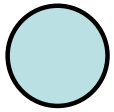
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

ПРЕБАРУВАЧКИ ДРВА

АЛГОРИТМИ И ПОДАТОЧНИ СТРУКТУРИ

- предавања -

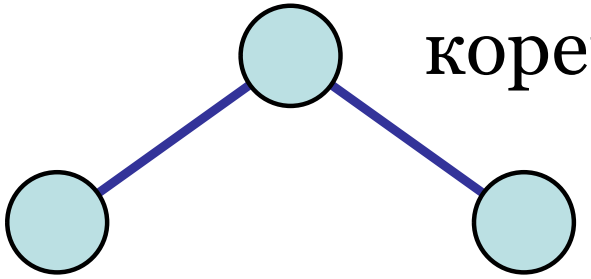
Број на јазли и висина на бинарно дрво



корен - ниво 0

1

Број на јазли и висина на бинарно дрво



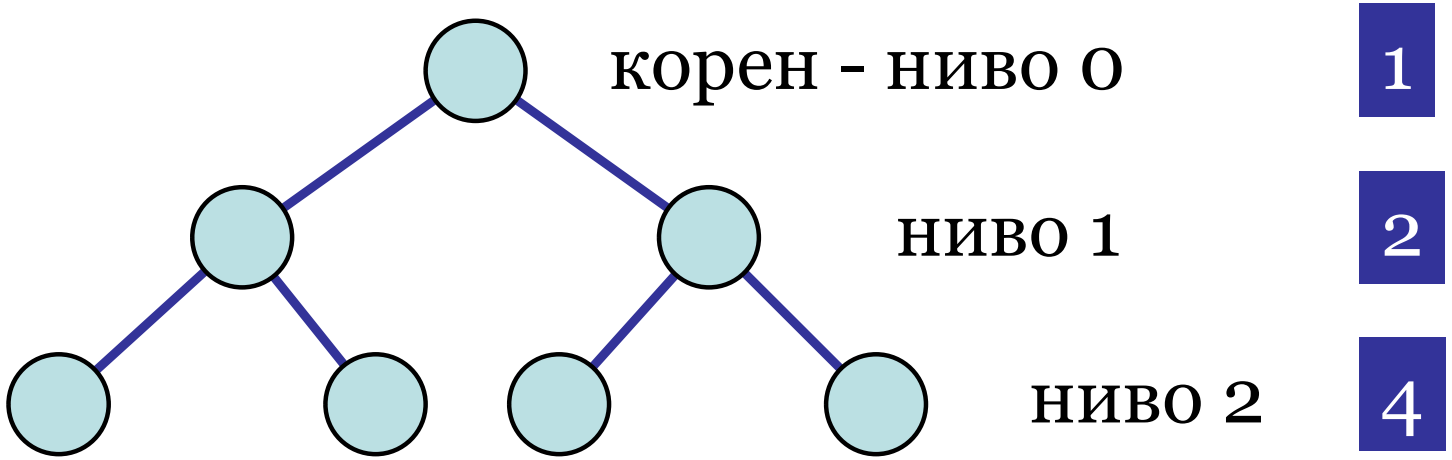
корен - ниво 0

ниво 1

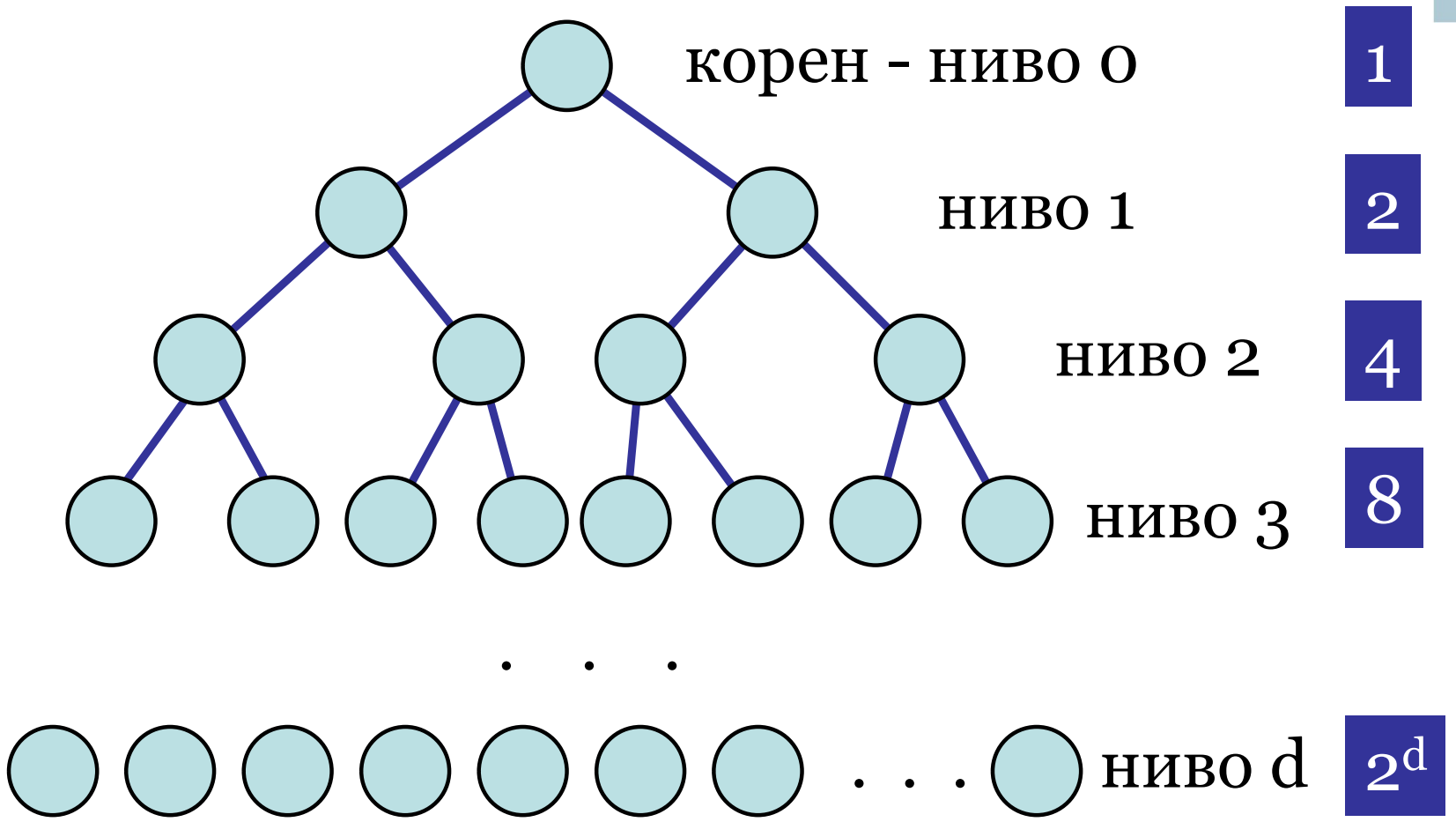
1

2

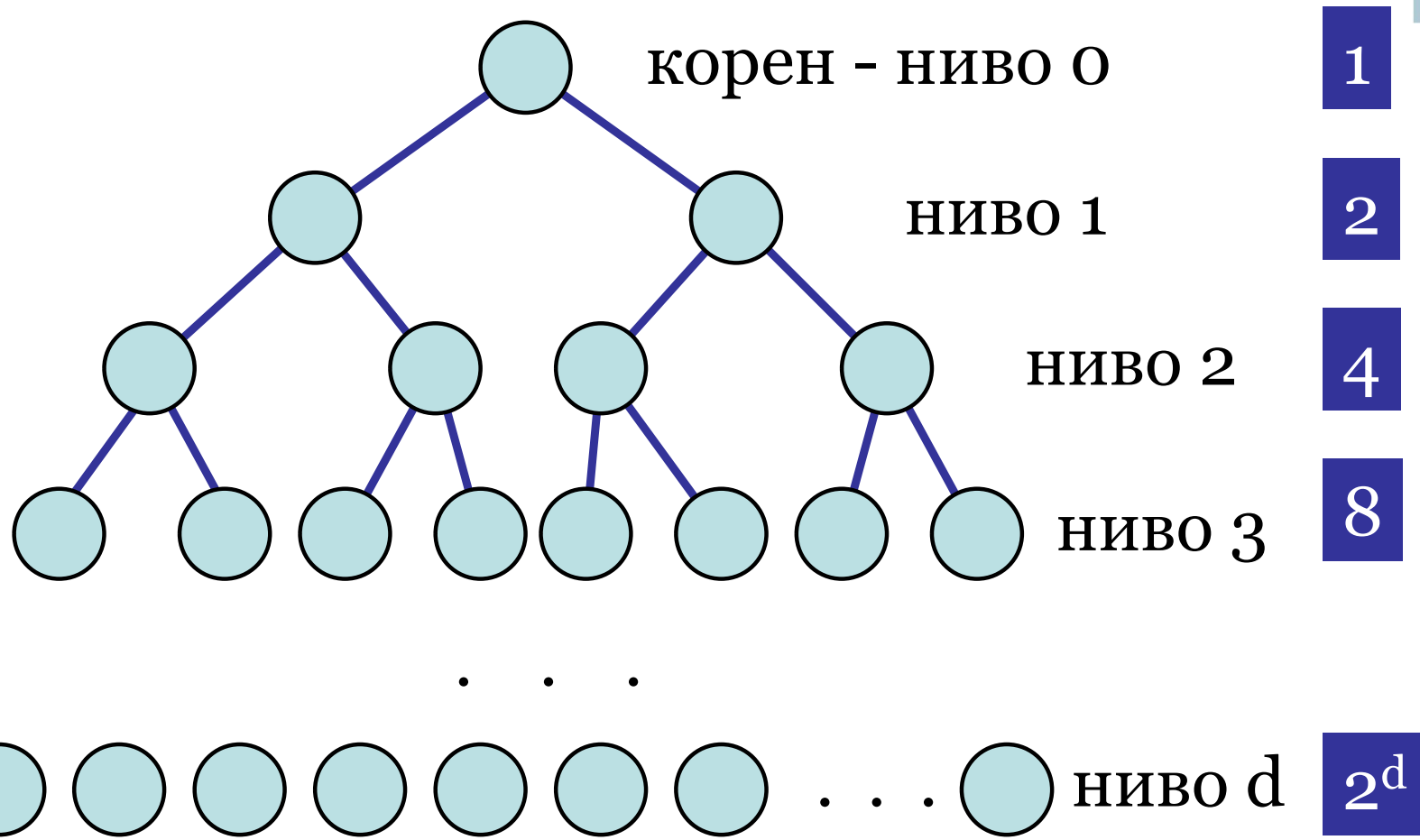
Број на јазли и висина на бинарно дрво



Број на јазли и висина на бинарно дрво



Број на јазли и висина на бинарно дрво



$$1 + 2 + 4 + \ldots + 2^d = 2^{d+1} - 1$$

Број на јазли и висина на бинарно дрво

- ако имаме бинарно дрво со *n* јазли и длабочина на дрвото *d*, тогаш

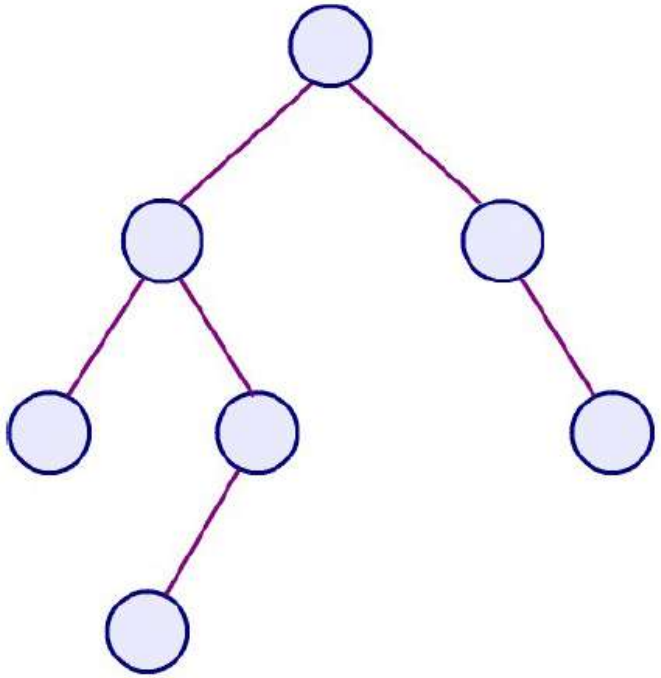
$$n \leq 2^d - 1$$

- минималната длабочина на бинарно дрво со n јазли може да се пресмета како

$$d_{min} = \lceil \log_2(n + 1) \rceil$$

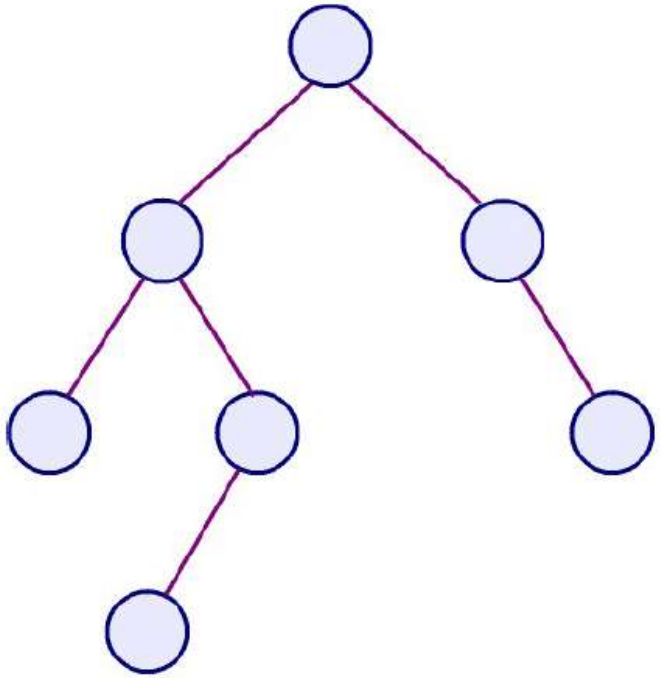
Балансирани бинарни дрва

