БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ОРГАНИЗАЦИЯ ПОИСКА

БИНАРНЫЕ ПОИСКОВЫЕ ДЕРЕВЬЯ

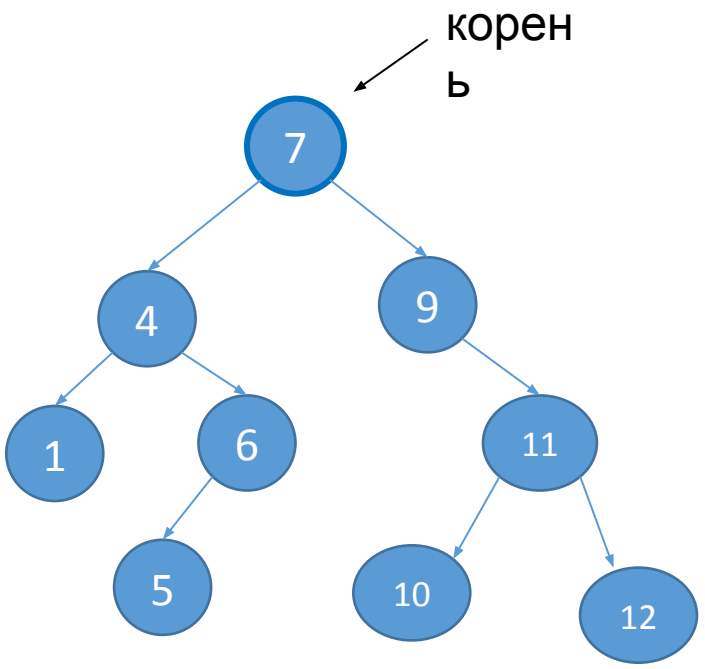
Словарные операции

- поиск элемента с заданным ключом x
- добавление нового элемента с заданным ключом x
- удаление элемента с заданным ключом x

Бинарное поисковое дерево

Корневое дерево

1. Ориентированный граф, в существует ровно одна вершина без входящих дуг (корень).
2. В каждую вершину, за исключением корня, входит ровно одна дуга.
3. Из корня дерева существует единственный путь в любую вершину.



Бинарное

4. Каждая вершина содержит не более 2-х сыновей (левый и, возможно, правый).

Поисковое

5. Каждой вершине поставлено в соответствие некоторое целое число - *ключ*. Для каждой вершины ***v*** все ключи в её левом поддереве строго меньше ключа вершины ***v***, а в правом – строго больше.

n – число вершин

m=n-1 – число дуг

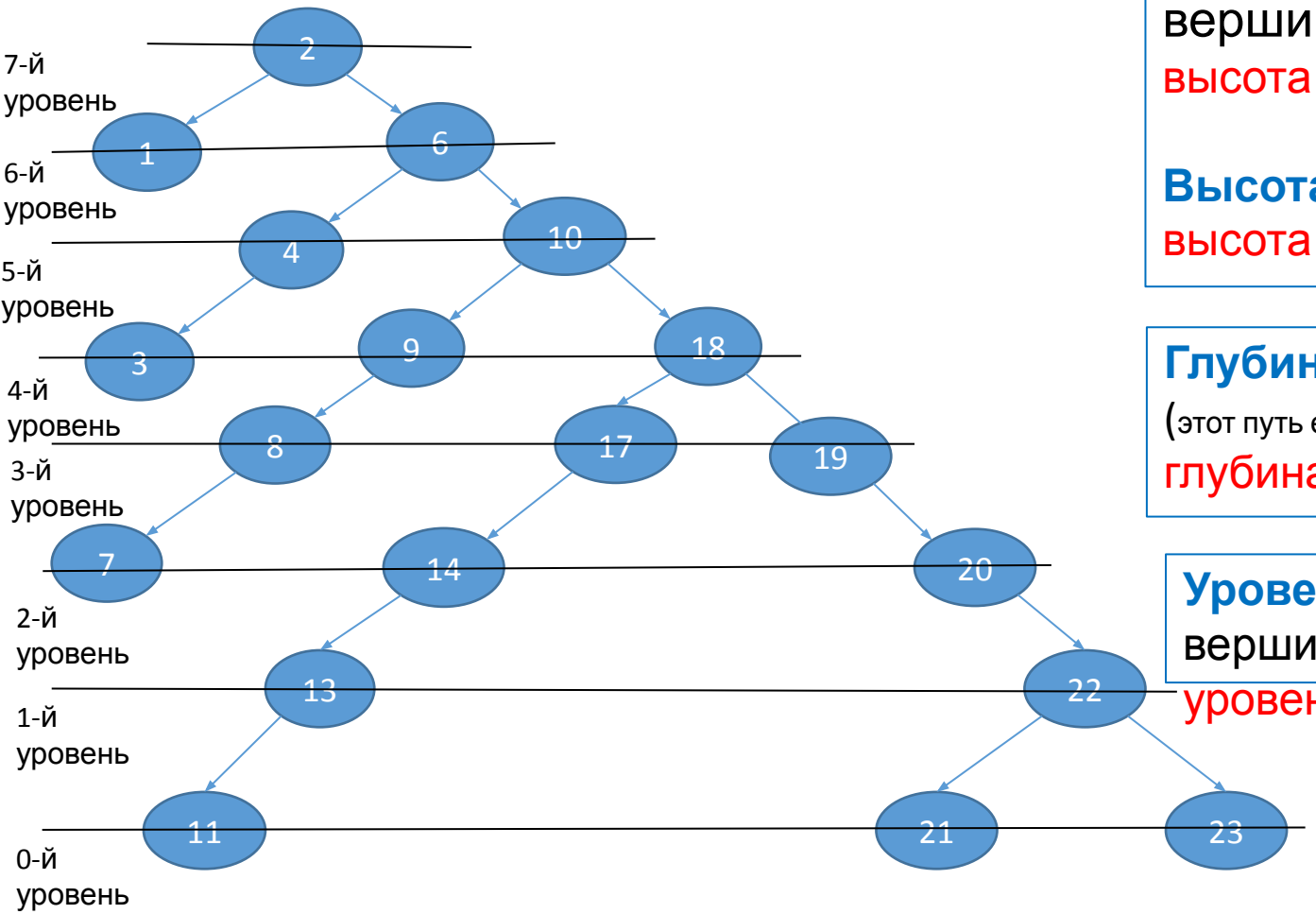
Построить бинарное поисковое дерево для последовательности элементов:

2, 1, 6, 4, 3, 10, 9, 8, 7, 18, 17, 14, 13, 11, 19, 20, 22, 23, 21 **2**

?

так как мы договорились (см. определение) , что в дереве все ключи различны, то одинаковые элементы при добавлении будем игнорировать

Высота, глубина, уровень



Высота вершины: длина наибольшего пути от вершины до одного из её потомков.

высота (10) = **5**

Высота дерева: высота корня.

высота (дерева) = **7**

Глубина вершины: длина пути из корня в вершину (этот путь единственный).

глубина (10) = **2**

Уровень вершины: высота дерева минус глубина вершины

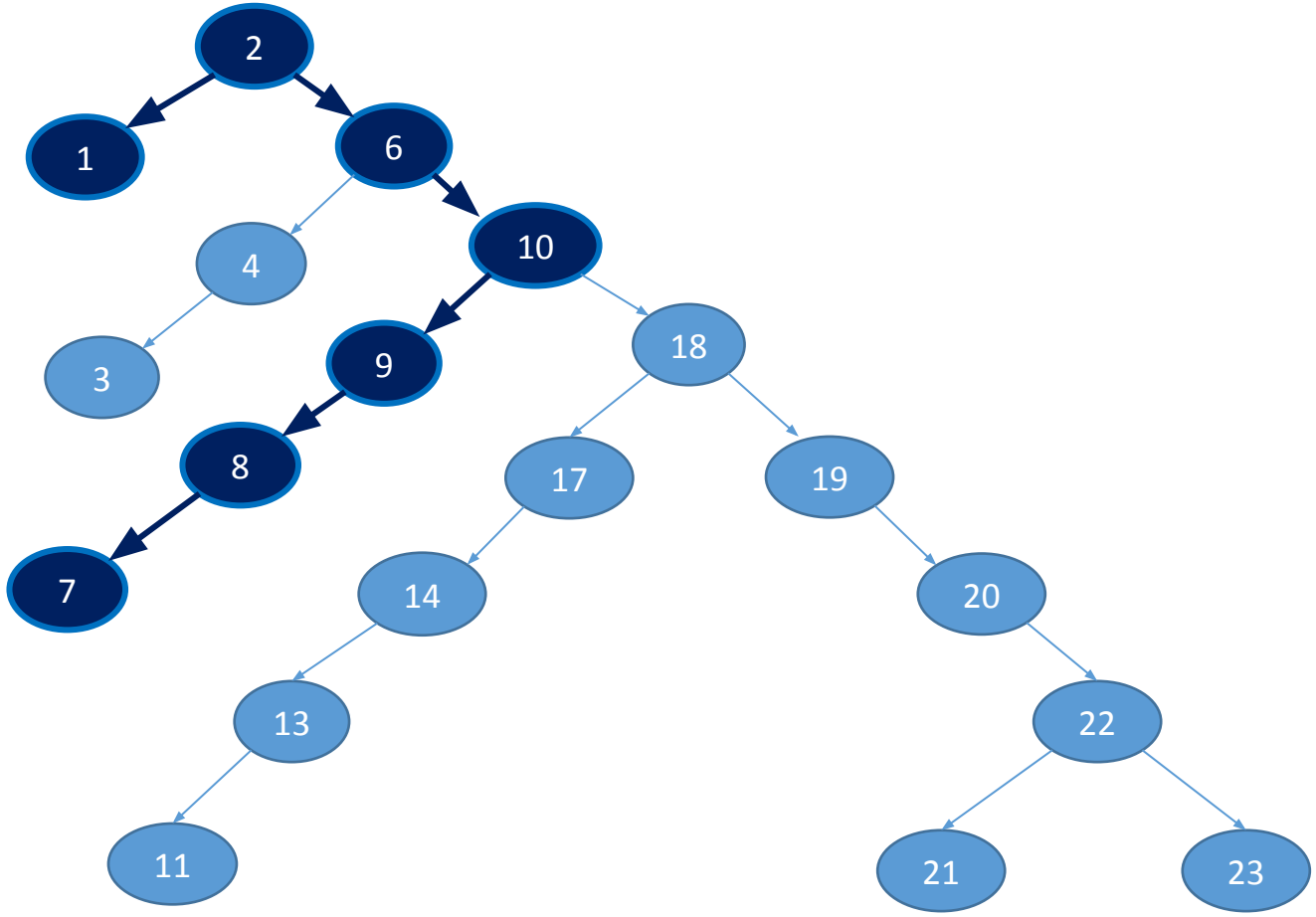
уровень (10) = высота (дерева)- глубина (10) =7 – 2 = **5**

ФПМИ БГУ

Путь, полупуть

Для полупути снимается ограничение на направление дуг.

