



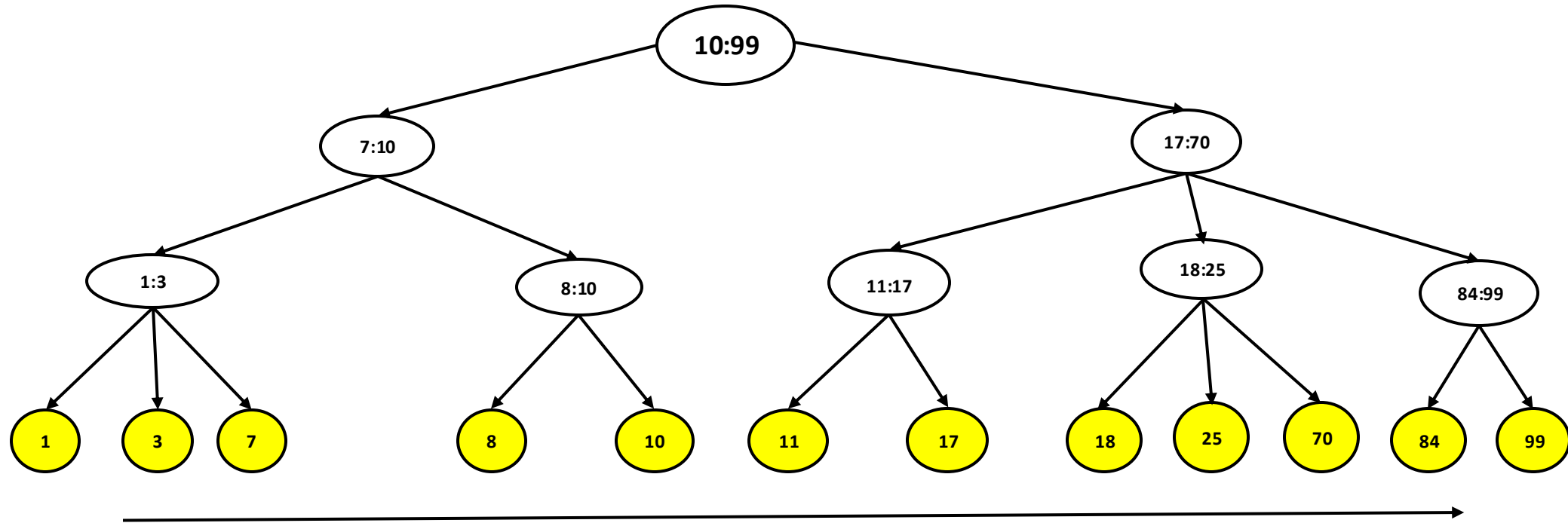
ОРГАНИЗАЦИЯ ПОИСКА

СБАЛАНСИРОВАННЫЕ ПОИСКОВЫЕ ДЕРЕВЬЯ

2-3 дерева

Поисковое дерево называется **2-3-деревом**, если оно обладает следующими свойствами:

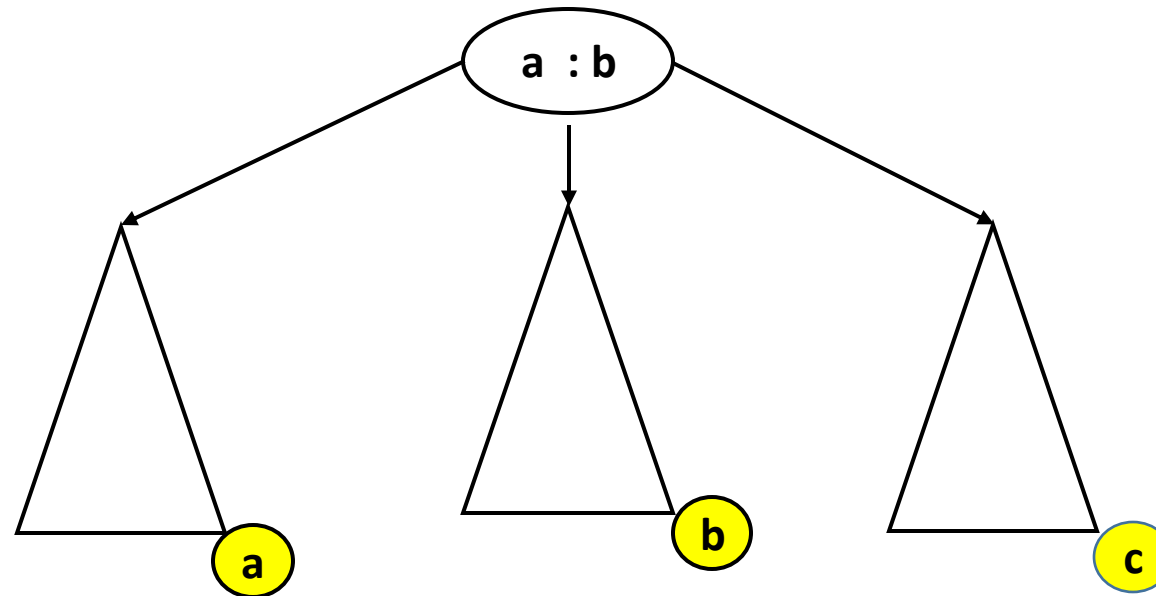
- 1) каждая вершина x , не являющаяся листом, содержит два или три сына (левый $l(x)$, средний $t(x)$ и (возможно) правый сын $r(x)$);
- 2) все висячие вершины находятся на одной глубине и содержат сами элементы;
- 3) внутренние вершины – справочные (внутренние – это все вершины дерева за исключением листьев).



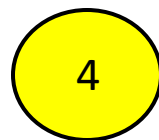
так как дерево поисковое, то ключи всех листьев идут слева направо по возрастанию

Каждая внутренняя вершина 2-3 –дерева является справочной и содержит две метки:

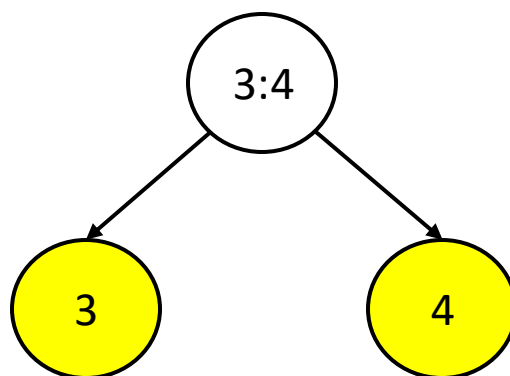
1. $a = \text{left_max_val}(v)$ – максимальное значение ключа в поддереве, корень которого – левый сын вершины v ;
2. $b = \text{midl_max_val}(v)$ – максимальное значение ключа в поддереве, корень которого – средний сын вершины v ;



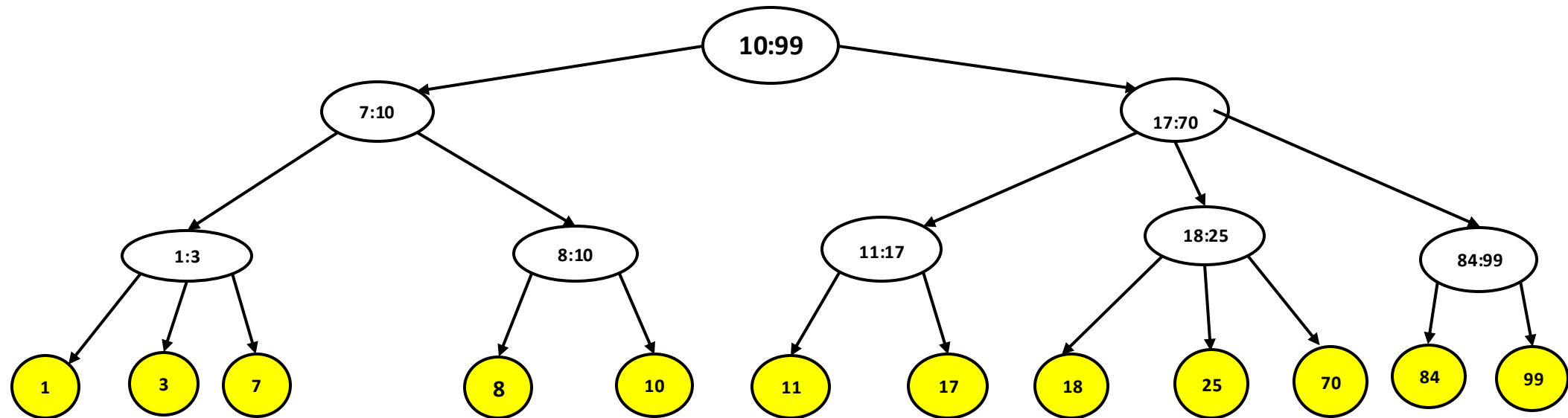
Если в 2-3-дереве только один лист, (например, 4), то это дерево имеет следующий вид:



Если в 2-3-дереве только два листа (например, 3 и 4), то это дерево имеет следующий вид:



Пример



ТЕОРЕМА

Пусть

n – общее количество вершин в 2-3-дереве (включая корень и листья);

l – количество листьев;

h – высота дерева.

