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

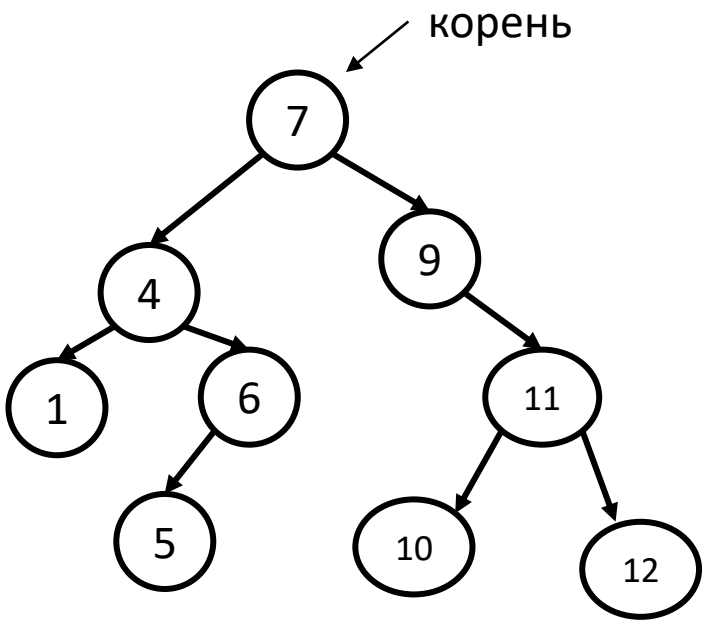
Бинарное поисковое дерево

Корневое дерево

1. Ориентированный граф, в котором 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.



Поисковое

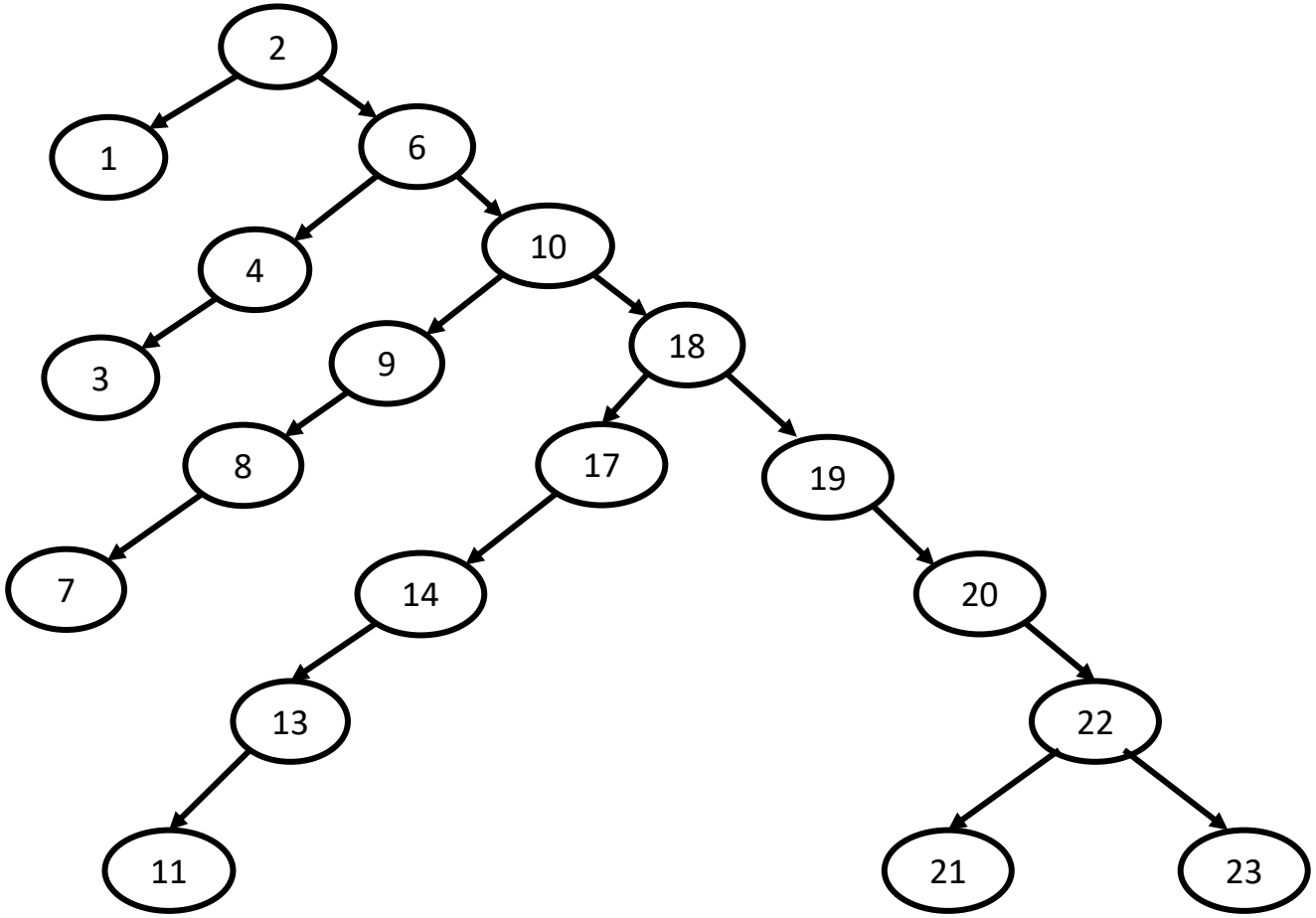
5. Каждой вершине поставлено в соответствие некоторое целое число - *ключ*. Для каждой вершины ***v*** все ключи в её левом поддереве строго меньше ключа вершины ***v***, а в правом – строго больше.

n – число вершин

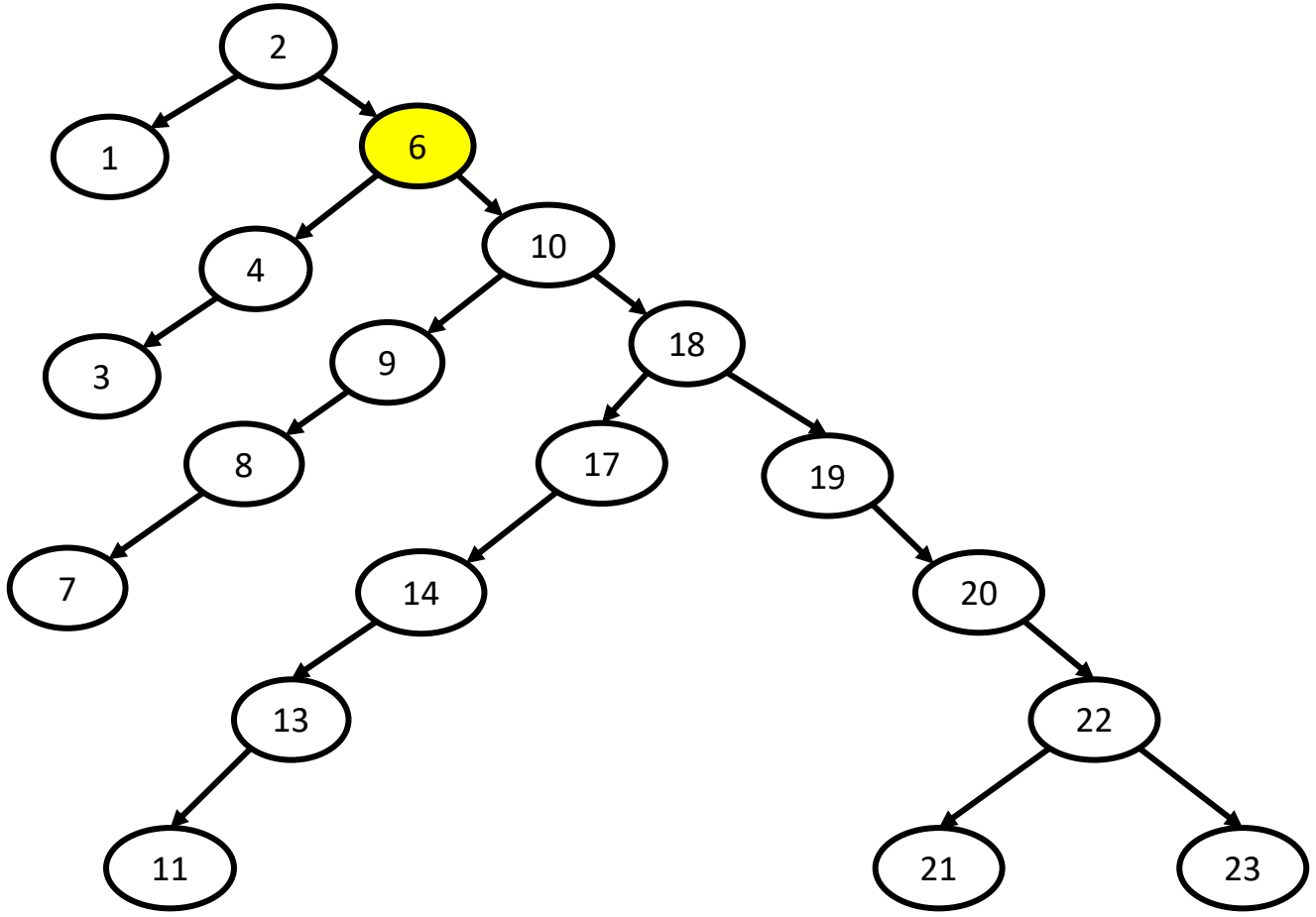
m=n-1 – число дуг

Построить бинарное поисковое дерево для последовательности элементов последовательным добавлением нового элемента:

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

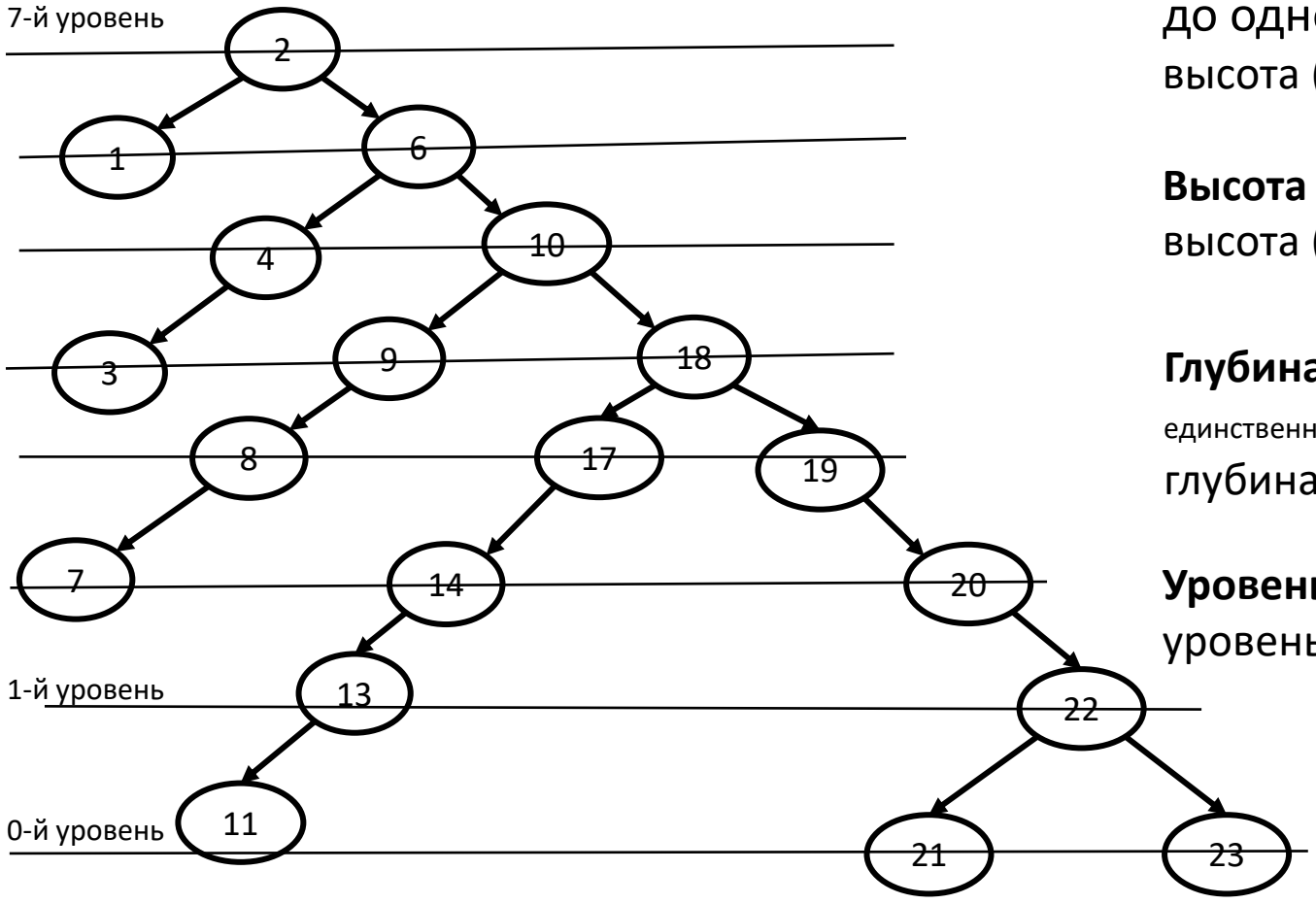


??? добавить **6**



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**

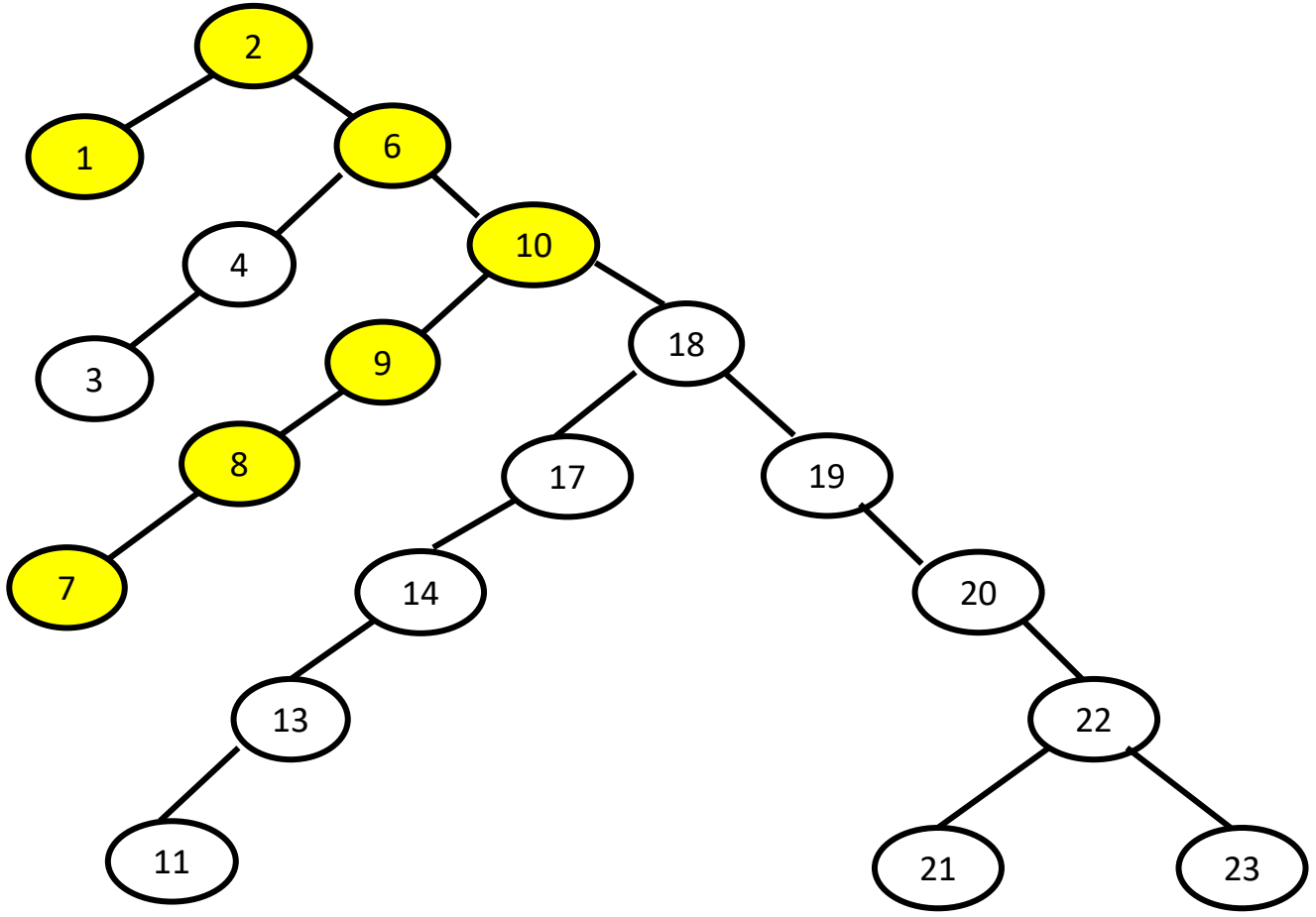
Глубина вершины: длина пути из корня в вершину (этот путь единственный).
глубина (10) = **2**

Уровень вершины: высота дерева минус глубина вершины
уровень (10) = высота (дерева)- глубина (10) =7 – 2 = **5**

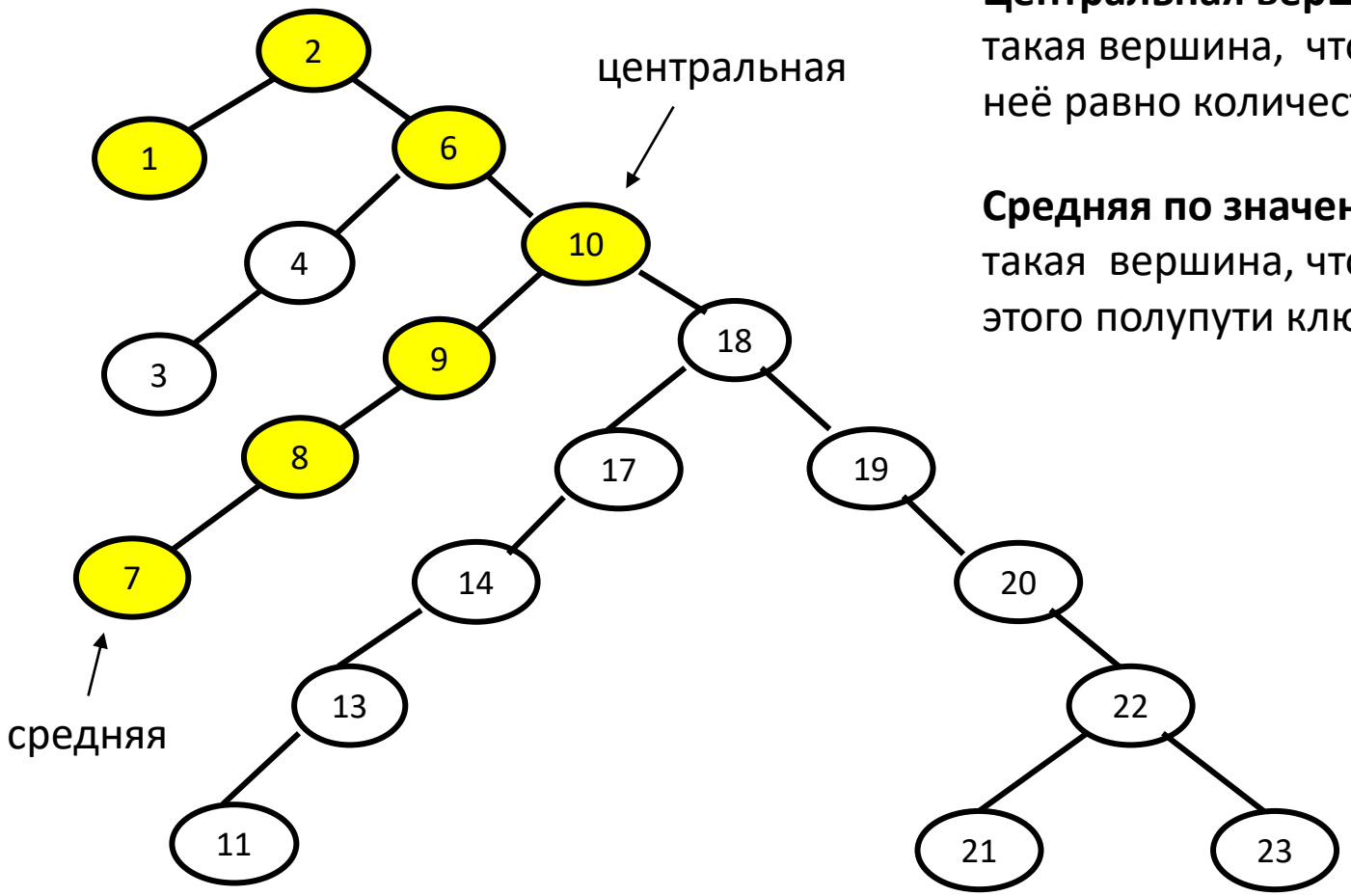
Путь, полупуть

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.

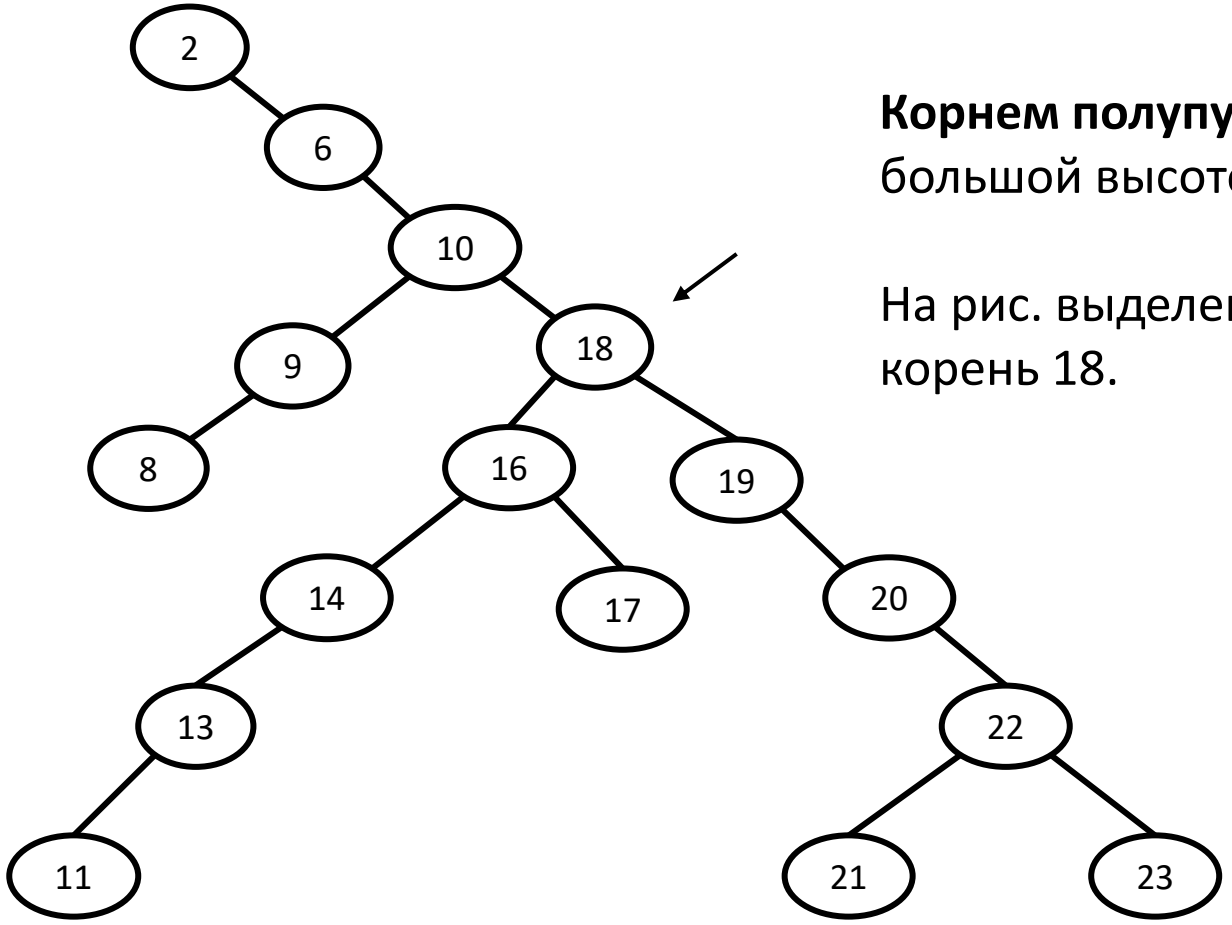
?

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 (напомним, что длина пути измеряется в рёбрах, а не вершинах).



Корнем полупути назовём вершину полупути с самой большой высотой.