Пример полупути, соединяющего вершины 1 и 7: **1 - 2 - 6 - 10 - 9 - 8 - 7**



Центральная и средняя вершины полупути

Центральная вершина полупути
такая вершина, что количество вершин в полупути до неё равно количеству вершин после неё

Средняя по значению (медиана) вершина полупути такая вершина, что у половины из оставшихся вершин этого полупути ключ меньше, а у половины – больше

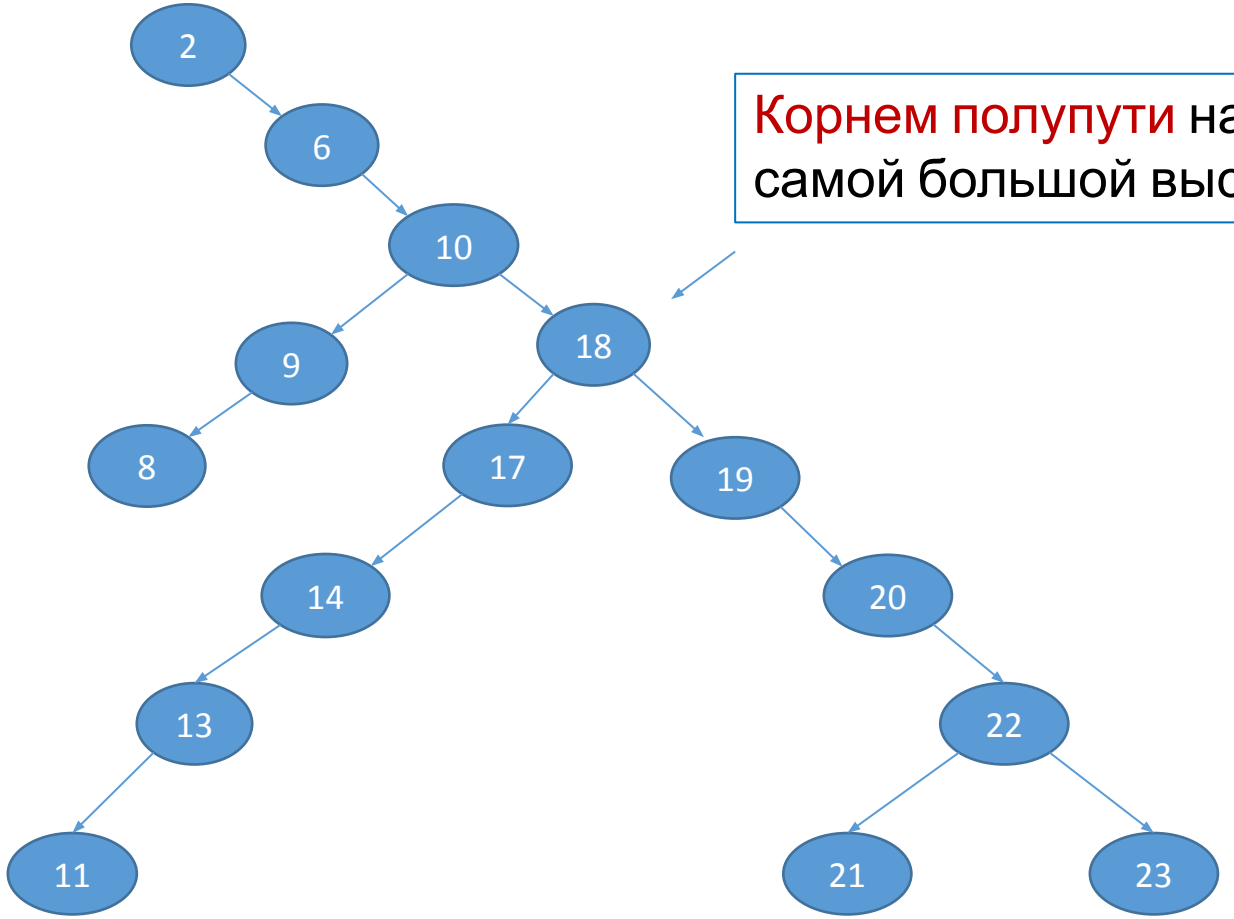
?

Что делать, если число вершин, среди которых надо найти центральную или среднюю ЧЁТНО?

↓

центральной и средней вершины
НЕ СУЩЕСТВУЕТ

Наибольшим полупутём в дереве будем называть полупуть наибольшей длины.



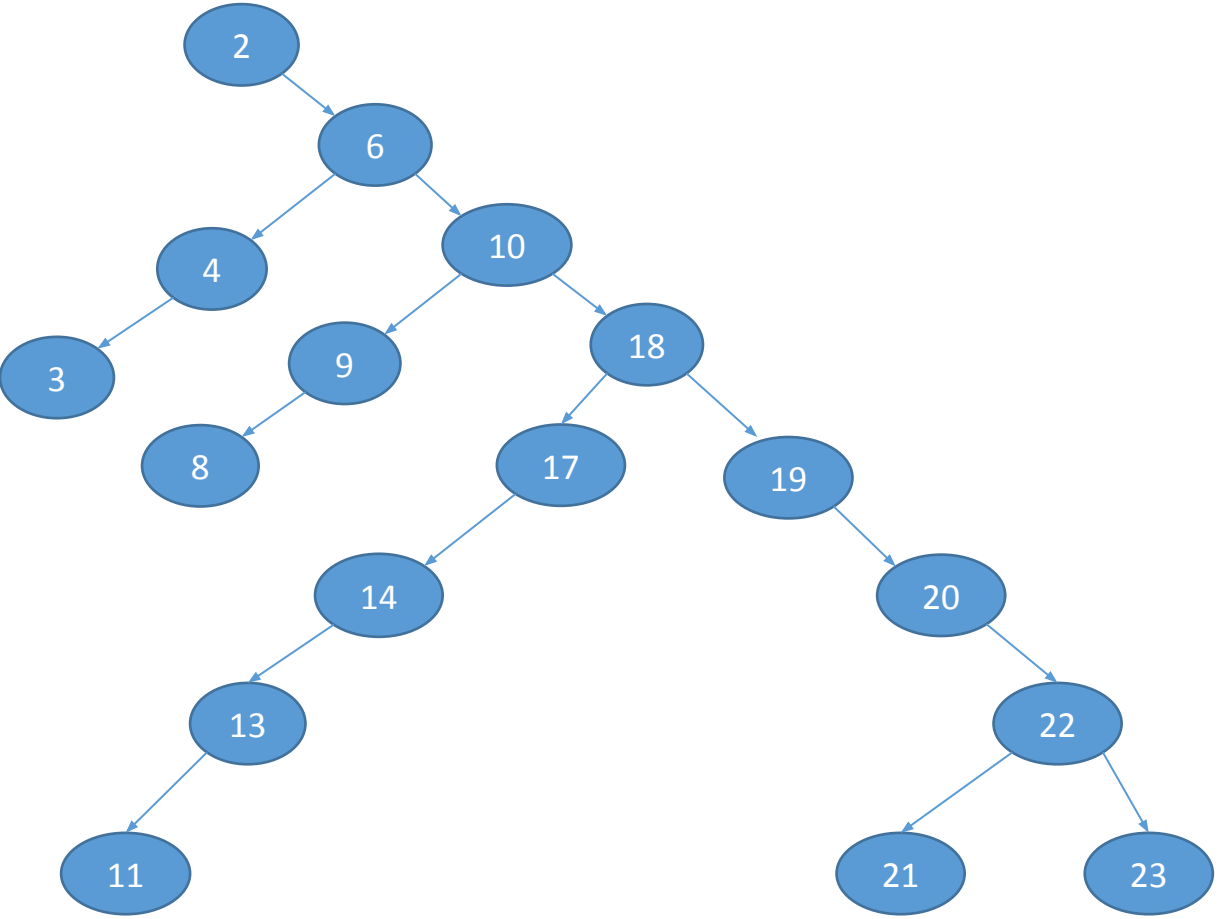
Корнем полупути назовём вершину полупути самой большой высотой.

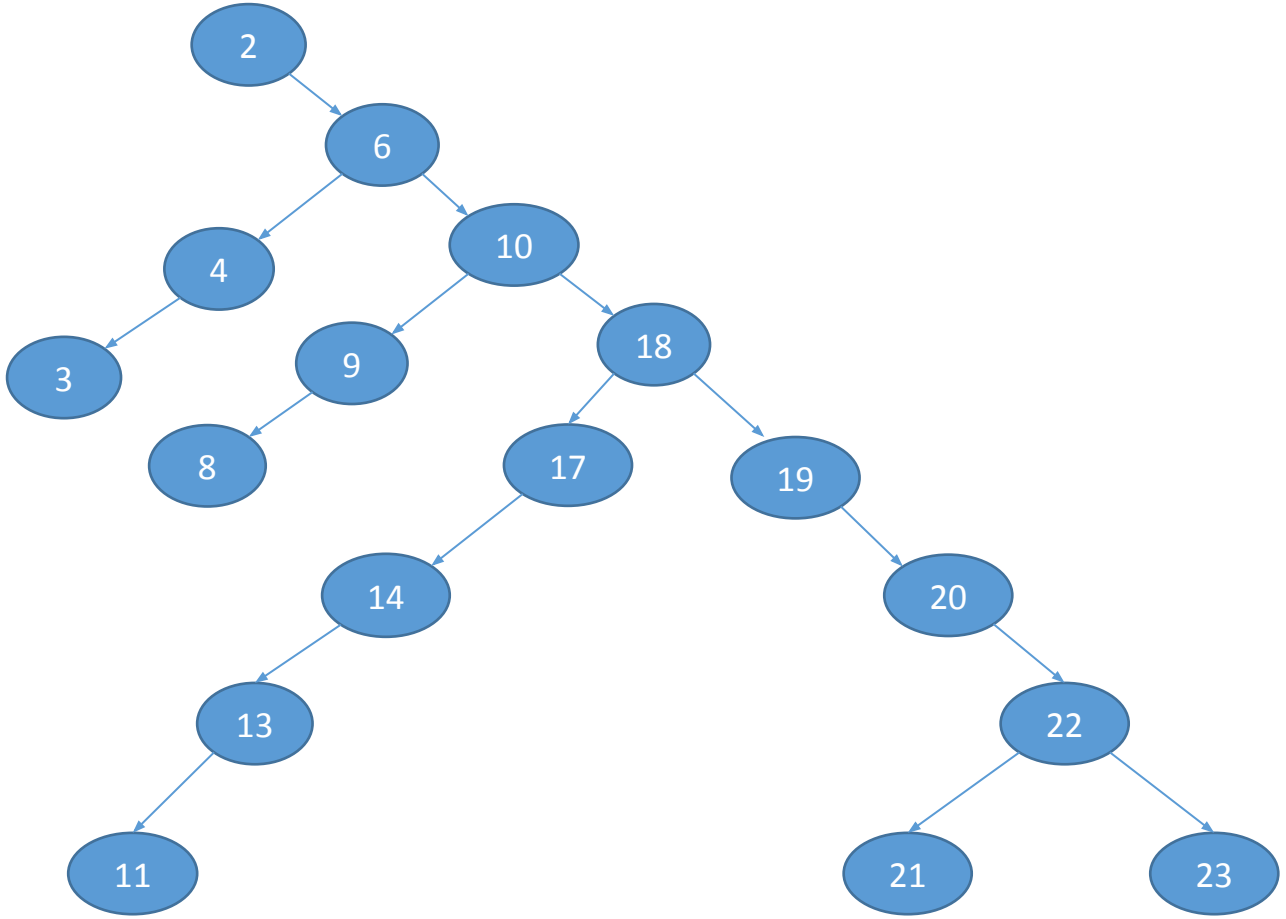
Пример.

1) Длина наибольшего полупути = **8.**

2) Вершины **18** и **6** являются корнями полупутей наибольшей длины.

3) Через вершину **18** проходить **5** попарно различных полупутей наибольшей длины.





