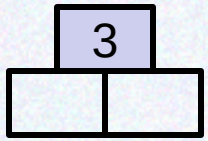


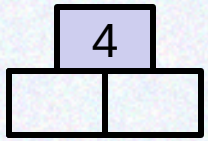
Пример

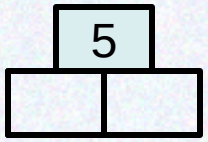
1. Узлы дерева
2. Поле *left*

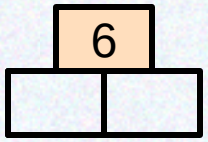


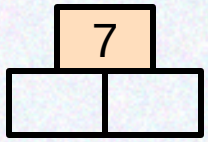


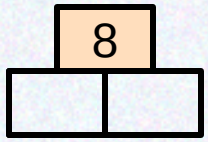


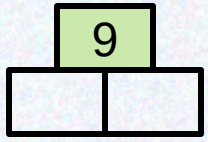


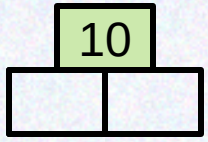


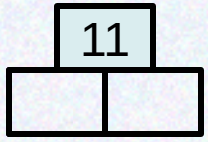






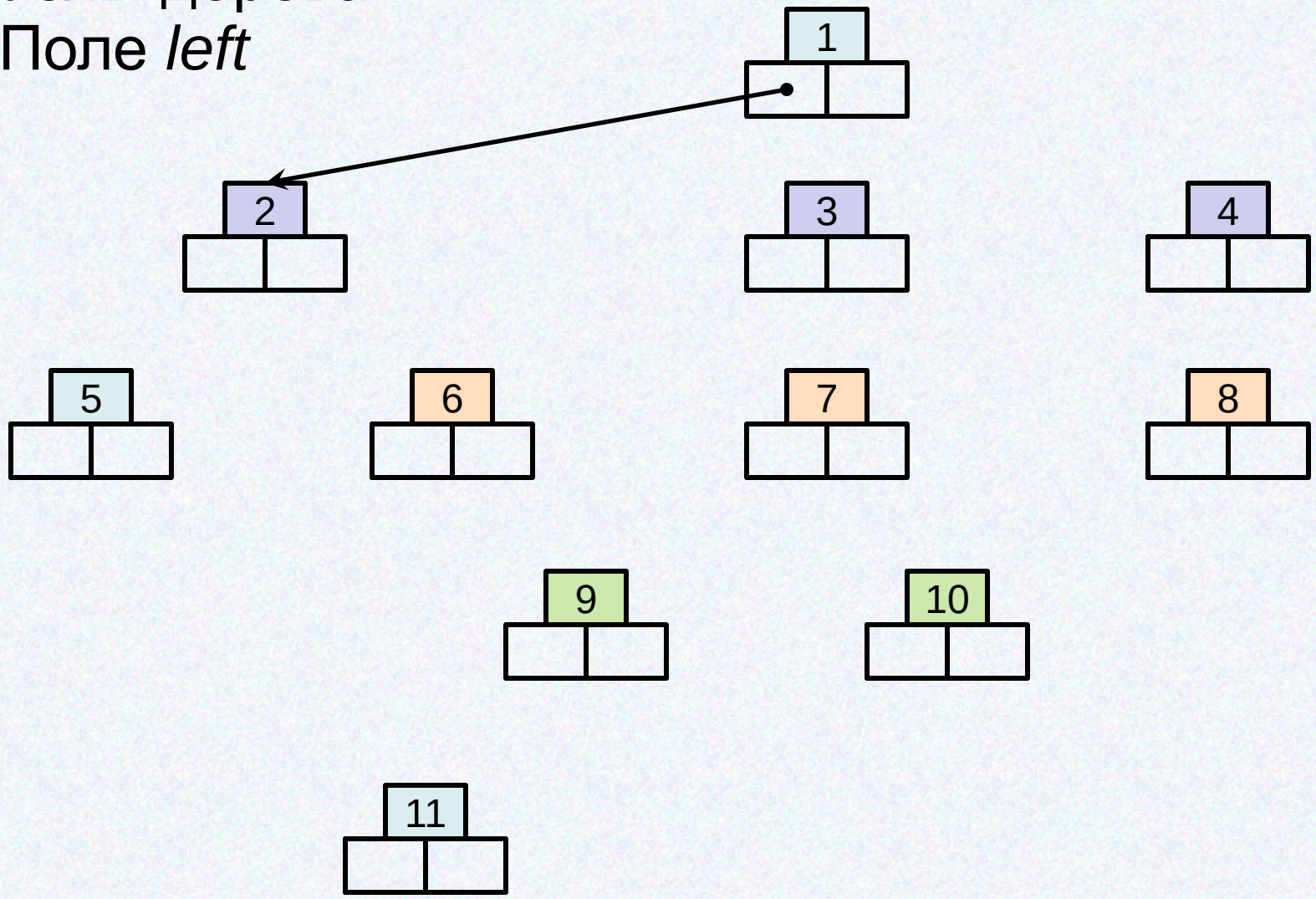






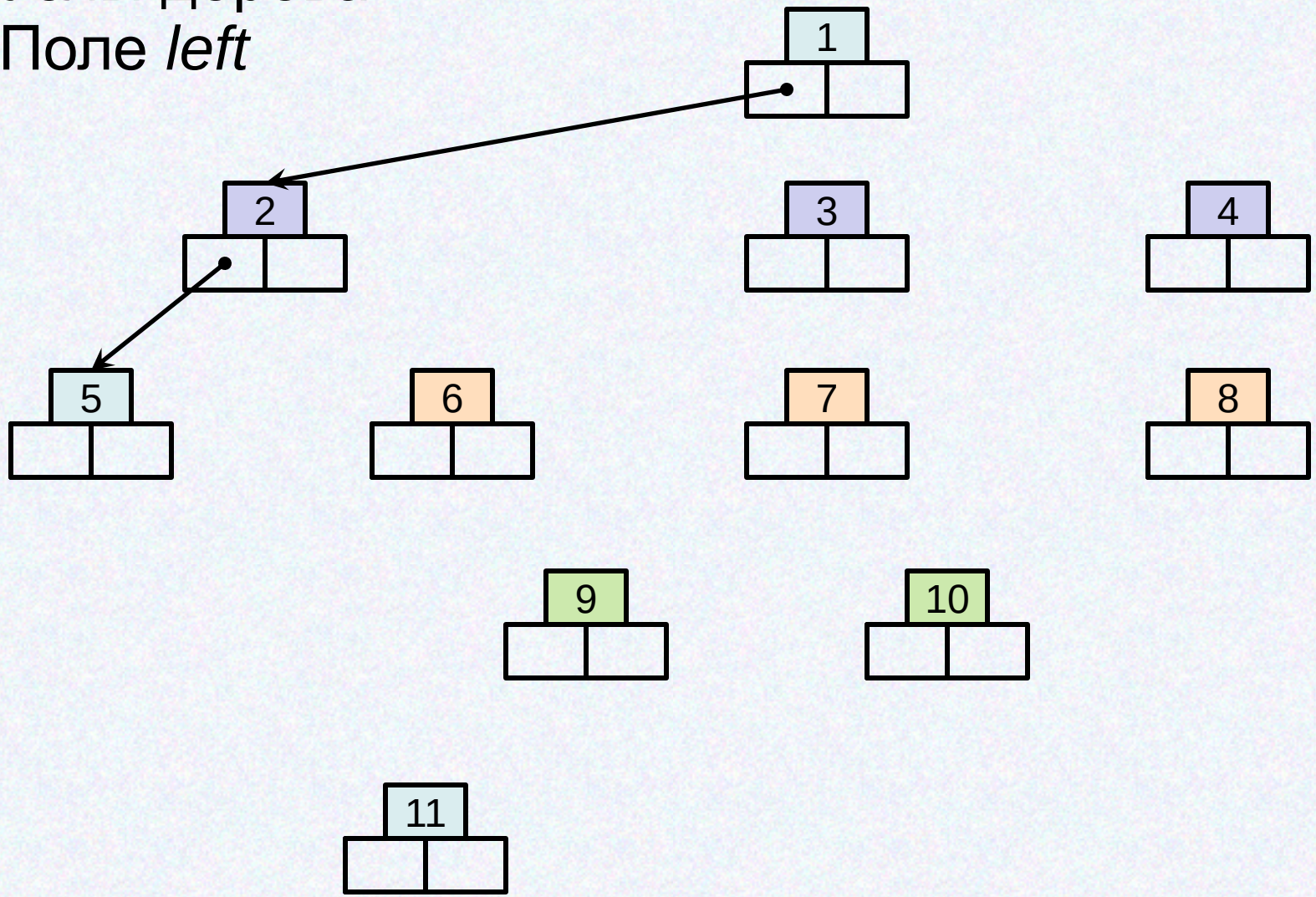
Пример

1. Узлы дерева
2. Поле *left*



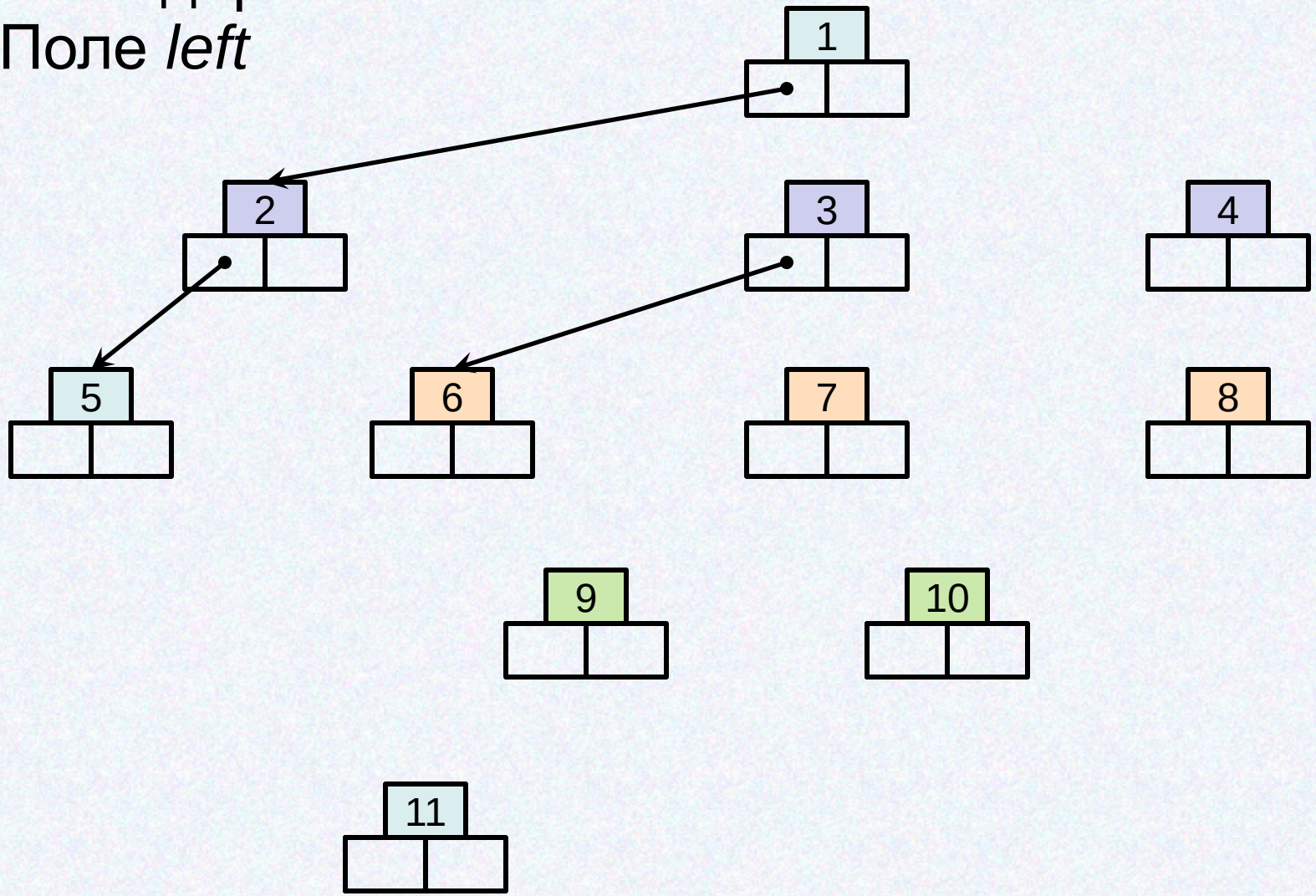
Пример

1. Узлы дерева
2. Поле *left*



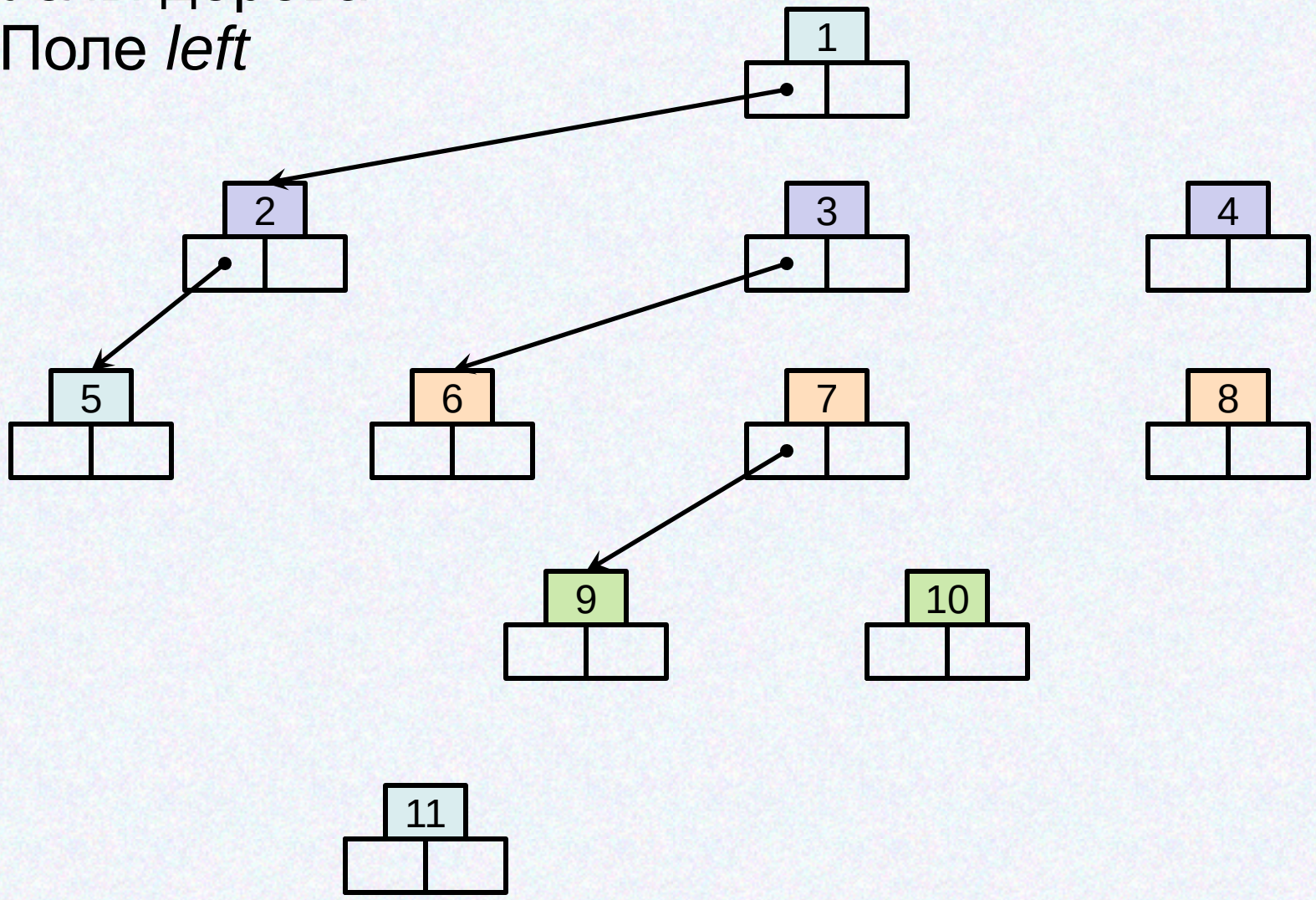
Пример

1. Узлы дерева
2. Поле *left*



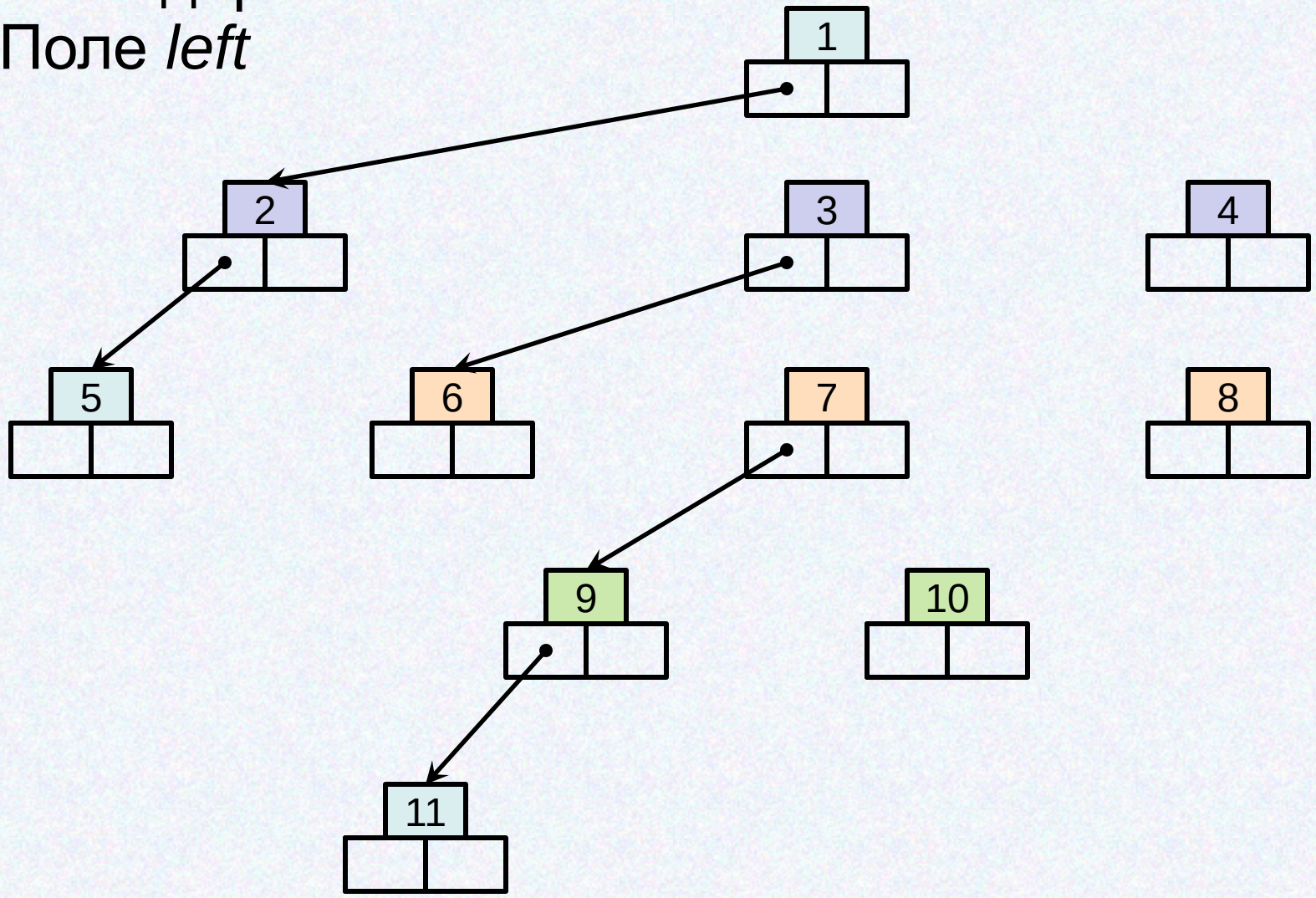
Пример

1. Узлы дерева
2. Поле *left*



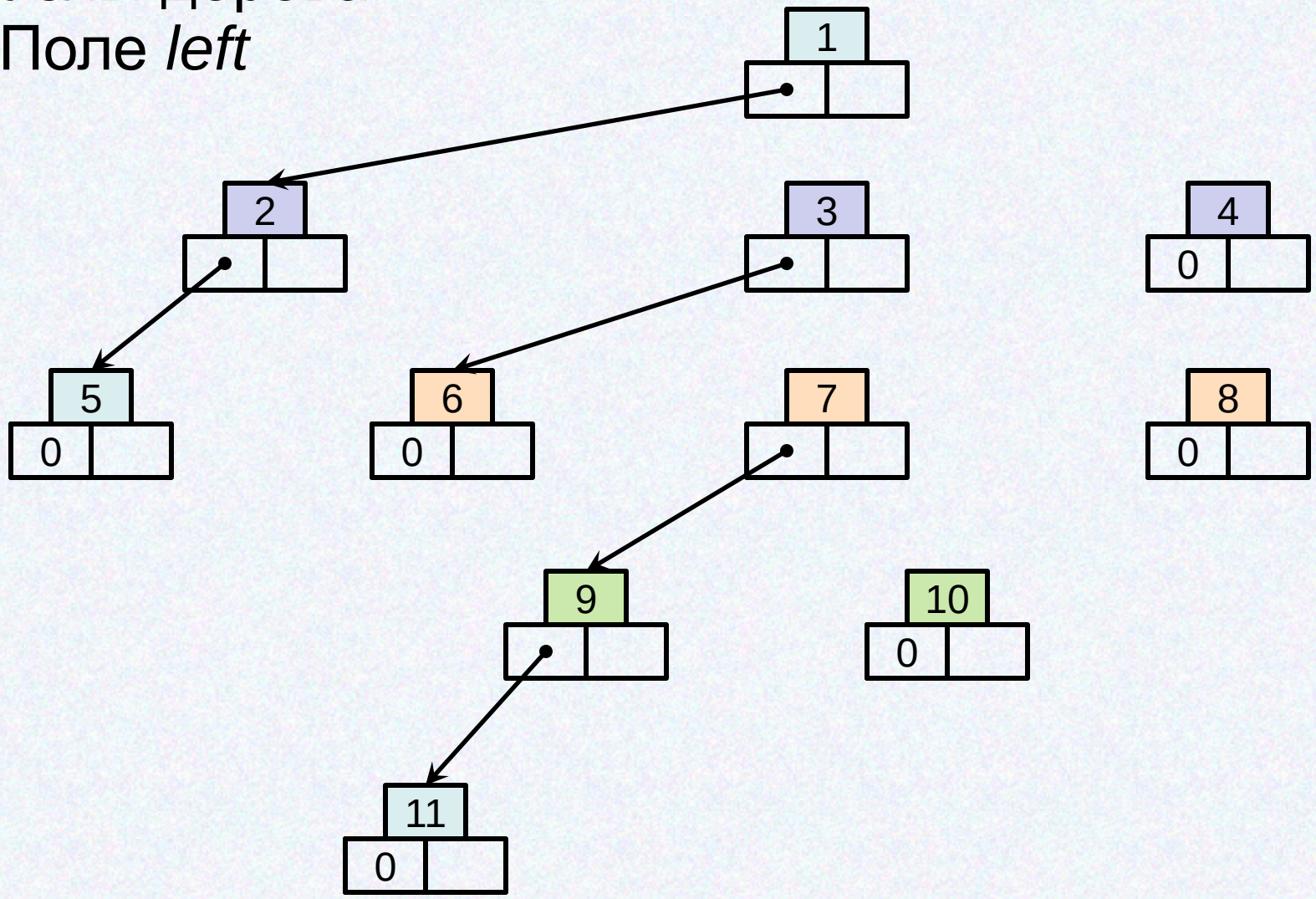
Пример

1. Узлы дерева
2. Поле *left*



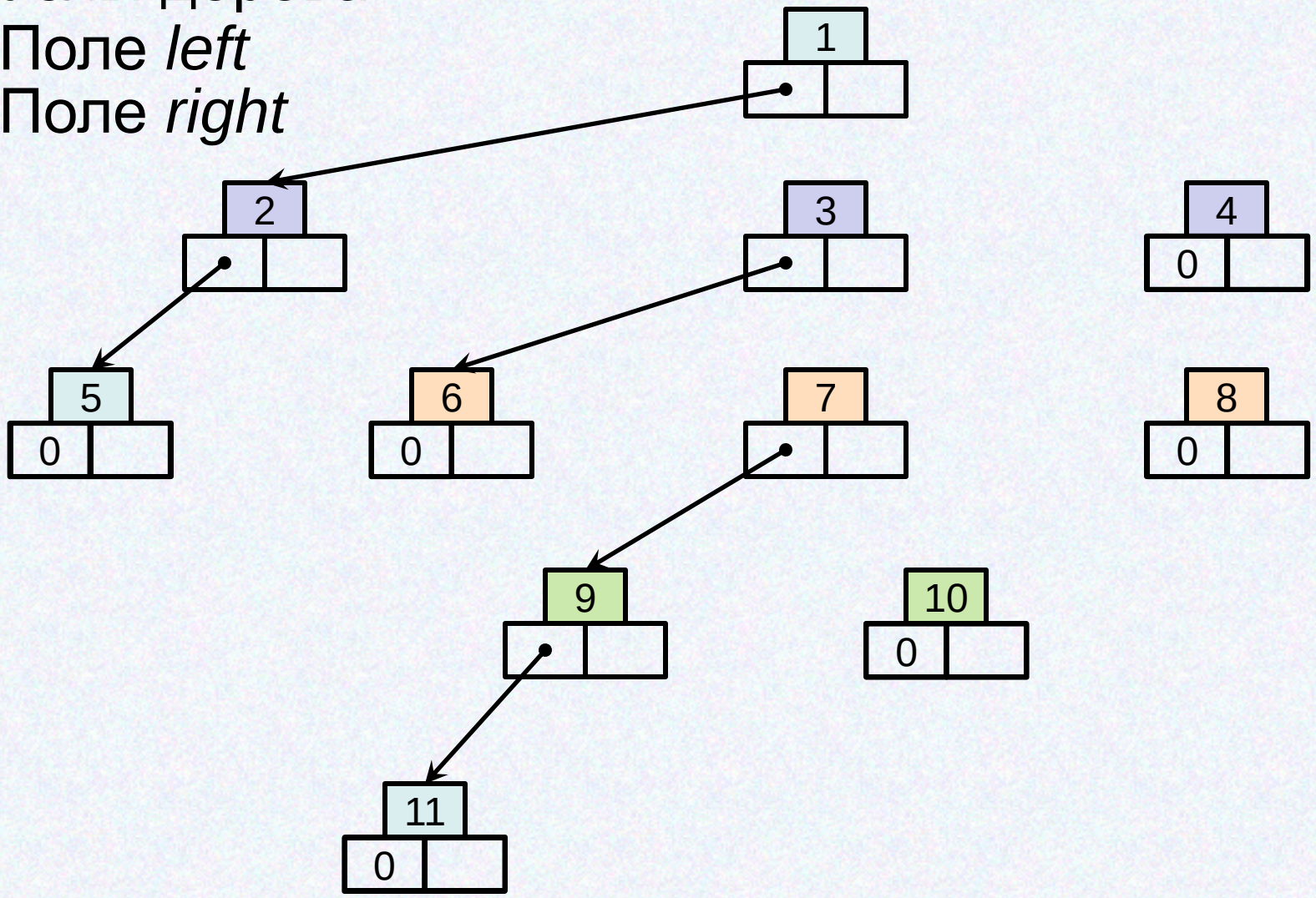
Пример

1. Узлы дерева
2. Поле *left*



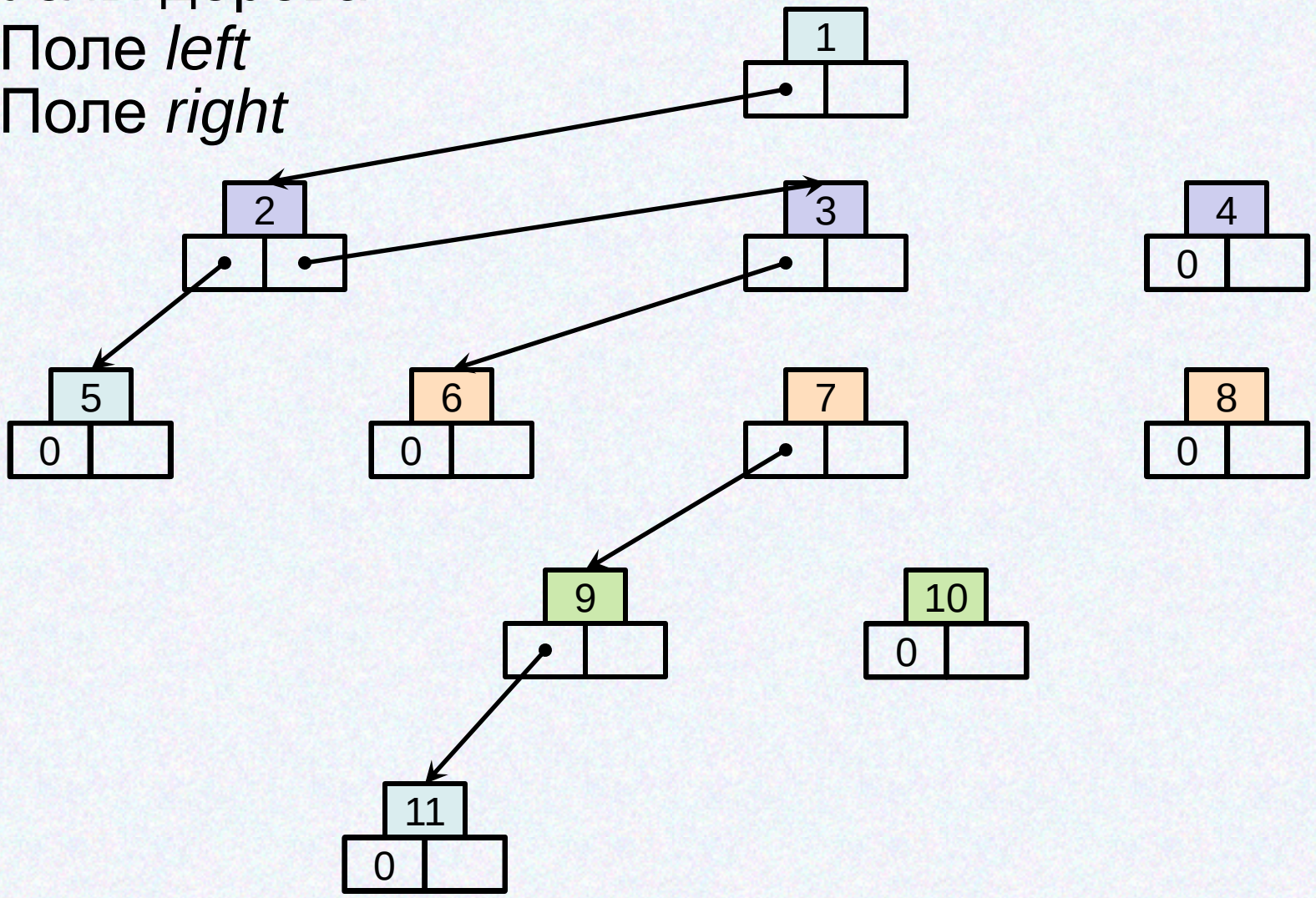
Пример

1. Узлы дерева
2. Поле *left*
3. Поле *right*



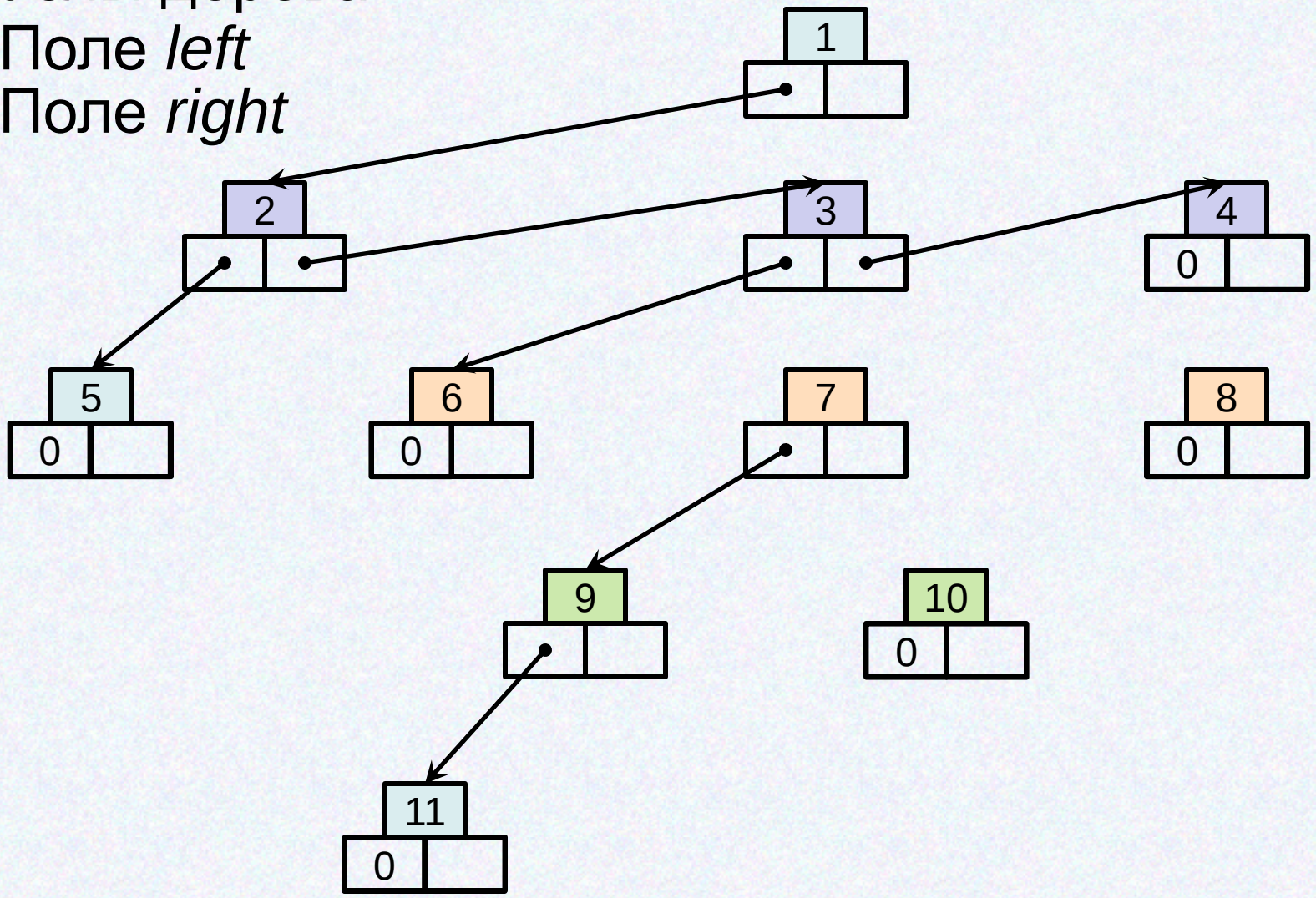
Пример

1. Узлы дерева
2. Поле *left*
3. Поле *right*



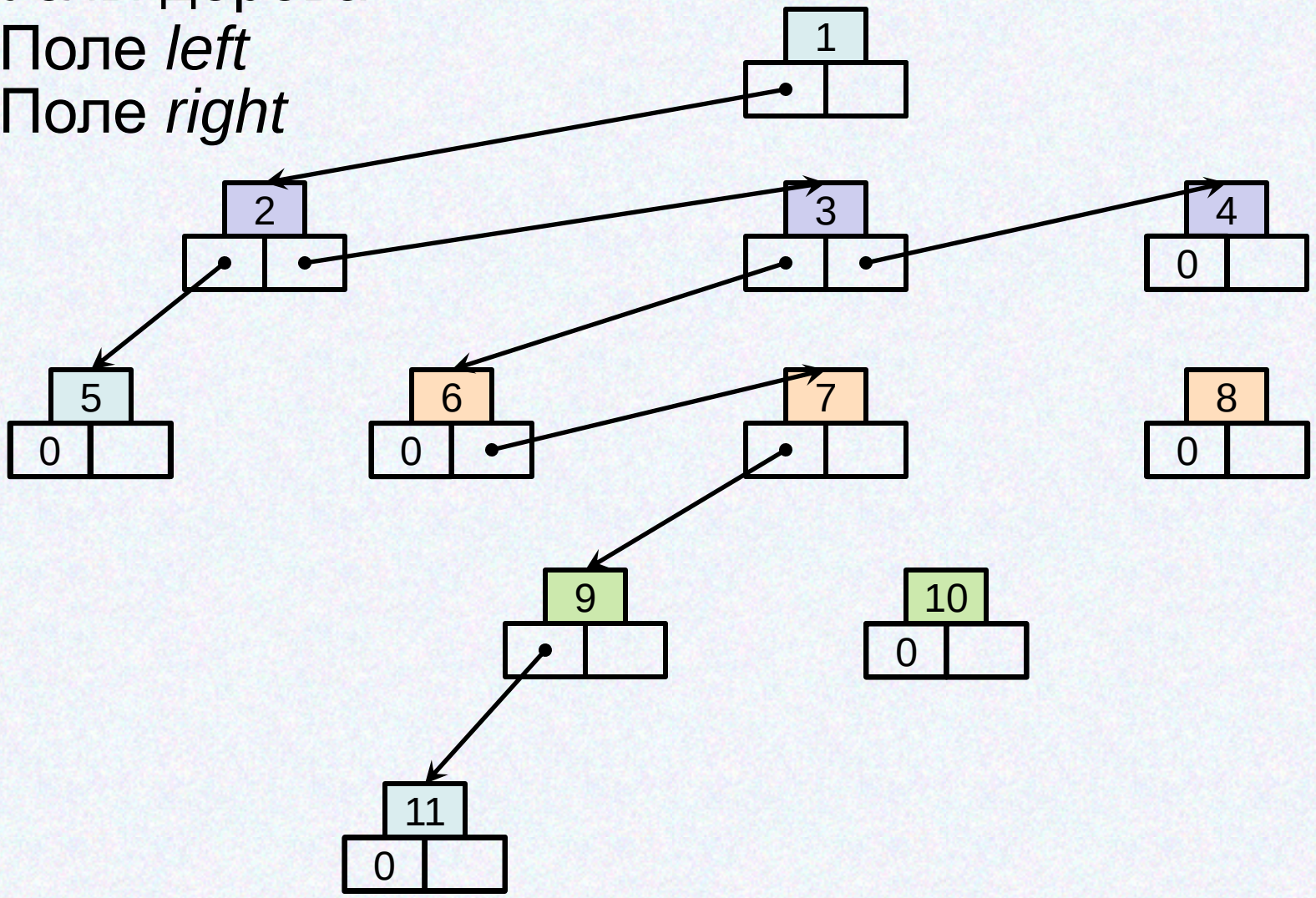
Пример

1. Узлы дерева
2. Поле *left*
3. Поле *right*



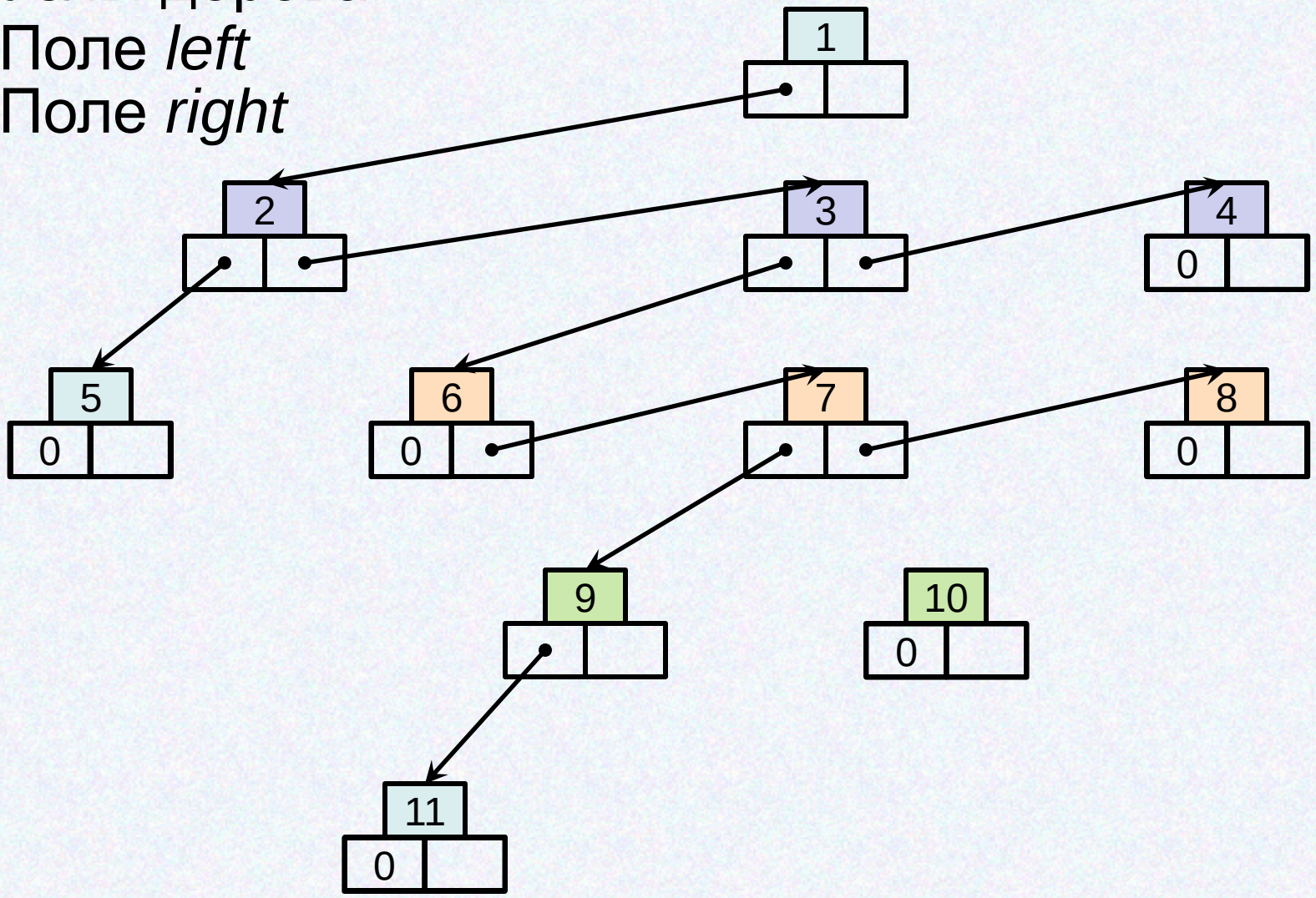
Пример

1. Узлы дерева
2. Поле *left*
3. Поле *right*



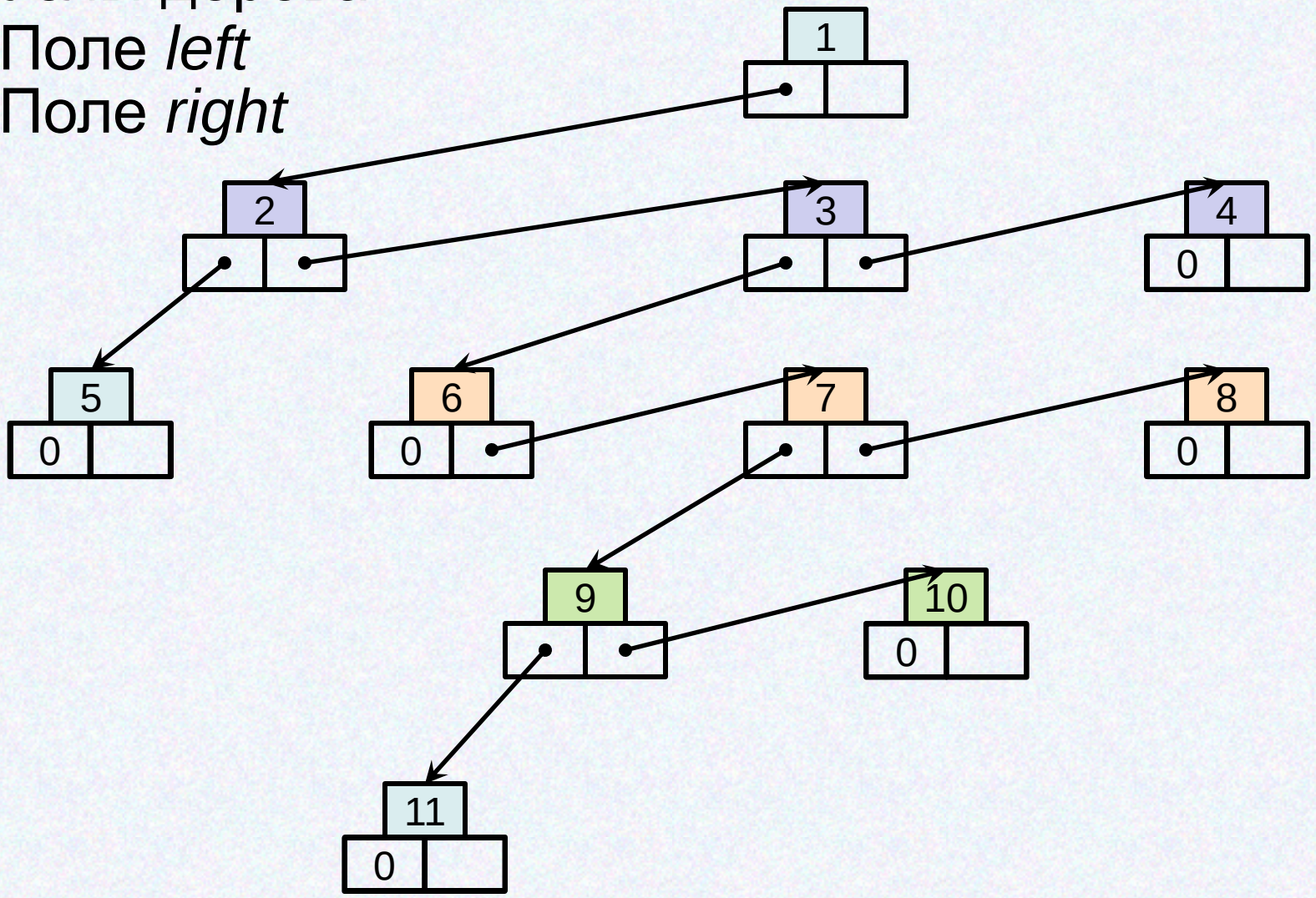
Пример

1. Узлы дерева
2. Поле *left*
3. Поле *right*



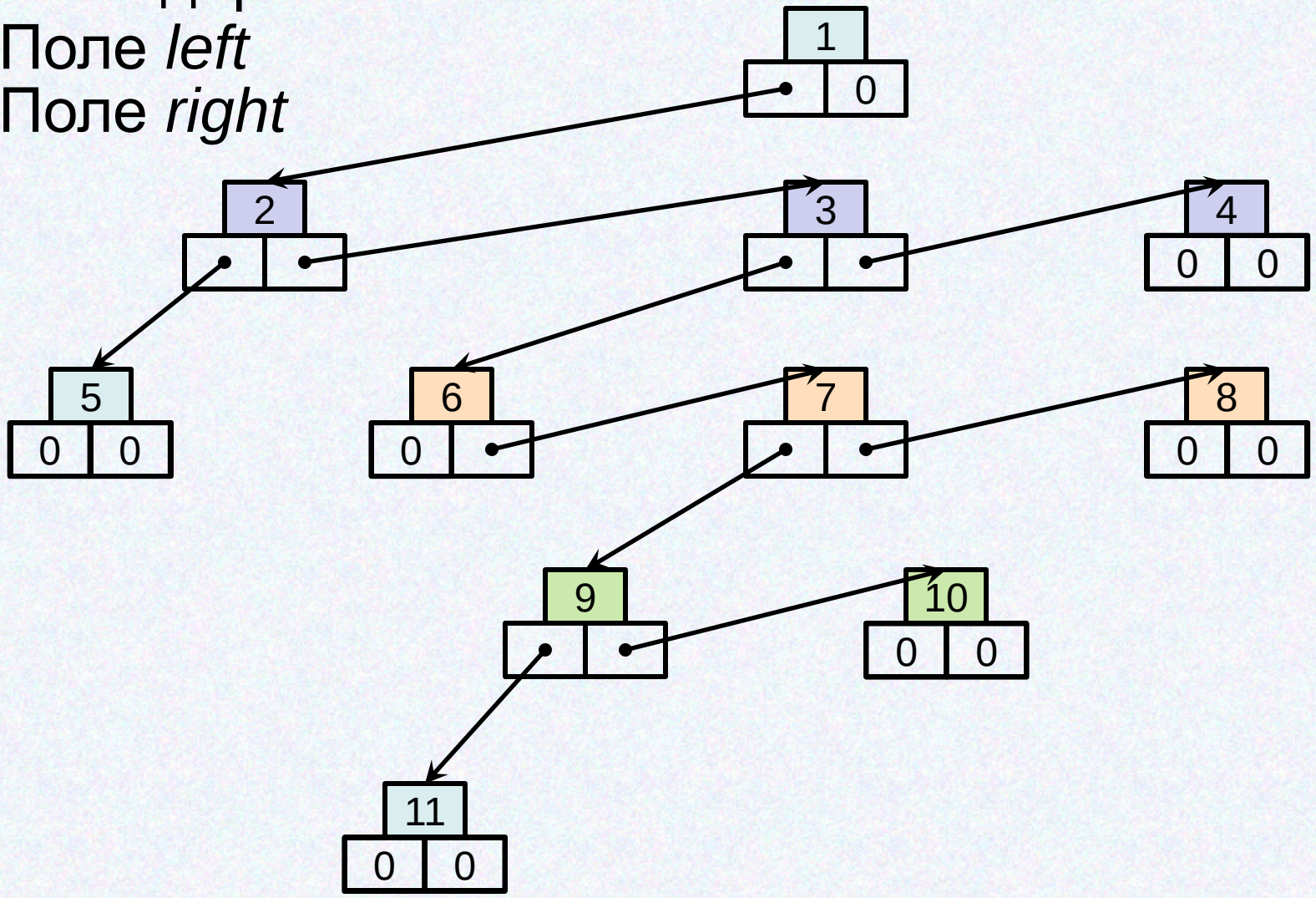
Пример

1. Узлы дерева
2. Поле *left*
3. Поле *right*

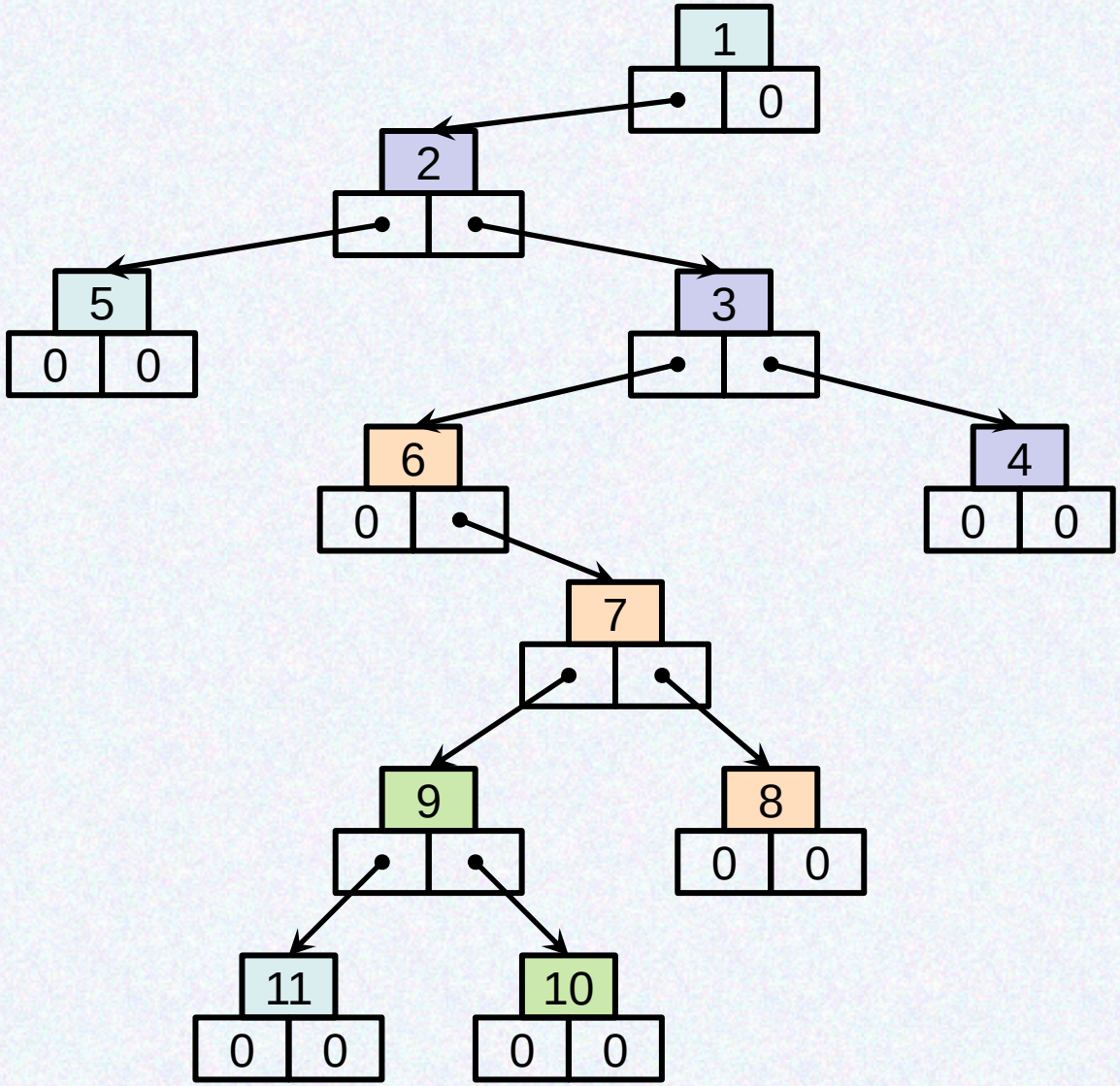


Пример

1. Узлы дерева
2. Поле *left*
3. Поле *right*



Пример



Обходы дерева

Обходы дерева

Обход дерева – операция, при которой все вершины дерева обрабатываются в некотором порядке, причем:

Обходы дерева

Обход дерева – операция, при которой все вершины дерева обрабатываются в некотором порядке, причем:

- ни одна вершина не будет пропущена,

Обходы дерева

Обход дерева – операция, при которой все вершины дерева обрабатываются в некотором порядке, причем:

- ни одна вершина не будет пропущена,
- ни одна вершина не будет обработана дважды

Обходы дерева

- *Прямой обход*

Обходы дерева

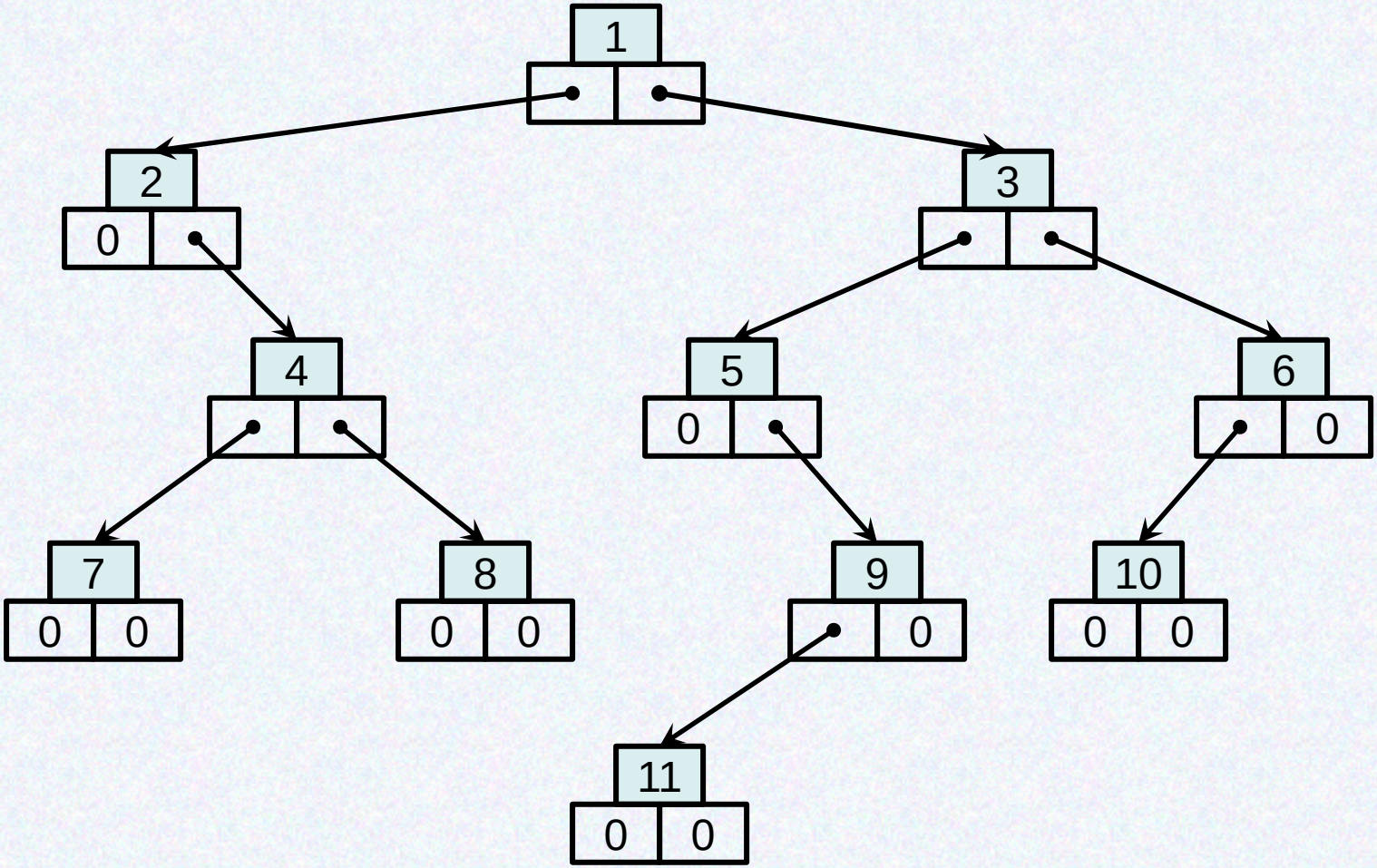
- *Прямой обход*
  - корень, левое поддерево, правое поддерево

Обходы дерева

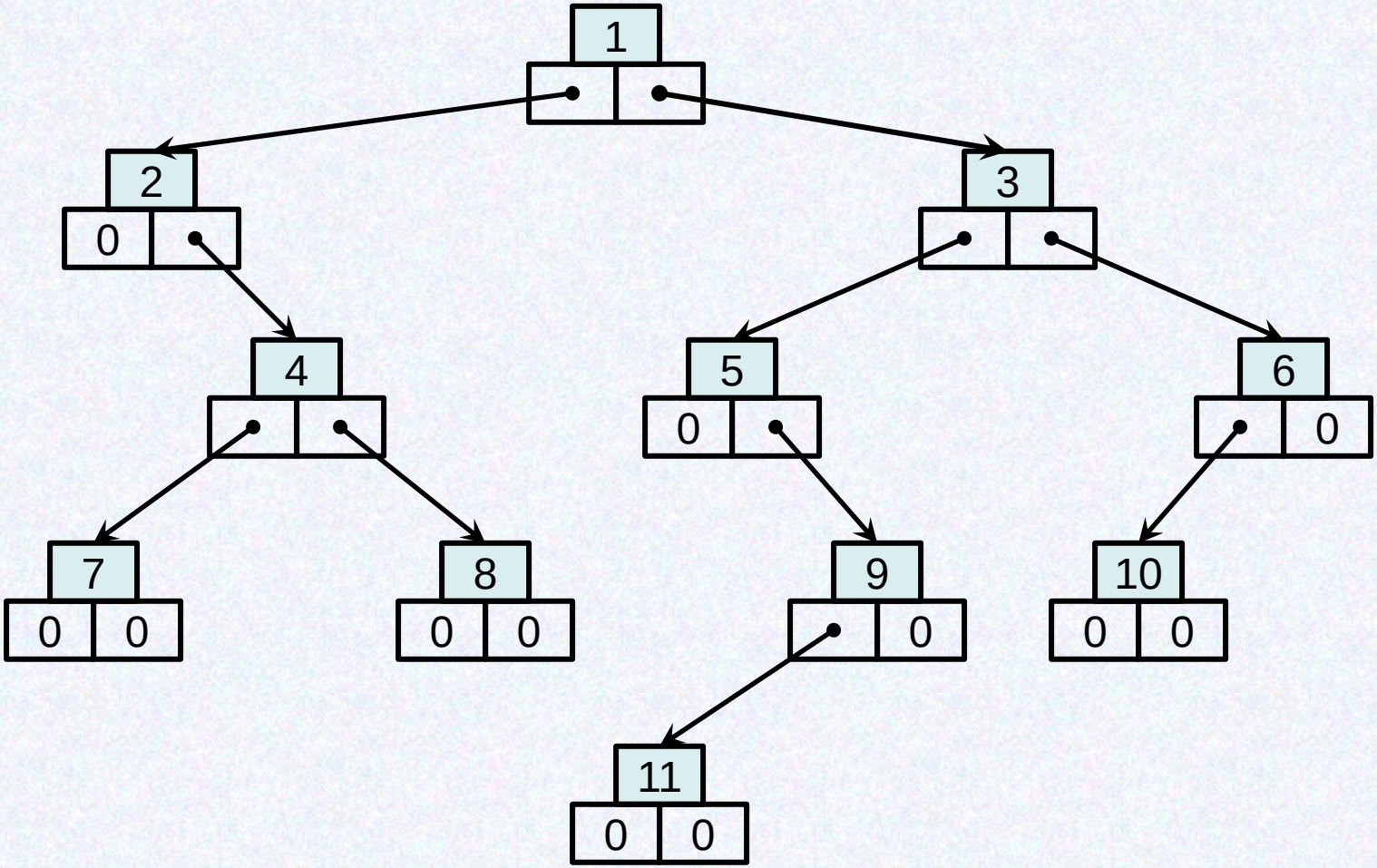
- *Прямой обход*
  - – корень, левое поддерево, правое поддерево