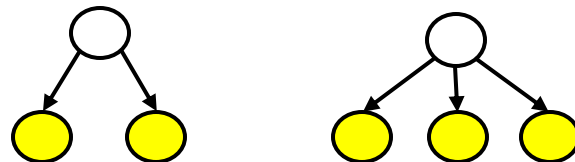
Тогда справедливы следующие неравенства:

$$2^h \leq l \leq 3^h$$

$$2^{h+1} - 1 \leq n \leq \frac{3^{h+1} - 1}{2}$$

Доказательство проводится, используя метод математической индукции.

База индукции: $h = 1$. Утверждение верно.



Предположим, что теорема верна для деревьев высоты h и докажем её для деревьев высоты $h + 1$.

$$(1) \quad 2^h \leq l \leq 3^h$$

$$(2) \quad 2^{h+1} - 1 \leq n \leq \frac{3^{h+1} - 1}{2}$$

Сначала докажем первое неравенство.

При увеличении высоты дерева на 1 минимальное число листьев в дереве получаем, когда в дереве высоты h с минимальным числом листьев к каждому листу добавлено по 2 сына, а максимальное, когда в дереве высоты h с максимальным числом листьев к каждому листу добавлено 3 сына:

$$\begin{aligned} 2 \cdot l_h^{\min} &\leq l_{h+1} \leq 3 \cdot l_h^{\max} \\ 2 \cdot 2^h &\leq l_{h+1} \leq 3 \cdot 3^h \end{aligned}$$

Первое неравенство доказано.

Докажем второе неравенство.

Предположим, что неравенство (2) выполняется

для деревьев высоты h и докажем его для деревьев высоты $h + 1$.

Пусть n_h – общее число вершин в дереве высоты h .

Тогда

$$n_h^{\min} + l_{h+1}^{\min} \leq n_{h+1} \leq n_h^{\max} + l_{h+1}^{\max}$$

$$(2^{h+1}-1) + 2^{h+1} = 2^{h+2} - 1$$

$$\frac{3^{h+1}-1}{2} + 3^{h+1} = \frac{3^{h+1}+2 \cdot 3^{h+1}-1}{2} = \frac{3^{h+2}-1}{2}$$

Неравенство (2) также доказано.

$$(1) \quad 2^h \leq l \leq 3^h$$

$$(2) \quad 2^{h+1} - 1 \leq n \leq \frac{3^{h+1} - 1}{2}$$

Из левой части неравенства:

$$2^{h+1} - 1 \leq n \leq \frac{3^{h+1} - 1}{2}$$

получаем, что высота 2-3-дерева из n вершин

$$\mathbf{h \leq \log_2(n + 1) - 1.}$$

Поиск элемента с ключом x

Двигаемся от корня.

Пусть t – текущая вершина.

повторять, пока t - не лист

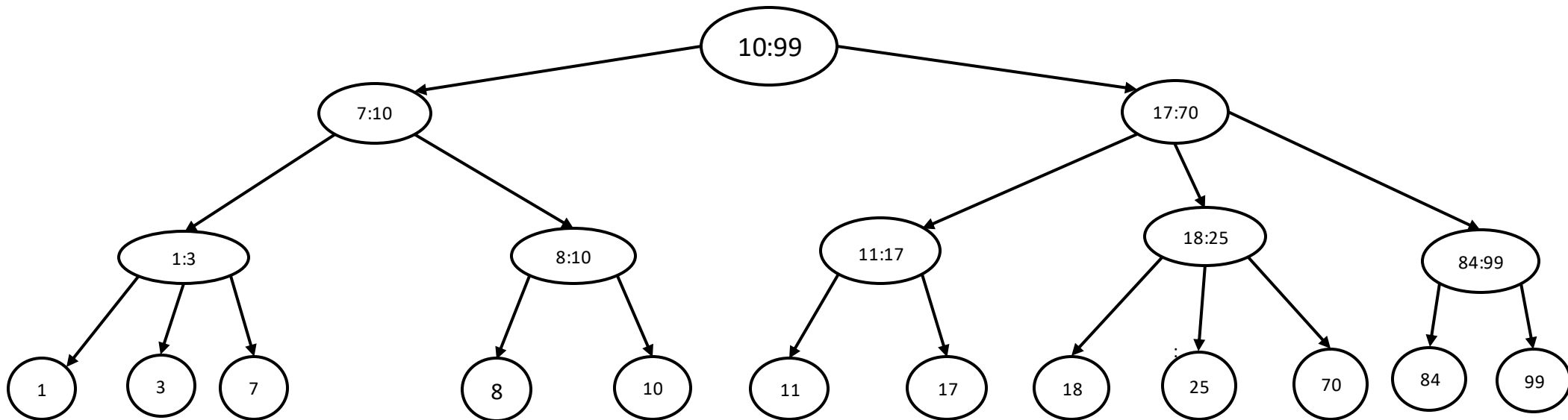
если $x \leq \text{left_max_val}$ то $t = l(t)$

иначе

если $x \leq \text{midl_max_val}$ или $r(t) = \text{NULL}$ то $t = m(t)$

иначе

$t = r(t)$



повторять, пока t - не лист

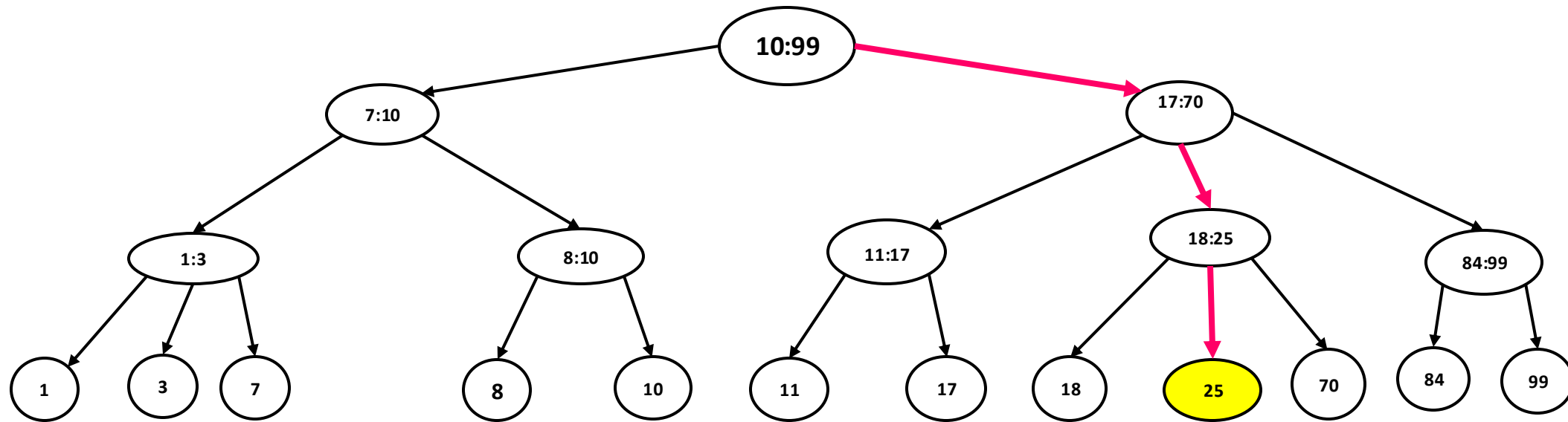
если $x \leq \text{left_max_val}$ то $t = l(t)$