Обходы

прямой (левый, правый) - **PreOrderTraversal** (v)

обратный (левый, правый) - **PostOrderTraversal** (v)

внутренний (левый, правый) - **InOrderTraversal** (v)

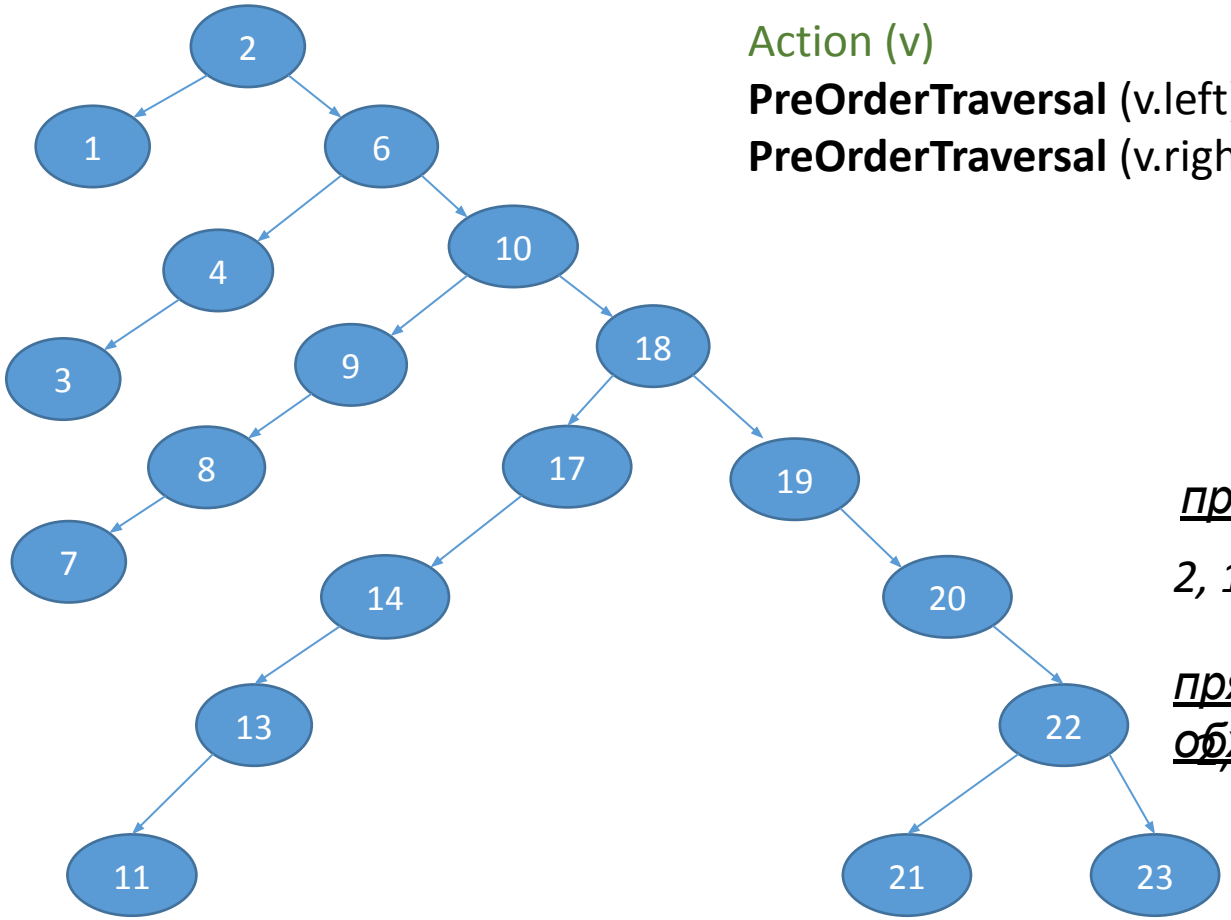
Время выполнения обхода: пропорционально числу вершин в дереве (=**n)**

прямой левый обход

```
def PreOrderTraversal (v):
    if v is not None:
        Action (v)
        PreOrderTraversal (v.left)
        PreOrderTraversal (v.right)
```

прямой правый обход

```
def PreOrderTraversal (v):
    if v is not None:
        Action (v)
        PreOrderTraversal (v.right)
        PreOrderTraversal (v.left)
```



прямой левый обход

2, 1, 6, 4, 3, 10, 9, 8, 7, 18, 17, 14, 13, 11, 19, 20, 22, 21, 23

прямой правый обход

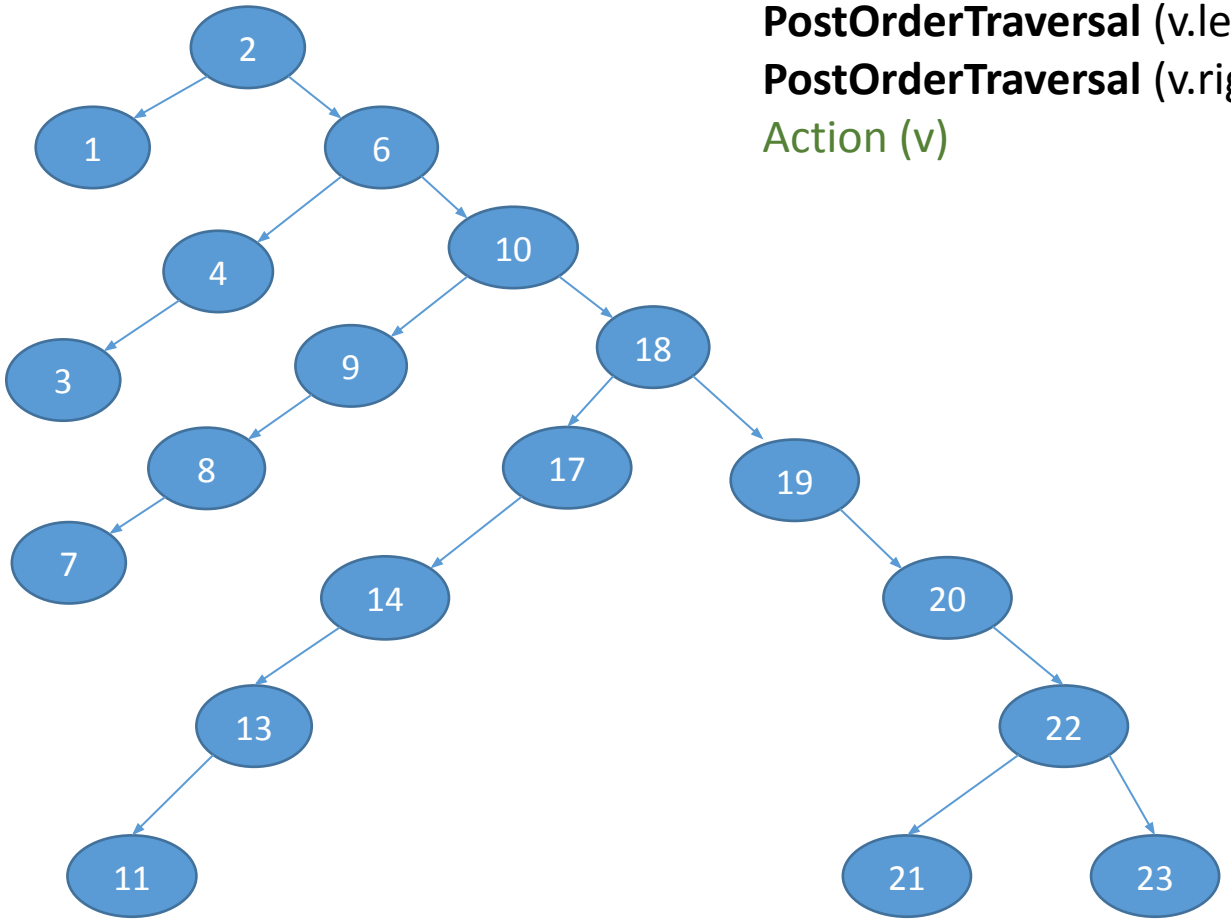
2, 6, 10, 18, 19, 20, 22, 23, 21, 17, 14, 13, 11, 9, 8, 7, 4, 3, 1

обратный левый обход

```
def PostOrderTraversal (v):
    if v is not None:
        PostOrderTraversal (v.left)
        PostOrderTraversal (v.right)
        Action (v)
```

обратный правый обход

```
def PostOrderTraversal (v):
    if v is not None:
        PostOrderTraversal (v.right)
        PostOrderTraversal (v.left)
        Action (v)
```



обратный левый обход

1 ,3, 4, 7, 8, 9, 11, 13, 14, 17, 21, 23, 22, 20, 19, 18, 10, 6, 2

обратный правый обход

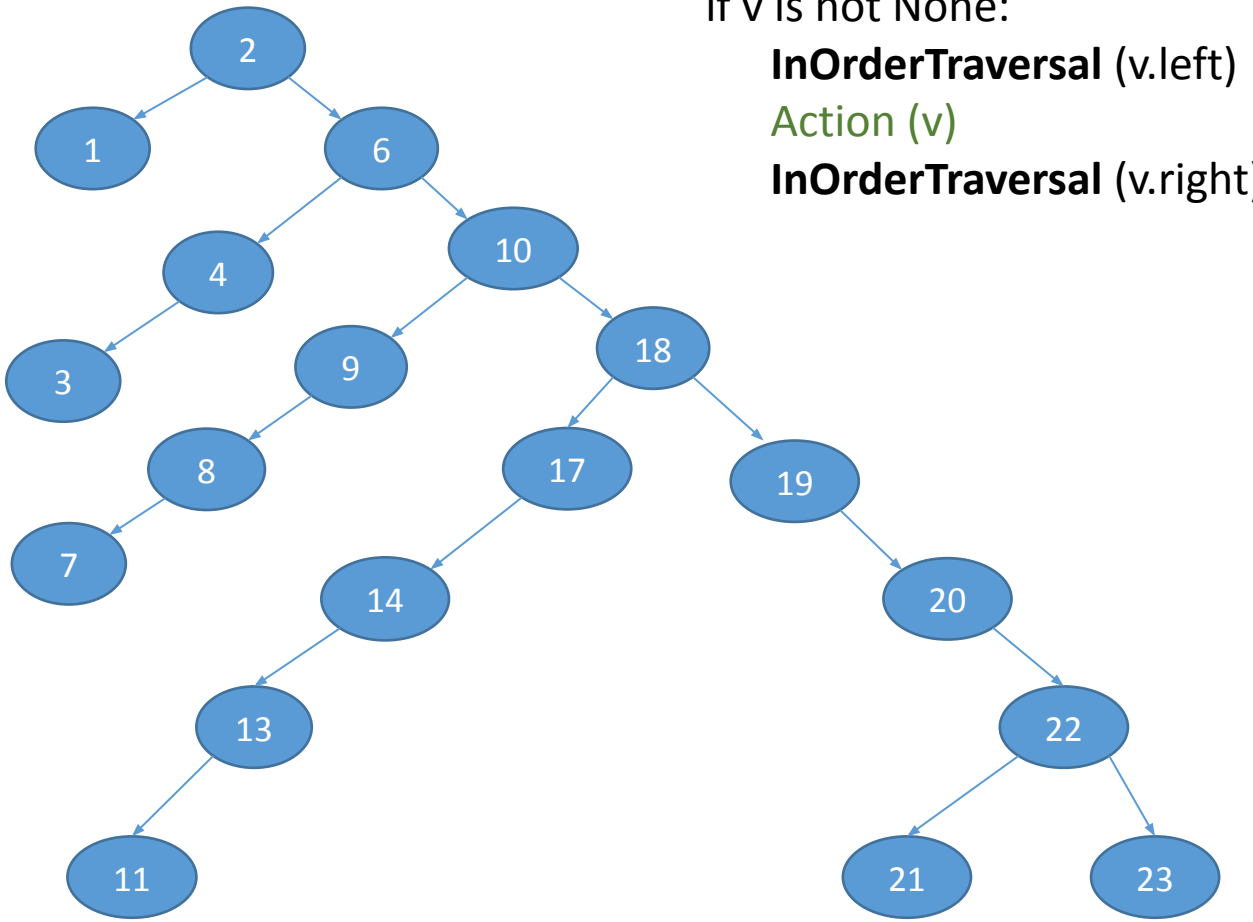
23, 21, 22, 20, 19, 11, 13, 14, 17, 18, 7, 8, 9, 10, 3, 4, 6, 1, 2

внутренний левый обход

```
def InOrderTraversal (v):
    if v is not None:
        InOrderTraversal (v.left)
        Action (v)
        InOrderTraversal (v.right)
```

внутренний правый обход

```
def InOrderTraversal (v):
    if v is not None:
        InTraversal (v.right)
        Action (v)
        InOrderTraversal (v.left)
```



внутренний левый обход

(ключи отсортированы по возрастанию)

1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13, 14, 17, 18, 19, 20, 21, 22, 23