Каждое поддерево обрабатывается в таком же порядке

Обходы дерева

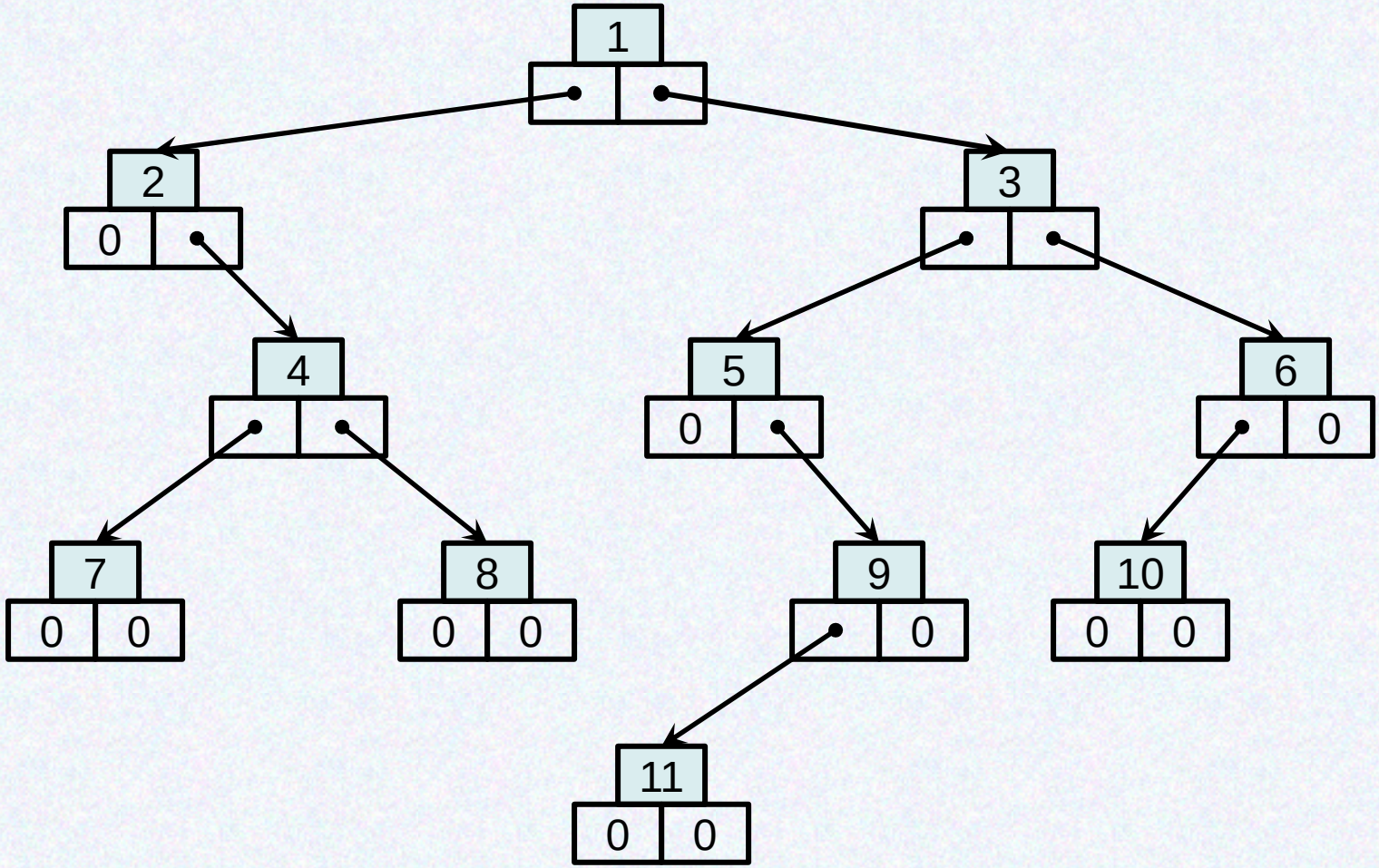


Обходы дерева



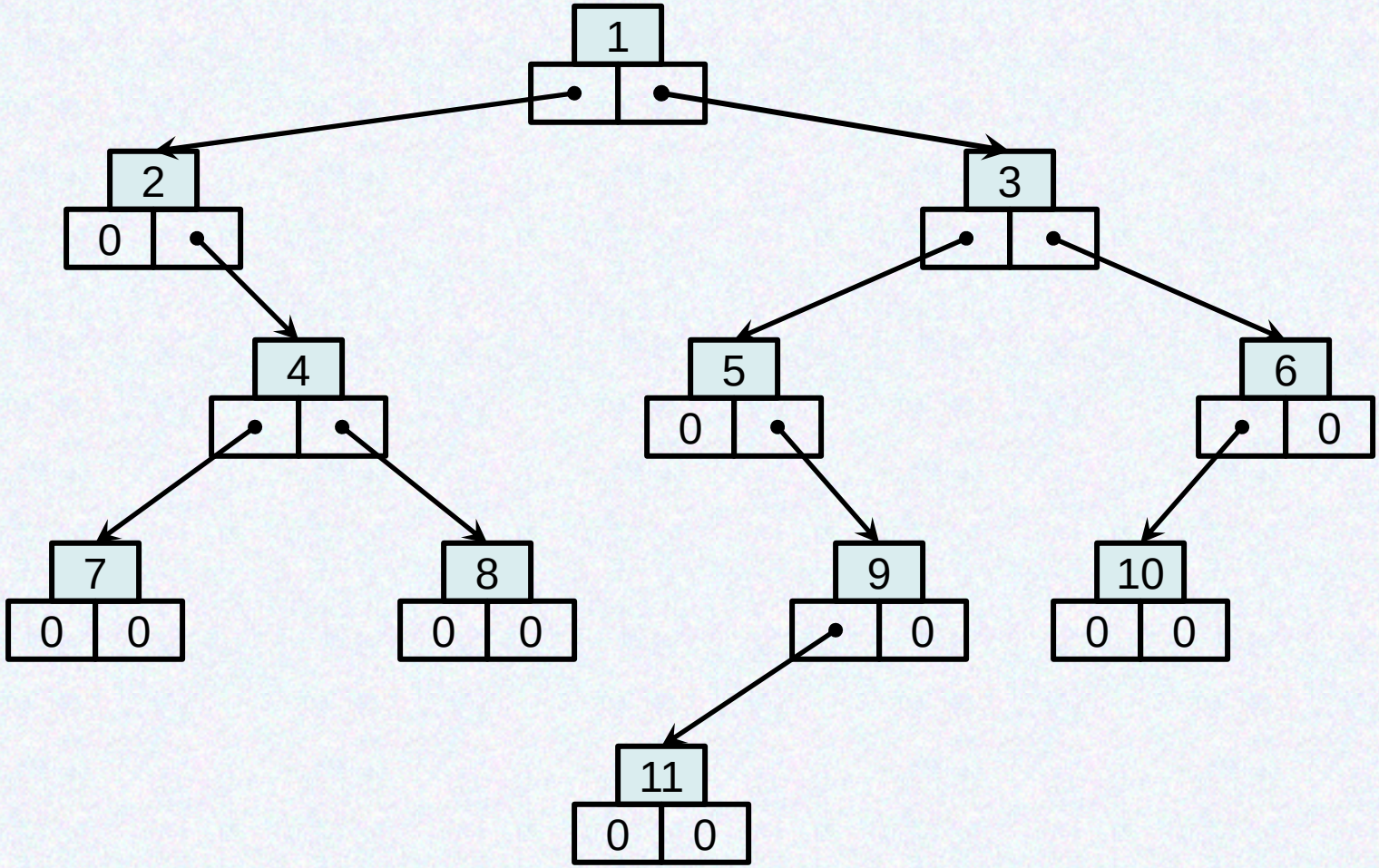
1

Обходы дерева



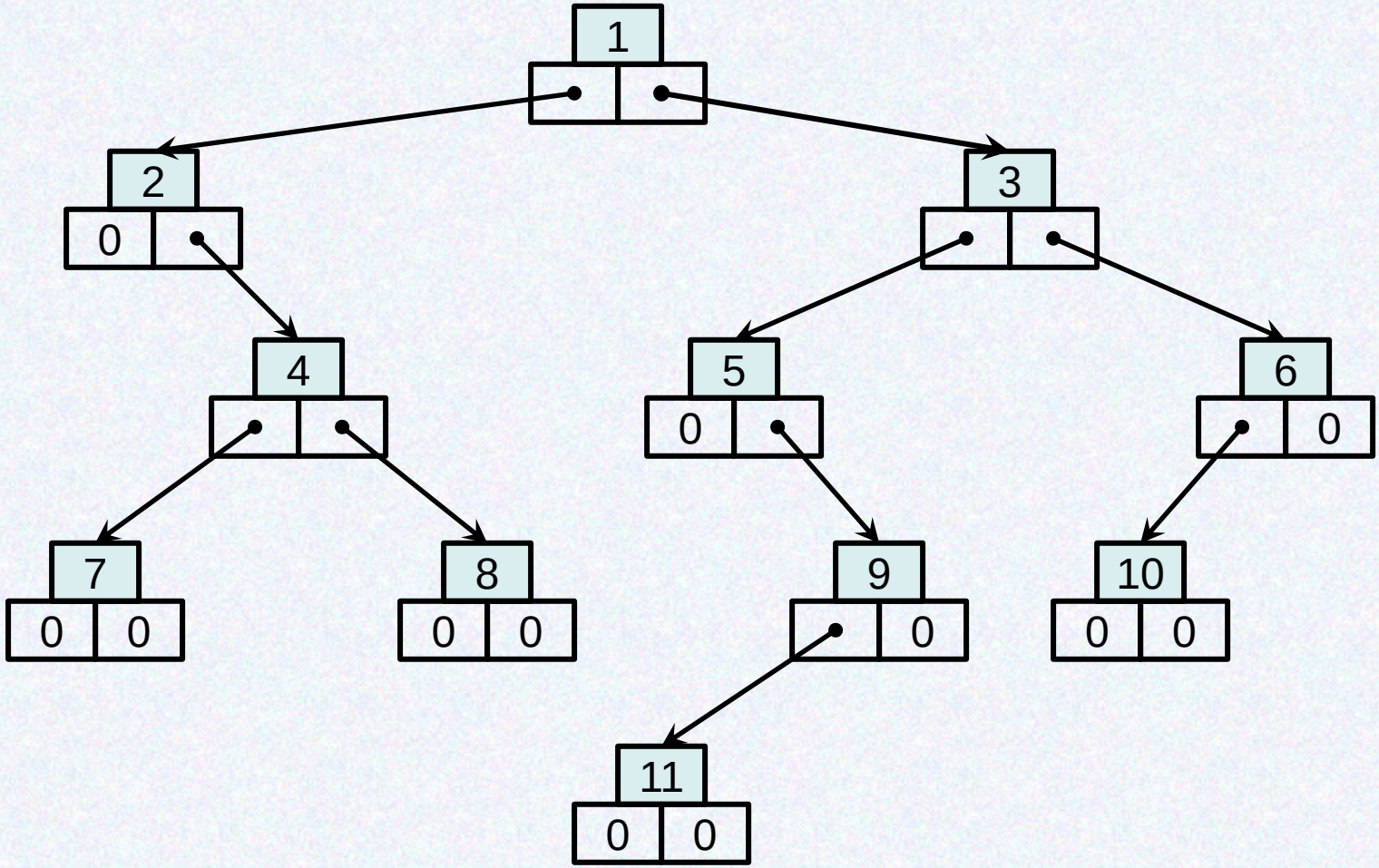
1 2

Обходы дерева



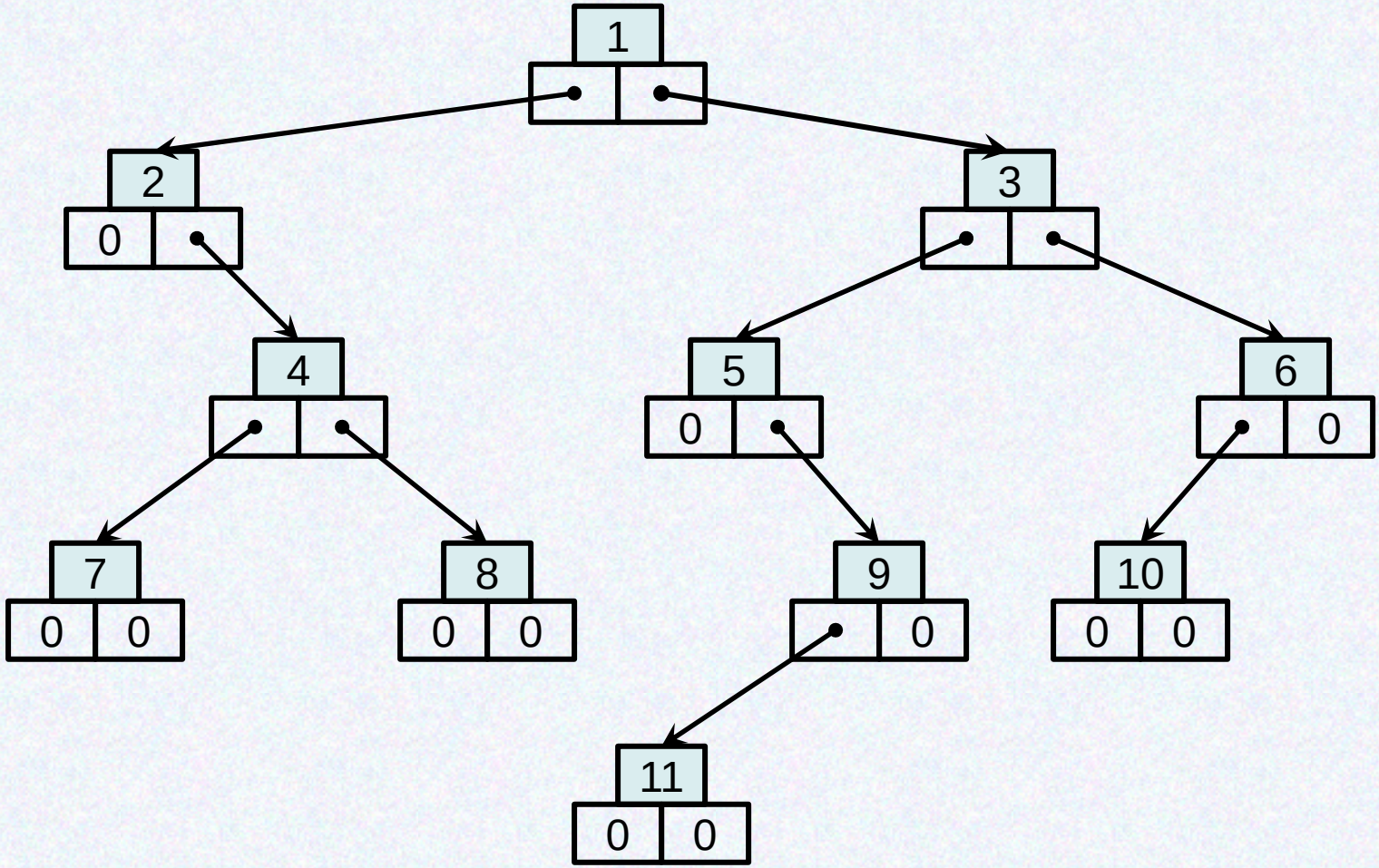
1 2 4

Обходы дерева



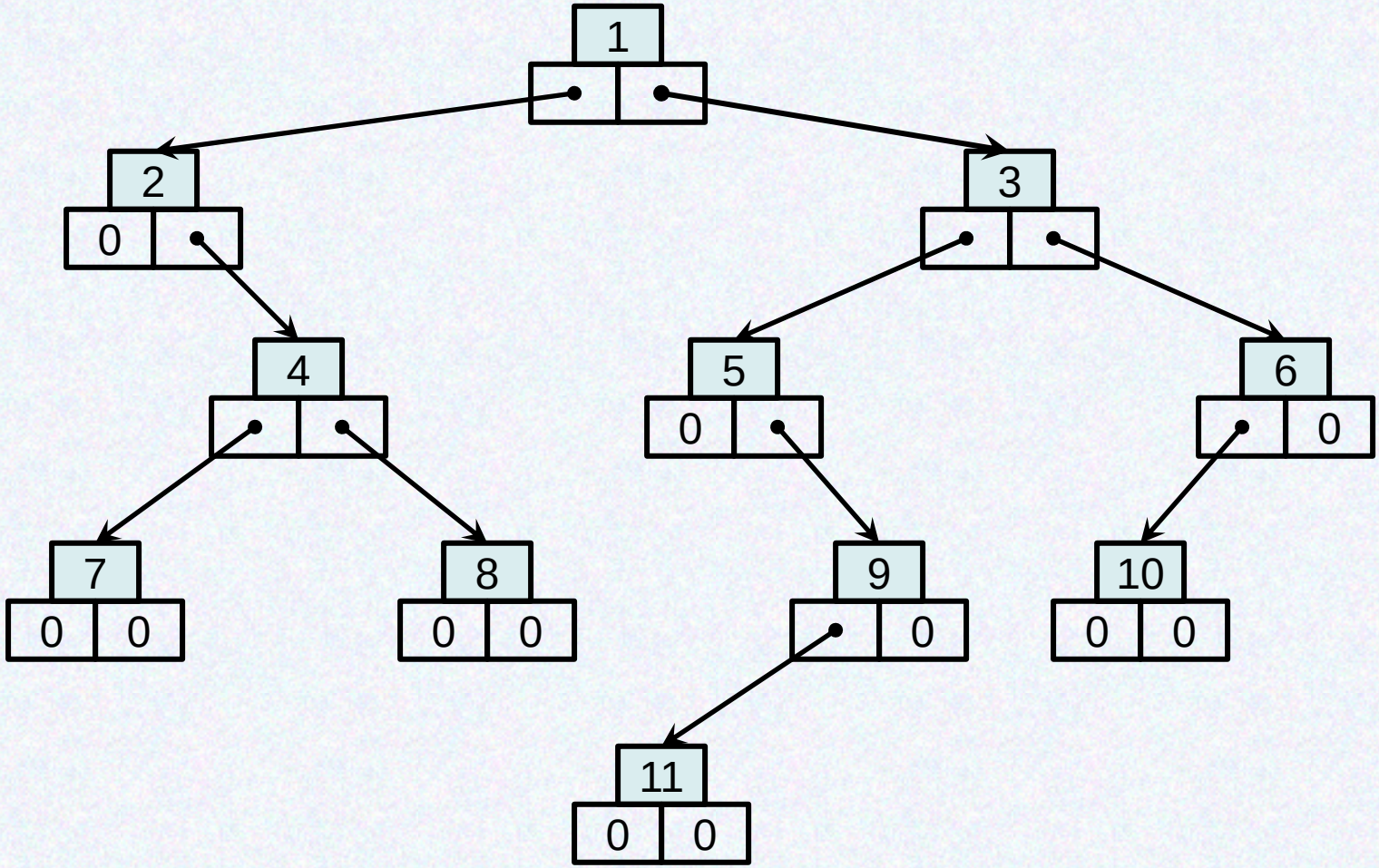
1 2 4 7

Обходы дерева



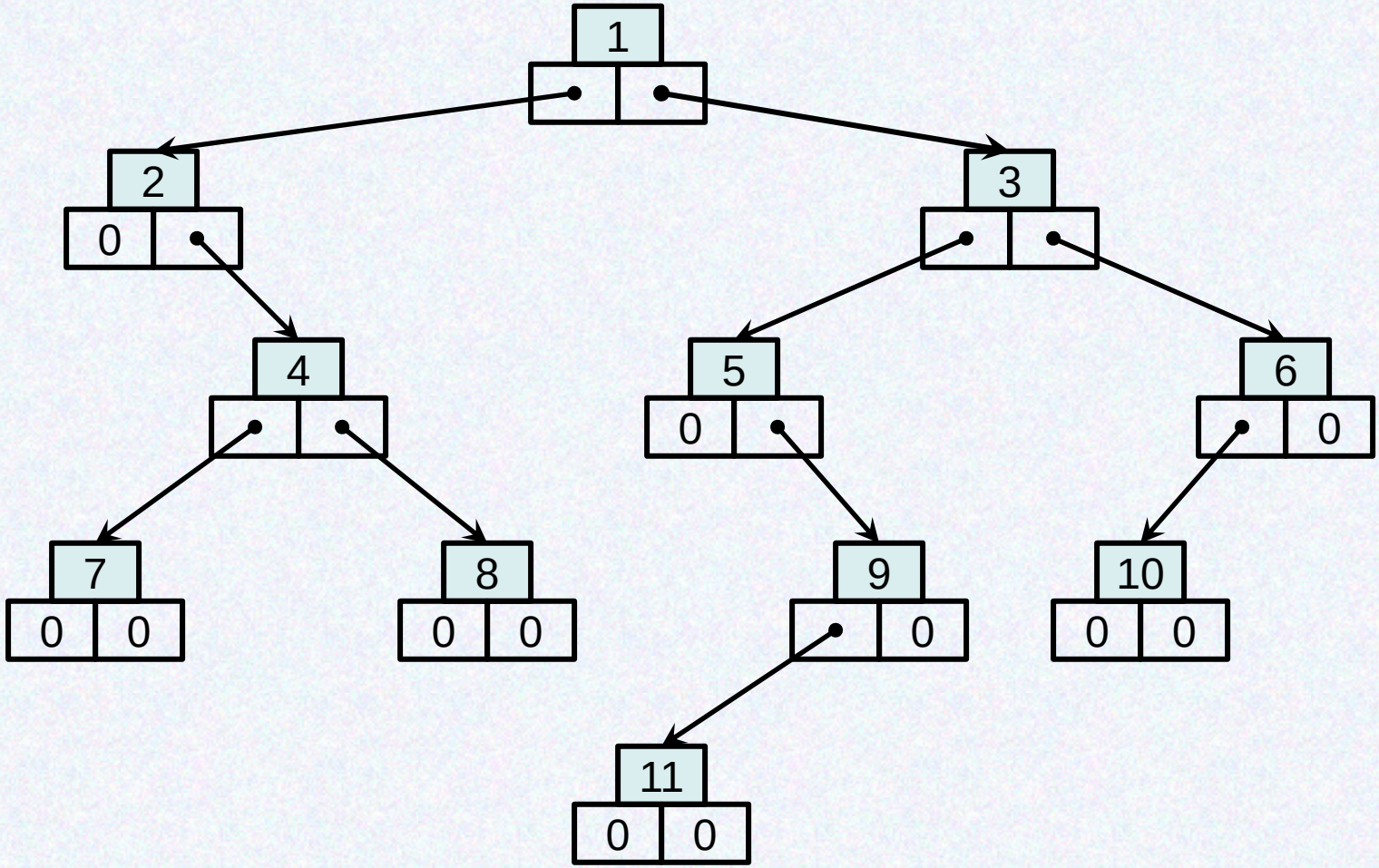
1 2 4 7 8

Обходы дерева



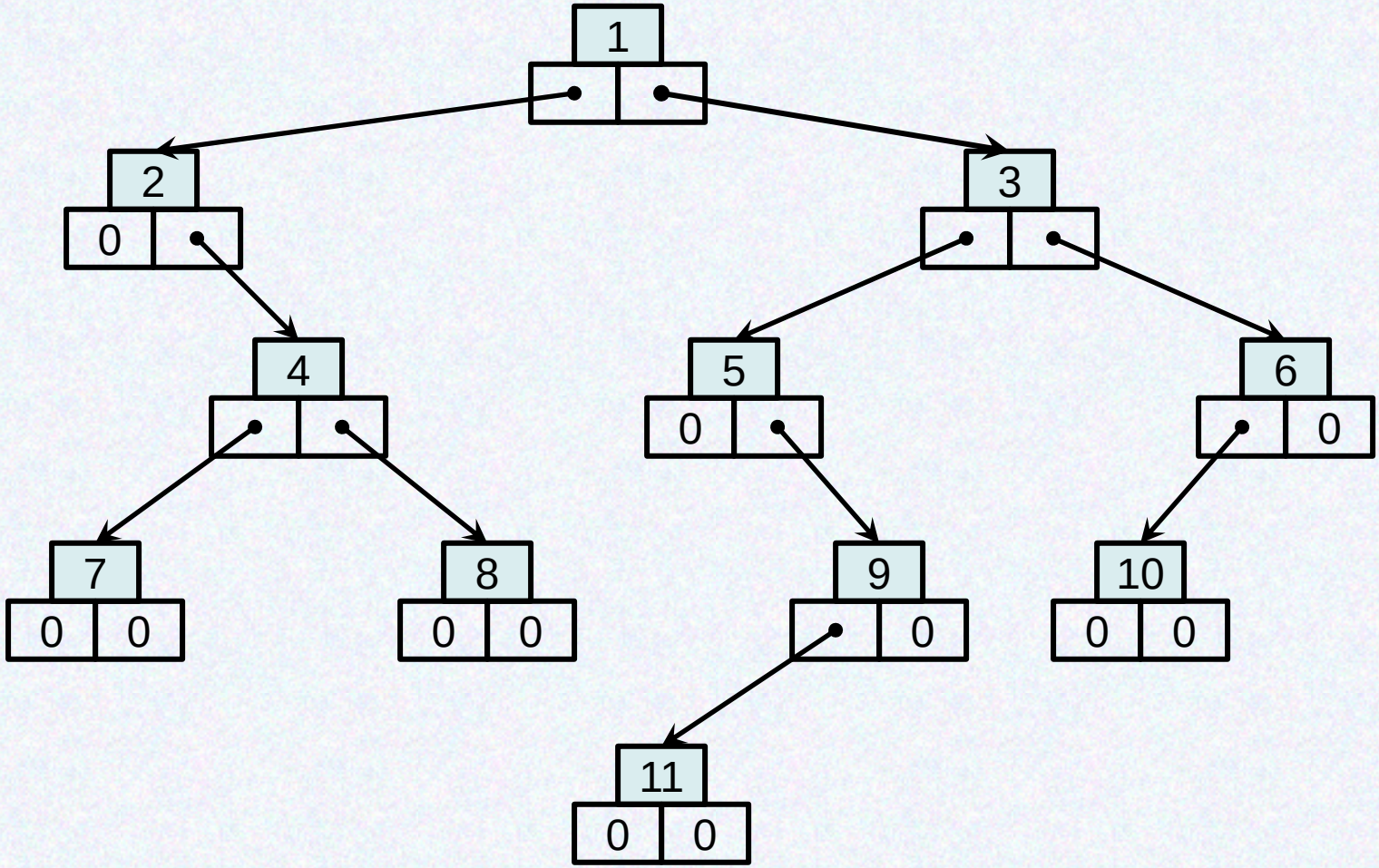
1 2 4 7 8 3

Обходы дерева



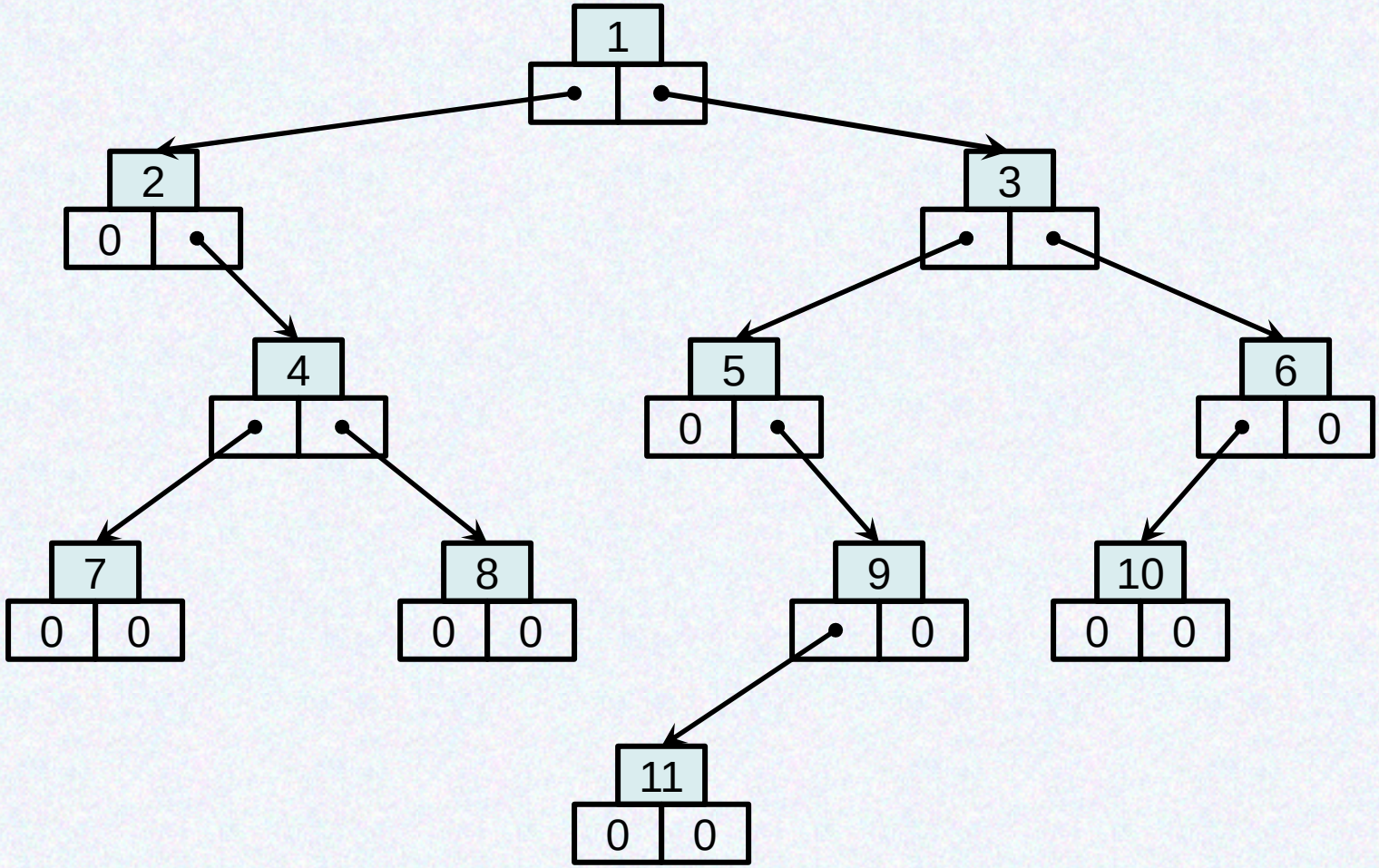
1 2 4 7 8 3 5

Обходы дерева



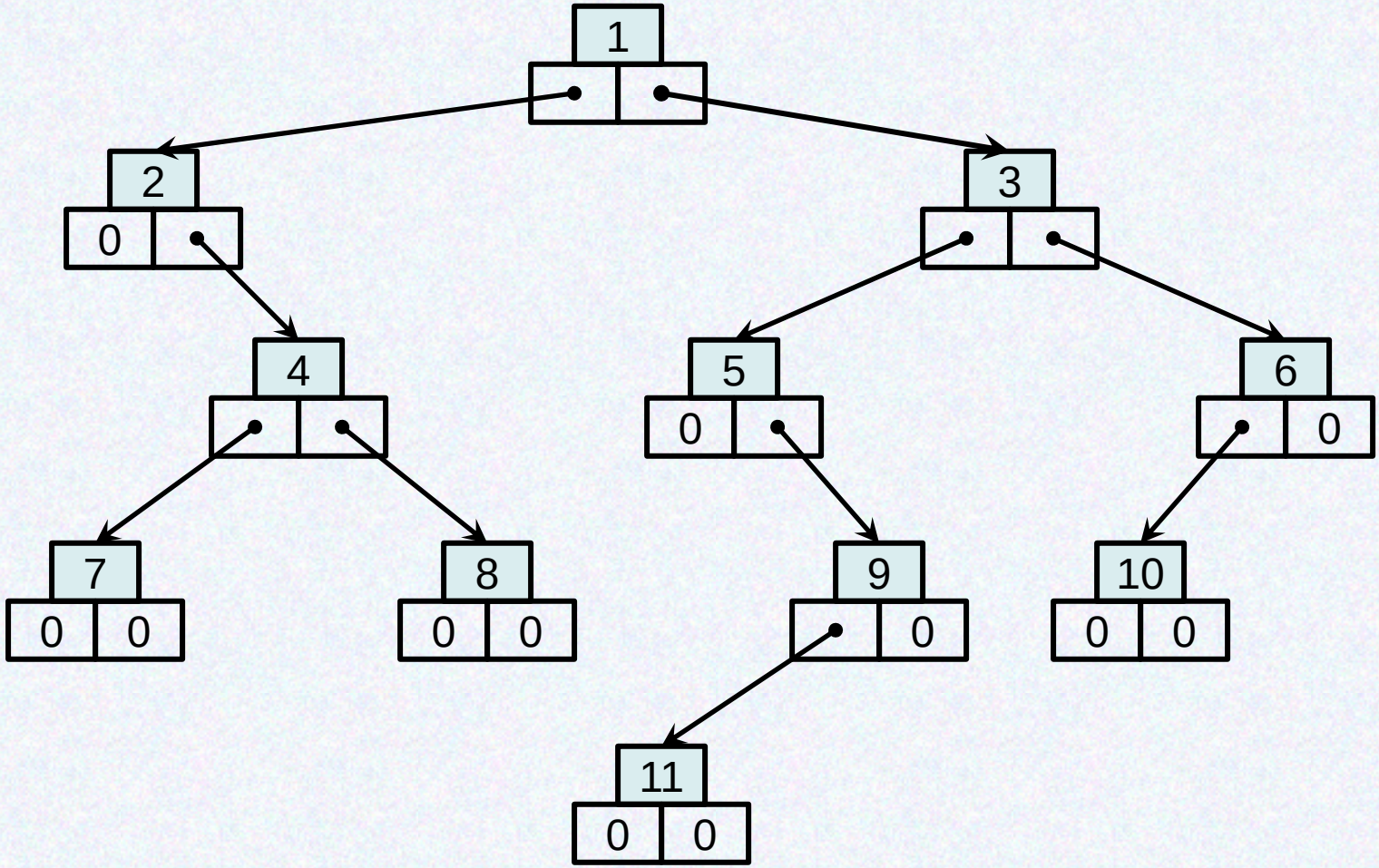
1 2 4 7 8 3 5 9

Обходы дерева



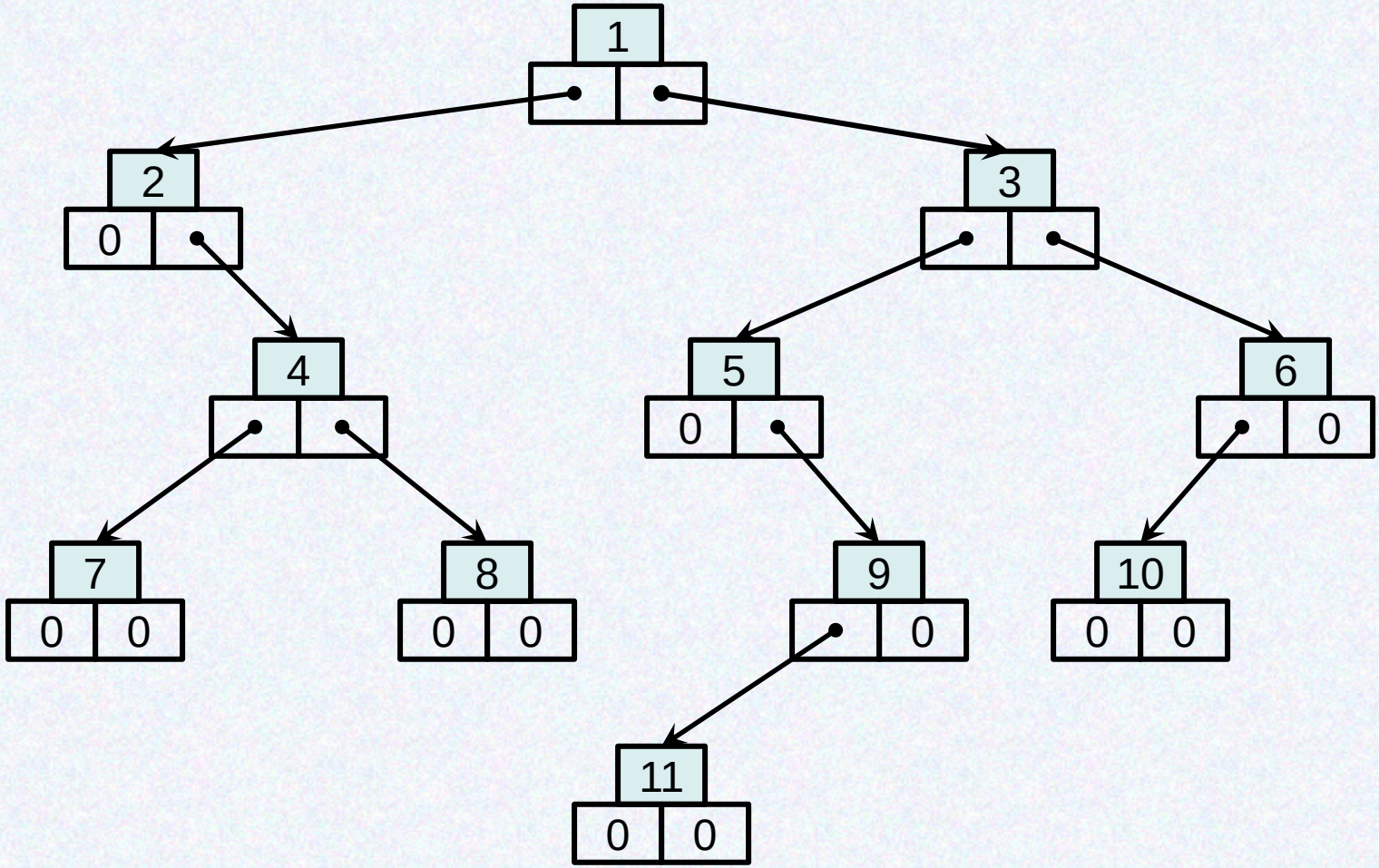
1 2 4 7 8 3 5 9 11

Обходы дерева



1 2 4 7 8 3 5 9 11 6

Обходы дерева



1 2 4 7 8 3 5 9 11 6 10

Обходы дерева

- *Центрированный обход (симметричный)*

Обходы дерева

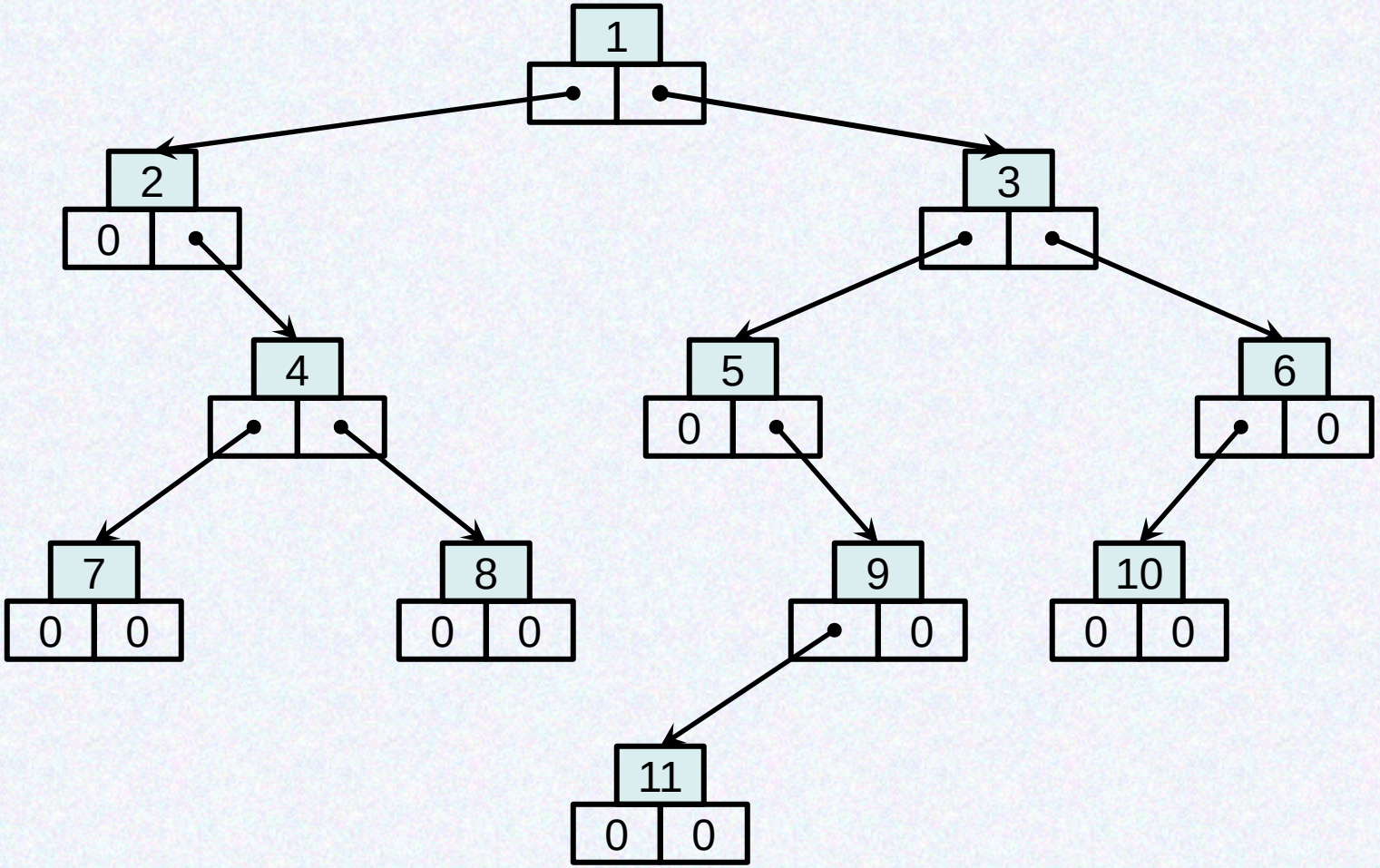
- *Центрированный обход (симметричный)*
  – левое поддерево, корень, правое поддерево

Обходы дерева

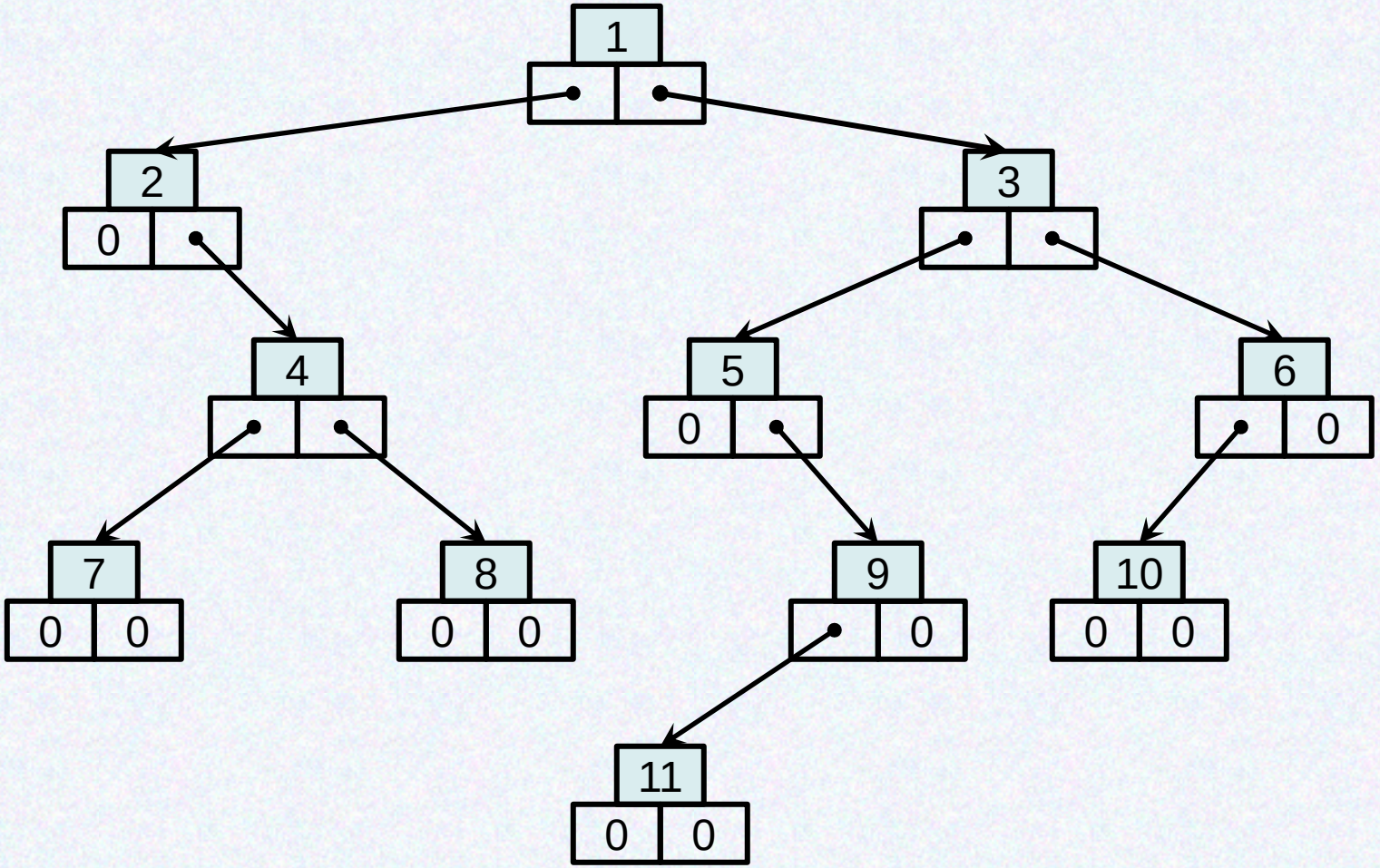
- *Центрированный обход (симметричный)*
  – левое поддерево, корень, правое поддерево