иначе

если $x \leq \text{midl_max_val}$ или $r(t) = \text{NULL}$ то $t = m(t)$

иначе

$t = r(t)$



? 9

повторять, пока t - не лист

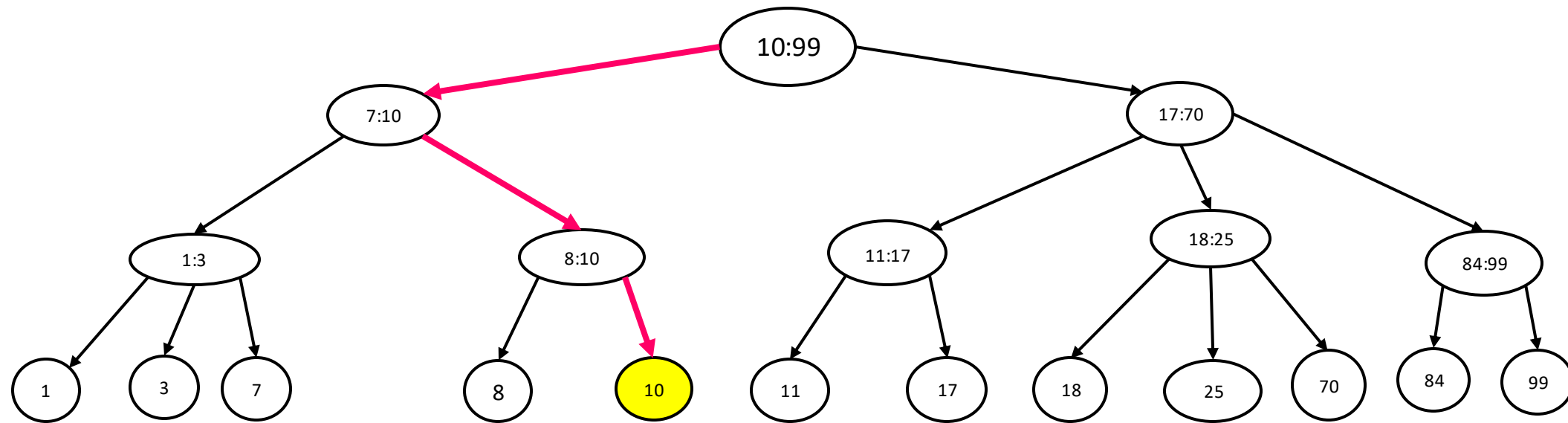
если $x \leq \text{left_max_val}$ то $t = l(t)$

иначе

если $x \leq \text{midl_max_val}$ или $r(t) = \text{NULL}$ то $t = m(t)$

иначе

$t = r(t)$



? = 9
НЕТ

Добавление элемента Insert (13)

повторять, пока t - не лист

если $x \leq \text{left_max_val}$ то $t = l(t)$

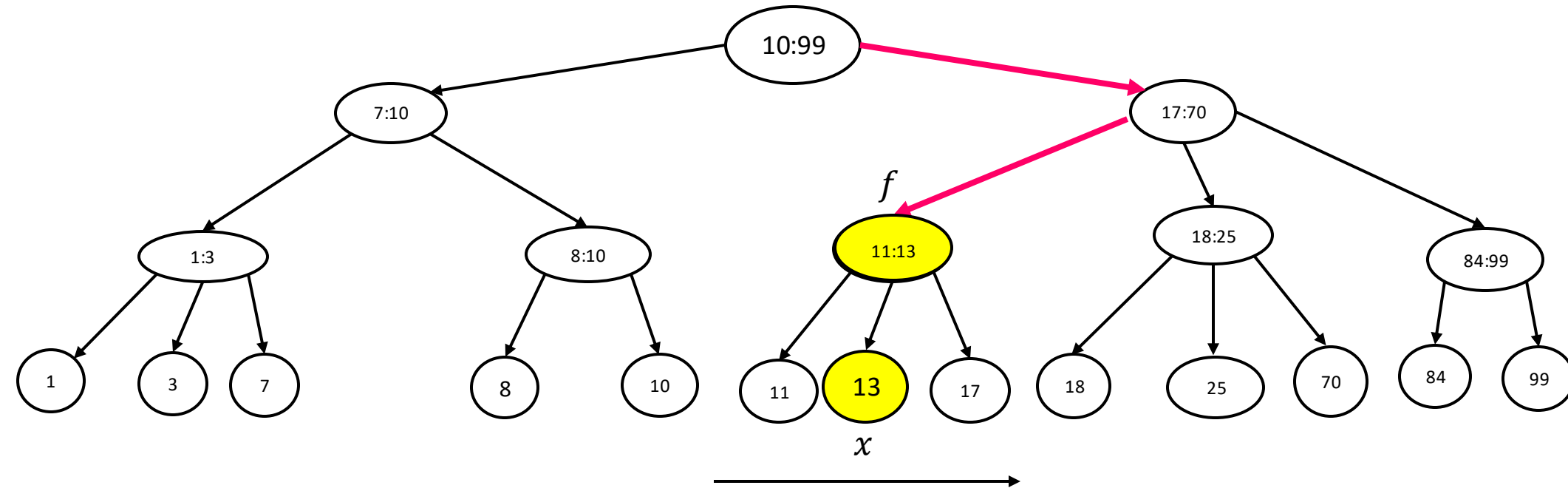
иначе

если $x \leq \text{midl_max_val}$ или $r(t) = \text{NULL}$ то $t = m(t)$

иначе

$t = r(t)$

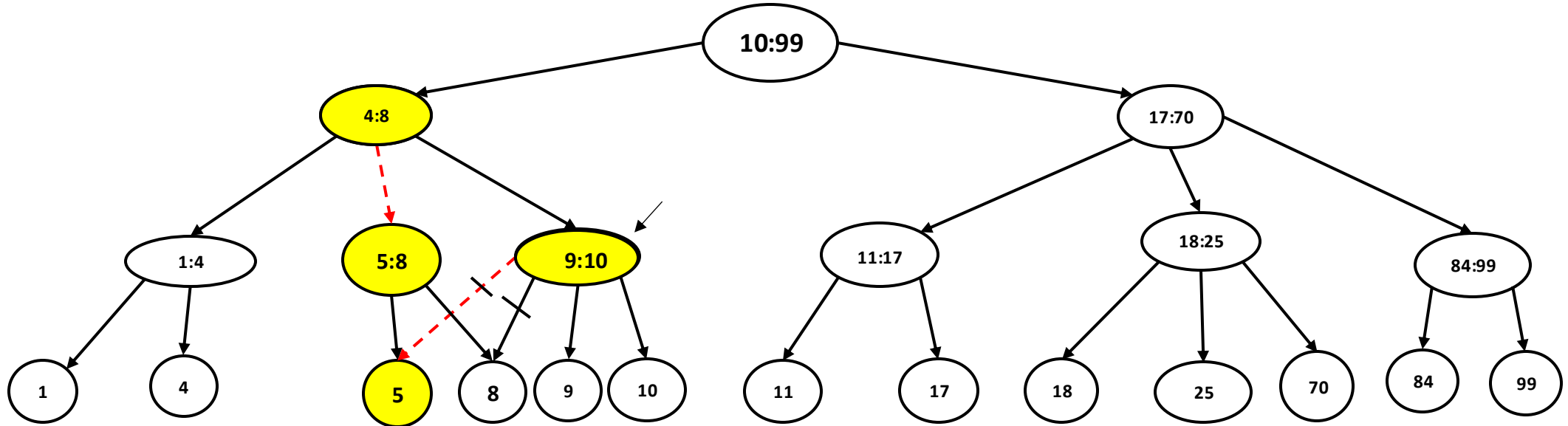
Сначала осуществляем поиск отца f для добавляемого элемента x (в качестве f берём отца вершины t):



- ✓ у вершины f после добавления становится 3 сына, что допустимо;
- ✓ корректируем метки справочных вершин вдоль пути поиска;
- ✓ завершаем процедуру добавления элемента x ;