внутренний правый обход

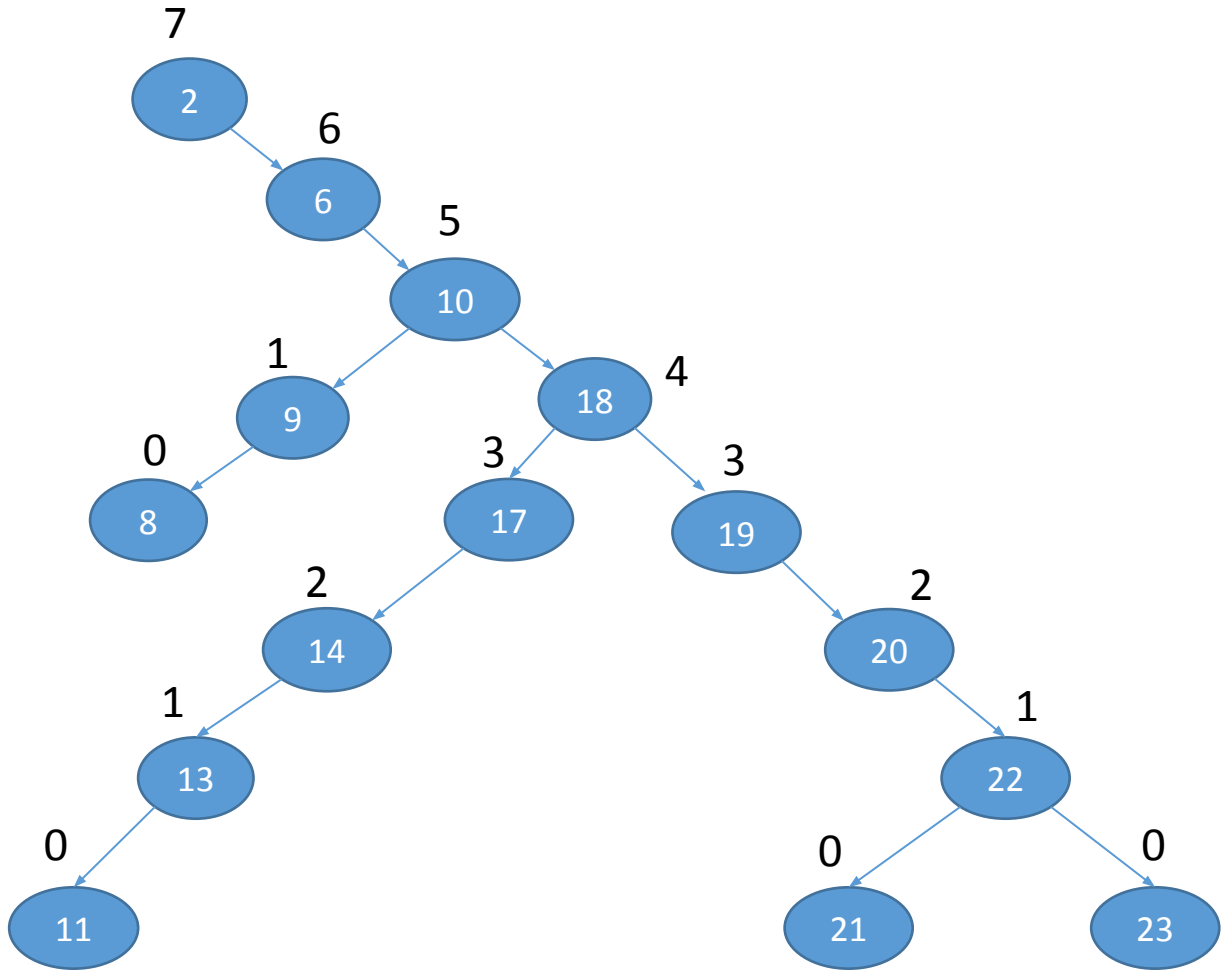
(ключи отсортированы по убыванию)

23, 22, 21, 20, 19, 18, 17, 14, 13, 11, 10, 9, 8, 7, 6, 4, 3, 2, 1

Примеры задач

1) Найти высоту дерева.
2) Определить, является ли дерево сбалансированнным по высоте?
3) Найти длину наибольшего полупути (корни полупутей наибольшей длины).
4) Проверить, является ли дерево идеально-сбалансированным по числу вершин?
5) Найти среднюю по значению вершину в дереве.
6) Найти среднюю по значению вершину среди вершин, у которых высоты поддеревьев совпадают.
7) Найти среднюю по значению вершину среди вершин некоторого пути.

1) Найти высоту дерева.
2) Определить, является ли дерево сбалансированнным по высоте (для всех вершин высоты их подеревьев должны отличаться не более, чем на 1)?



Нужно знать высоту каждой вершины.

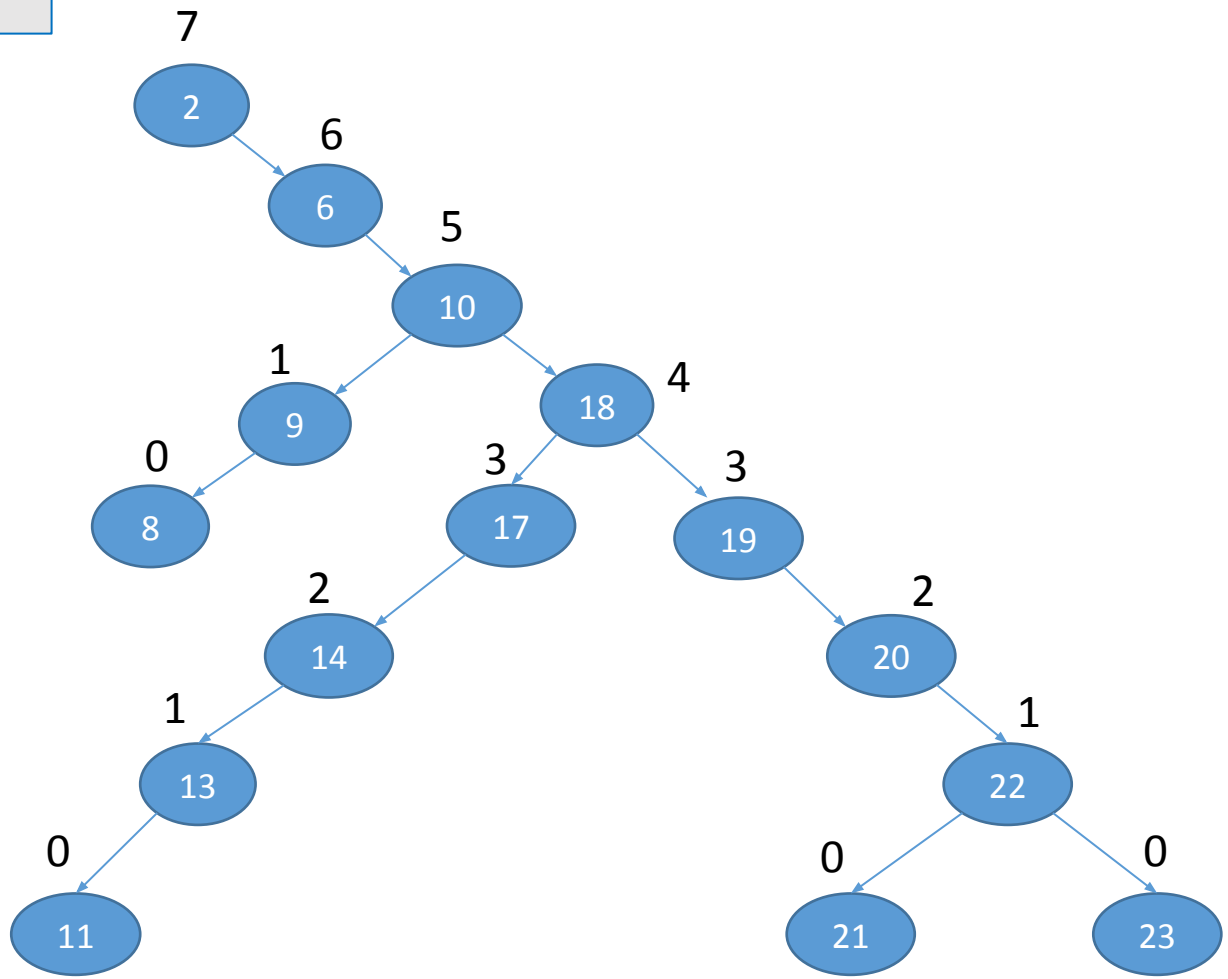
Обратный левый (или правый) **обход**

- если вершина **v** лист, то её высота равна 0;
- если у вершины **v** только одно поддерево, то её высота равна высоте этого поддерева +1;
- если у вершины **v** есть оба поддерева, то её высота равна максимуму из высот поддеревьев, увеличенному на 1.

3) Найти длину наибольшего полупути (корни полупутей наибольшей длины).

Зная метки высот, **можно найти длину наибольшего полупути и его корень**:

> найдём ту вершину **v**, для которой сумма меток высот её поддеревьев, увеличенное на 2 или 1 (в зависимости от того, сколько поддеревьев у вершины **v**), является наибольшей.

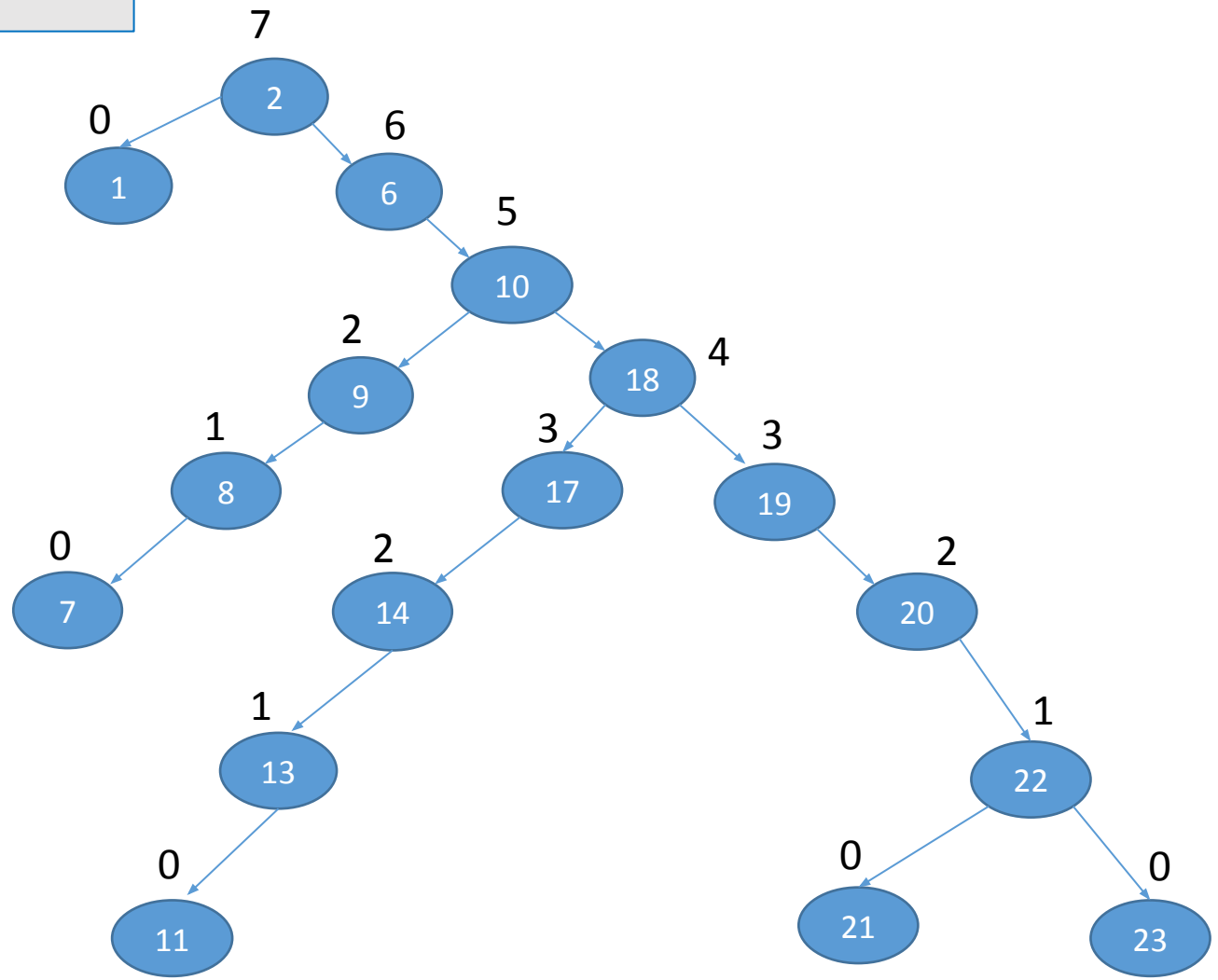


Вершина 18: 3+3+2=8
является корнем наибольшего полупути.

Длина наибольшего полупути равна 8.

3) Найти длину наибольшего полупути (корни полупутей наибольшей длины).

Вершины **2, 10, 18** являются корнями полупутей наибольшей длины **8**.