Каждое поддерево обрабатывается в таком же порядке

Обходы дерева

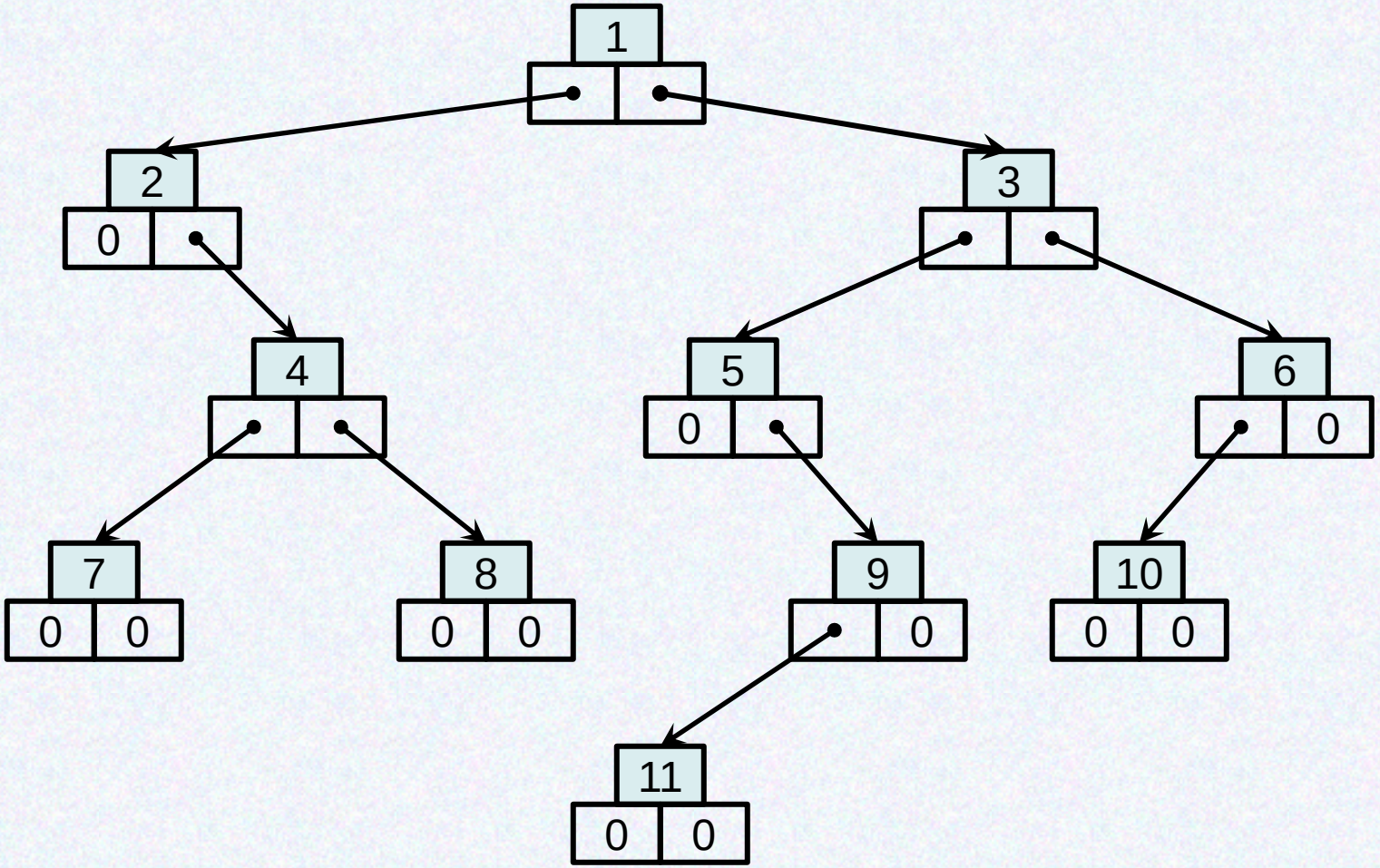


Обходы дерева



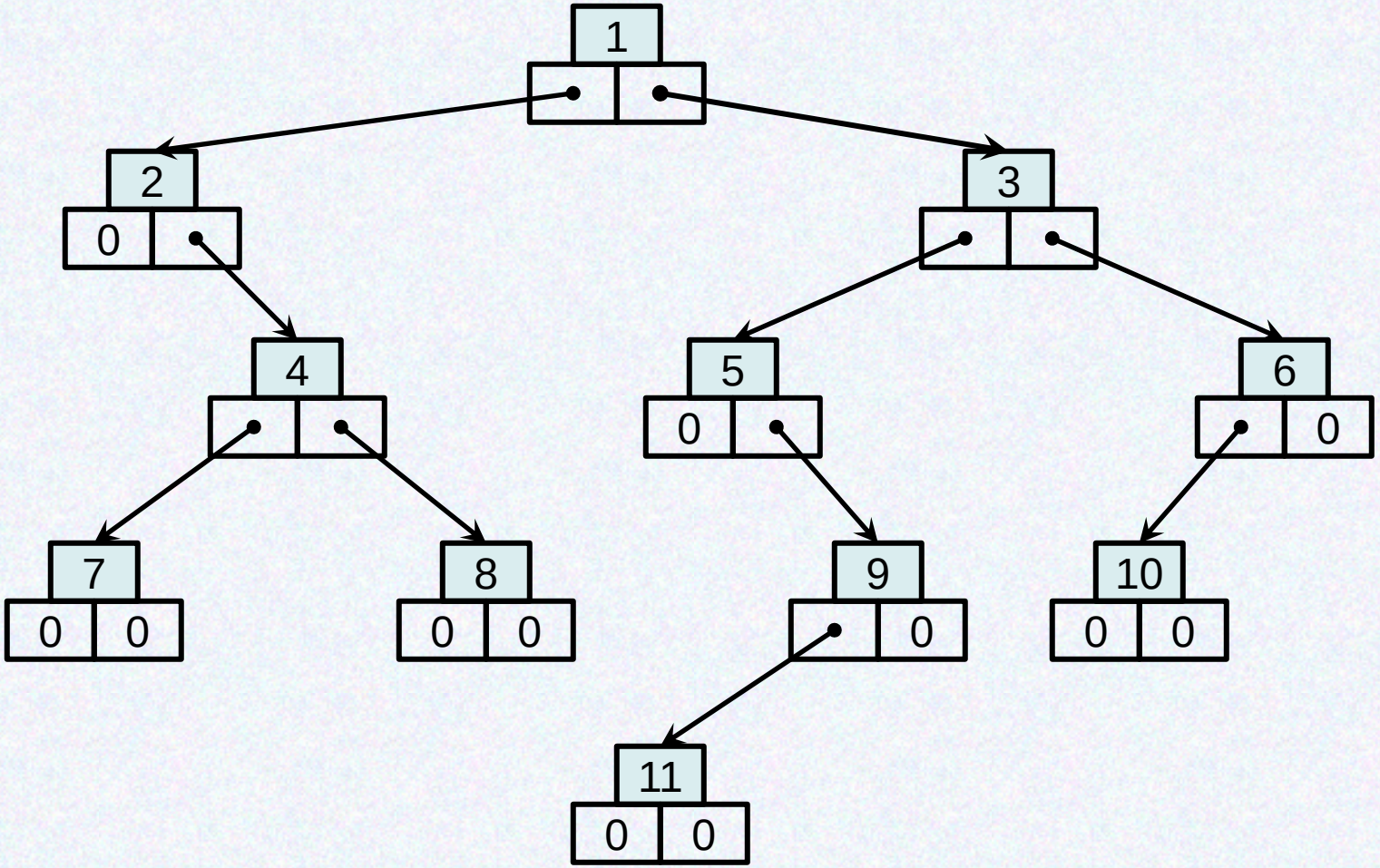
2

Обходы дерева



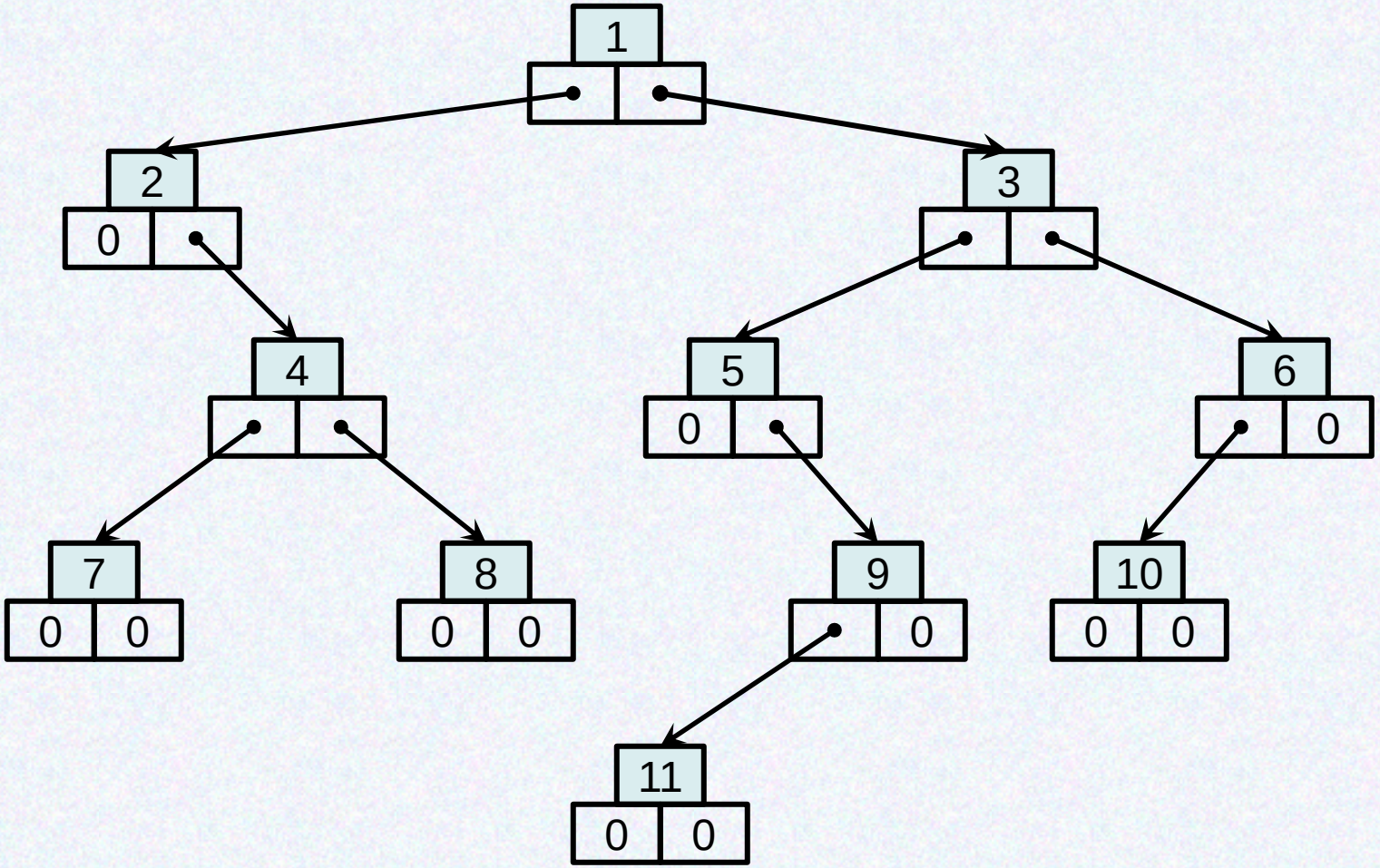
2 7

Обходы дерева



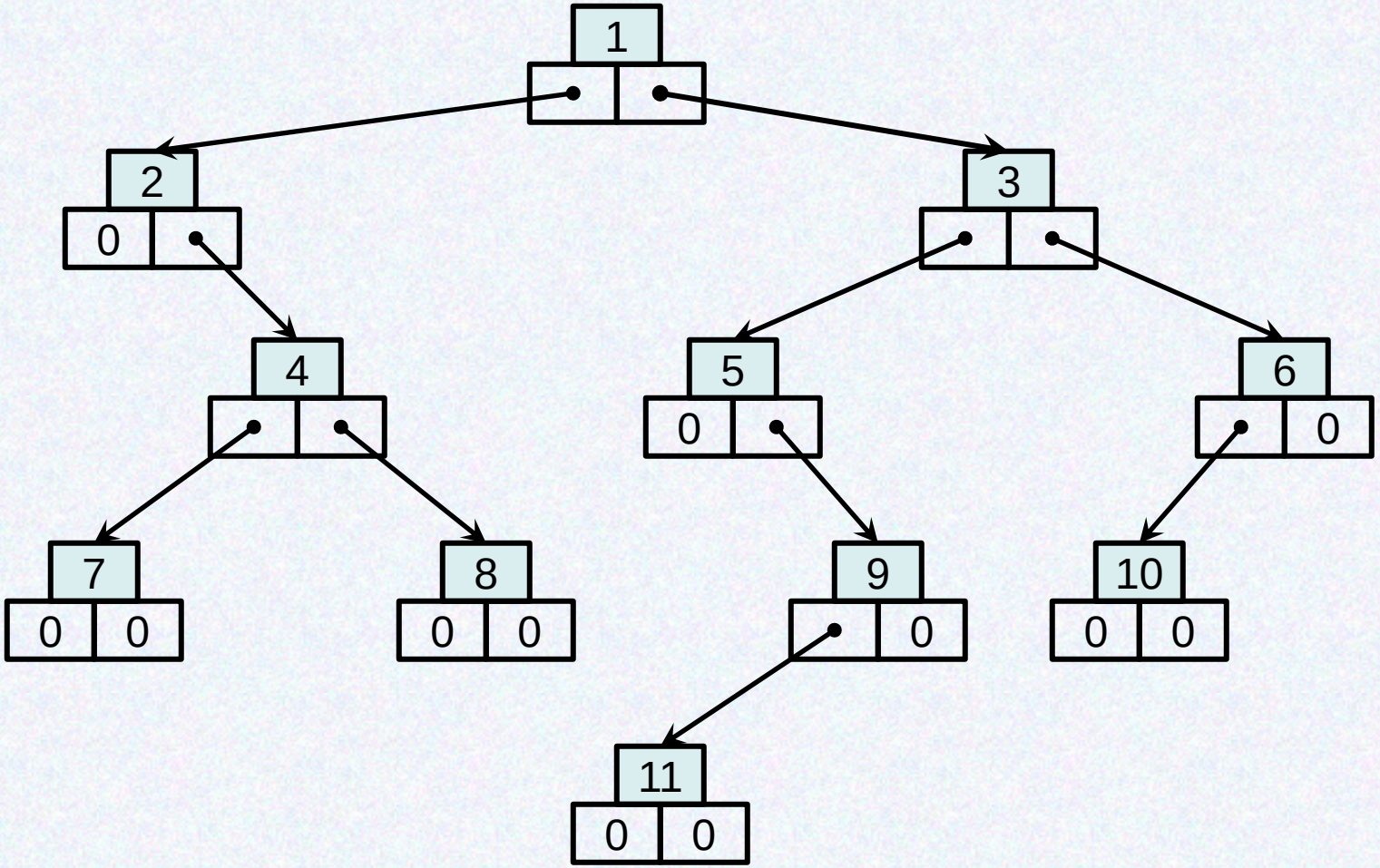
2 7 4

Обходы дерева



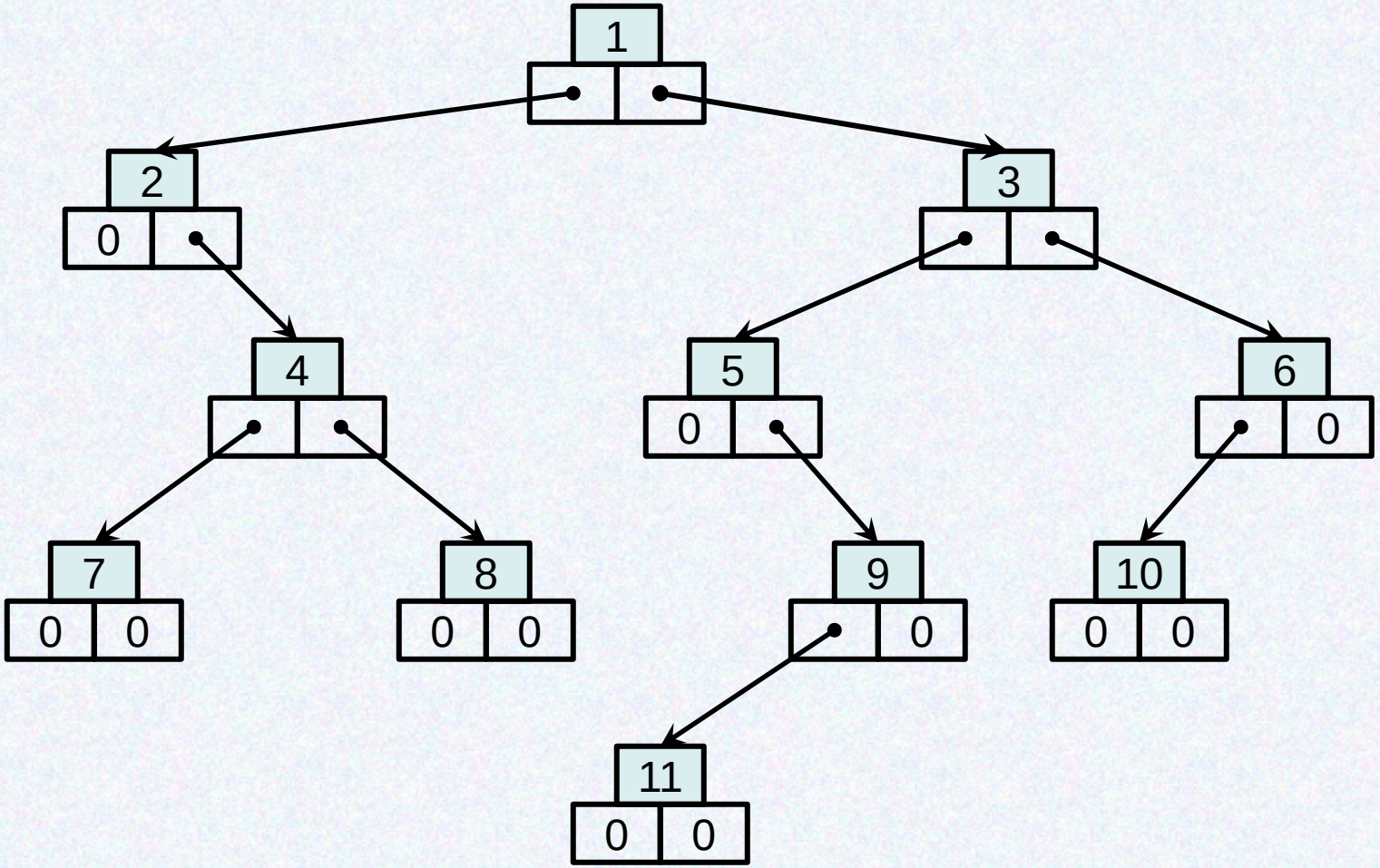
2 7 4 8

Обходы дерева



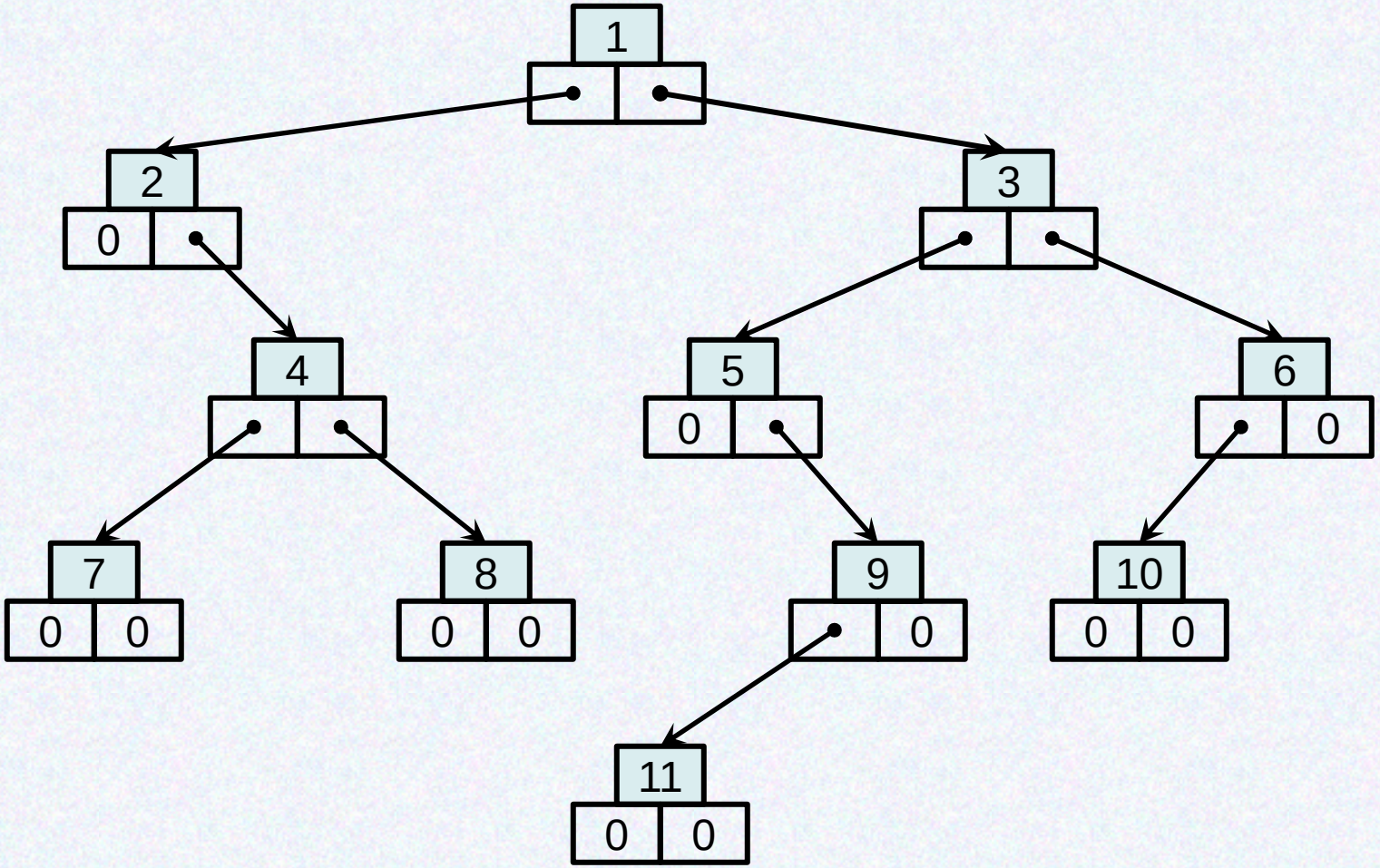
2 7 4 8 1

Обходы дерева



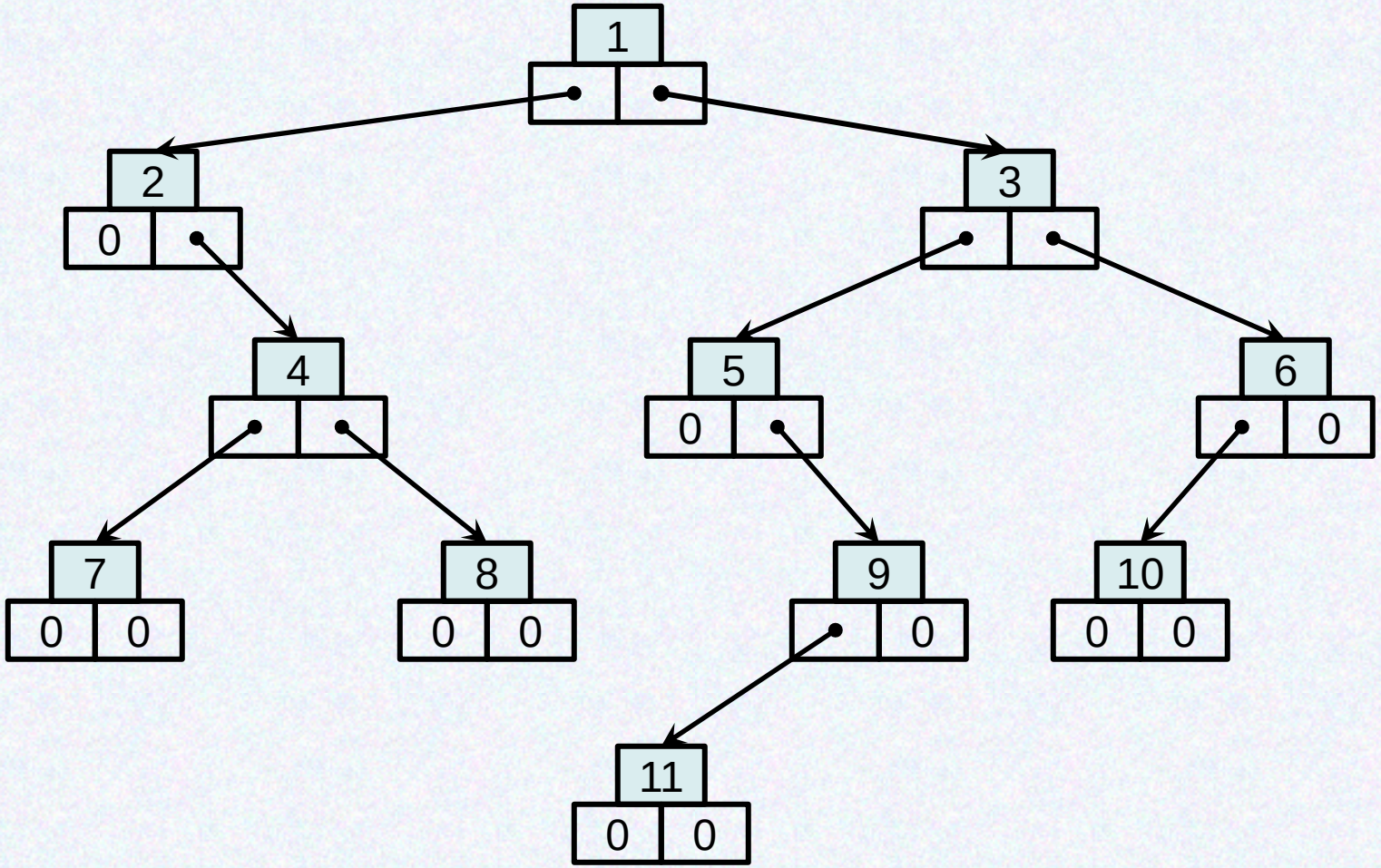
2 7 4 8 1 5

Обходы дерева



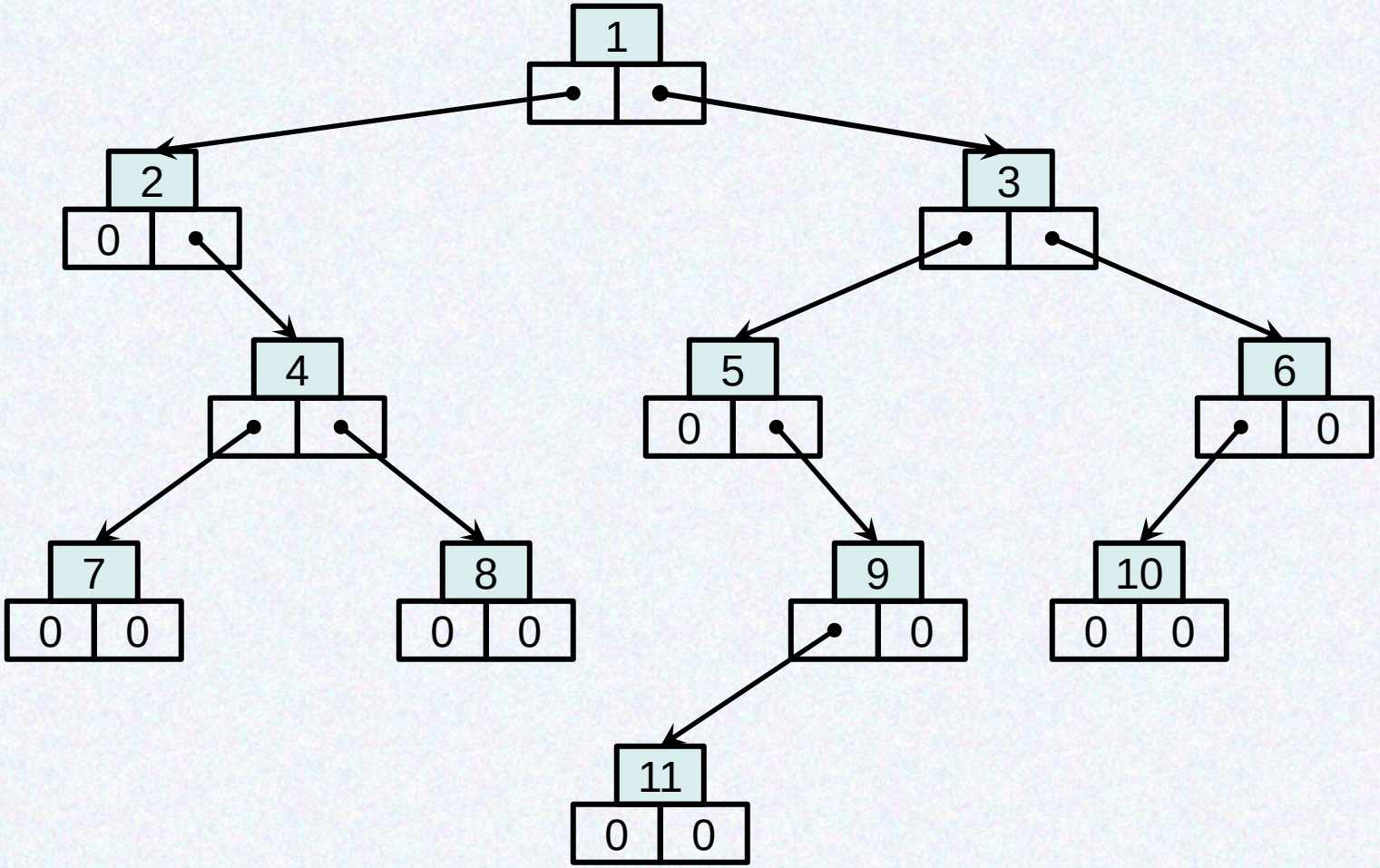
2 7 4 8 1 5 11

Обходы дерева



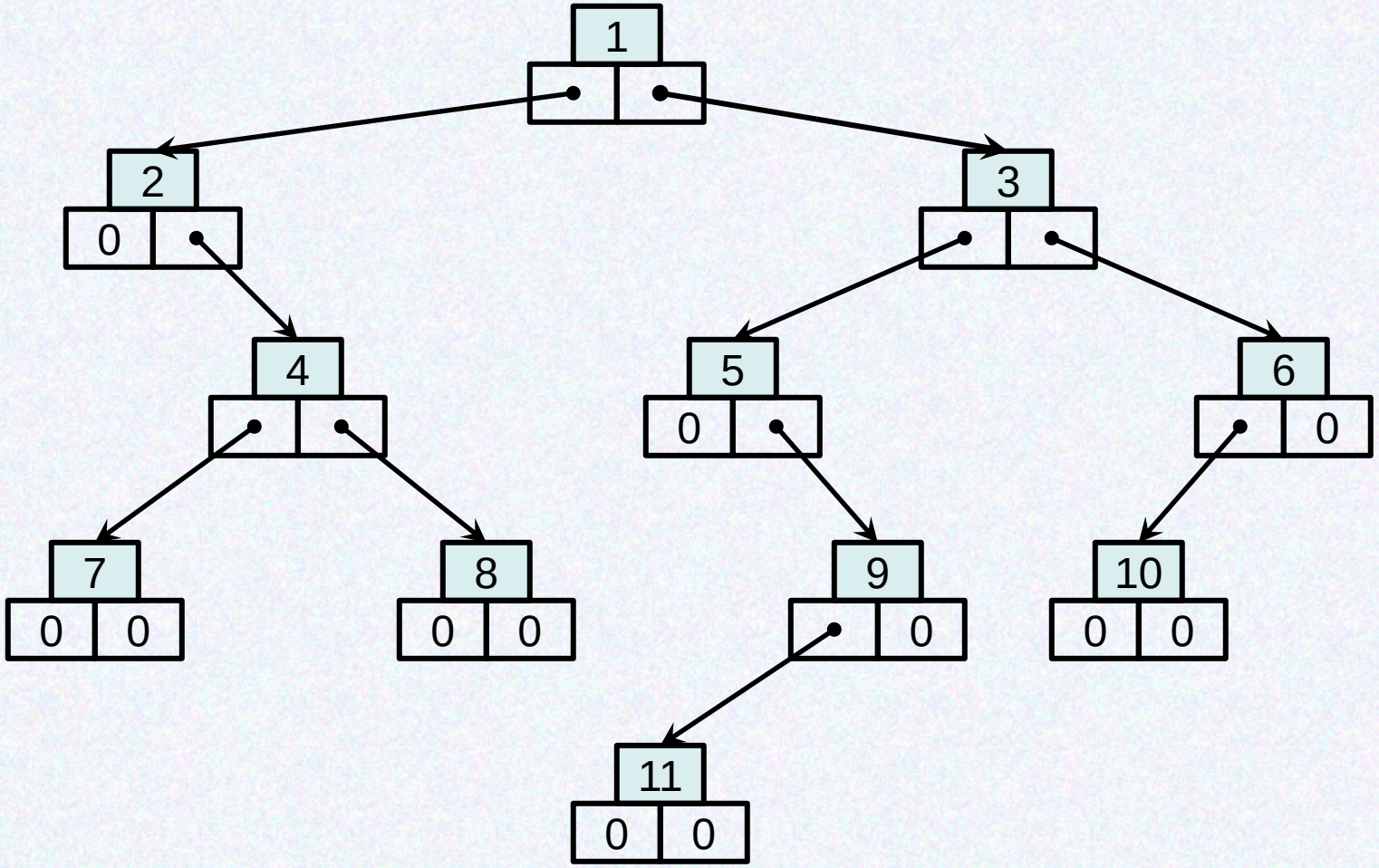
2 7 4 8 1 5 11 9

Обходы дерева



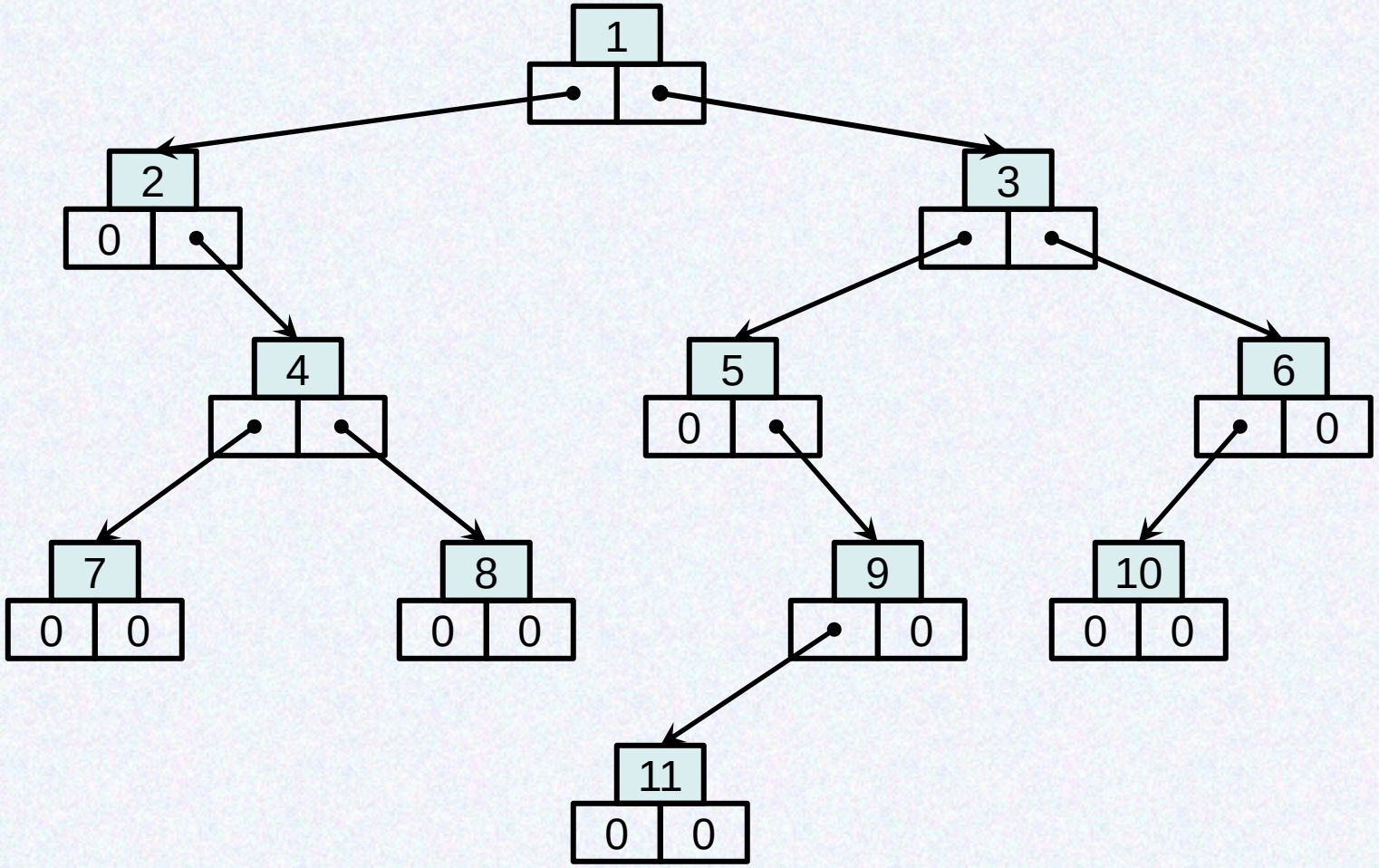
2 7 4 8 1 5 11 9 3

Обходы дерева



2 7 4 8 1 5 11 9 3 10

Обходы дерева



2 7 4 8 1 5 11 9 3 10 6

Обходы дерева

- *Концевой обход (обратный, в глубину)*

Обходы дерева

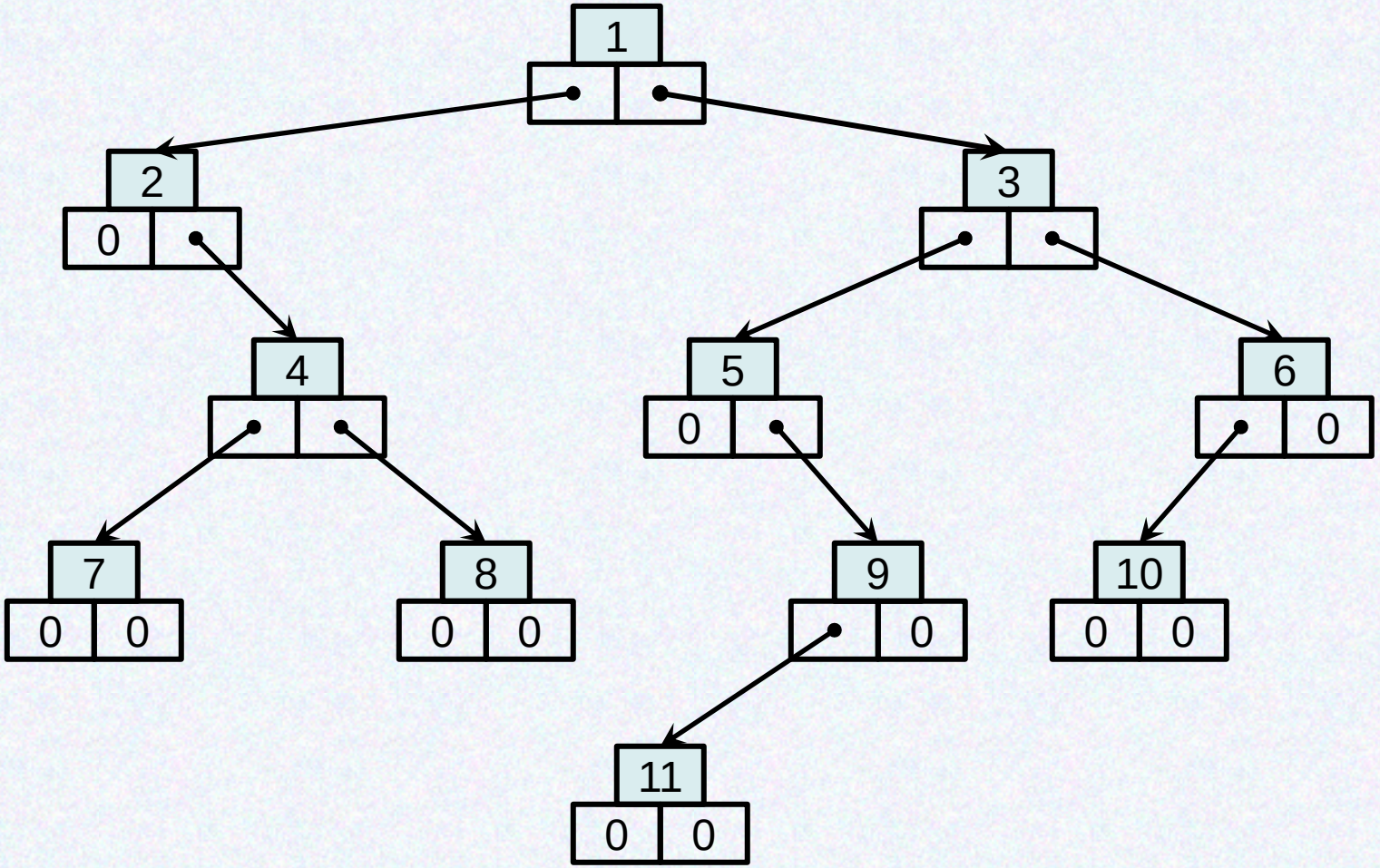
- *Концевой обход (обратный, в глубину)*
  – левое поддерево, правое поддерево, корень

Обходы дерева

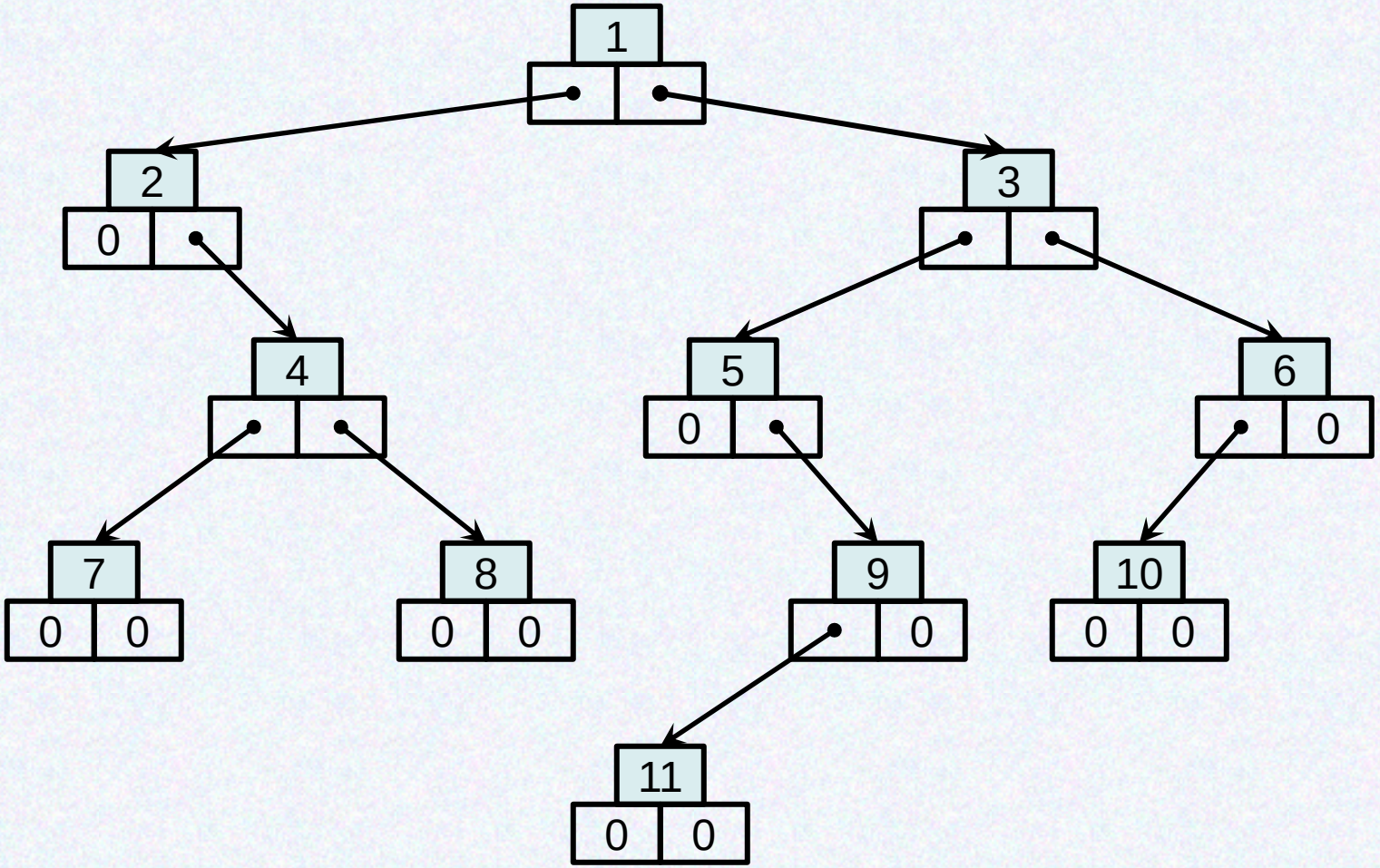
- *Концевой обход (обратный, в глубину)*
  – левое поддерево, правое поддерево, корень