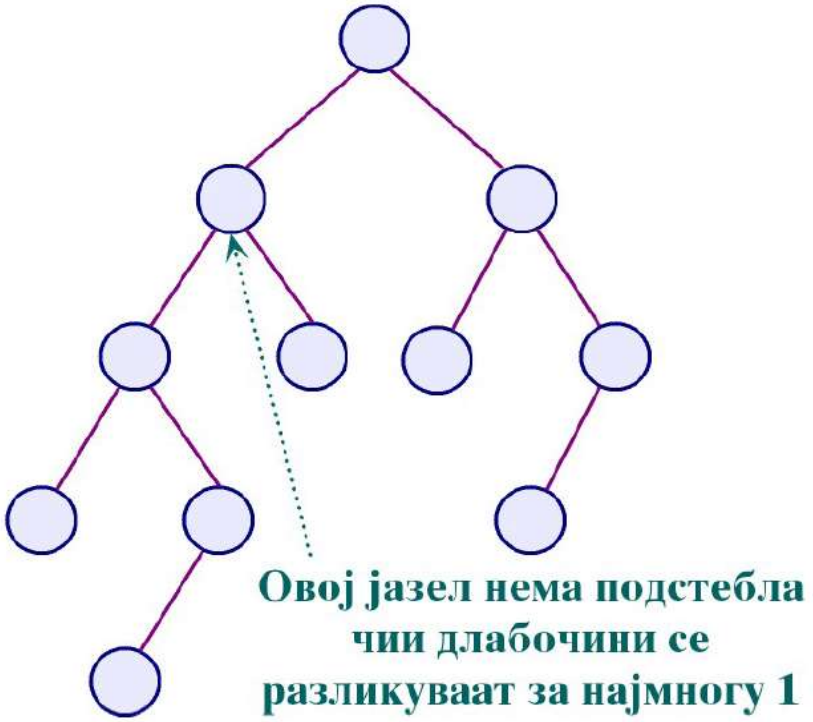
- Балансирано бинарно дрво е она бинарно дрво каде што за секој јазел од дрвото важи дека висините на неговото лево и десно поддрво **не се разликуваат за повеќе од еден**



Балансирани бинарни дрва

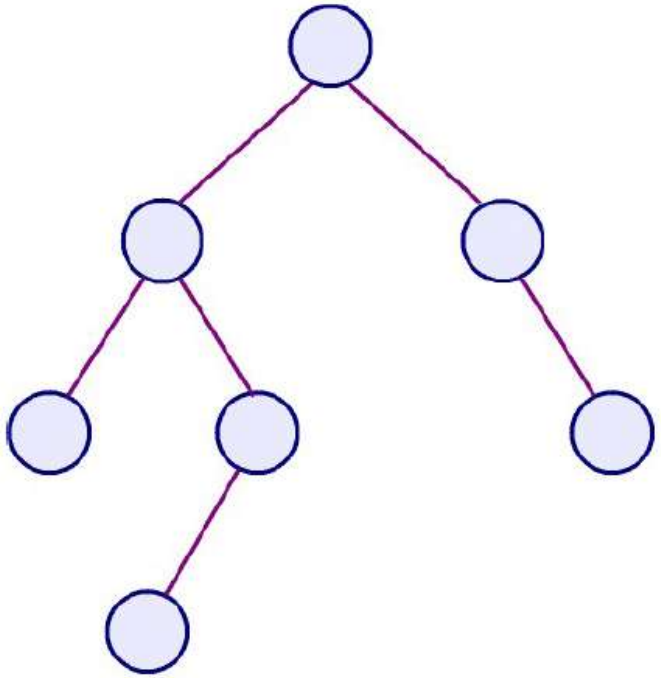
- Балансирано бинарно дрво е она бинарно дрво каде што за секој јазел од дрвото важи дека висините на неговото лево и десно поддрво **не се разликуваат за повеќе од еден**



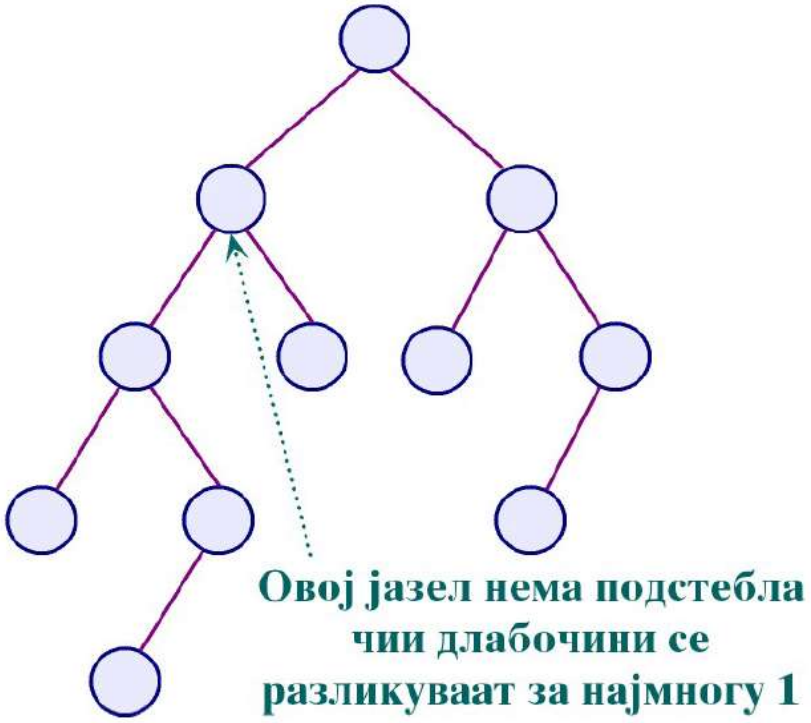


Балансирани бинарни дрва

- Балансирано бинарно дрво е она бинарно дрво каде што за секој јазел од дрвото важи дека висините на неговото лево и десно поддрво **не се разликуваат за повеќе од еден**



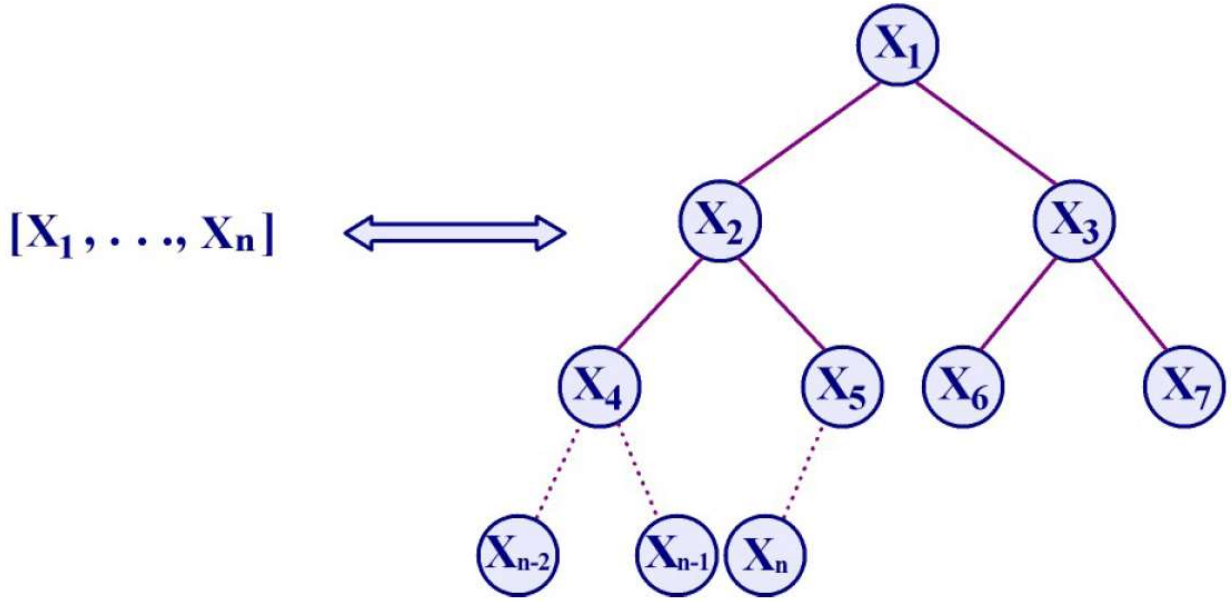
балансирано



небалансирано

Примена на бинарни дрва- Heap дрва

- Дадена низа (x_1, ...,x_n) може да е претставена во форма на бинарно дрво
- При тоа дрвото се пополнува од коренот кон листовите на начин на кој ќе биде максимално пополнето