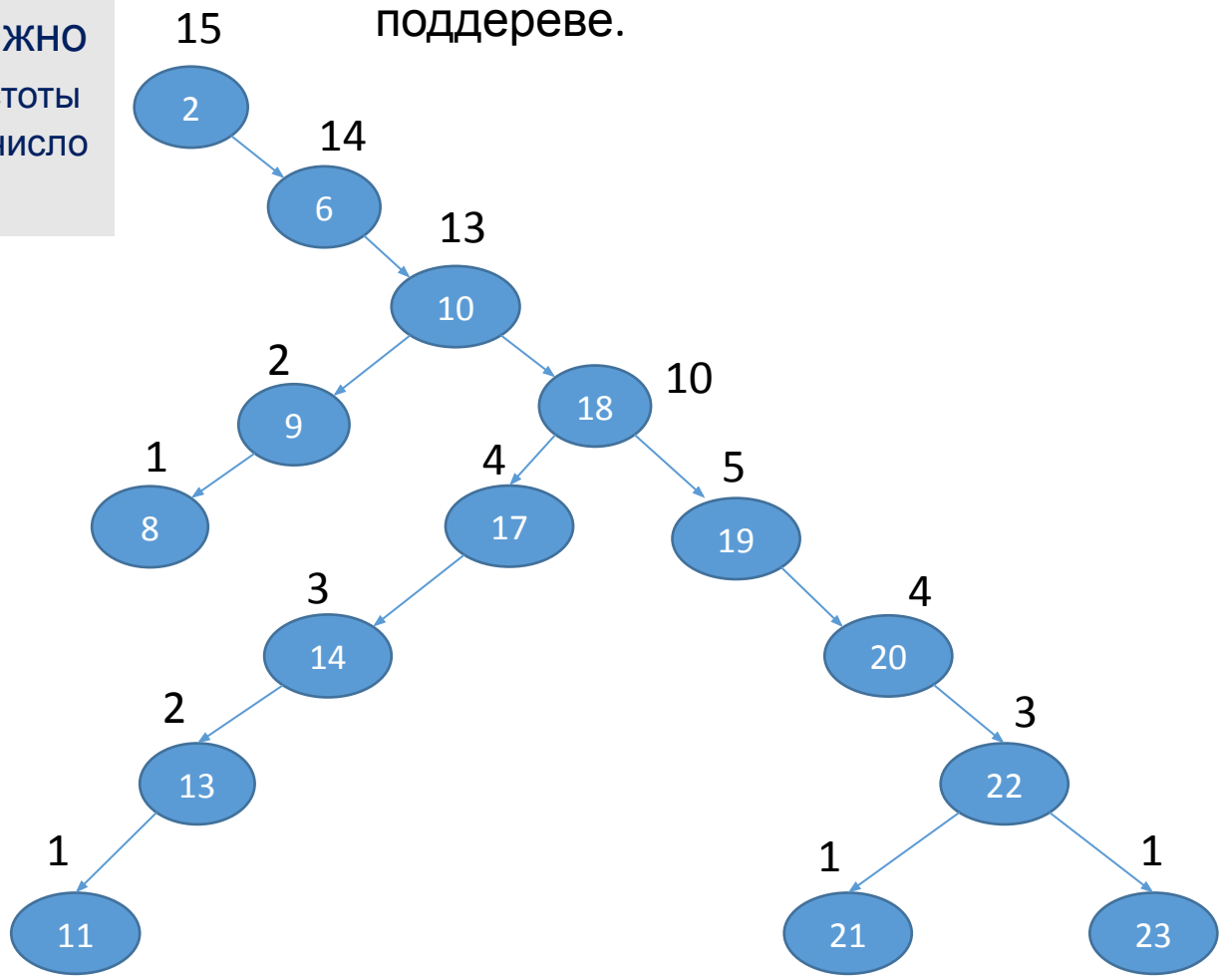


Вопрос 1.
Сколько полупутей наибольшей длины проходит через вершины?
1
2
10
18
19

Вопрос 2.
Через какие вершины пройдёт наибольшее число полупутей наибольшей длины?

4) Проверить, является ли дерево идеально-сбалансированным по числу вершин: для каждой вершины число вершин в поддеревьях должно отличаться не более, чем на 1 (для простоты вычислений, если у вершины отсутствует поддерево, то число вершин в таком поддереве полагается равным 0).

Нужно знать число вершин в каждом поддереве.



Обратный левый (или правый) **обход**

- если вершина v лист, то её метка равна 1;
- если у вершины v только одно поддерево, то её метка равна метке этого поддерева +1;
- если у вершины v есть оба поддерева, то её метка равна сумме меток поддеревьев, увеличенной на 1.

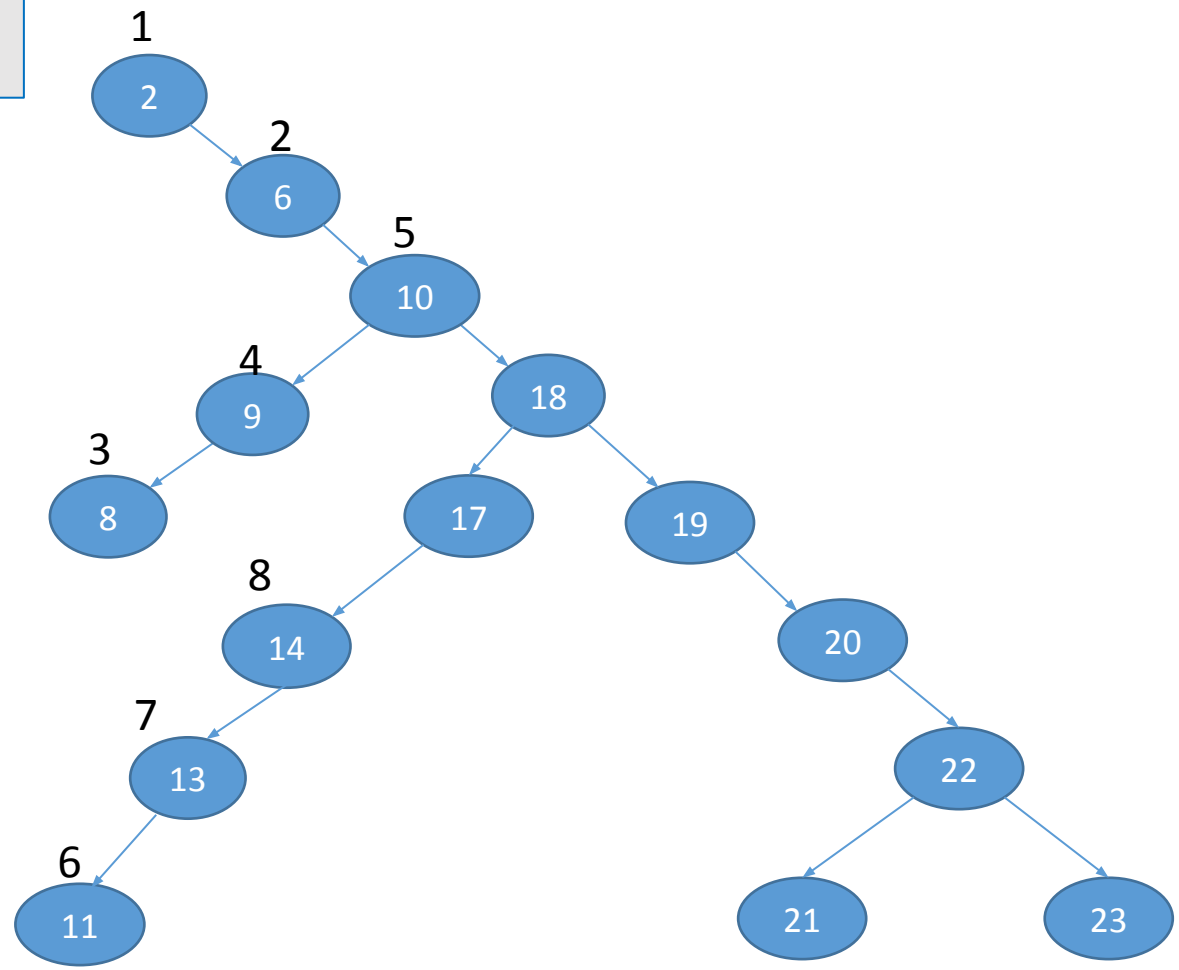
5) Найти среднюю по значению вершину в дереве

(не использовать дополнительную память, зависящую от n).

Сначала нужно узнать чётно или нет число вершин в дереве.

Если число вершин нечётно, то средняя вершина существует и её можно найти, просматривая вершины дерева, например, по не убыванию ключей.

1. Выполнить **любой обход** дерева и подсчитать число вершин (**n=15**).
2. Если n – чётно, то полагаем, что средней не существует.
3. Если n – не чётно, то выполним **внутренний обход**, считая пройденные во время этого обхода вершины. Остановимся, как только счётчик пройденных вершин станет равным [n/2]+1 **(=8)**.



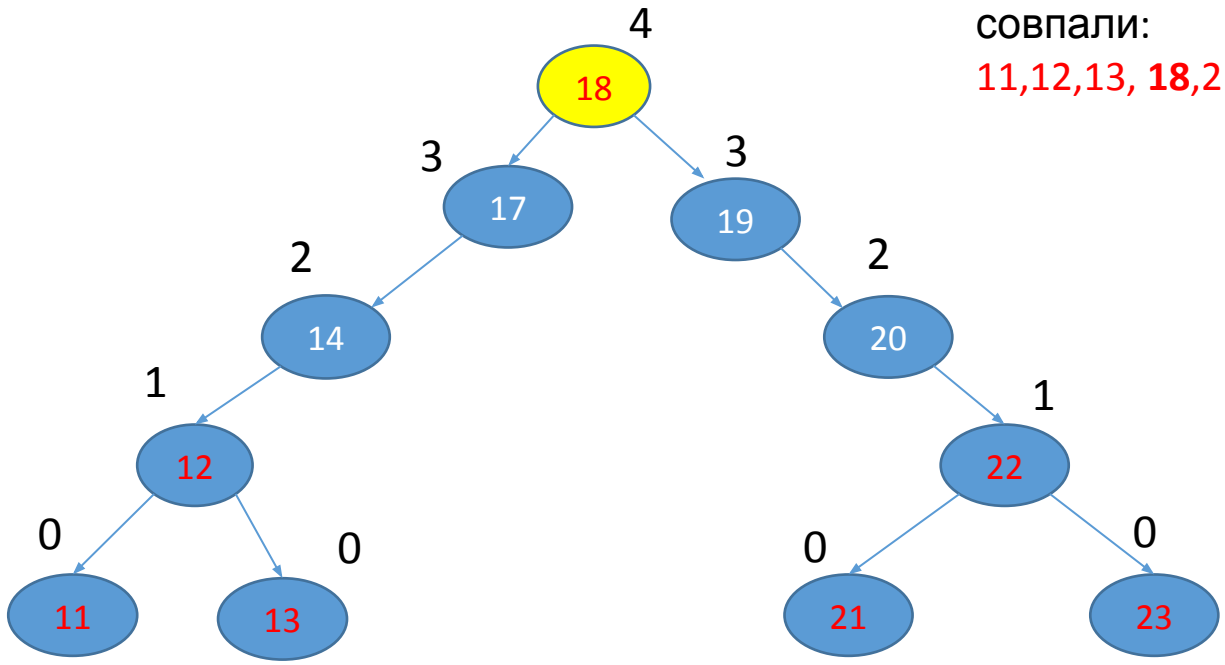
(на рис. нумерация вершин при левом InOrderTraversal)

вершина 14 является средней по значению

6) Найти среднюю по значению вершину среди вершин, у которых высоты поддеревьев совпадают.

Сначала нужно узнать чётно или нет число нужных вершин. Если число нечётно, то средняя вершина существует и её можно найти, просматривая нужные вершины, например, по не убыванию ключей.

1. Сначала **обратным** обходом расставить вершинам метки высот. Во время этого же обхода подсчитать количество вершин, у которых метки высот поддеревьев совпали. Пусть у нас **m** таких вершин.
2. Если **m** – чётно, то полагаем, что считаем, что средней не существует.
3. Если **m** – не чётно, то выполним **внутренний** обход, считая при этом только лишь те вершины, для которых высоты их поддеревьев совпадают. Остановимся, как только счётчик станет равным [**m/2]+1**.