Каждое поддерево обрабатывается в таком же порядке

Обходы дерева

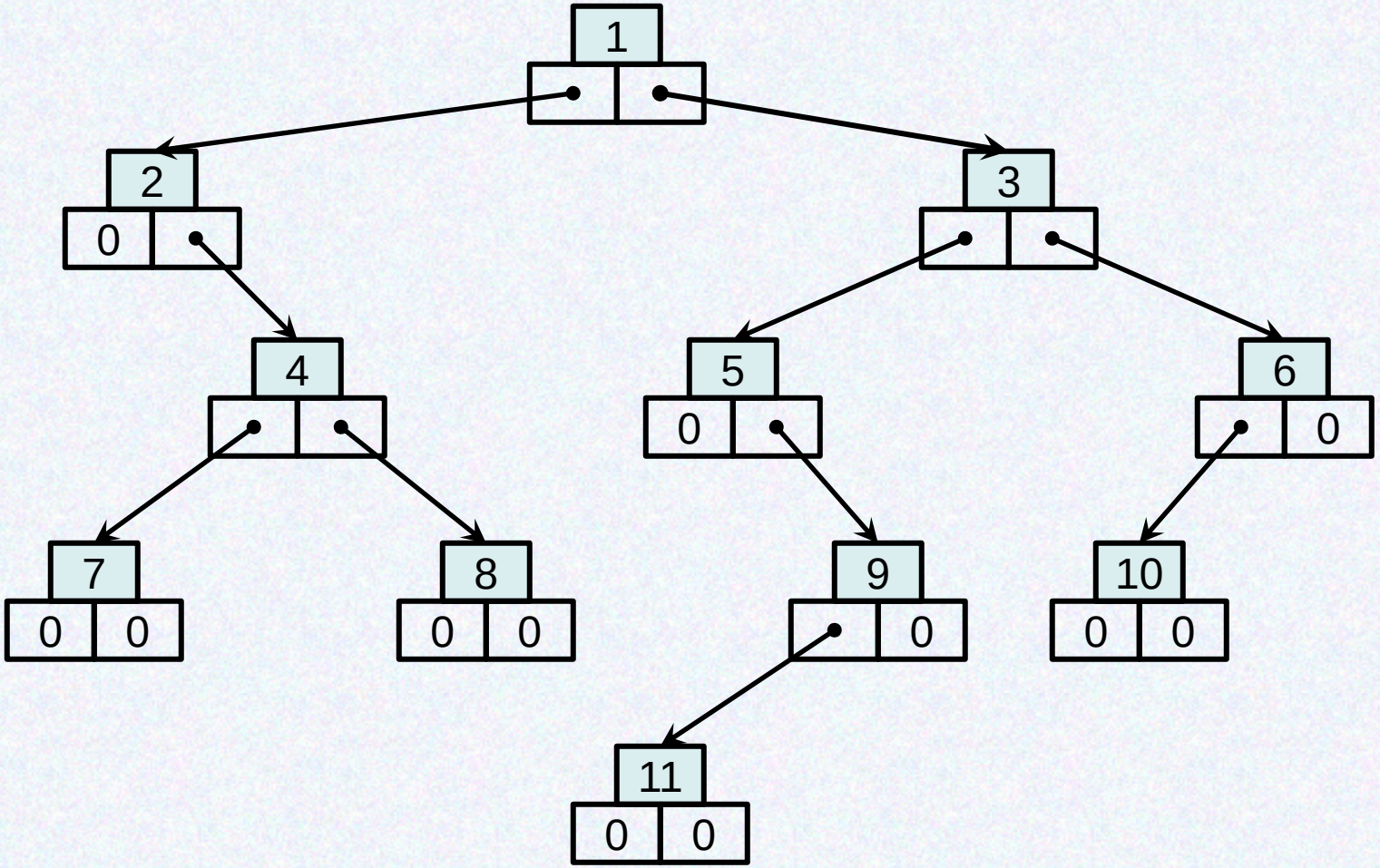


Обходы дерева



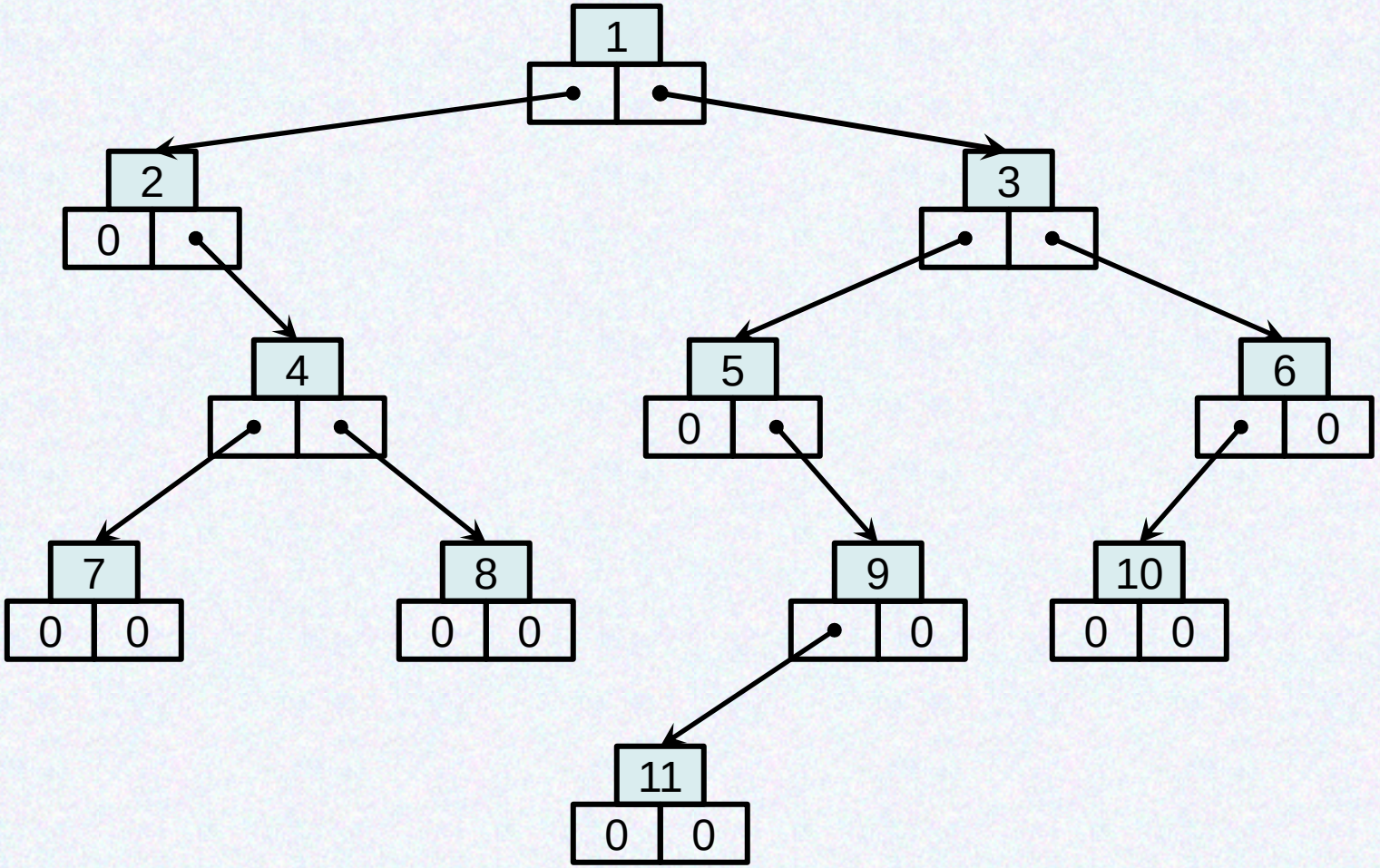
7

Обходы дерева



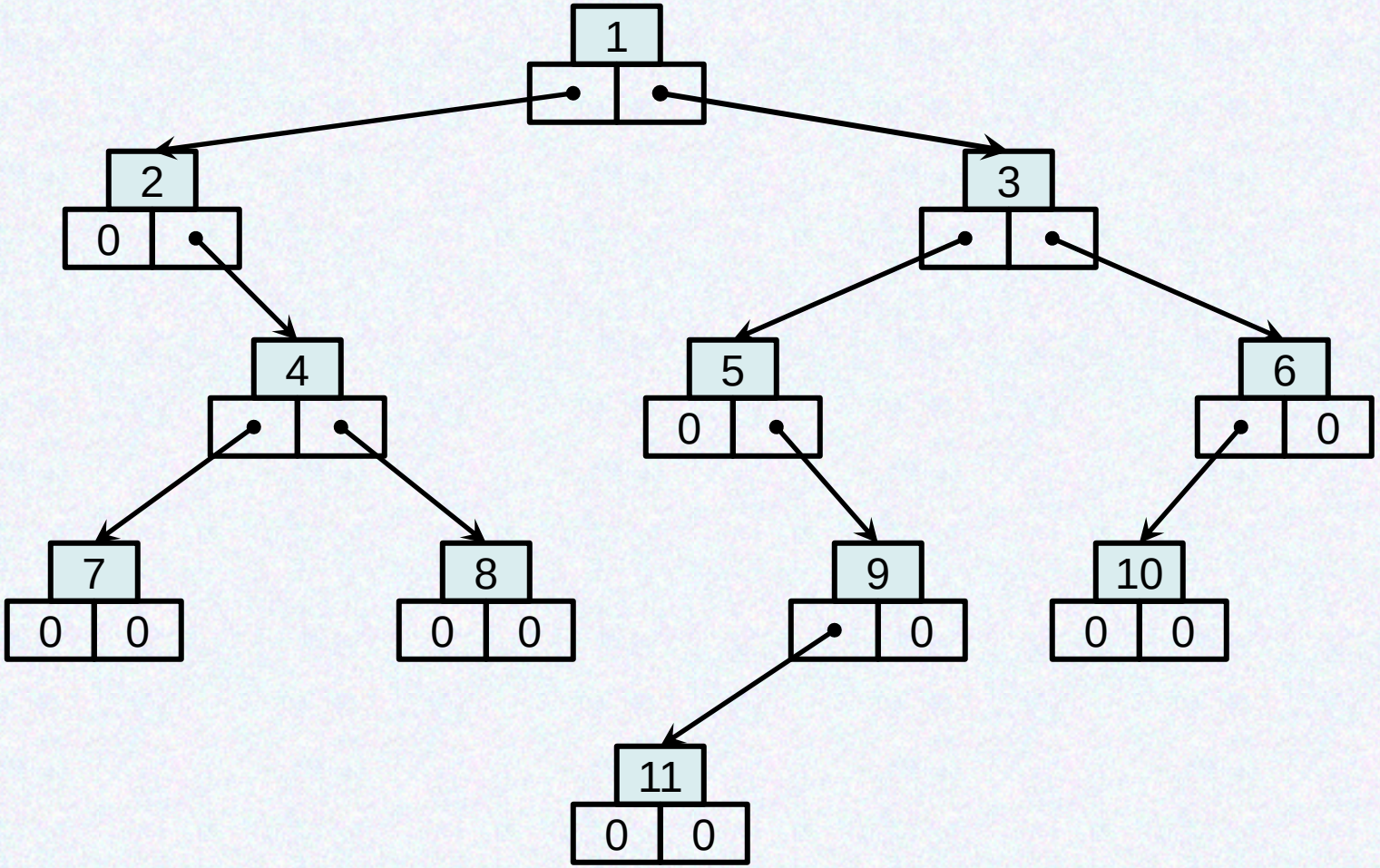
7 8

Обходы дерева



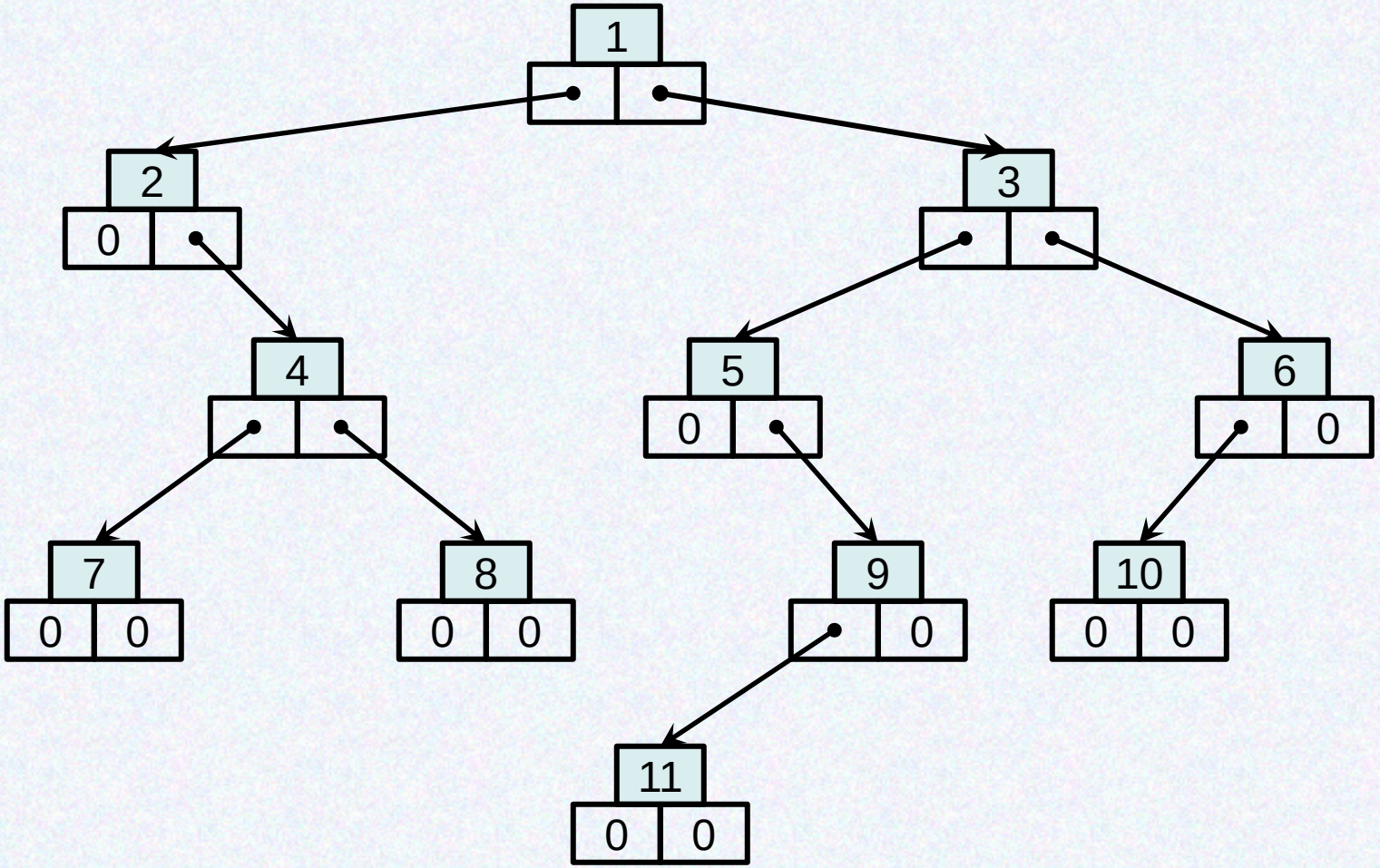
7 8 4

Обходы дерева



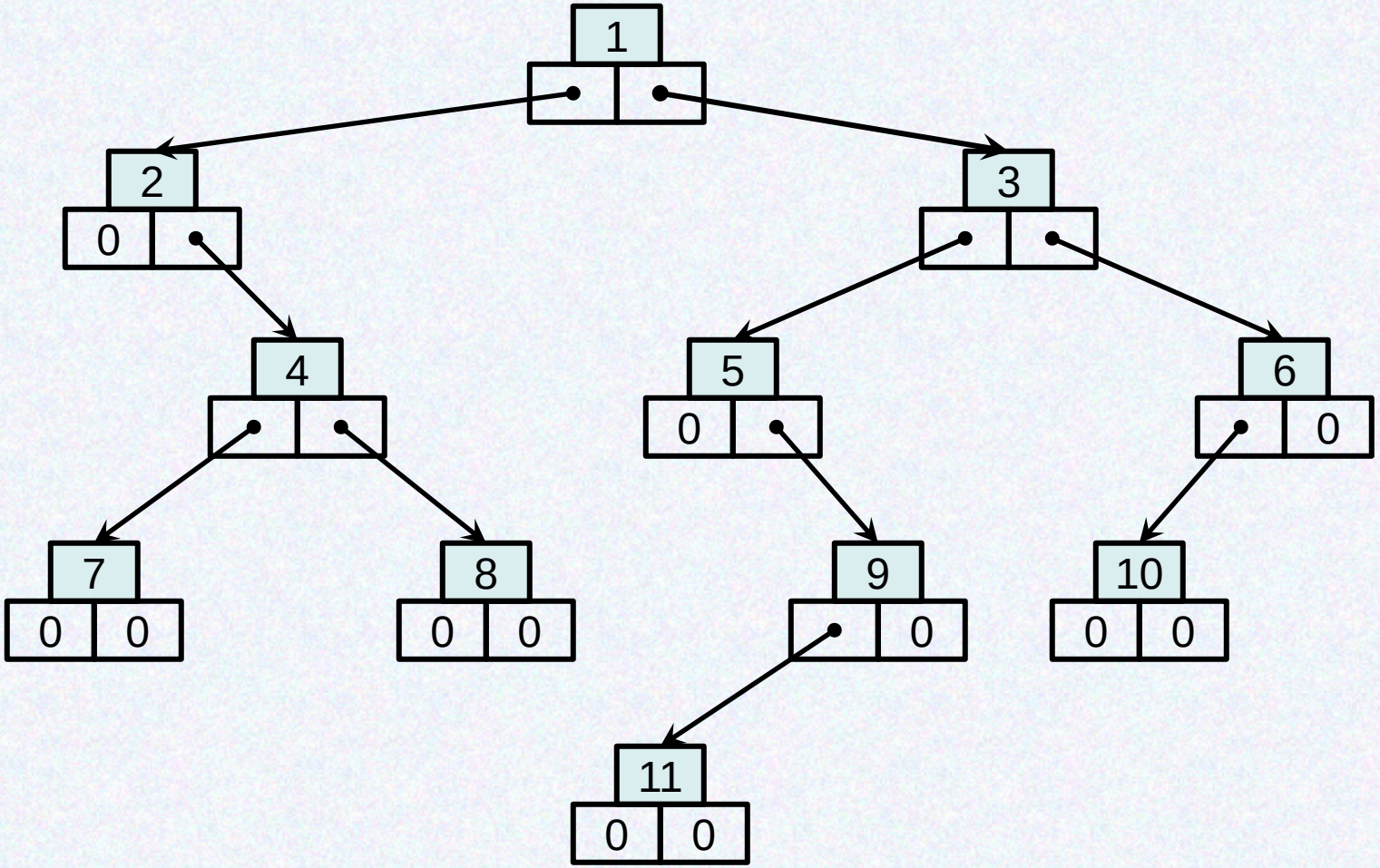
7 8 4 2

Обходы дерева



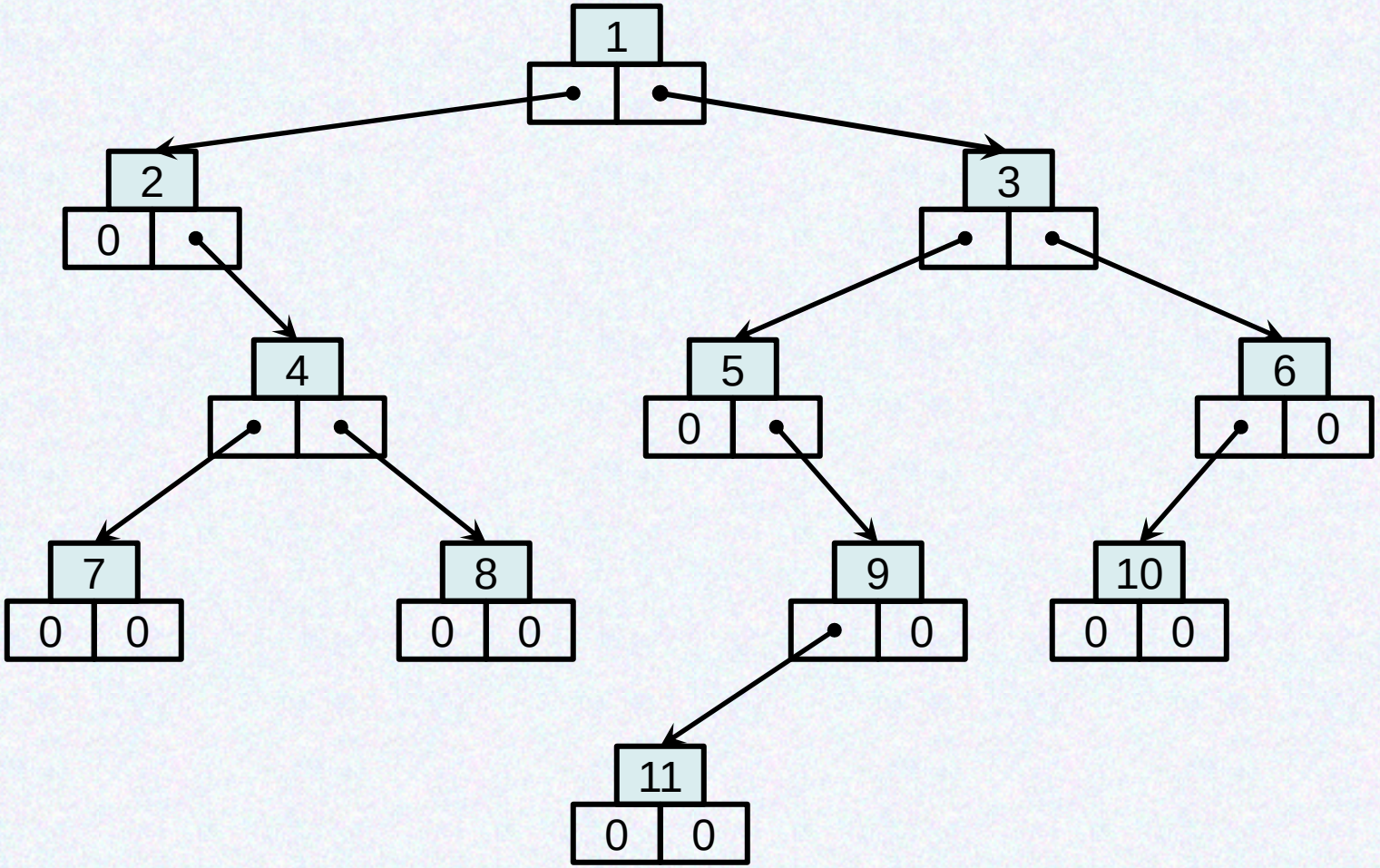
7 8 4 2 11

Обходы дерева



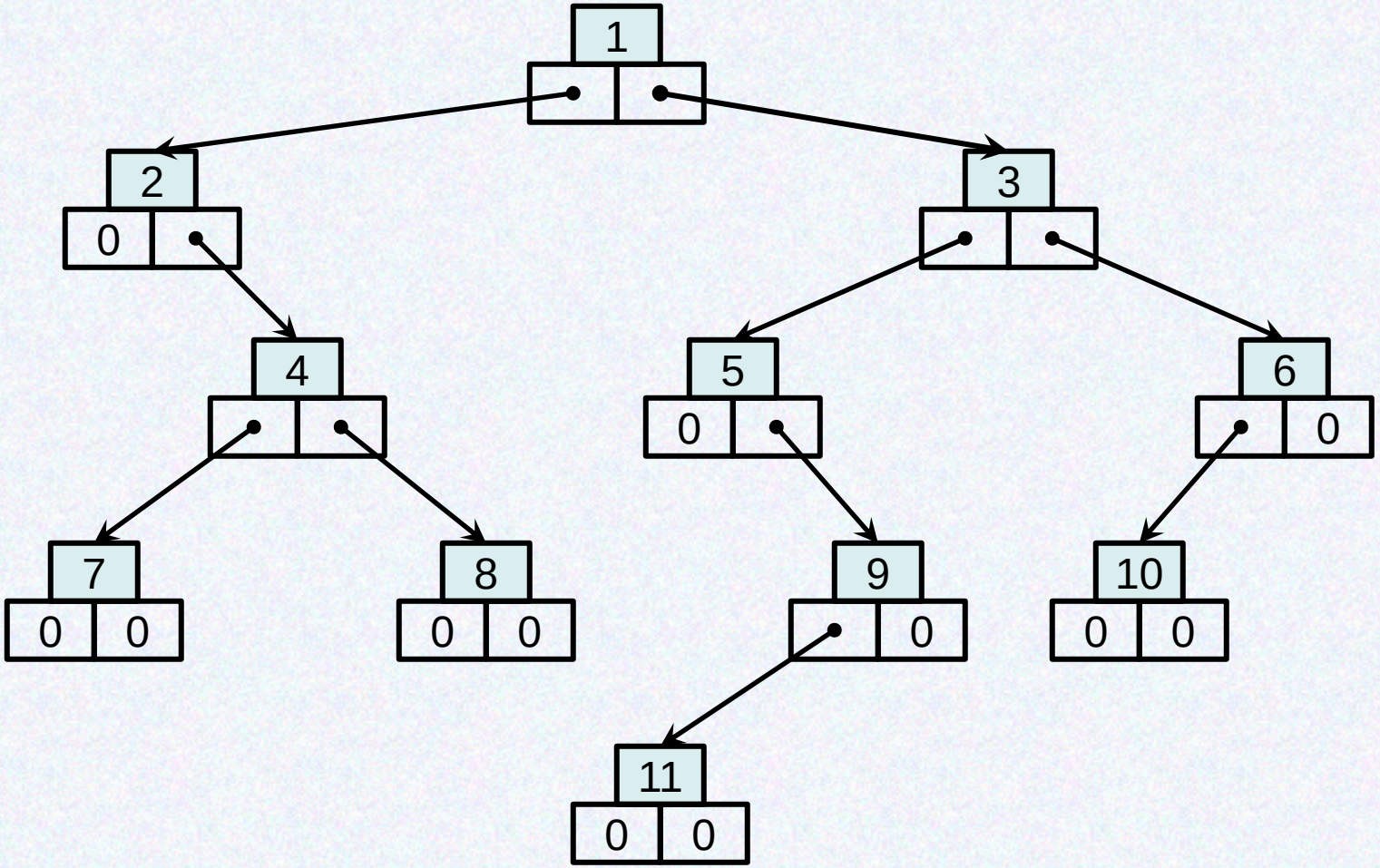
7 8 4 2 11 9

Обходы дерева



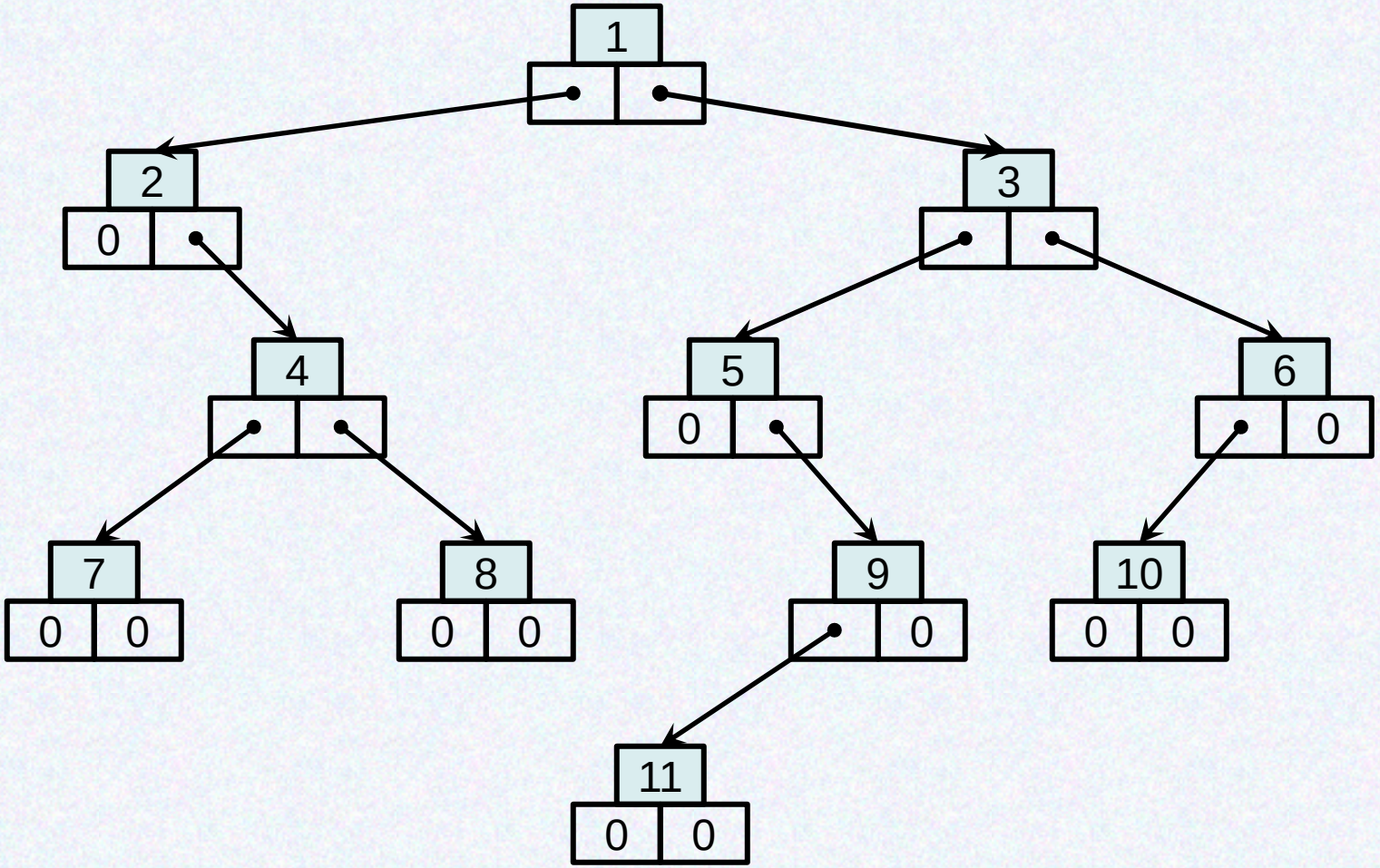
7 8 4 2 11 9 5

Обходы дерева



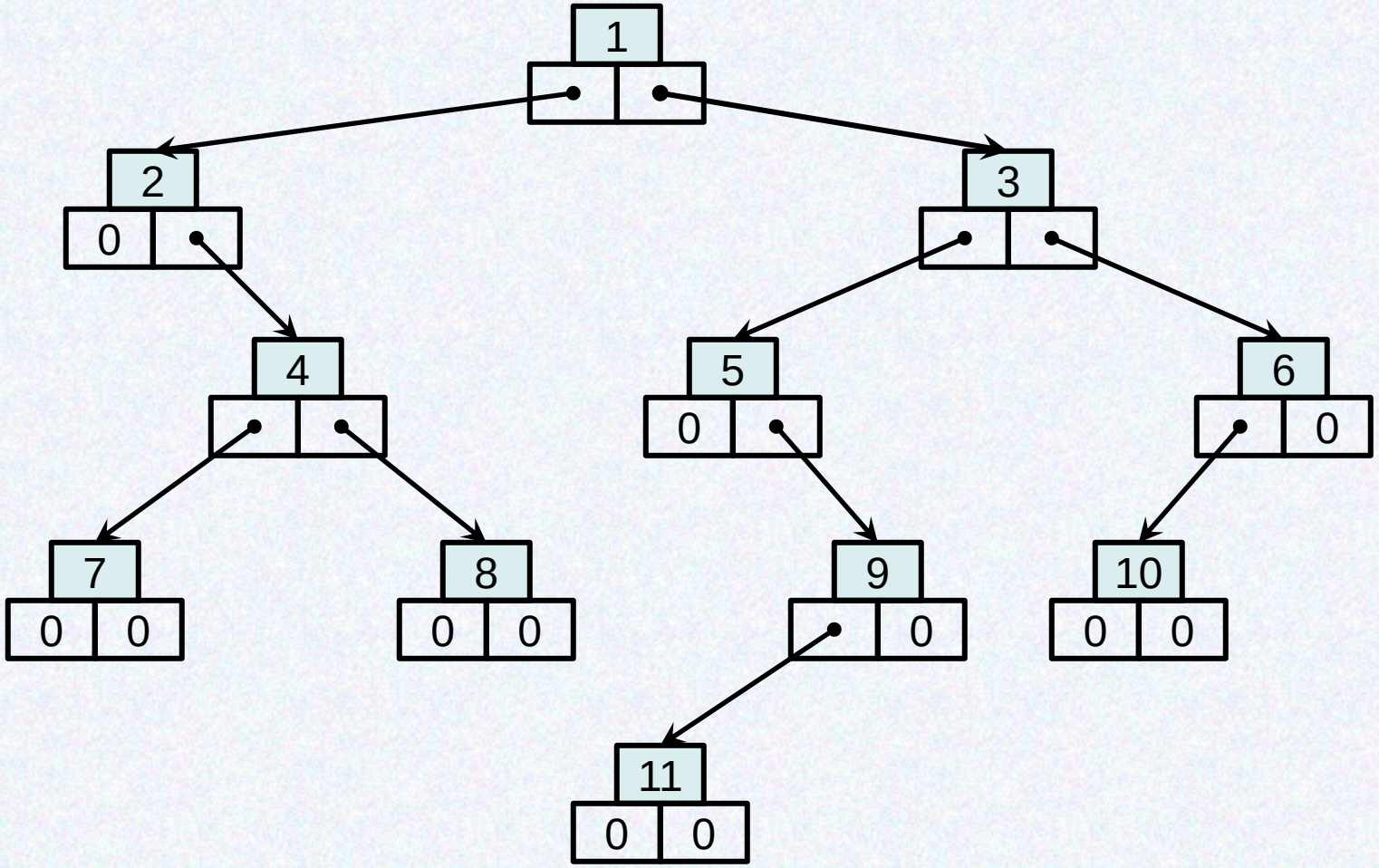
7 8 4 2 11 9 5 10

Обходы дерева



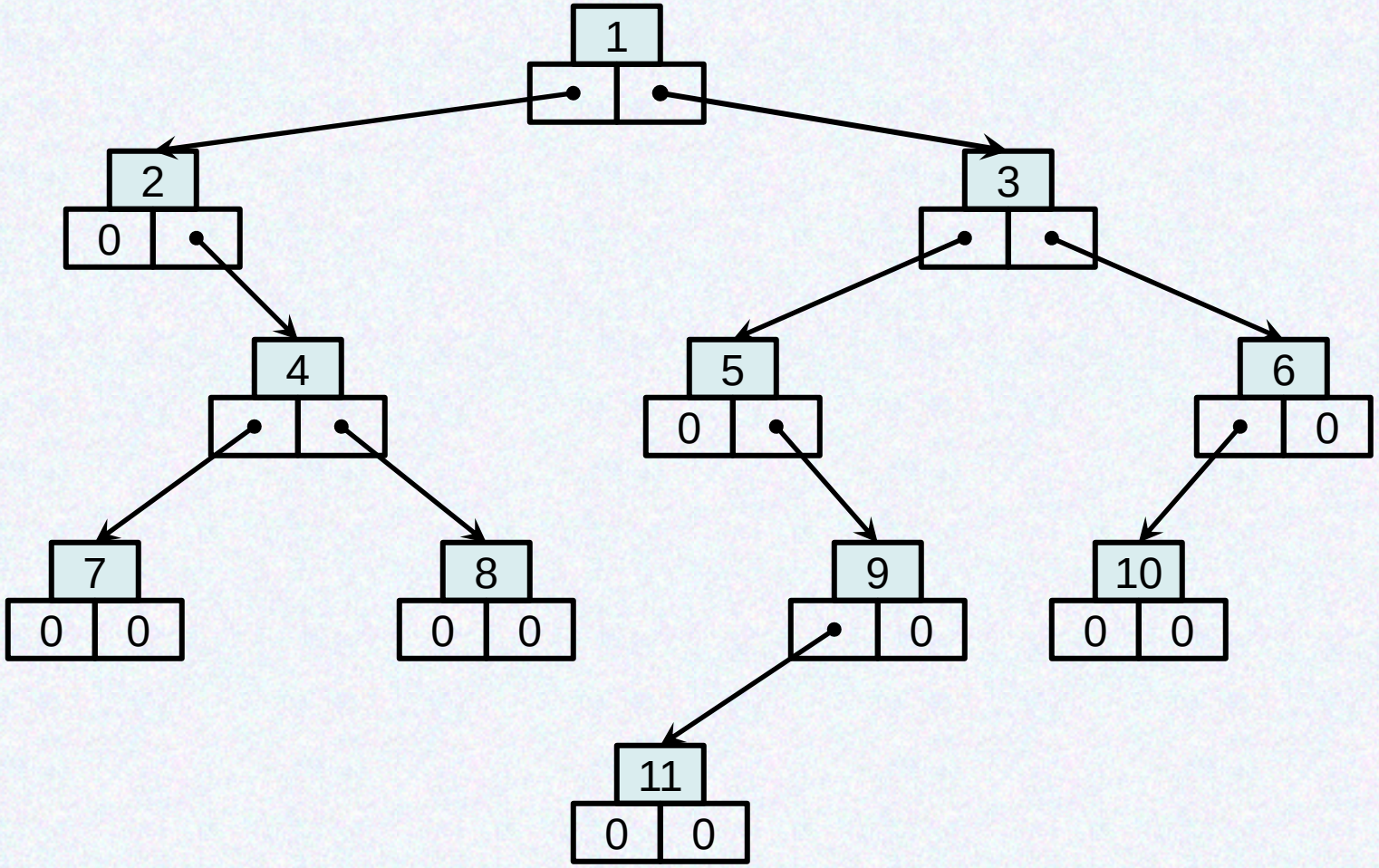
7 8 4 2 11 9 5 10 6

Обходы дерева



7 8 4 2 11 9 5 10 6 3

Обходы дерева



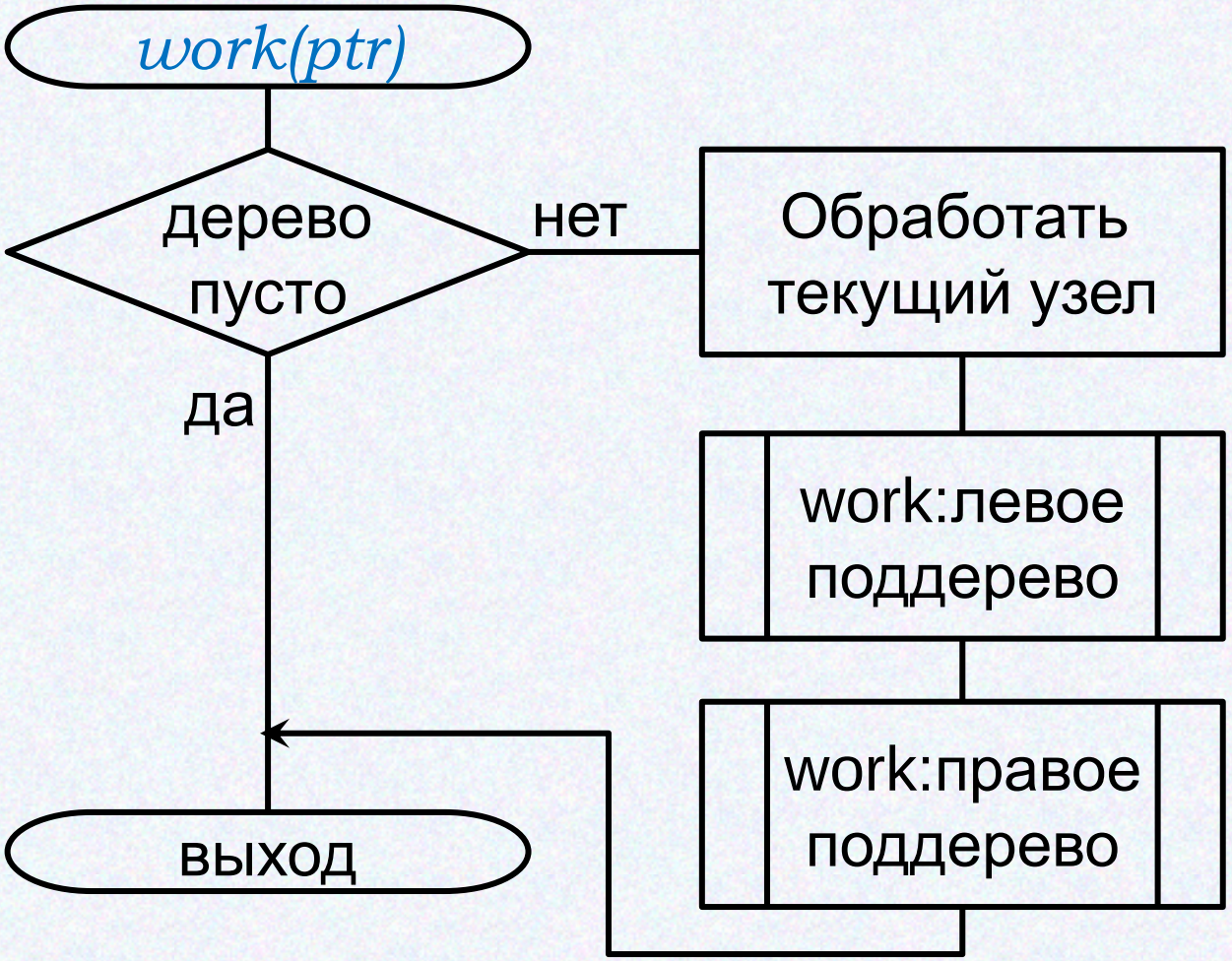
7 8 4 2 11 9 5 10 6 3 1

Алгоритм прямого обхода

Рекурсивный алгоритм; ptr – указатель на корень дерева

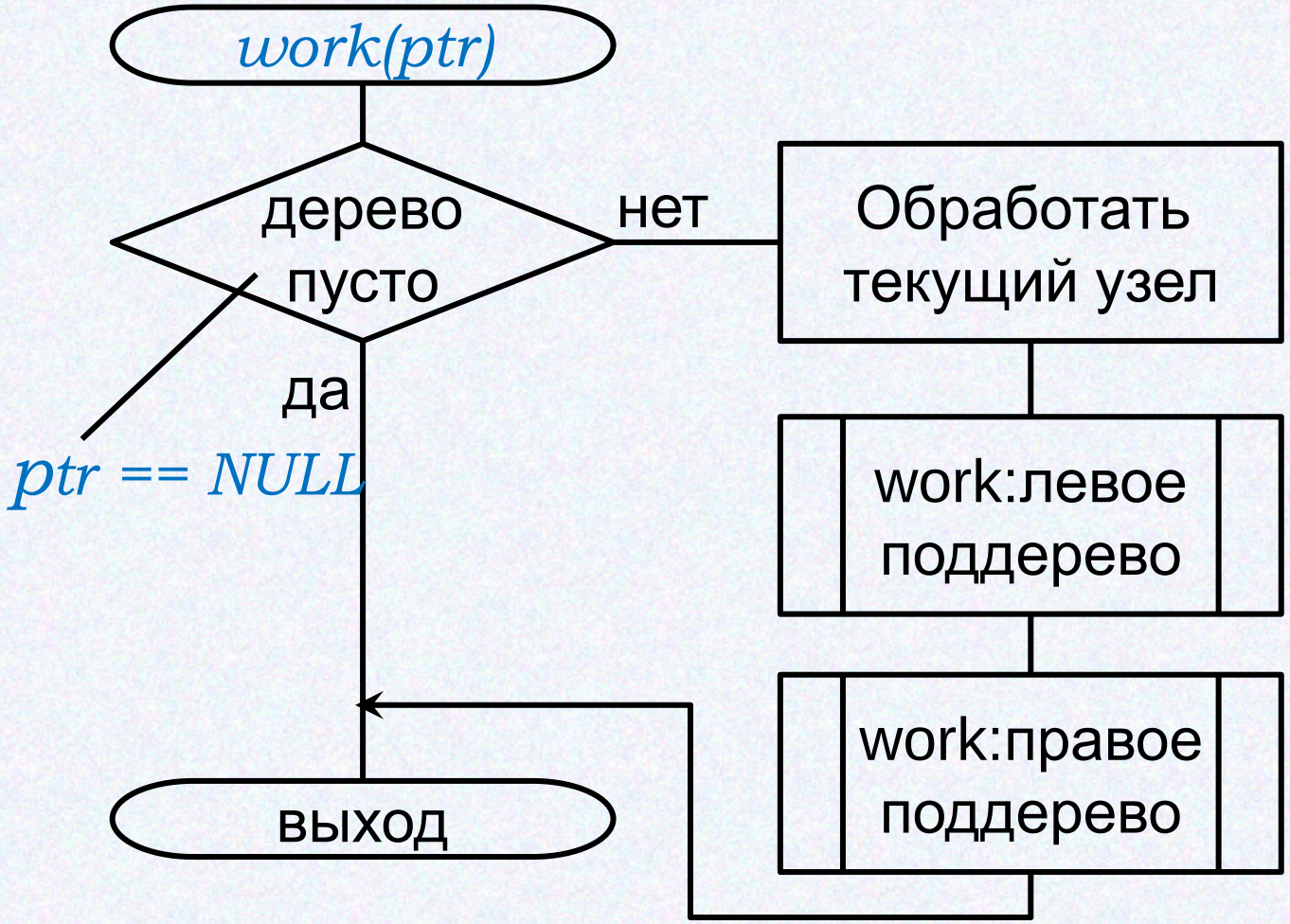
Алгоритм прямого обхода

Рекурсивный алгоритм; *ptr* – указатель на корень дерева



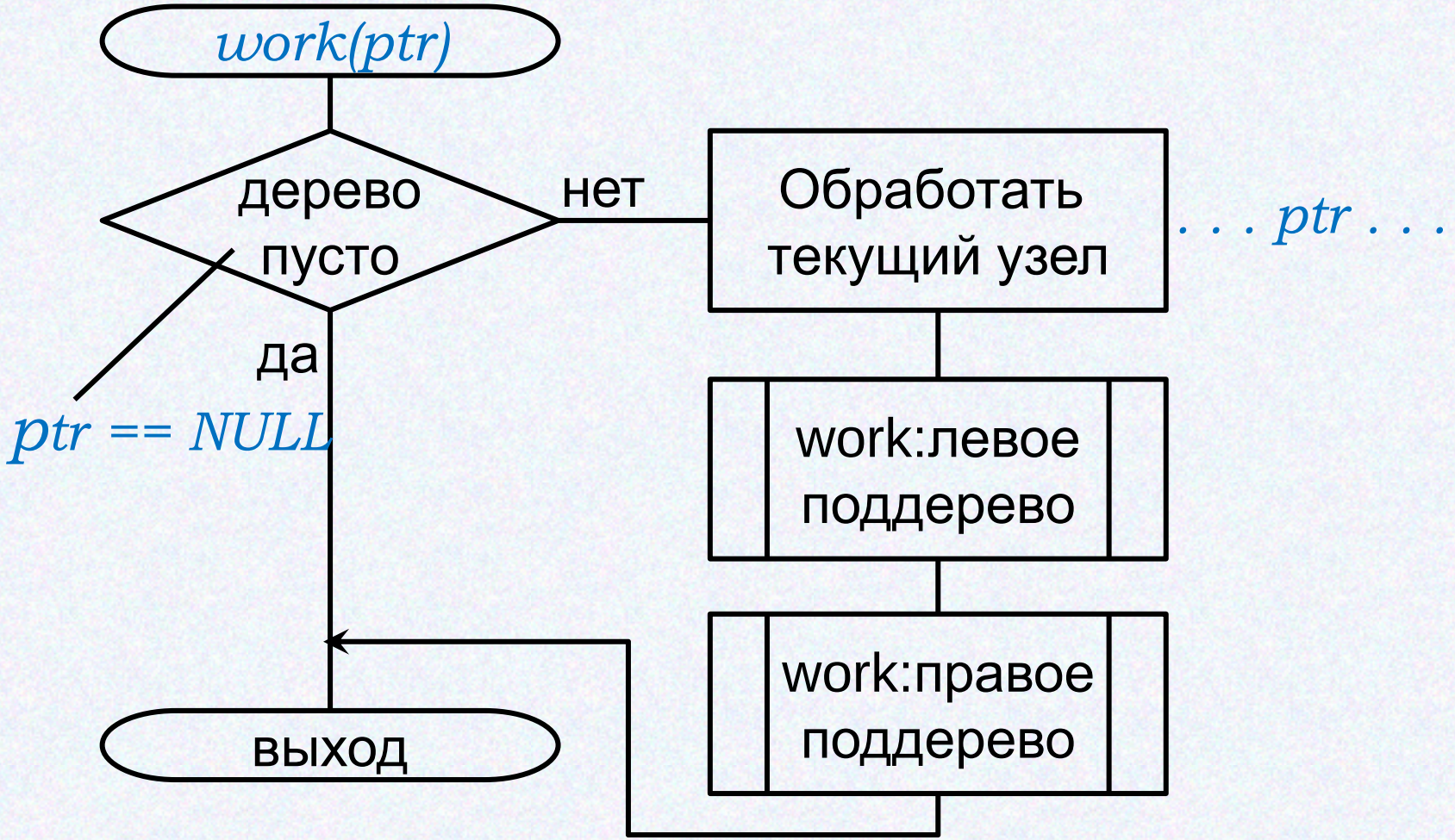
Алгоритм прямого обхода

Рекурсивный алгоритм; *ptr* – указатель на корень дерева



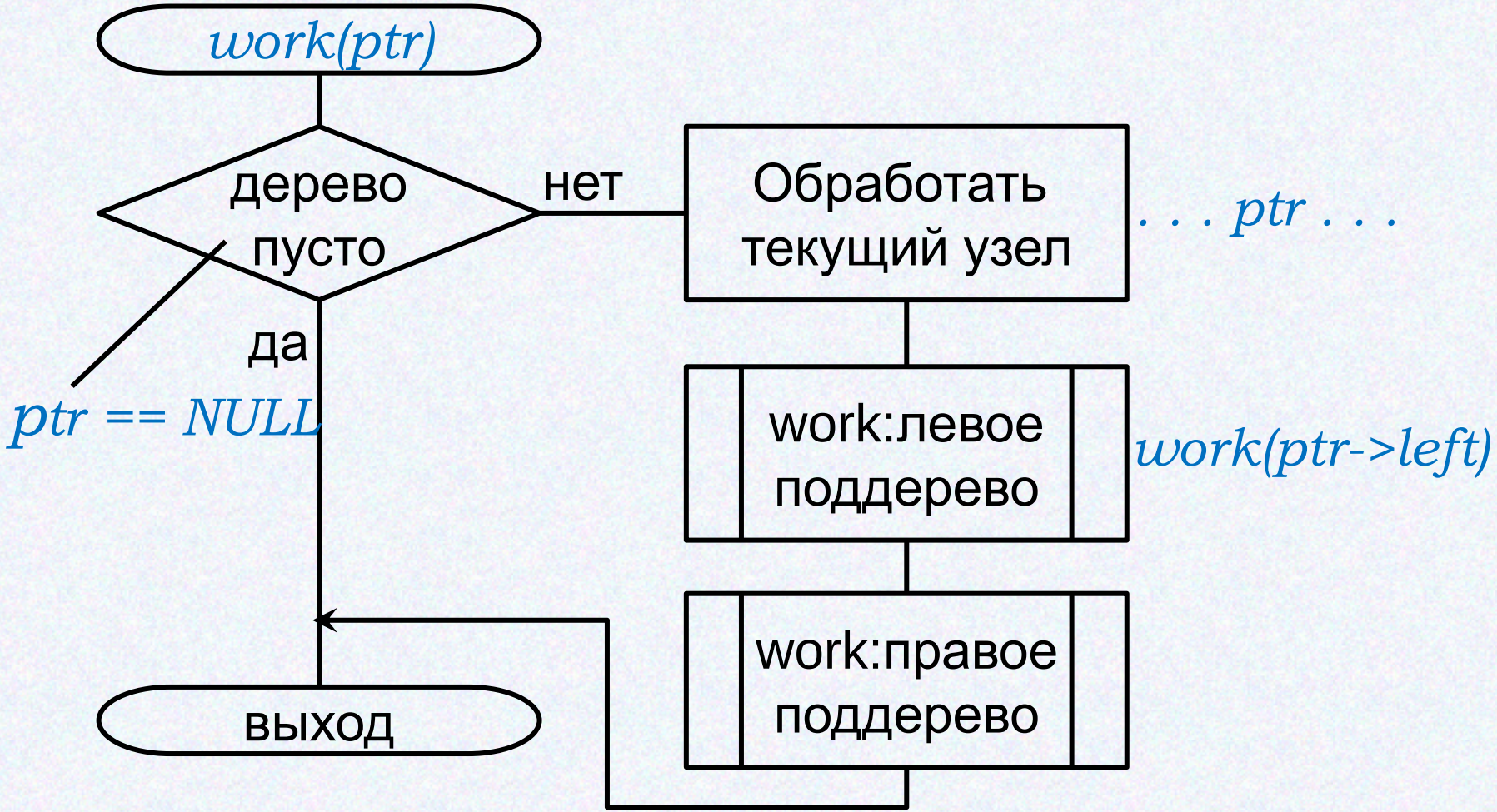
Алгоритм прямого обхода

Рекурсивный алгоритм; *ptr* – указатель на корень дерева



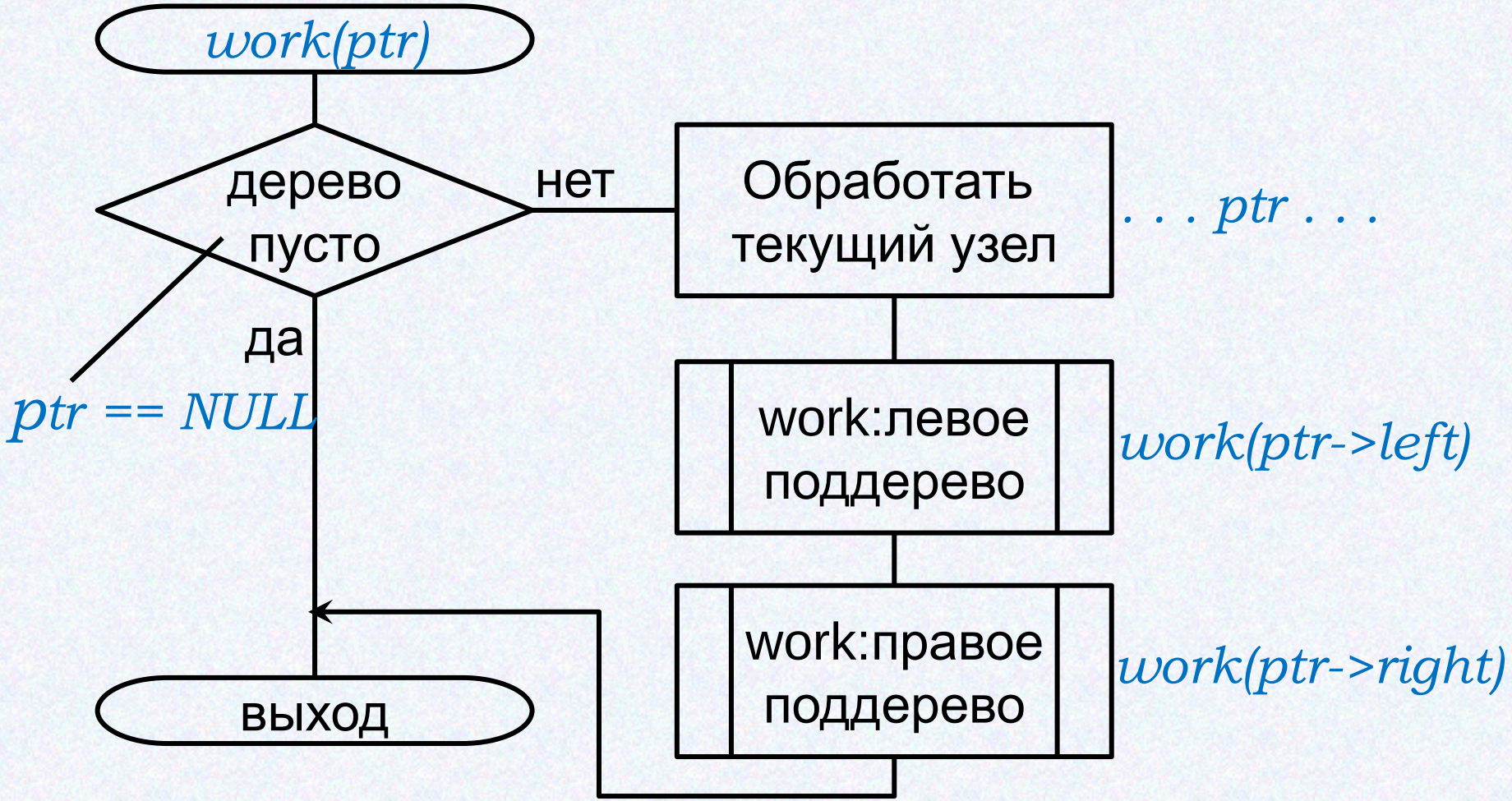
Алгоритм прямого обхода

Рекурсивный алгоритм; *ptr* – указатель на корень дерева



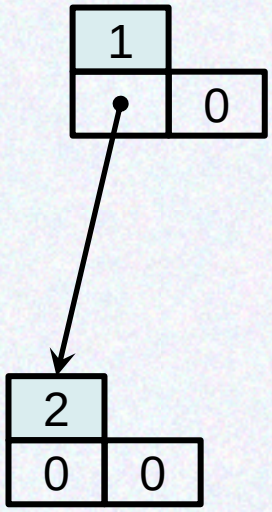
Алгоритм прямого обхода

Рекурсивный алгоритм; *ptr* – указатель на корень дерева



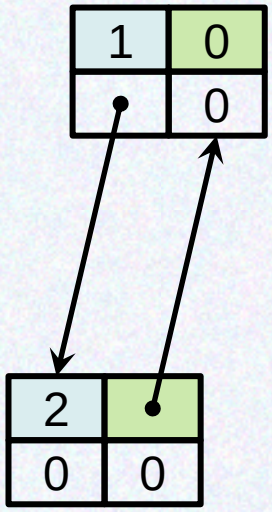
Дополнительные возможности

Добавление указателя на родительскую вершину



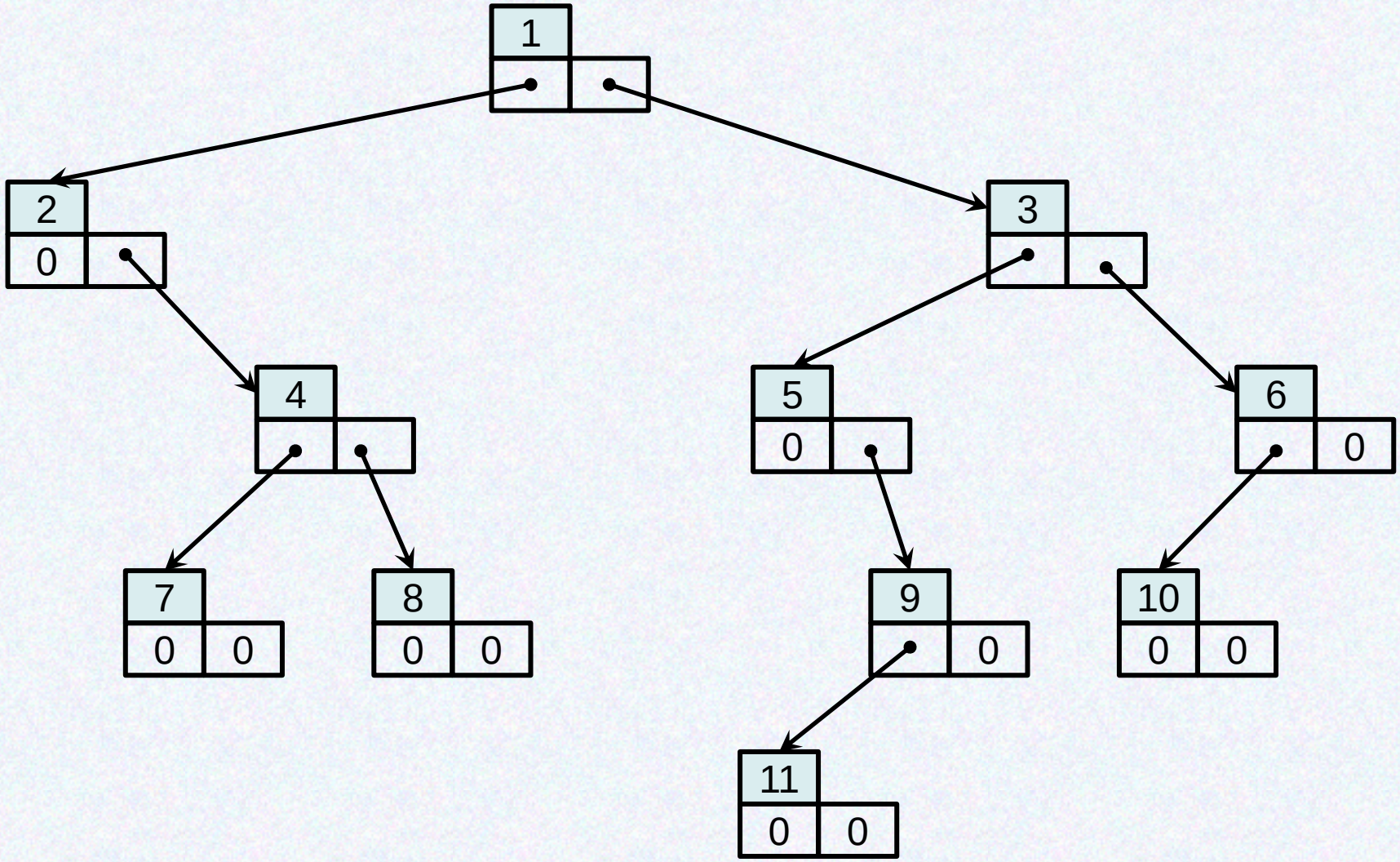
Дополнительные возможности

Добавление указателя на родительскую вершину



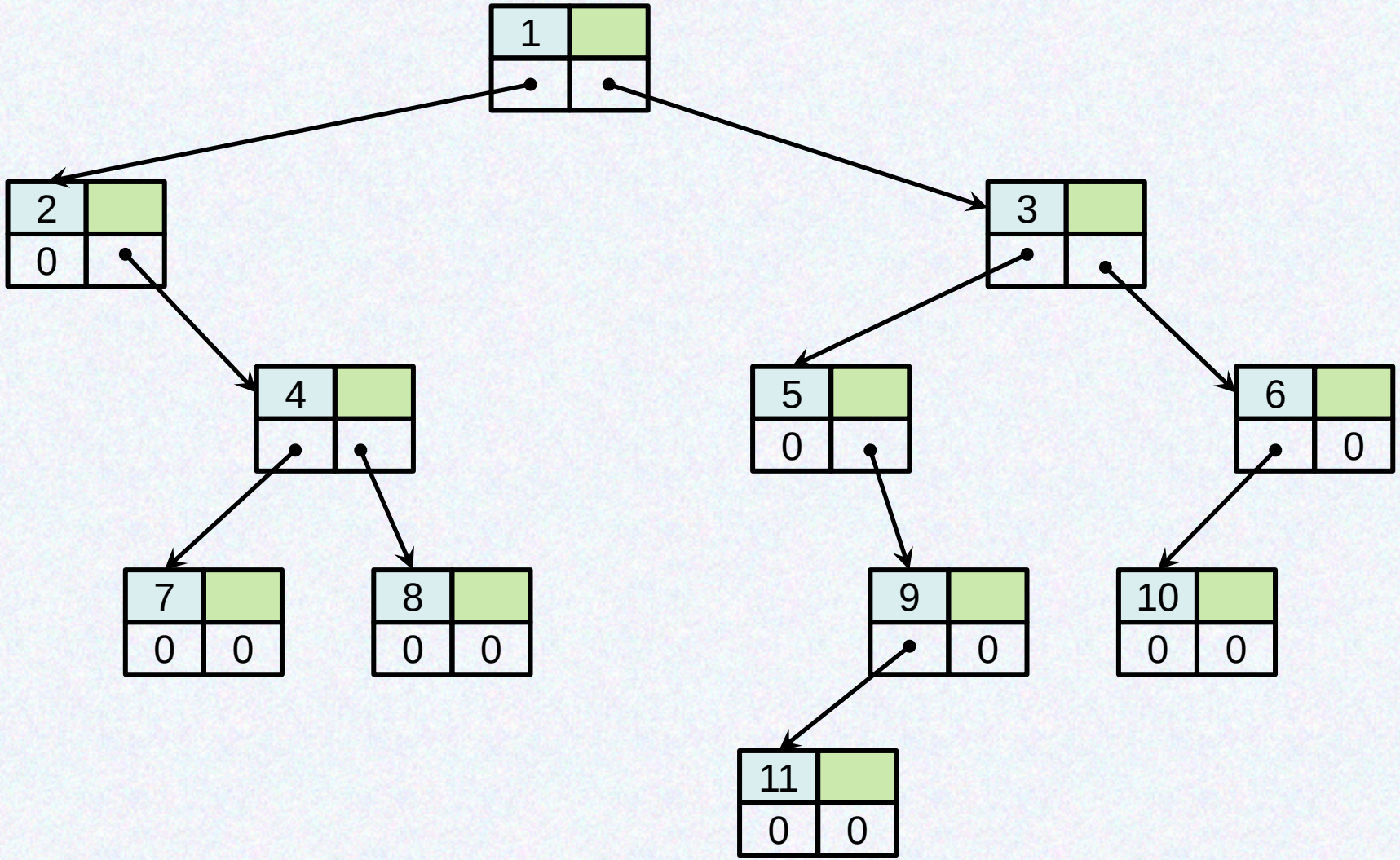
Дополнительные возможности

Добавление указателя на родительскую вершину



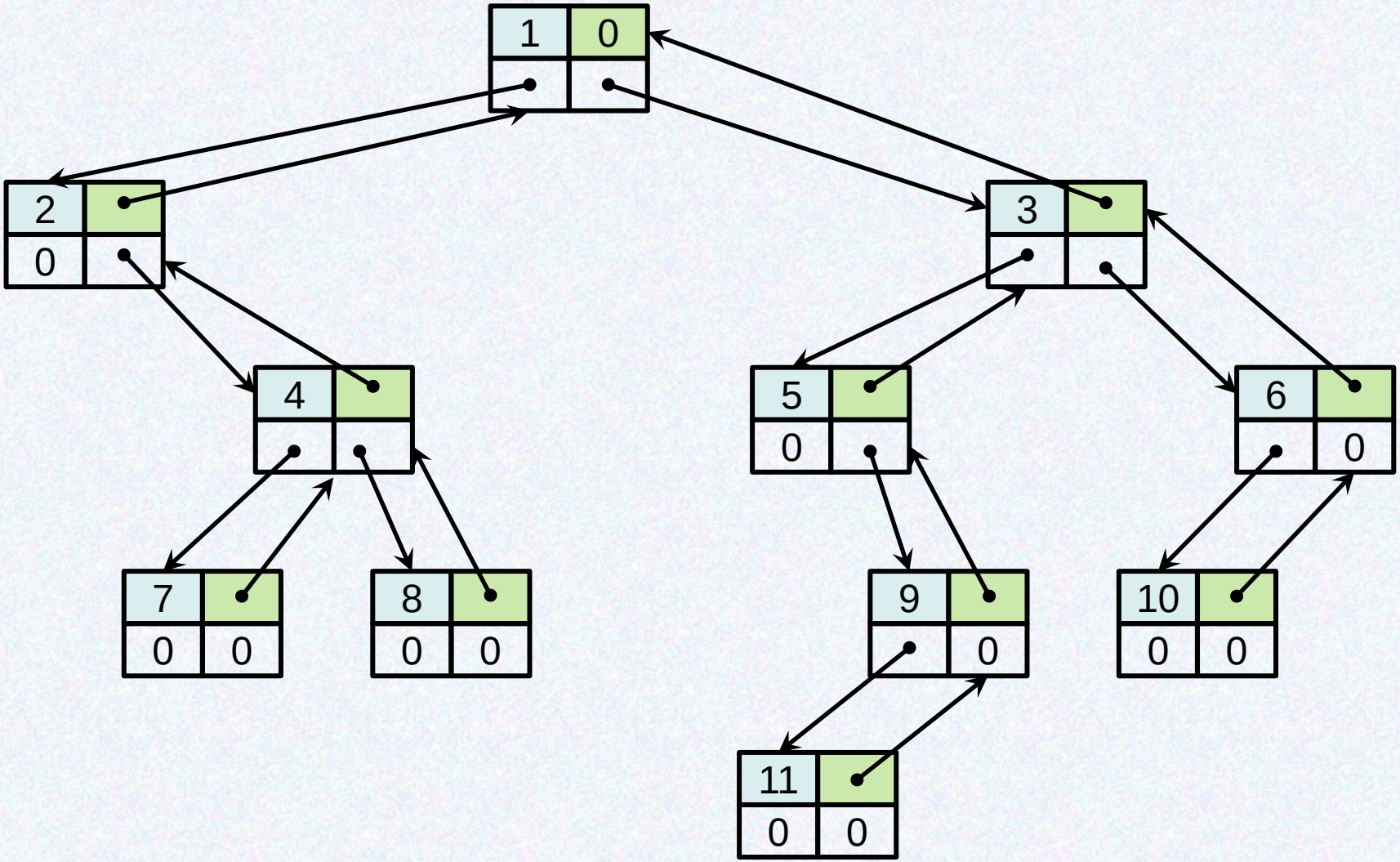
Дополнительные возможности

Добавление указателя на родительскую вершину



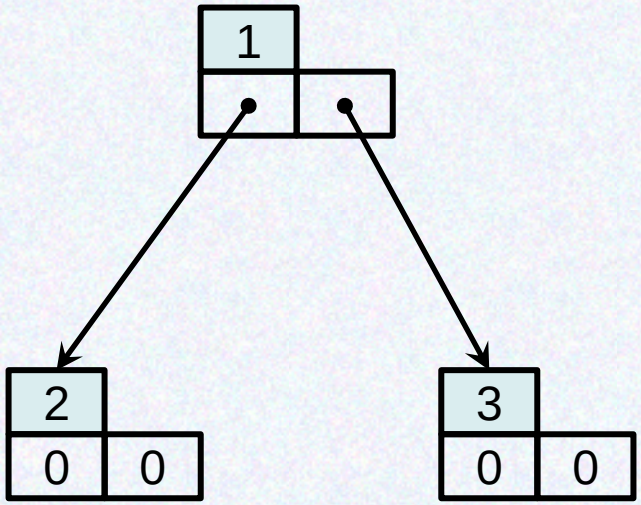
Дополнительные возможности

Добавление указателя на родительскую вершину



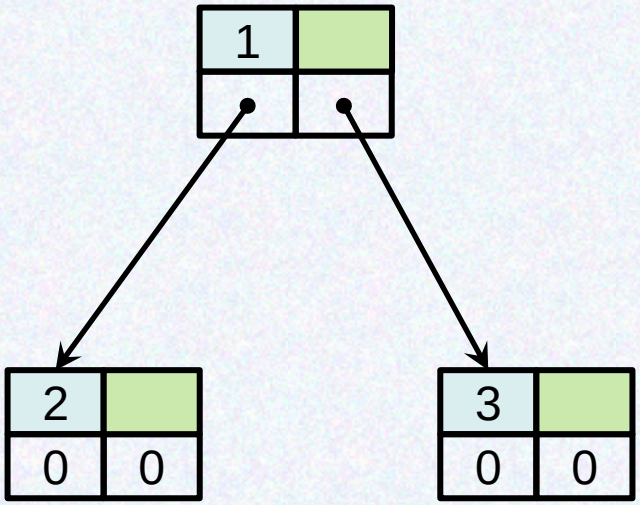
Дополнительные возможности

Добавление нити



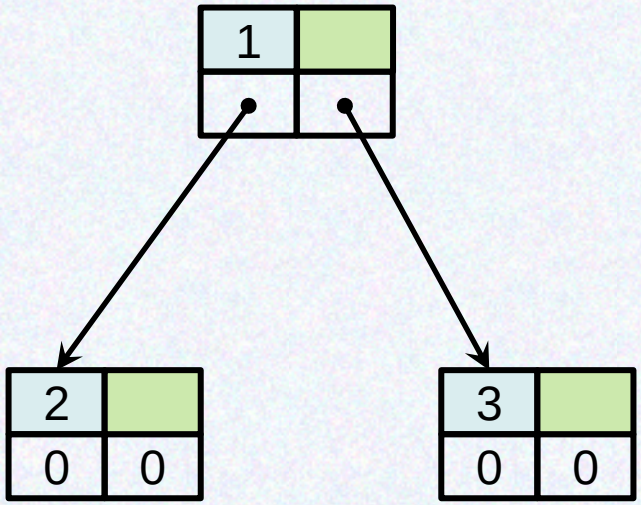
Дополнительные возможности

Добавление нити (прошивка дерева)



Дополнительные возможности

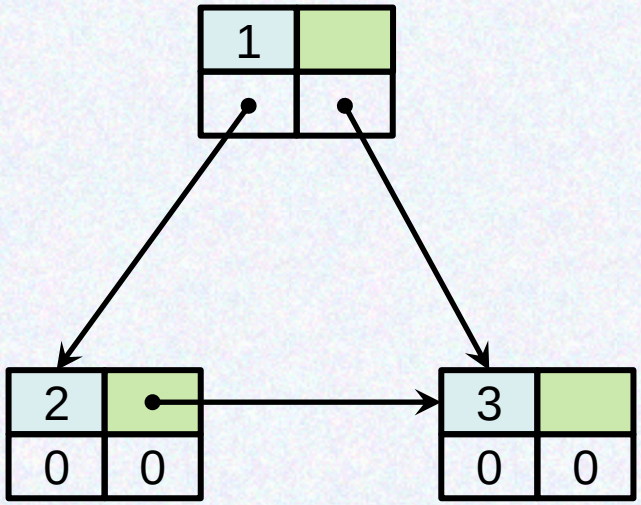
Добавление нити (прошивка дерева)



В соответствии с каким-либо обходом, например, прямым: 1, 2, 3.

Дополнительные возможности

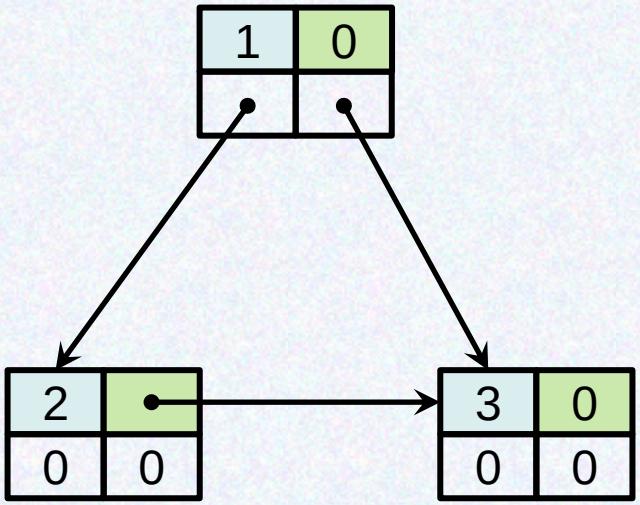
Добавление нити (прошивка дерева)



В соответствии с каким-либо обходом, например, прямым: 1, 2, 3.

Дополнительные возможности

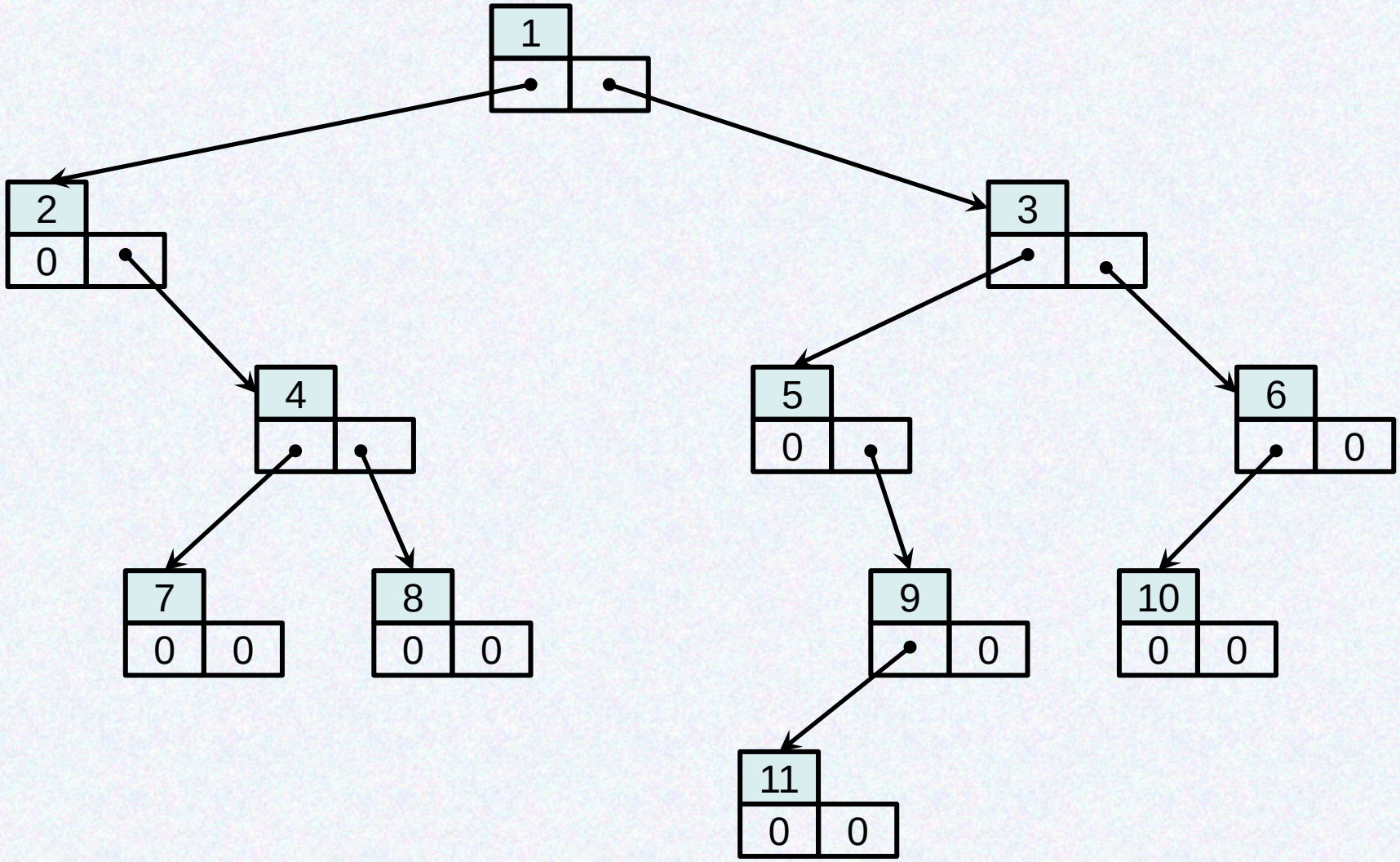
Добавление нити (прошивка дерева)



В соответствии с каким-либо обходом, например, прямым: 1, 2, 3.

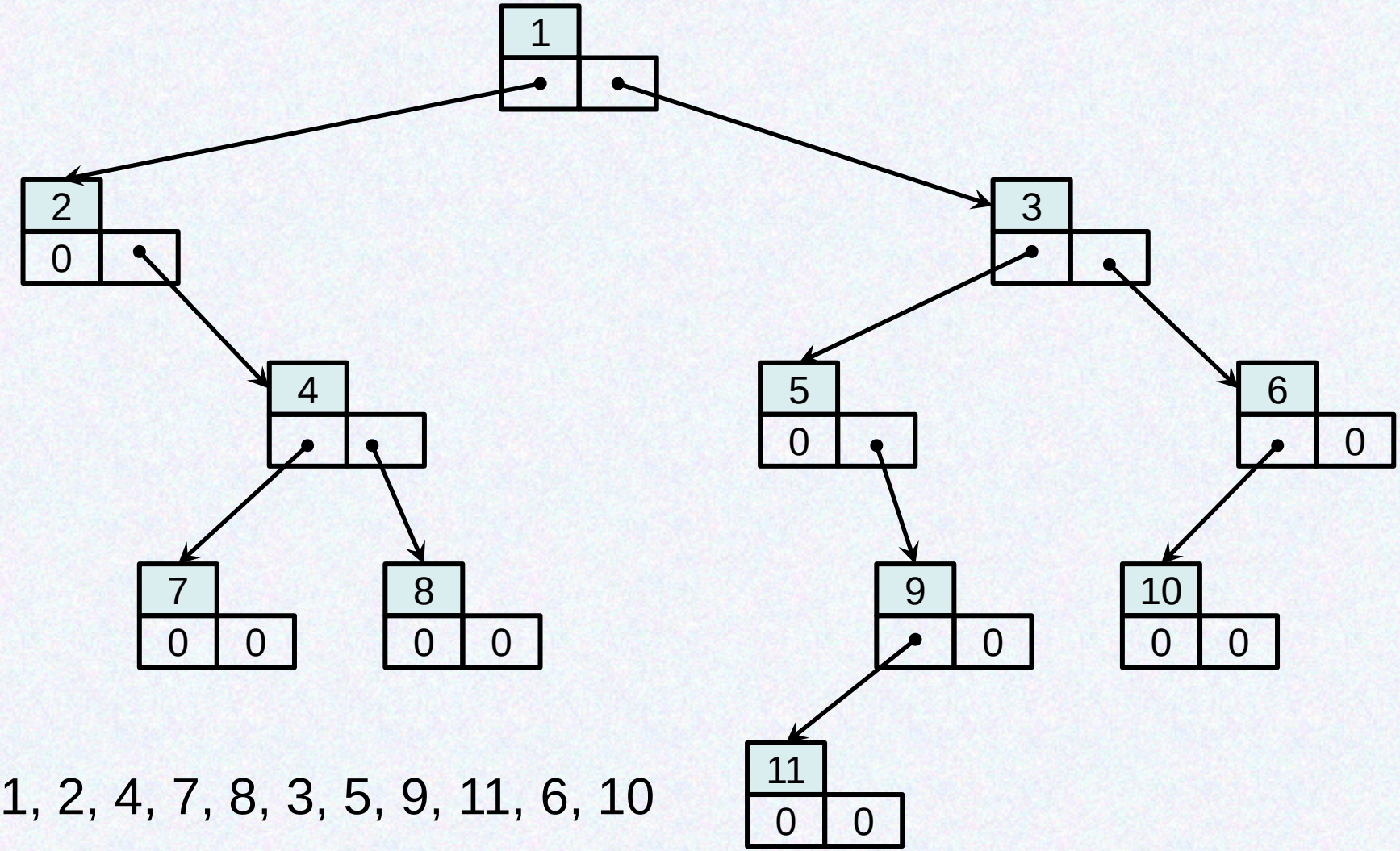
Дополнительные возможности

Добавление нити (прямой обход)



Дополнительные возможности

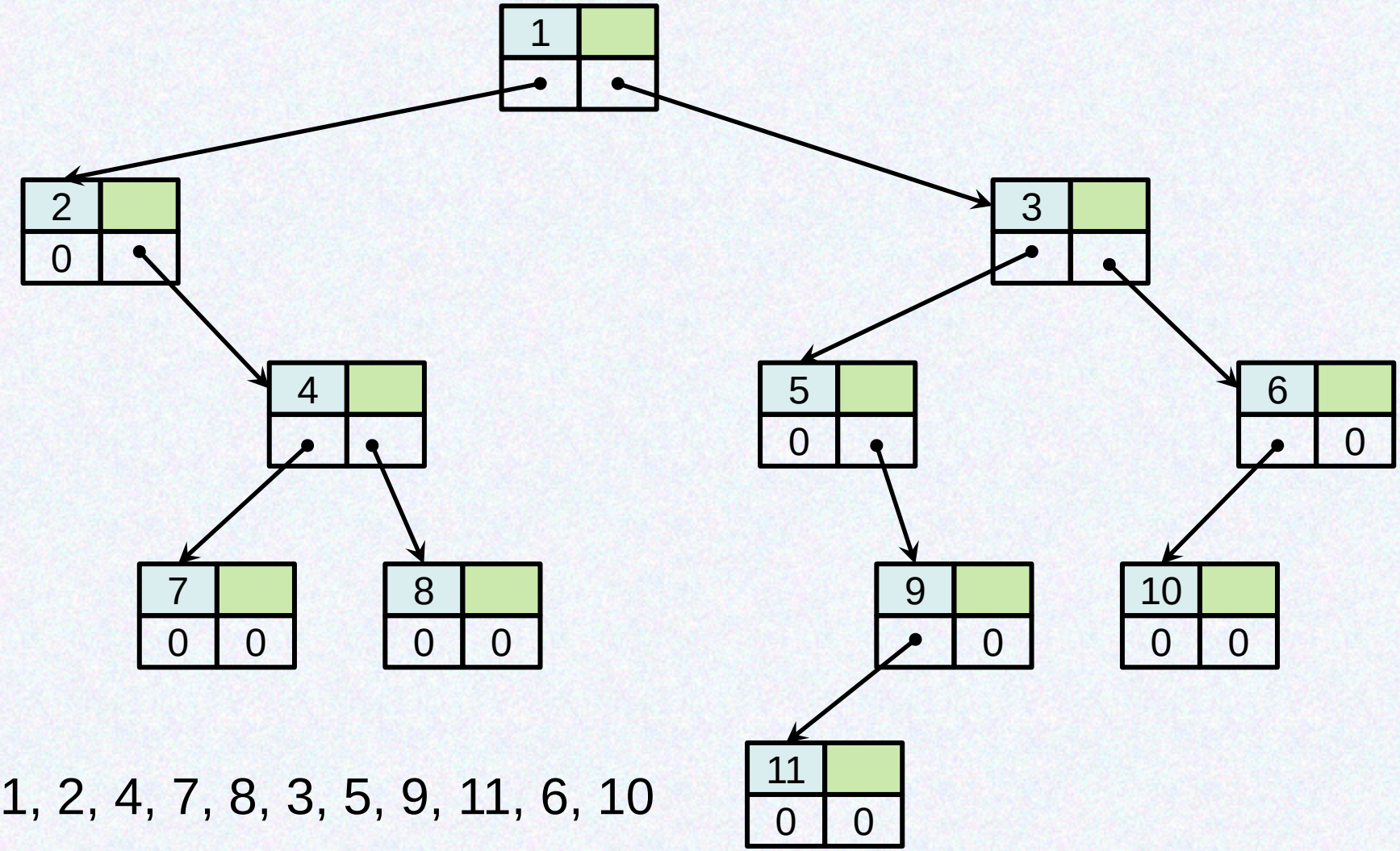
Добавление нити (прямой обход)



1, 2, 4, 7, 8, 3, 5, 9, 11, 6, 10

Дополнительные возможности

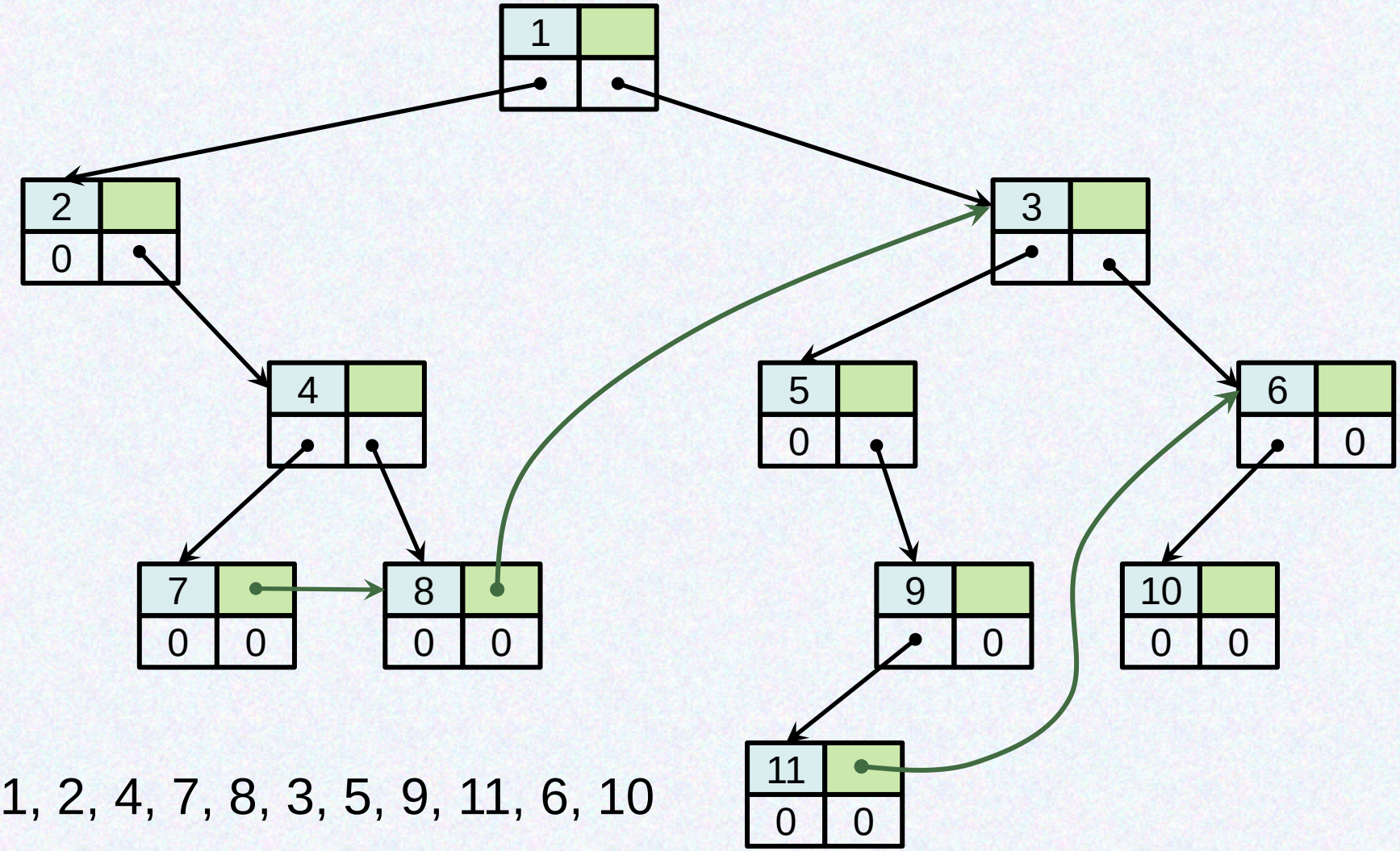
Добавление нити (прямой обход)



1, 2, 4, 7, 8, 3, 5, 9, 11, 6, 10

Дополнительные возможности

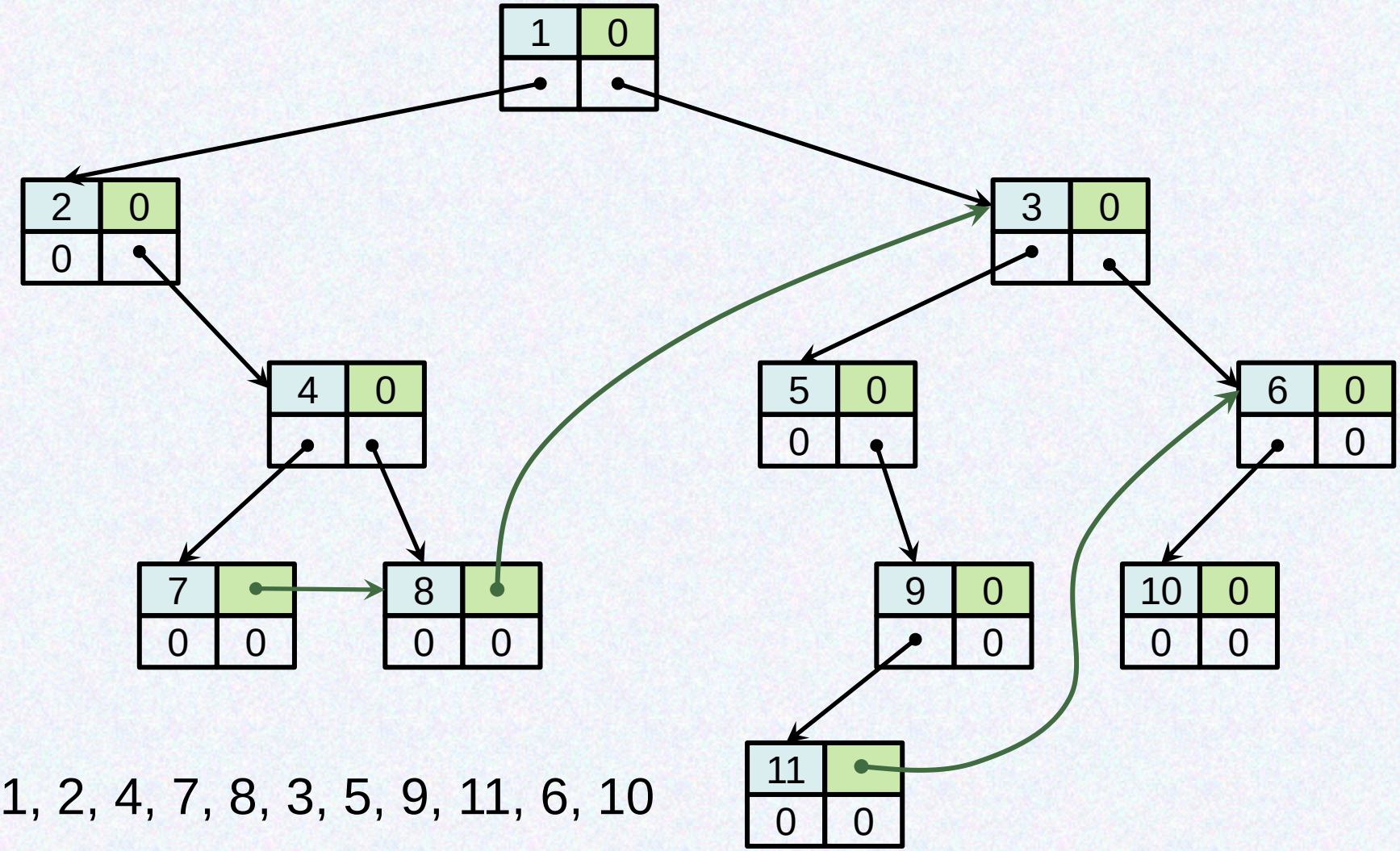
Добавление нити (прямой обход)



1, 2, 4, 7, 8, 3, 5, 9, 11, 6, 10

Дополнительные возможности

Добавление нити (прямой обход)



1, 2, 4, 7, 8, 3, 5, 9, 11, 6, 10

Бинарные деревья поиска

Дано некоторое множество ключей $NR = \{k_i\}$

Бинарное дерево поиска:

Бинарные деревья поиска

Дано некоторое множество ключей $NR = \{k_i\}$

Бинарное дерево поиска:

- каждая вершина дерева помечена отдельным ключом,

Бинарные деревья поиска

Дано некоторое множество ключей $NR = \{k_i\}$

Бинарное дерево поиска:

- каждая вершина дерева помечена отдельным ключом,
- для любой вершины x с ключом k_x:

Бинарные деревья поиска

Дано некоторое множество ключей $NR = \{k_i\}$

Бинарное дерево поиска:

- каждая вершина дерева помечена отдельным ключом,
- для любой вершины x с ключом k_x:
  - для любого $k_i \in$ Л$[x]$ $k_i \leq k_x$,

Бинарные деревья поиска

Дано некоторое множество ключей $NR = \{k_i\}$

Бинарное дерево поиска:

- каждая вершина дерева помечена отдельным ключом,
- для любой вершины x с ключом k_x:
  - для любого $k_i \in$ Л$[x]$ $k_i \leq k_x$,
  - для любого $k_i \in$ П$[x]$ $k_i \geq k_x$

Бинарные деревья поиска

Дано некоторое множество ключей $NR = \{k_i\}$

Бинарное дерево поиска:

- каждая вершина дерева помечена отдельным ключом,
- для любой вершины x с ключом k_x:
  - для любого $k_i \in$ Л$[x]$ $k_i \leq k_x$,
  - для любого $k_i \in$ П$[x]$ $k_i \geq k_x$

Л$[x]$, П$[x]$ – левое и правое поддеревья узла x

Бинарные деревья поиска

Типы вершин дерева:

- корень
- промежуточные вершины
- листья

Бинарные деревья поиска

Типы вершин дерева:

- корень
- промежуточные вершины
- листья

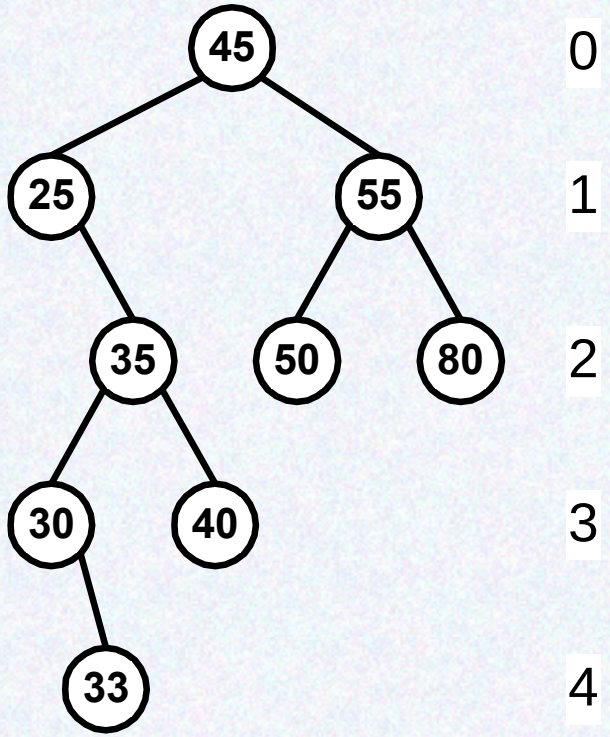
Уровень вершины дерева – расстояние от корня до данной вершины

Бинарные деревья поиска

Типы вершин дерева:

- корень
- промежуточные вершины
- листья

Уровень вершины дерева – расстояние от корня до данной вершины



Высота дерева = 4

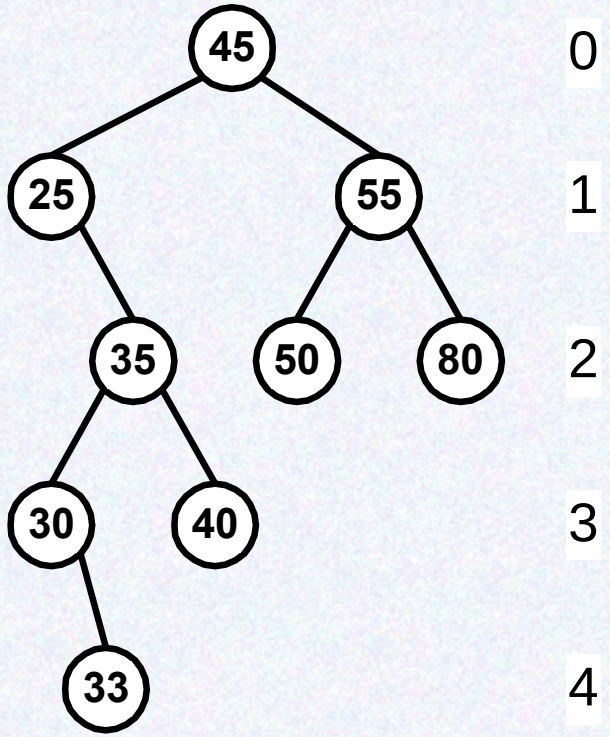
Бинарные деревья поиска

Типы вершин дерева:

- корень
- промежуточные вершины
- листья

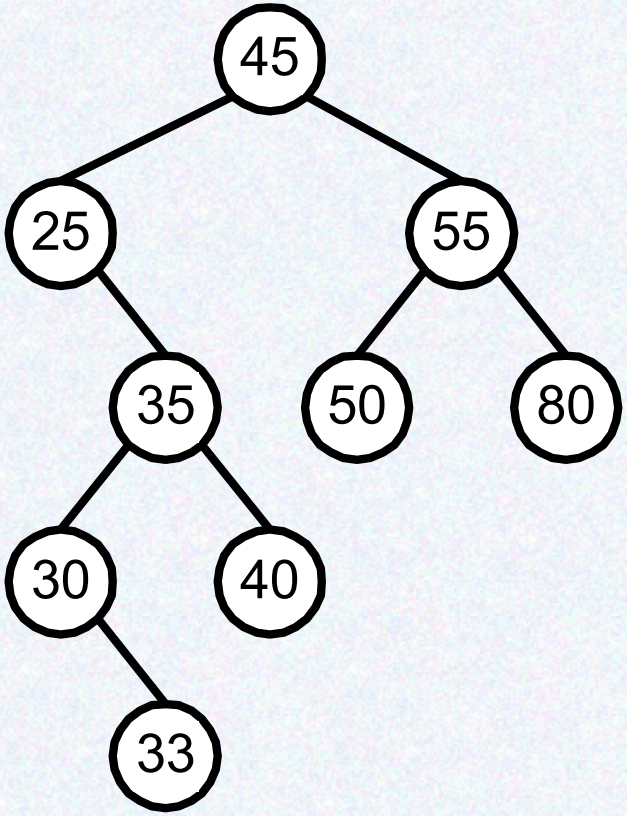
Уровень вершины дерева – расстояние от корня до данной вершины

Высота дерева – максимальный уровень вершины

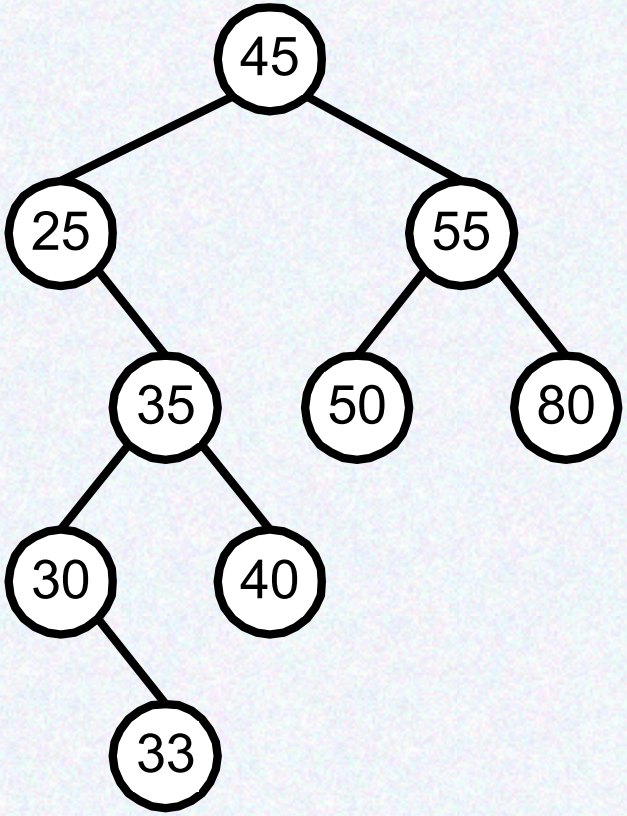


Высота дерева = 4

Обход дерева

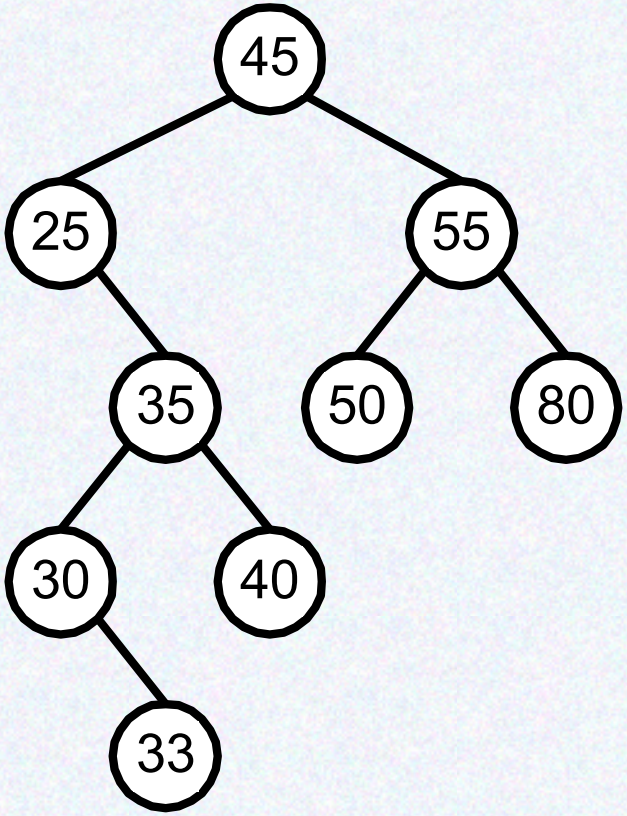


Обход дерева



Центрированный обход дерева:

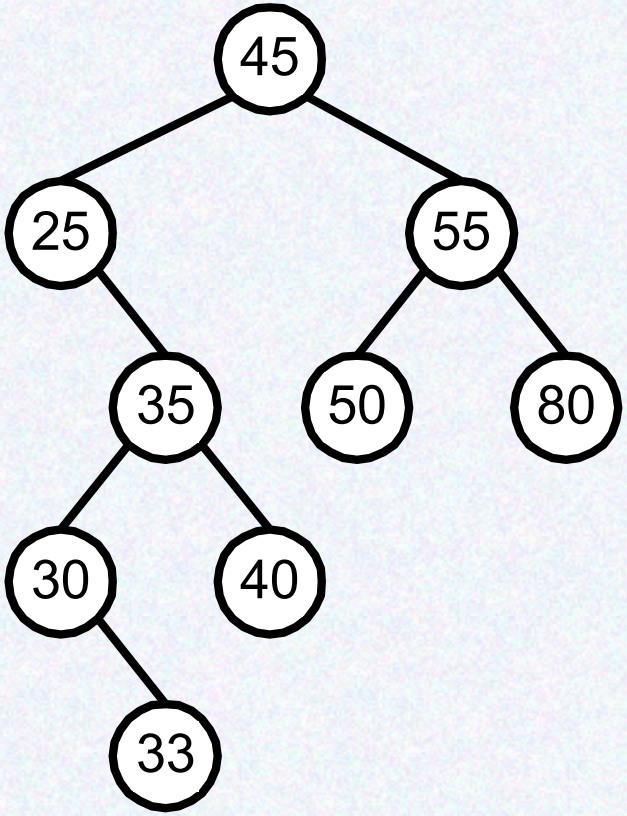
Обход дерева



Центрированный обход дерева:

25,

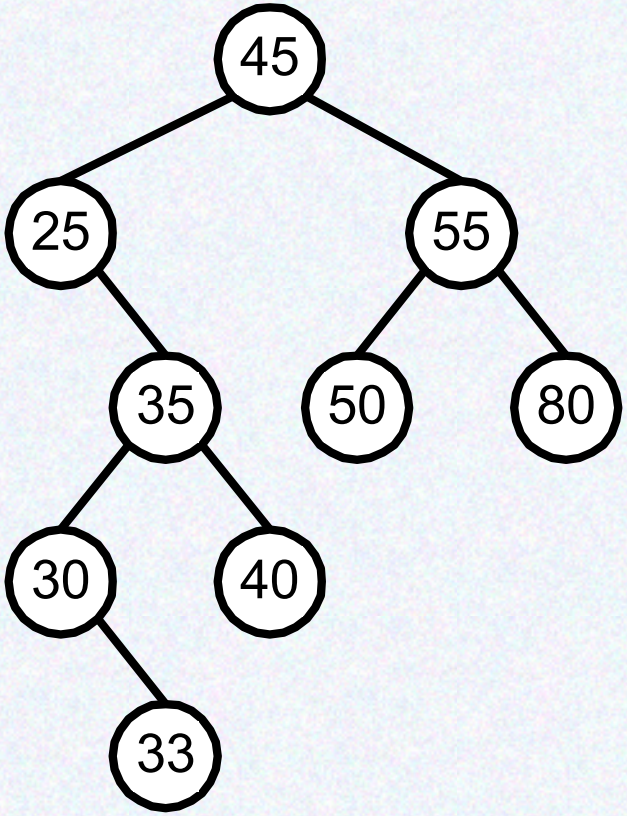
Обход дерева



Центрированный обход дерева:

25, 30,

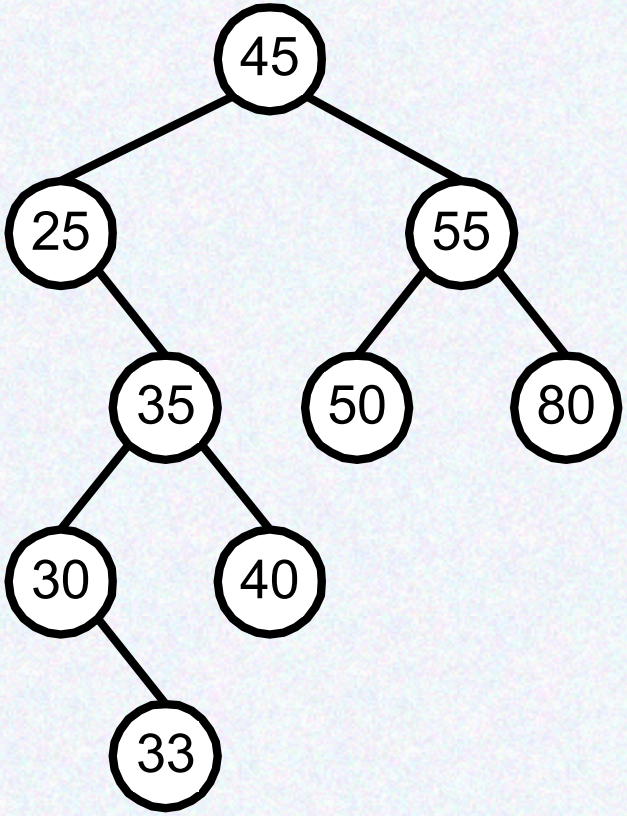
Обход дерева



Центрированный обход дерева:

25, 30, 33,

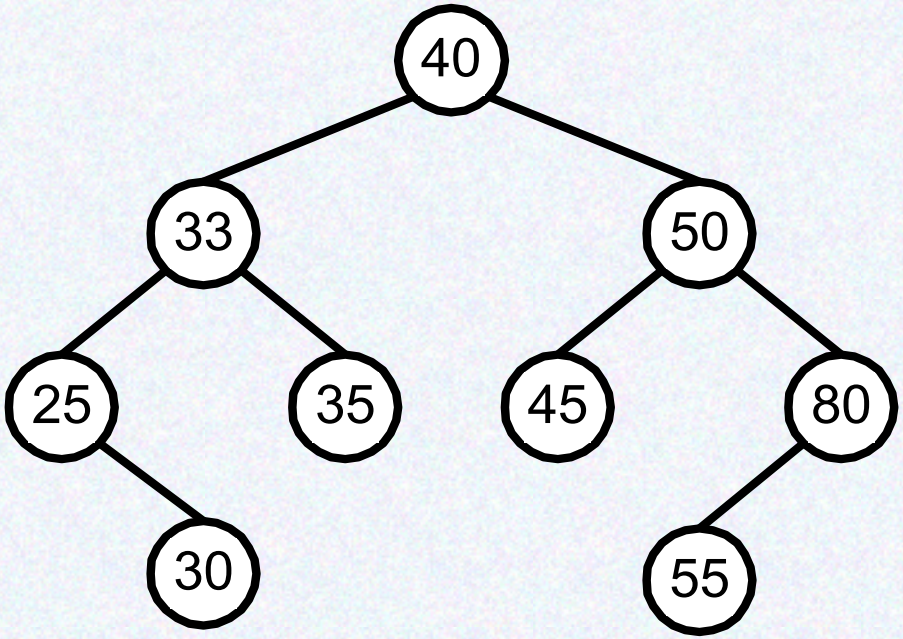
Обход дерева



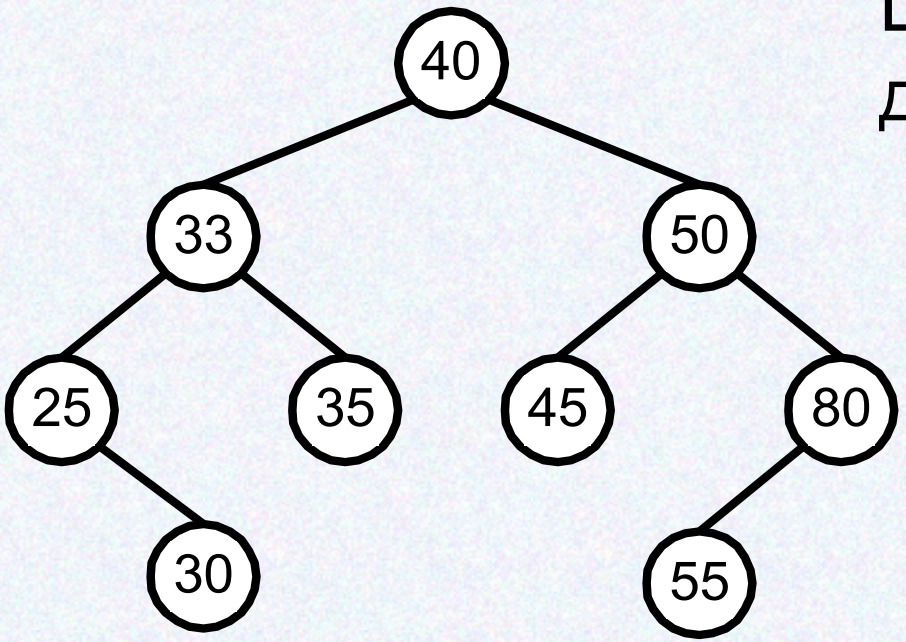
Центрированный обход дерева:

25, 30, 33, 35, 40,
45, 50, 55, 80

Обход дерева



Обход дерева



Центрированный обход дерева:

25, 30, 33, 35, 40, 45, 50, 55, 80

Операции поиска

Операции поиска

- Поиск элемента по заданному ключу

Операции поиска

- Поиск элемента по заданному ключу
- Поиск минимального и максимального элементов

Операции поиска

- Поиск элемента по заданному ключу
- Поиск минимального и максимального элементов
- Поиск предшествующего и последующего элементов