Примена на бинарни дрва- Heap дрва

Heap дрво е комплетно бинарно дрво за кое важи дека вредноста на клучот за јазелот родител е поголема или еднаква на вредноста на клучевите на неговите деца, за секој јазел во дрвото

- Ќе покажеме дека внесувањето и бришењето елемент во вакво дрво има комплексност O(logN)
- Heap дрвата имаат примена во реализација на ефикасни алгоритми за сортирање и реализација на **приоритетни листи**

Примена на бинарни дрва - Heap sort

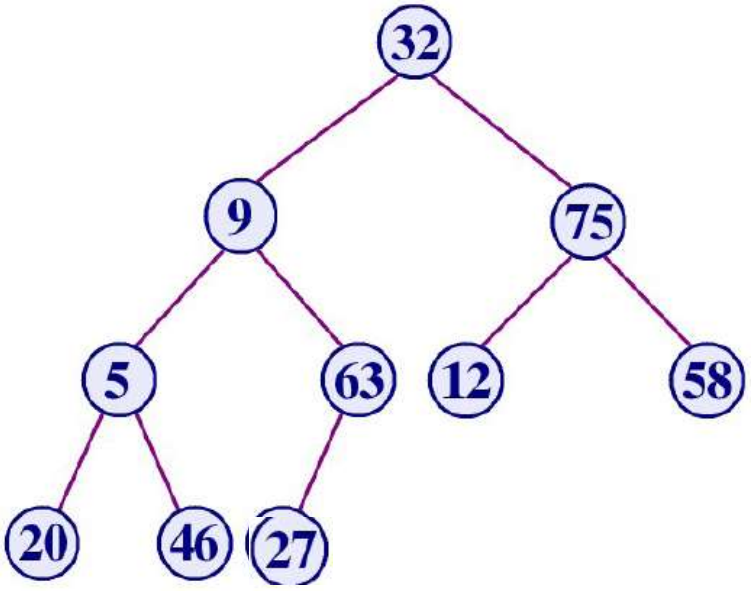
- Чекори во heap sort алгоритмот:
  - Креира heap дрво
  - Додека heap дрвото не се испразни
    - да се смести записот (клучот) од коренот на heap дрвото во резултантната сортирана низа
    - да се извади тој елемент од heap дрвото
    - повторно да се формира heap дрво

    - **Секое теме R_j има деца R_{2j} и R_{2j+1}.**

- Визуелизација за Heap дрва / Heap sort
  - https://www.cs.usfca.edu/~galles/visualization/Heap.html