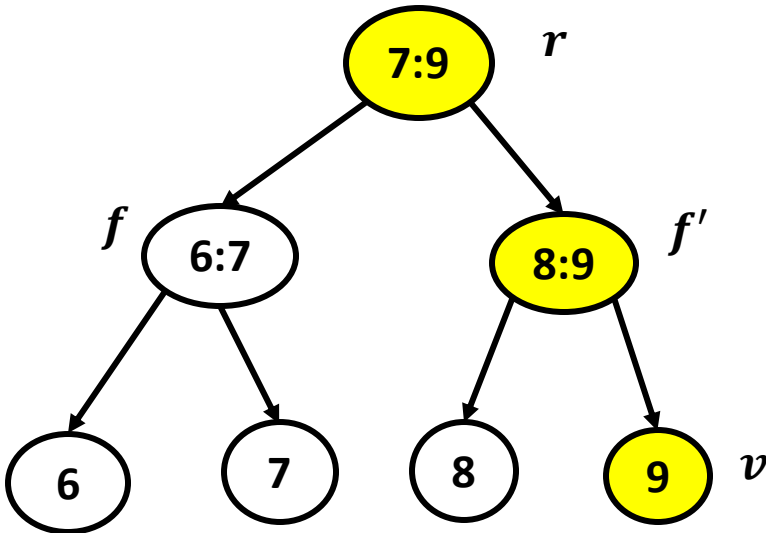
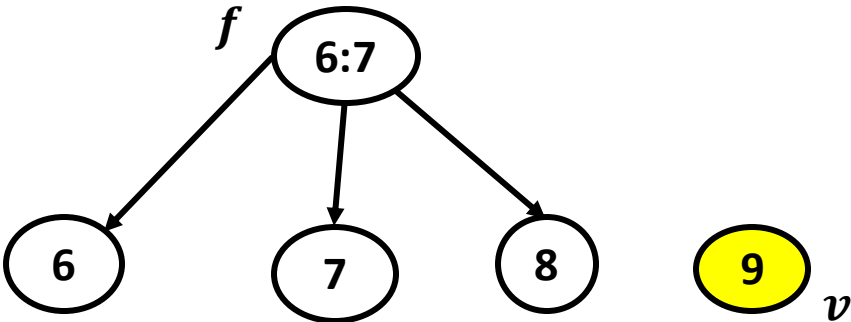
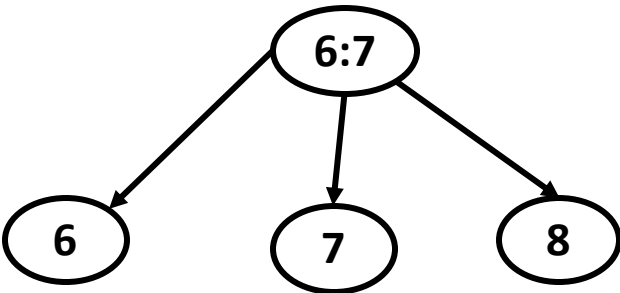
Insert (5)

повторять, пока t - не лист
если $x \leq left_max_val$ то $t = l(t)$
иначе
если $x \leq midl_max_val$ или $r(t) = NULL$ то $t = m(t)$
иначе
 $t = r(t)$



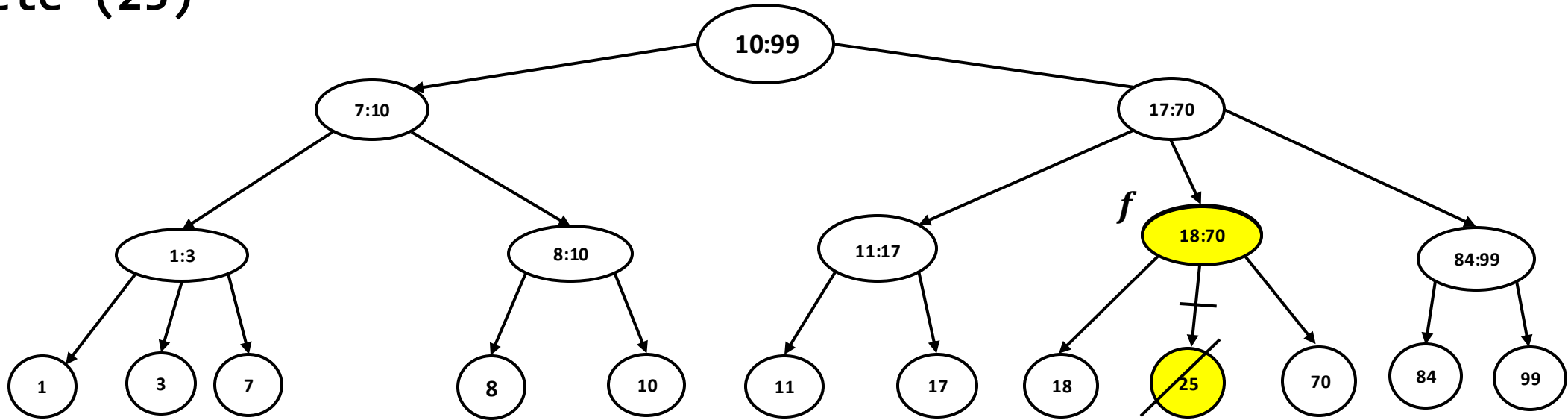
Случай увеличения высоты дерева после добавления элемента.

Insert (9)



Удаление элемента

Delete (25)



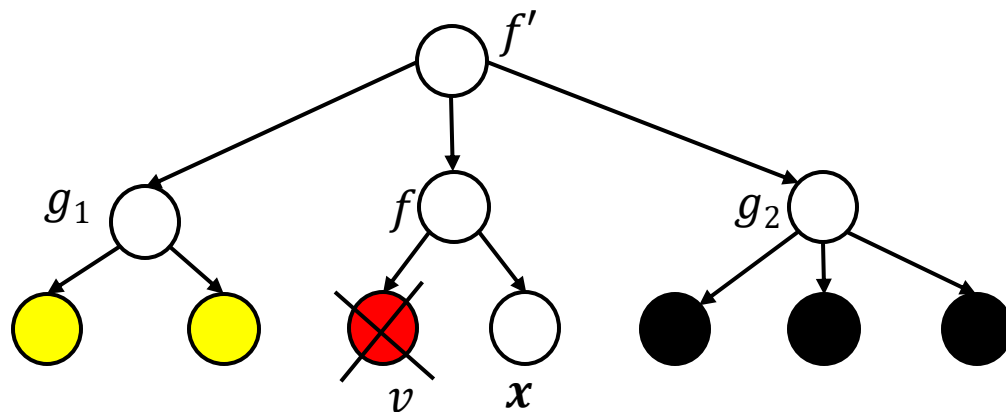
Удаляем вершину.

Если у отца f удаляемой вершины останется 2 сына, то это допустимо для 2-3-дерева.

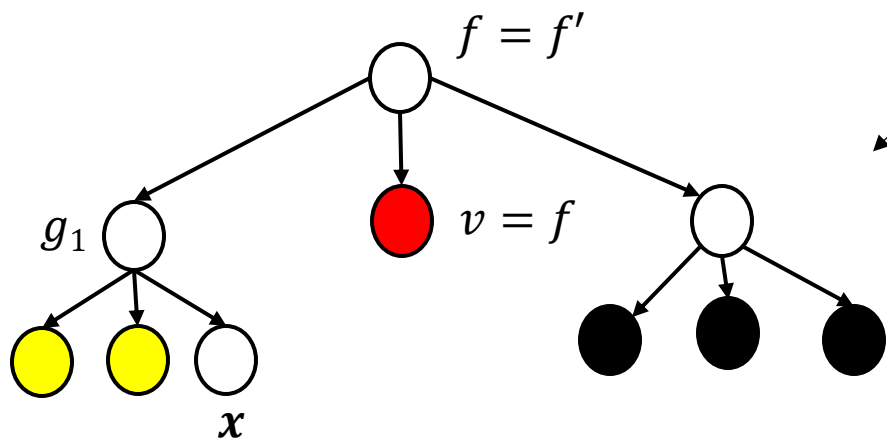
Корректируем метки справочных вершин вдоль пути поиска.

Завершаем процедуру удаления.