Алгоритм поиска по ключу

Рекурсивный алгоритм

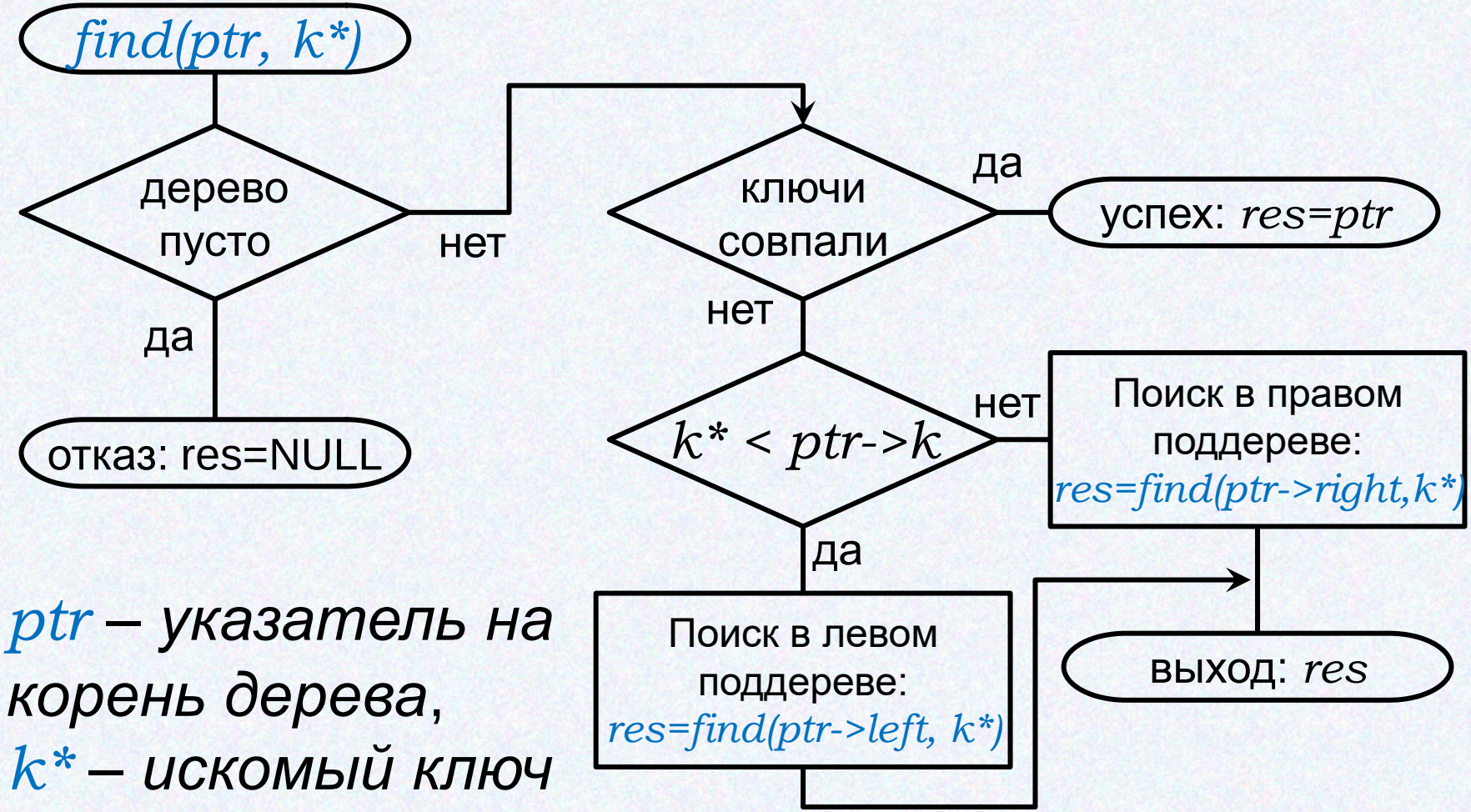
Алгоритм поиска по ключу

Рекурсивный алгоритм

ptr – *указатель на корень дерева*,
k^* – *искомый ключ*

Алгоритм поиска по ключу

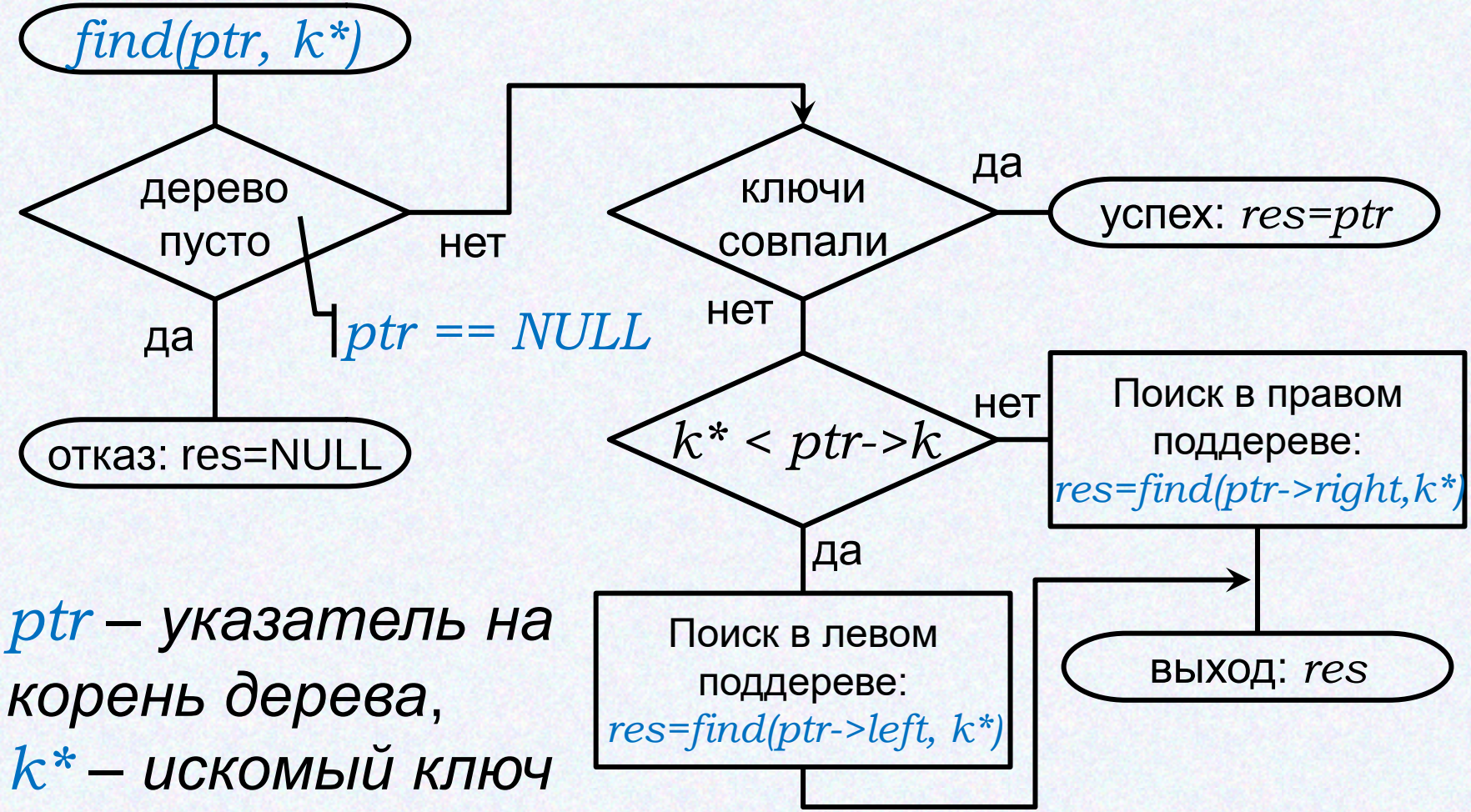
Рекурсивный алгоритм



ptr – указатель на корень дерева,
k – искомый ключ*

Алгоритм поиска по ключу

Рекурсивный алгоритм

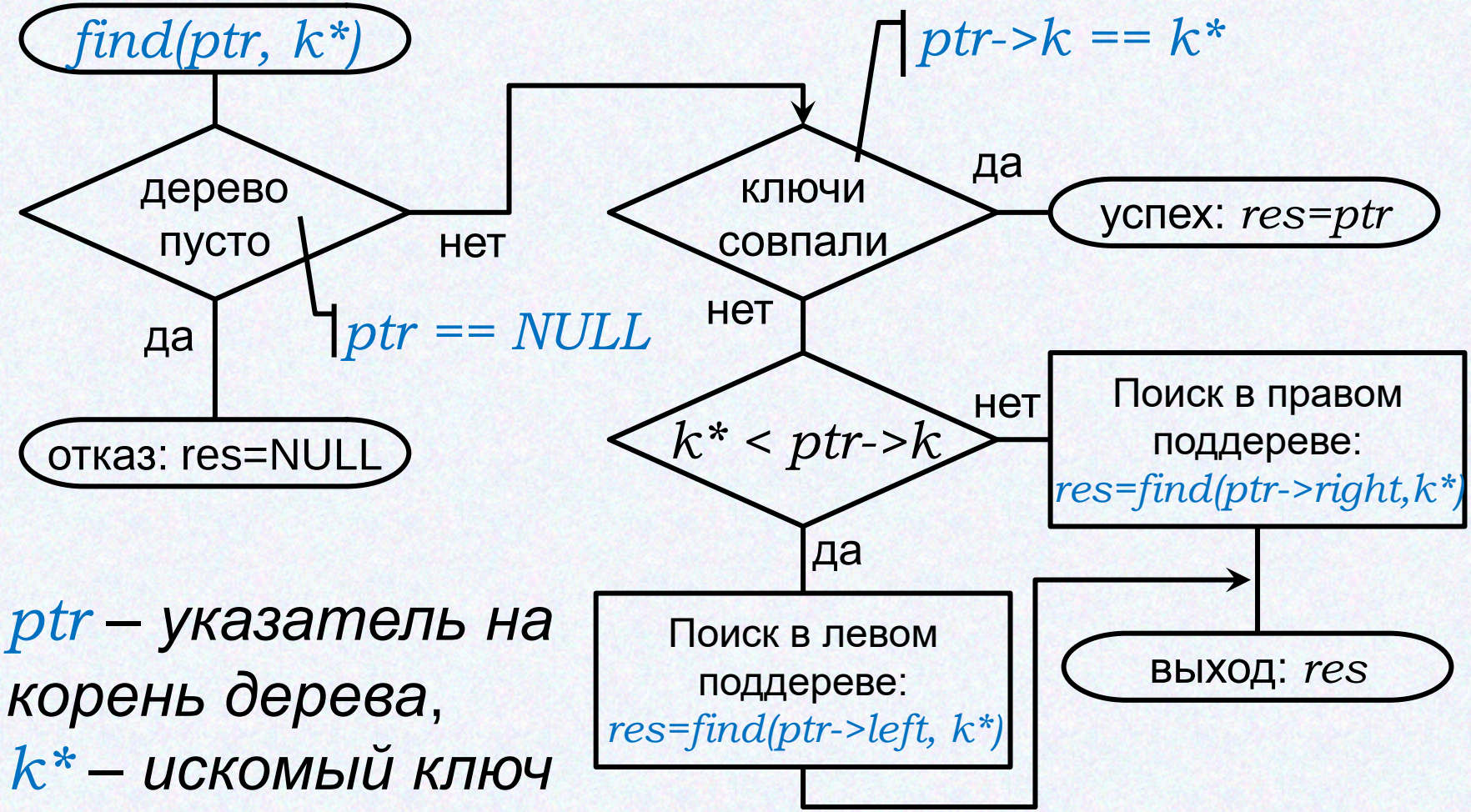


ptr – *указатель на корень дерева,*

*k** – *искомый ключ*

Алгоритм поиска по ключу

Рекурсивный алгоритм



ptr – *указатель на корень дерева*,
*k** – *искомый ключ*

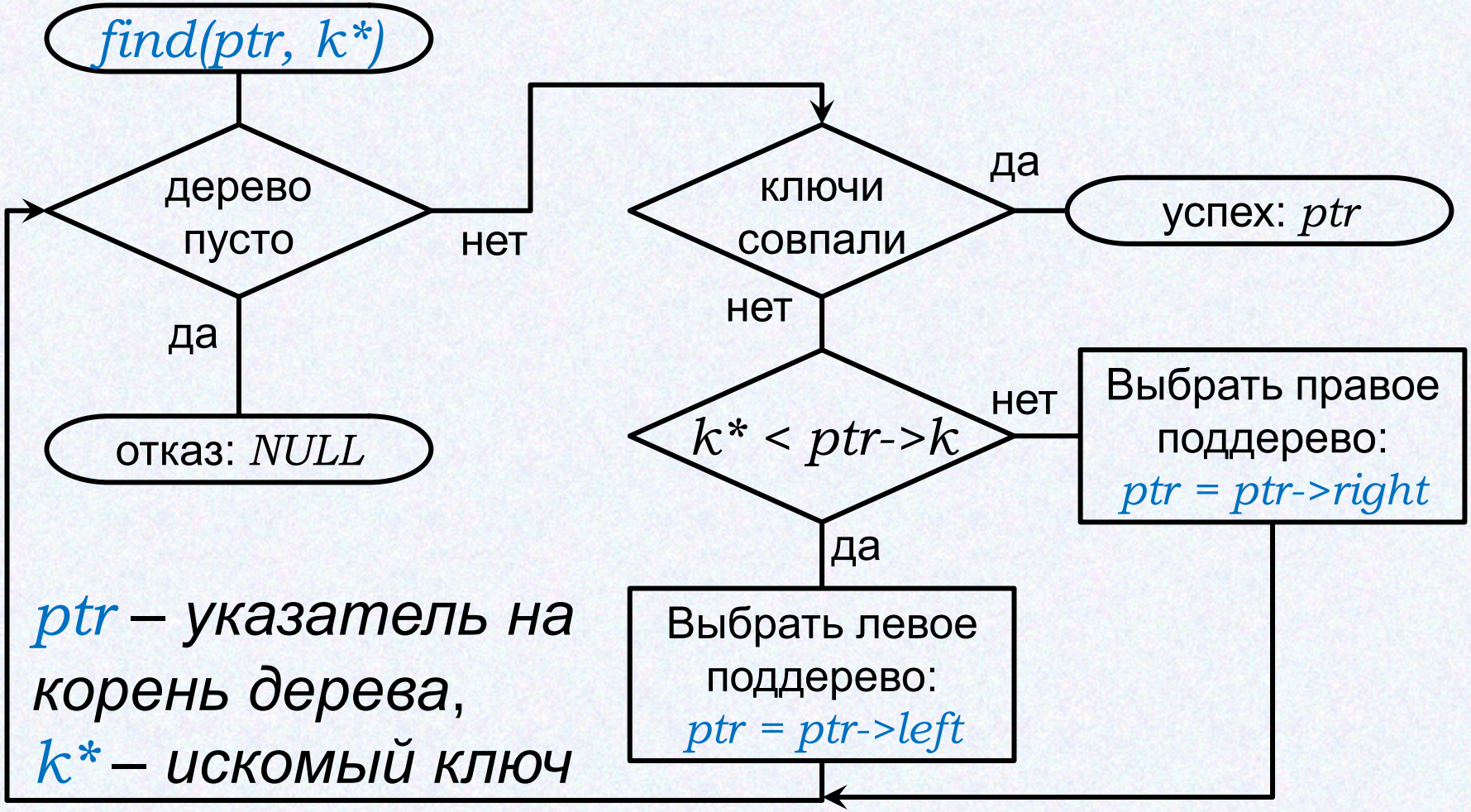
Алгоритм поиска по ключу

Итерационный алгоритм

ptr – *указатель на корень дерева*,
k^* – *искомый ключ*

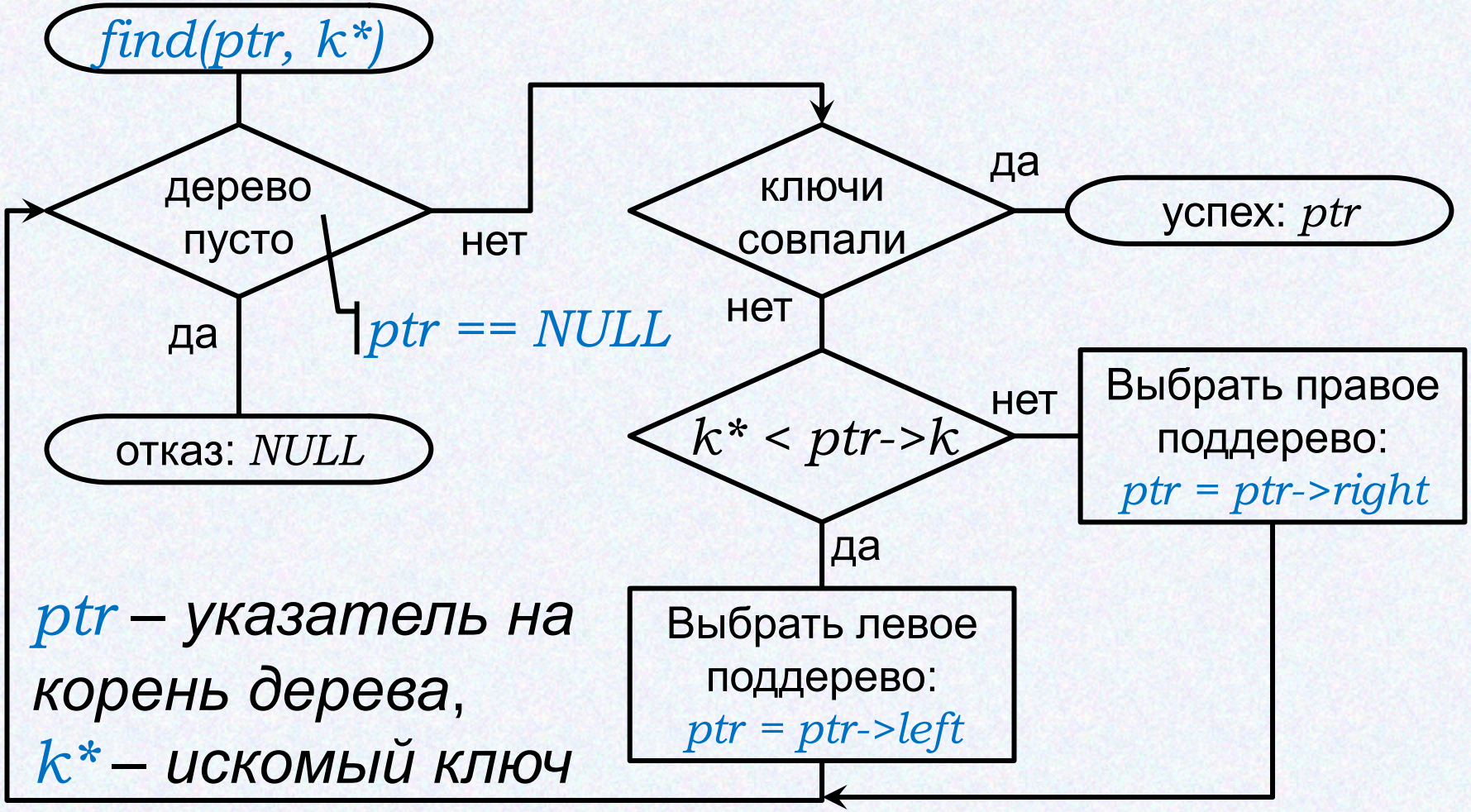
Алгоритм поиска по ключу

Итерационный алгоритм



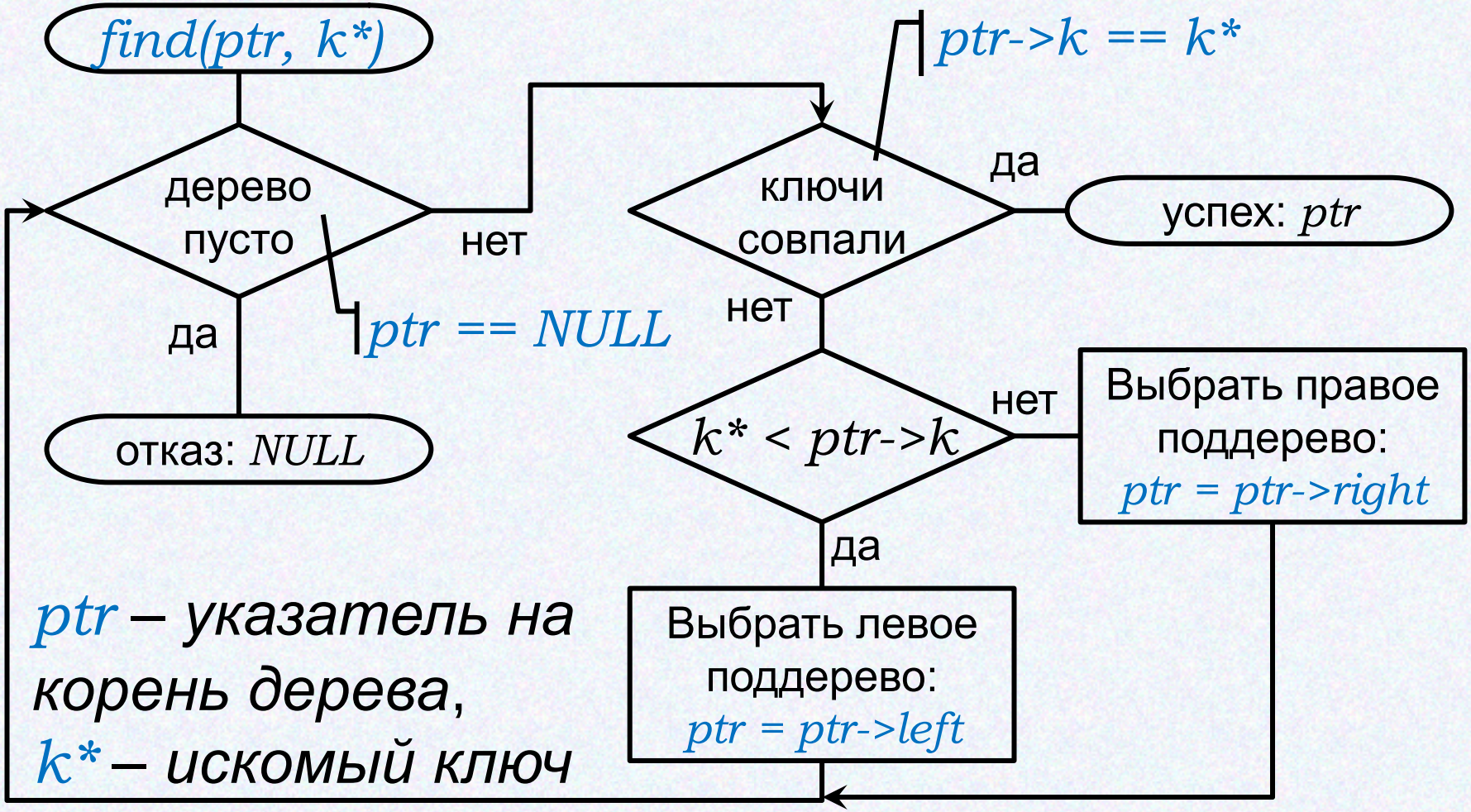
Алгоритм поиска по ключу

Итерационный алгоритм



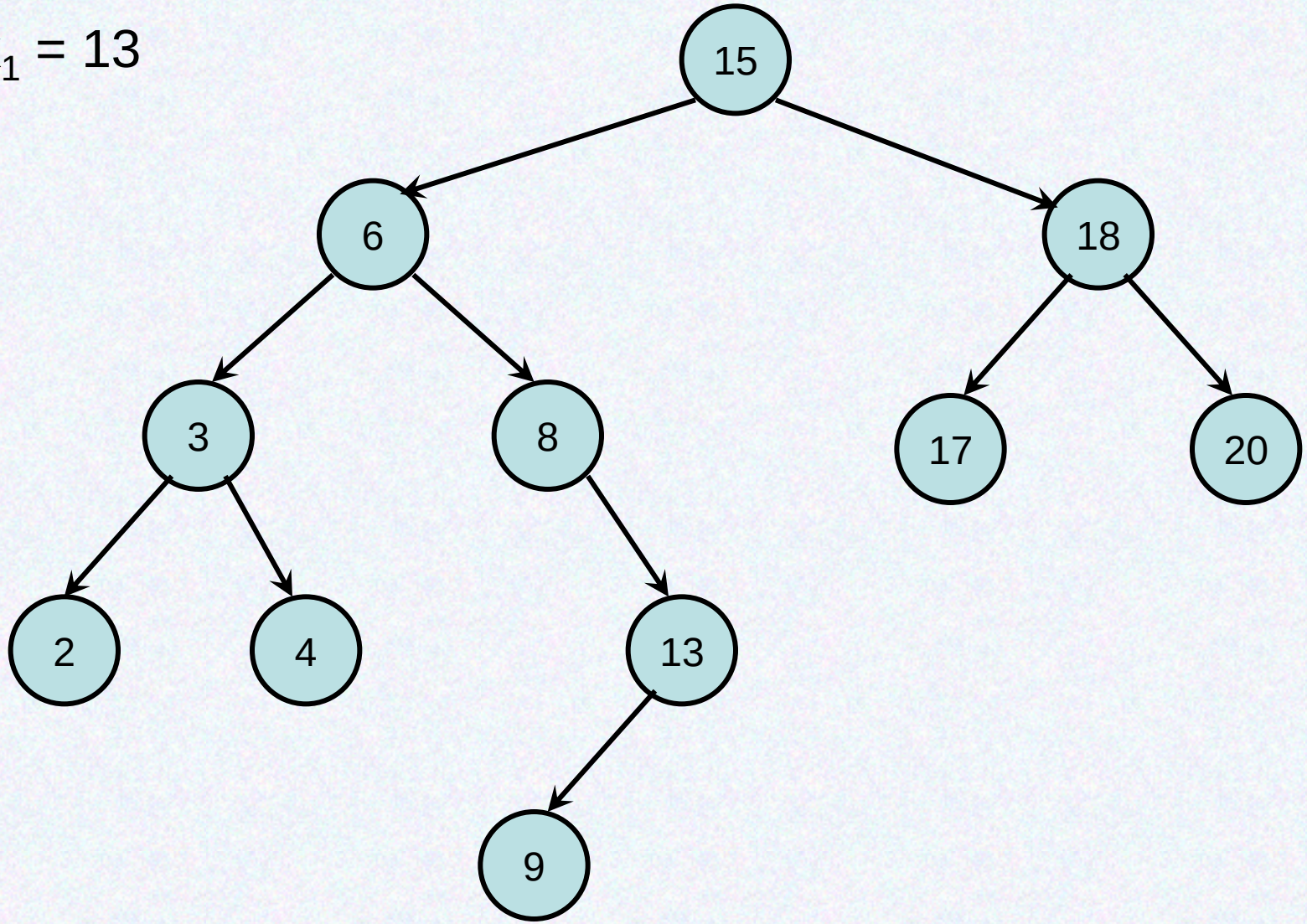
Алгоритм поиска по ключу

Итерационный алгоритм



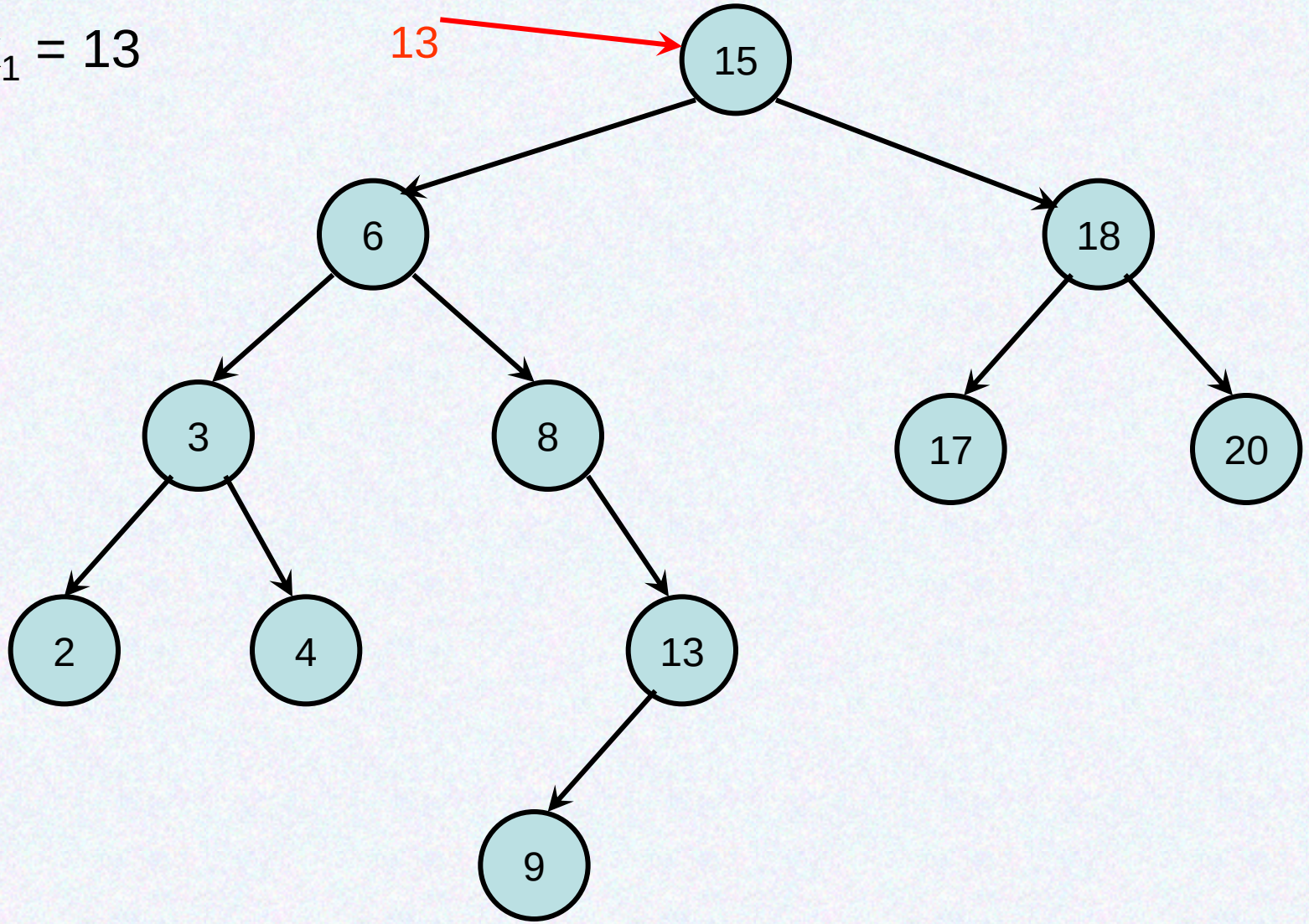
Алгоритм поиска по ключу

$k_1 = 13$



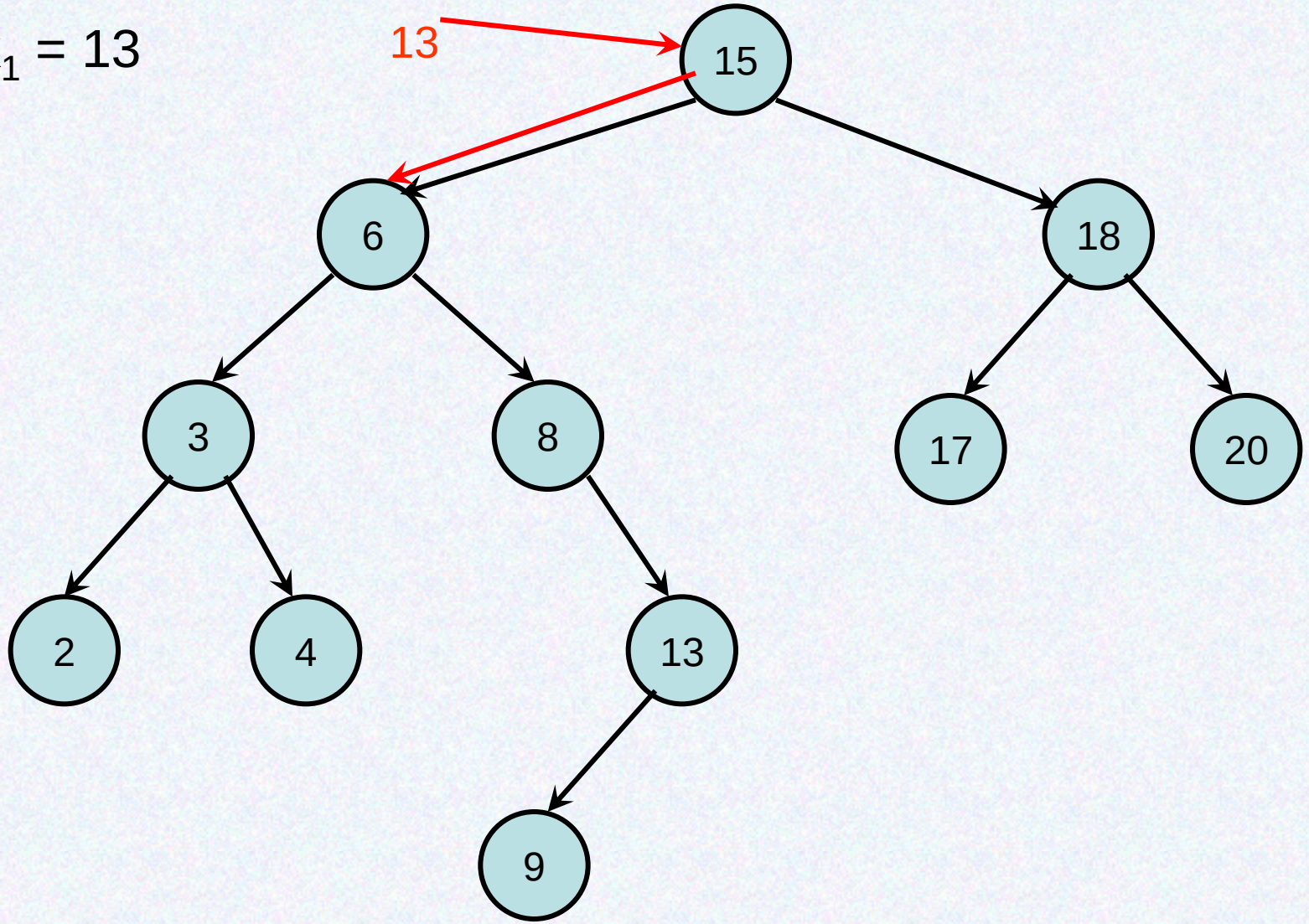
Алгоритм поиска по ключу

$k_1 = 13$



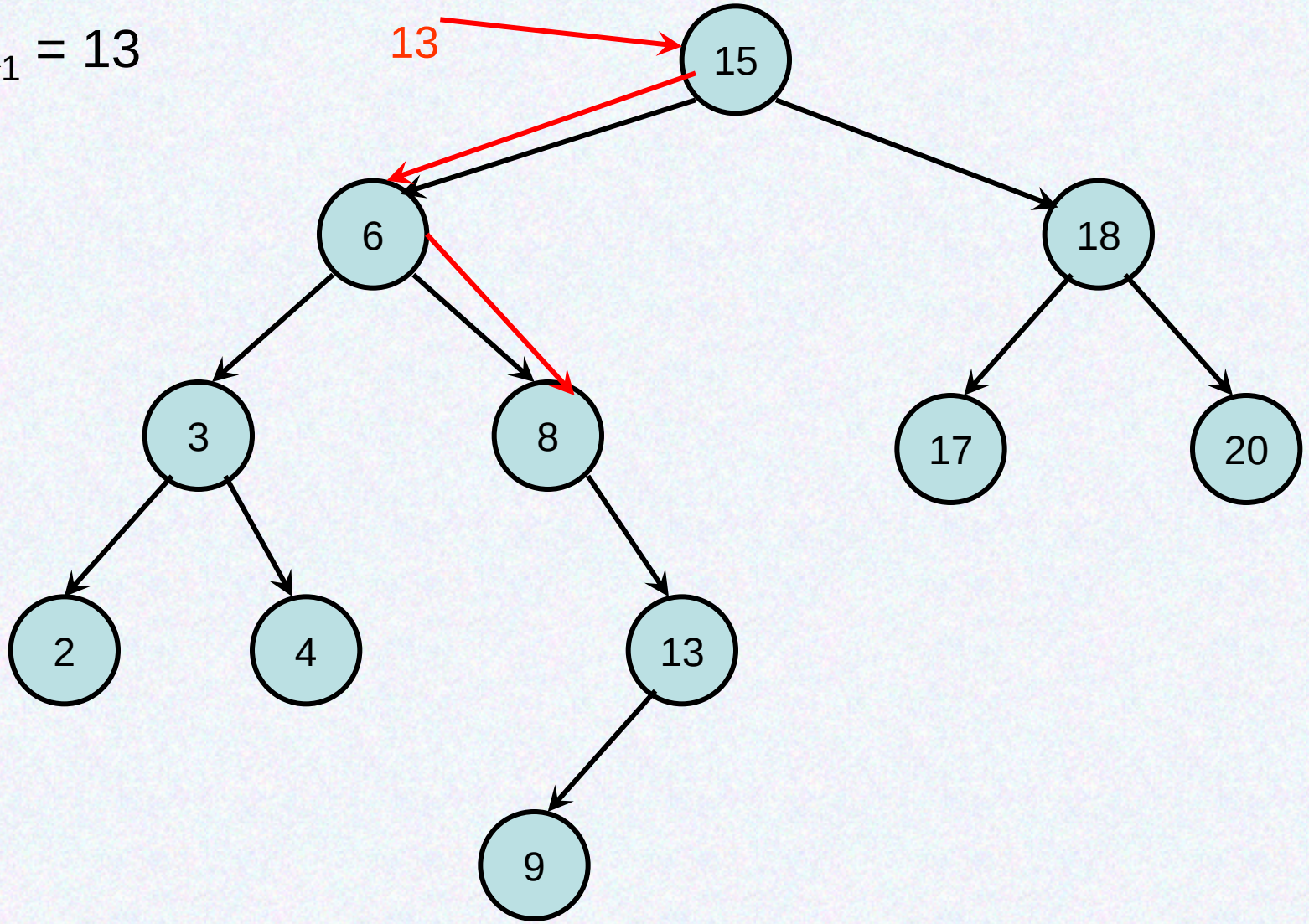
Алгоритм поиска по ключу

$k_1 = 13$



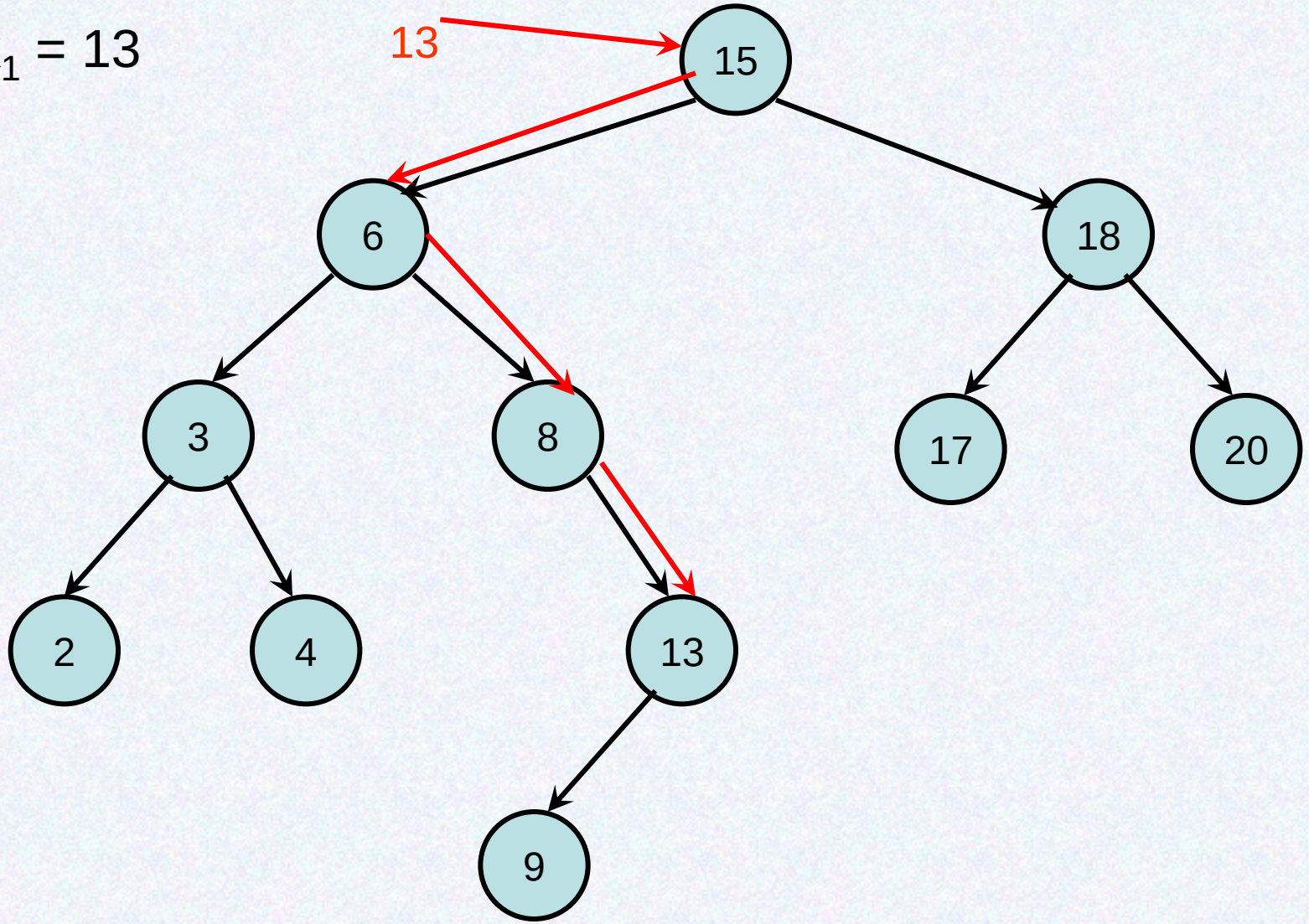
Алгоритм поиска по ключу

$k_1 = 13$



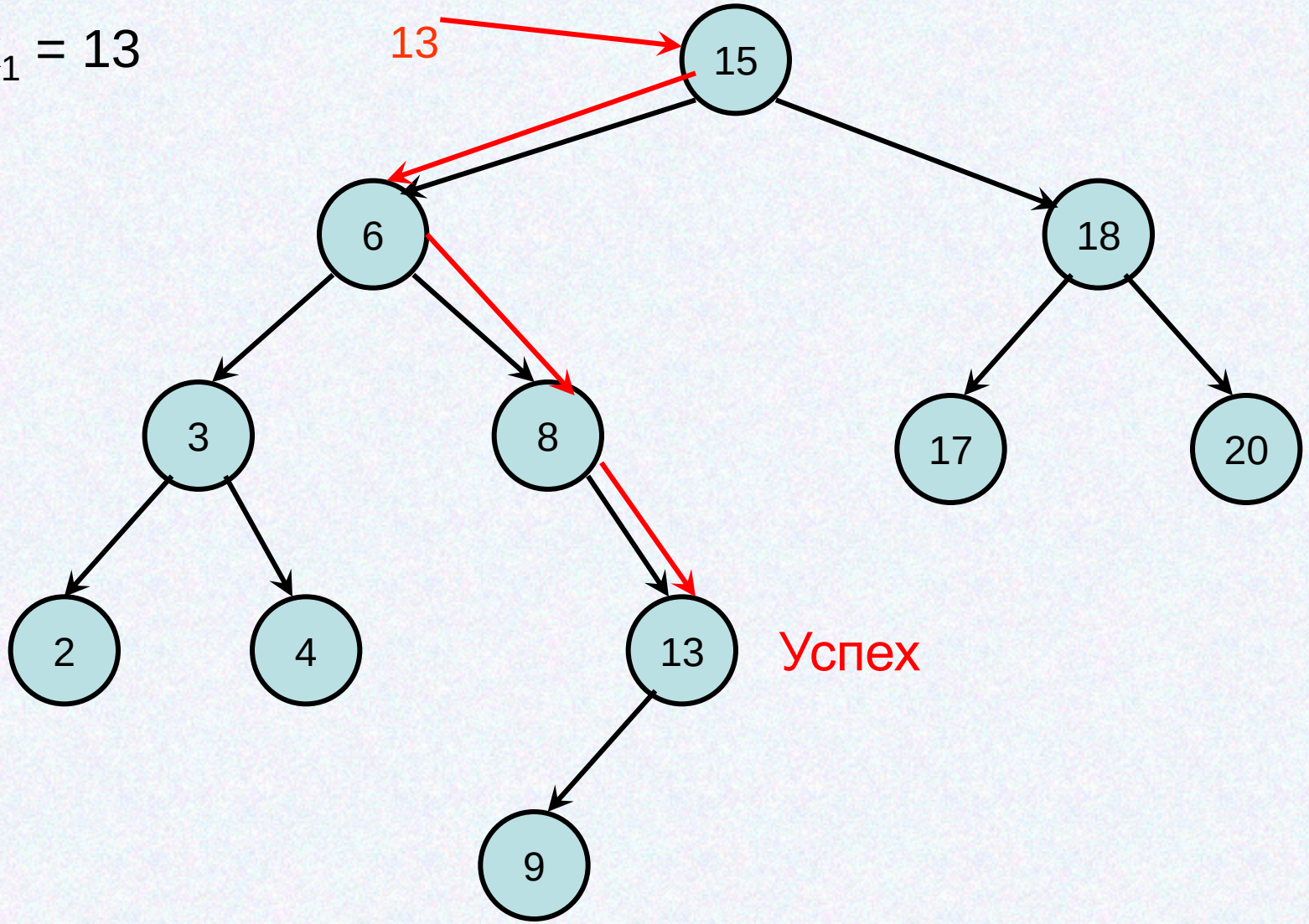
Алгоритм поиска по ключу

$k_1 = 13$



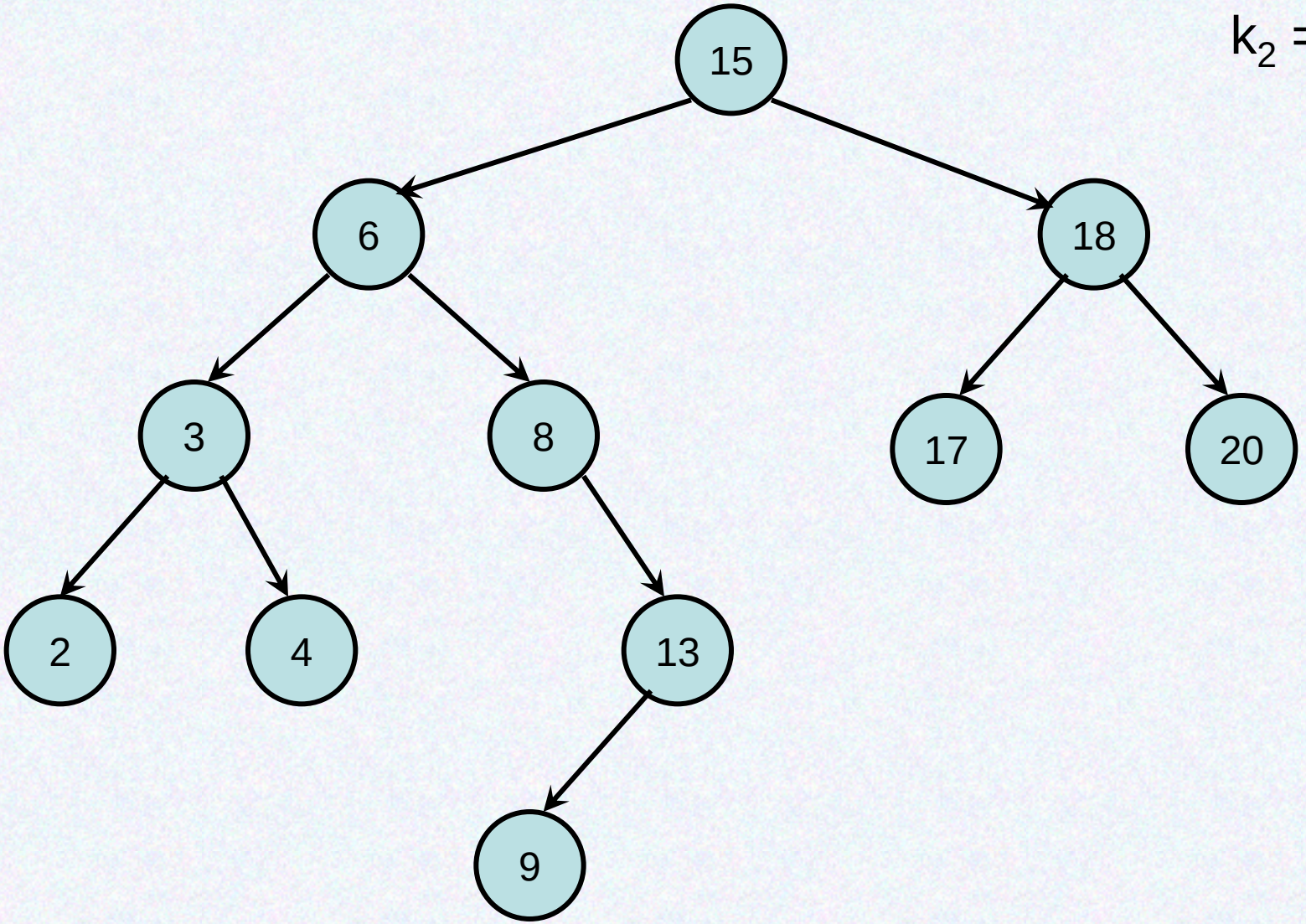
Алгоритм поиска по ключу

$k_1 = 13$

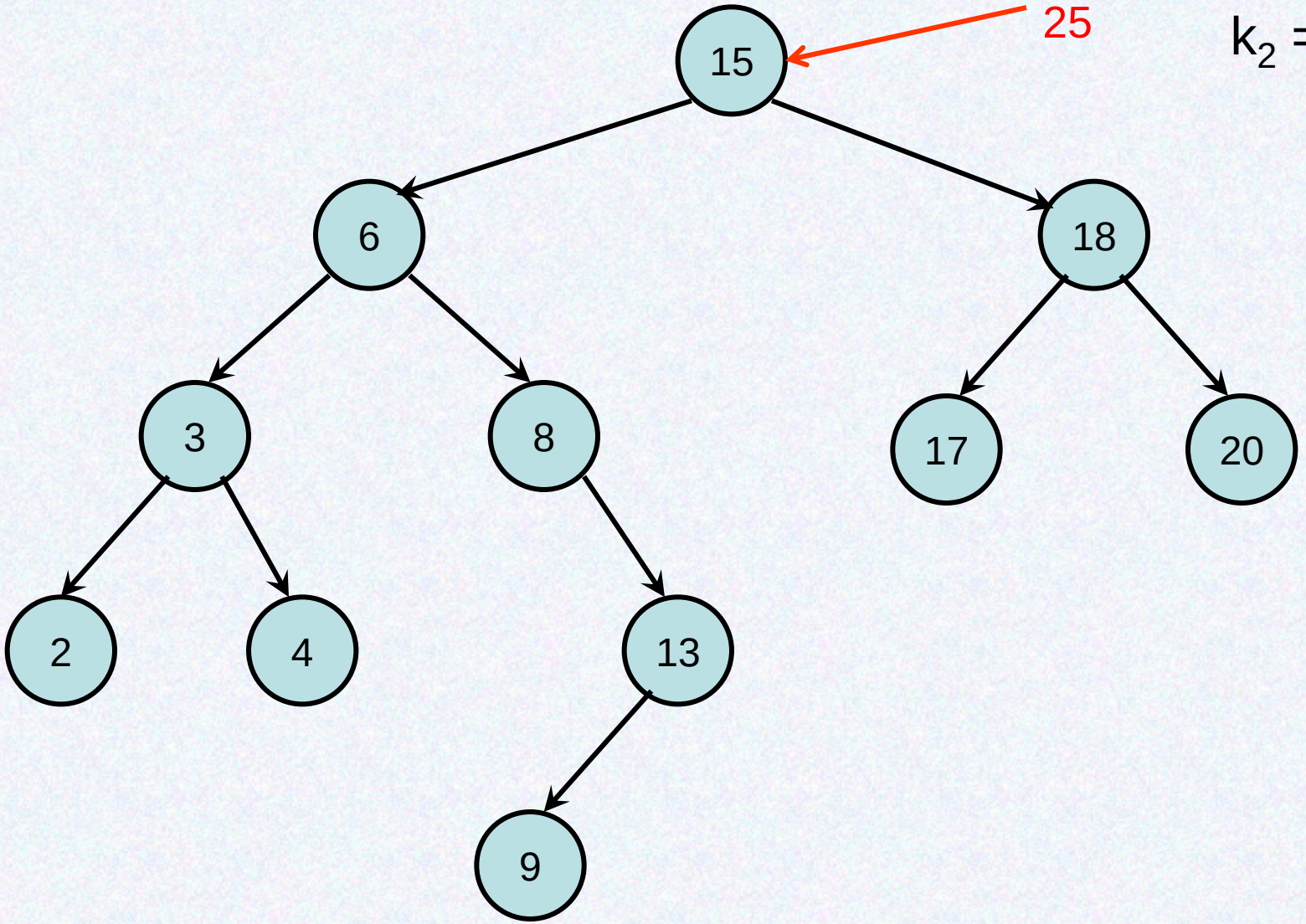


Алгоритм поиска по ключу

$k_2 = 25$

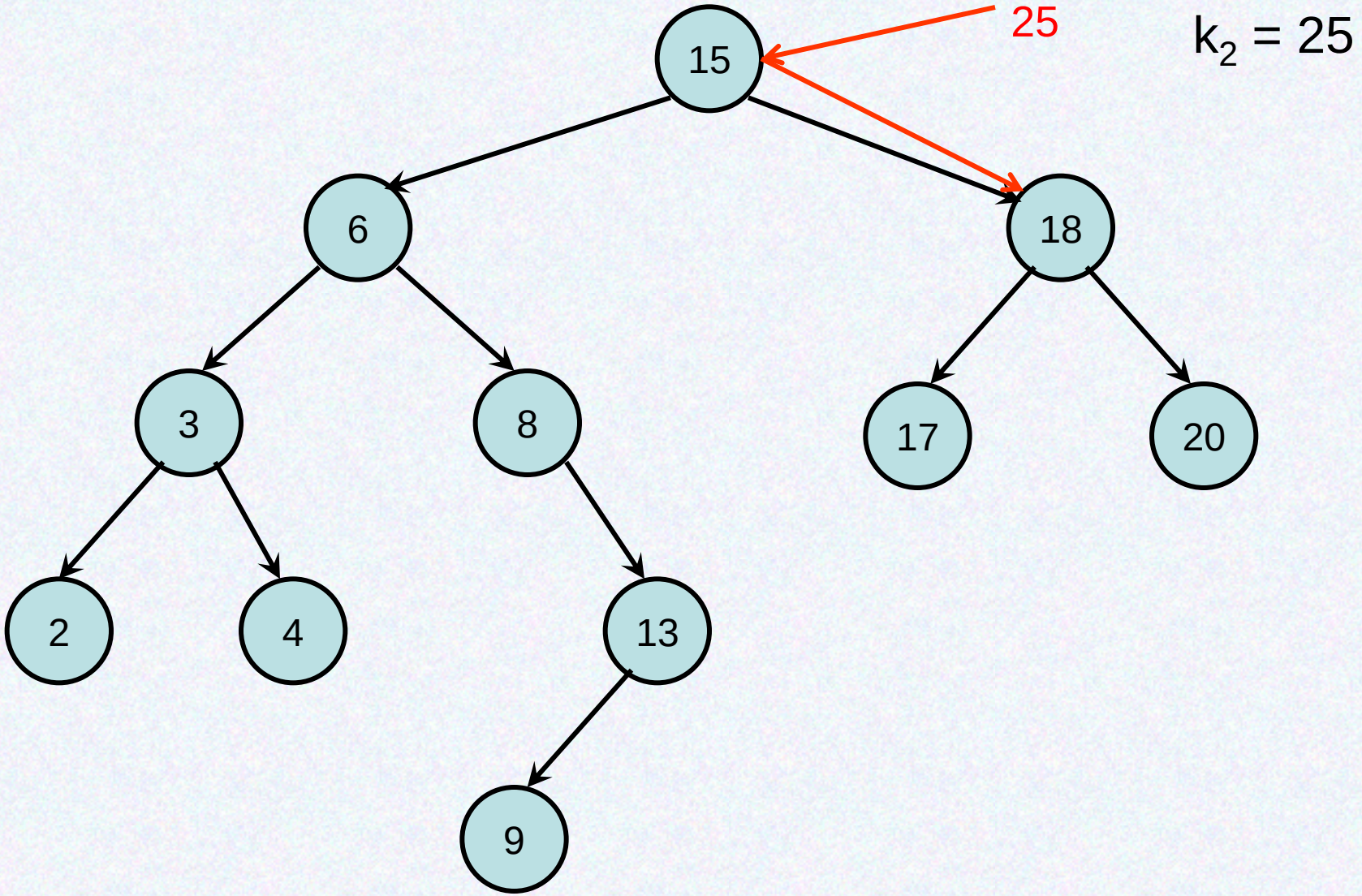


Алгоритм поиска по ключу

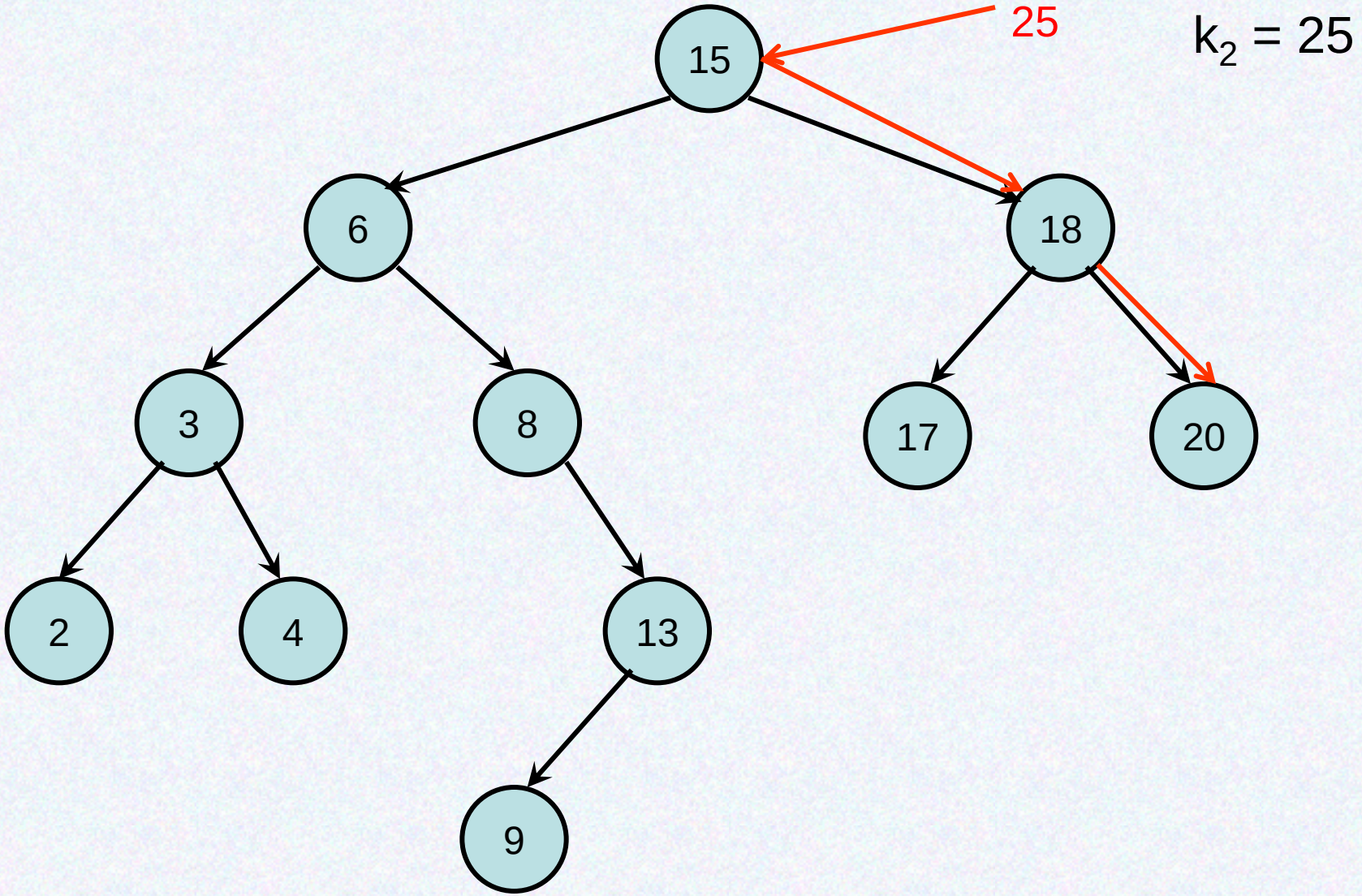


$k_2 = 25$

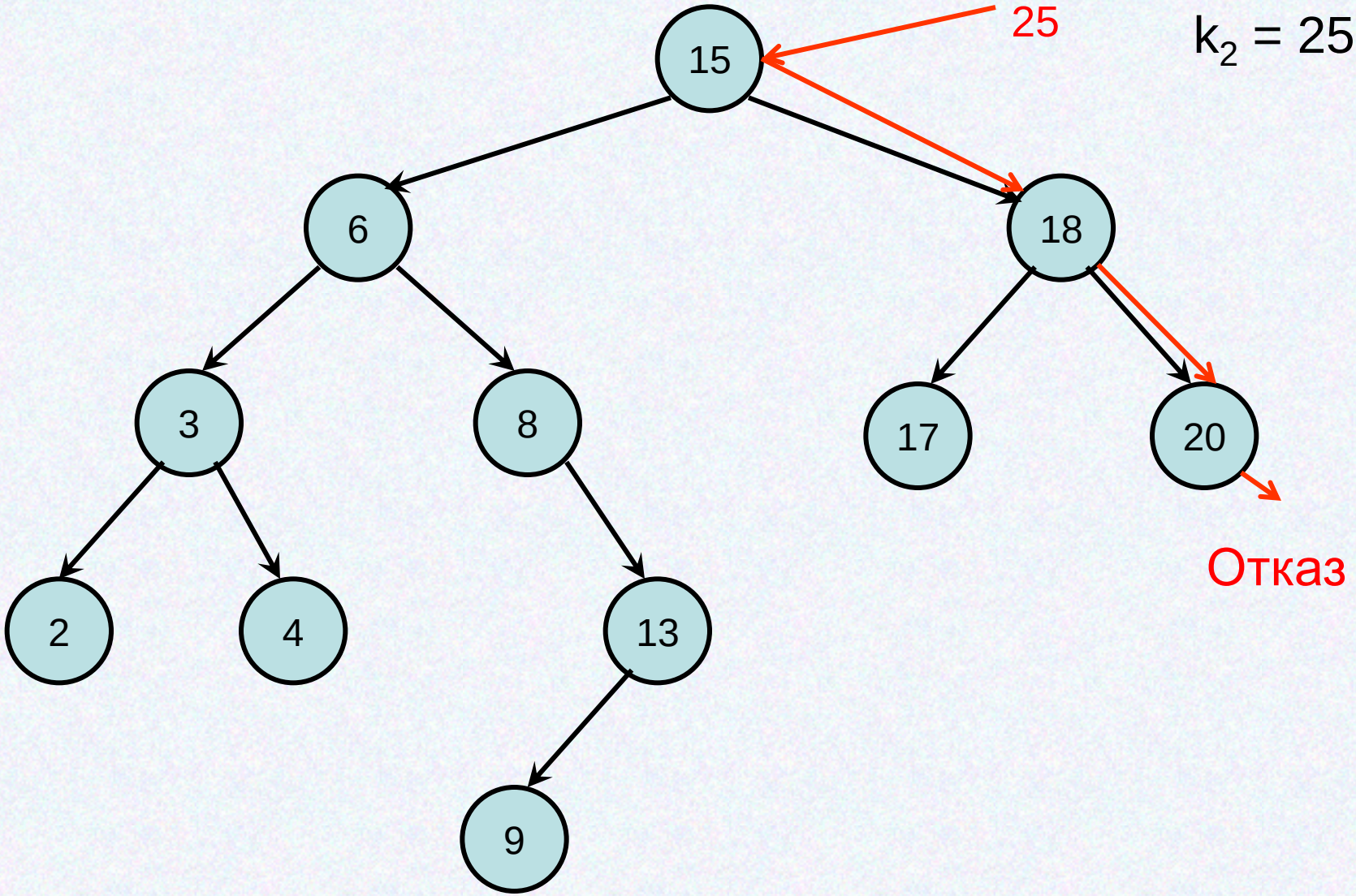
Алгоритм поиска по ключу



Алгоритм поиска по ключу



Алгоритм поиска по ключу



Поиск минимума и максимума

Поиск минимума и максимума

Поиск минимума – перемещение по *левому* поддереву

Поиск минимума и максимума

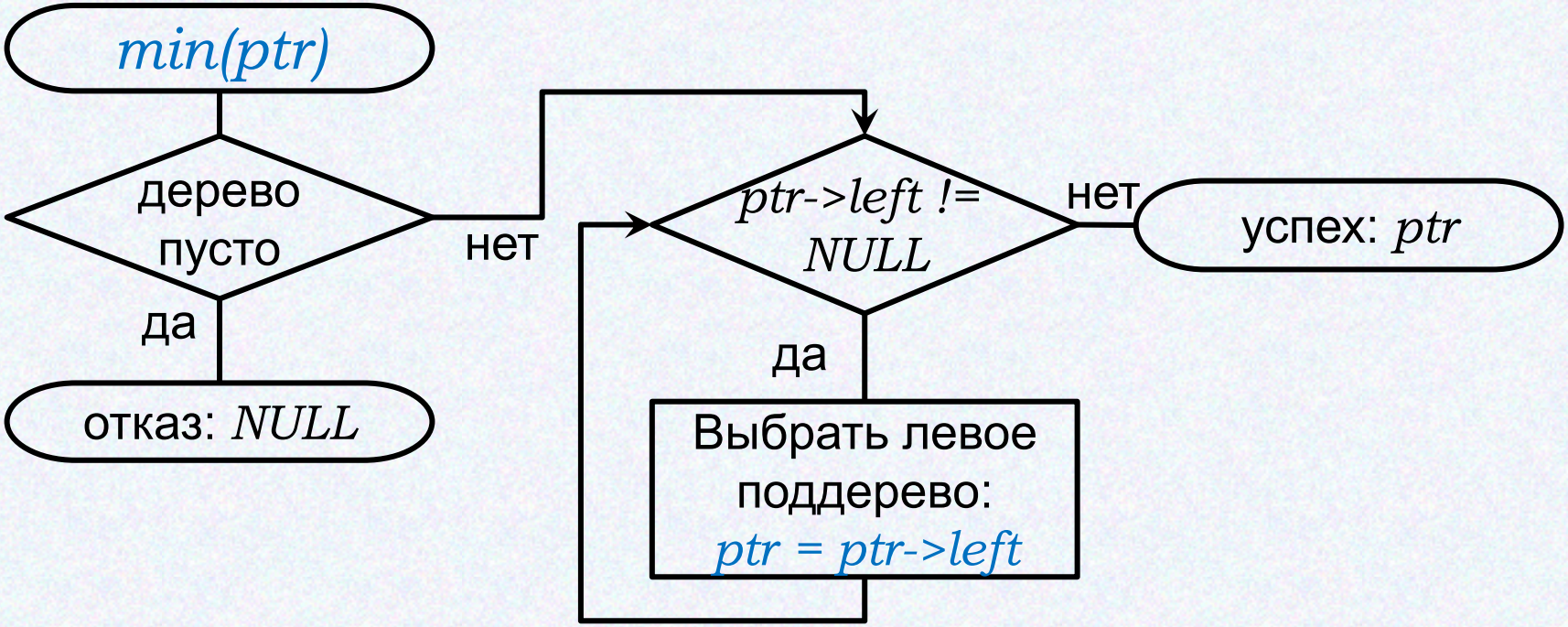
Поиск минимума – перемещение по *левому* поддереву

Поиск максимума – перемещение по *правому* поддереву

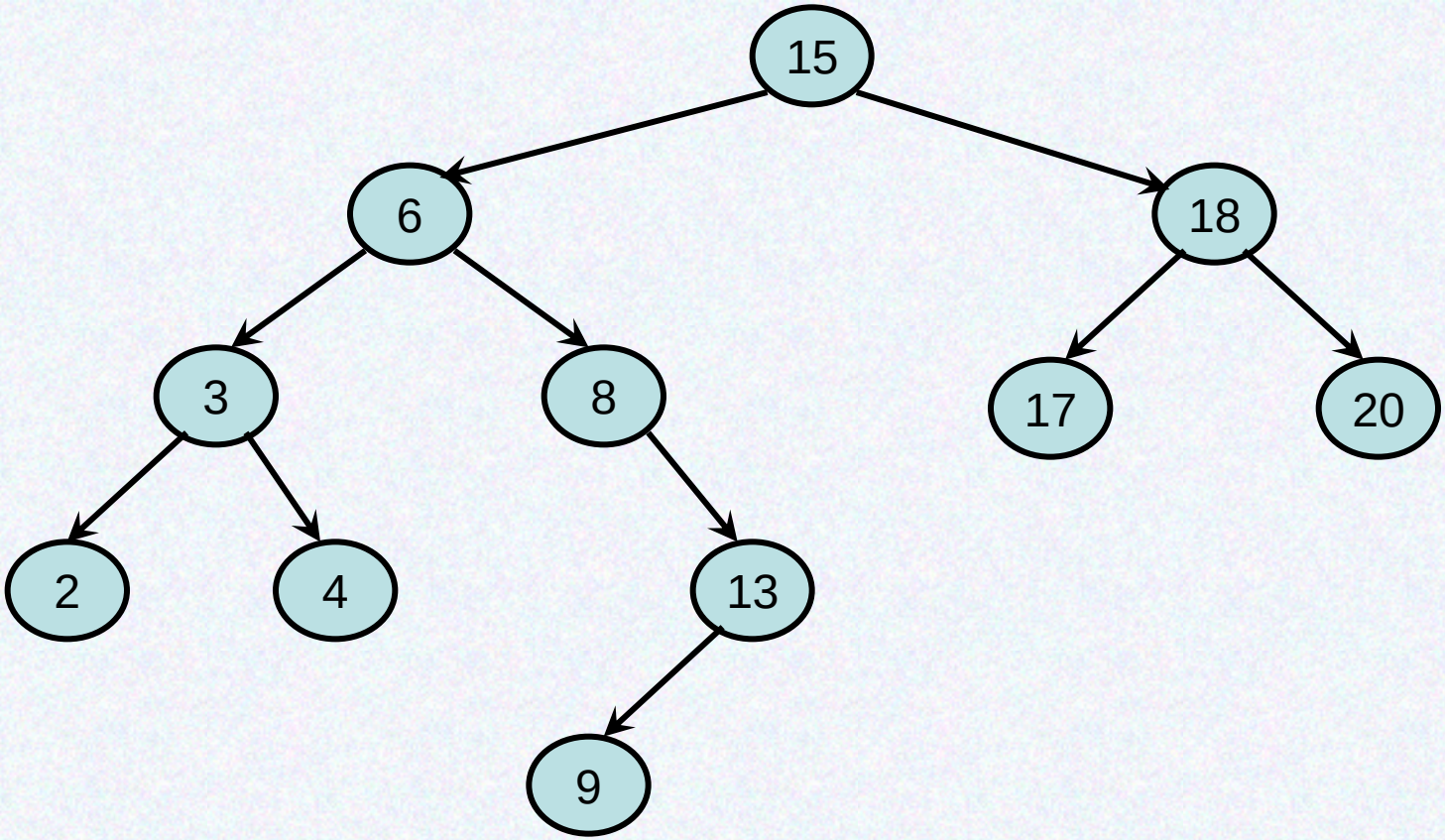
Поиск минимума и максимума

Поиск минимума – перемещение по *левому* поддереву

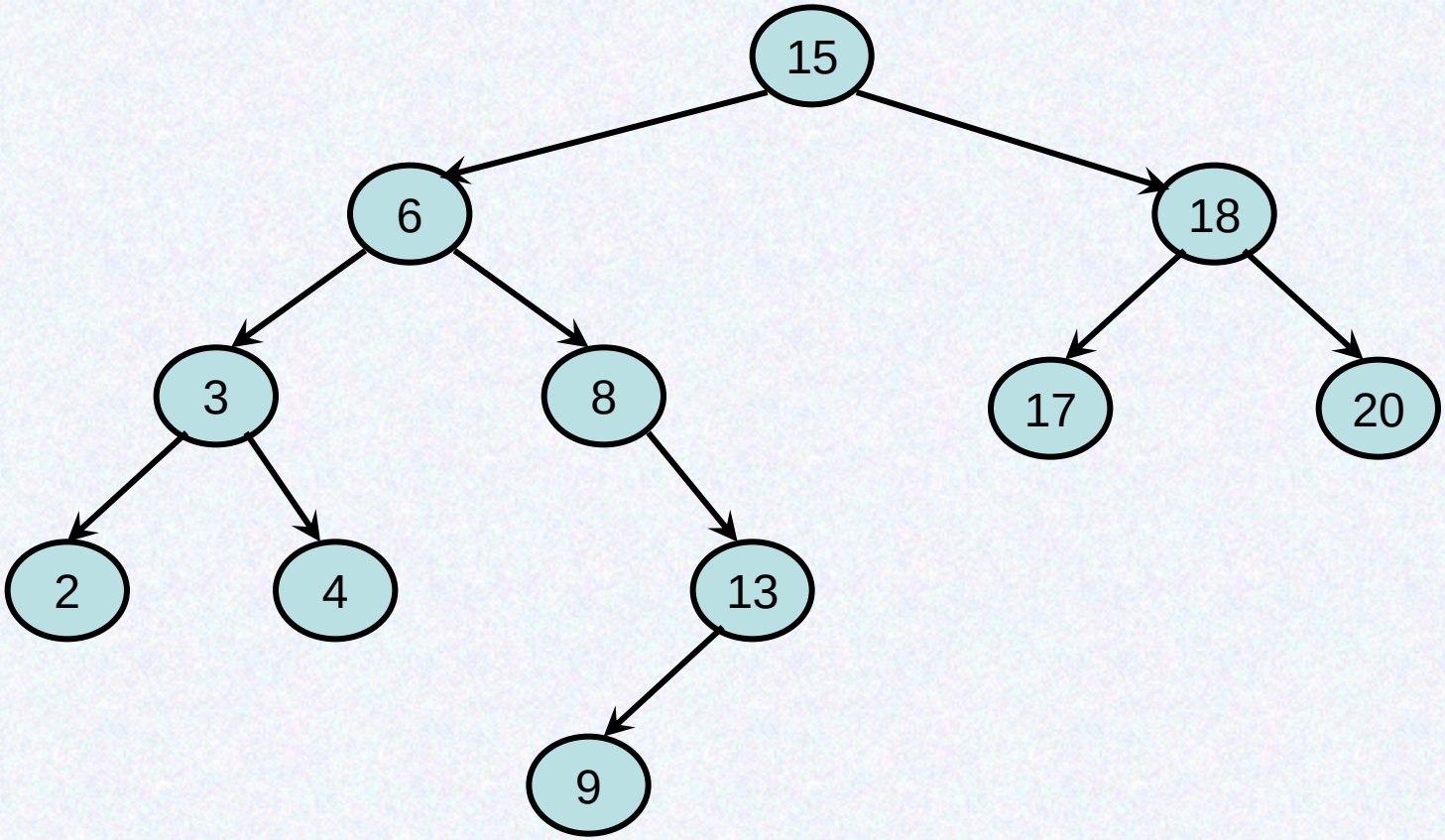
Поиск максимума – перемещение по *правому* поддереву