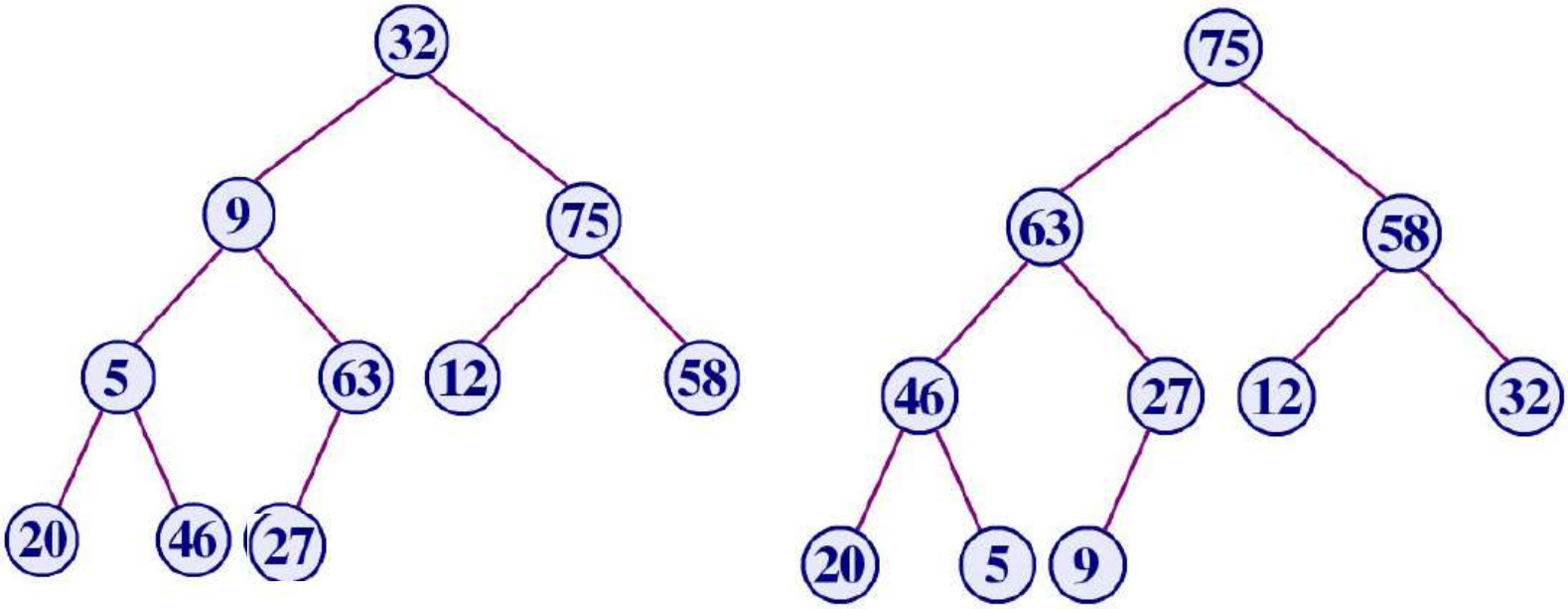
Примена на бинарни дрва - Heap sort

Влезна низа: 32, 9, 75, 5, 63, 12, 58, 20, 46, 27



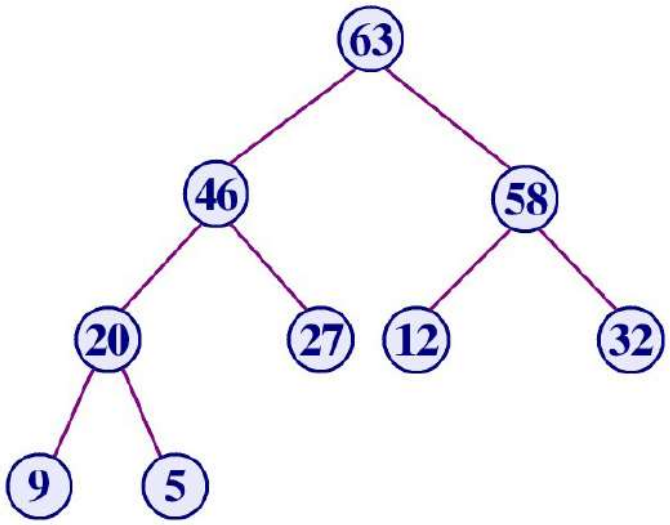
Примена на бинарни дрва - Heap sort

Влезна низа: 32, 9, 75, 5, 63, 12, 58, 20, 46, 27



Почетно heap дрво

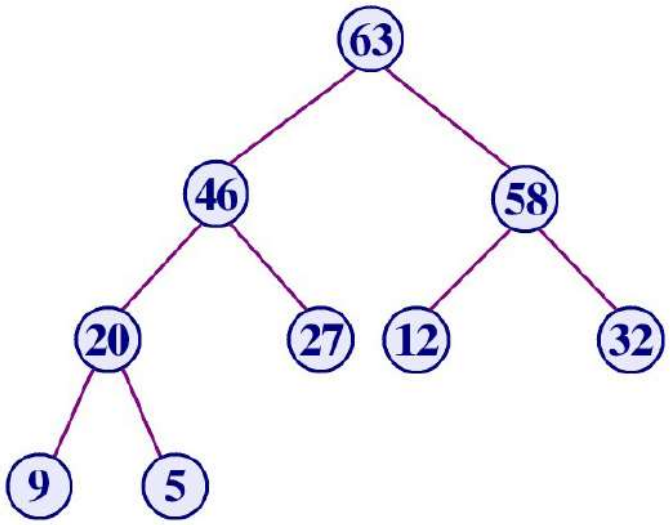
Примена на бинарни дрва - Heap sort

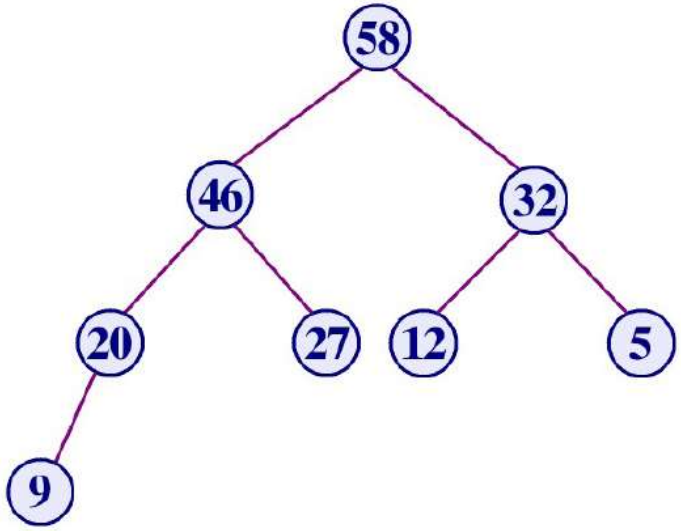


Сортирано: **75**

Heap големина: **i = 9**

Примена на бинарни дрва - Heap sort





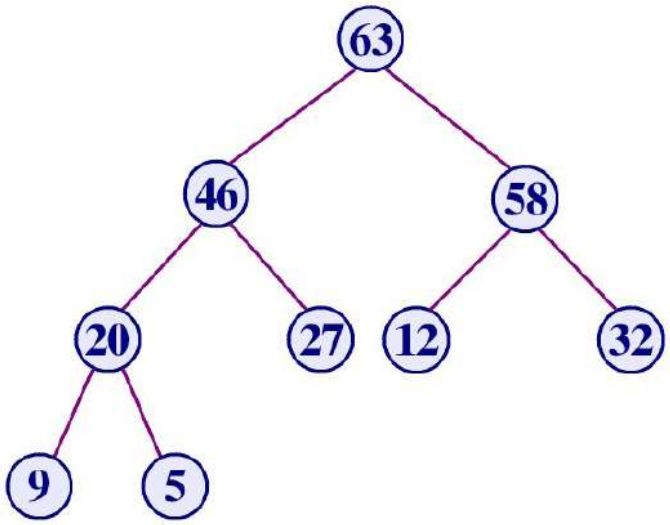
Сортирано: 75

Heap големина: i = 9

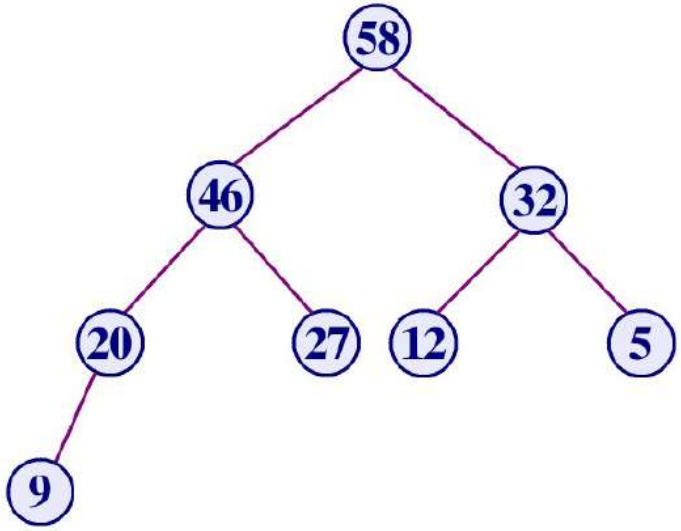
63, 75

i = 8

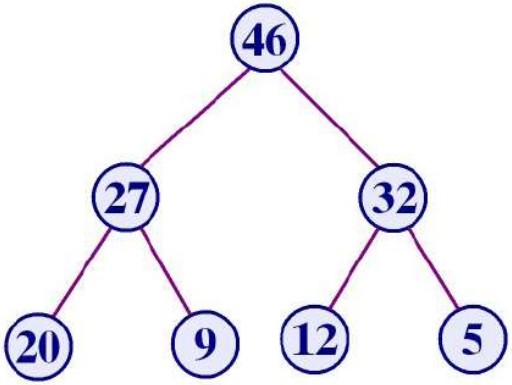
Примена на бинарни дрва - Heap sort



Сортирано: 75
Heap големина: i = 9

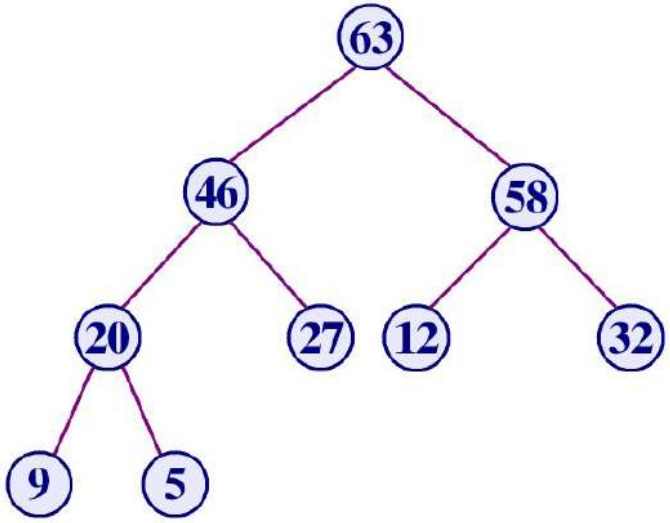


63, 75
i = 8

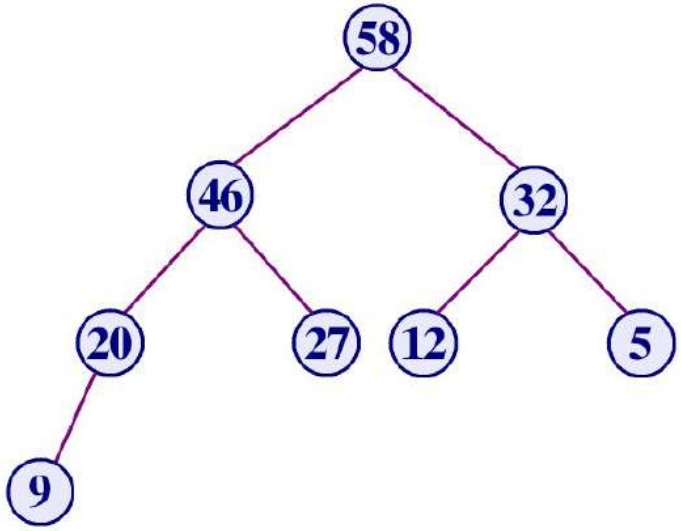


Сортирано: 58 , 63, 75
Heap големина: i = 7

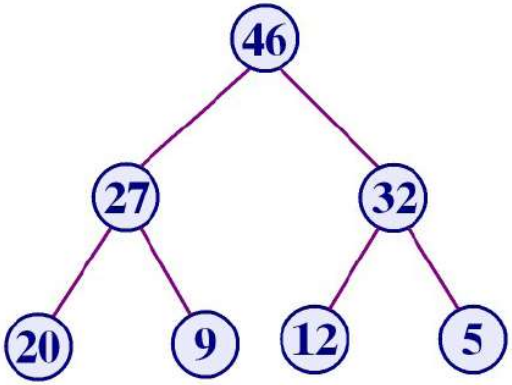
Примена на бинарни дрва - Heap sort



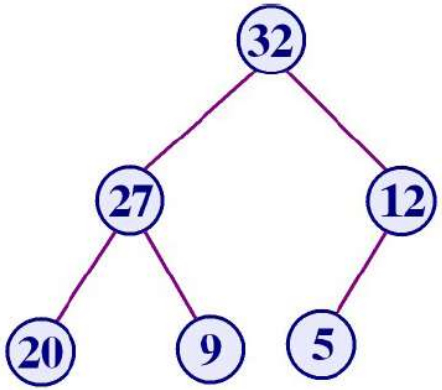
Сортирано: 75
Heap големина: i = 9



63, 75
i = 8

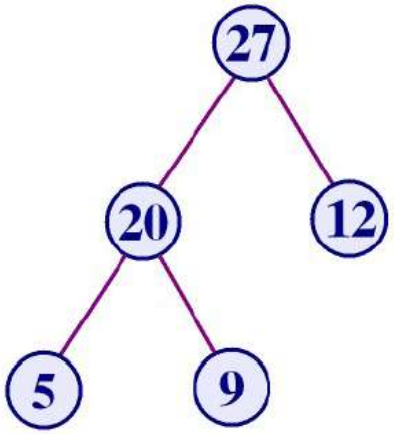


Сортирано: 58 , 63, 75
Heap големина: i = 7



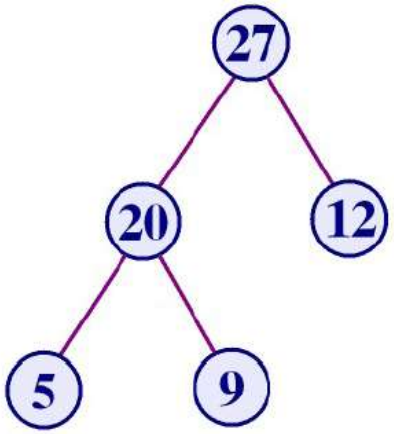
46, 58, 63, 75
i = 6

Примена на бинарни дрва - Heap sort

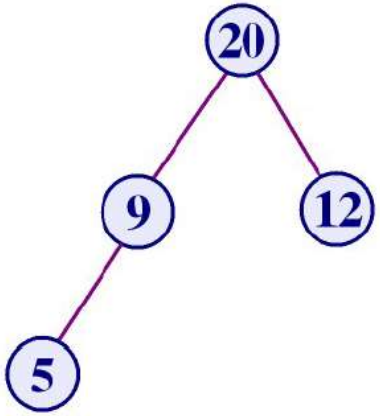


Сортирано: 32, 46, 58 , 63, 75
Heap големина: i = 5

Примена на бинарни дрва - Heap sort

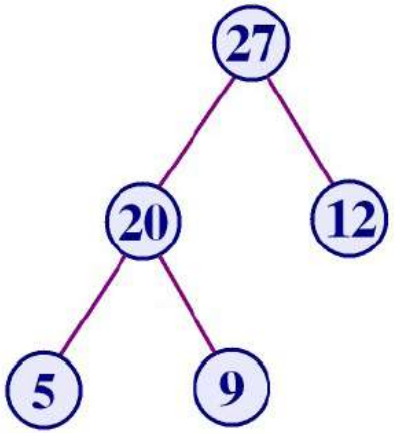


Сортирано: 32, 46, 58 , 63, 75
Heap големина: i = 5

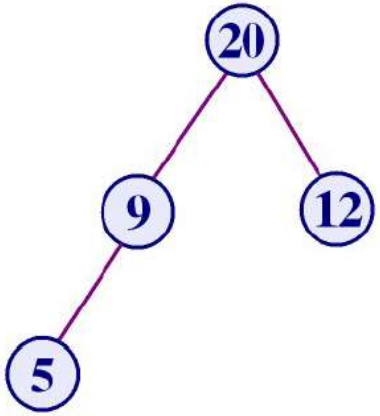


27, 32, 46, 58, 63, 75
i = 4

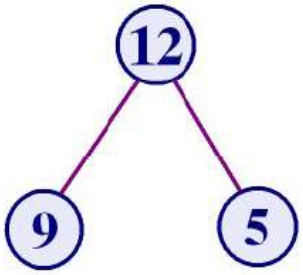
Примена на бинарни дрва - Heap sort



Сортирано: 32, 46, 58 , 63, 75
Heap големина: i = 5

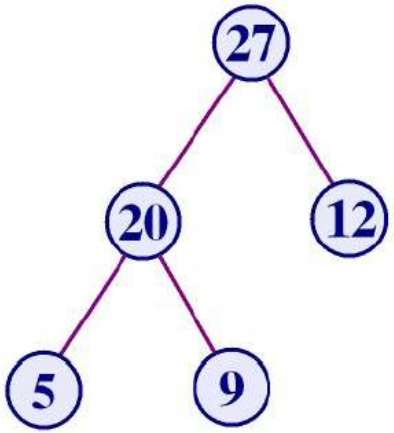


27, 32, 46, 58, 63, 75
i = 4

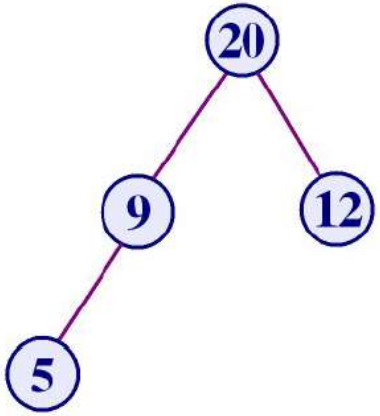


Сортирано: 20, 27, 32, 46, 58, 63, 75
Heap големина: i = 3

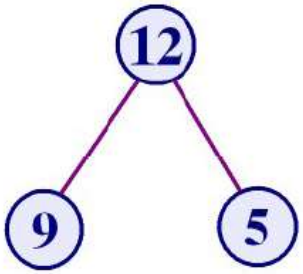
Примена на бинарни дрва - Heap sort



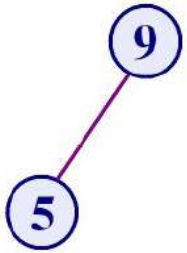
Сортирано: 32, 46, 58 , 63, 75
Heap големина: i = 5