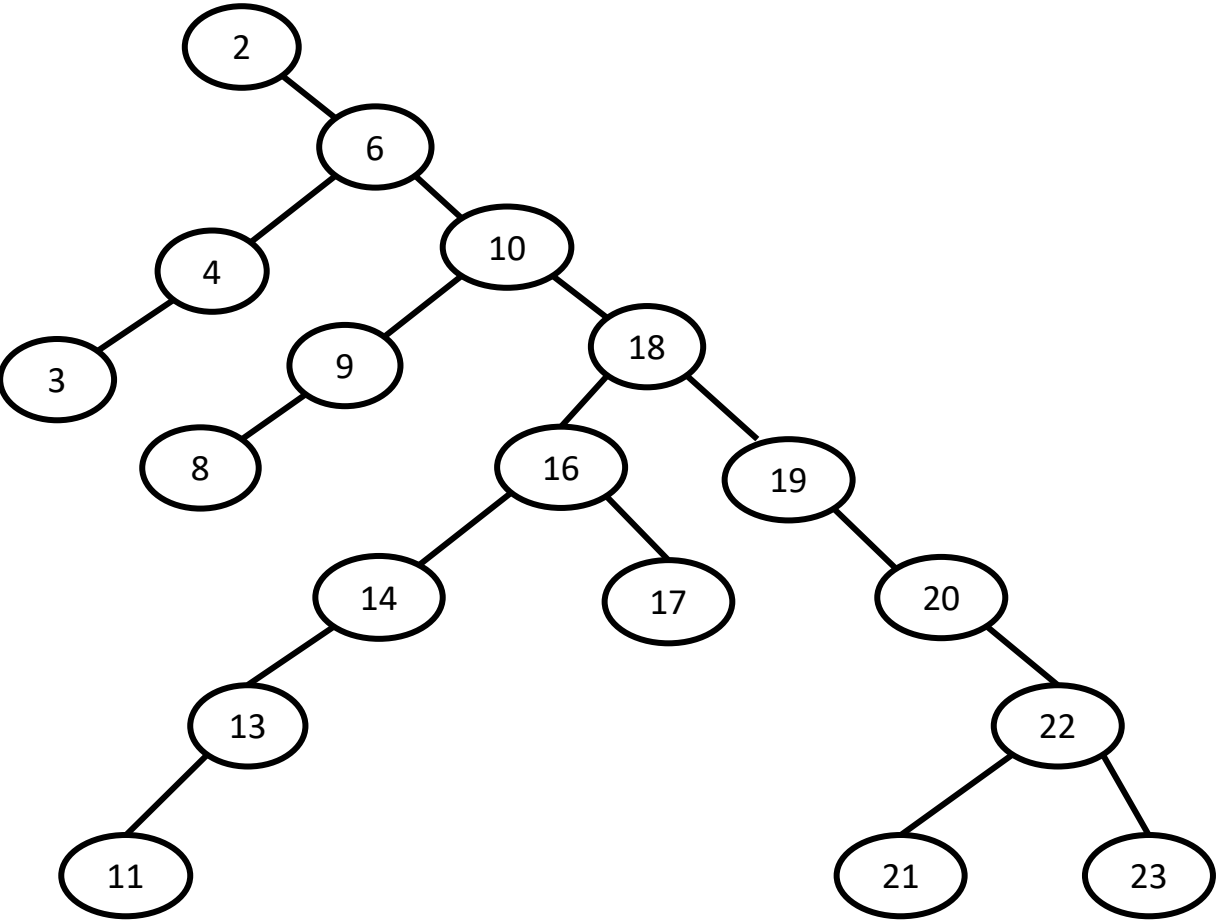
На рис. выделены два наибольших полупути и у них общий корень 18.

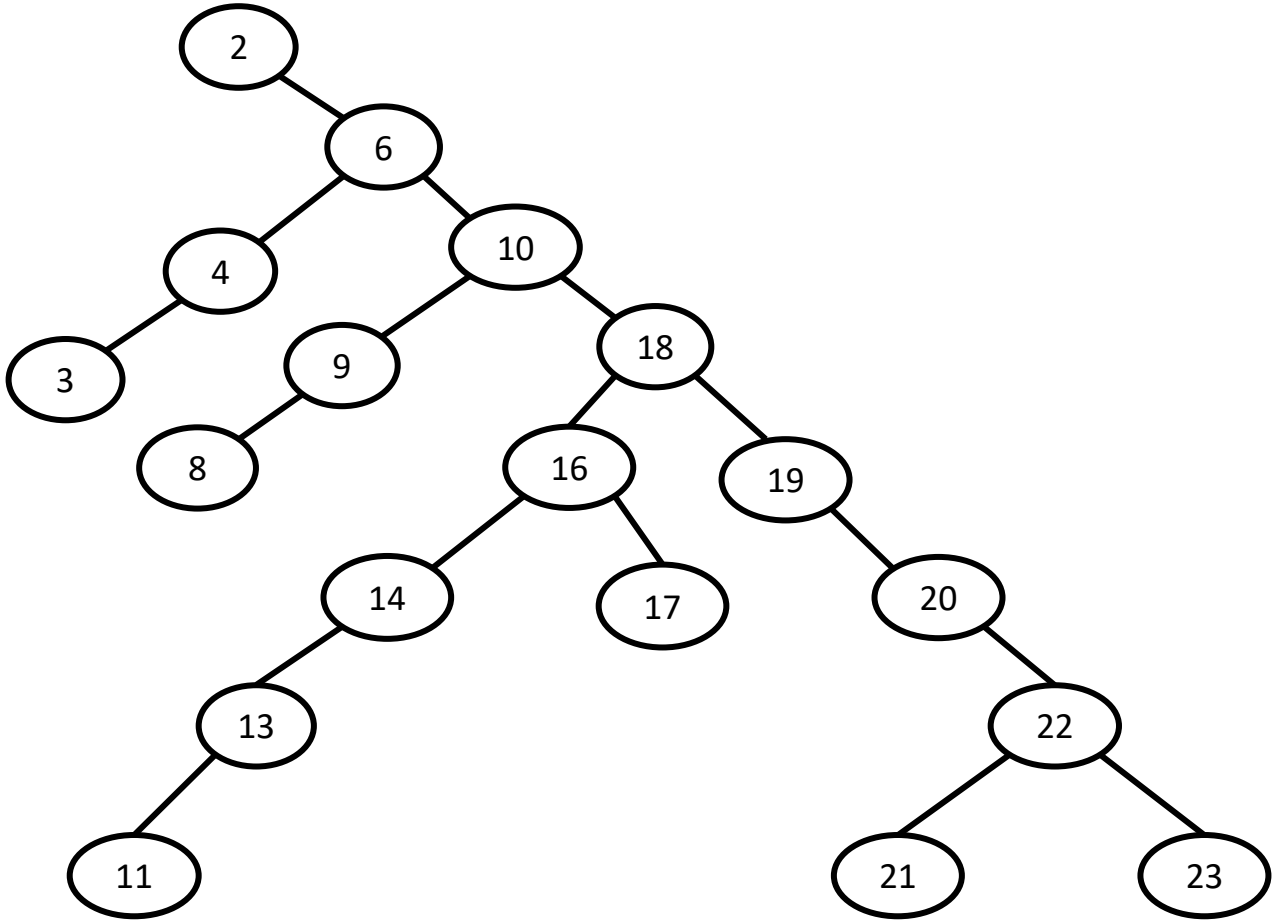
Пример.

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

2) Какие вершины являются корнями полупутей наибольшей длины? **=18 и 6**

3) Сколько попарно различных полупутей наибольшей длины проходит через вершину 18? **=5**





Обходы

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

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

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

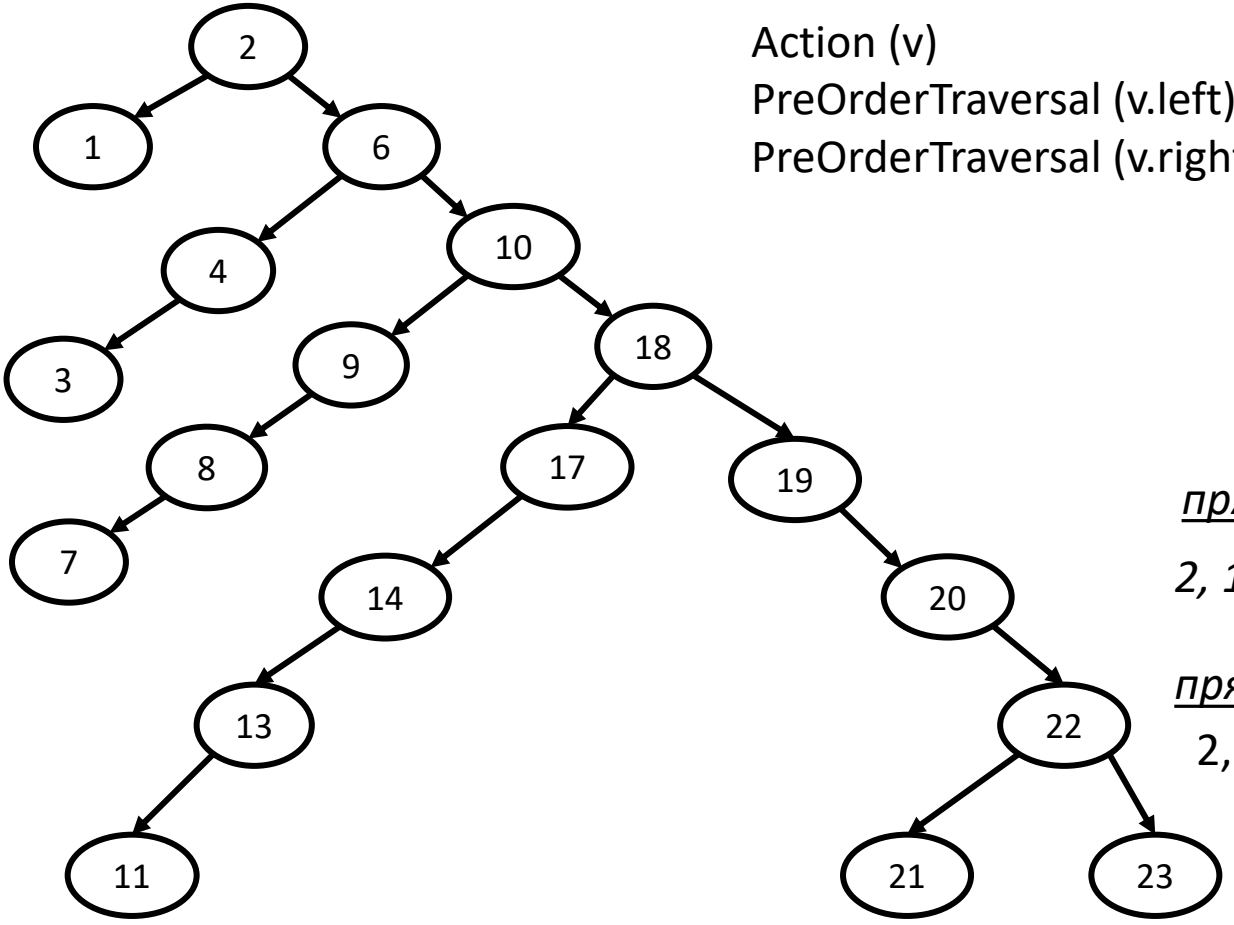
Время выполнения обхода: пропорционально числу вершин в дереве (=**n)**

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

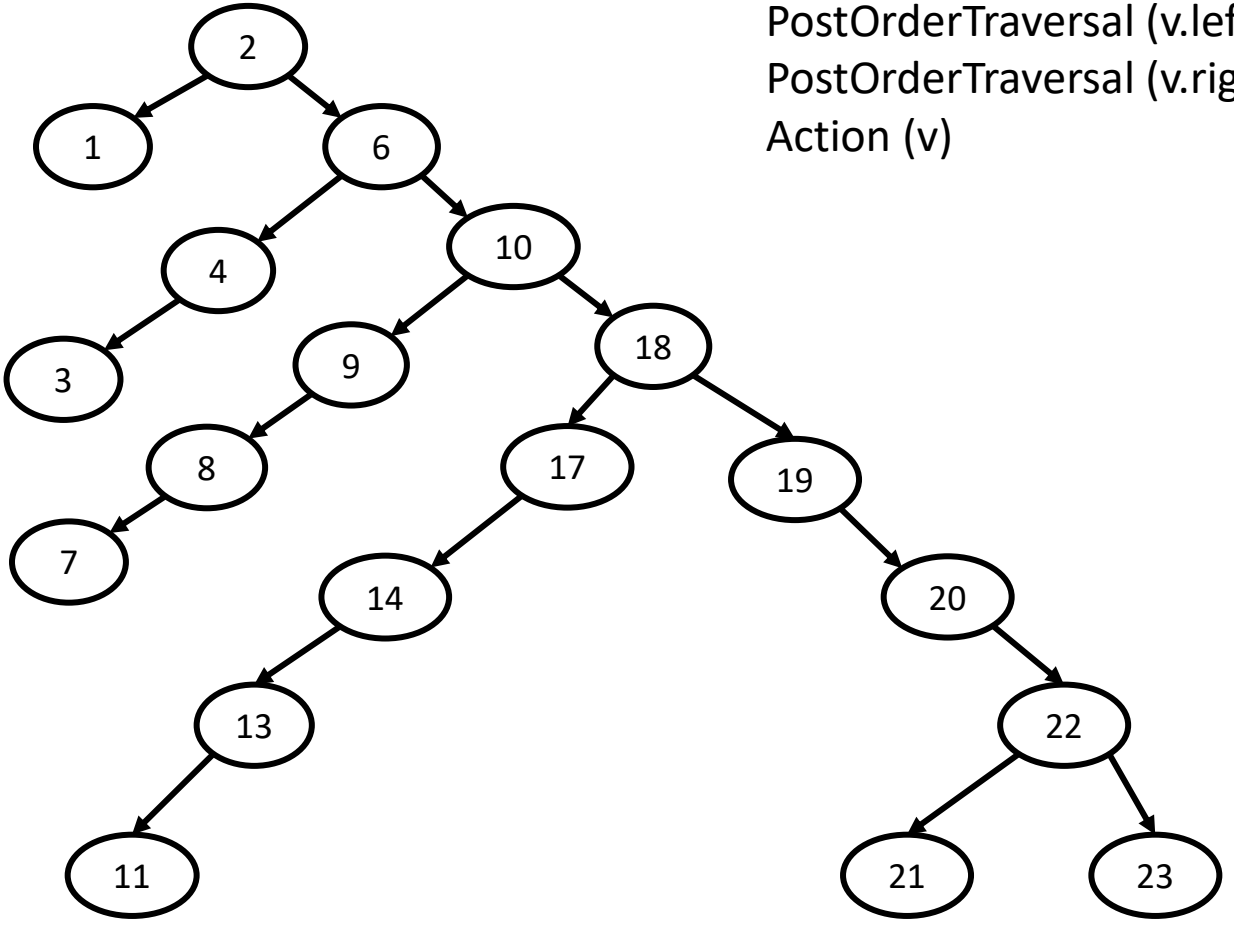
2, 6, 10, 18, 19, 20, 22, 23, 21, 17, 14, 13, 11, 9, 8, 7, 4, 3, 1