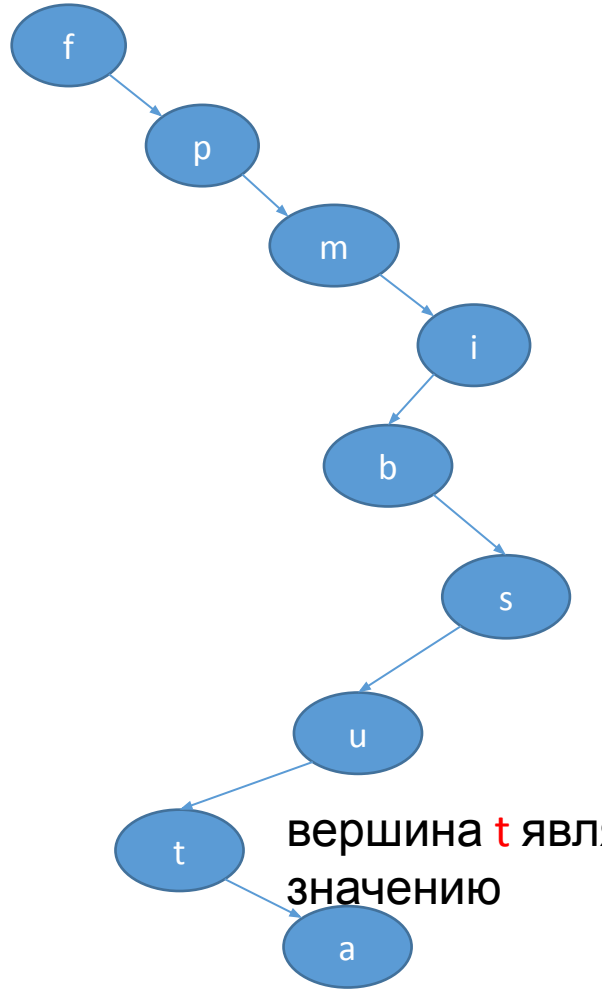
вершины, для которых высоты поддеревьев совпали:
11,12,13, **18**,21,22,23

Внутренний (левый) обход:
11,12,13, 14, 17, *18*,19,20,21,22,23

7) Найти среднюю по значению вершину среди вершин некоторого пути.

Предположим, что задан корень этого пути.

f	f				!!!			
p	f	p			!!!			
m	f	p	m		!!!			
i	f	p	m		!!!			i
b	f	p	m	b	!!!			i
s	f	p	m	b	!!!		s	i
u	f	p	m	b	!!!	u	s	i
t	f	p	m	b	t	u	s	i



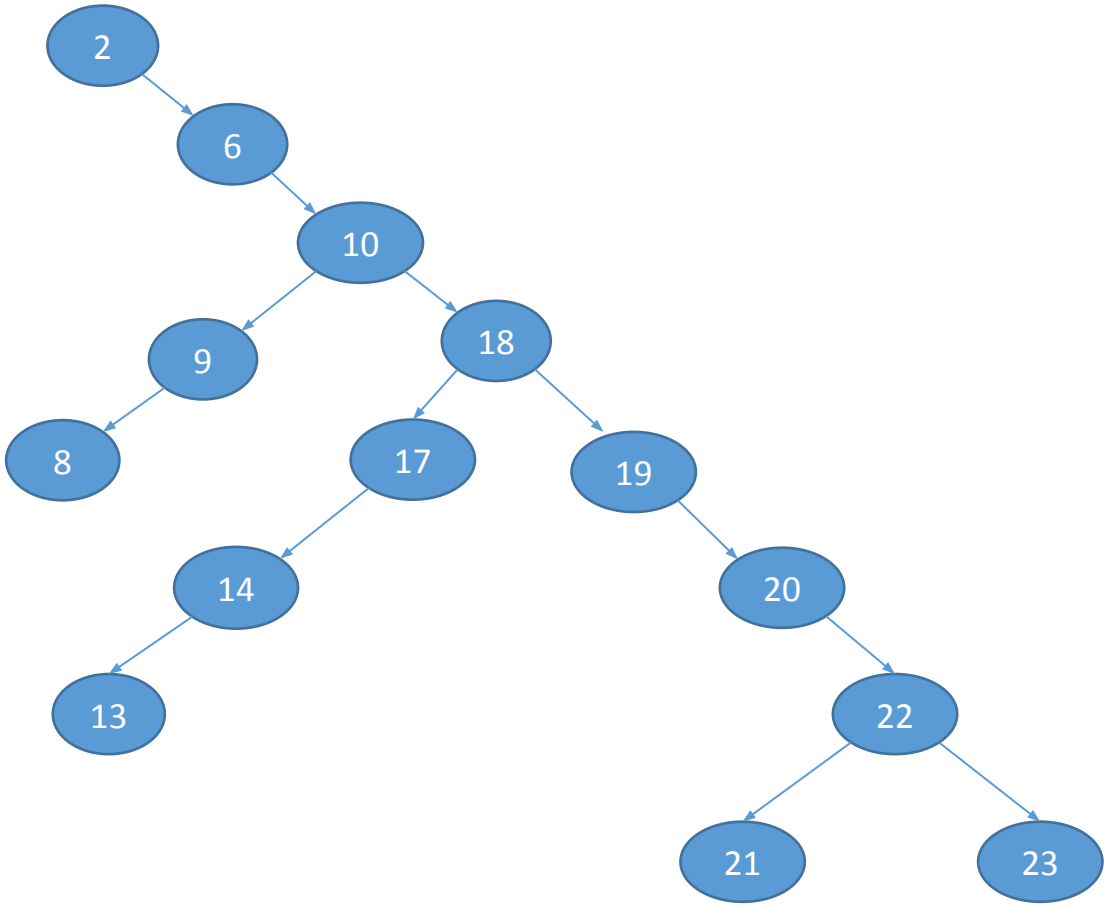
вершина t является средней по значению

Удаление вершины

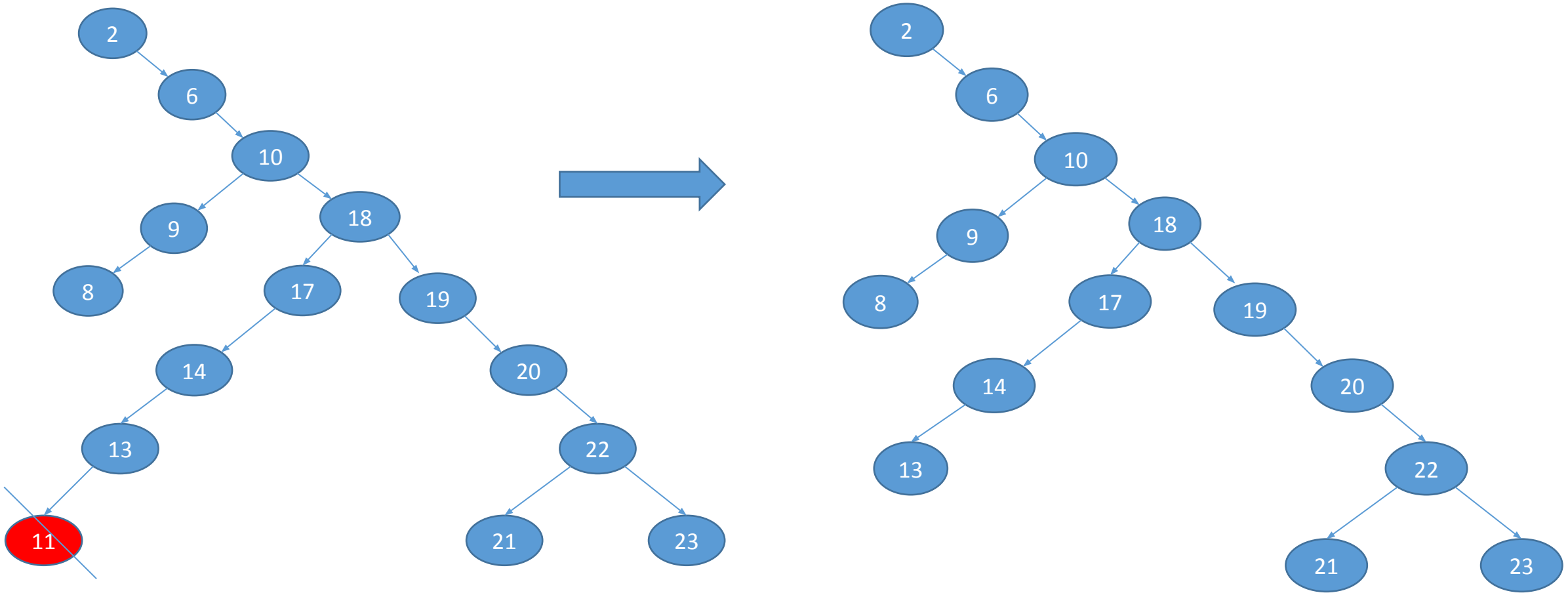
Случай 1. Удаляется лист.

Случай 2. Удаляется вершина, у которой есть только одно поддерево

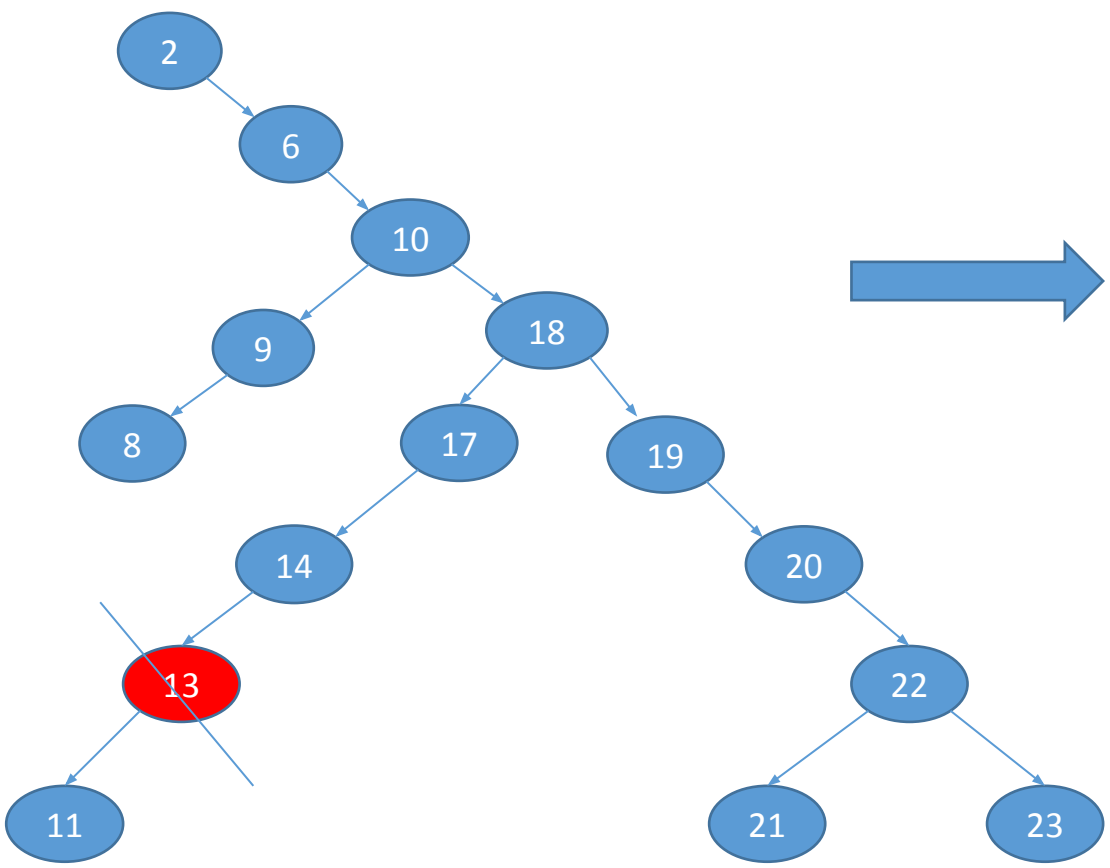
Случай 3. Удаляется вершина, у которой есть оба поддерева (возможно «правое» или «левое» удаление).