27, 32, 46, 58, 63, 75
i = 4

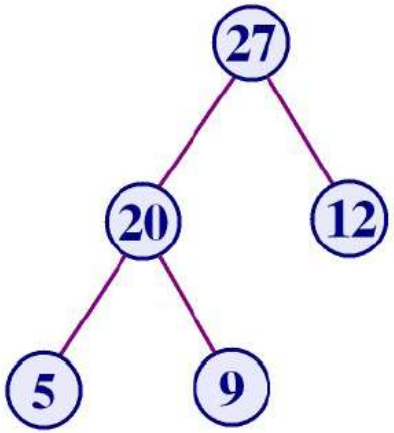


Сортирано: 20, 27, 32, 46, 58, 63, 75
Heap големина: i = 3

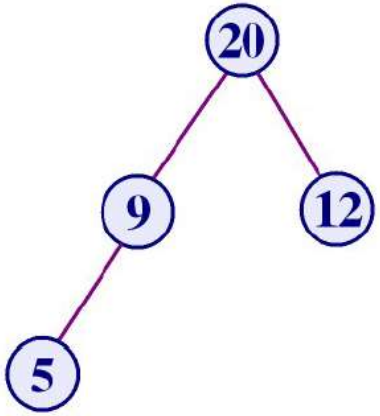


12, 20, 27, 32, 46, 58, 63, 75
i = 2

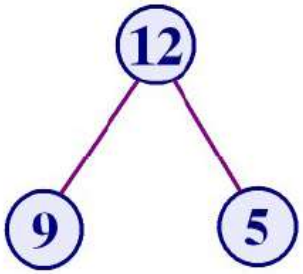
Примена на бинарни дрва - Heap sort



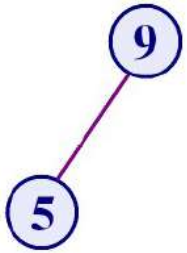
Сортирано: 32, 46, 58 , 63, 75
Heap големина: i = 5



27, 32, 46, 58, 63, 75
i = 4



Сортирано: 20, 27, 32, 46, 58, 63, 75
Heap големина: i = 3



12, 20, 27, 32, 46, 58, 63, 75
i = 2

Сортирано: 5, 9, 12, 15, 27, 32, 46, 58, 63, 77
резултат

Примена на бинарни дрва - Heap sort

```
procedure ADJUST (i,n)
R ← R_i; K ← K_i; j ← 2i
while j ≤ n do
    if j < n and K_j < K_{j+1} then j ← j + 1
    if K ≥ K_j then exit
    R_{⌊j/2⌋} ← R_j; j ← 2j
end
R_{⌊j/2⌋} ← R
end ADJUST
```

Комплексност на алгоритмот ***O(nlogn)***

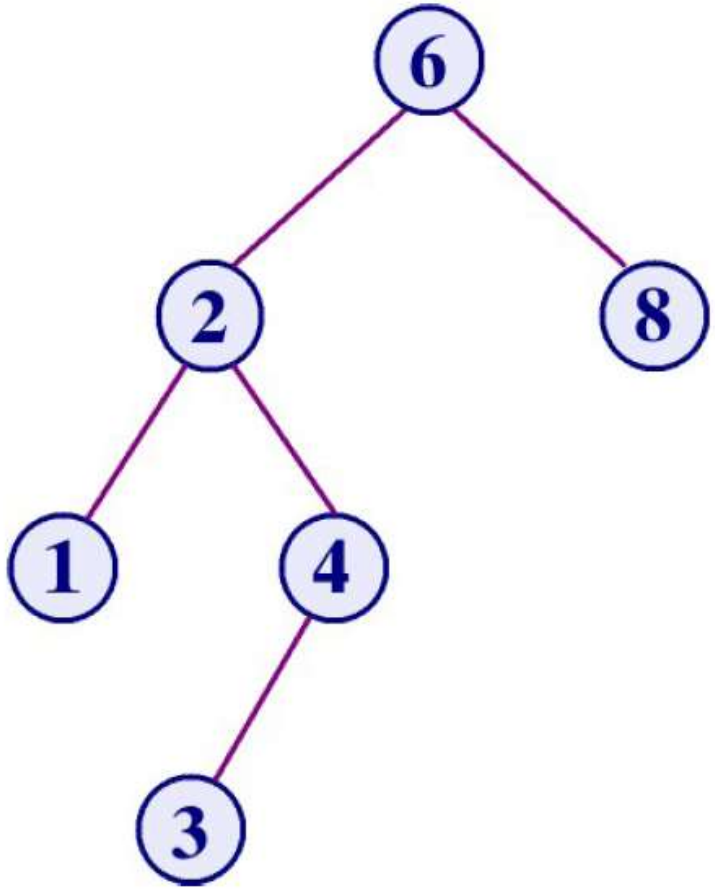
```
procedure HSORT (R,n)
for i ← ⌊n/2⌋ to 1 by -1 do call ADJUST (i,n)
for i ← n-1 to 1 by -1 do
    T ← R_{i+1}; R_{i+1} ← R_1; R_1 ← T;
    call ADJUST (1,i)
end HSORT
```

Бинарни пребарувачки дрва

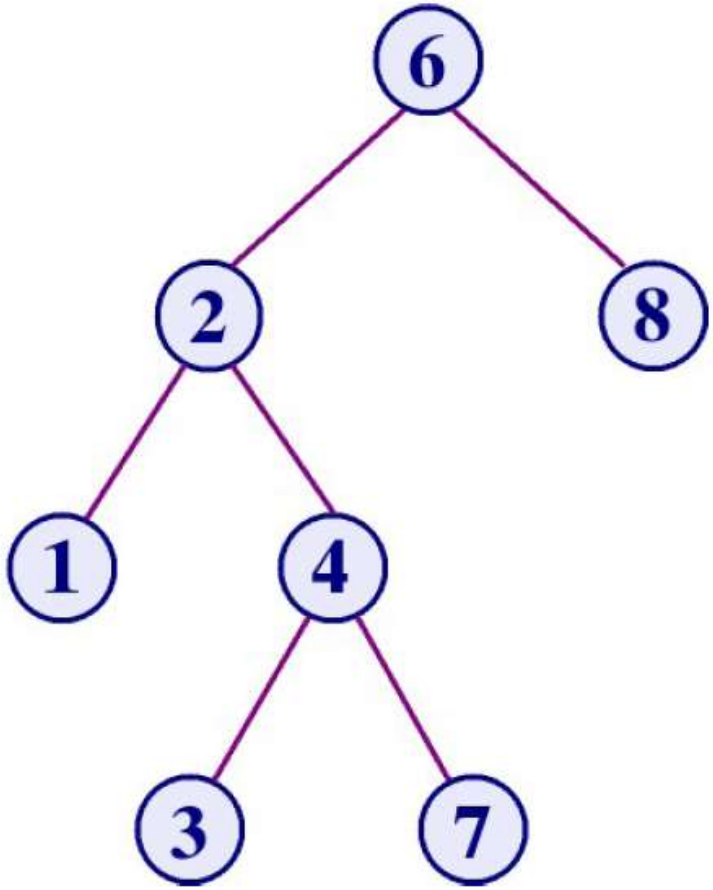
- Секој јазел од дрвото содржи информација која се нарекува **клуч**

- За секој јазел кој има вредност X за клучот важи следното правило:
  - сите јазли од неговото лево поддрво имаат вредности за клучот **помали** од вредноста X
  - сите јазли од неговото десно поддрво имаат вредности за клучот **поголеми** од вредноста X
  - нема дупликати на клучевите

Бинарни пребарувачки дрва



Бинарно пребарувачко дрво



НЕ е бинарно
пребарувачко дрво

Бинарни пребарувачки дрва

- Интересни својства на бинарните пребарувачки дрва:
  - Како ќе се пронајде јазелот со најмала вредност за клучот?
  - Како ќе се пронајде јазелот со најголема вредност на клучот?
  - Што се добива кога дрвото ќе се измине во inorder?

Бинарни пребарувачки дрва

- Дефиниција на јазел за бинарно пребарувачко дрво:

Бинарни пребарувачки дрва

- Дефиниција на јазел за бинарно пребарувачко дрво:

Дефиницијата на јазелот е иста со дефиницијата на јазел на било кое бинарно дрво!

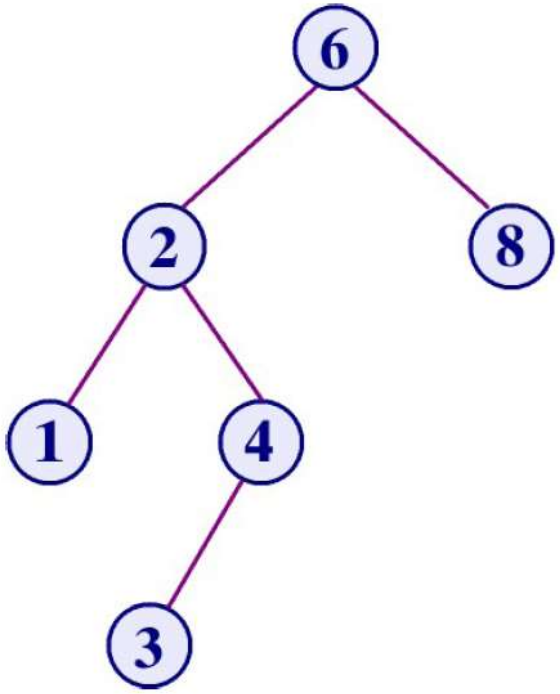
Бинарни пребарувачки дрва

- Операции со бинарни пребарувачки дрва:
  - вметнување на јазел во дрвото
  - бришење на јазел од дрвото
  - пребарување низ дрвото