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

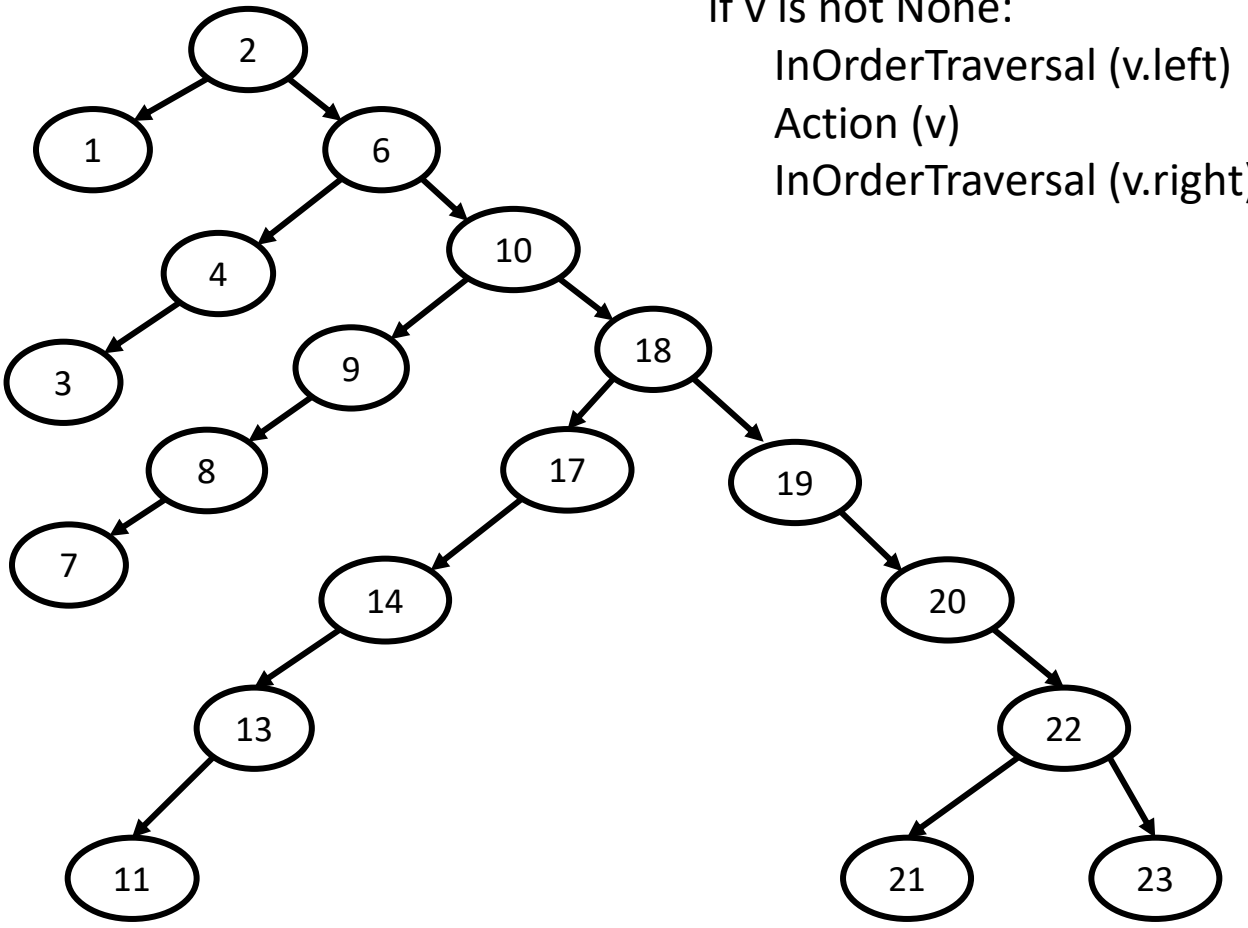
23, 21, 22, 20, 19, 11, 13, 14, 17, 18, 7, 8, 9, 10, 3, 4, 6, 1, 2

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:
        InOrderTraversal (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)

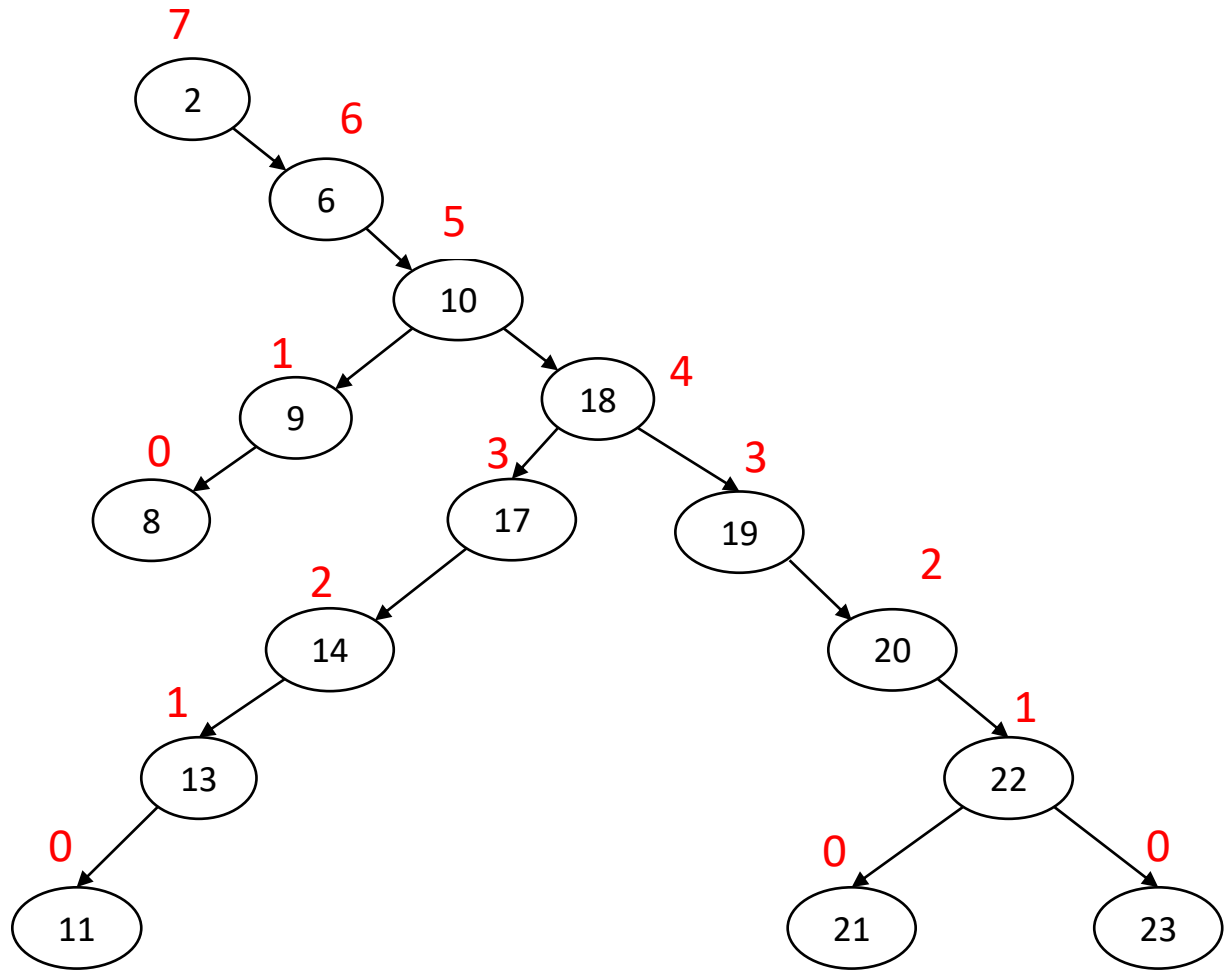
23, 22, 21, 20, 19, 18, 17, 14, 13, 11, 10, 9, 8, 7, 6, 4, 3, 2, 1

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.
8) Проверить, является ли бинарное дерево поисковым.

1) Найти **высоту** дерева.

2) Определить, является ли дерево **сбалансированным по высоте:** для всех вершин высоты их поддеревьев должны отличаться не более, чем на 1.



Нужно знать высоту каждой вершины.

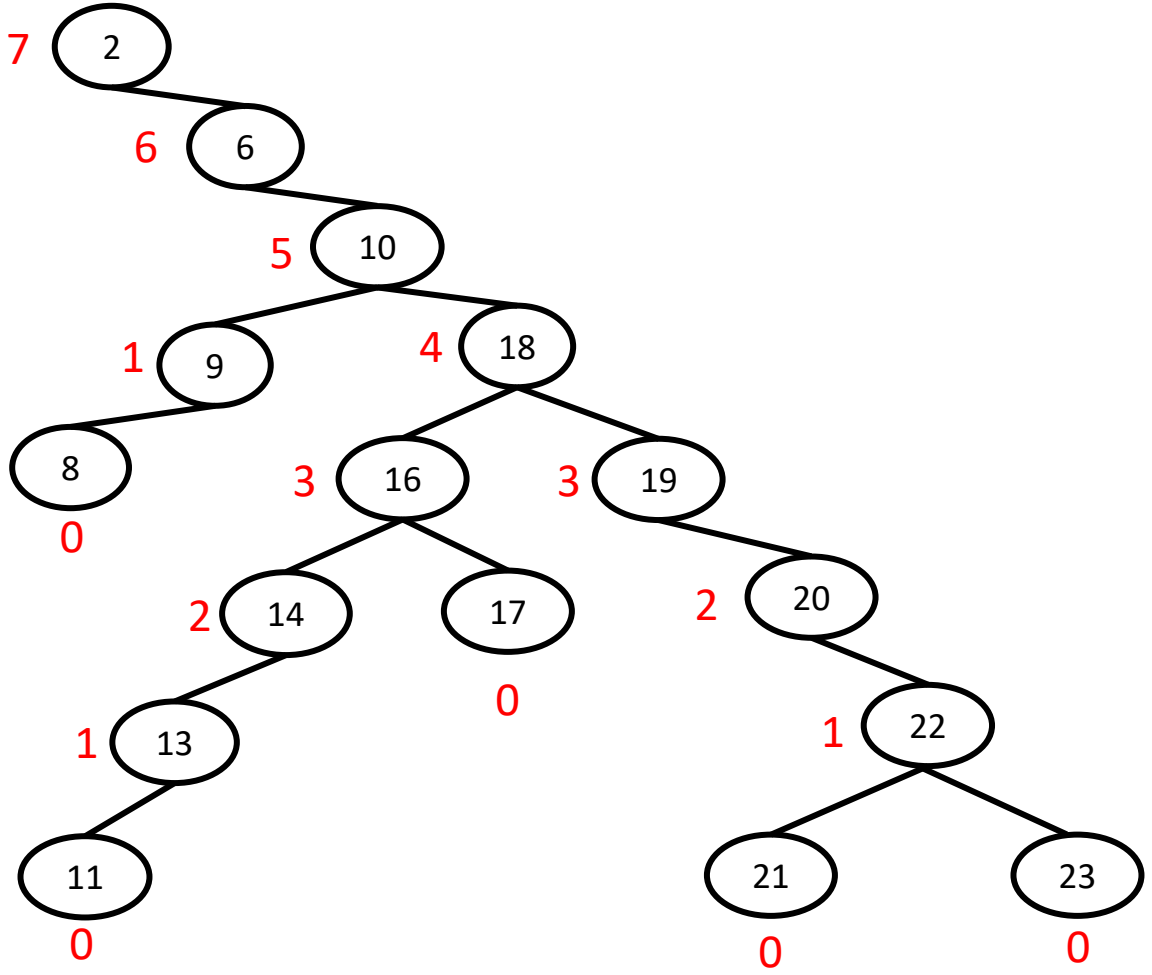
Обратный левый (или правый) **обход**

- 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, увеличенная на число 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** и указать корни полупутей наибольшей длины.

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

Вопрос 1. Сколько полупутей наибольшей длины проходит через вершины?