Предшествующий и последующий элементы

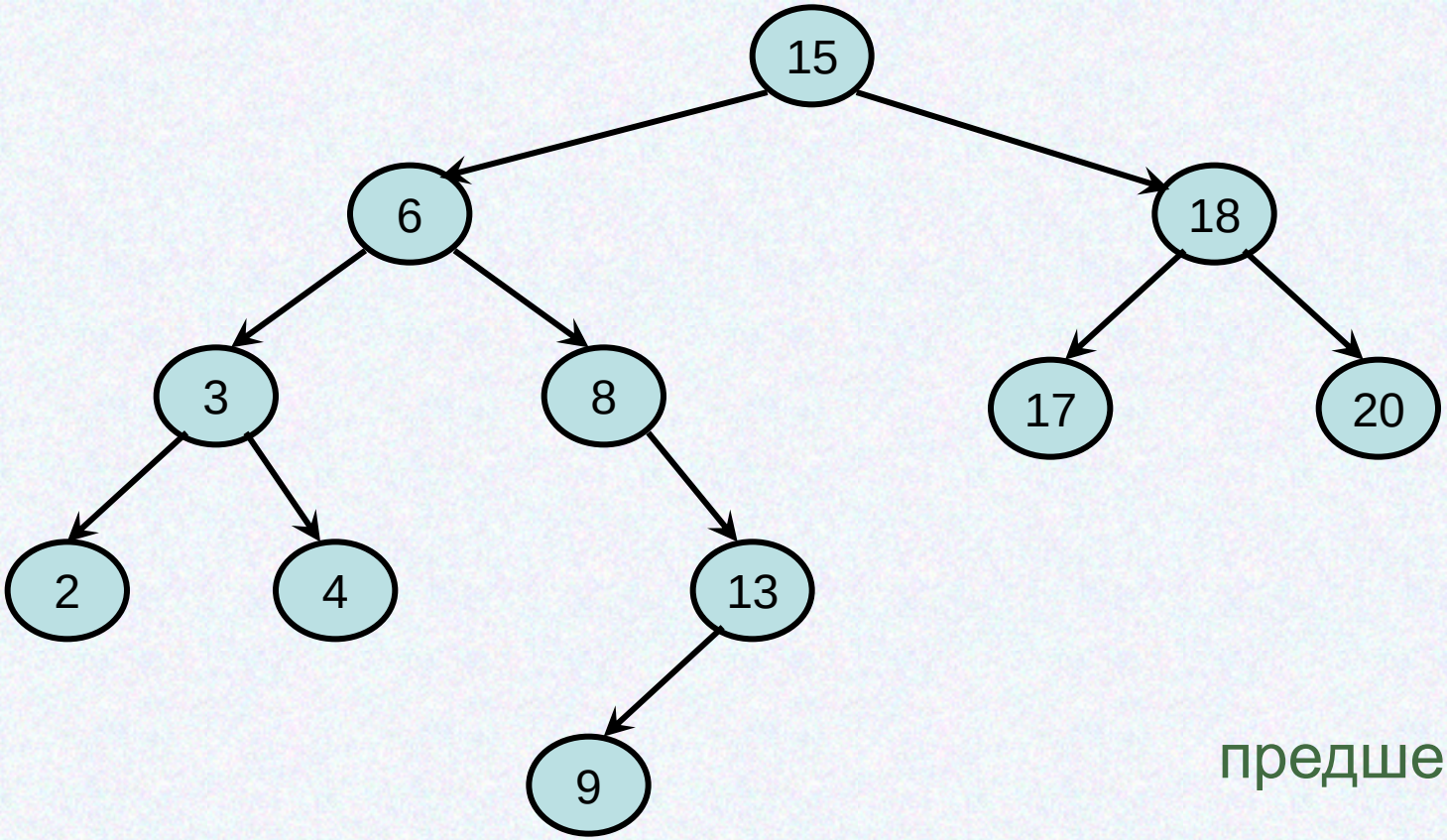


Предшествующий и последующий элементы



Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

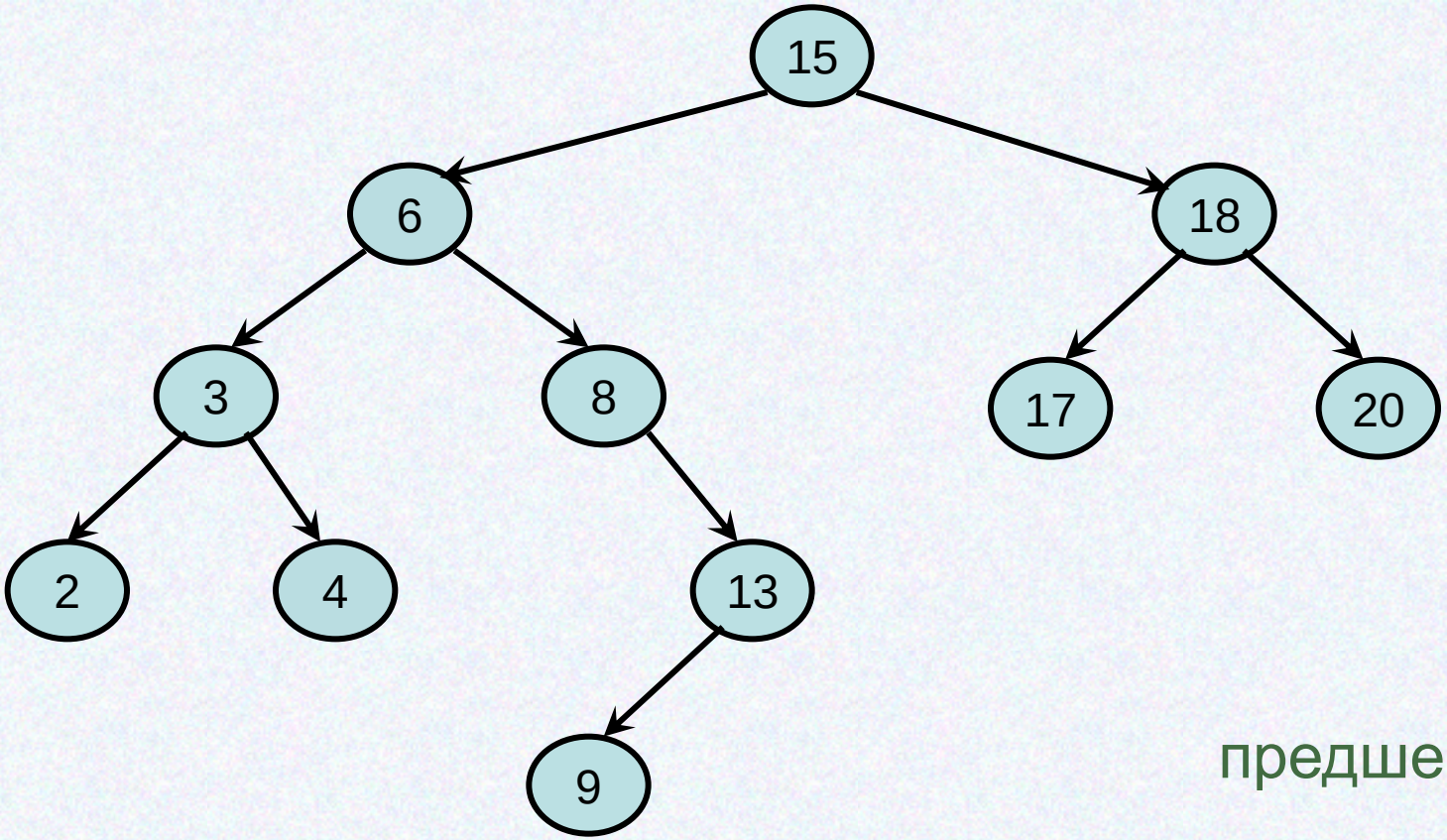
Предшествующий и последующий элементы



предшествующий

Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

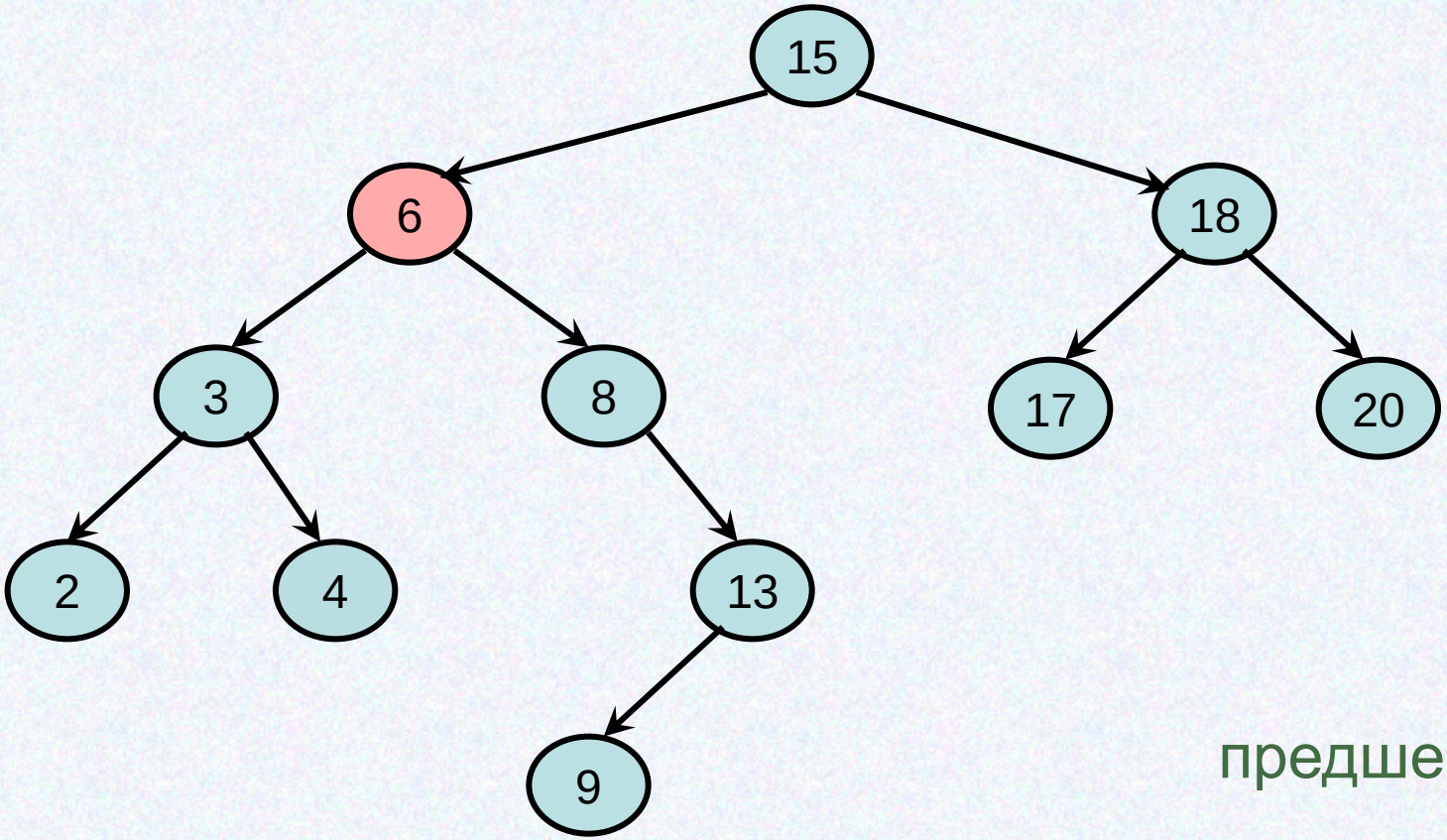
Предшествующий и последующий элементы



предшествующий

Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

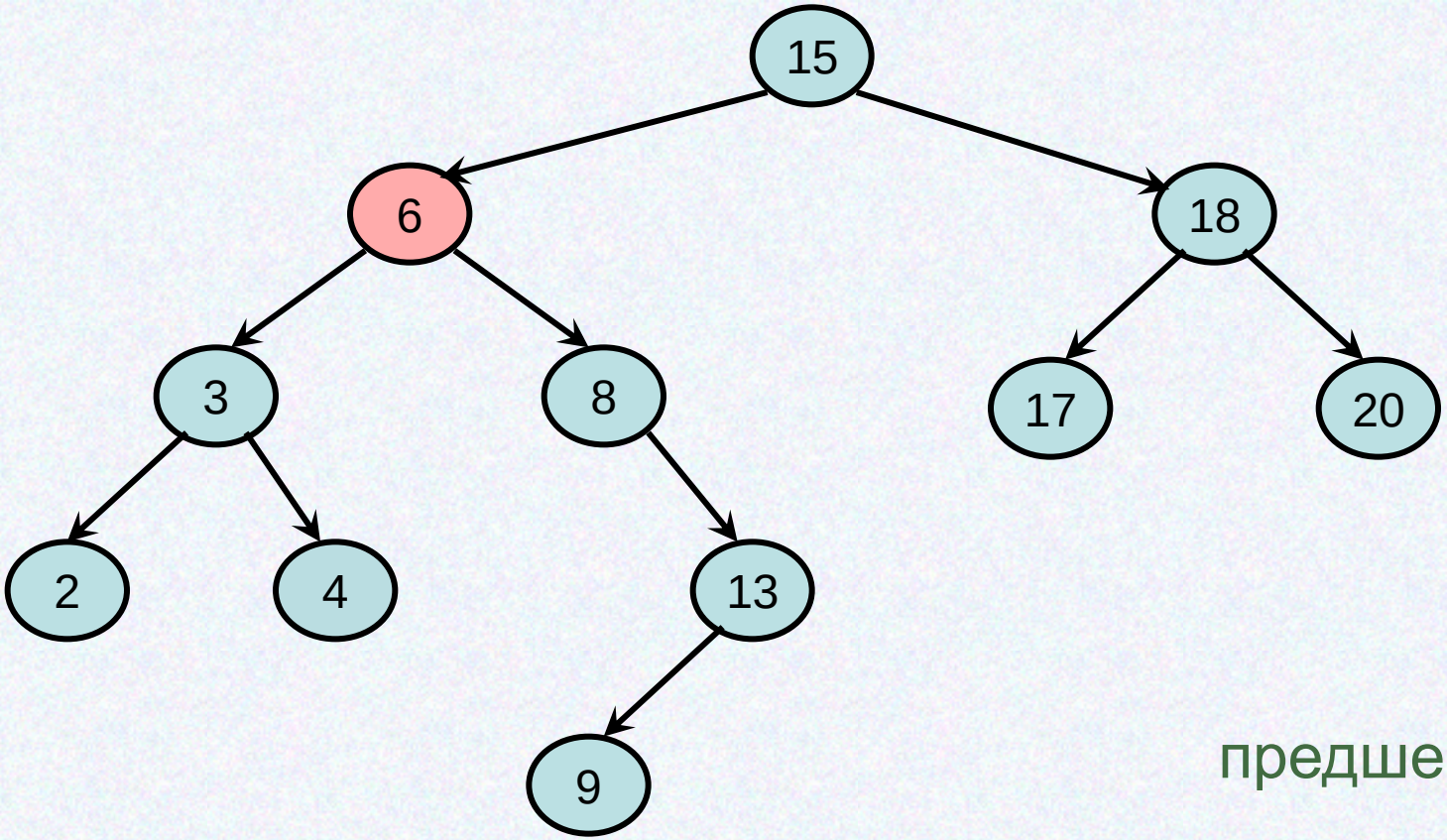
Предшествующий и последующий элементы



предшествующий

Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

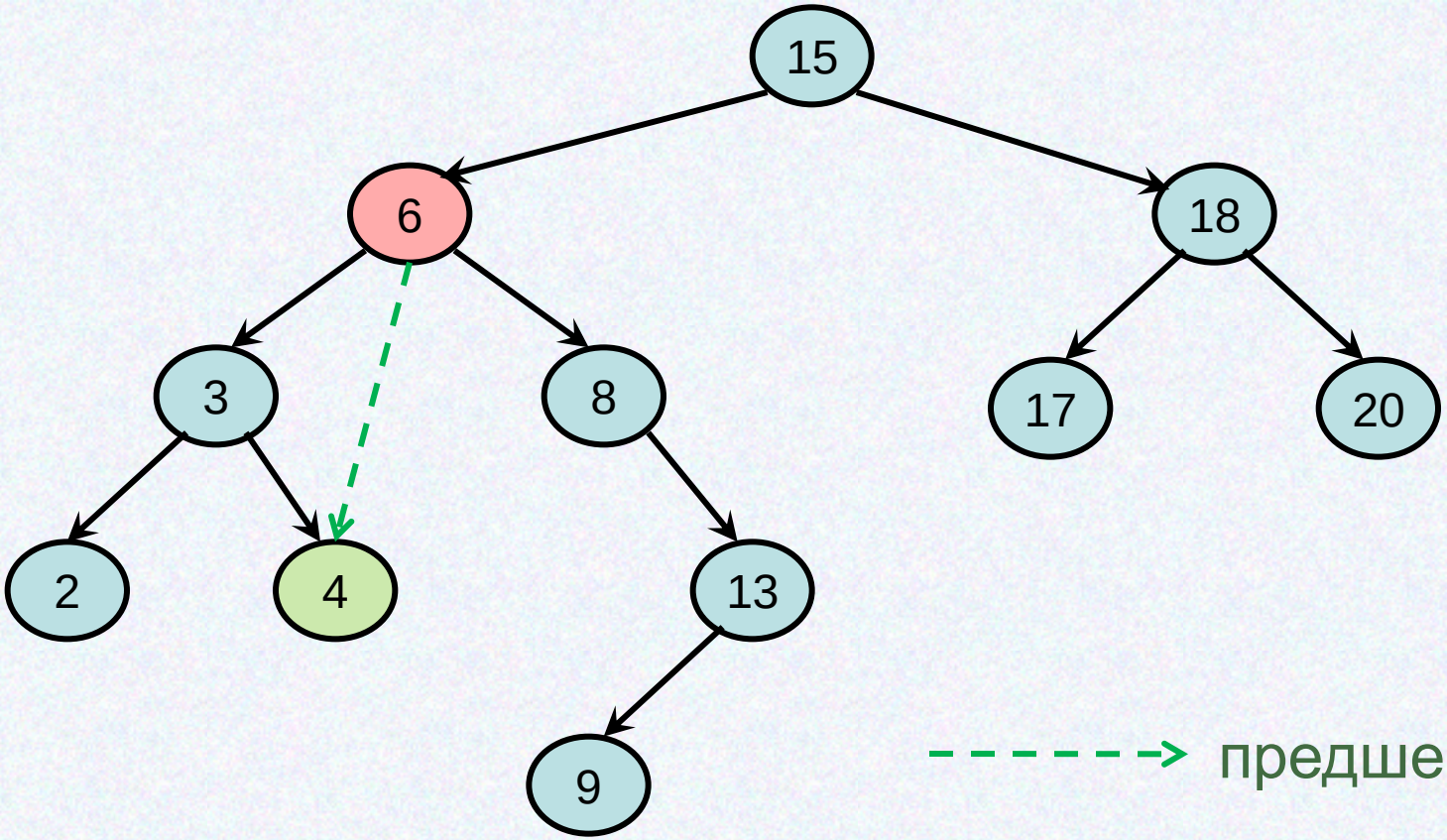
Предшествующий и последующий элементы



предшествующий

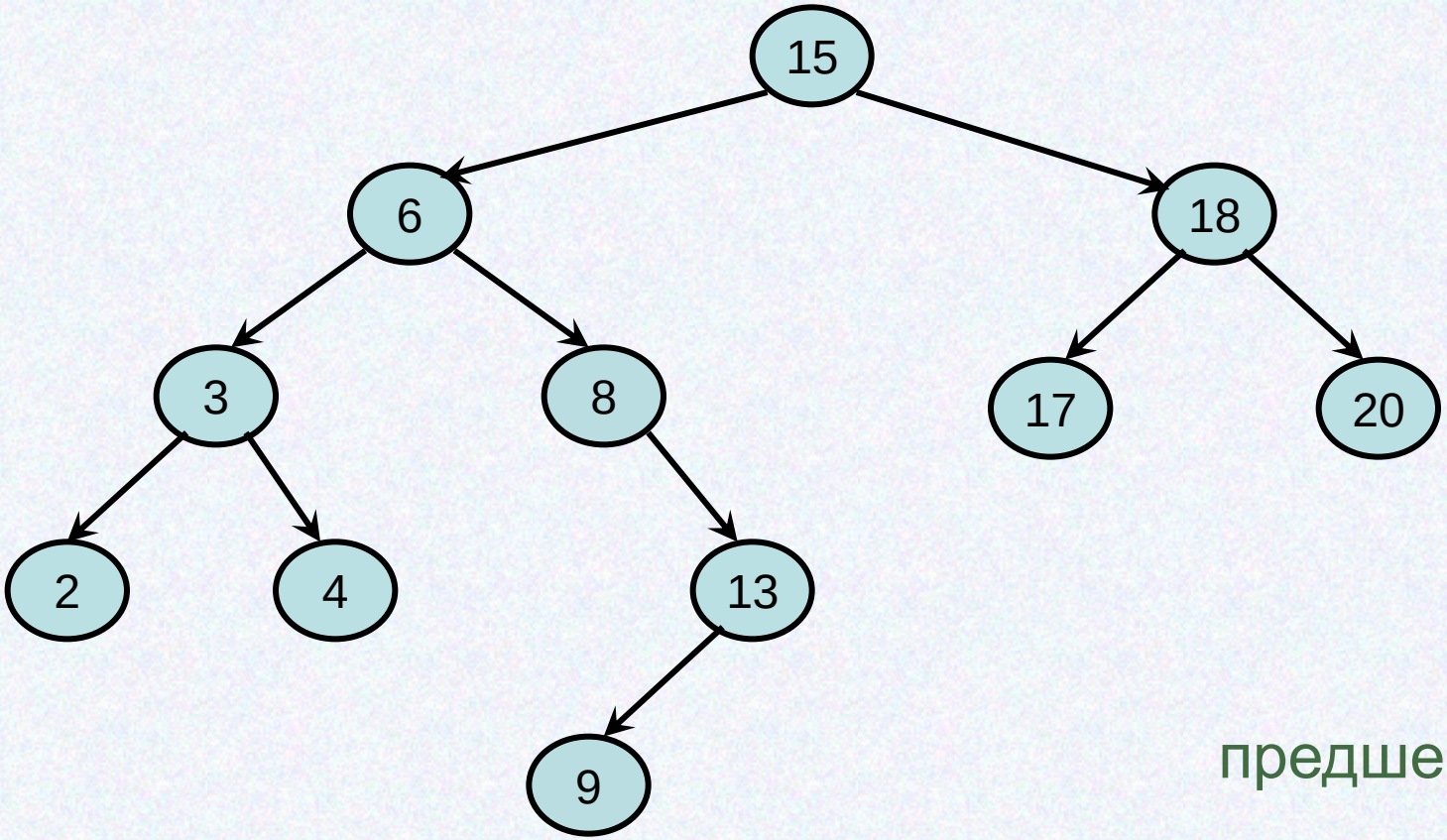
Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

Предшествующий и последующий элементы



Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

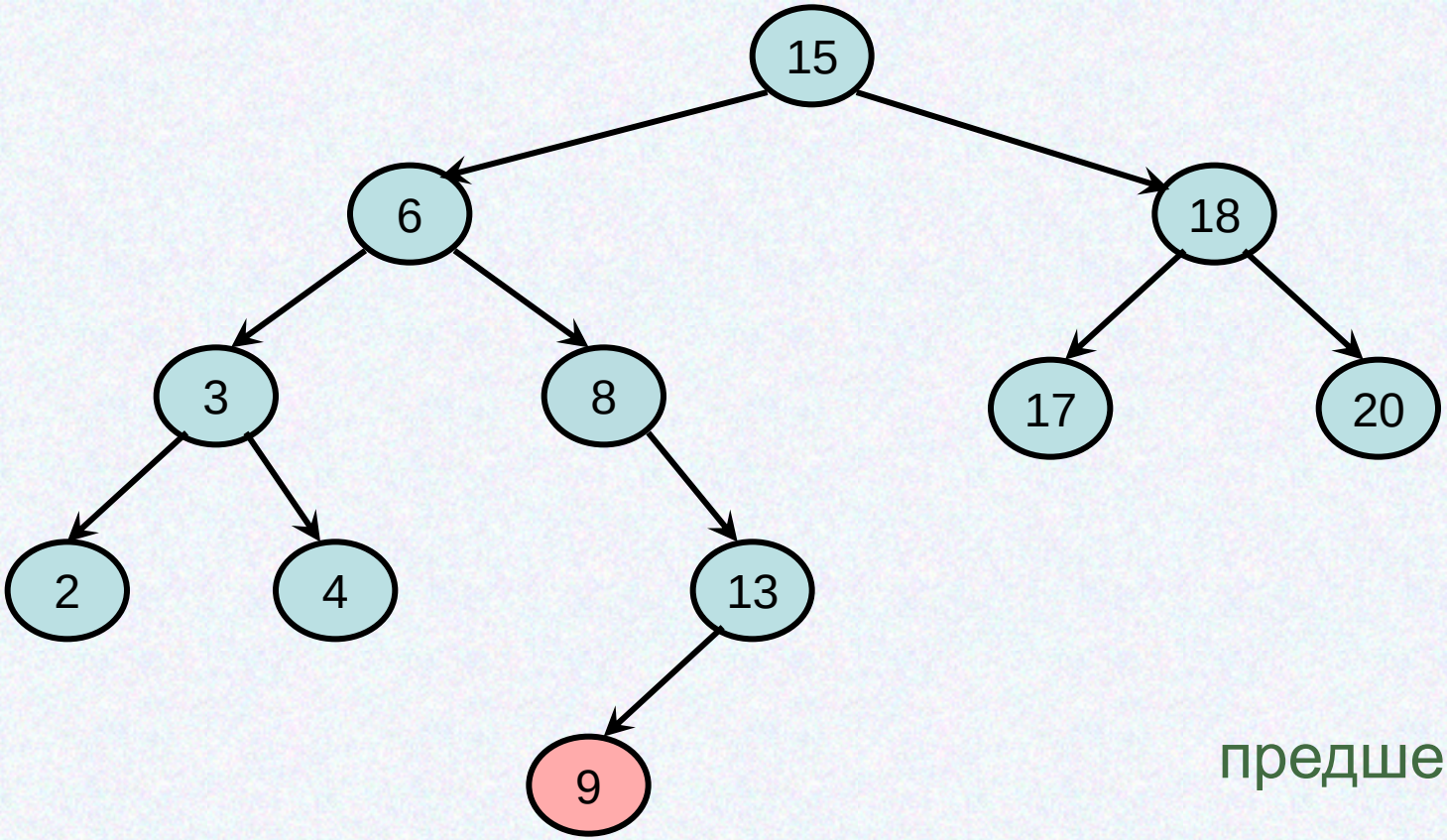
Предшествующий и последующий элементы



предшествующий

Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

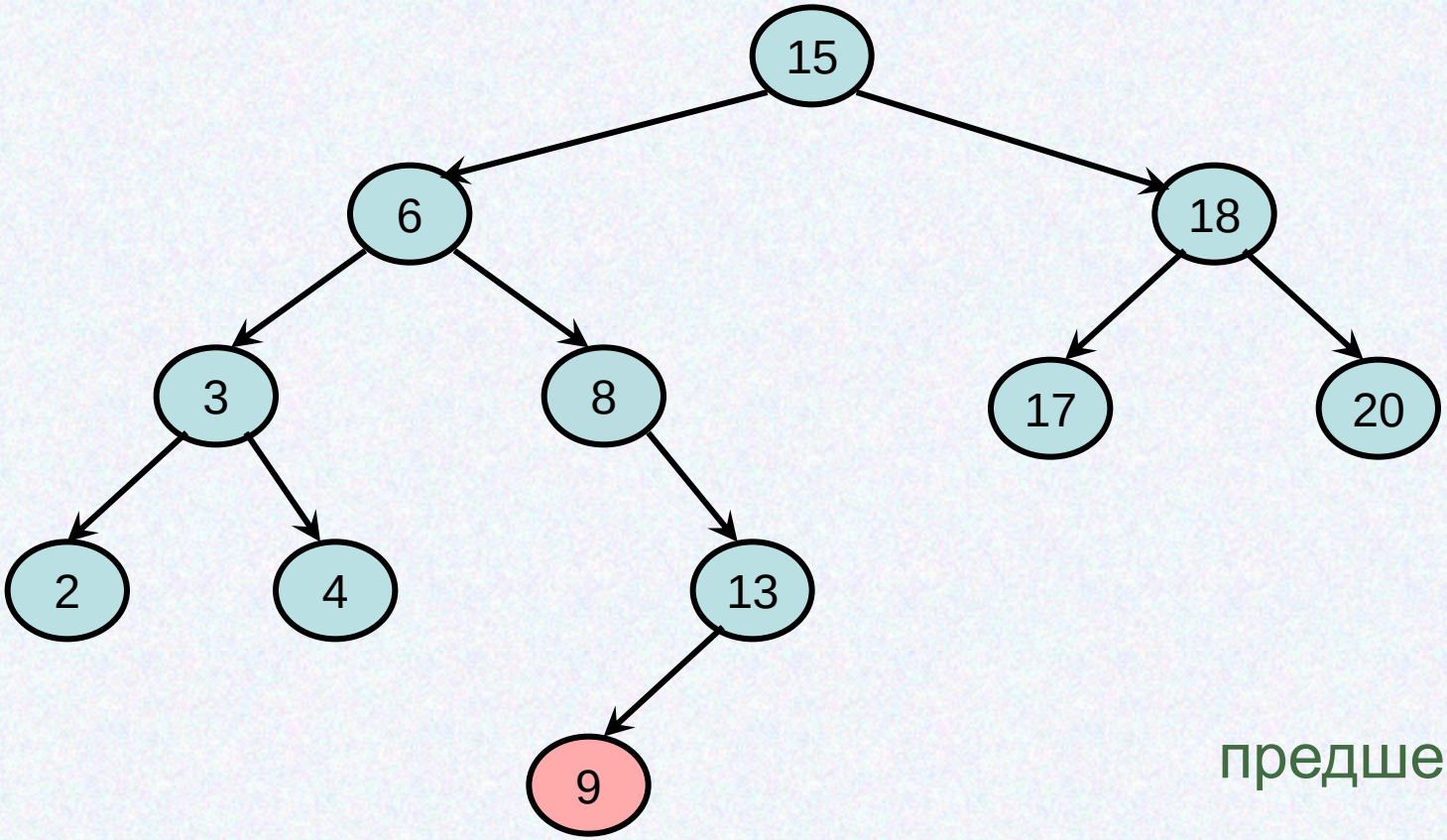
Предшествующий и последующий элементы



предшествующий

Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

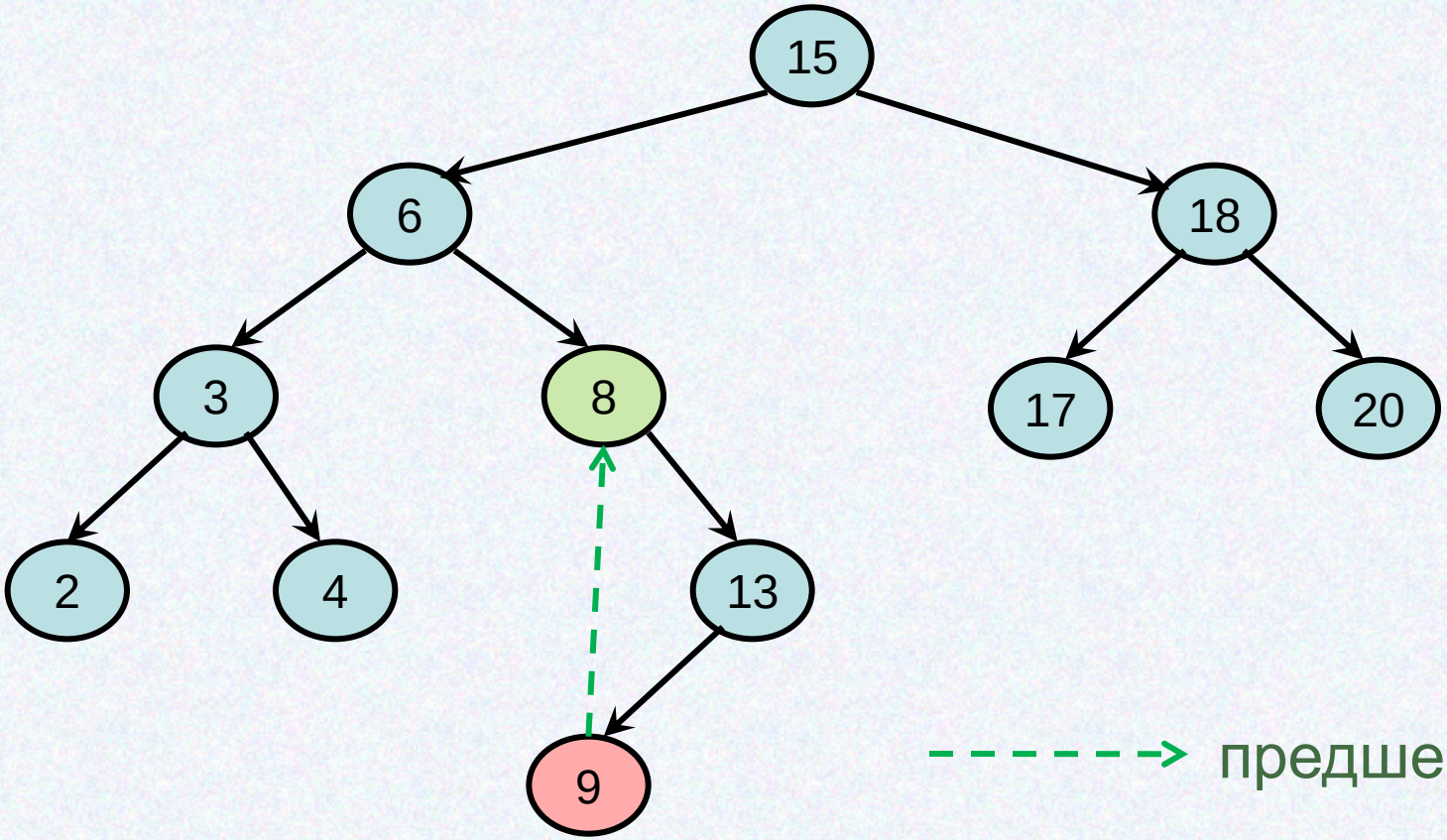
Предшествующий и последующий элементы



предшествующий

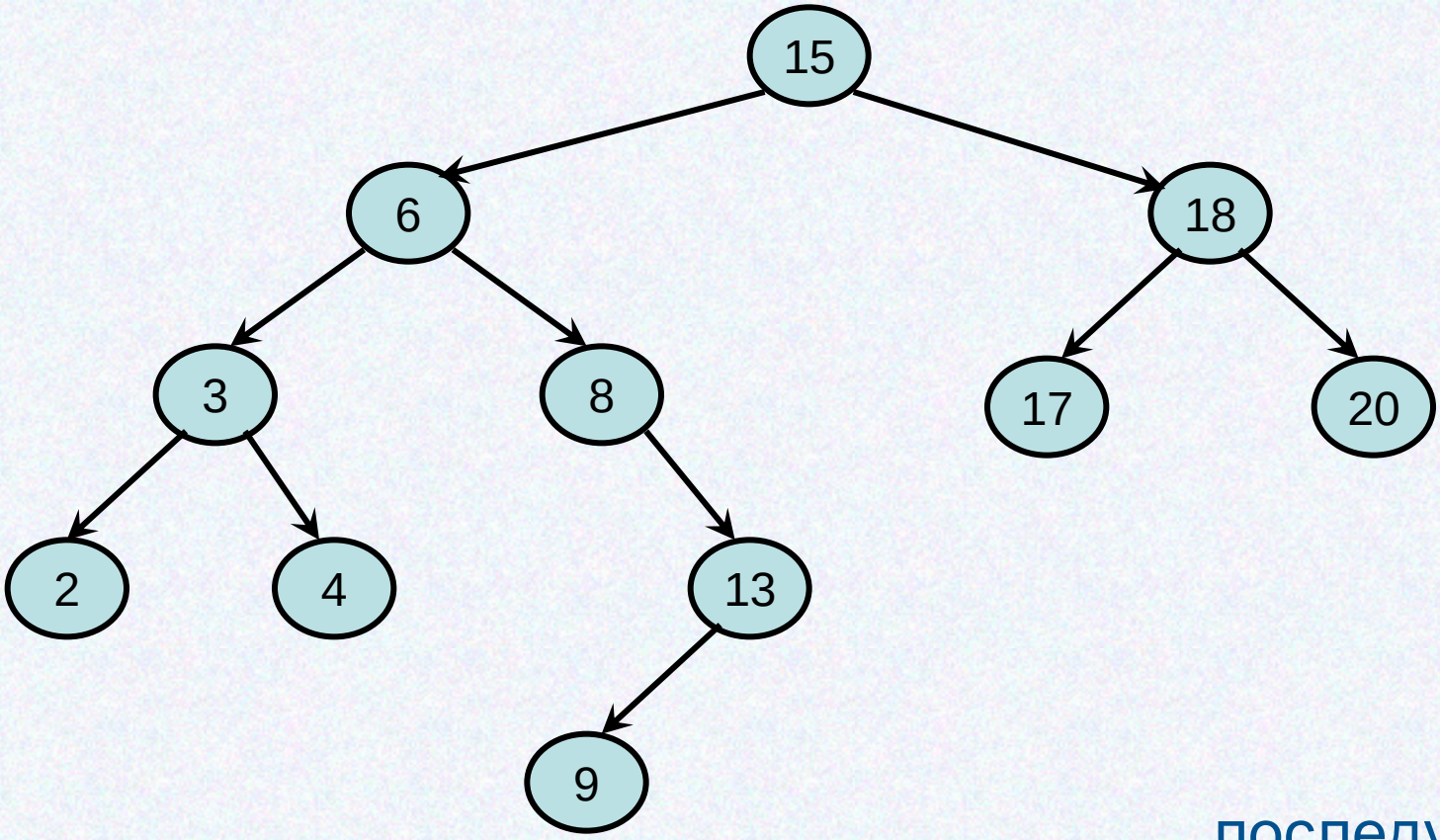
Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

Предшествующий и последующий элементы



Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

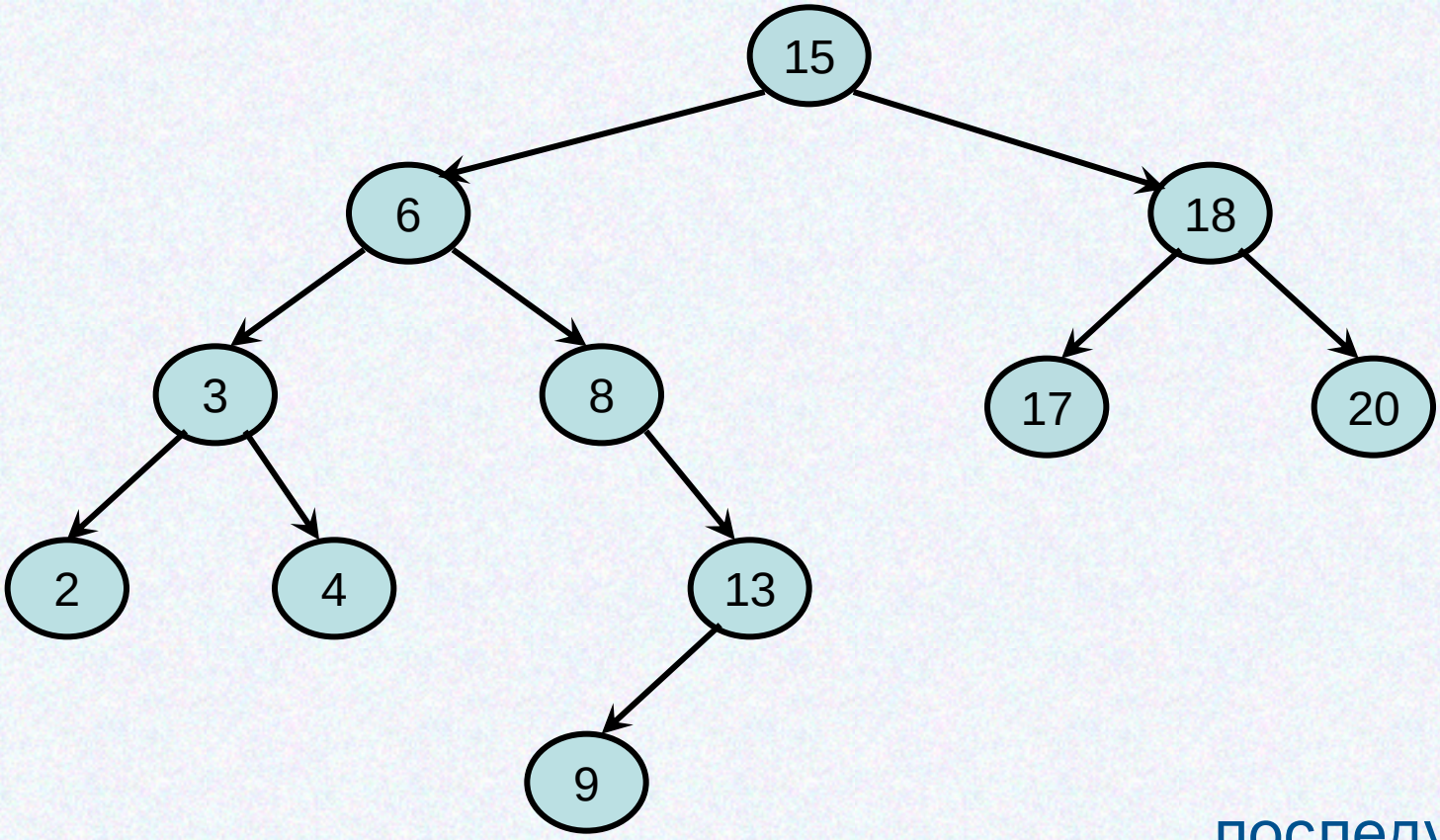
Предшествующий и последующий элементы



последующий

Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

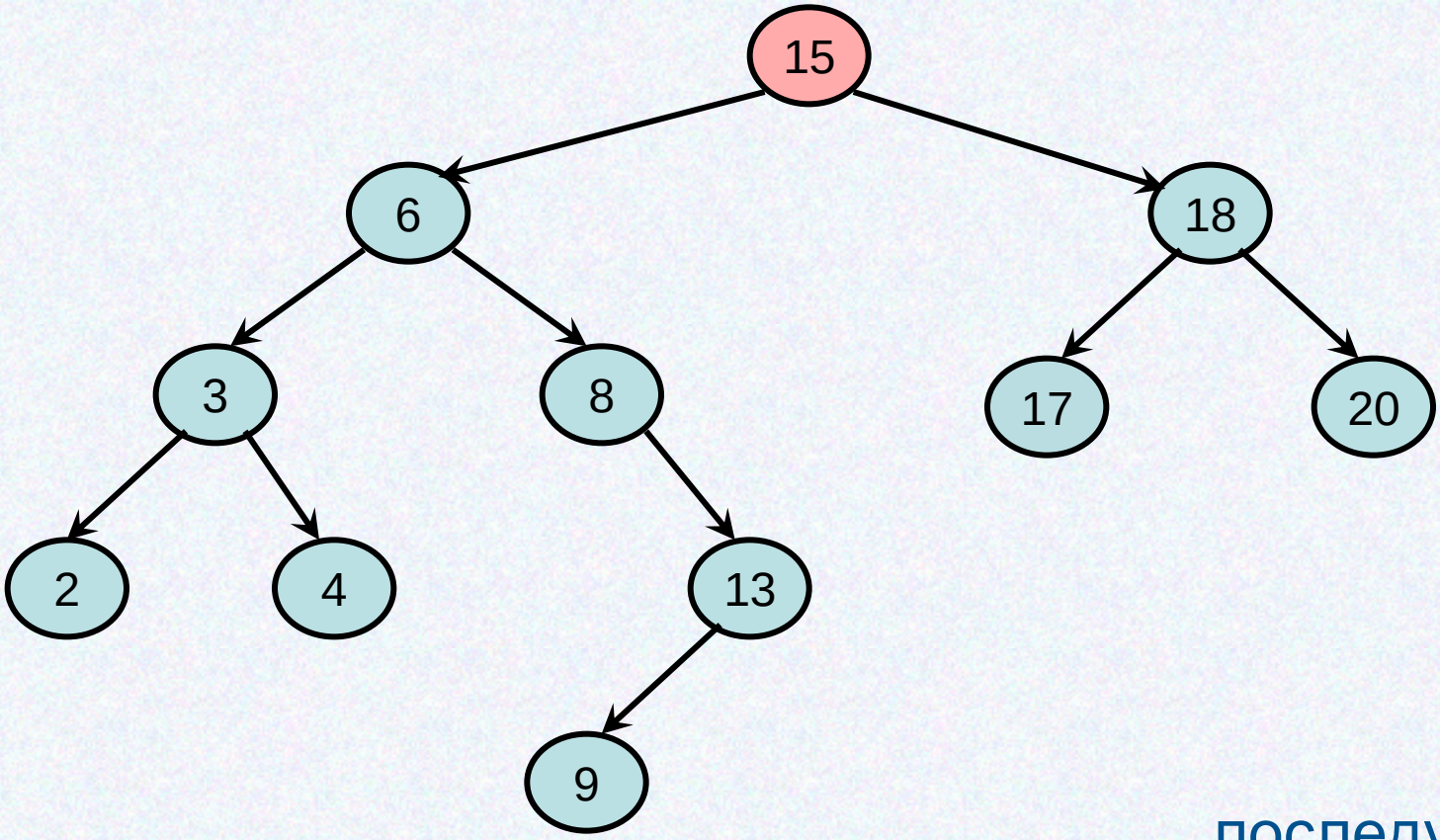
Предшествующий и последующий элементы



последующий

Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

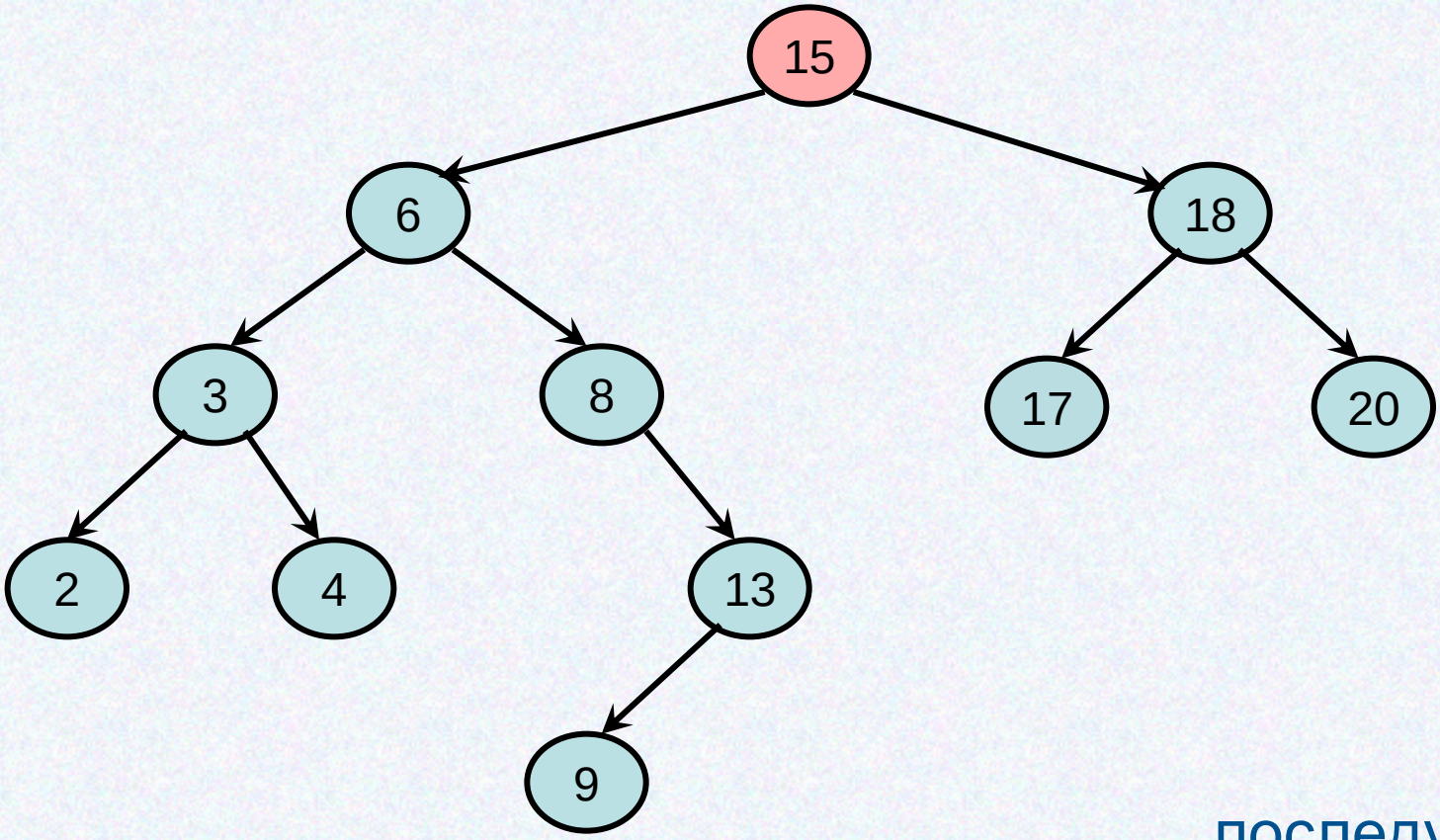
Предшествующий и последующий элементы



последующий

Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

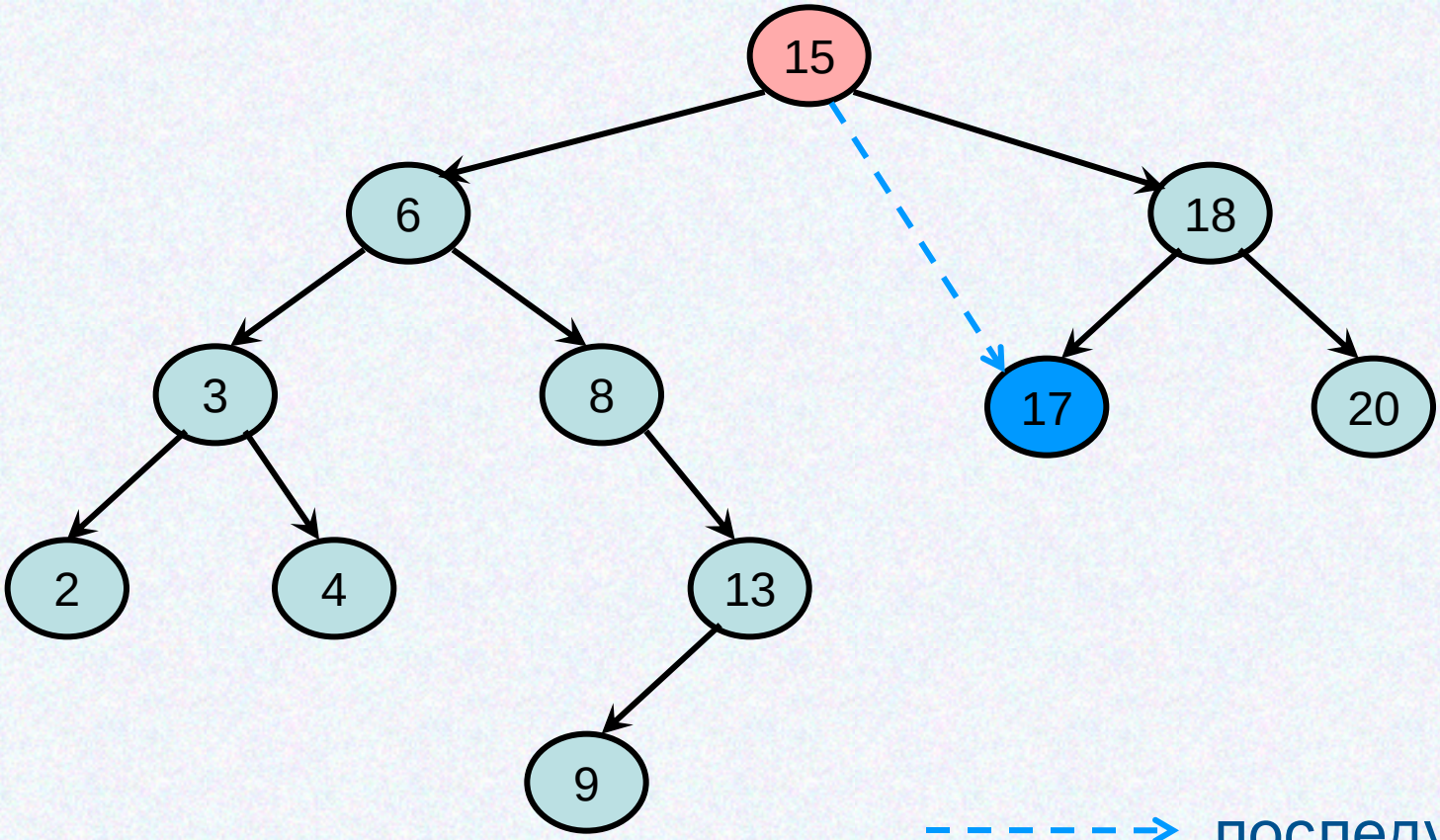
Предшествующий и последующий элементы



последующий

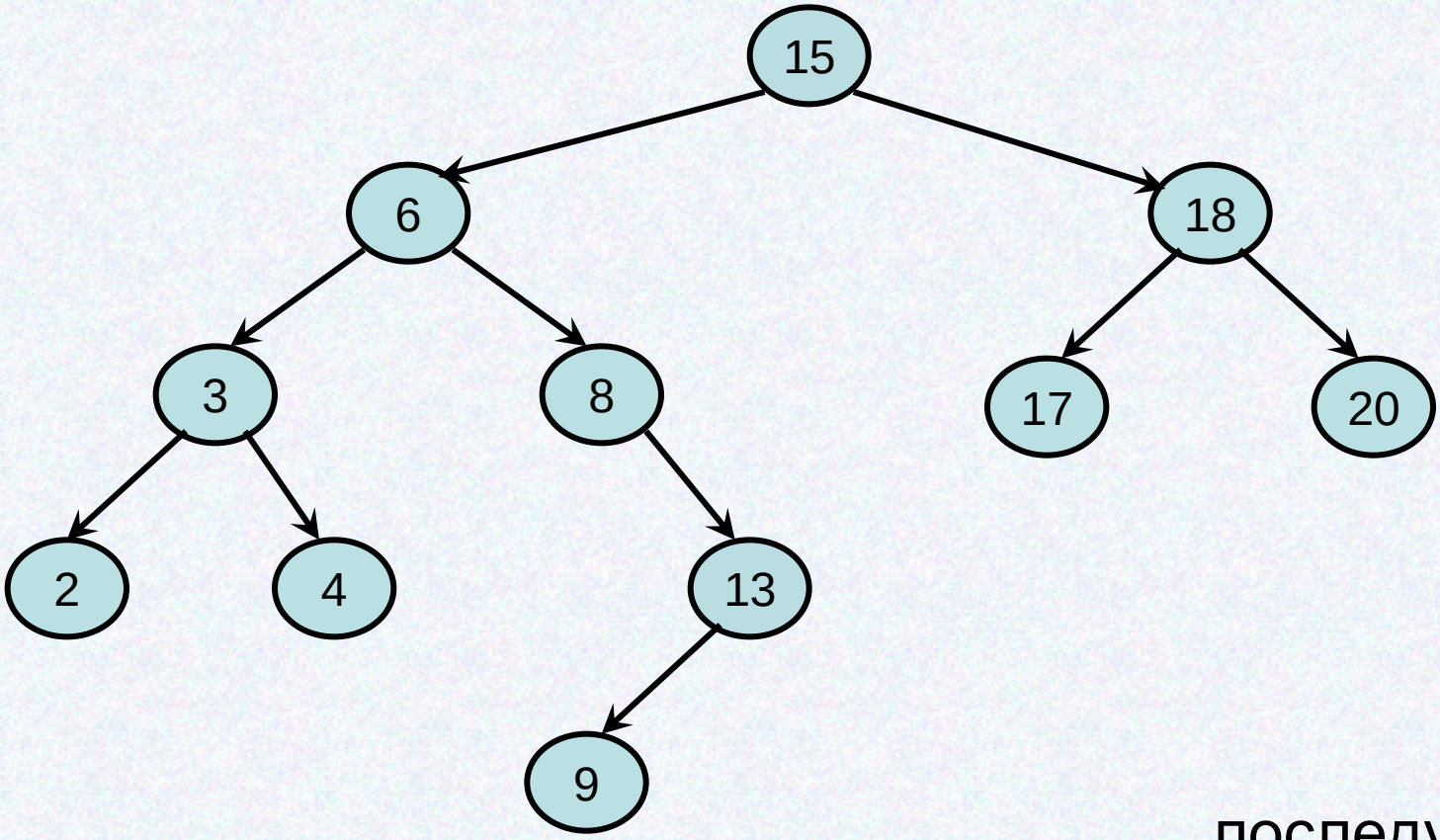
Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

Предшествующий и последующий элементы



Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

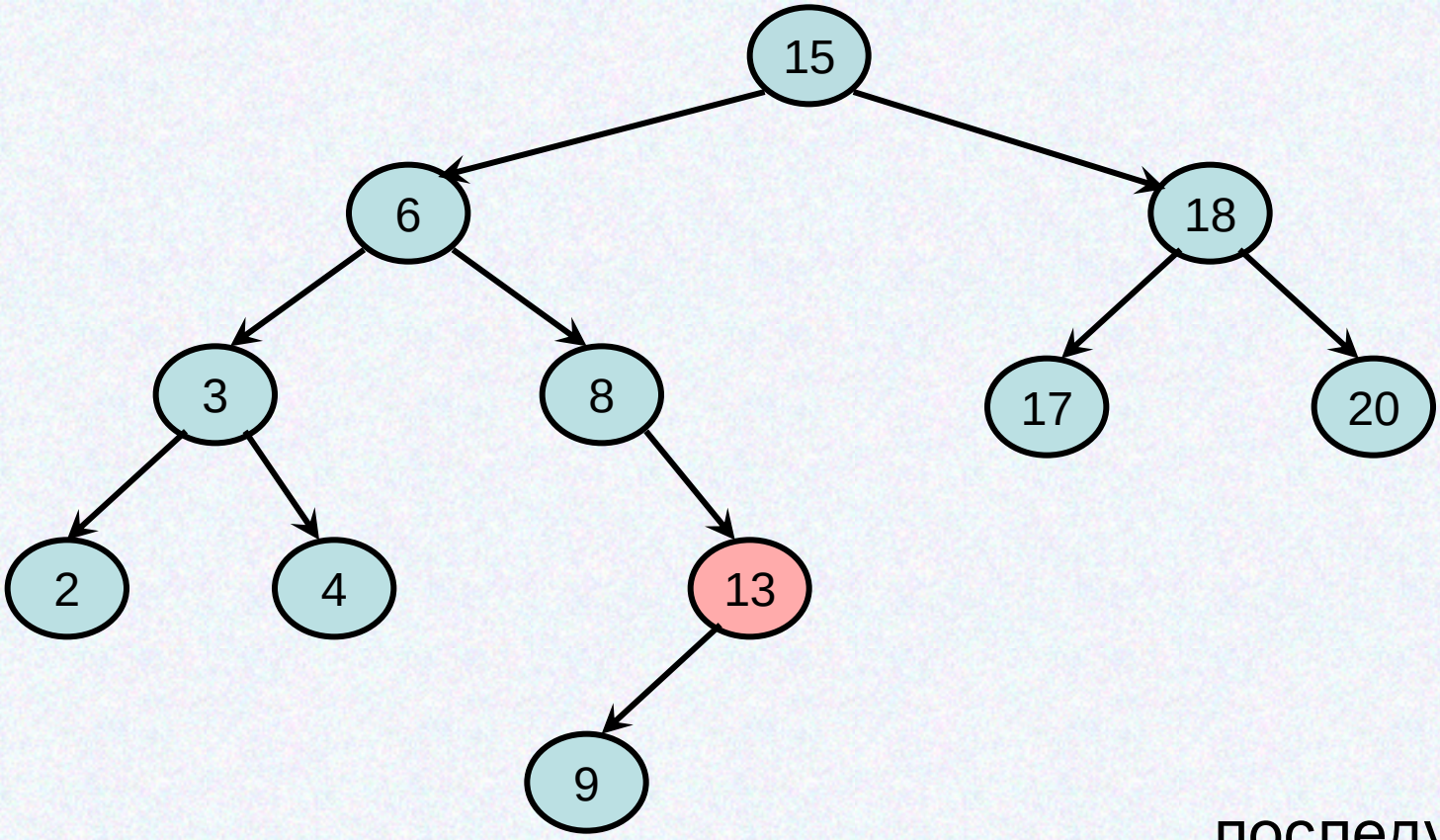
Предшествующий и последующий элементы



последующий

Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

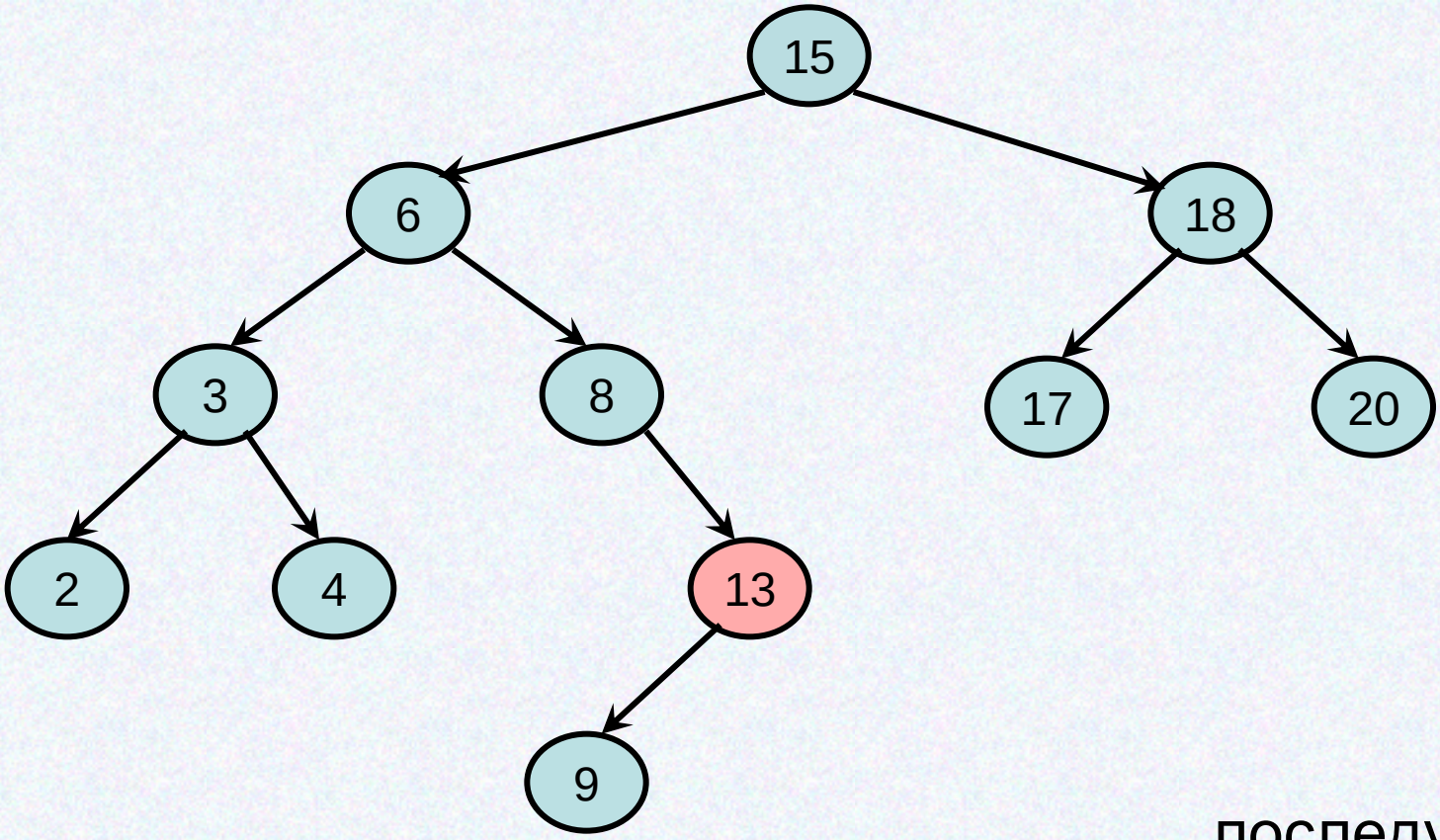
Предшествующий и последующий элементы



последующий

Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

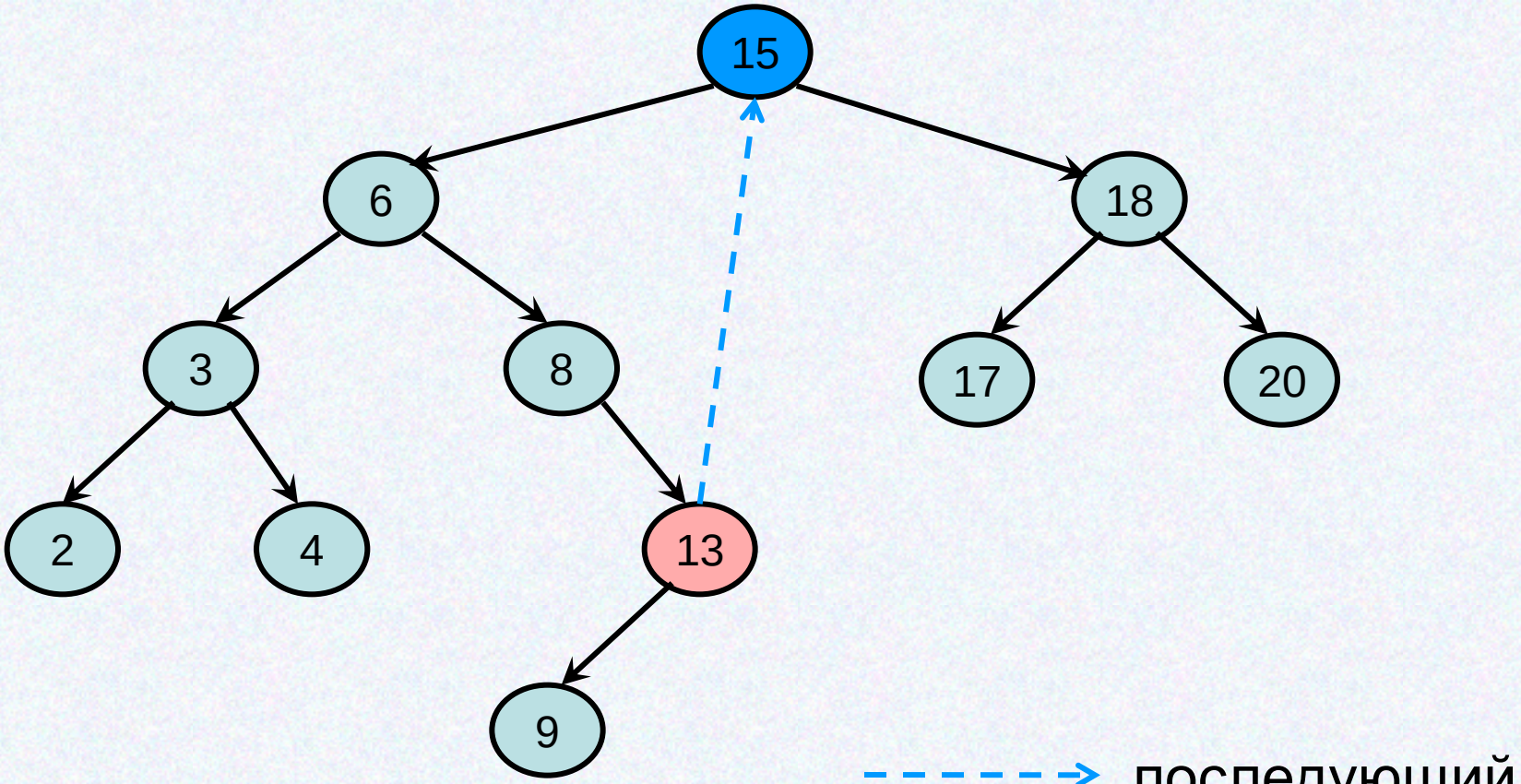
Предшествующий и последующий элементы



последующий

Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

Предшествующий и последующий элементы



Обход дерева: 2, 3, 4, 6, 8, 9, 13, 15, 17, 18, 20

Поиск следующего элемента

Обозначения:

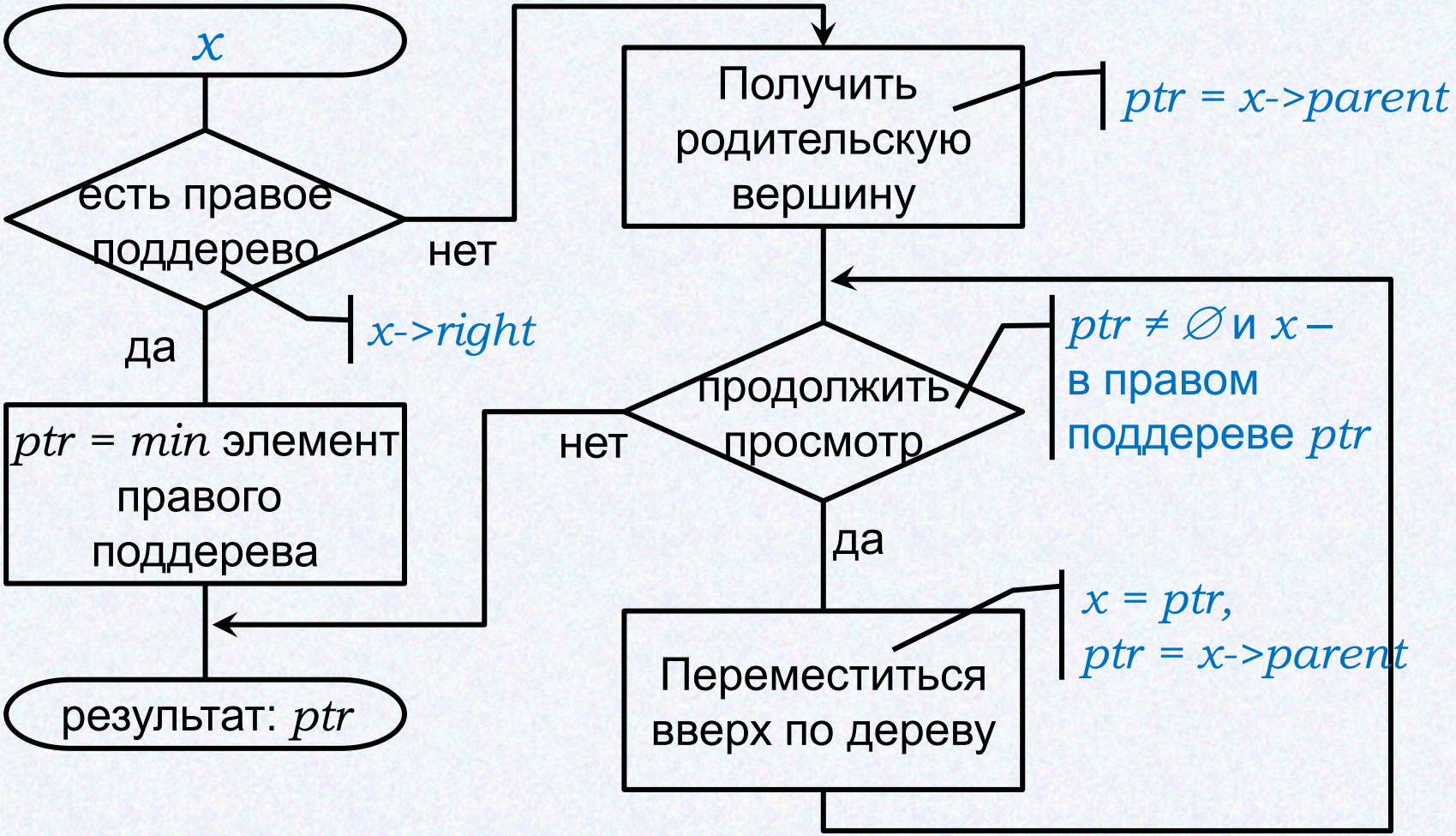
left, *right* – указатели на левое и правое поддеревья, соответственно

parent – указатель на родительскую вершину

x – указатель на заданную вершину

ptr – указатель на вершину, содержащую следующее значение

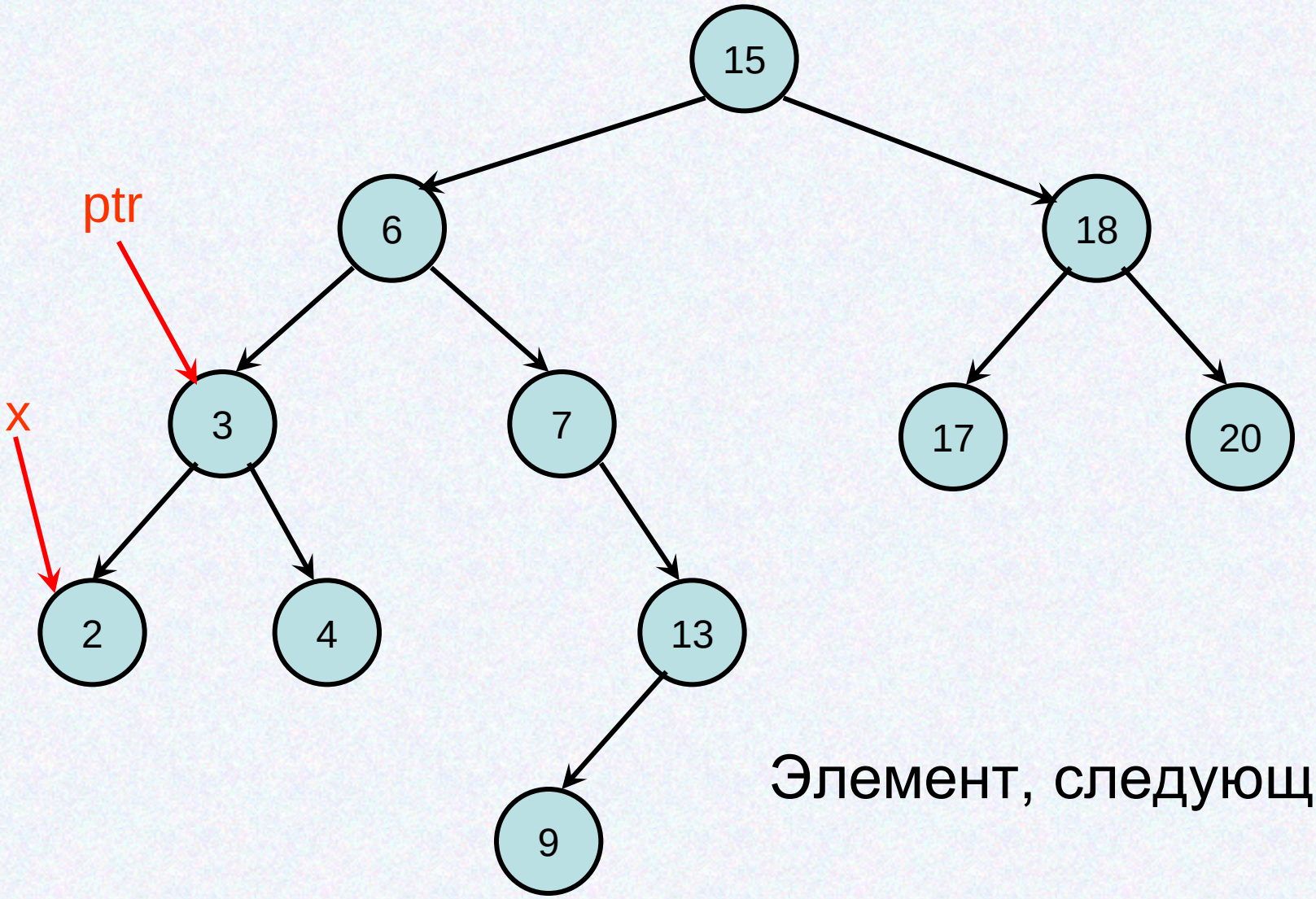
Поиск следующего элемента



Поиск следующего элемента

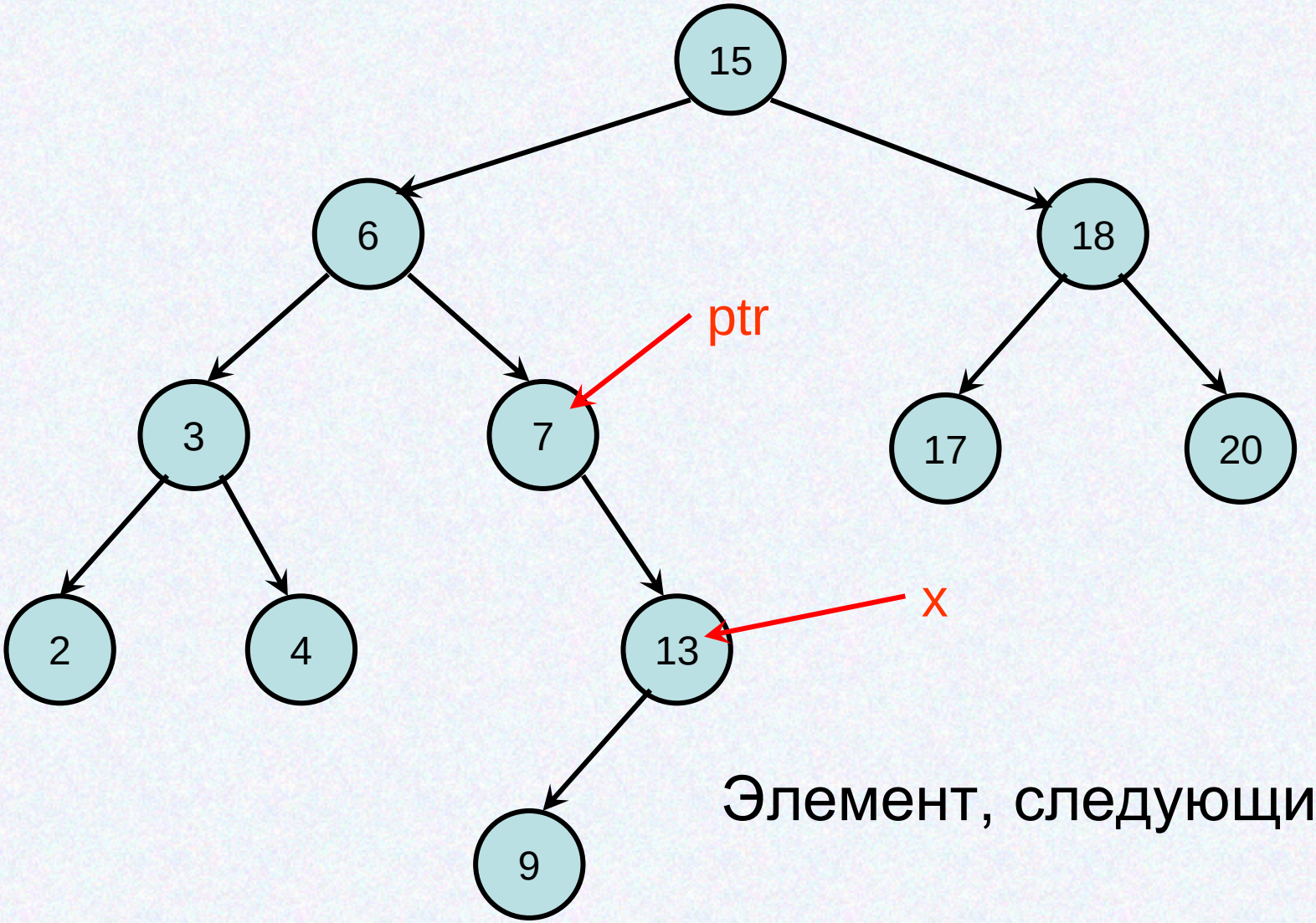
if есть правое поддерево {
    ptr = минимальный элемент правого поддерева
    Результат: *ptr*
}
Родительская вершина: *ptr* = *x->parent*
while есть родительская вершина и *x* – правое поддерево родительской вершины {
   x = *ptr* – новая текущая вершина
   новая родительская вершина: *ptr* = *x->parent*
}
Результат: *ptr*

Поиск следующего элемента



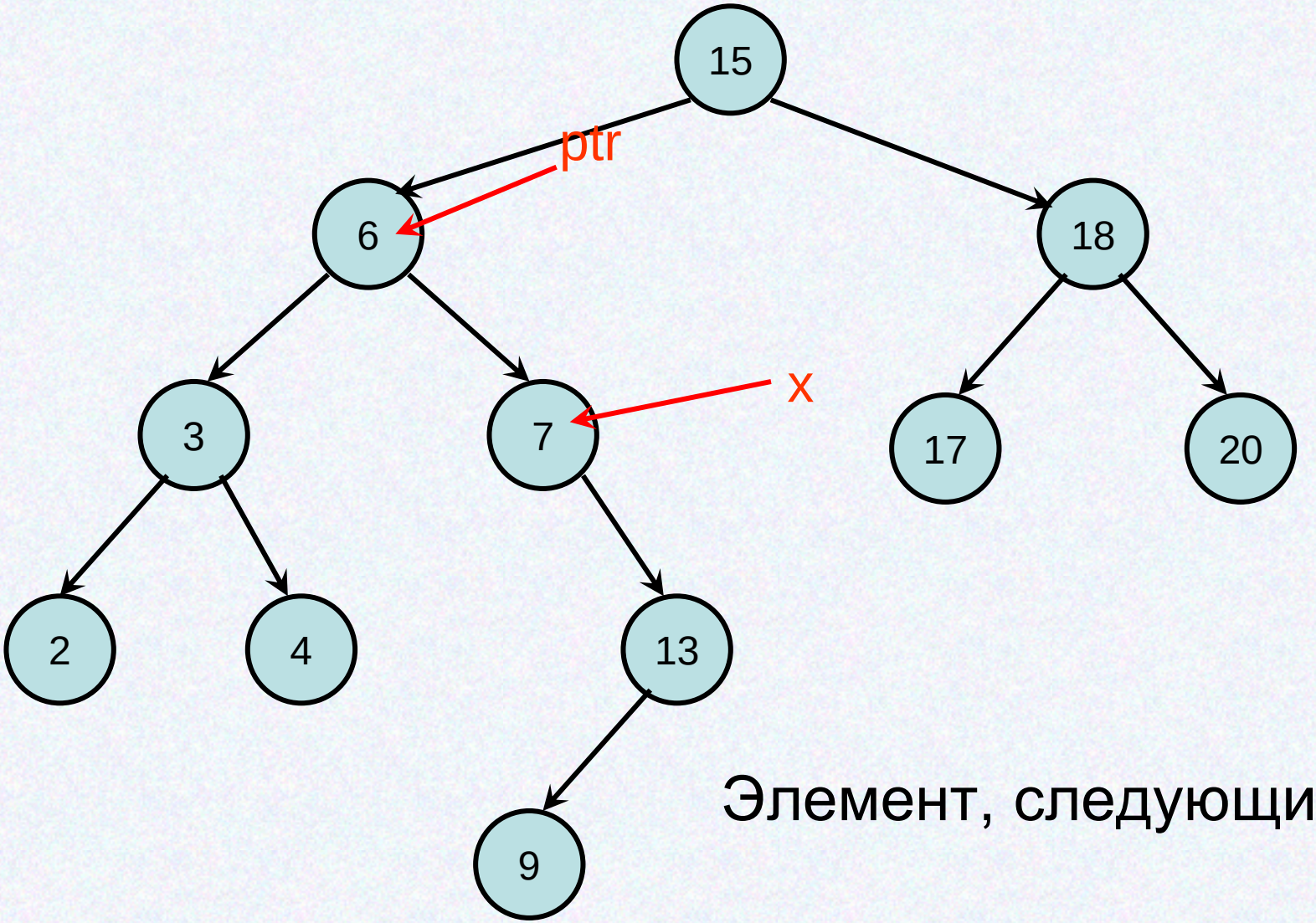
Элемент, следующий за 2

Поиск следующего элемента



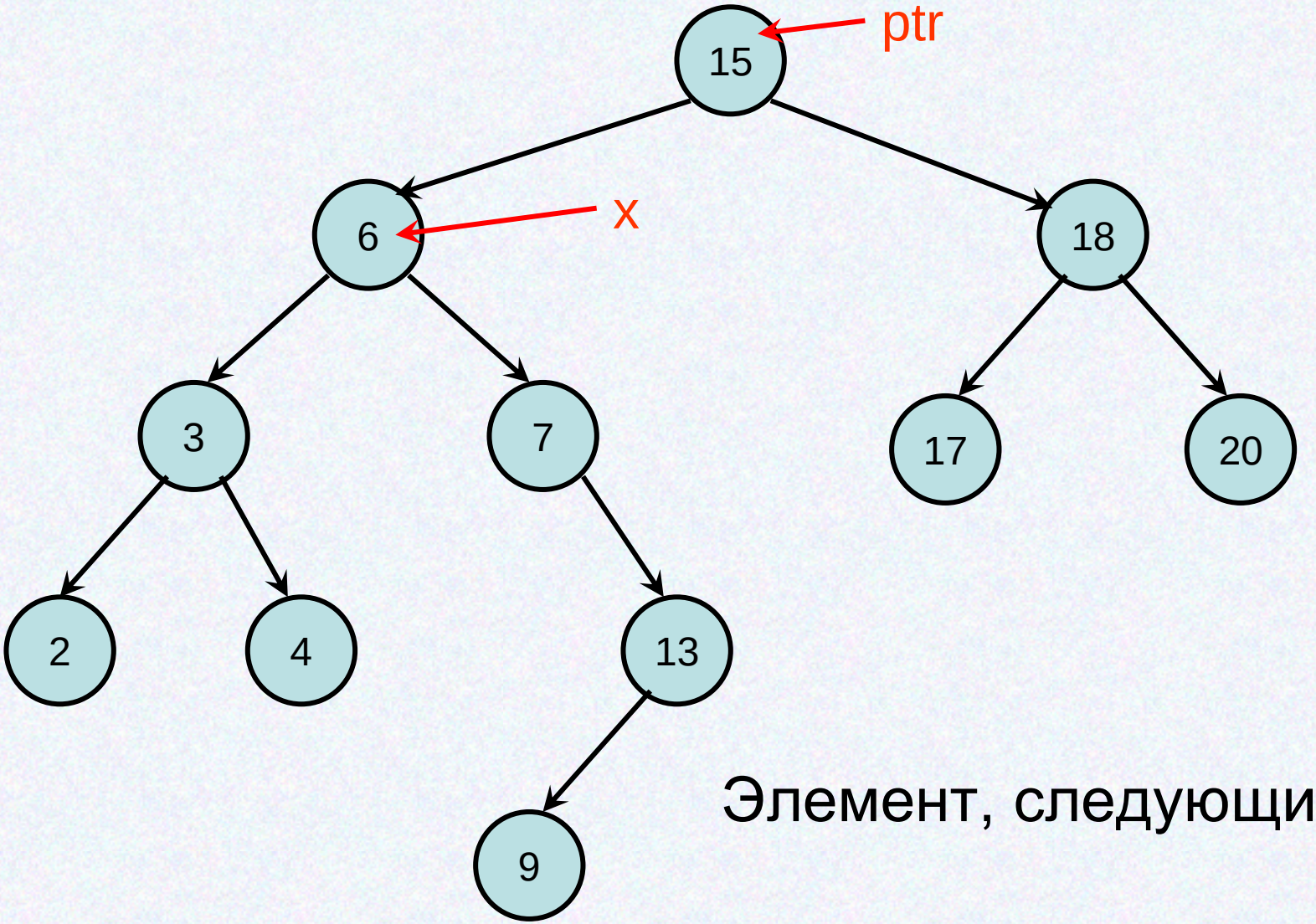
Элемент, следующий за 13

Поиск следующего элемента



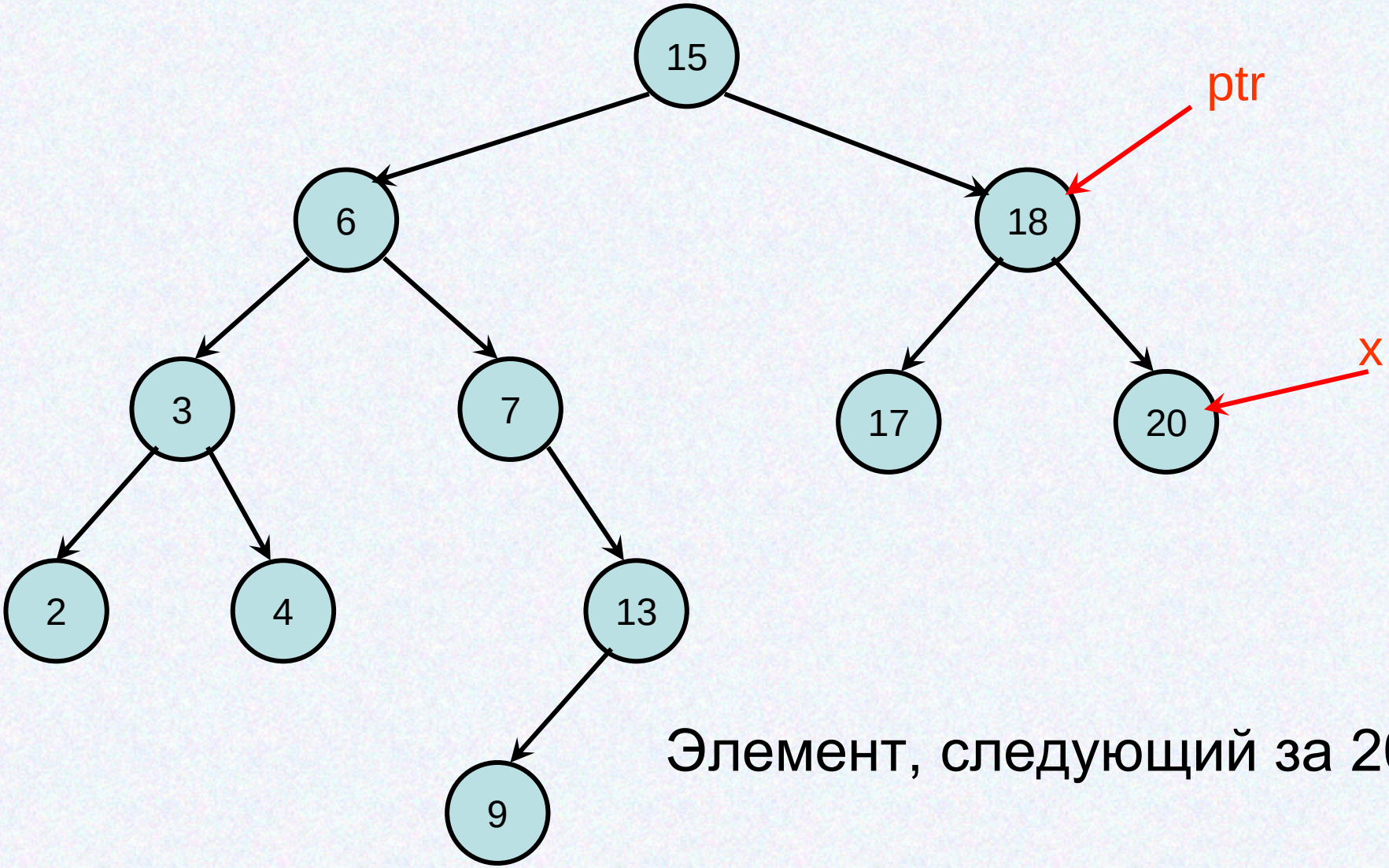
Элемент, следующий за 13

Поиск следующего элемента



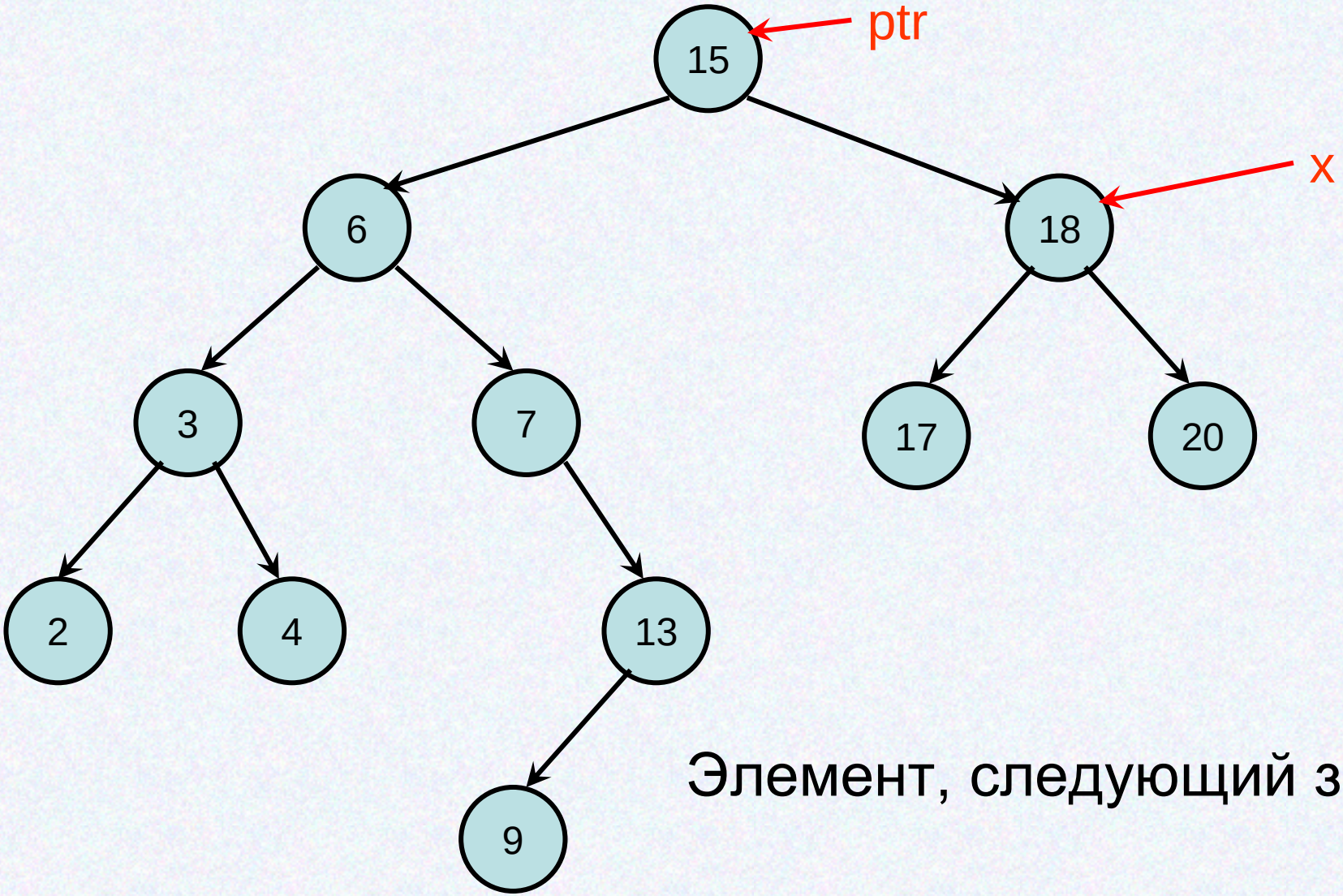
Элемент, следующий за 13

Поиск следующего элемента



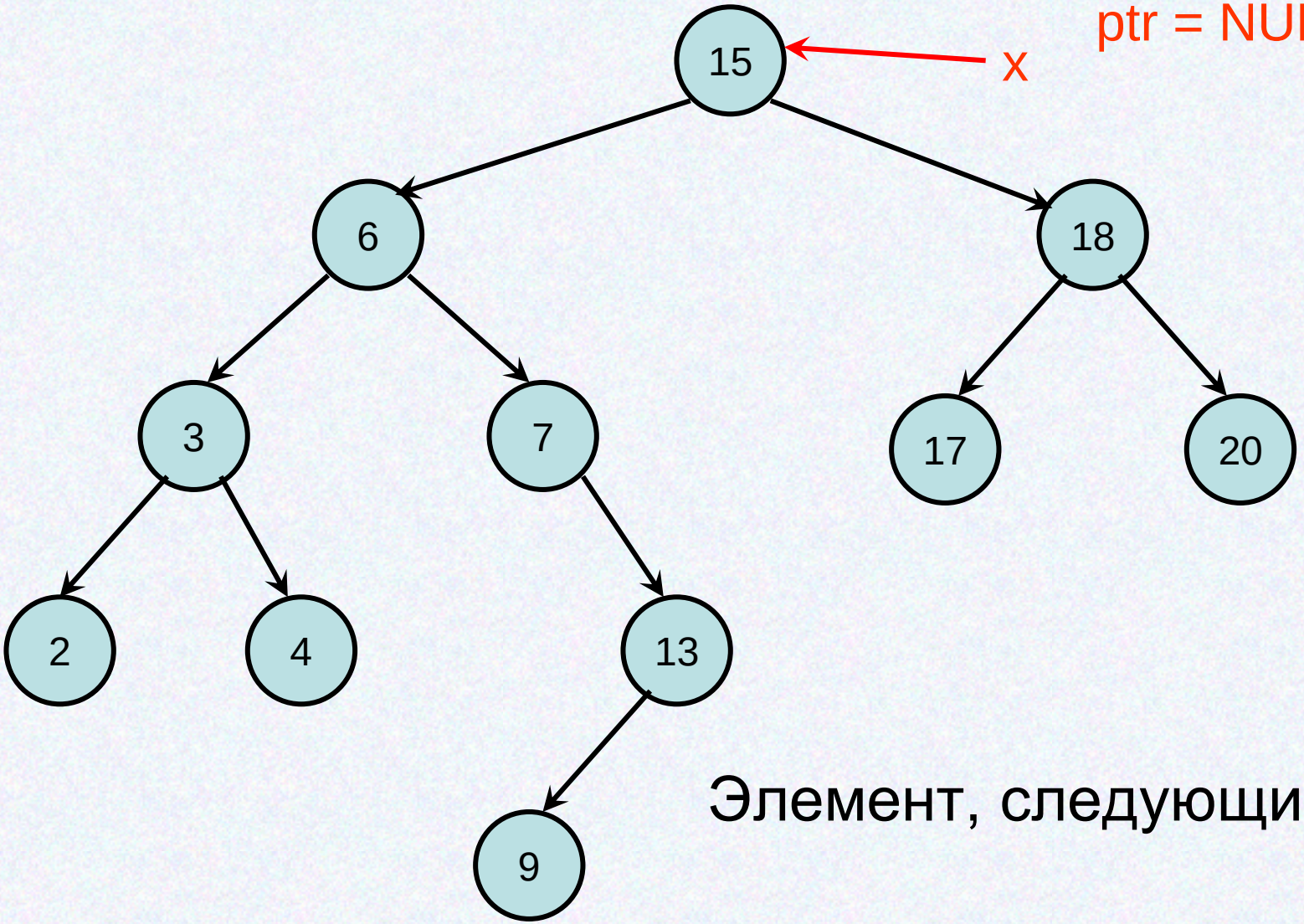
Элемент, следующий за 20

Поиск следующего элемента



Элемент, следующий за 20

Поиск следующего элемента

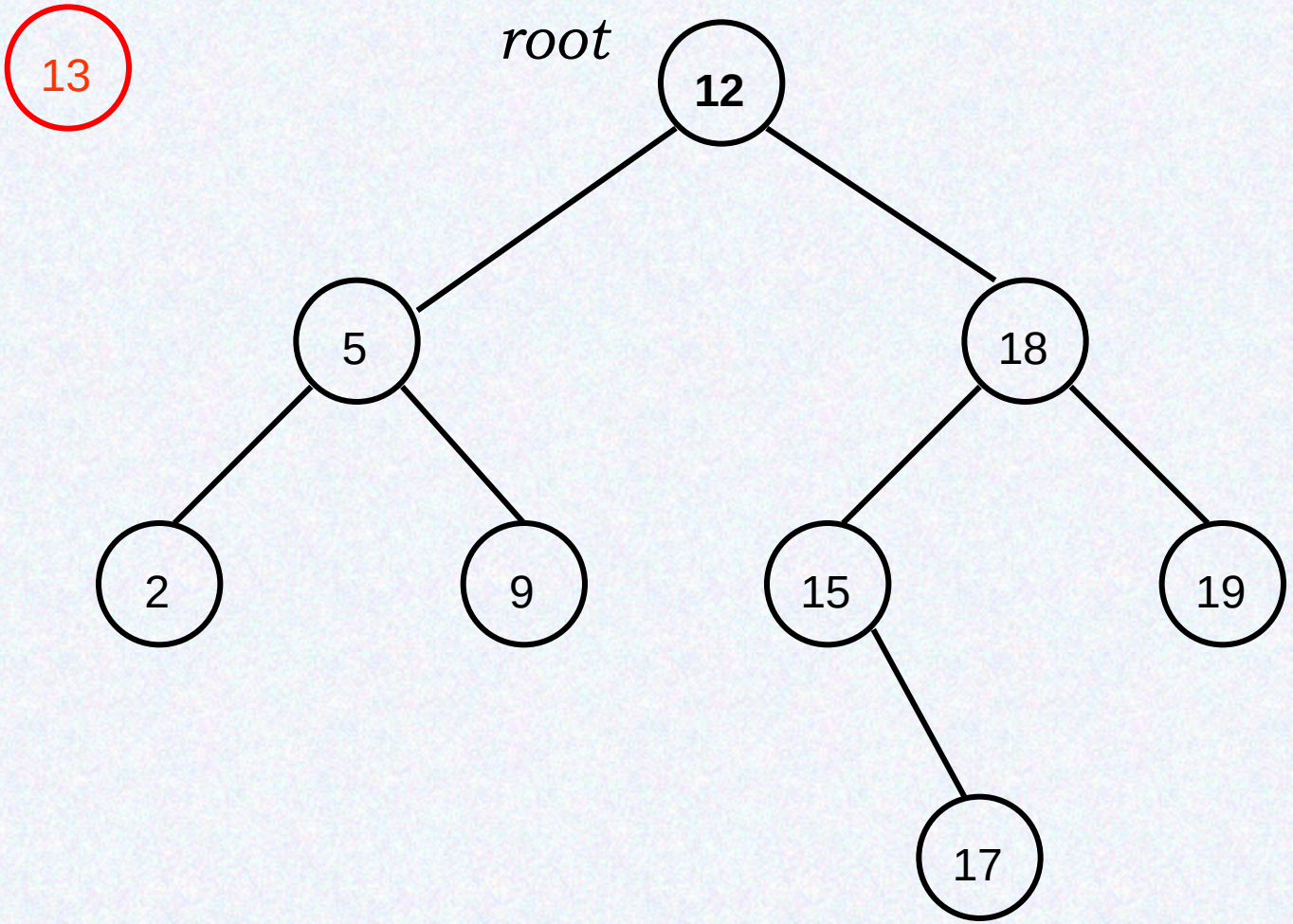


Элемент, следующий за 20

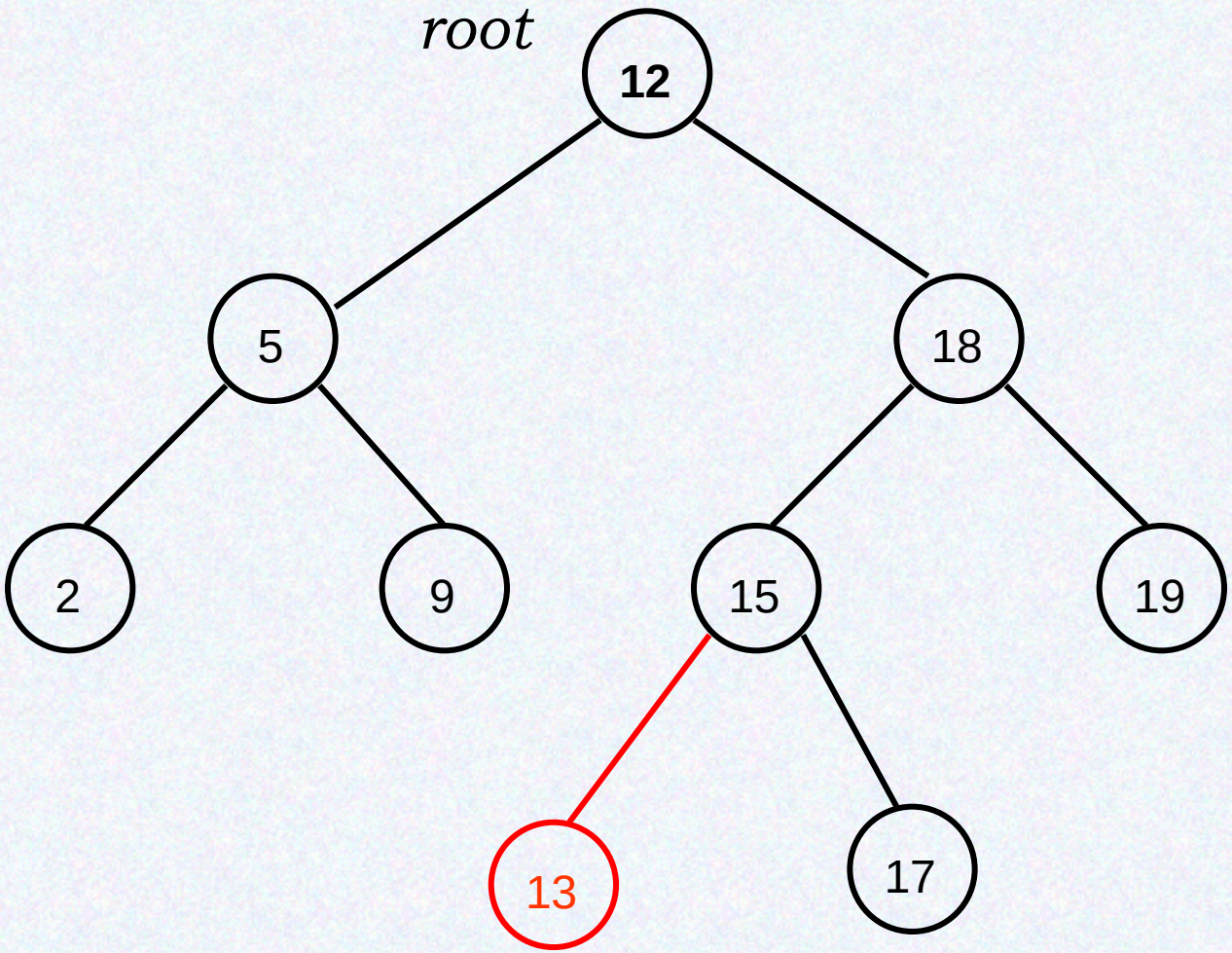
Характеристики поиска

Операции поиска, определения максимального и минимального элемента, а также предшествующего и последующего в бинарном дереве поиска высоты h могут быть выполнены за время $O(h)$

Вставка нового элемента

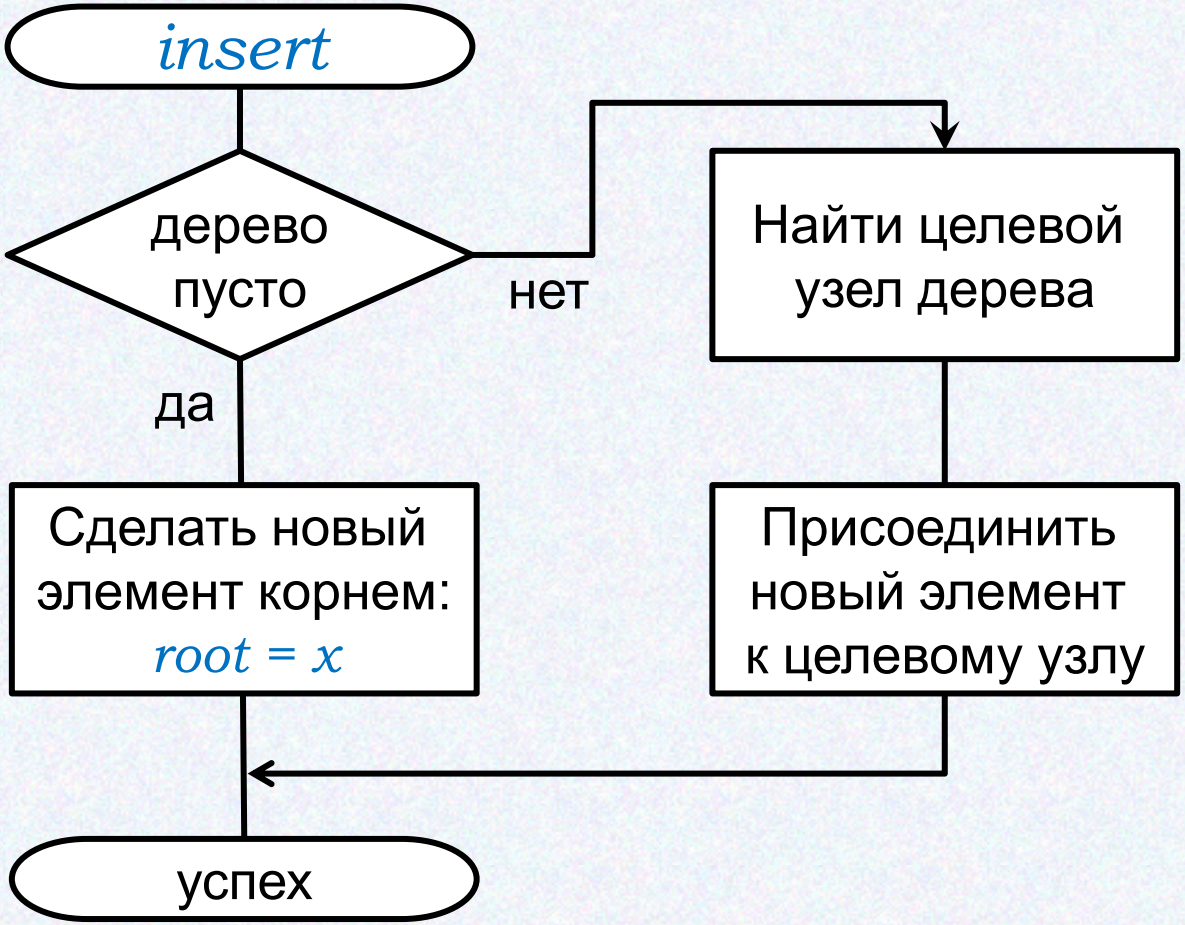


Вставка нового элемента

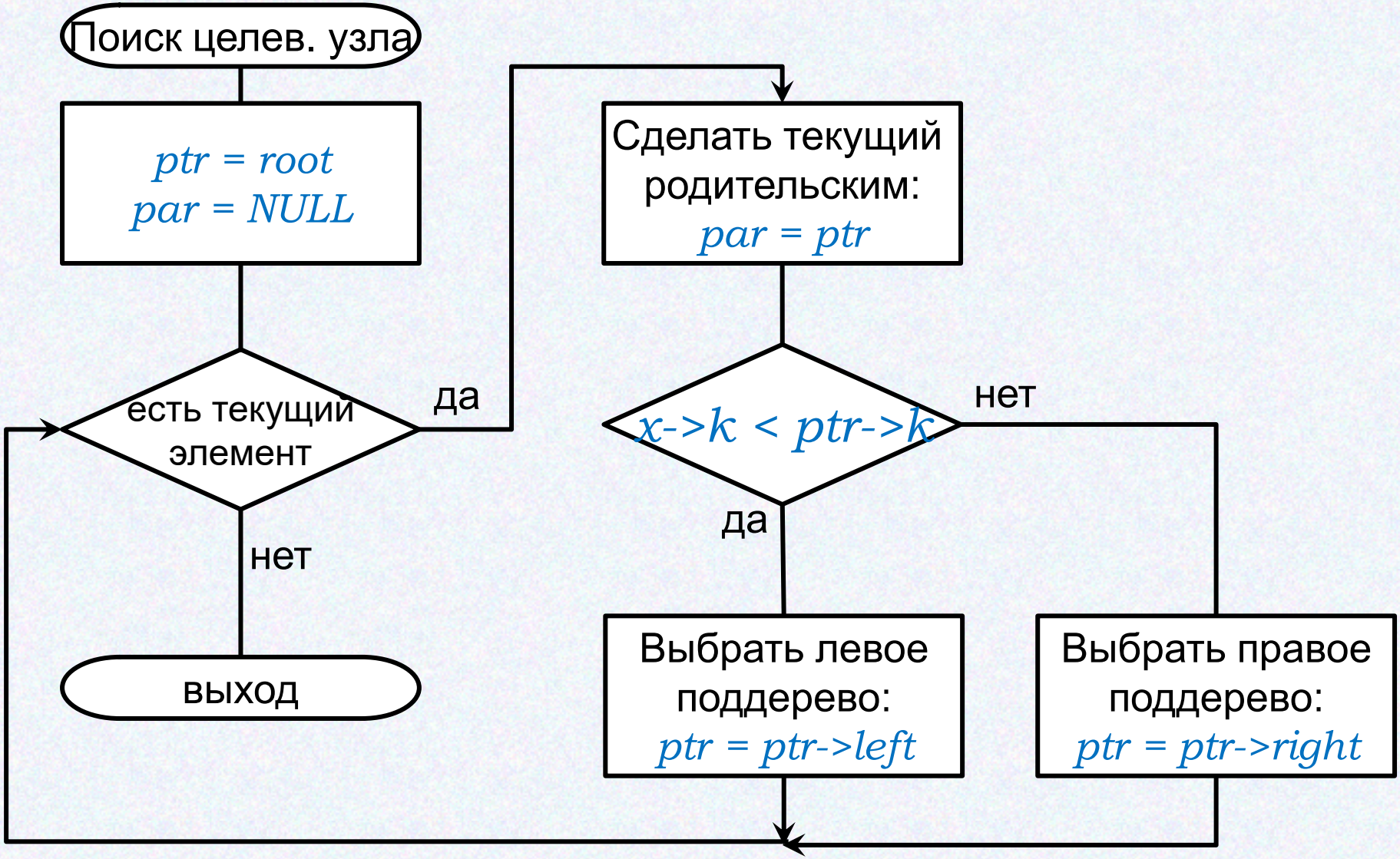


Вставка нового элемента

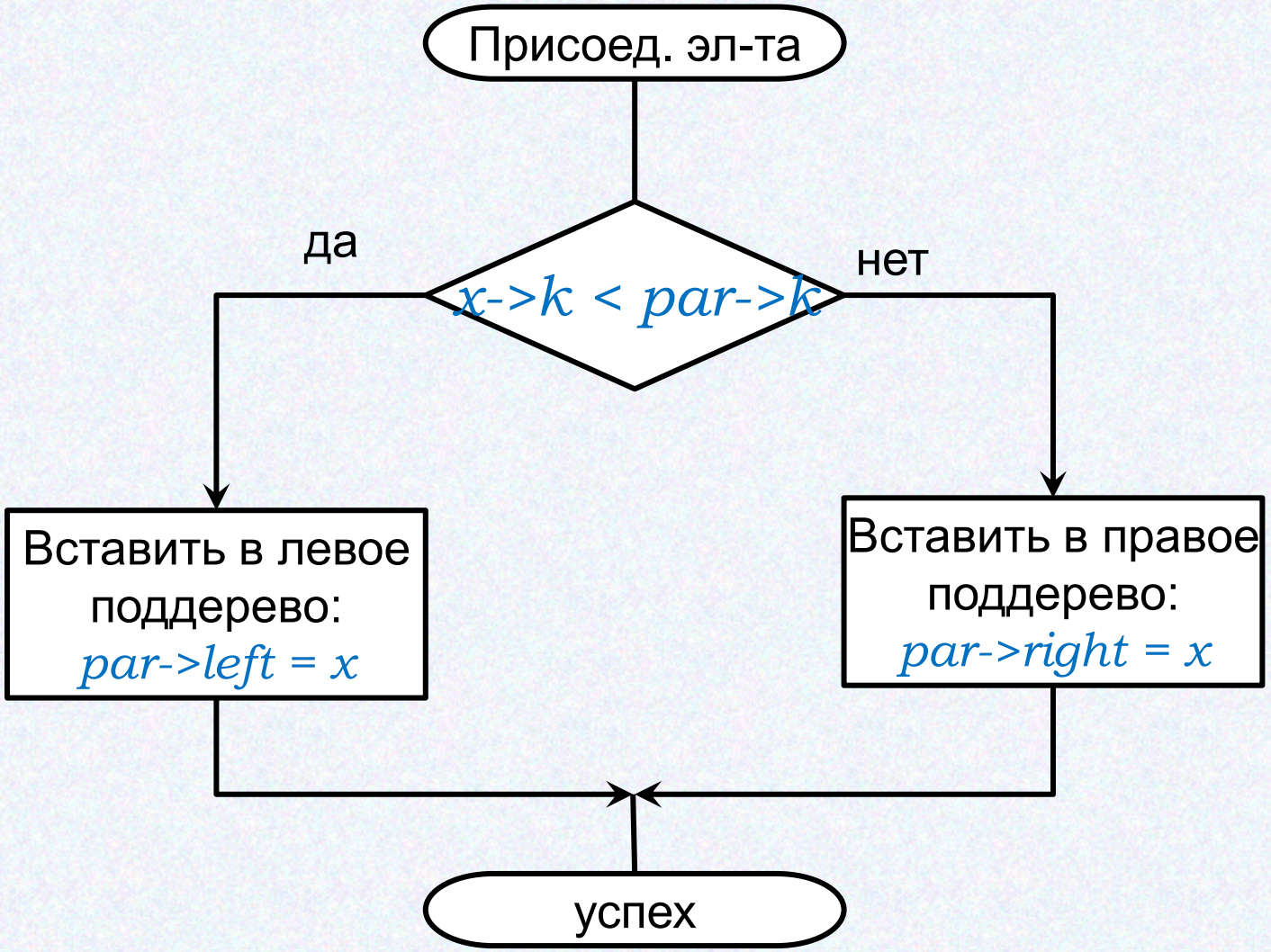
root – указатель на корень дерева
x – указатель на новый элемент



Вставка нового элемента



Вставка нового элемента



Вставка нового элемента

root – указатель на корень дерева

x – указатель на новый элемент

if дерево пусто: *root == NULL* {

  сделать новый элемент корнем: *root = x*

  успех

}

текущий элемент *ptr = root*

родительский элемент *par = NULL*

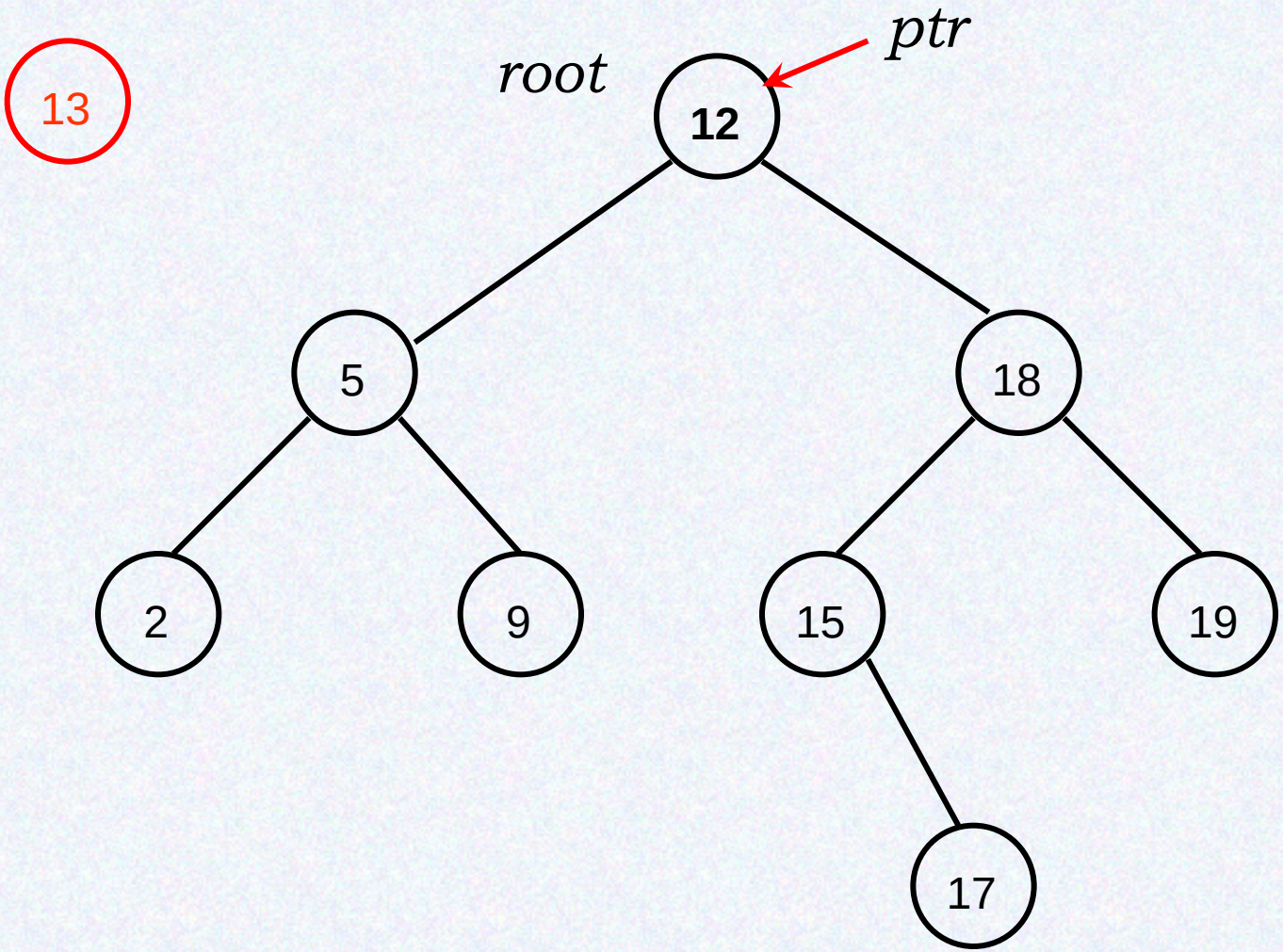
Вставка нового элемента

while есть текущий элемент: *ptr ≠ NULL* {
  par = ptr
  if ключ нового элемента < ключа текущего элемента: *x->key < ptr->key*
    выбрать левое поддерево: *ptr = ptr->left*
  else
    выбрать правое поддерево: *ptr = ptr->right*
}

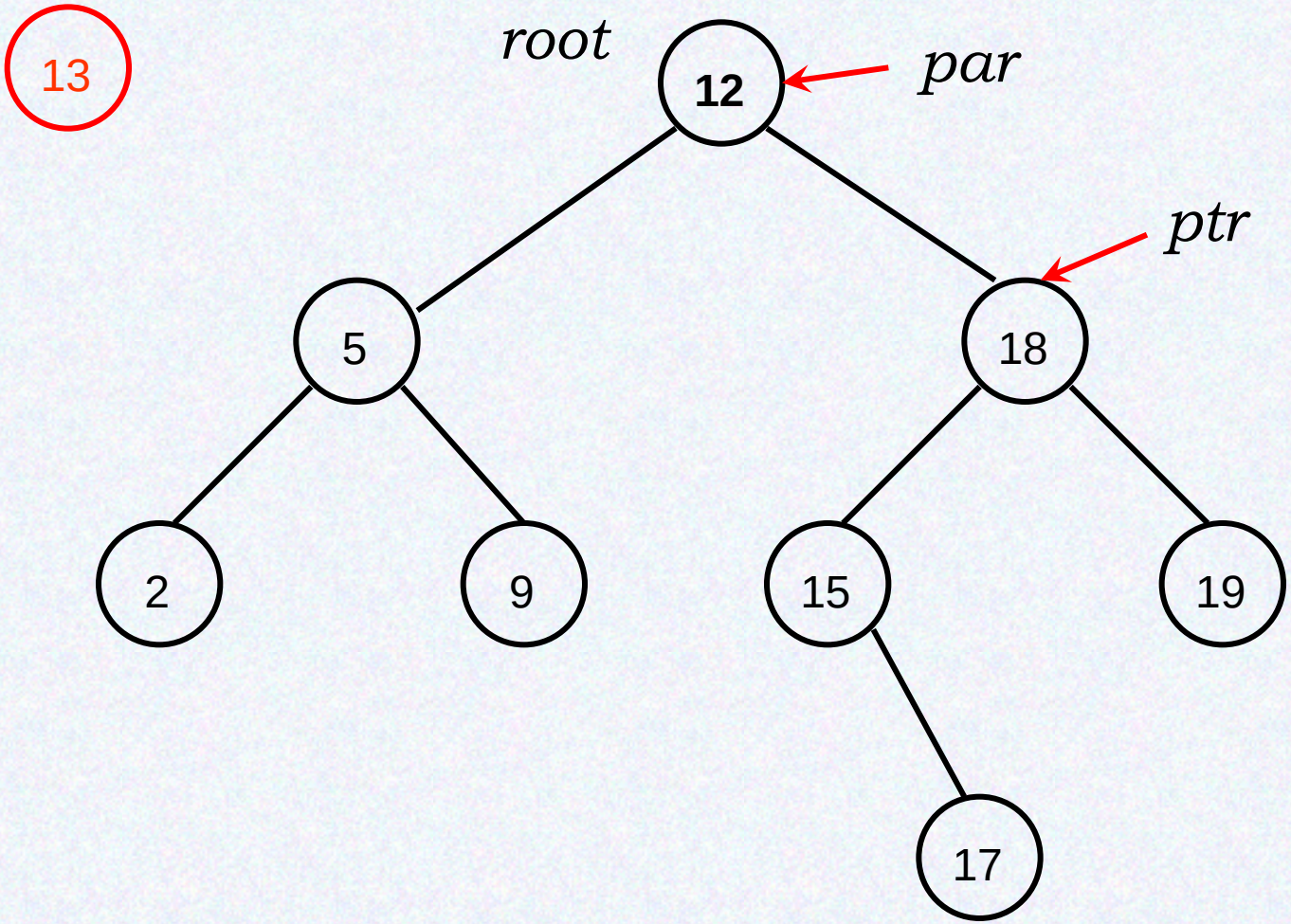
Вставка нового элемента

if ключ нового элемента < ключа элемента
вершины *par* *(x->key < par->key)*
включить новый элемент как левое
поддерево вершины *par* *(par->left = x)*
else
включить новый элемент как правое
поддерево вершины *par* *(par->right = x)*
успех