Визуелизација: https://visualgo.net/bn/bst

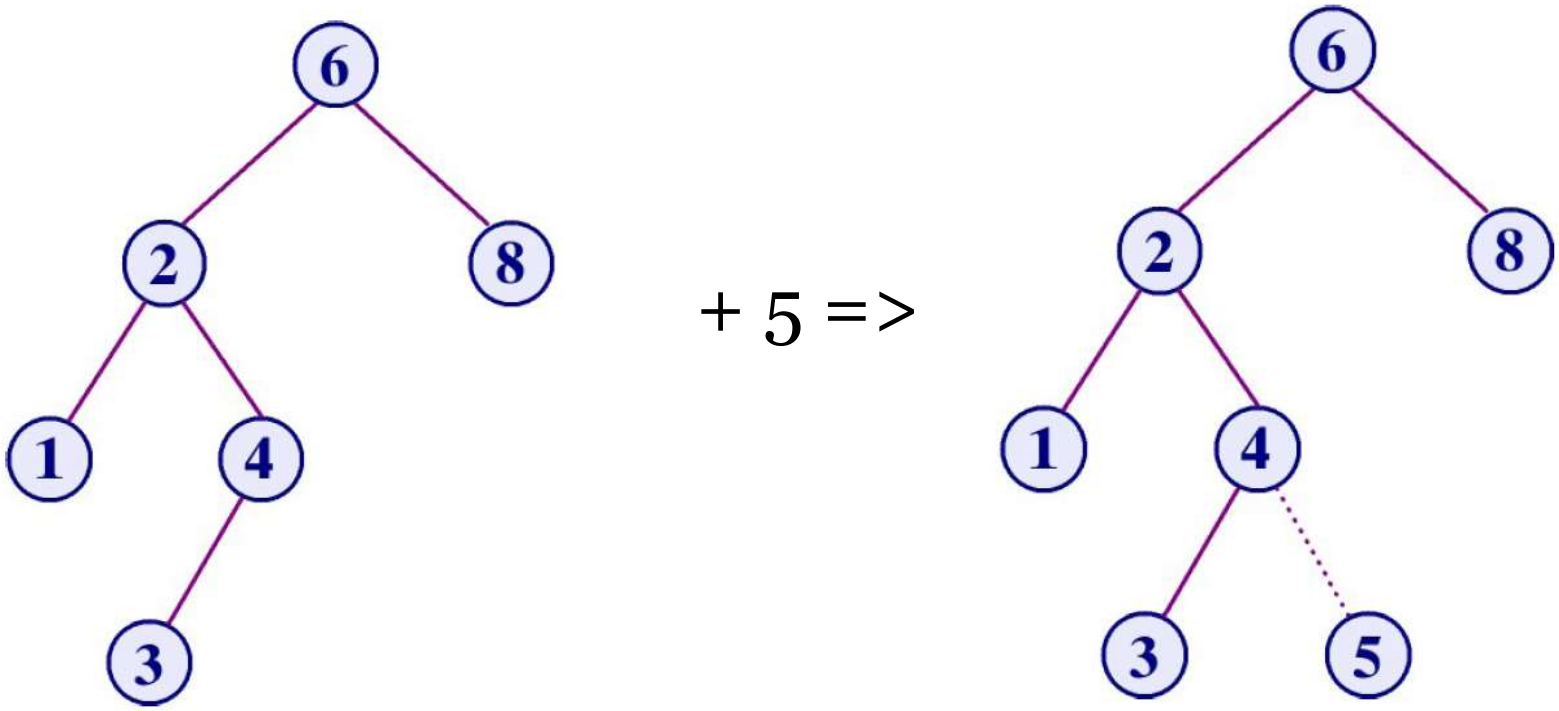
Бинарни пребарувачки дрва

- Вметнување на јазел:



Бинарни пребарувачки дрва

- Вметнување на јазел:



Бинарни пребарувачки дрва

- Вметнување на јазел:

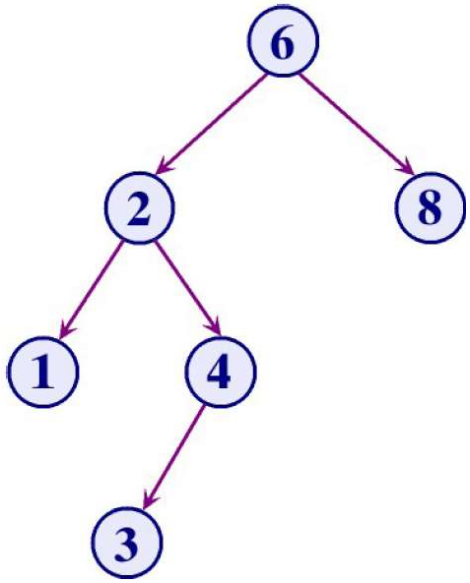
```
TREE-INSERT(T, z)
  y ← NULL
  x ← root[T]
  while x ≠ NULL
  do begin
        y ← x
        if key[z] < key[x] then x ← left[x]
        else x ← right[x]
  end do
  if y = NULL then root[T] ← z
  else if key[z] < key[y] then left[y] ← z
       else right[y] ← z
```

Бинарни пребарувачки дрва

- Бришење на јазел:
  - оваа операција е малку покомплицирана
  - бришење на јазел кој е лист
  - бришење на јазел кој има едно дете
  - бришење на јазел кој има две деца

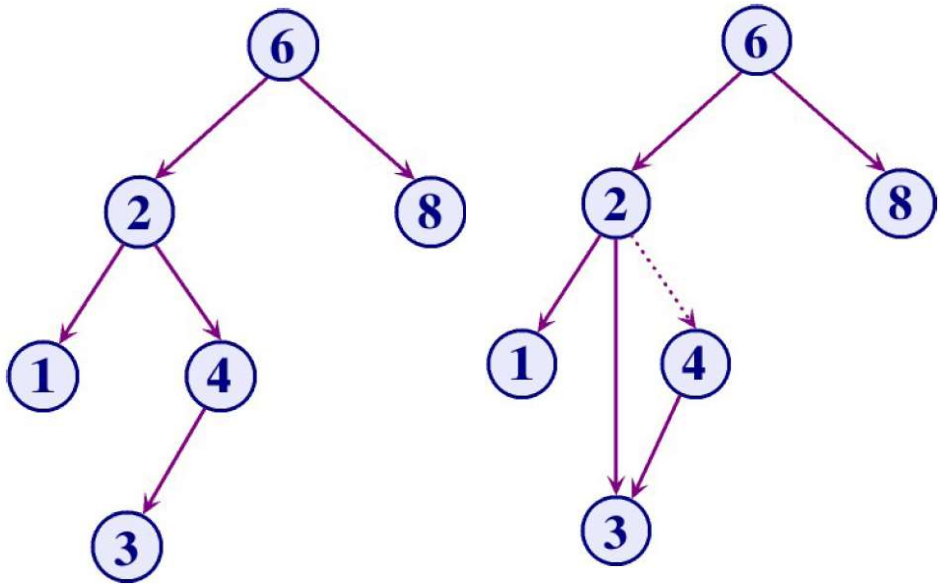
Бинарни пребарувачки дрва

- Бришење на јазел:



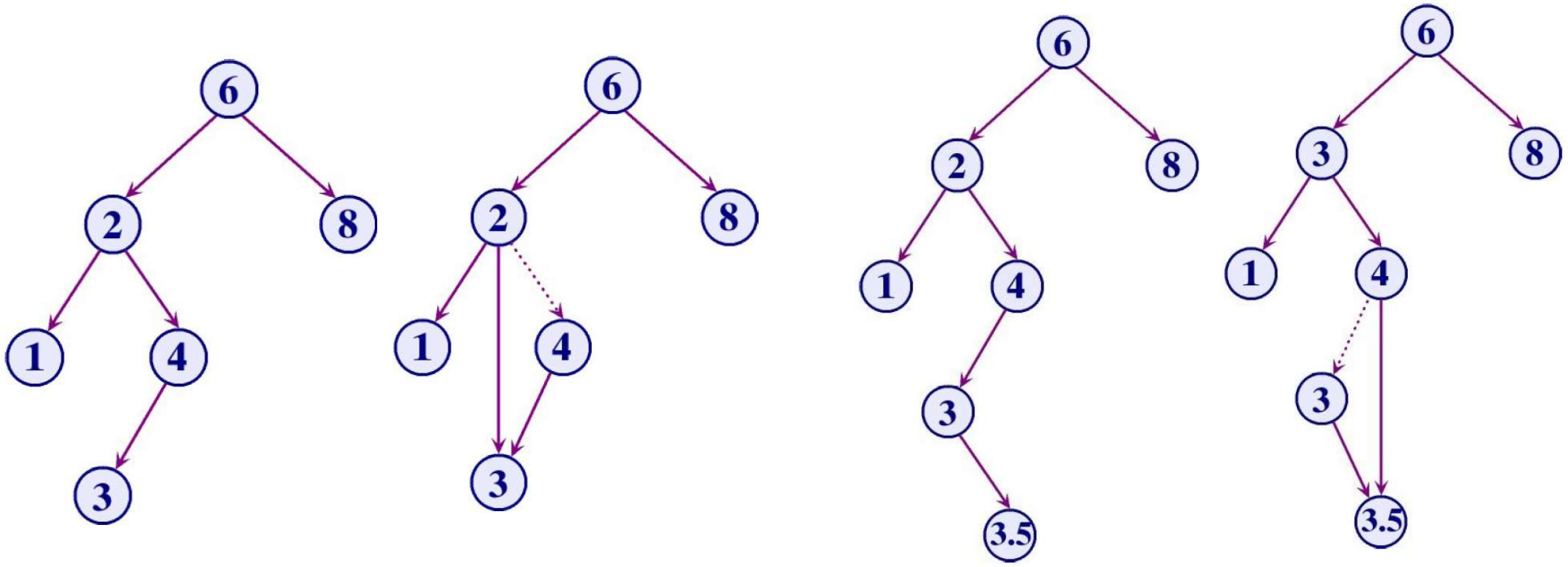
Бинарни пребарувачки дрва

- Бришење на јазел:



Бинарни пребарувачки дрва

- Бришење на јазел:



Бинарни пребарувачки дрва

```
delete( element_type x, SEARCH_TREE T )
{
    tree_ptr tmp_cell, child;
    if( T == NULL )
        error("Elementot ne e pronajden");
    else
    if( x < T->element ) /* Odi levo */
        T->left = delete( x, T->left );
    else
        if( x > T->element ) /* Odi desno */
            T->right = delete( x, T->right );
        else /* Najdeniot element da se izbrise */
        if( T->left && T->right ) /* Dve deca */
        { /* Zameni so najmaliot od desnoto podsteblo */
            tmp_cell = find_min( T->right );
            T->element = tmp_cell->element;
            T->right = delete( T->element, T->right );
        }
        else /* Edno dete */
```

Бинарни пребарувачки дрва

```
    else /* Edno dete - prodolzenie*/
    {
        tmp_cell = T;
        if( T->left == NULL )     /* Samo desno dete */
            child = T->right;
        if( T->right == NULL )    /* Samo levo dete */
            child = T->left;
        free( tmp_cell );
        return child;
    }
return T;
}
```

Бинарни пребарувачки дрва

- Пребарување во дрвото:

рекурзивно

```
TREE-SEARCH (x, k)
if x = NULL or k = key[x]
then return x
if k < key[x] then return TREE-SEARCH (left[x], k)
else return TREE-SEARCH (right[x], k)
```

Бинарни пребарувачки дрва

- Пребарување во дрвото:

рекурзивно

```
TREE-SEARCH (x, k)
if x = NULL or k = key[x]
then return x
if k < key[x] then return TREE-SEARCH (left[x], k)
else return TREE-SEARCH (right[x], k)
```

нерекурзивно

```
ITERATIVE-TREE-SEARCH (x,k)
while x ≠ NULL and k ≠ key[x] do
    if k < key[x] then x ← left[x]
    else x ← right[x]
return x
```