случай, когда у отца f
удаляемой вершины v останется
только один сын x :

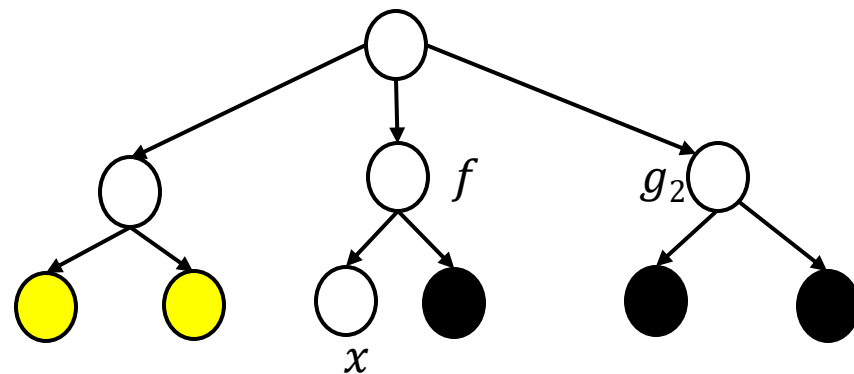


или



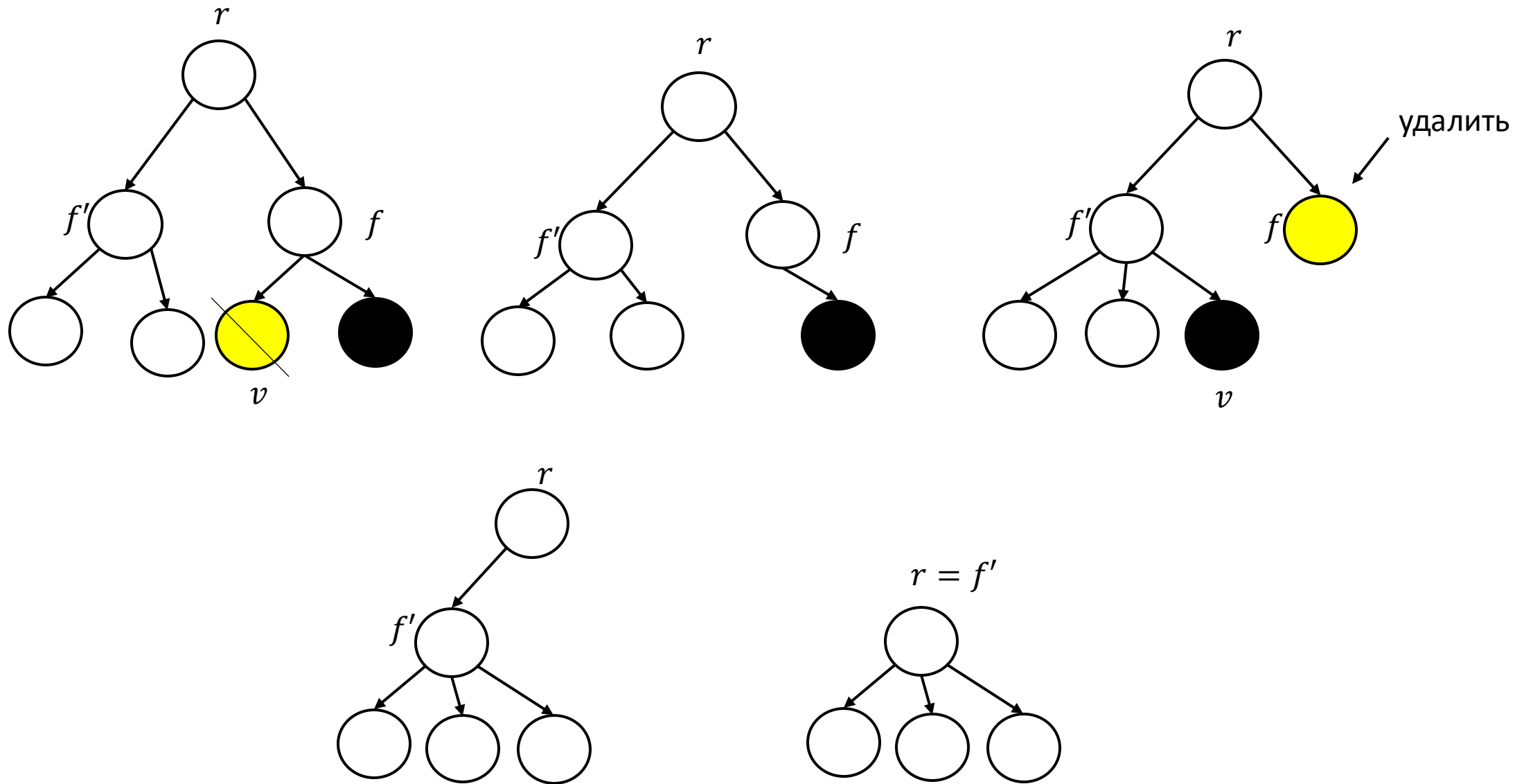
рекурсивно продолжаем удаление:

$$\begin{aligned} v &= f \\ f &= f' \end{aligned}$$



корректируем метки на пути от корня до f ;
удаление завершено;

После удаления вершины v высота дерева может уменьшиться на 1:



Оценки

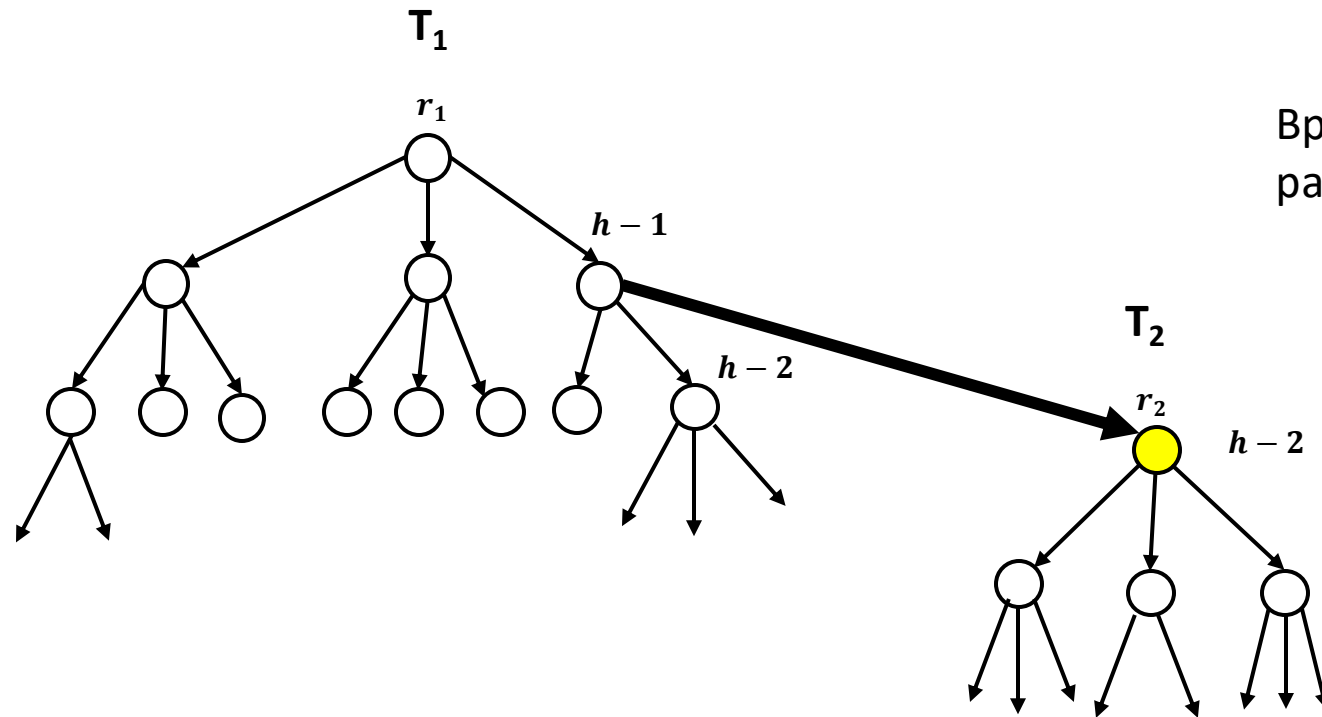
1. Поиск элемента
2. Добавление элемента
3. Удаление элемента

} **$O(\log n)$**

Важными дополнительными операциями, которые можно эффективно выполнять для 2-3-дерева являются:

- ✓ **Join** (T_1, T_2) – объединение двух 2-3-деревьев, при условии, все ключи в дереве T_1 меньше, чем ключи в дереве T_2
- ✓ **Split** (x) – разделение дерева T по ключу x на два дерева T_1 и T_2 , при этом ключи всех вершин в дереве T_1 меньше x , а в дереве T_2 – больше x

✓ **Join (T1,T2)** – объединение двух 2-3-деревьев, при условии, что все ключи в дереве T_1 меньше, чем ключи в дереве T_2