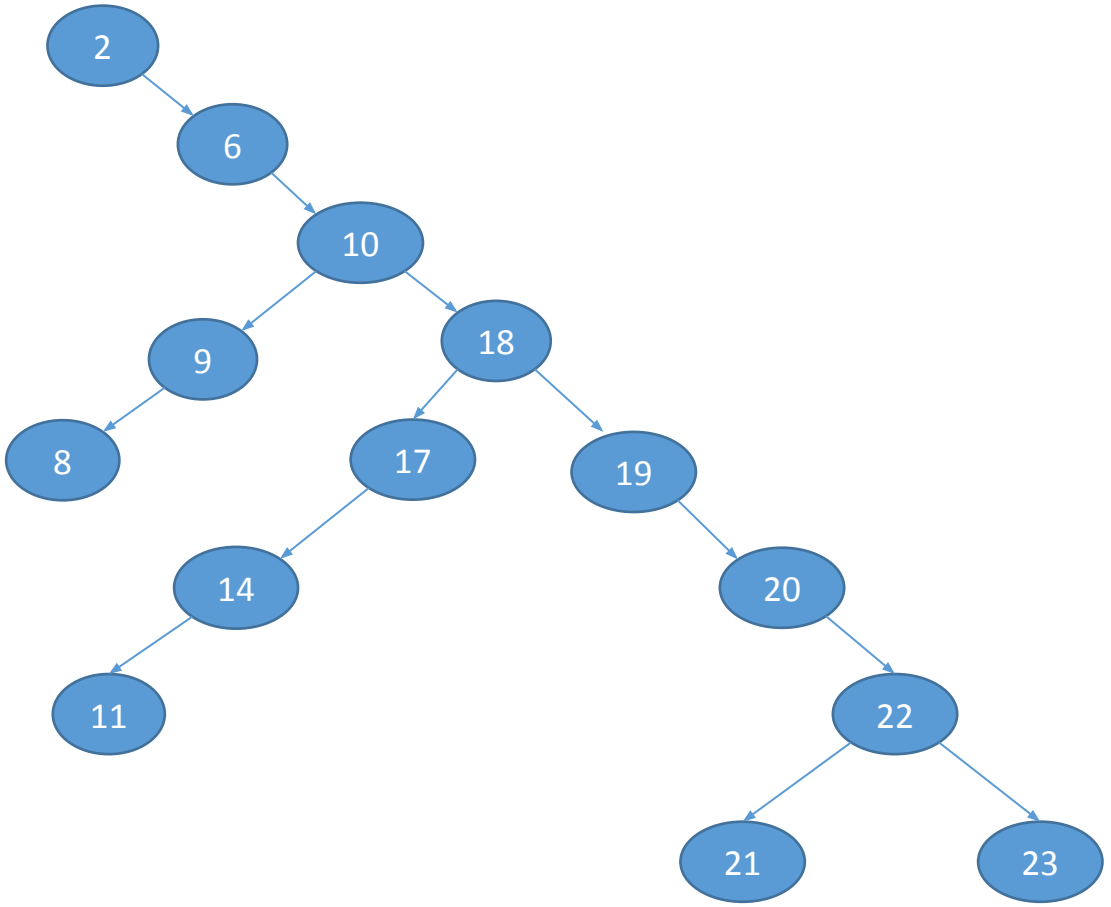


Случай 1. Удаляется лист.

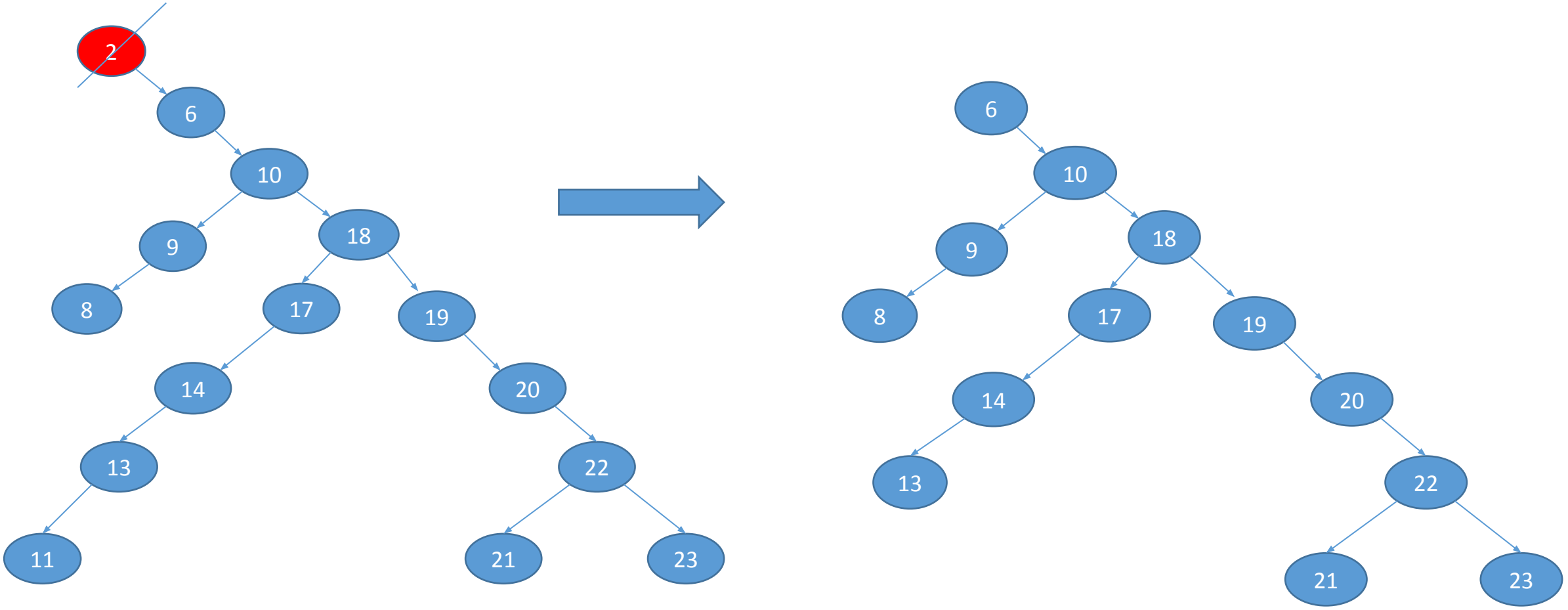


Случай 2. Удаляется вершина, у которой есть только одно поддерево

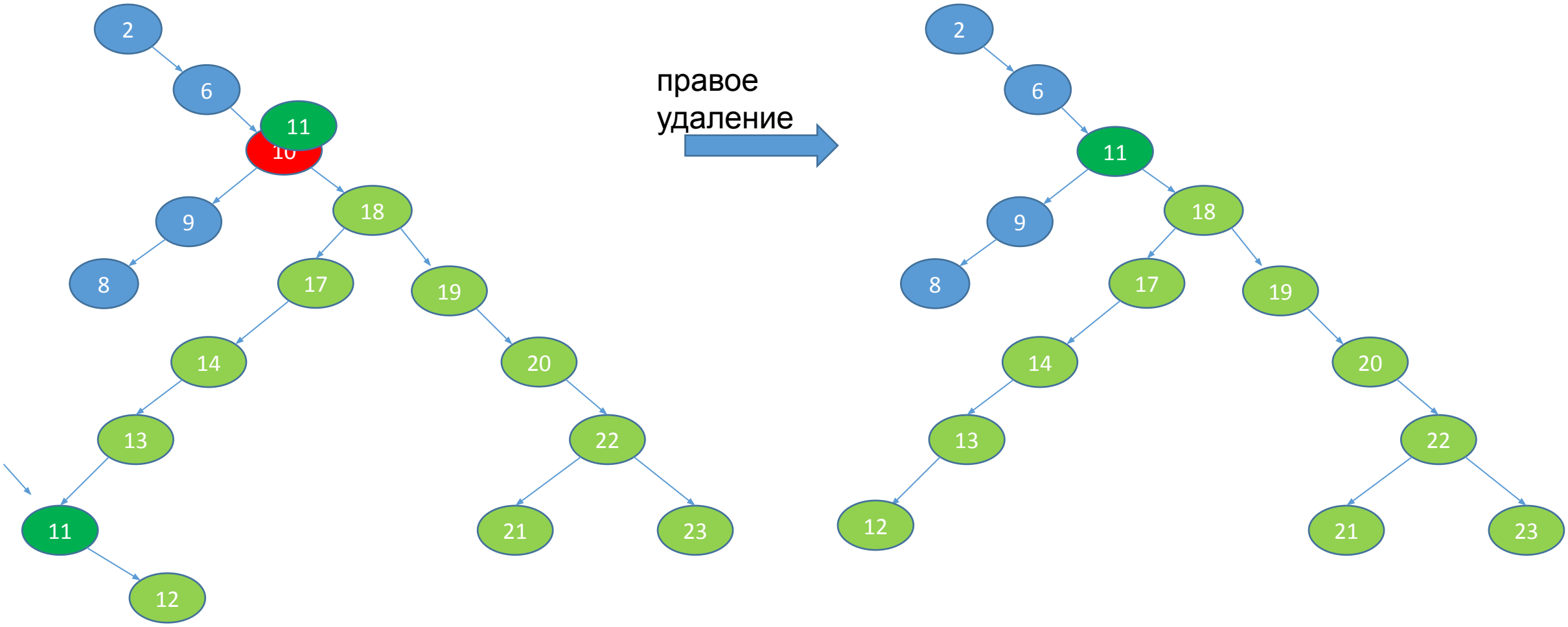




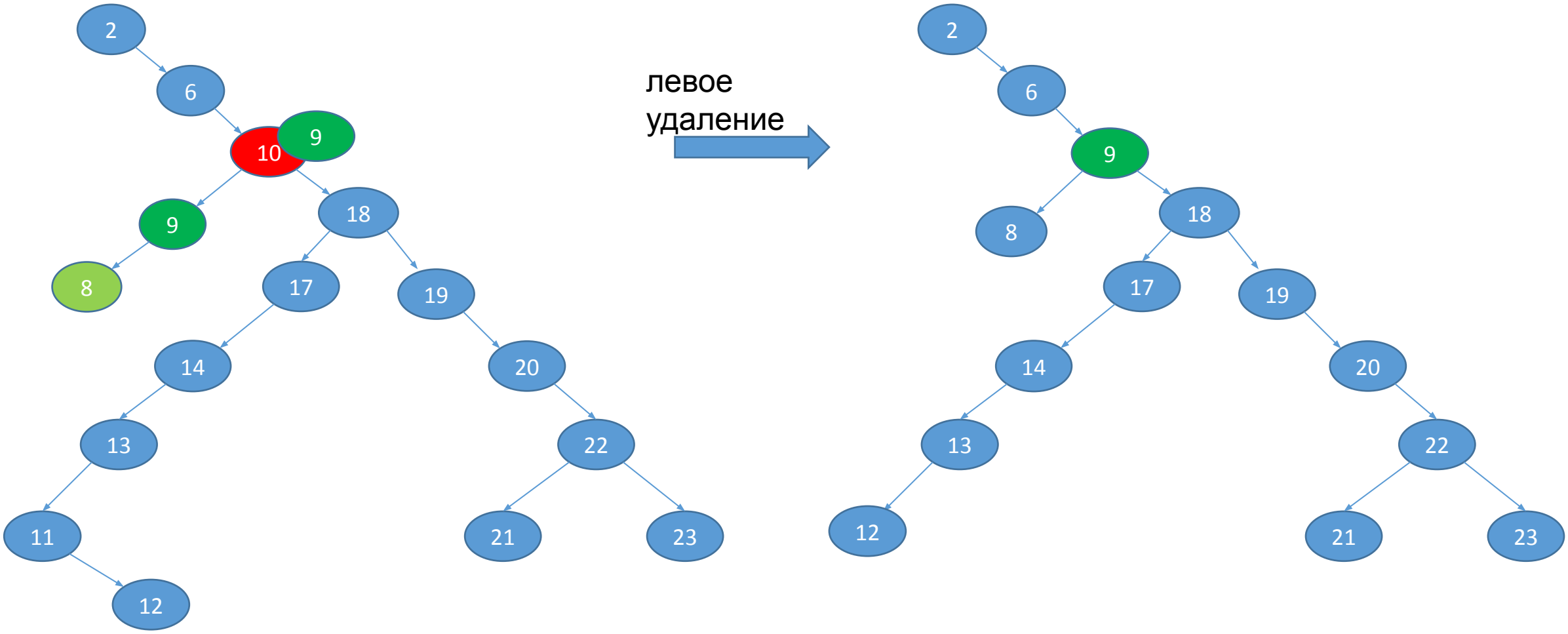
Случай 2. Удаляется вершина, у которой есть только одно поддерево



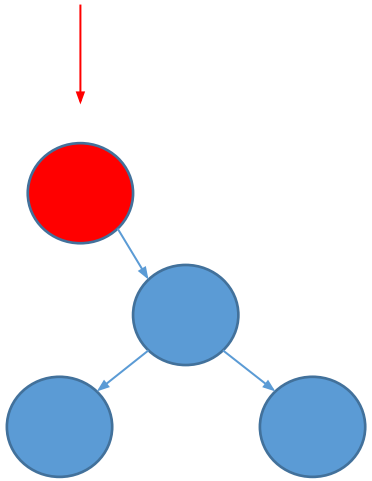
Случай 3. Удаляется вершина, у которой есть оба поддерева («правое» удаление).