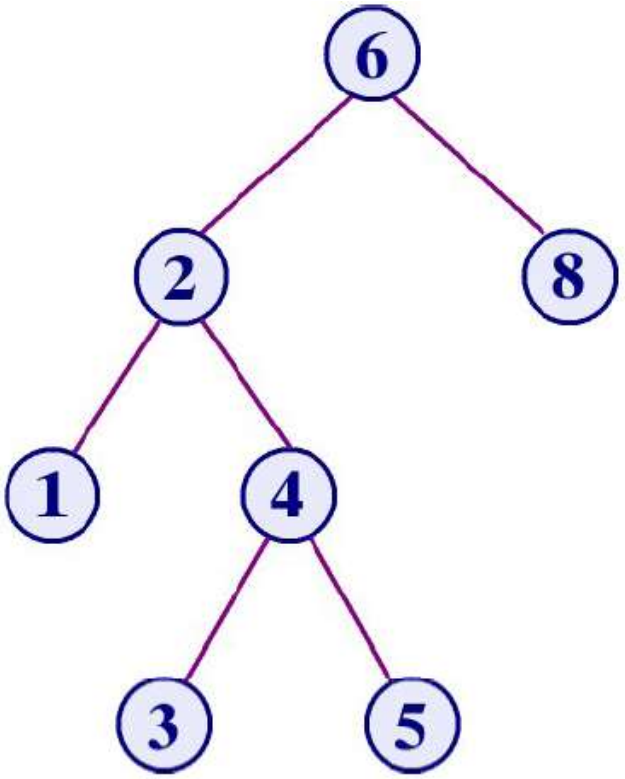
Бинарни пребарувачки дрва

- Времетраењето на сите разгледани операции кај бинарните пребарувачки дрва зависи од висината на јазелот кој треба да се обработи
- Нивото на јазлите во дрво со n елементи може значајно да варира во интервалот од $\log_2 n$ до n
- Доколку сакаме подобри перформанси дрвата треба да бидат **балансирани**

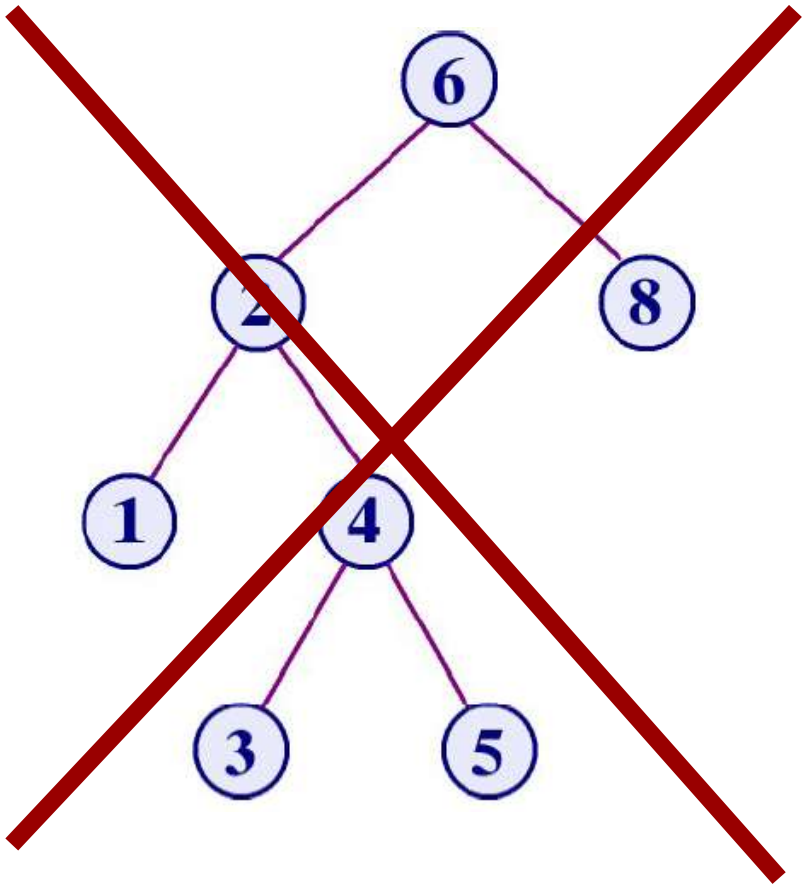
AVL дрва

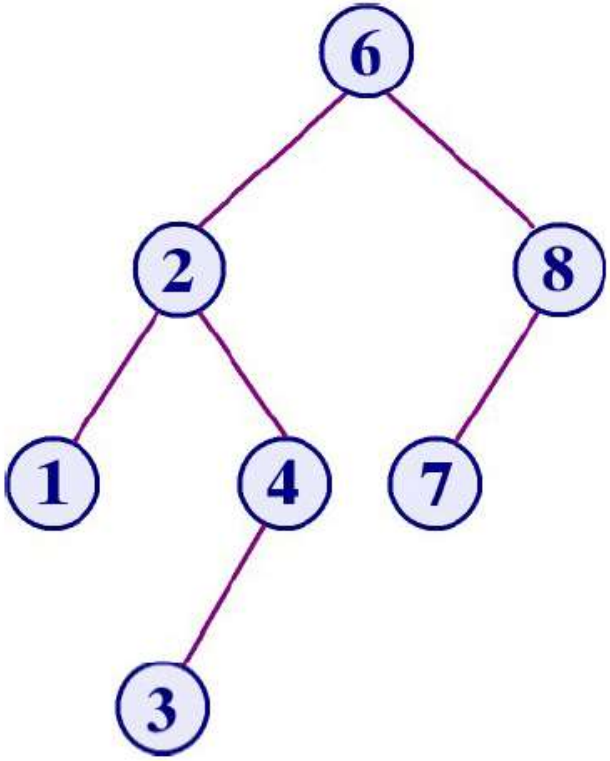
- AVL (Adelson-Velskii & Landis) дрво е бинарно пребарувачко дрво кое во истовреме е и балансирано дрво
- Со ова се осигурува дека длабочината на дрвото (а со тоа и комплексноста на најчестите операции) е од редот $O(\log n)$

AVL дрва



AVL дрва





AVL дрва

- AVL дрвата имаат подобри перформанси
- Реализацијата на овие дрва мора да се изврши програмски
- Операциите на внесување и бришење на јазел ќе бидат покомплицирани од истите операции кај бинарните пребарувачки дрва

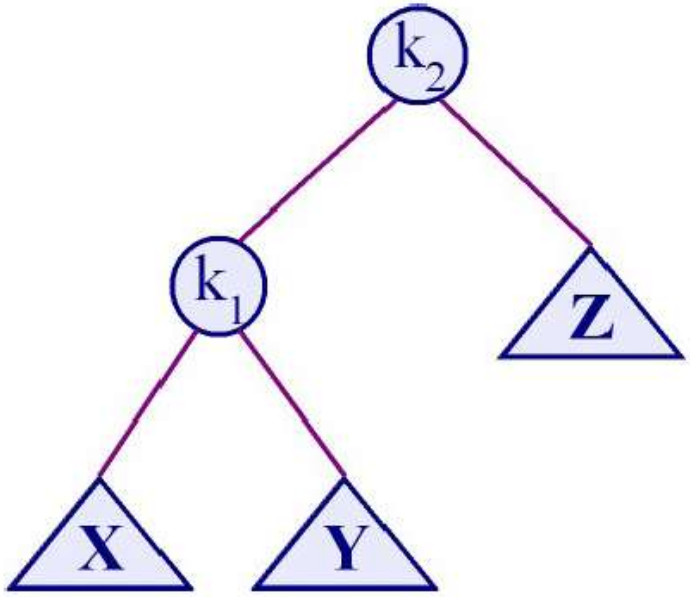
AVL дрва

- AVL дрвата имаат подобри перформанси
- Реализацијата на овие дрва мора да се изврши програмски
- Операциите на внесување и бришење на јазел ќе бидат покомплицирани од истите операции кај бинарните пребарувачки дрва

Проблем: Нарушување на балансираноста на дрвото!

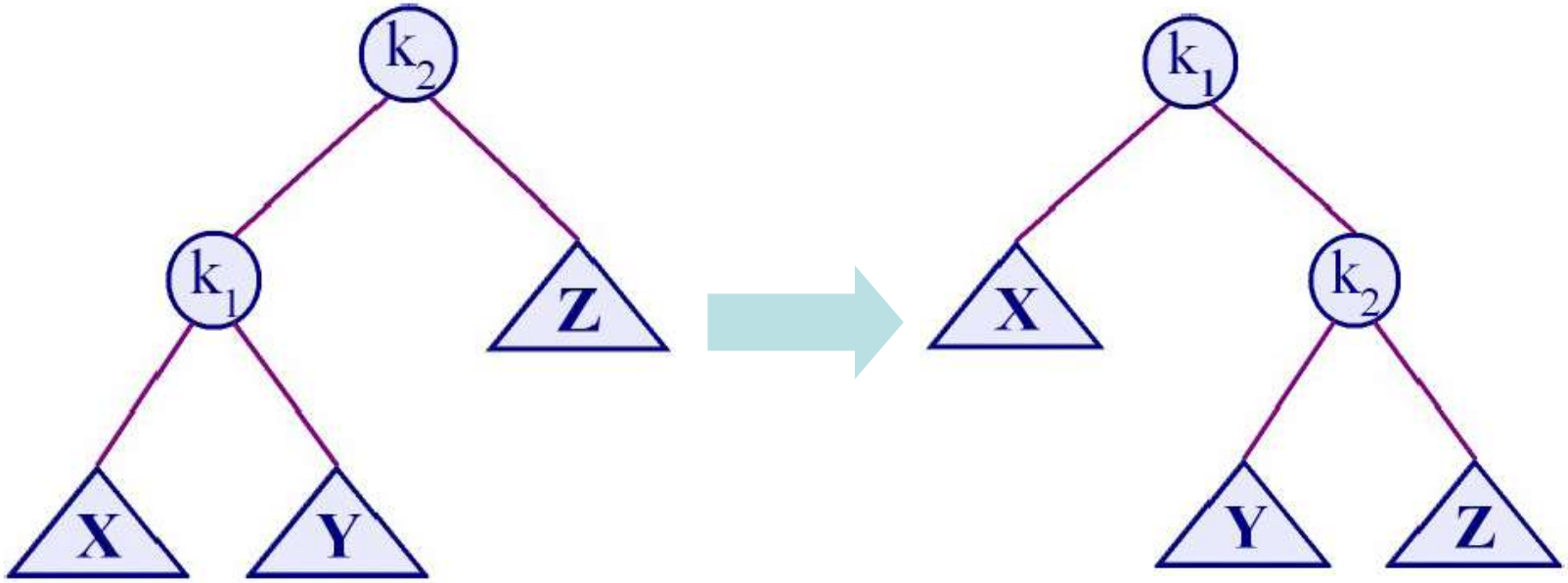
AVL дрва

- Балансираноста на едно AVL дрво се одржува со операцијата еднократна **ротација** на јазел