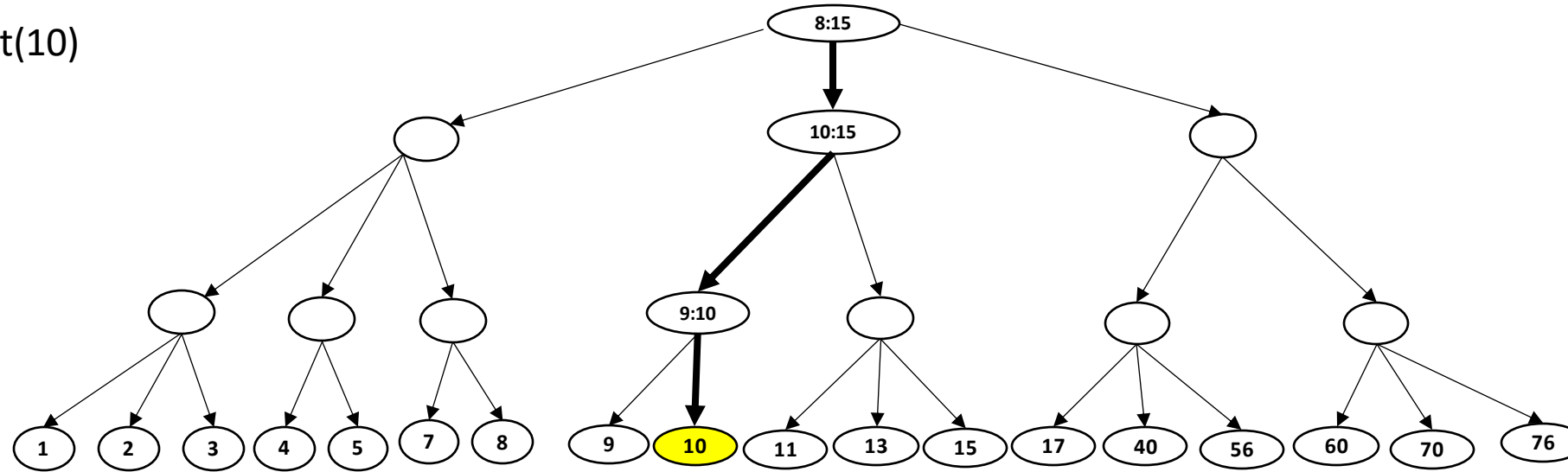


Время работы пропорционально модулю
разности высот объединяемых деревьев:

$$|h(T_1) - h(T_2)|$$

- ✓ **Split (x)** – разделение дерева T по ключу x на два дерева T_1 и T_2 , при этом ключи всех вершин в дереве T_1 меньше ключа x , а в дереве T_2 – больше x

Split(10)

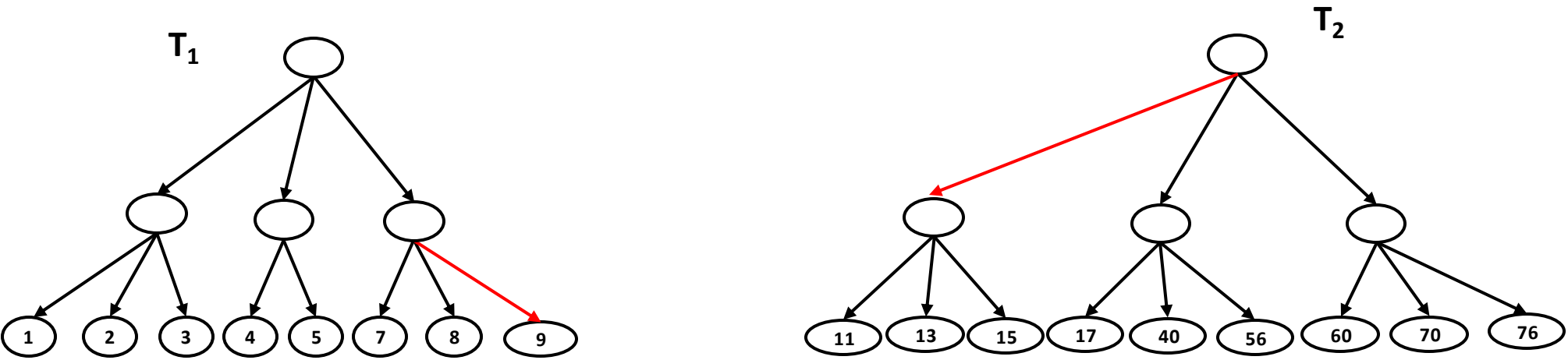
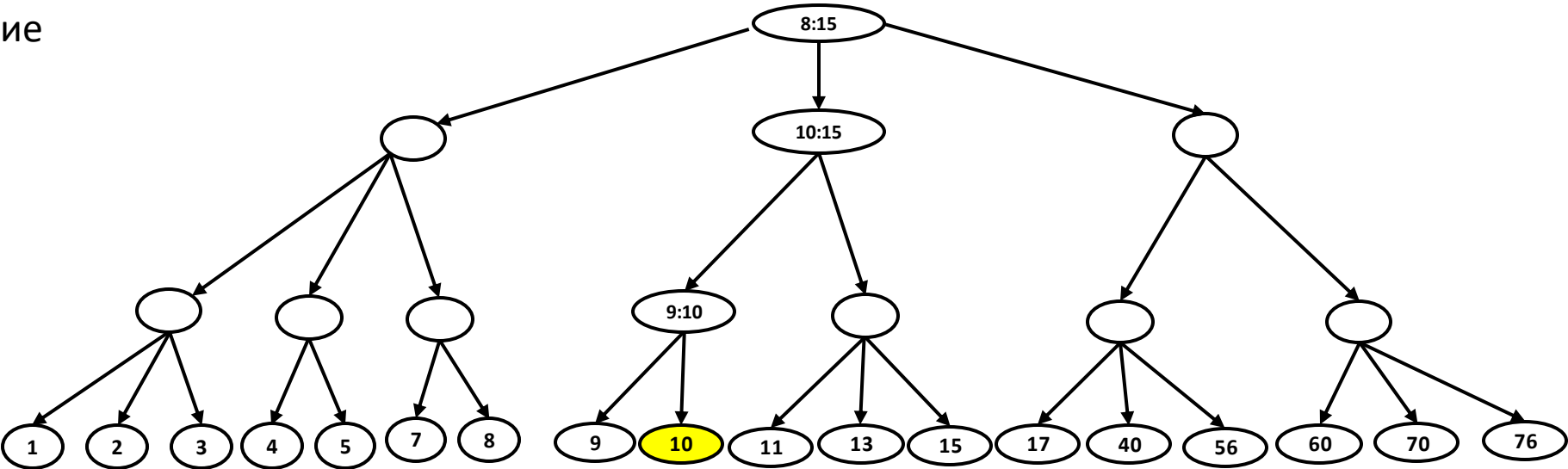


- количество деревьев в каждой из полученных частей не превосходит $-\log n$
- при слиянии деревьев в левой (правой) частях будем всегда выполнять процедуру **Join** над деревьями меньшей высоты, тогда время, затраченное на построение каждого из деревьев T_1 и T_2 - $O(\log n)$