1 **=3**

2 **=3**

10 **=6**

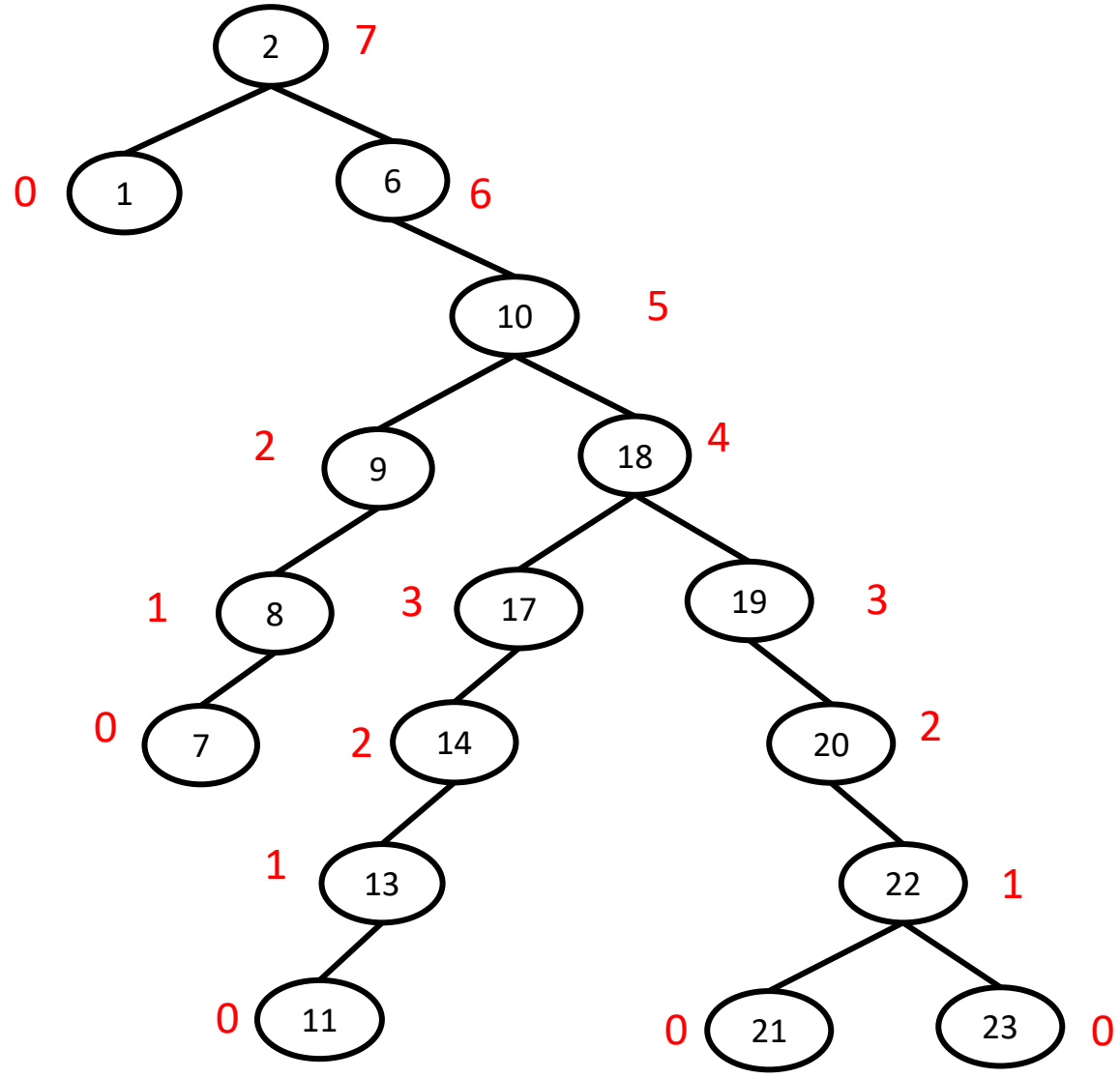
18 **=8**

19 **=6**

17 **=4**

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

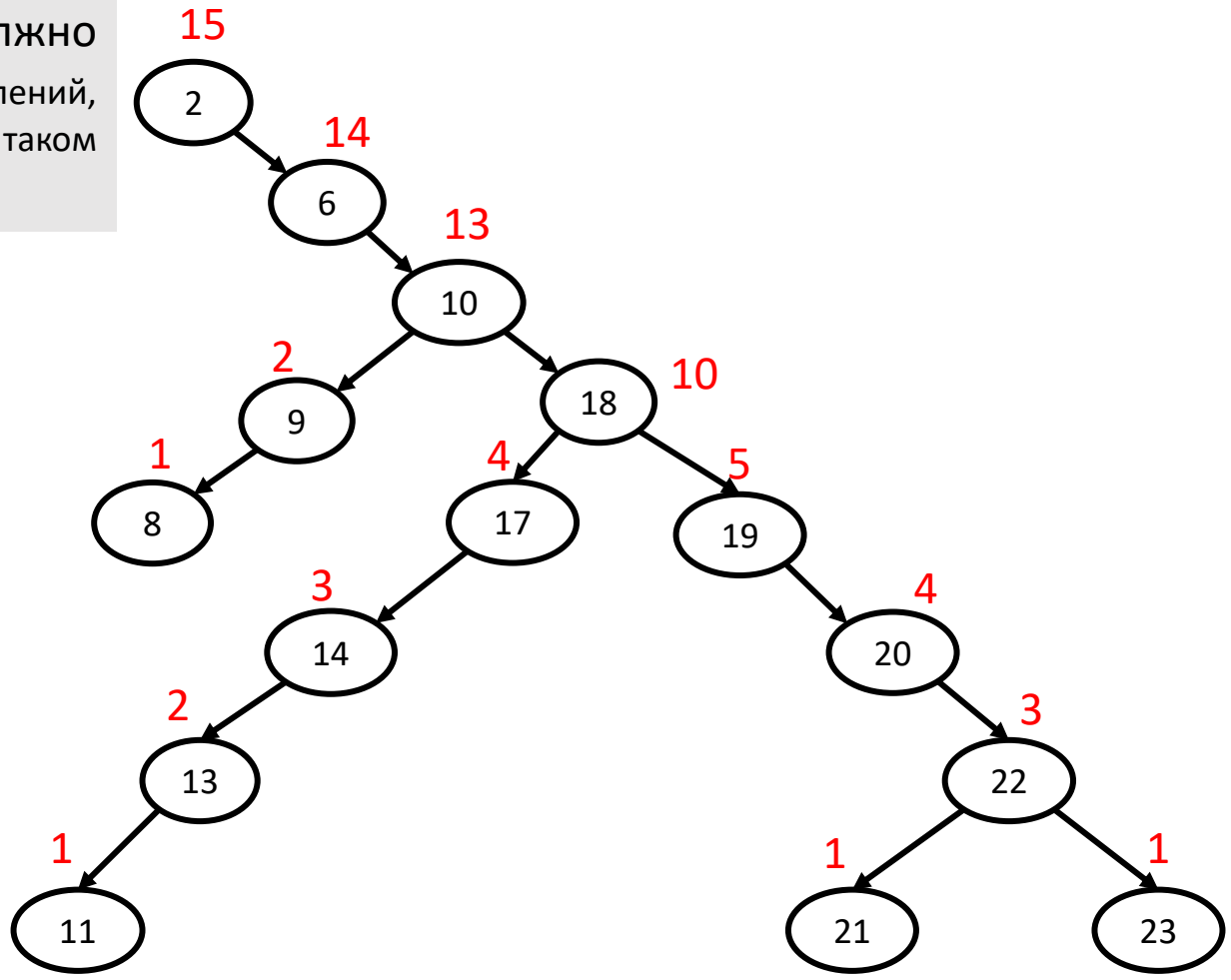
v=18



ФПМИ БГУ

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.

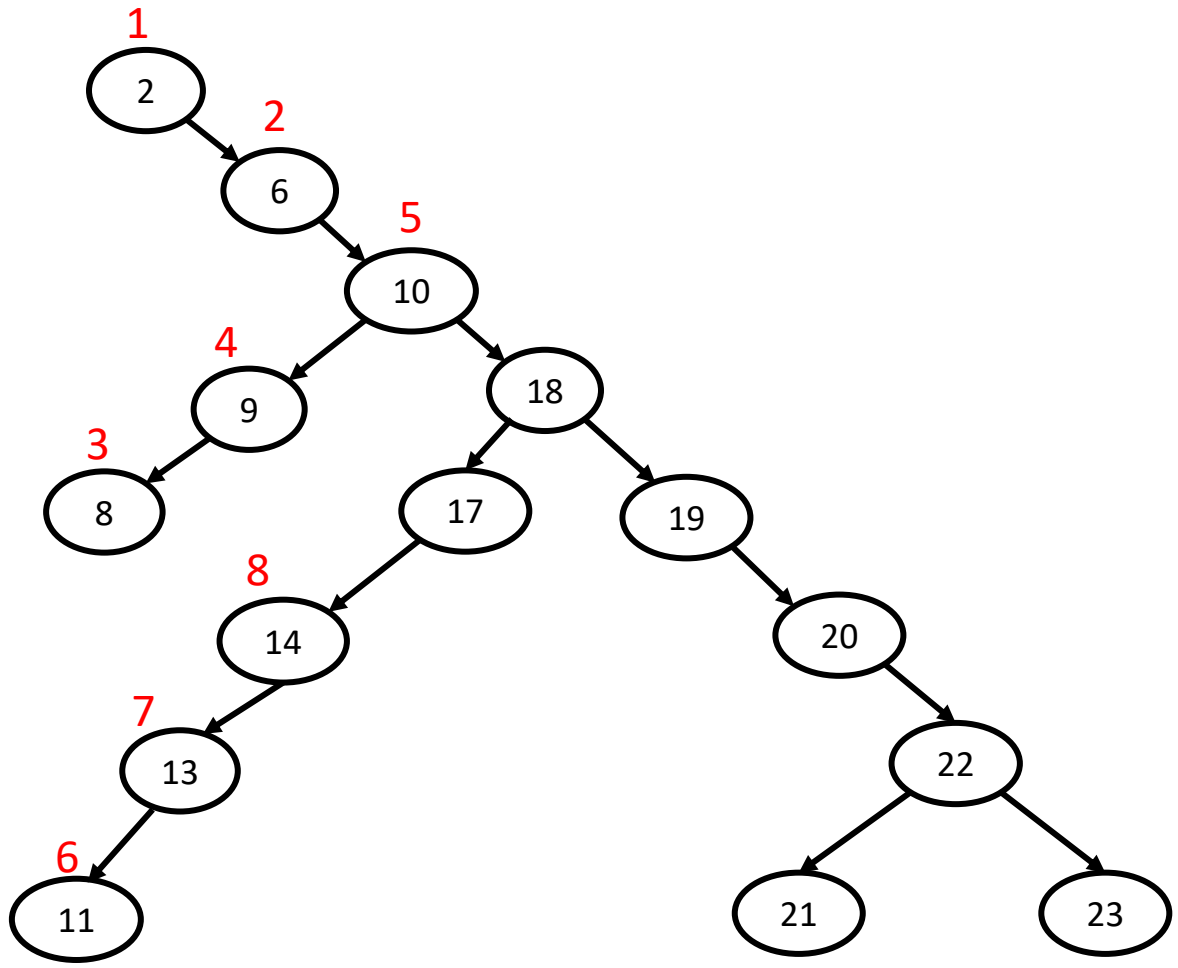
5) Найти **среднюю по значению** вершину в дереве

(не использовать дополнительную память, зависящую от n, решить задачу за линейное от количества вершин время).

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 неубыванию ключей.

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



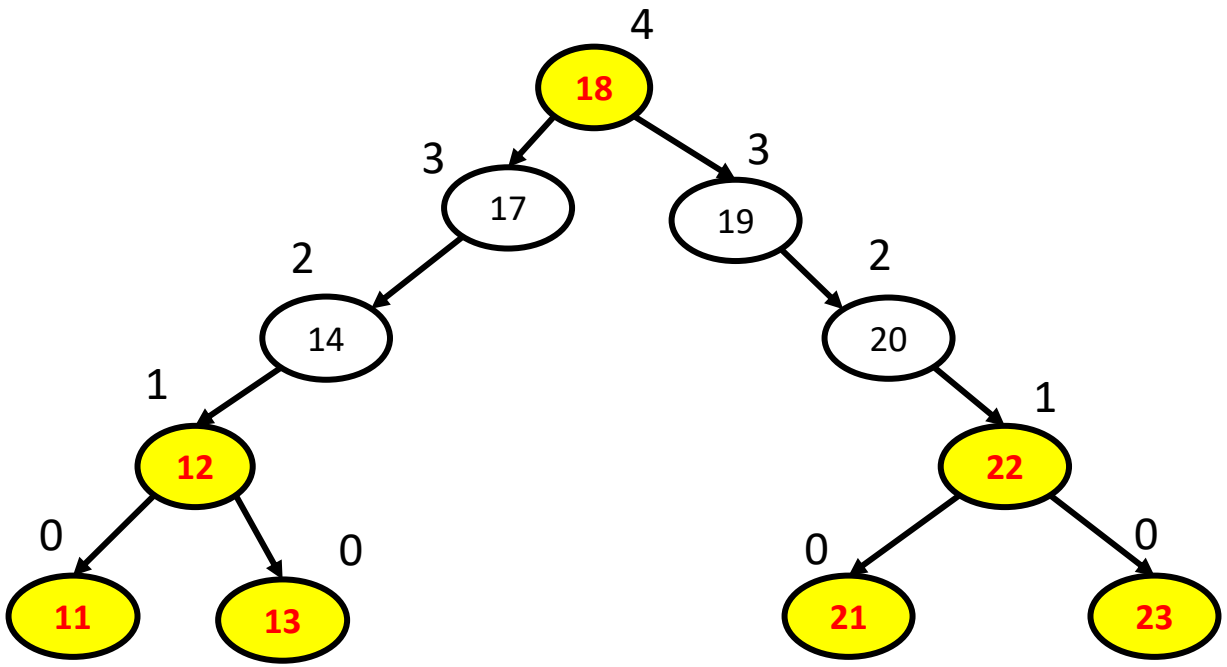
на рис. показана нумерация вершин при внутреннем обходе (красные числа):

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

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

Сначала нужно определить нужные вершину, узнать чётно или нет их количество. Если нечётно, то средняя вершина существует и её можно найти, просматривая нужные вершины, например, по неубыванию 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 средней не существует.
3. Если **m** – нечётно, 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**.