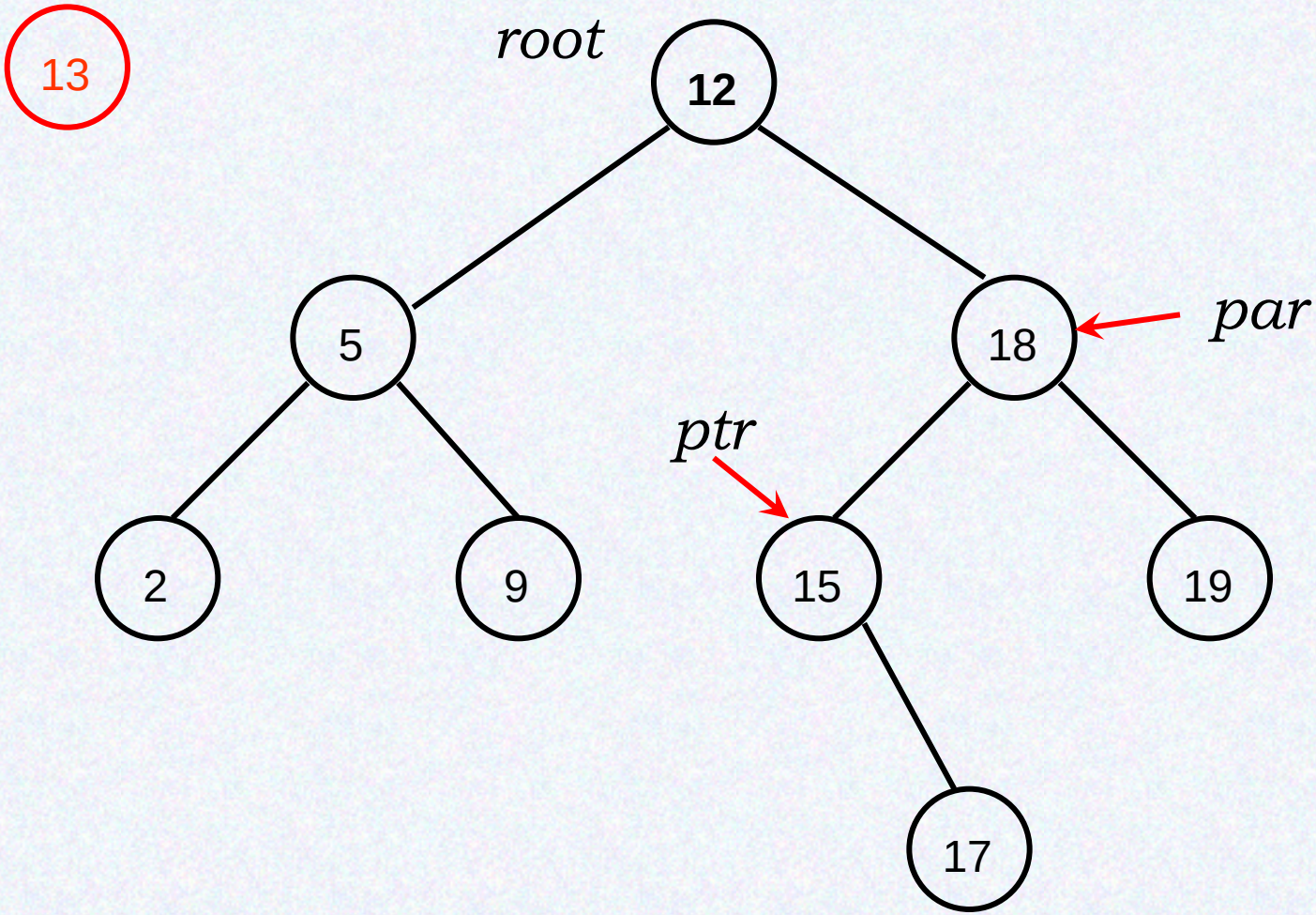
Вставка нового элемента



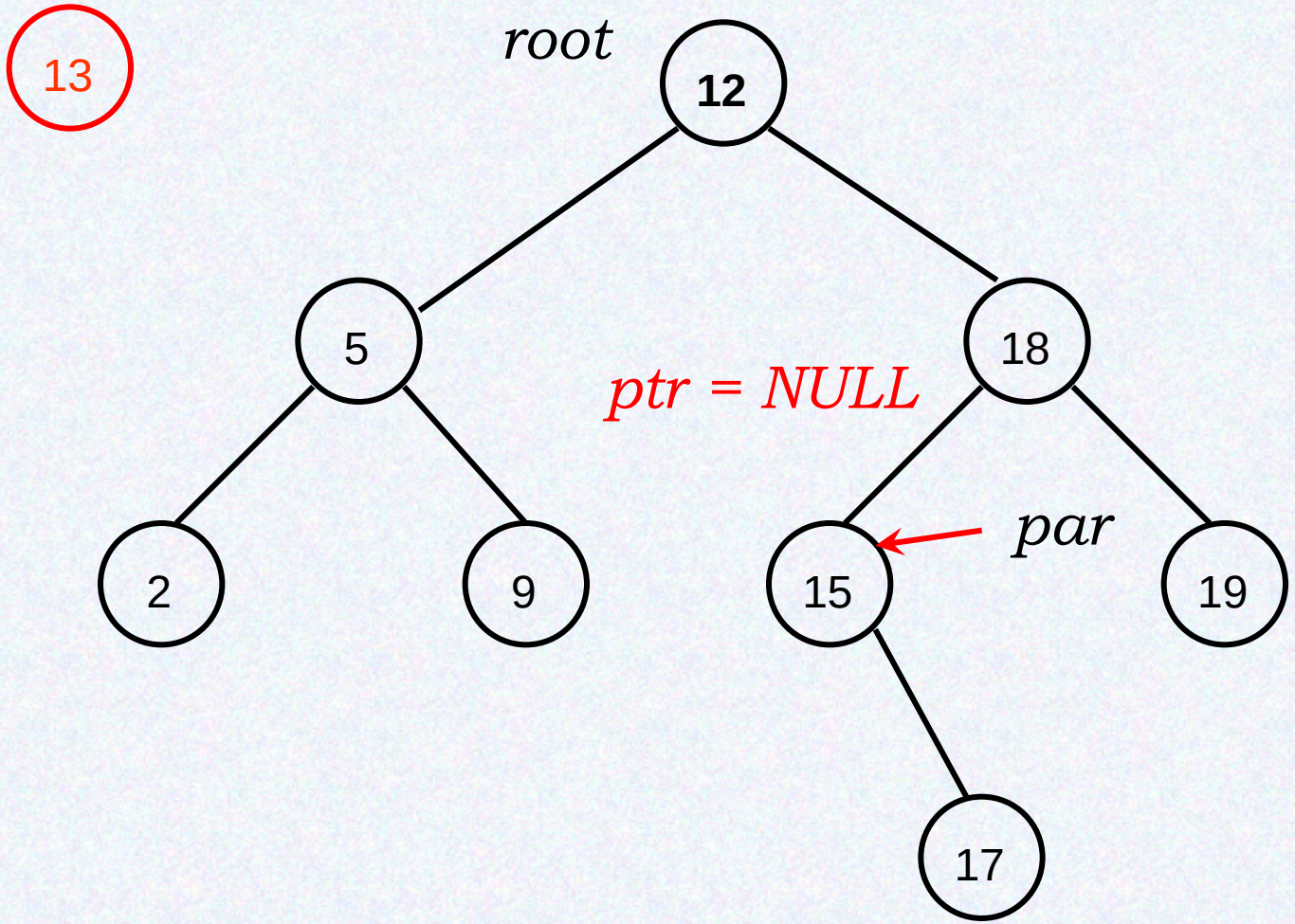
Вставка нового элемента



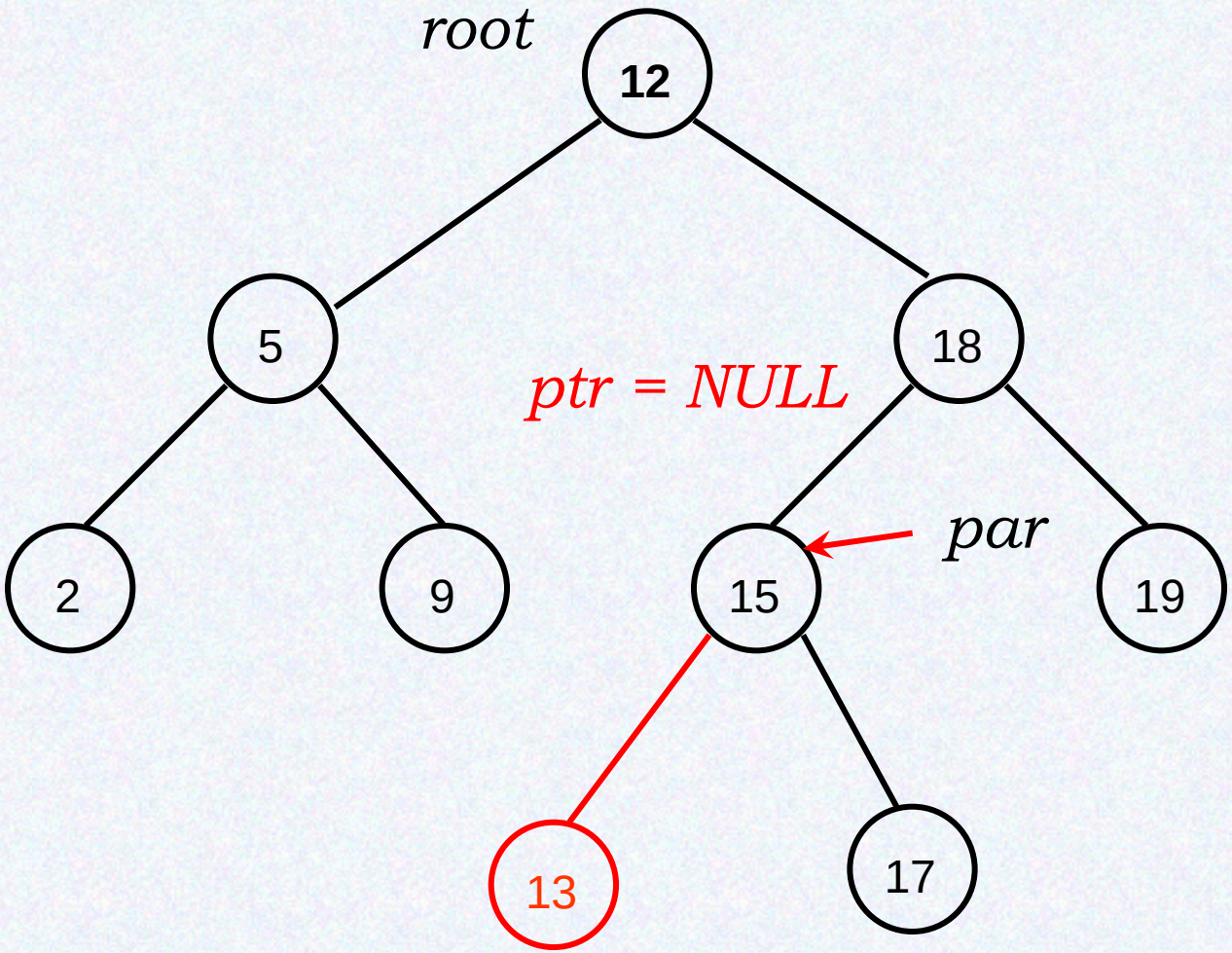
Вставка нового элемента



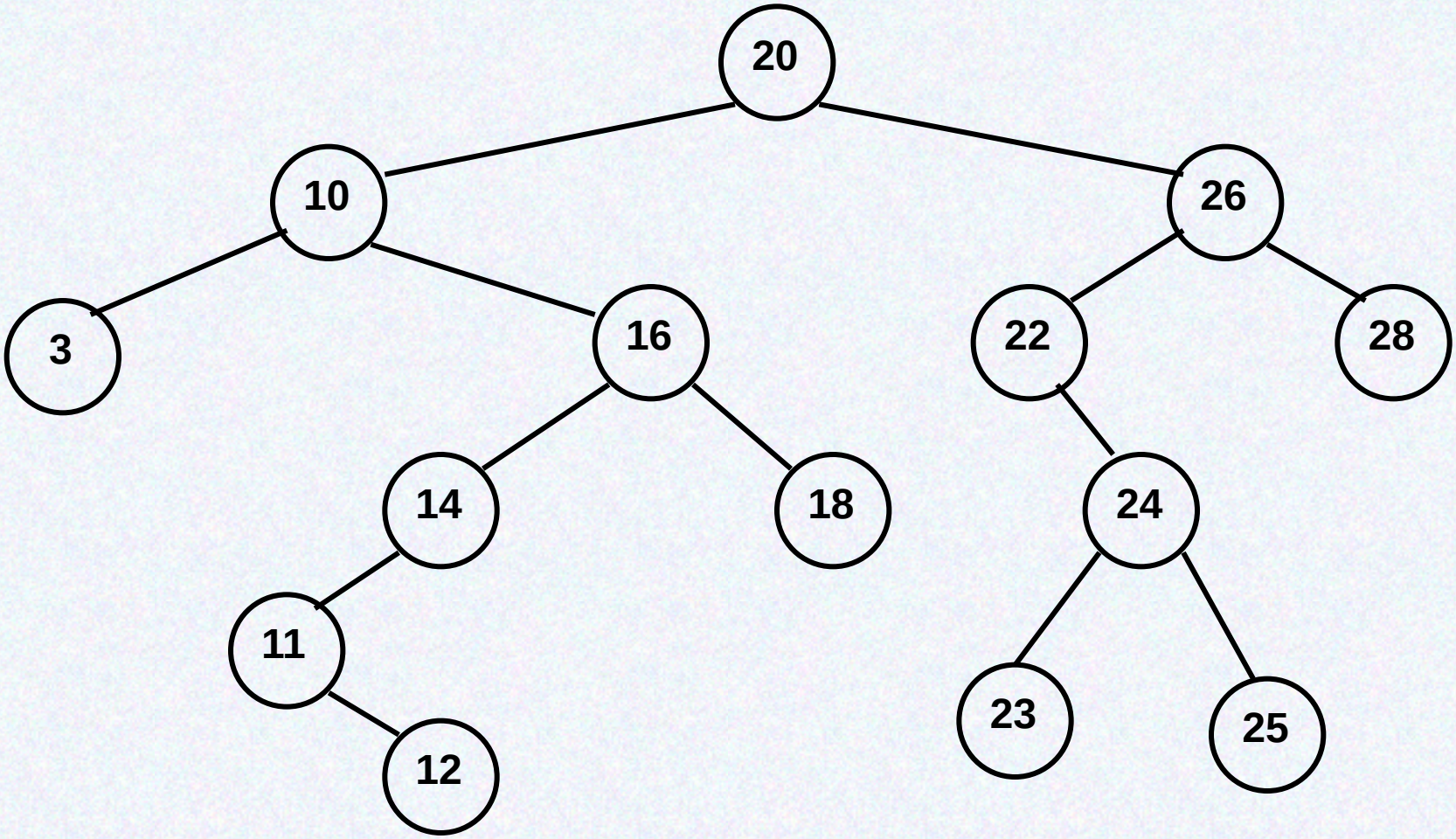
Вставка нового элемента



Вставка нового элемента

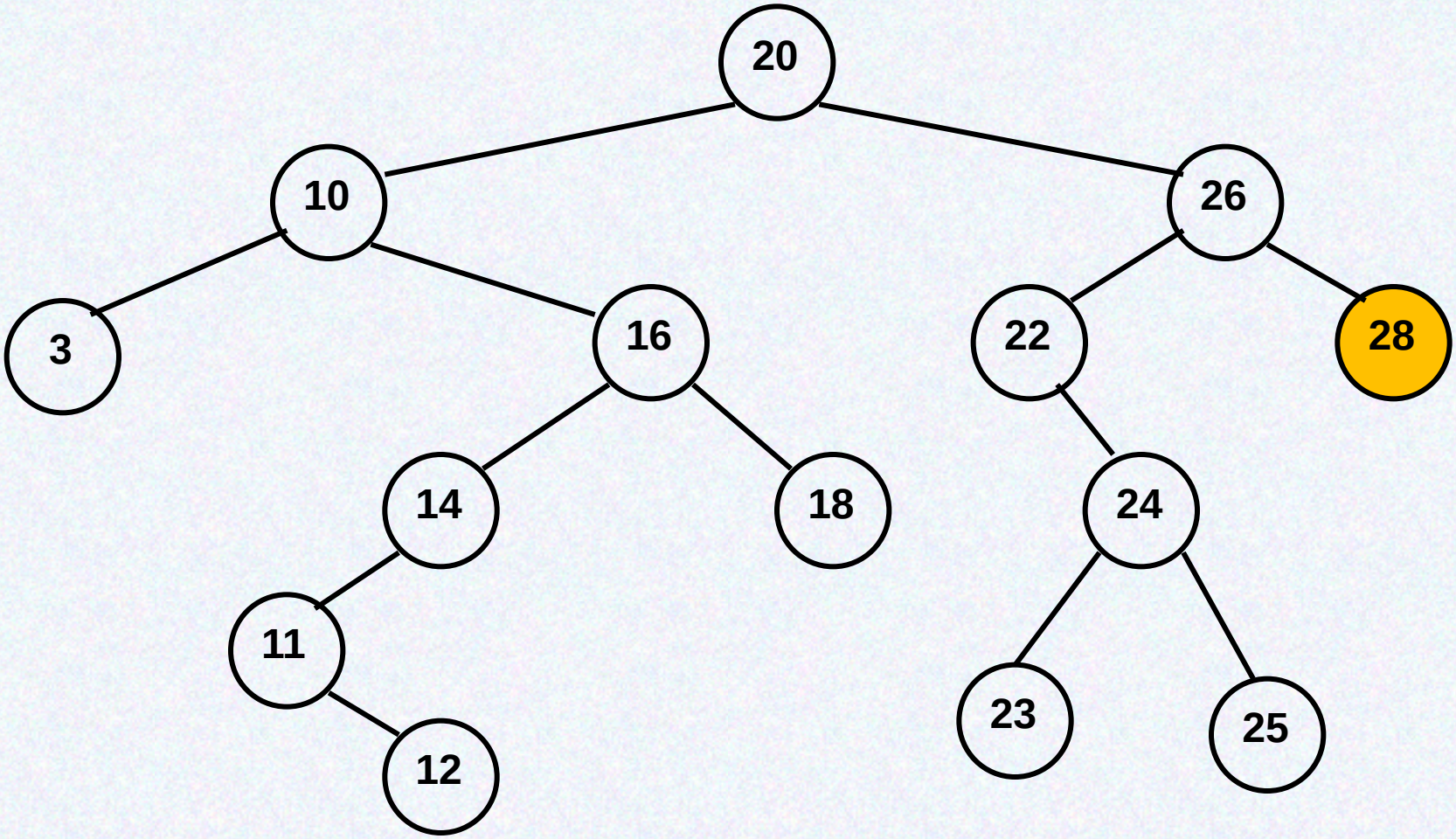


Удаление элемента



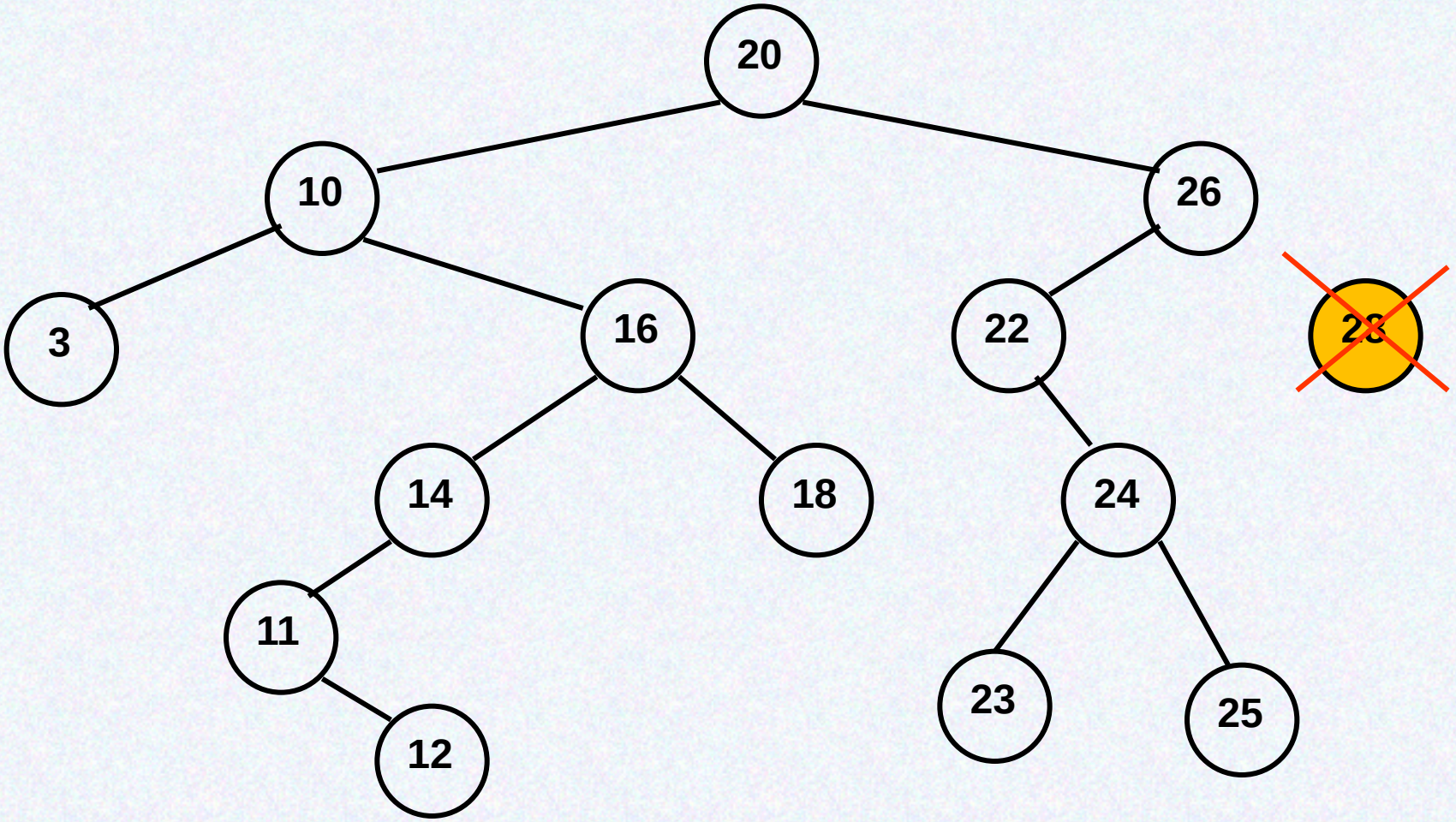
Удалить 28

Удаление элемента



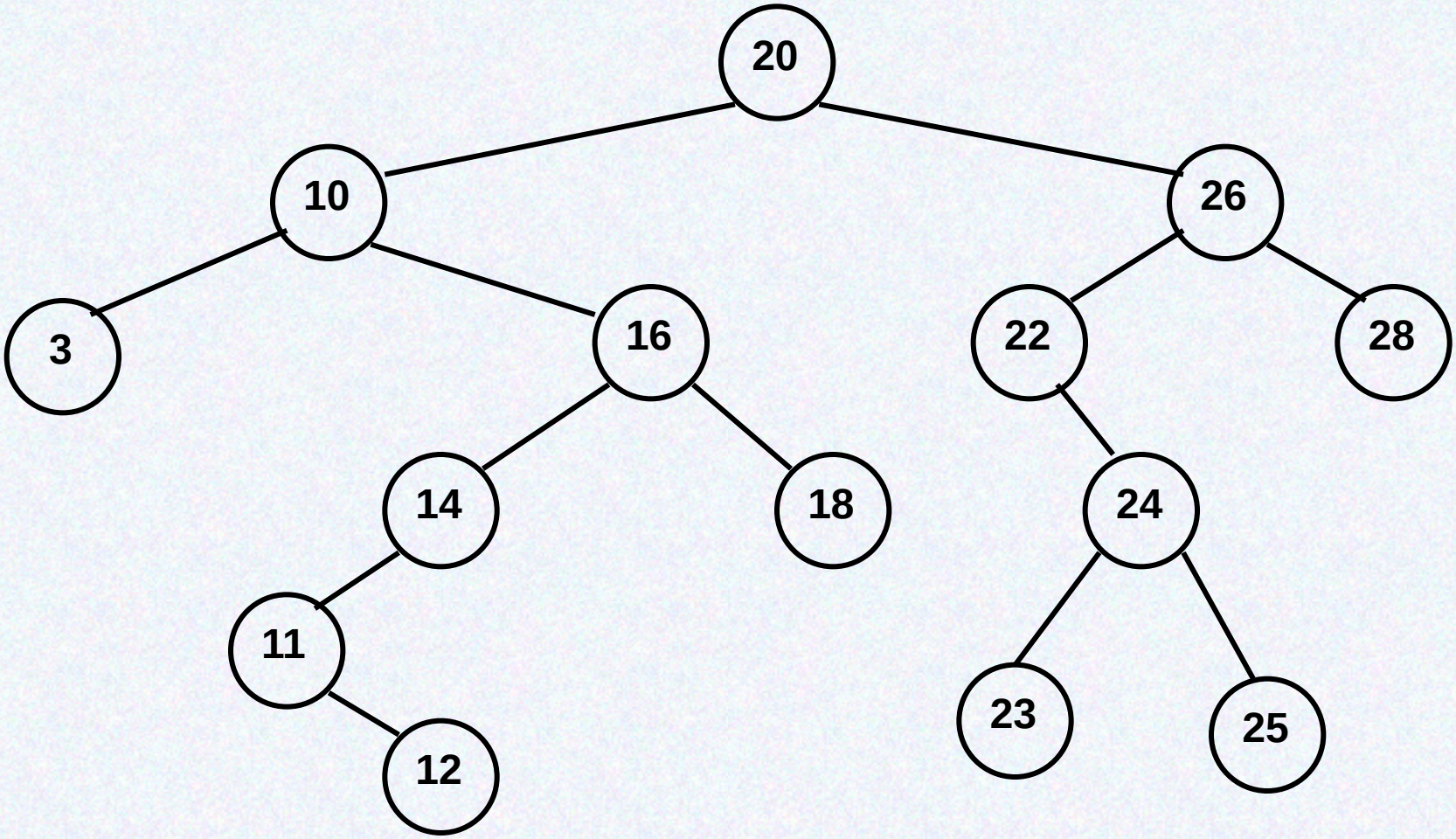
Удалить 28

Удаление элемента



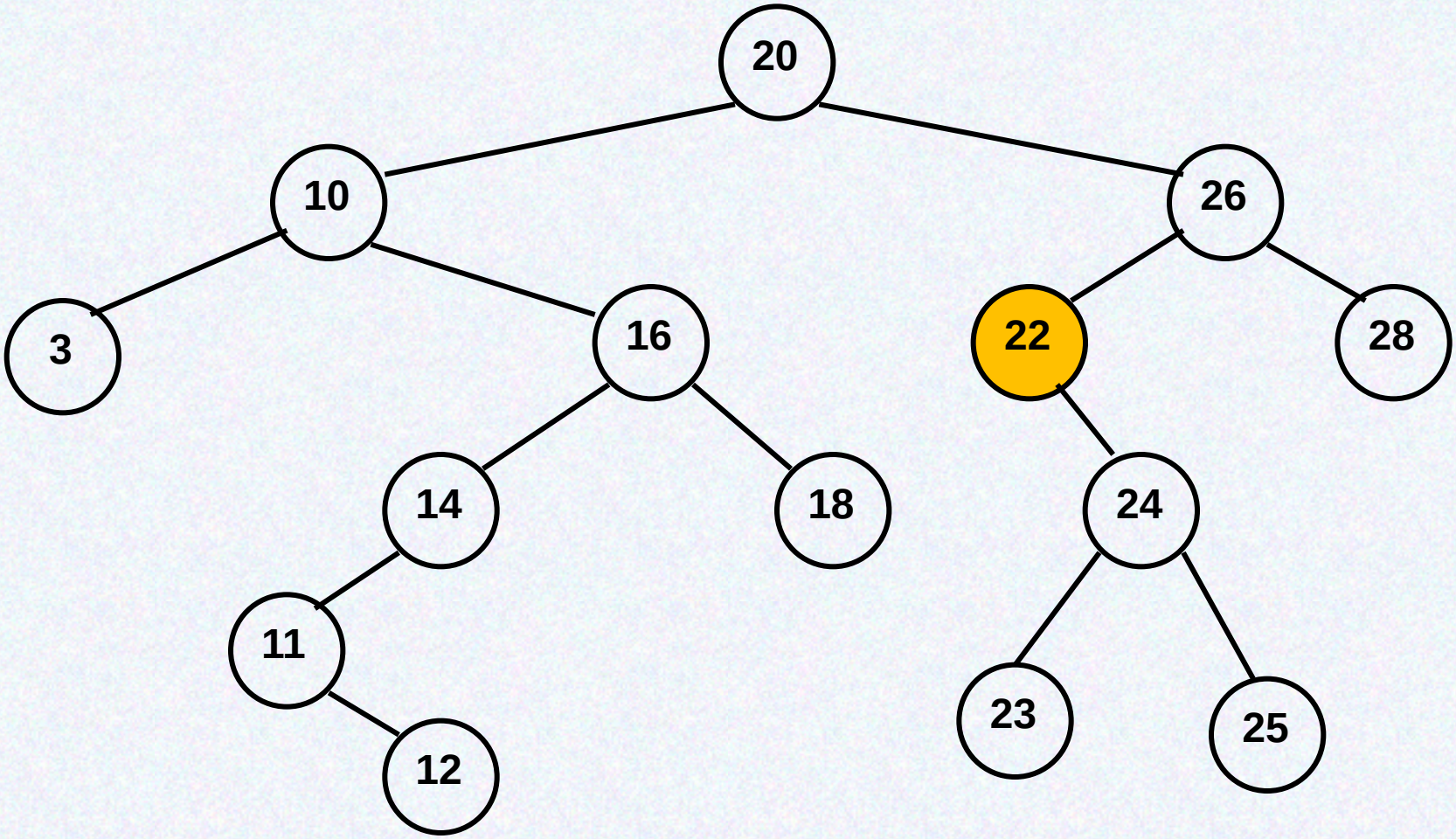
Удалить 28

Удаление элемента



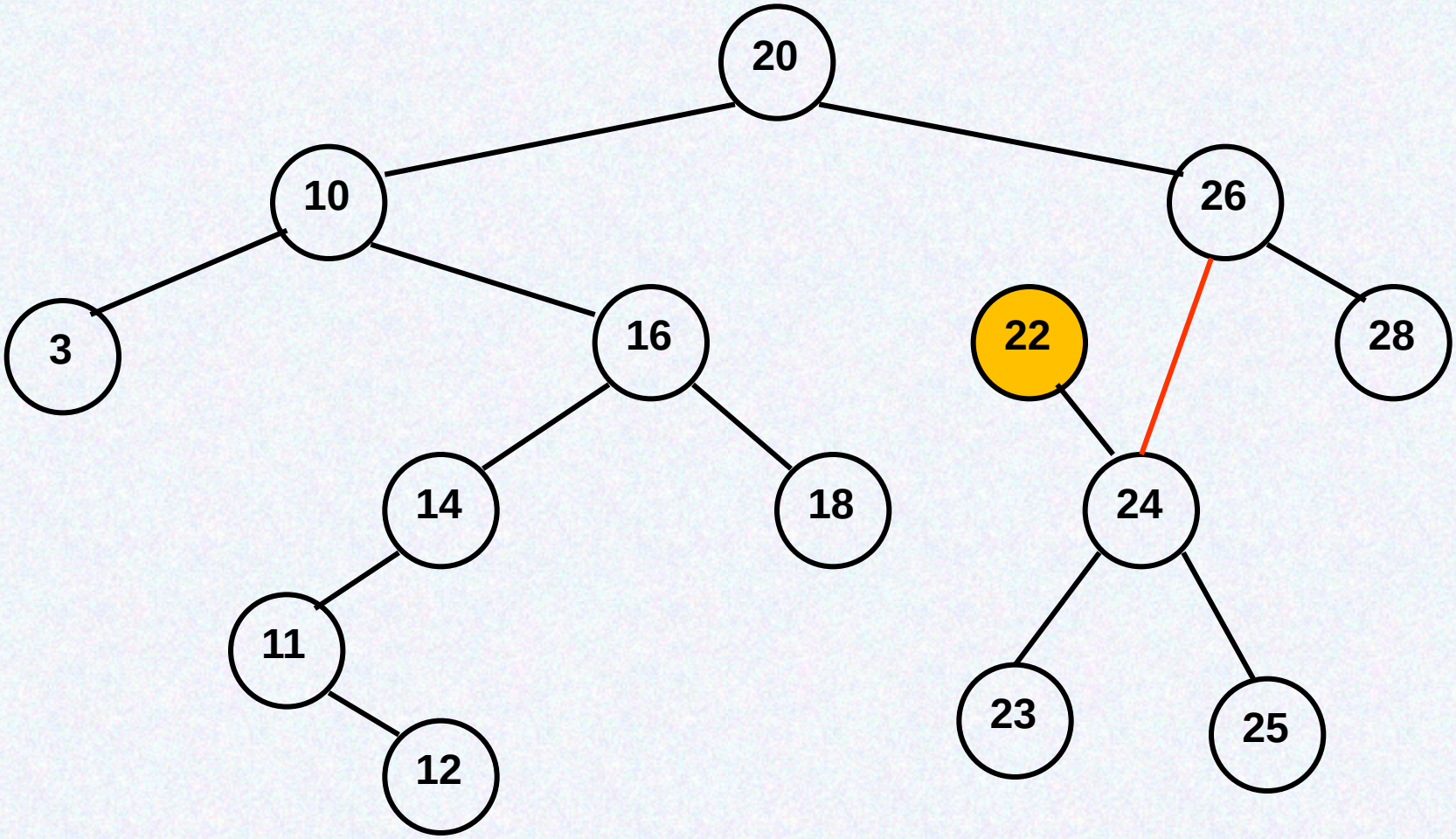
Удалить 22

Удаление элемента



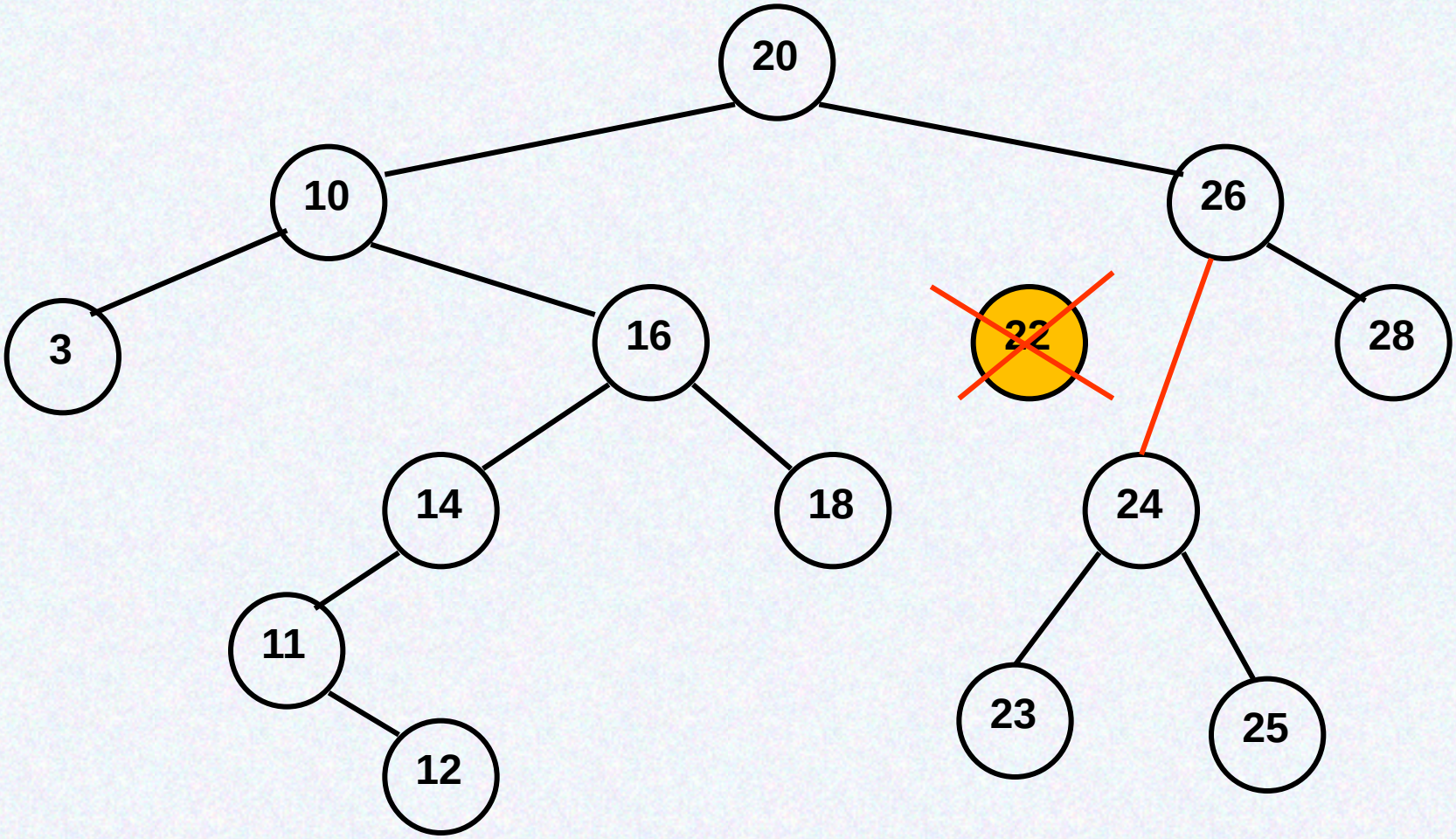
Удалить 22

Удаление элемента



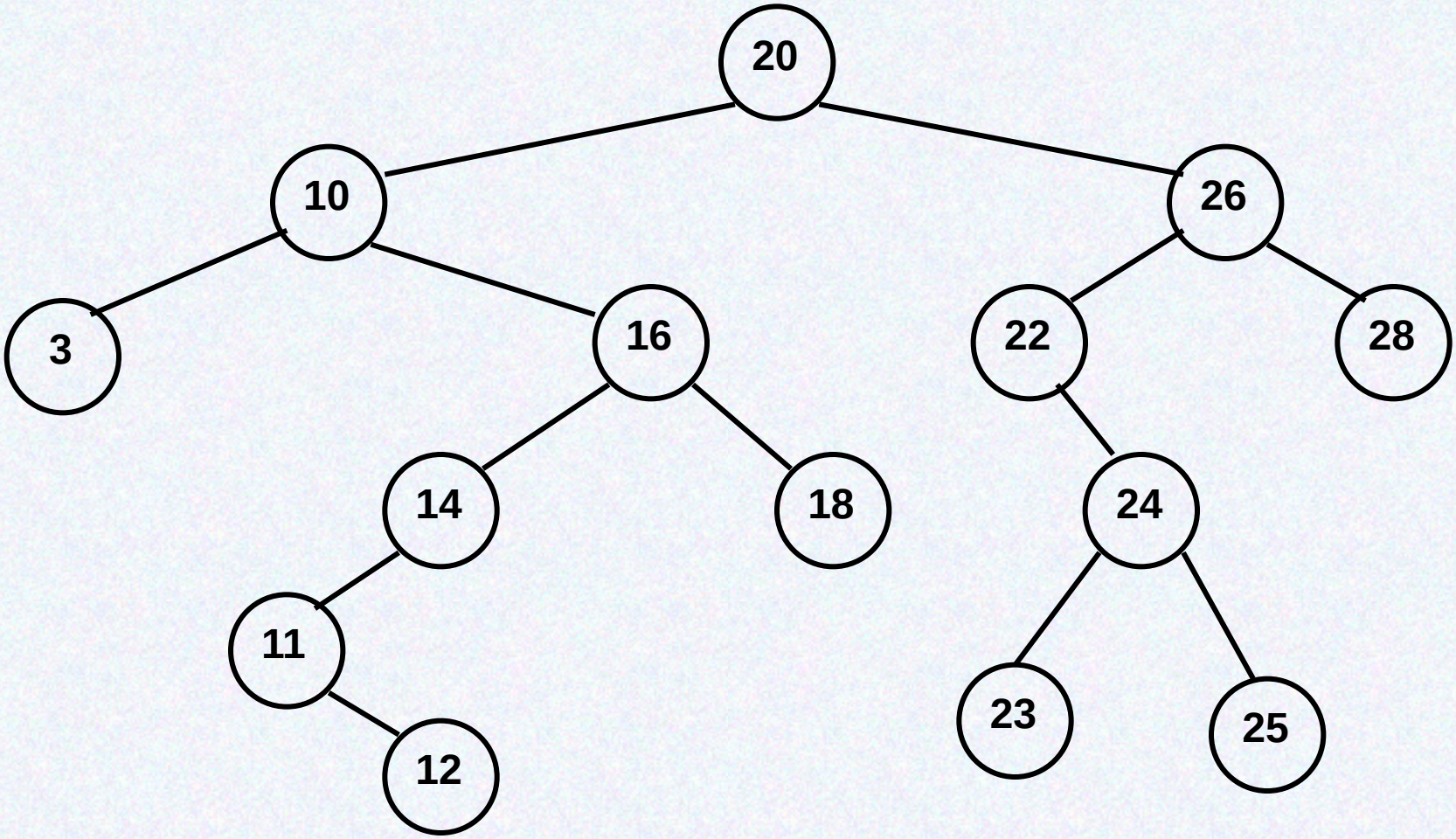
Удалить 22

Удаление элемента



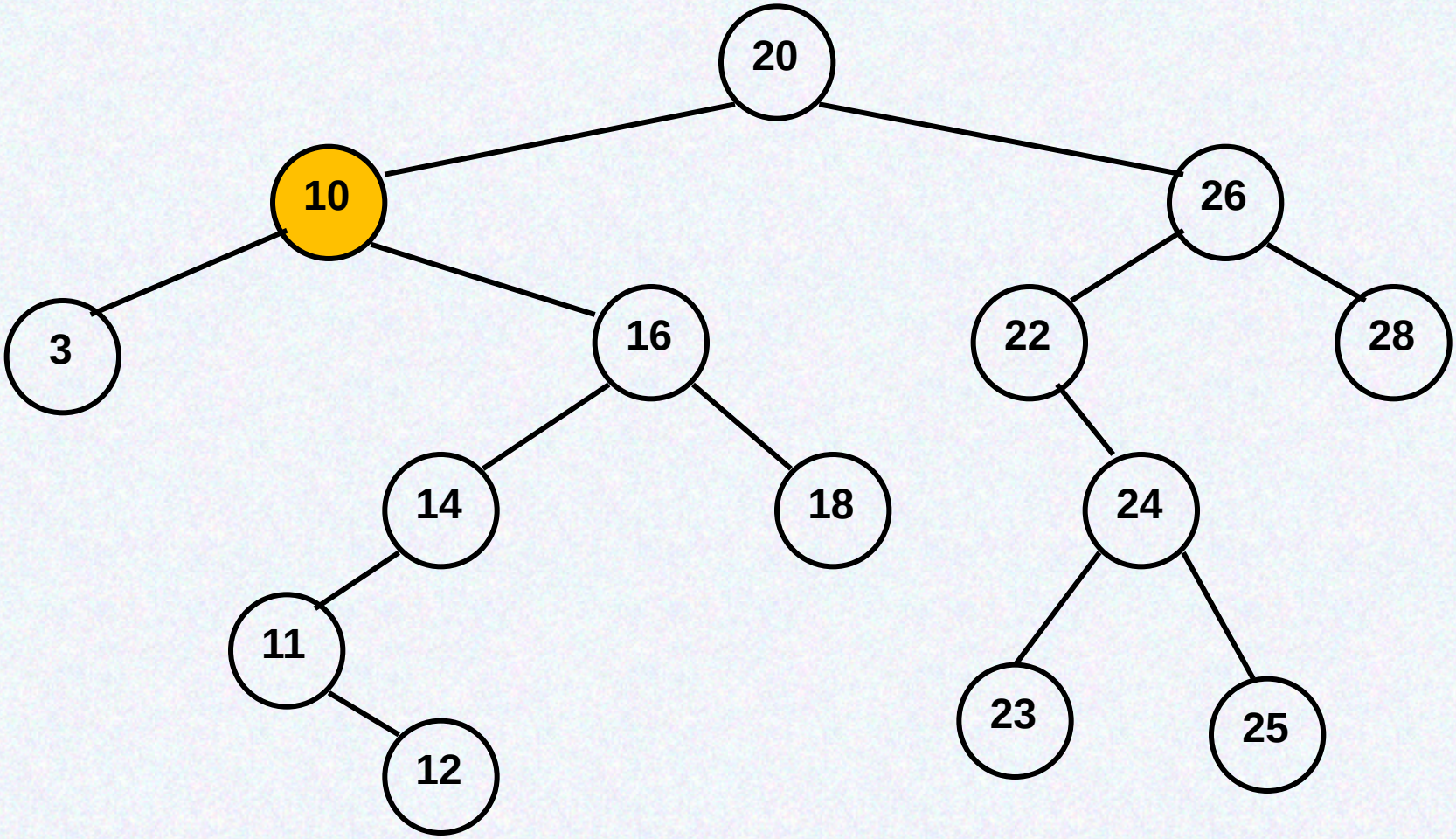
Удалить 22

Удаление элемента



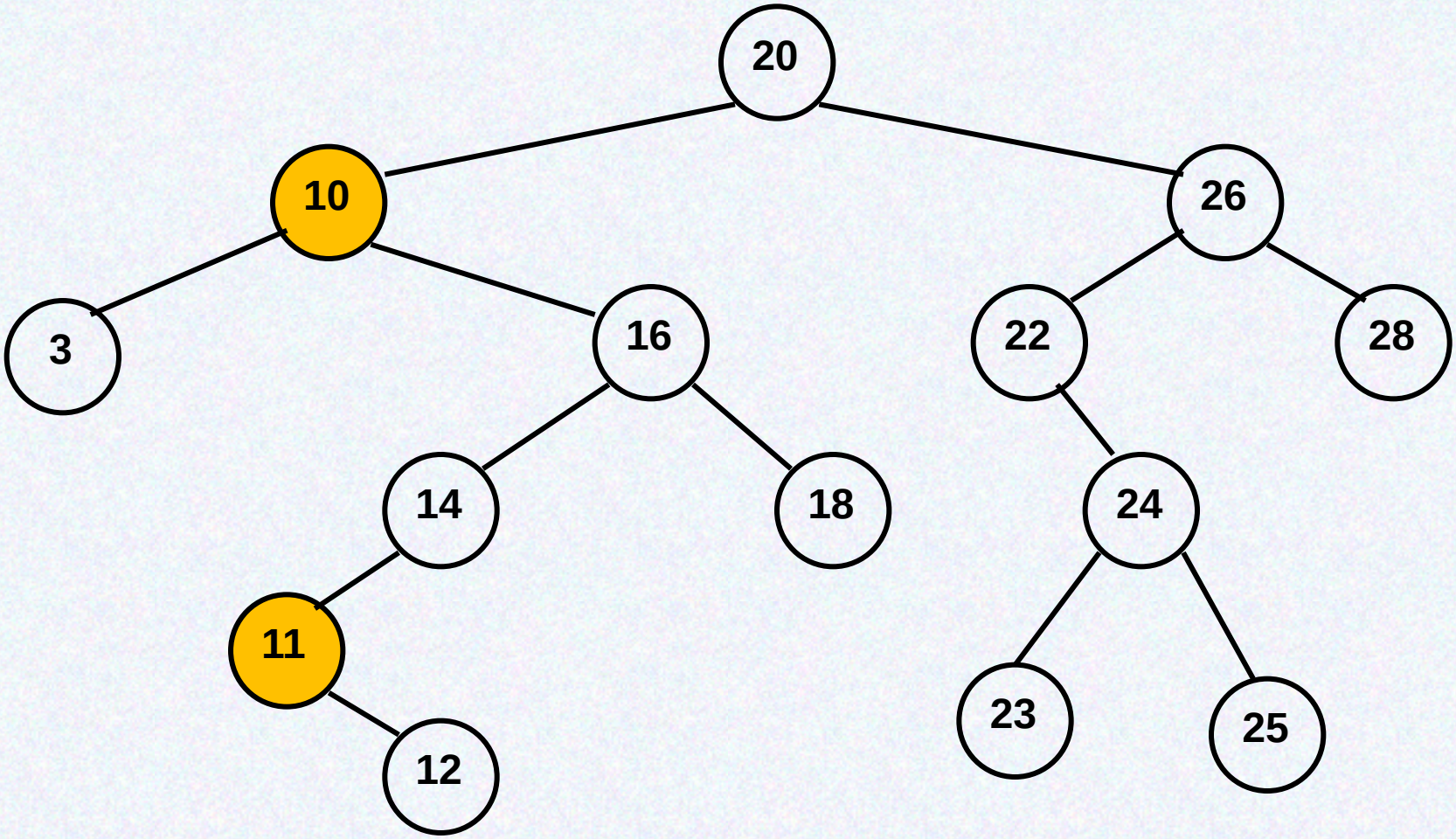
Удалить 10

Удаление элемента



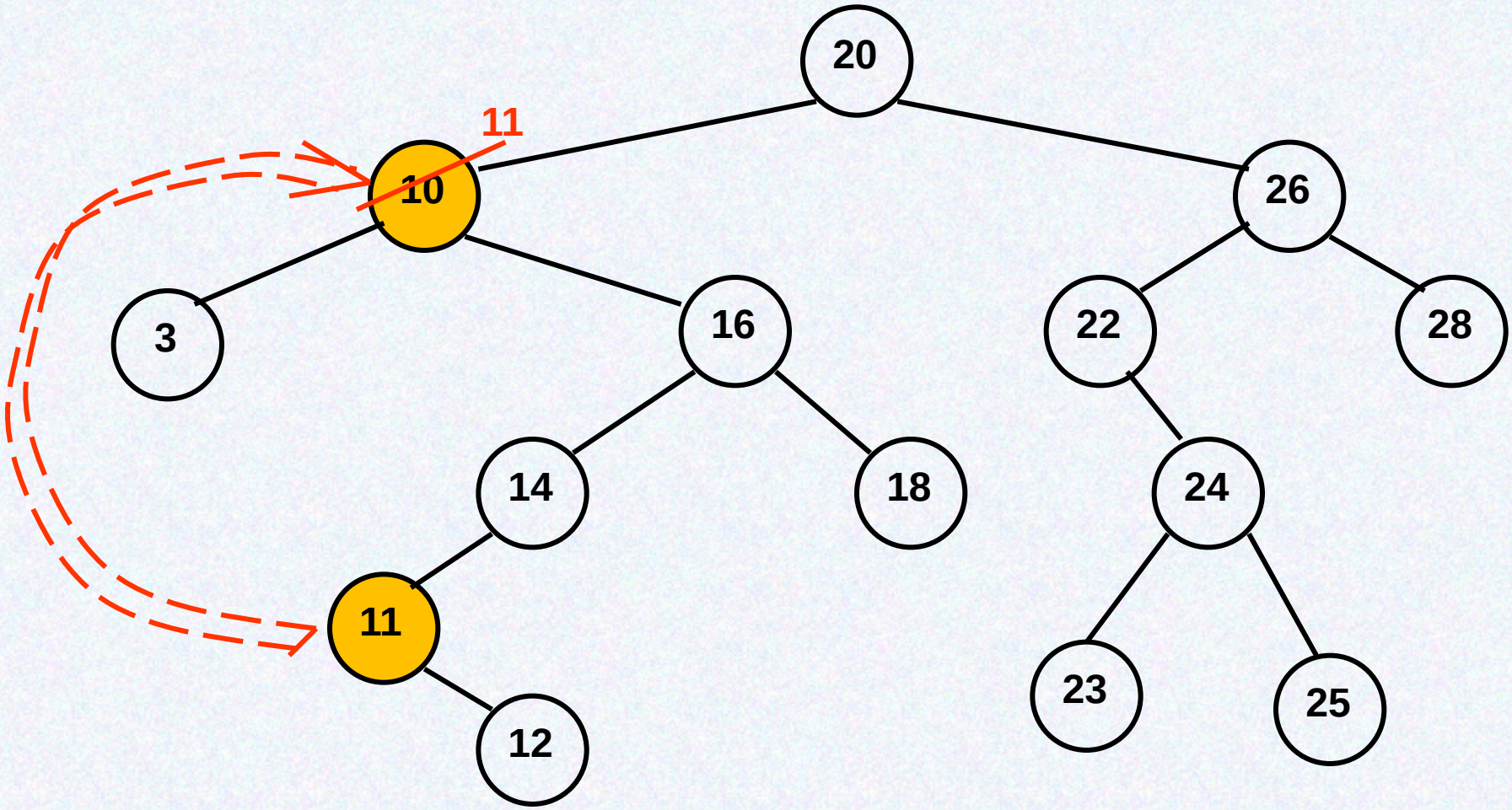
Удалить 10

Удаление элемента



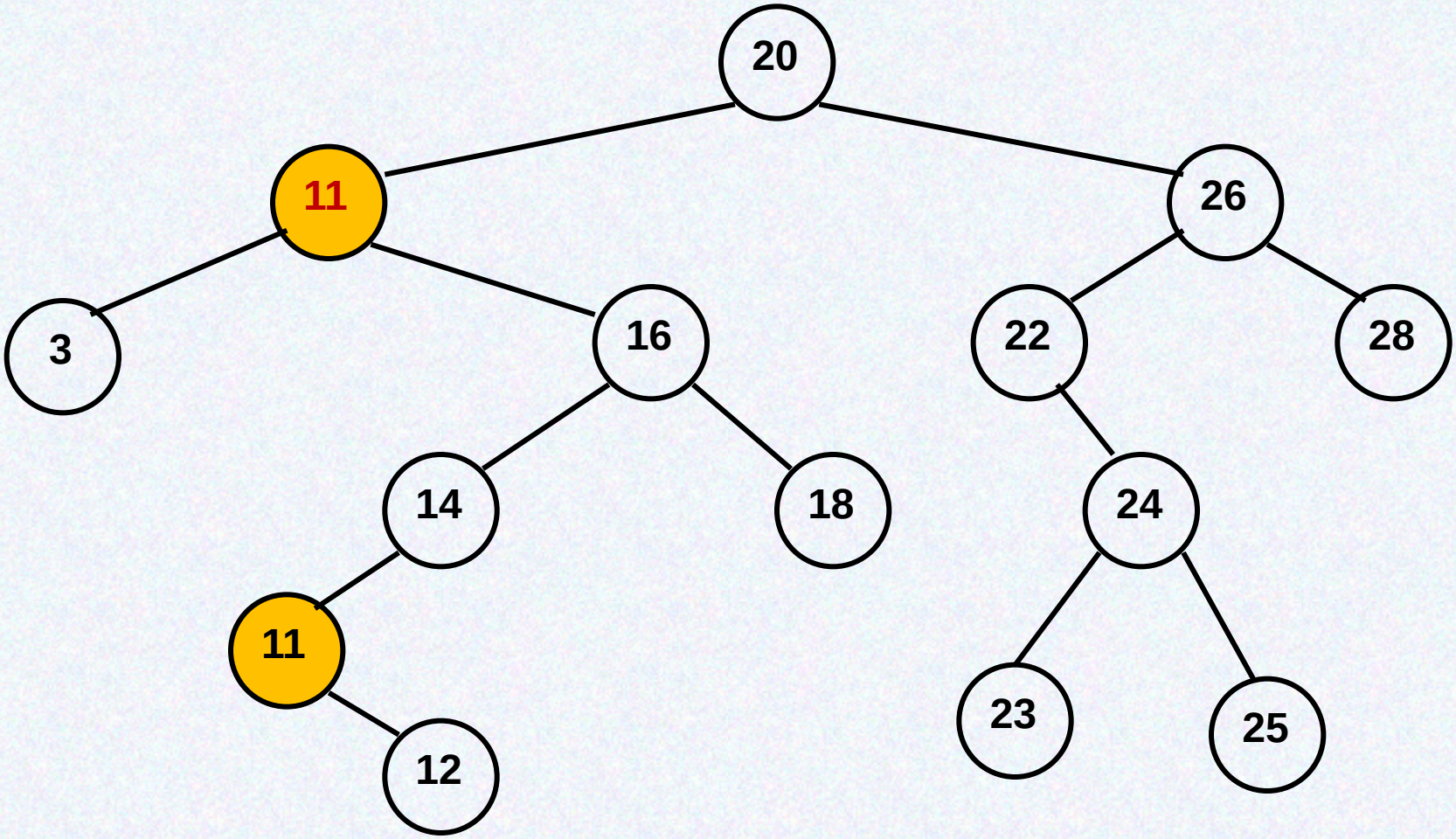
Удалить 10

Удаление элемента



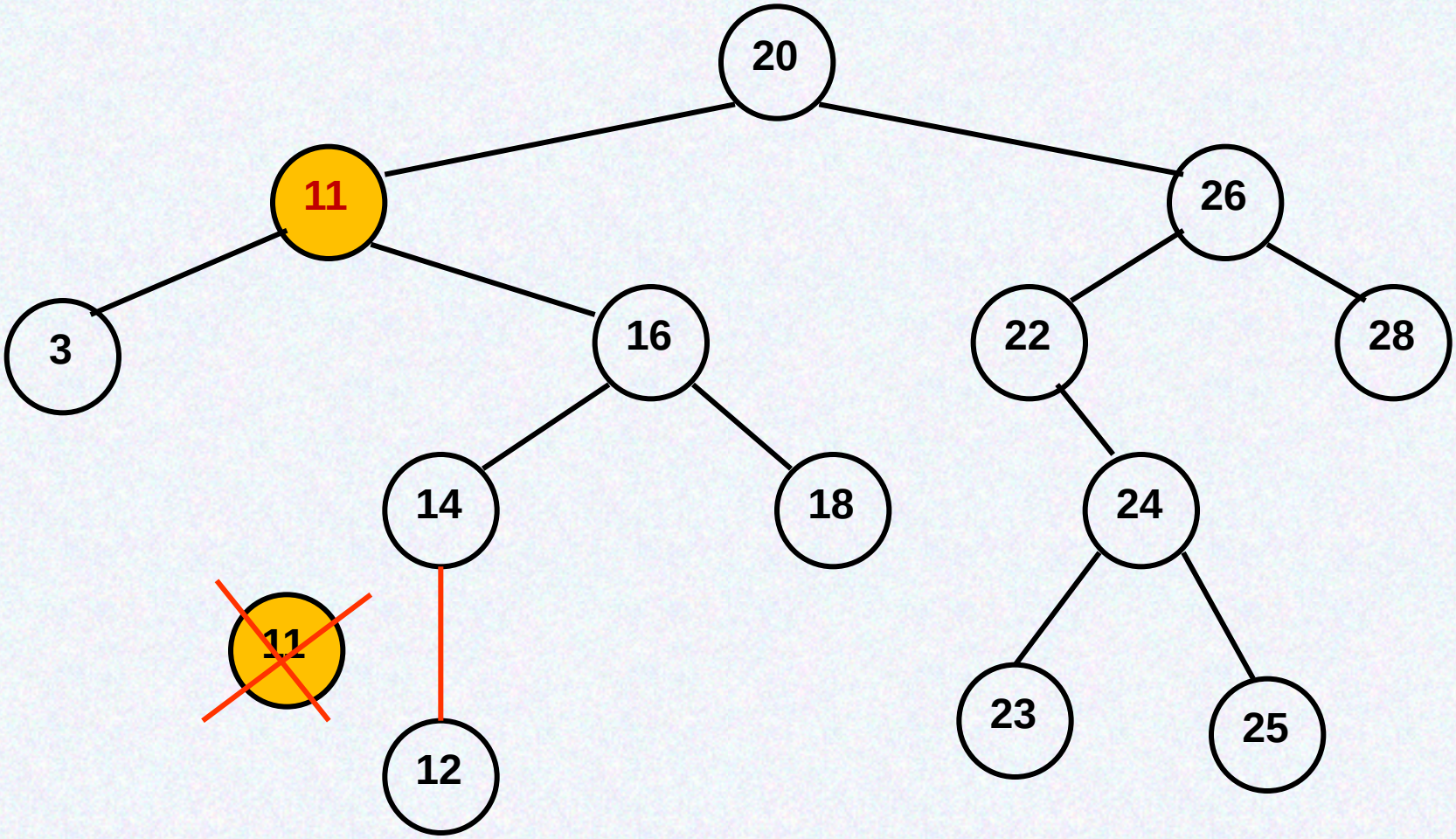
Удалить 10

Удаление элемента



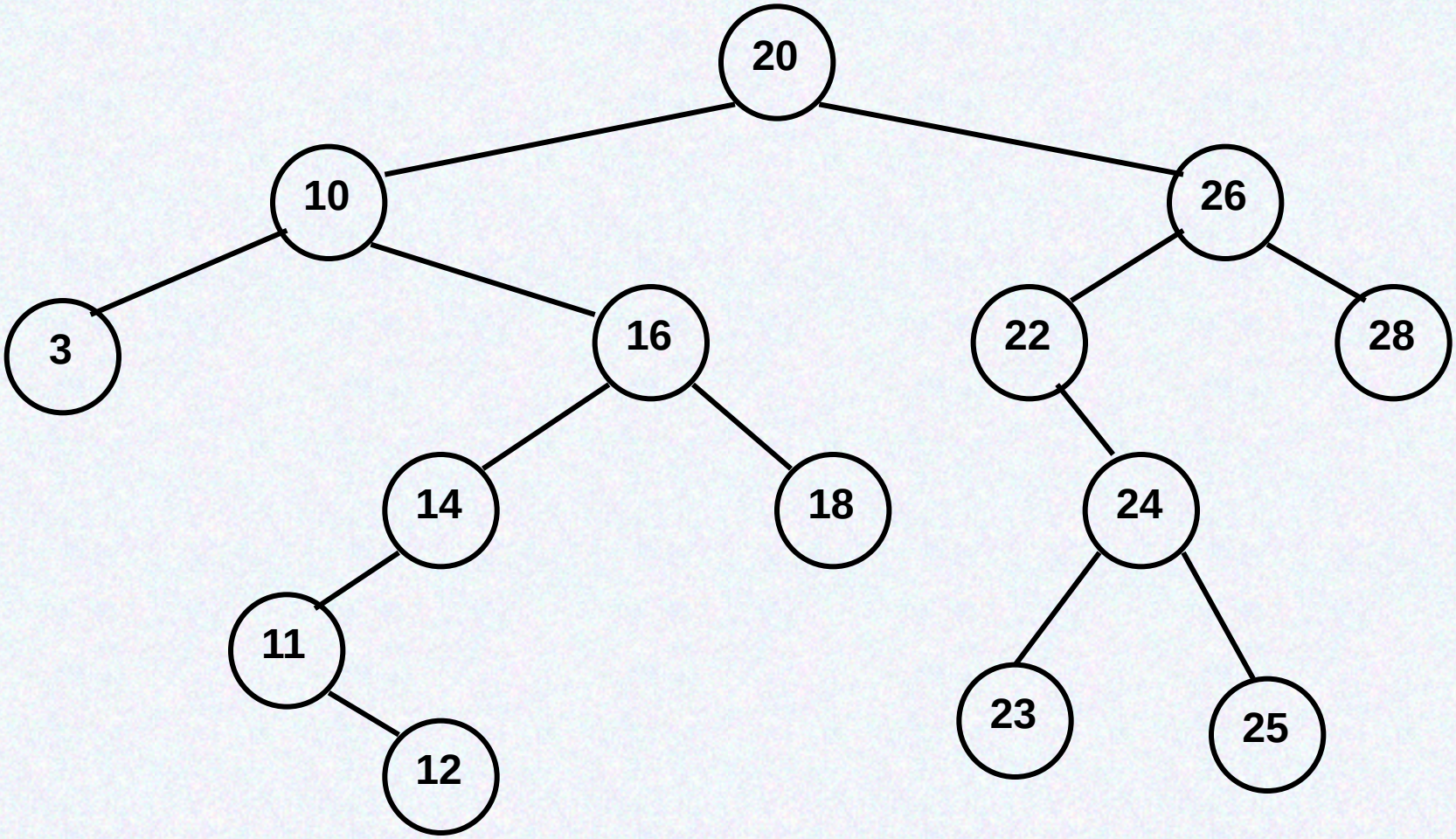
Удалить 10

Удаление элемента



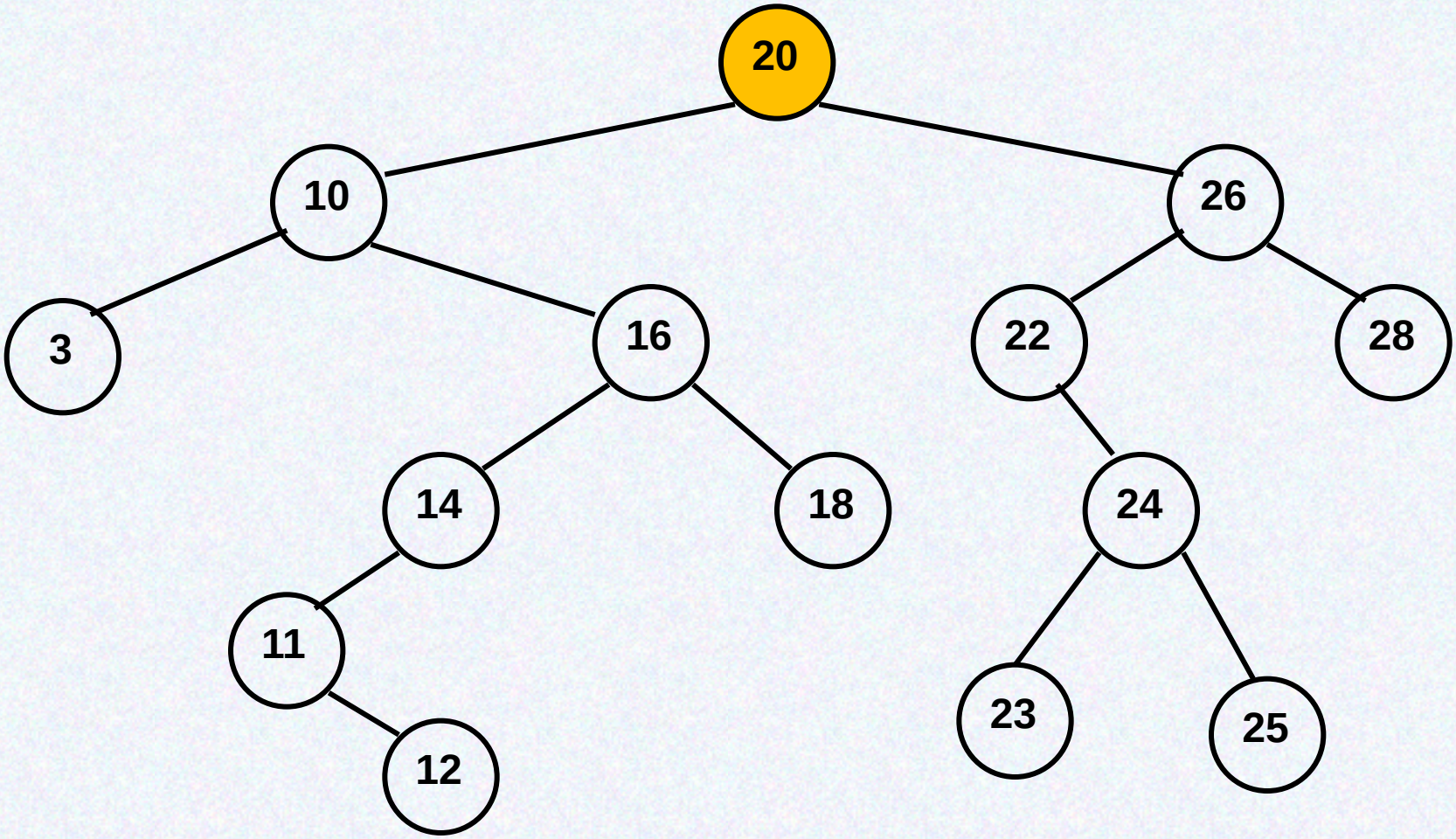
Удалить 10

Удаление элемента



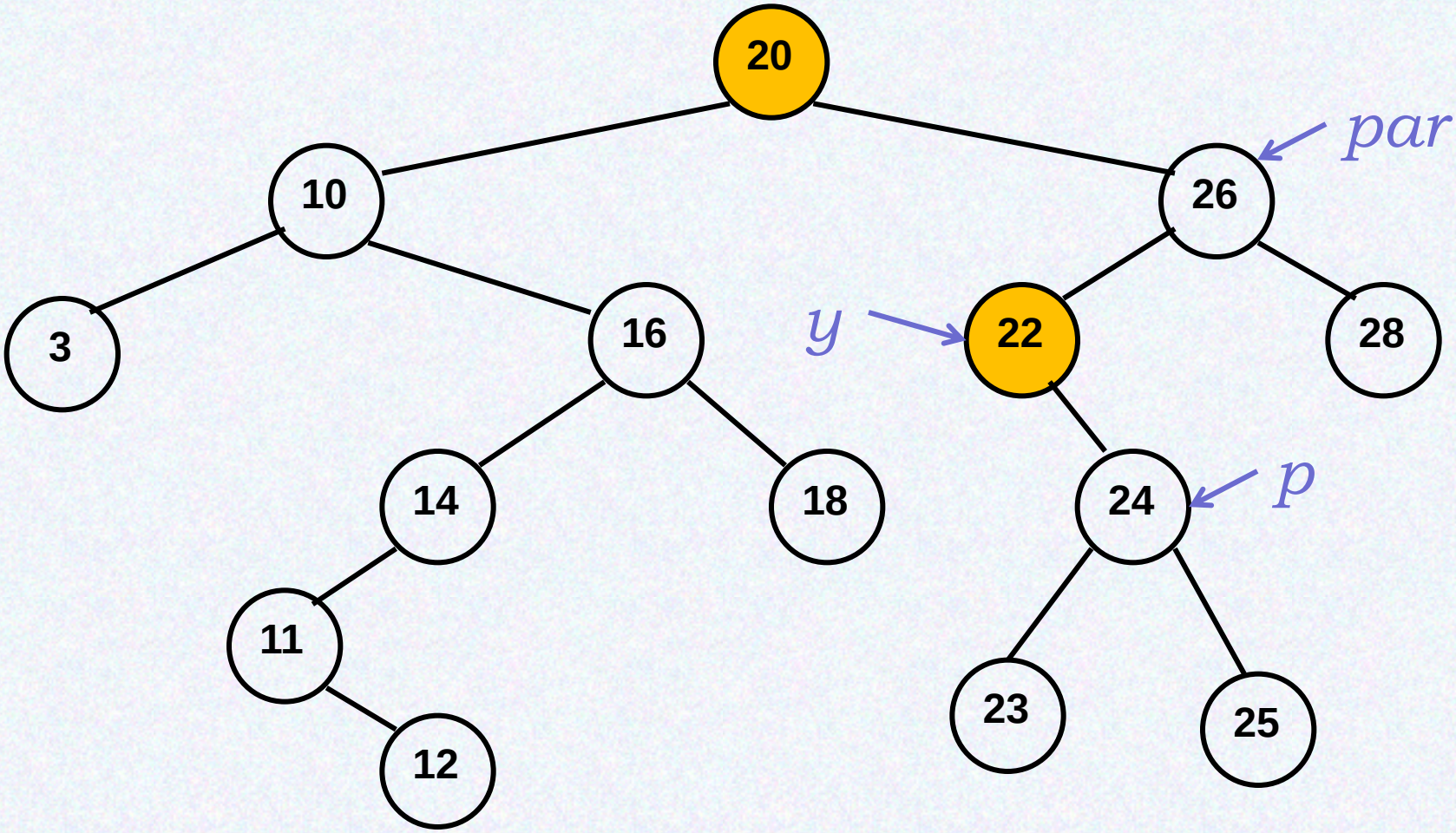
Удалить 20

Удаление элемента



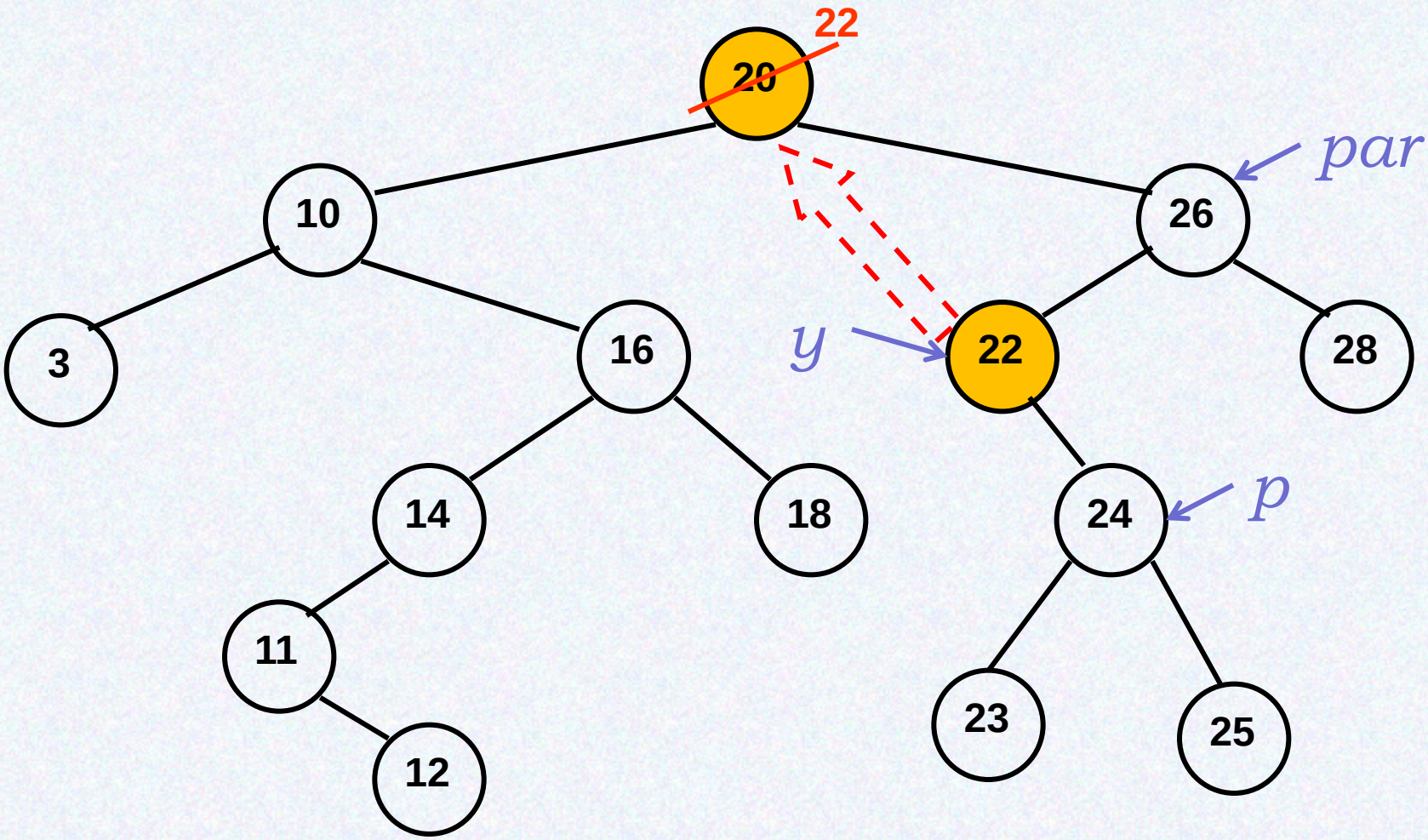
Удалить 20

Удаление элемента



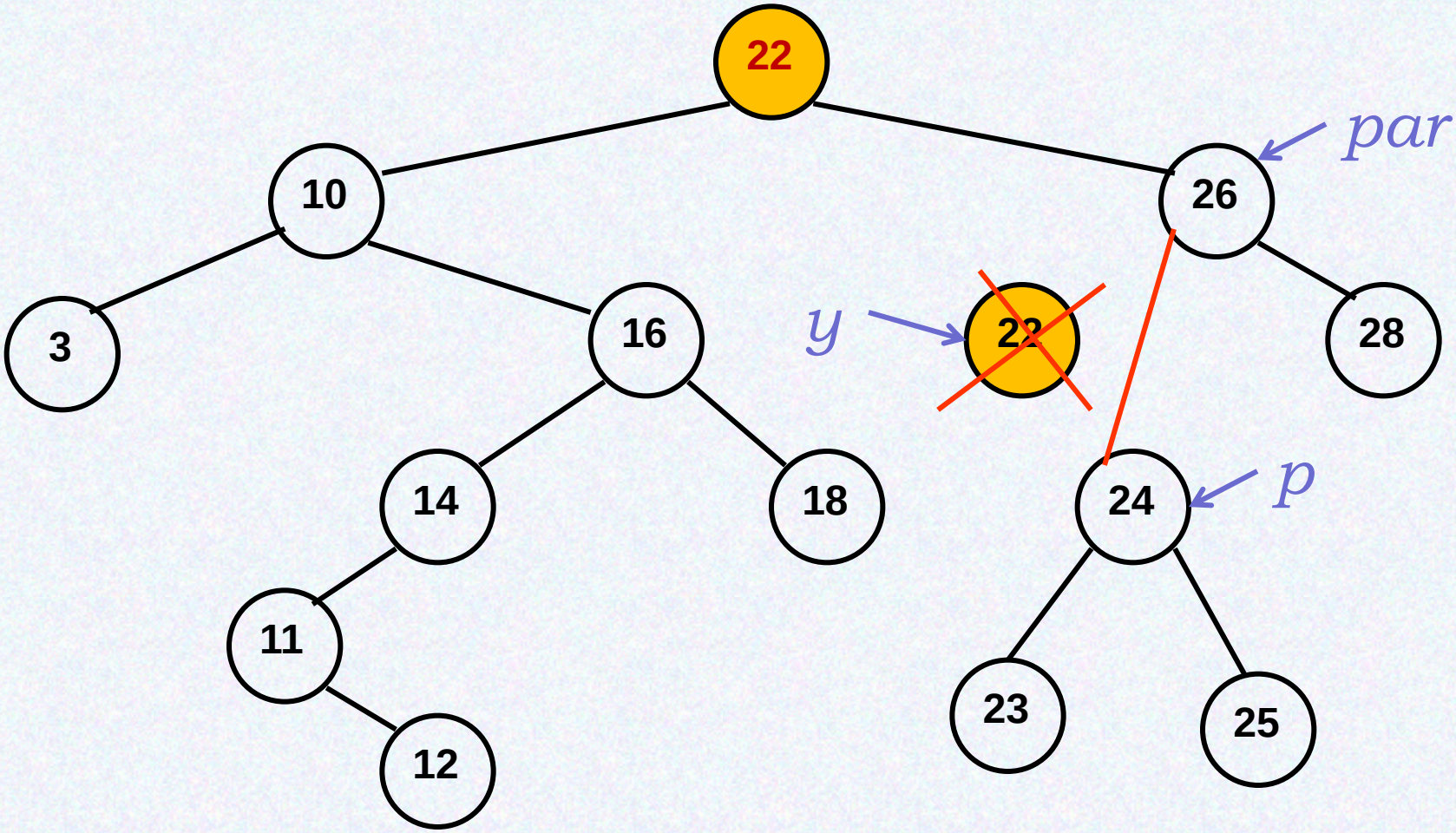
Удалить 20

Удаление элемента



Удалить 20

Удаление элемента



Удалить 20

Удаление элемента

1. Найти реально удаляемый элемент – y

Удаление элемента

1. Найти реально удаляемый элемент – y
2. Найти поддерево удаляемого элемента – p; родительский узел для узла y – par;

Удаление элемента

1. Найти реально удаляемый элемент – y
2. Найти поддерево удаляемого элемента – p; родительский узел для узла y – par;
3. Переопределить родительский узел для поддерева p

Удаление элемента

1. Найти реально удаляемый элемент – y
2. Найти поддерево удаляемого элемента – p; родительский узел для узла y – par;
3. Переопределить родительский узел для поддерева p
4. Проверить случай, что y – корень дерева

Удаление элемента

1. Найти реально удаляемый элемент – y
2. Найти поддерево удаляемого элемента – p; родительский узел для узла y – par;
3. Переопределить родительский узел для поддерева p
4. Проверить случай, что y – корень дерева
5. Переопределить дочерний узел для узла par

Удаление элемента

1. Найти реально удаляемый элемент – y
2. Найти поддерево удаляемого элемента – p; родительский узел для узла y – par;
3. Переопределить родительский узел для поддерева p
4. Проверить случай, что y – корень дерева
5. Переопределить дочерний узел для узла par
6. Скопировать необходимые данные

Удаление элемента

Вершина дерева

```
struct Node {
   int key;
   Node *left, *right, *parent;
};
```

Удаление элемента

Вершина дерева

```
struct Node {
   int key;
   Node *left, *right, *parent;
};
```

x – указатель на удаляемый элемент

Удаление элемента

Вершина дерева

```
struct Node {
   int key;
   Node *left, *right, *parent;
};
```

x – указатель на удаляемый элемент

root – указатель на корень дерева

Удаление элемента

1. Находим реально удаляемый элемент y:

Удаление элемента

1. Находим реально удаляемый элемент y:

if левое или правое поддерево узла x пусто

$y = x$

else

$y =$ указатель на элемент, следующий за x

Удаление элемента

2. У удаляемого элемента может быть только одно поддерево – находим его

Удаление элемента

2. У удаляемого элемента может быть только одно поддерево – находим его

if y имеет левое поддерево

$p =$ левое поддерево y

else

$p =$ правое поддерево y

Удаление элемента

2. У удаляемого элемента может быть только одно поддерево – находим его

if *y* имеет левое поддерево

p = левое поддерево *y*

else

p = правое поддерево *y*

Родительский узел *par = y->parent*

Удаление элемента

3. Если поддерево есть – для его корня необходимо переустановить связь с родительской вершиной

Удаление элемента

3. Если поддерево есть – для его корня необходимо переустановить связь с родительской вершиной

if $p \neq NULL$

p->parent = par

Удаление элемента

3. Если поддерево есть – для его корня необходимо переустановить связь с родительской вершиной

if $p \neq NULL$

$p\text{->}parent = par$

4. Возможно, удаляется корень дерева

Удаление элемента

3. Если поддерево есть – для его корня необходимо переустановить связь с родительской вершиной

```
if p ≠ NULL
   p->parent = par
```

4. Возможно, удаляется корень дерева

```
if par == NULL
   root = p
else
```

Удаление элемента

5. Удаляется промежуточная вершина дерева. Переустанавливаем соответствующее поддерево для родительской вершины

Удаление элемента

5. Удаляется промежуточная вершина дерева. Переустанавливаем соответствующее поддерево для родительской вершины

```
if par->left == y
   par->left = p
else
   par->right = p
```

Удаление элемента

6. Копируем необходимые данные

Удаление элемента

6. Копируем необходимые данные

if $y \neq x$ {

$x\text{->}key = y\text{->}key$

копирование сопутствующих данных

}

Успех: y

Характеристика вставки и удаления

Операции вставки и удаления в бинарном дереве поиска высоты h могут быть выполнены за время $O(h)$

В наихудшем случае $h = n$

Реализация алгоритма удаления на C++ упростится, если использовать двойной указатель на вершину дерева:
*Node **pptr;*