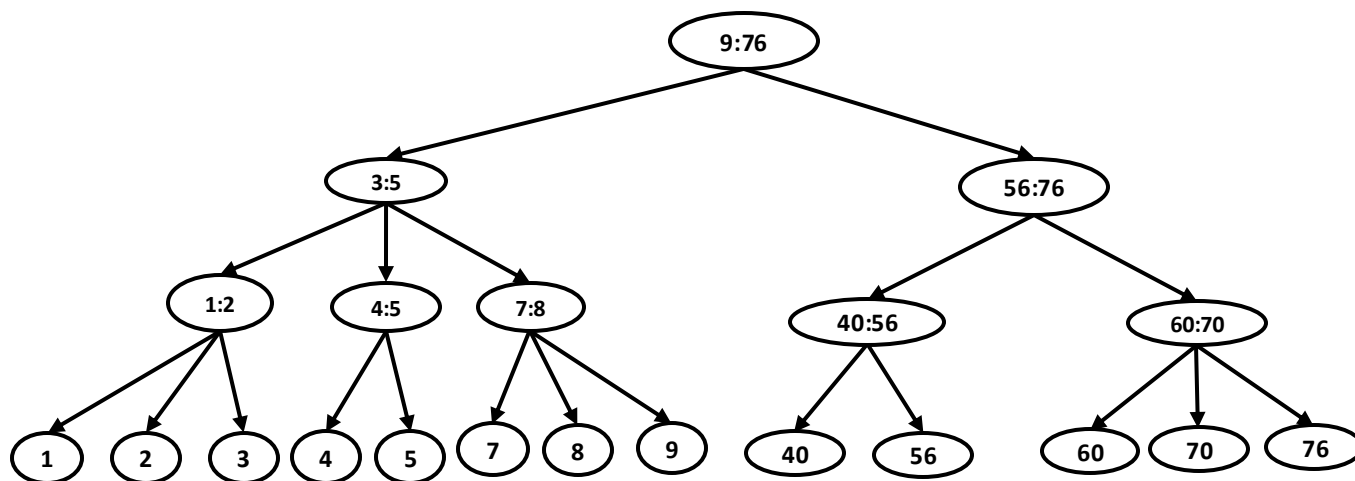
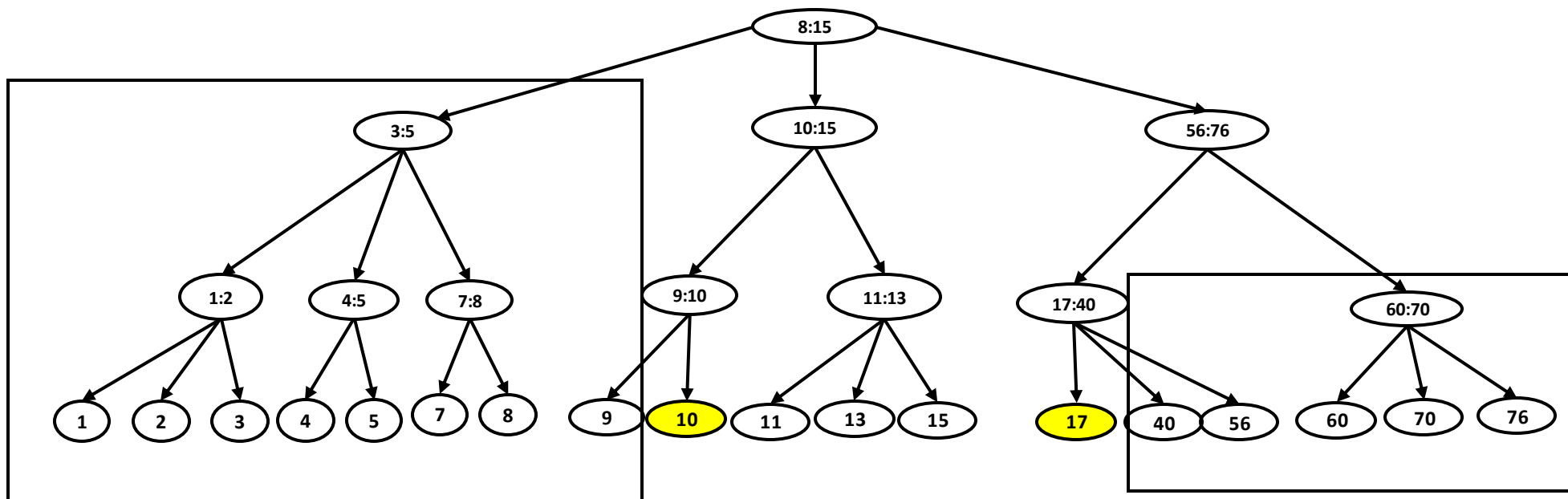
Время выполнения **Split (x)**
 $O(\log n)$

продолжение

Split (10)



Удаление из дерева непрерывного участка
ключей, лежащих в интервале $[a, b]$



Время работы: $O(\log n)$

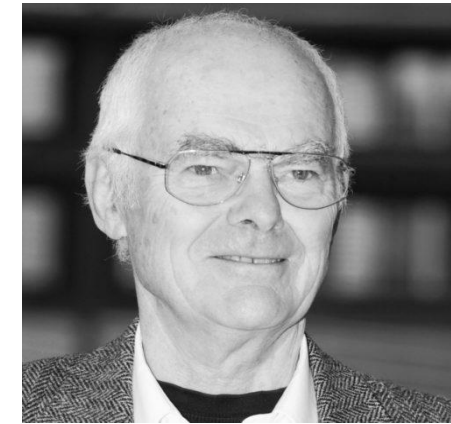
не зависит от размера удаляемого участка

В-дерево

произносится: «би»-дерев)
(англ. B-tree)



В 1970 году Р.Байером и Э.Маккрейтом разработана структура данных **сбалансированного по высоте сильно ветвящегося поискового дерева** (степень ветвления – количество сыновей у вершины на практике, как правило, от 50 до 2000), которое в последствии получило название **В-дерева**.



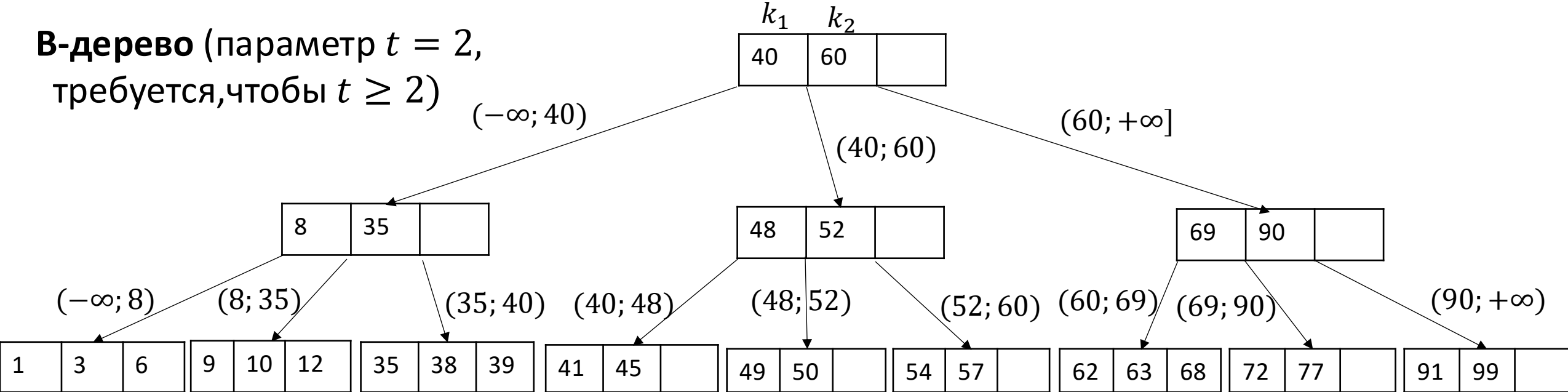
Рудольф Байер
[нем.](#) *Rudolf Bayer*
родился в 1939
немецкий
учёный

На практике структура данных используется при обработке больших данных, хранящихся во внешней памяти. Размер вершины В-дерева обычно соответствует размеру дисковой страницы. Большая степень ветвления снижает высоту дерева, что в свою очередь снижает число обращений к внешней памяти, которое необходимо для выполнения операций с В-деревом.