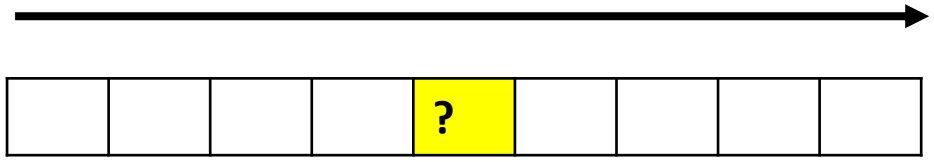


Внутренний (левый) обход:

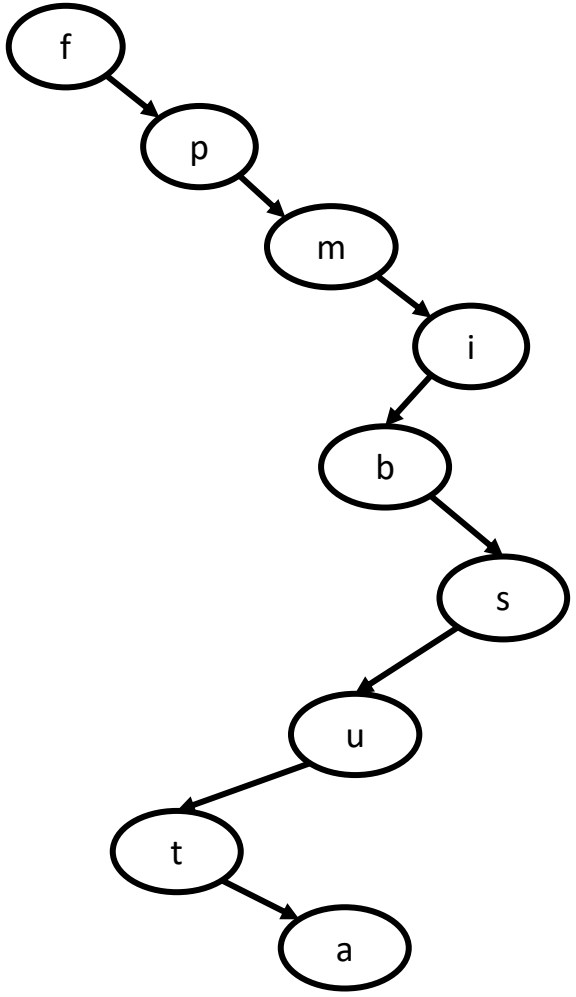
11, 12, 13, 14, 17, ***18***, 19, 20, **21, 22, 23**

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

8) Проверить, является ли бинарное дерево поисковым:

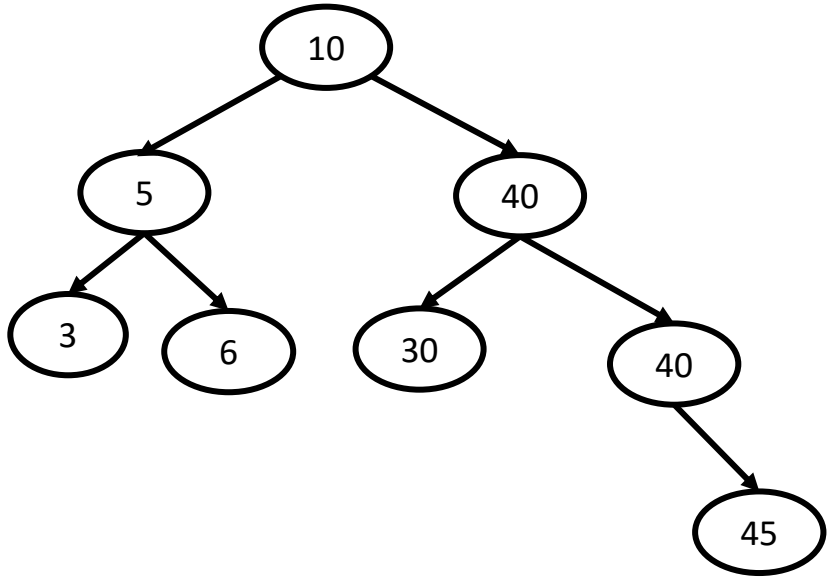
- допускаются одинаковые элементы, которые при добавлении идут в правое поддерево;
- на входе

  в первой строке указывается количество вершин,

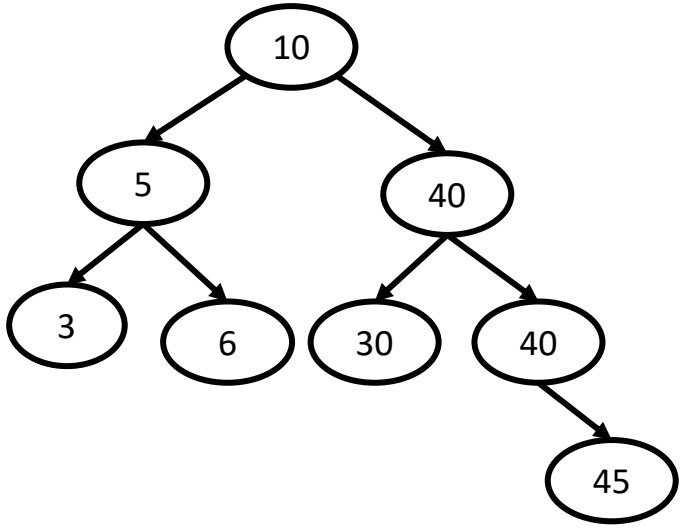
  во второй – ключ корня;

  далее, каждая строка содержит информацию о вершине v: ключ, L или R сыном своего отца она является, а также номер строки входного файла, в которой располагалась информация об отце вершины v;
- гарантируется, что бинарное дерево задано корректно и для каждого элемента информация об отце поступила раньше;

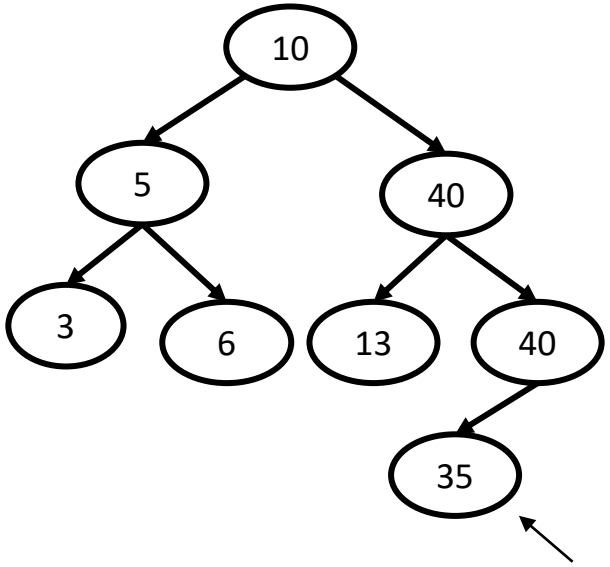
0-я строка	**8**
1-я строка	**10**
2-я строка	**5** L 1
3-я строка	**40** R 1
4-я строка	**3** L 2
5-я строка	**6** R 2
6-я строка	**30** L 3
7-я строка	**40** R 3
8-я строка	**45** R 7



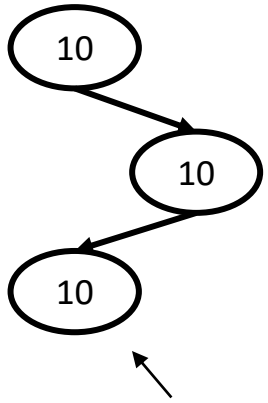
?



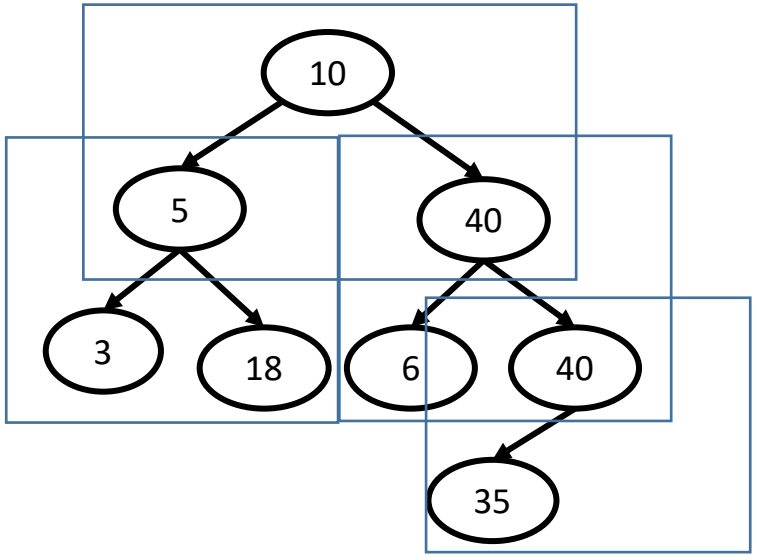
поисковое



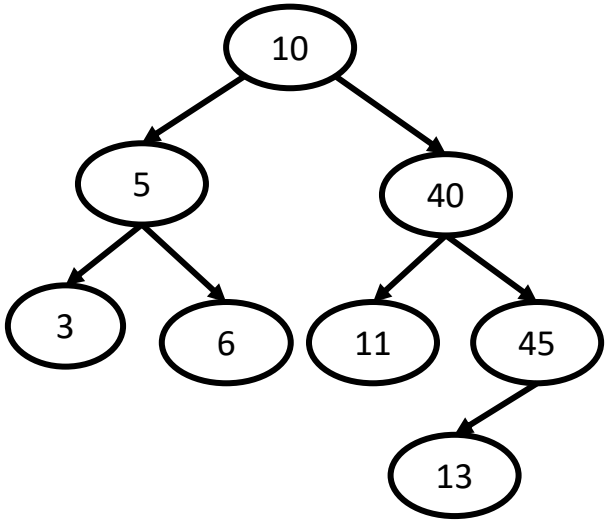
не поисковое



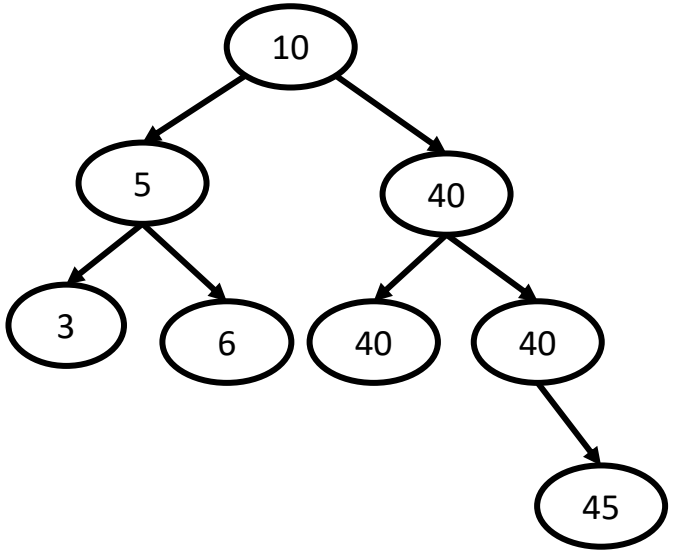
не поисковое



Гарантируется ли поисковость дерева, если для каждой вершины ключ её левого сына строго меньше, чем ключ данной вершины, а ключ её правого сына – не меньше?