Случай 3. Удаляется вершина, у которой есть оба поддерева («левое» удаление).

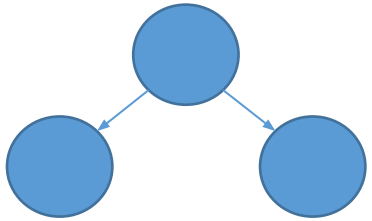


Найти вершину, которая удовлетворяет заданному свойству. Удалить эту вершину (правое удаление).

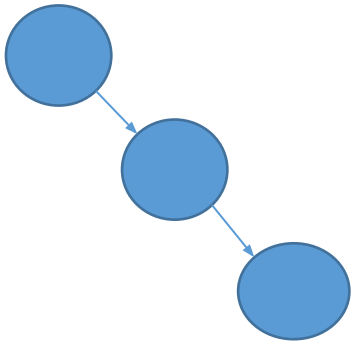


?

или



(рис.1)

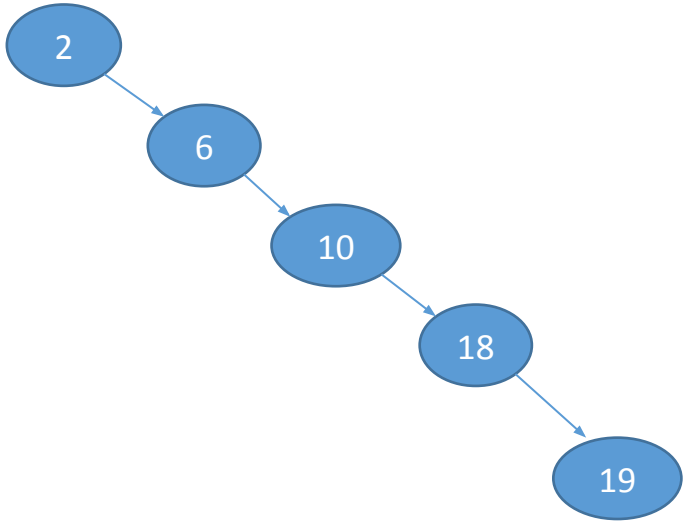


(рис.2)

!!! Если у удаляемой вершины только одно поддерево, то НЕТ ПОНЯТИЯ ПРАВОЕ/ЛЕВОЕ
удаление. Удаление всегда выполняется однозначною.

Ответ: правильно выполнено удаление на рис.
1.

Оценки числа операций в худшем случае



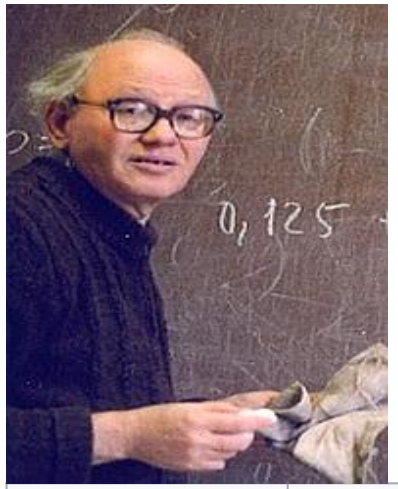
в худшем случае высота дерева
h = n-1

поиск элемента с заданным ключом ***x***	**h**
добавление элемента с заданным ключом ***x***	**h**
построение дерева для последовательности из **n** элементов	**n·h**
удаление элемента с заданным ключом ***x***	**h**
обход дерева из **n** вершин	**n**

В **1962** году советские учёные

Г.М.Адельсон-Вельский
и
Е.М.Ландис

предложили структуру данных сбалансированного поискового дерева.



Георгий Максимович Адельсон-Вельский

Дата рождения	8 января 1922
Место рождения	Самара, РСФСР
Дата смерти	26 апреля 2014 (92 года)
Место смерти	Гиватаим, Израиль
Страна	СССР → Израиль
Научная сфера	математик
Место работы	Институт теоретической и экспериментальной физики
Альма-матер	МГУ (мехмат)
Учёная степень	кандидат ф.-м. наук

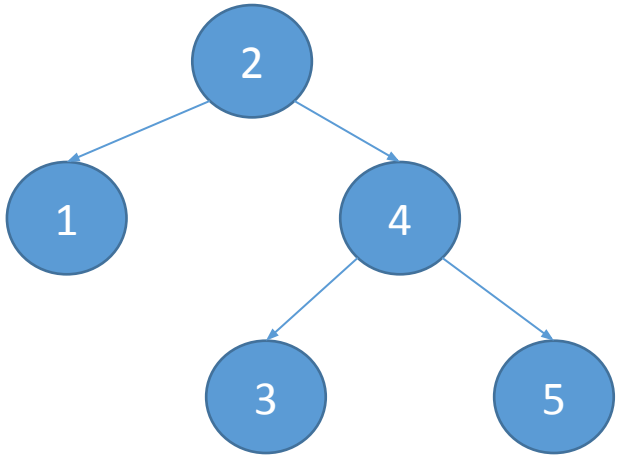
Евгений Михайлович Ландис

Дата рождения	6 октября 1921
Место рождения	Харьков
Дата смерти	12 декабря 1997 (76 лет)
Место смерти	Москва, Россия
Страна	СССР → Россия
Научная сфера	математика
Место работы	Московский государственный университет
Альма-матер	МГУ (мехмат)
Учёная степень, звание	доктор ф.-м. наук, профессор

В рамках дисциплины в дальнейшем мы подробно исследуем эту структуру данных, а пока лишь получим краткую информацию о ней.

АВЛ-дерево – это бинарное поисковое дерево, которое является сблансированным по высоте.

АВЛ — аббревиатура, образованная первыми буквами фамилий создателей.



для каждой вершины дерева высоты её поддеревьев отличаются не более, чем на 1

ТЕОРЕМА. Пусть n – число внутренних вершин АВЛ-дерева, а h – его высота.

Тогда справедливы следующие неравенства:

$$\log(n+1) \le h < 1,4404 \cdot \log(n+2) - 0,328$$

Использование поисковых деревьев на практике

Сортировка деревом

Предположим, что на вход поступаю числа, среди которых нет повторяющихся.

Рассмотрим следующий алгоритм сортировки.

1. По последовательности чисел сначала построим АВЛ-дерево.

$n*\log n$

2. Выполним внутренний левый обход построенного дерева.

n

В результате работы алгоритма числа будут выданы в порядке возрастания.

Какое время работы алгоритма в худшем случае?

$n*\log n$

Абстрактный тип данных: множество (**set**)

Множество (англ. set) —хранит набор попарно различных объектов без определённого порядка.

Интерфейс множества включает три основные операции:

1) Insert(x) — добавить в множество ключ x;
2) Contains(x) — проверить, содержится ли в множестве ключ x;
3) Remove(x) — удалить ключ x из множества.

Для реализации интерфейса множества обычно используются такие структуры данных, как:

- сбалансированные поисковые деревья: например, AVL-деревья, 2-3-деревья, красно-чёрные деревья.
- хеш-таблицы.

В стандартной библиотеке C++ есть контейнер std::set, который реализует множество на основе сбалансированного дерева (обычно красно-чёрного), и контейнер std::unordered_set, построенный на базе хеш-таблицы.

В языке Java определён интерфейс Set, у которого есть несколько реализаций, среди которых классы TreeSet (работает на основе красно-чёрного дерева) и HashSet (на основе хеш-таблицы).

В языке Python есть только встроенный тип set, использующий хеширование, но нет готового класса множества, построенного на сбалансированных деревьях.

Абстрактный тип данных ассоциативный массив (**map**)

Ассоциативный массив (англ. associative array), или отображение (англ. map), или словарь (англ. dictionary), — хранит пары вида (ключ, значение), при этом каждый ключ встречается не более одного раза.

Название «ассоциативный» происходит от того, что значения ассоциируются с ключами.

Интерфейс ассоциативного массива включает операции:

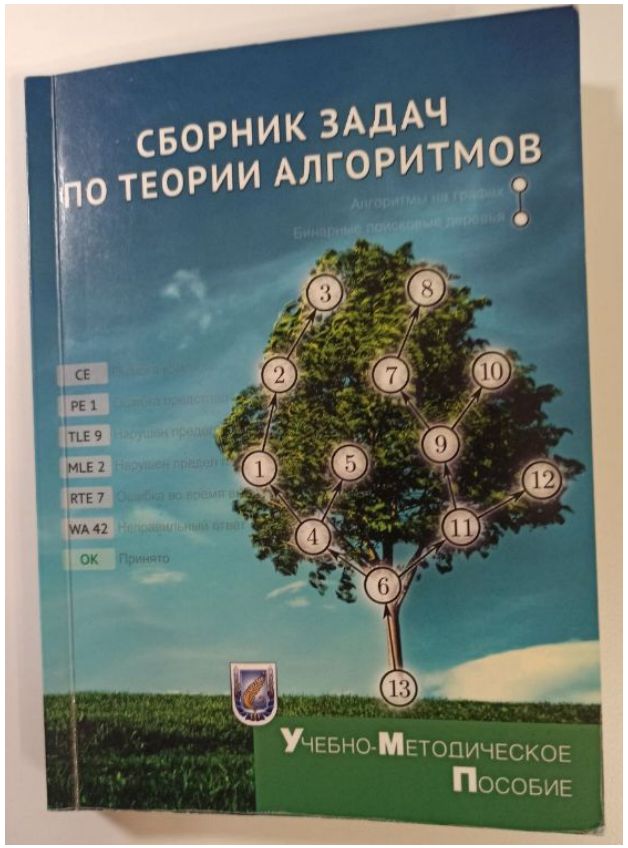
1) Insert(k,v) — добавить пару, состоящую из ключа k и значения v;
2) Find(k) — найти значение, ассоциированное с ключом k, или сообщить, что значения, связанного с заданным ключом, нет;
3) Remove(k) — удалить пару, ключ в которой равен k.

Данный интерфейс реализуется на практике теми же способами, что и интерфейс множества. Реализация технически немного сложнее, чем множества, но использует те же идеи.

Для языка программирования C++ в стандартной библиотеке доступен контейнер std::map, работающий на основе сбалансированного дерева (обычно красно-чёрного), и контейнер std::unordered_map, работающий на основе хеш-таблицы.

В языке Java определён интерфейс Map, который реализуется несколькими классами, в частности классом TreeMap (базируется на красно-чёрном дереве) и HashMap (базируется на хеш-таблице).

В языке Python очень широко используется встроенный тип dict. Этот словарь использует внутри хеширование.



Литература

Сборник задач по теории алгоритмов : учеб.-метод. пособие / В.М. Котов, Ю.Л. Орлович, Е.П. Соболевская, С.А. Соболь – Минск : БГУ, 2017. С. 122-180

https://acm.bsu.by/wiki/Программная_реализация_бинарных_поисковых_деревьев

Общие задачи

[0.1 построение дерева]
[0.2 удаление вершин из дерева]
[0.3 проверка является ли бинарное дерево поисковым]

Спасибо за внимание!