Эдвард Маккрейт
[англ.](#) *Edward M. McCreight*;
родился в 1941
американский
учёный

B-дерево (параметр $t = 2$,
требуется, чтобы $t \geq 2$)



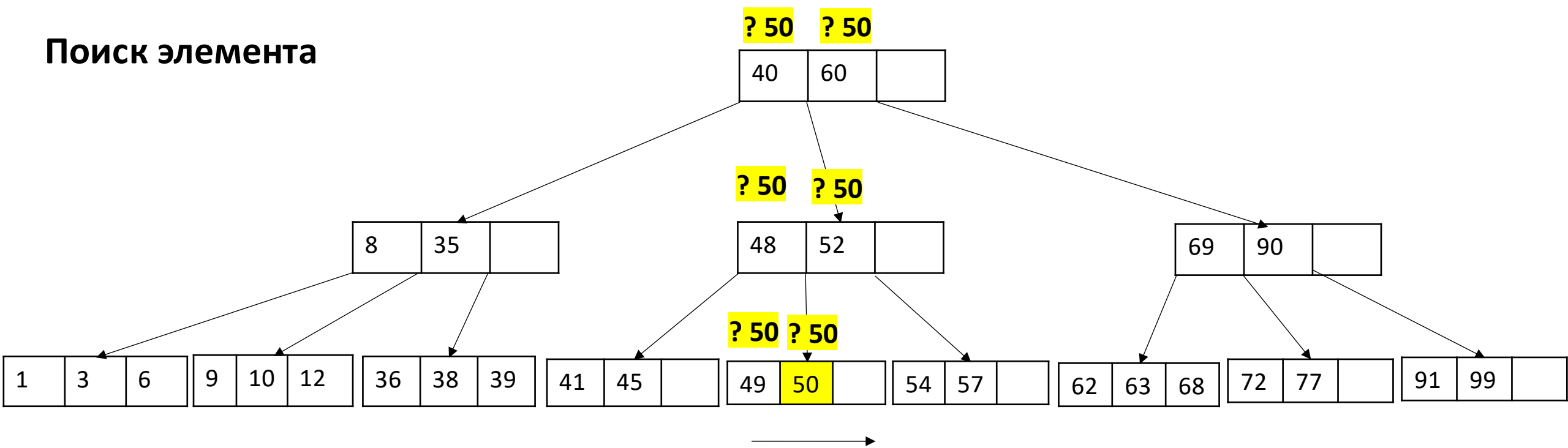
- ✓ корень дерева содержит от **1** до **$(2t - 1)$** ключей, которые располагаются в порядке возрастания;
- ✓ каждая некорневая внутренняя вершина B -дерева порядка $t \geq 2$ имеет от **$t - 1$** до **$2t - 1$ ключей**, которые располагаются в порядке **возрастания**;
- ✓ если внутренняя вершина содержит **n** ключей, то она имеет **$(n + 1)$** потомка (у листьев потомков нет);
- ✓ все листья имеют одинаковую глубину;
- ✓ если k_1, k_2, \dots, k_n последовательность ключей, хранящихся в некоторой вершине, то все ключи первого поддерева потомков лежат в диапазоне $(-\infty, k_1)$, второго поддерева — (k_1, k_2) , третьего — (k_2, k_3) , ..., $(n + 1) - (k_n, +\infty)$.

Пусть В-дерево с параметром $t \geq 2$ содержит n вершин, тогда для его высоты справедливо следующее неравенство:

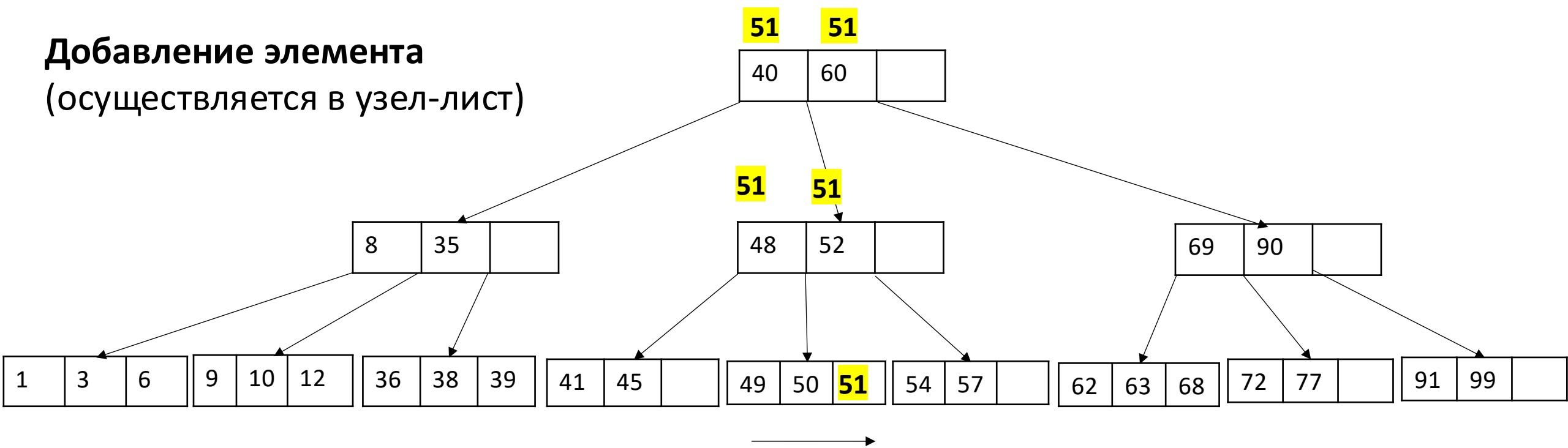
$$h \leq \log_t \frac{n + 1}{2}$$



Поиск элемента



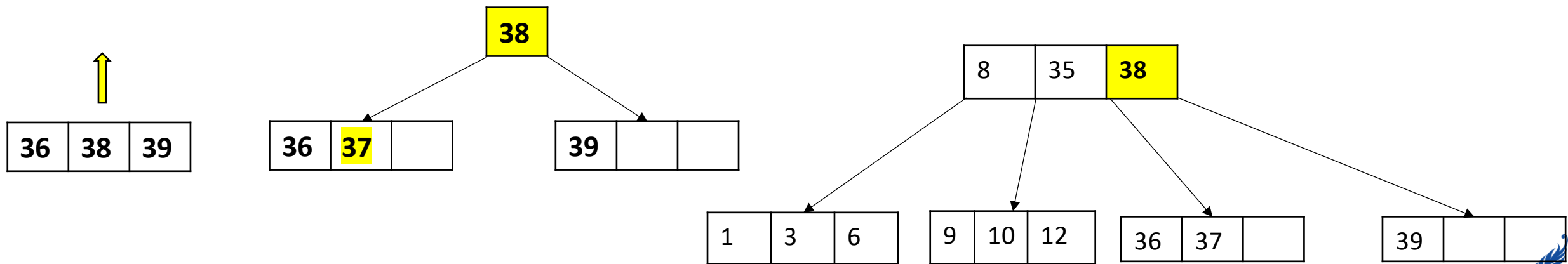
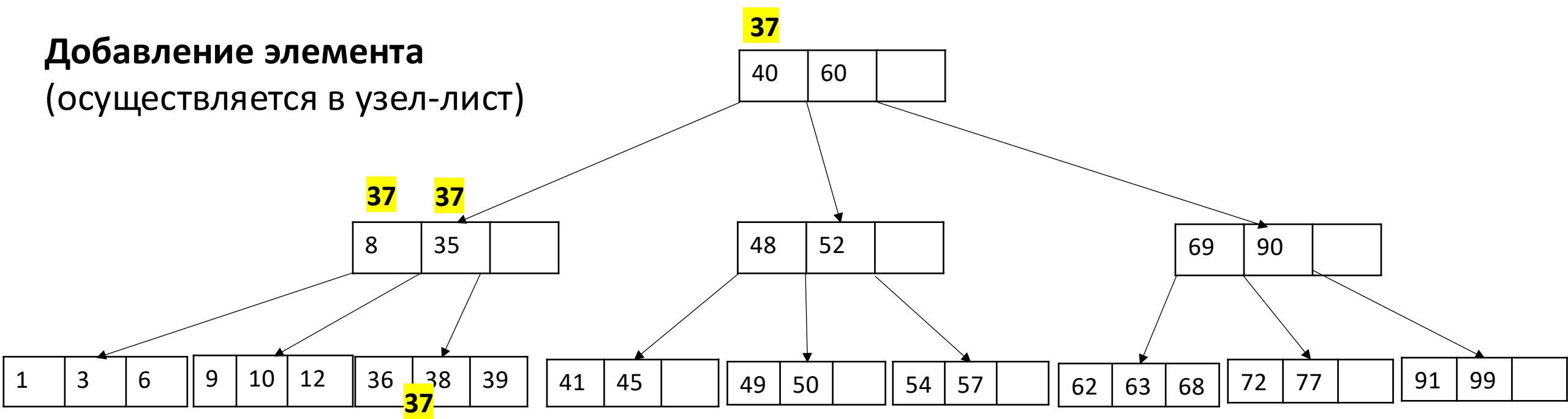
Добавление элемента (осуществляется в узел-лист)



Если в листе, в который мы пришли в результате поиска (поиск осуществляем, двигаясь от корня дерева), есть свободная ячейка, то добавляем новый ключ в эту ячейку так, чтобы все ключи листа шли в возрастающем порядке.



Добавление элемента (осуществляется в узел-лист)





Спасибо за внимание!