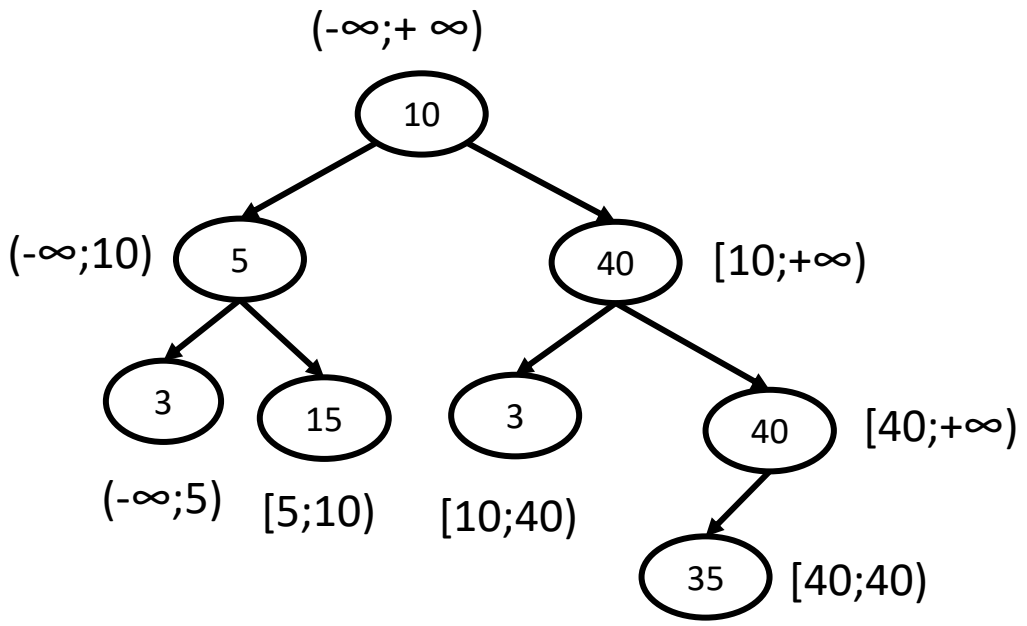


3,5,6,10,11,40,13,45



3,5,6,10,40,40,40,45

Гарантируется ли поисковость дерева, если при выполнении его внутреннего (левого) обхода мы получим неубывающую последовательность ключей?



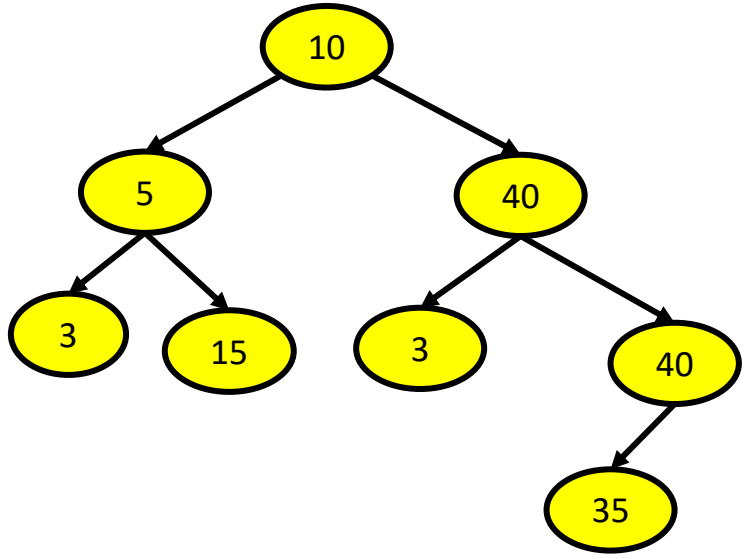
0-я	**8**
1-я	**10**
2-я	**5** L 1
3-я	**40** R 1
4-я	**3** L 2
5-я	**15** R 2
6-я	**3** L 3
7-я	**40** R 3
8-я	**35** L 7

при формировании промежутка допустимых значений для вершины v дерева, необходимо знать промежуток допустимых значений у родителя этой вершины и ключ родителя:

если ключ вершины v не попадает в указанный промежуток, то дерево не является поисковым;

Как представить дерево в памяти компьютера?

0-я	**8**
1-я	**10**
2-я	**5** L 1
3-я	**40** R 1
4-я	**3** L 2
5-я	**15** R 2
6-я	**3** L 3
7-я	**40** R 3
8-я	**35** L 7



номер строки входного файла

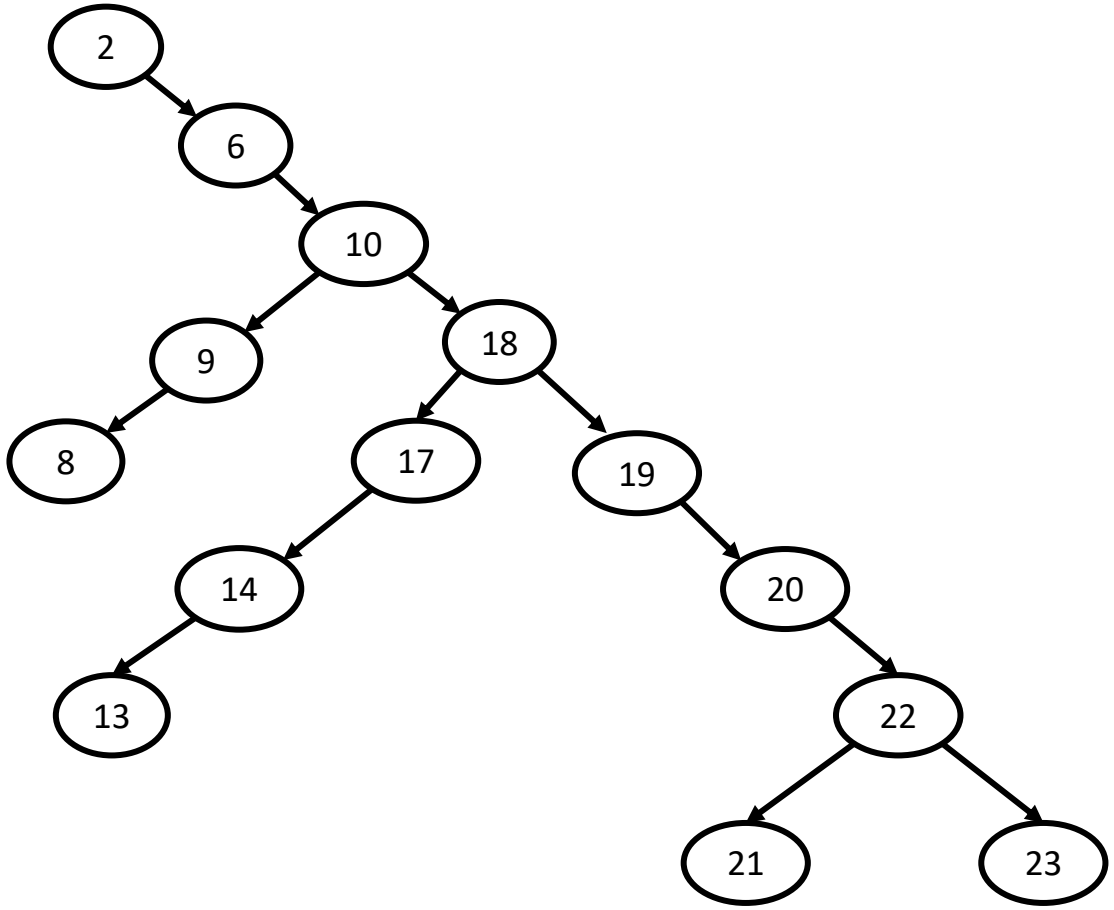
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
10	5	40	3	15	3	40	35
-	L	R	L	R	L	R	L
-	1	1	2	2	3	3	7
(-∞;+ ∞)	(-∞;10)	[10;+∞)	(-∞;5)	[5;10)	[10;40)	[40;+∞)	[40;40)
				ошибка	ошибка		ошибка

Удаление вершины

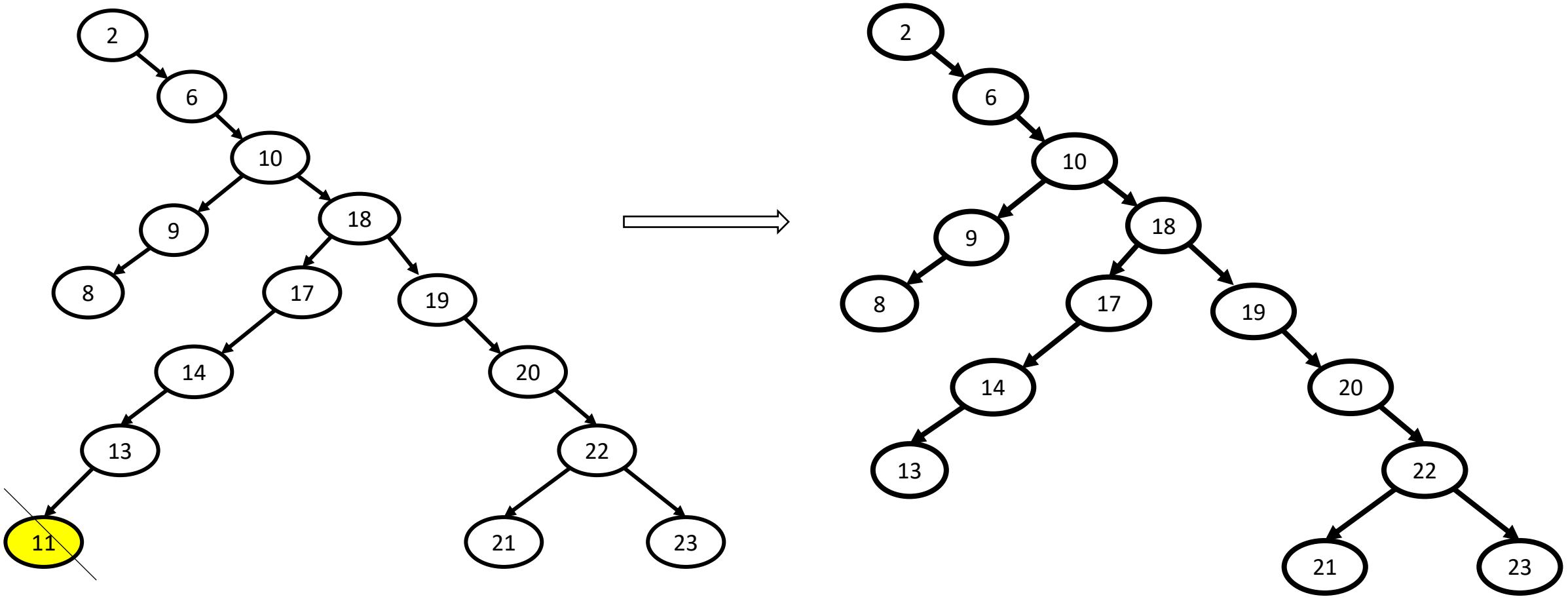
Случай 1. Удаляется лист.

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

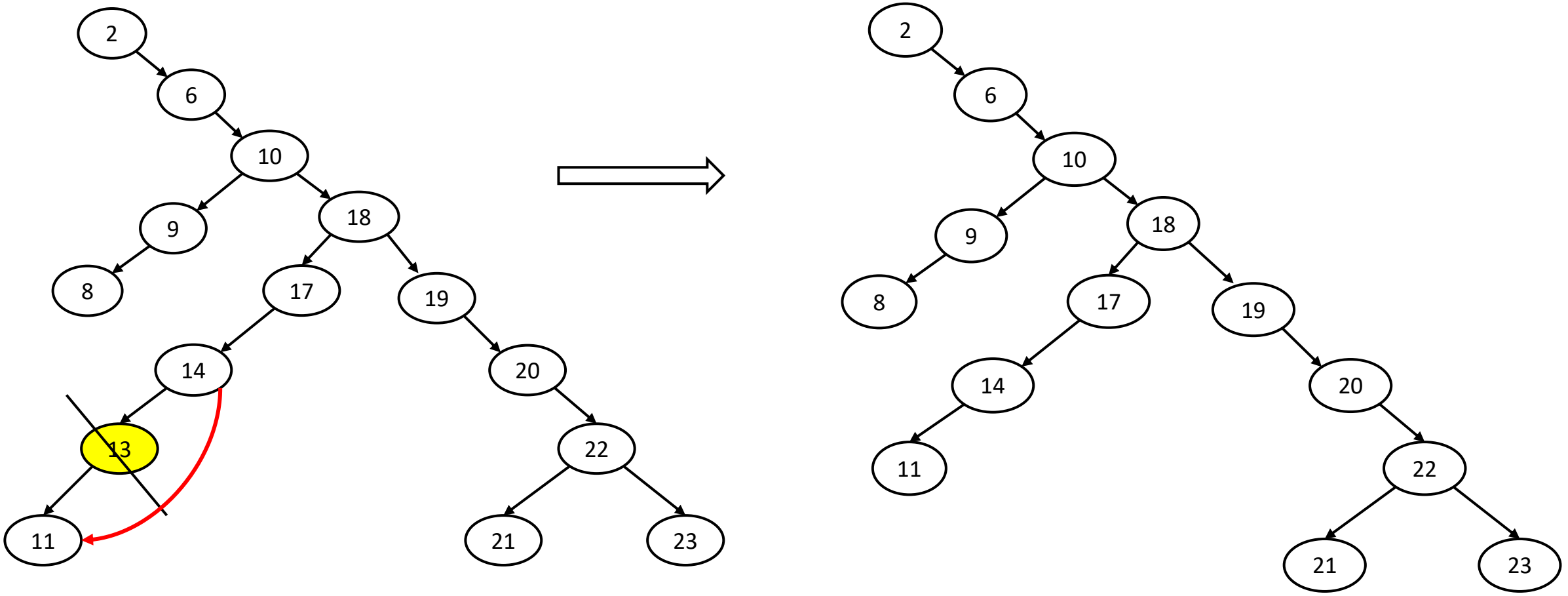
Случай 3. Удаляется вершина, у которой есть оба поддерева (возможно «правое» или «левое» удаление).



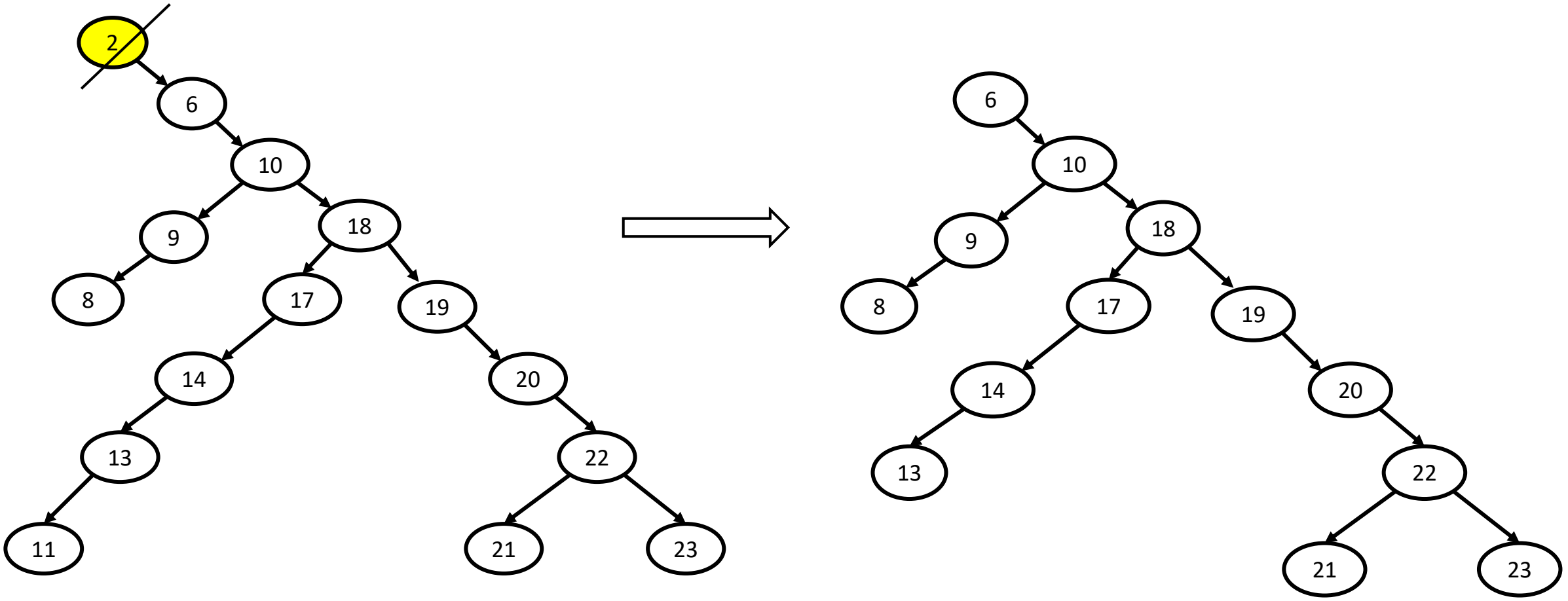
Случай 1. Удаляется лист.



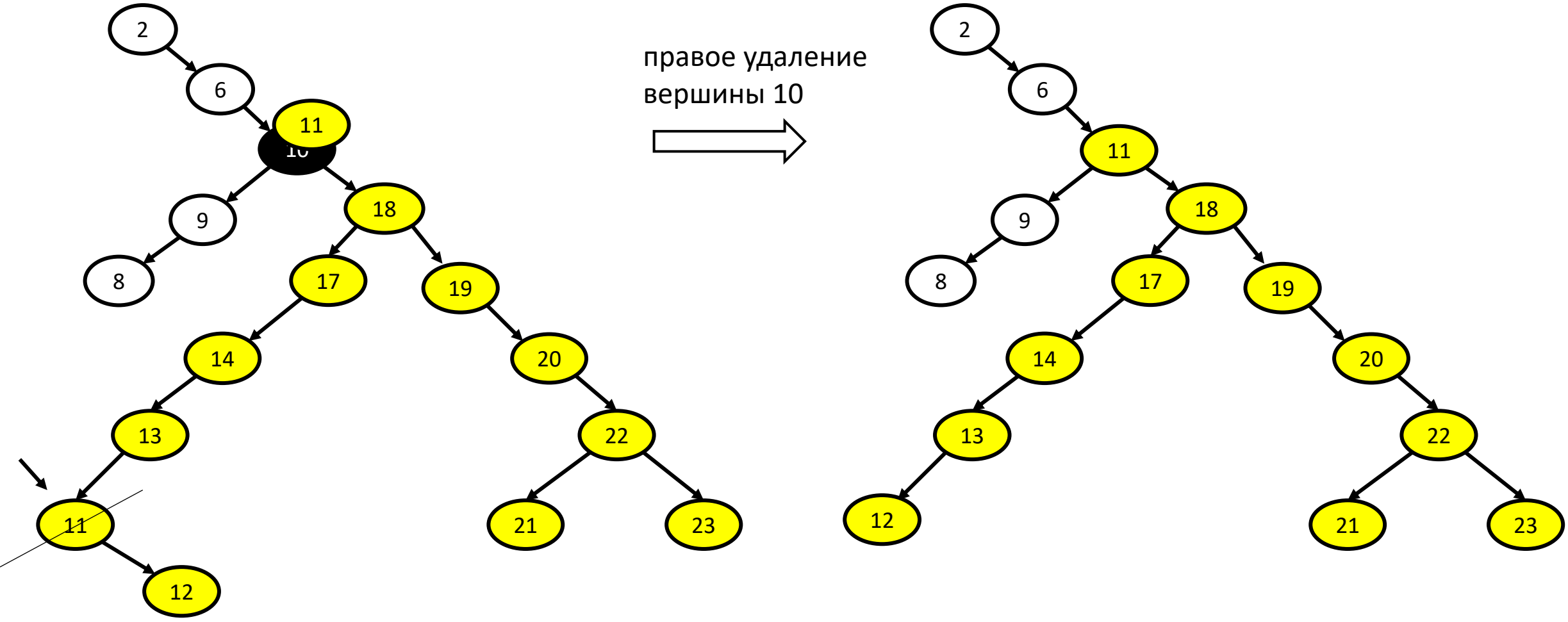
Случай 2. Удаляется вершина, у которой есть только одно поддерево.



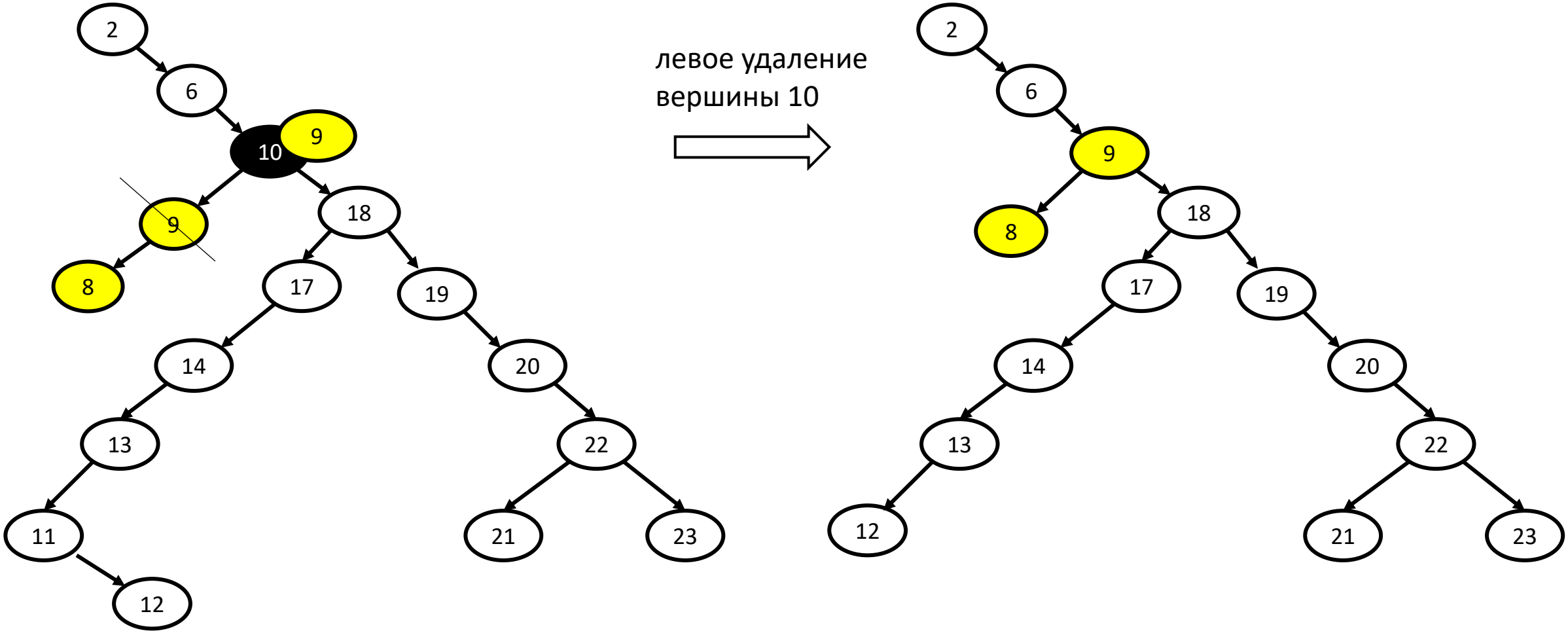
Случай 2. Удаляется вершина, у которой есть только одно поддерево



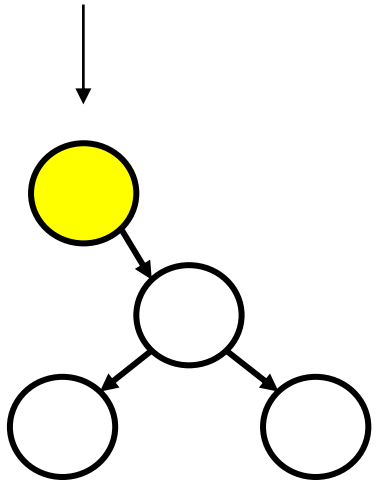
Случай 3. Удаляется вершина, у которой есть оба поддерева («**правое**» **удаление**).



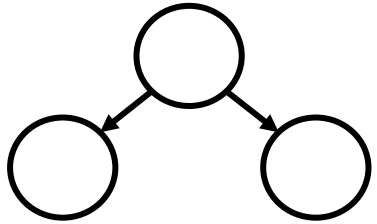
Случай 3. Удаляется вершина, у которой есть оба поддерева («**левое**» **удаление**).



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

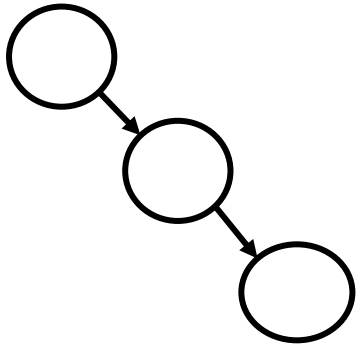


?



(рис.1)

или

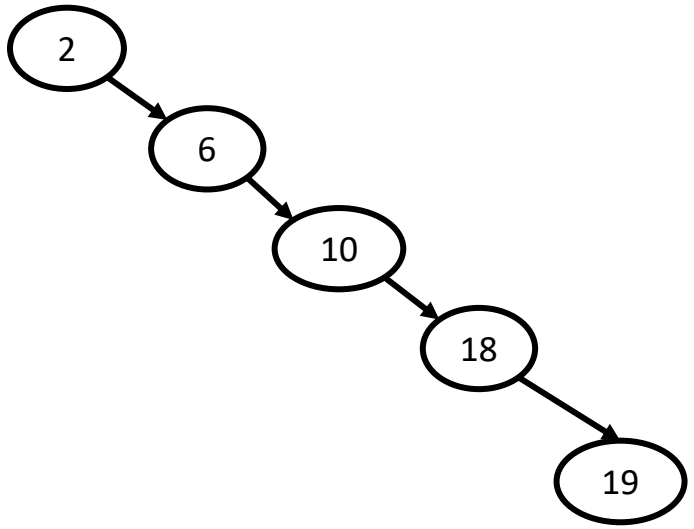


(рис.2)

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

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	$n \cdot h$
удаление элемента с заданным ключом x	h
обход дерева из n вершин	n

Литература по теме: «Бинарные поисковые деревья»

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 в iRunner

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

Индивидуальная задача по теме «Деревья поиска» в iRunner

Спасибо за внимание!