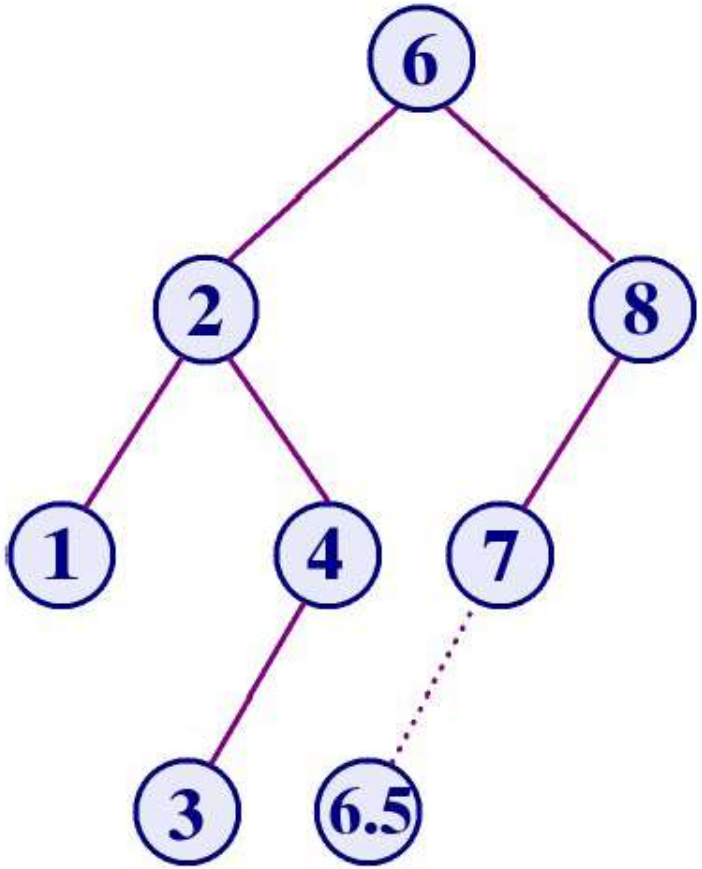


AVL дрва

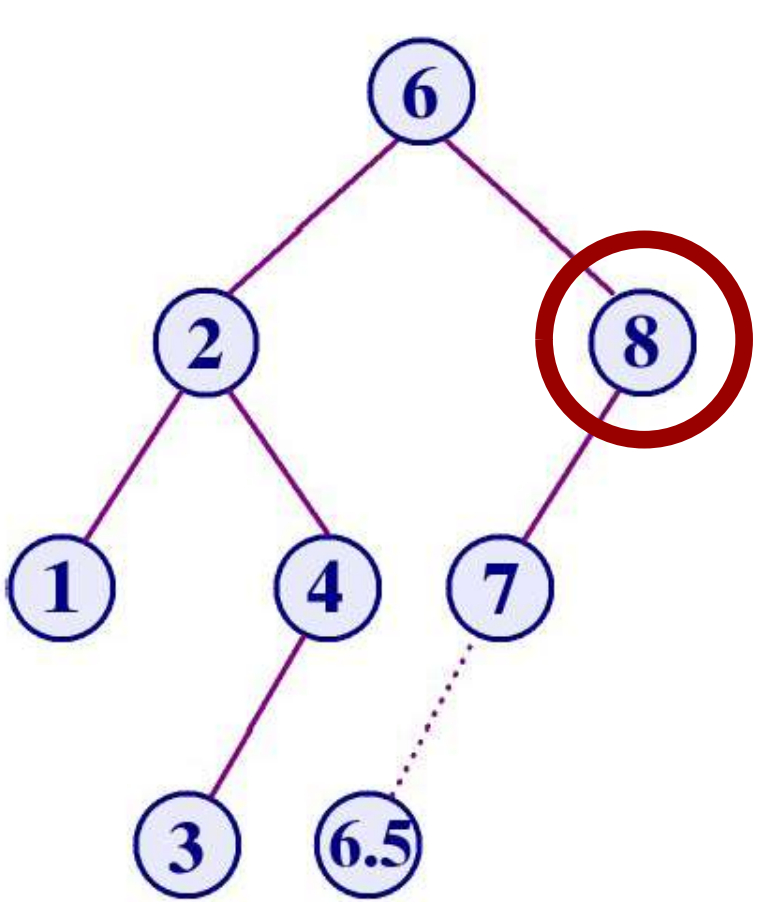
- Балансираноста на едно AVL дрво се одржува со операцијата еднократна **ротација** на јазел

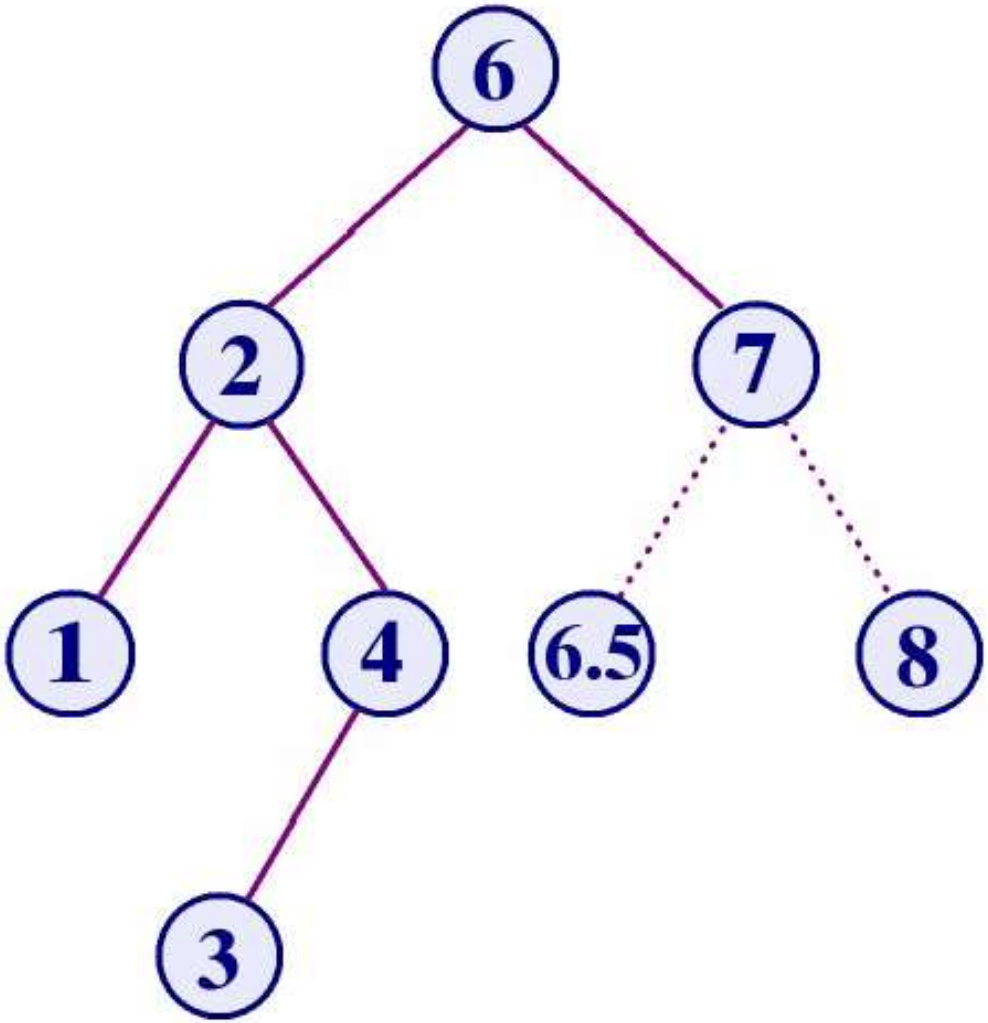


AVL дрва



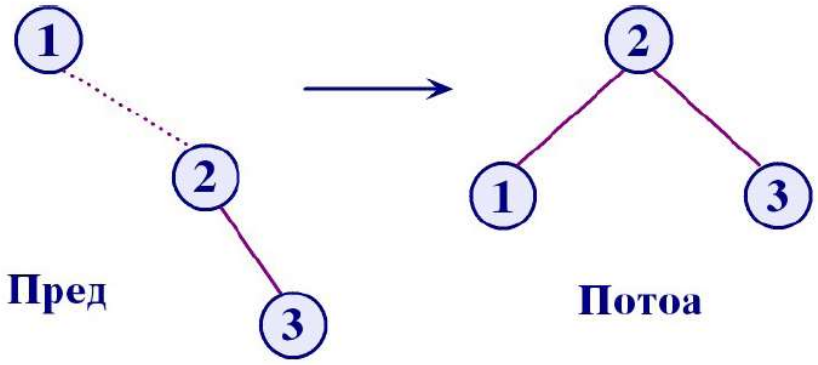
AVL дрва





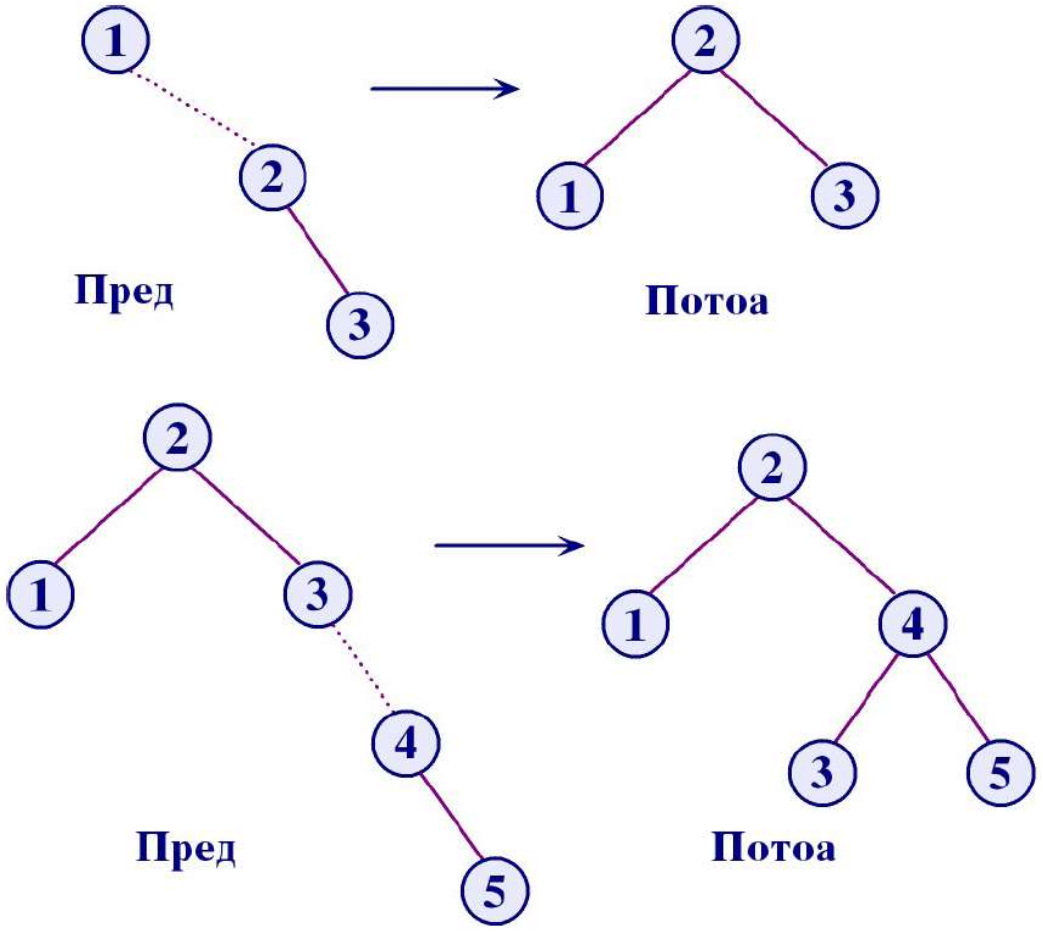
AVL дрва

- Пример: Градење на AVL дрво од вредностите 1, 2, 3, 4, 5, 6 и 7

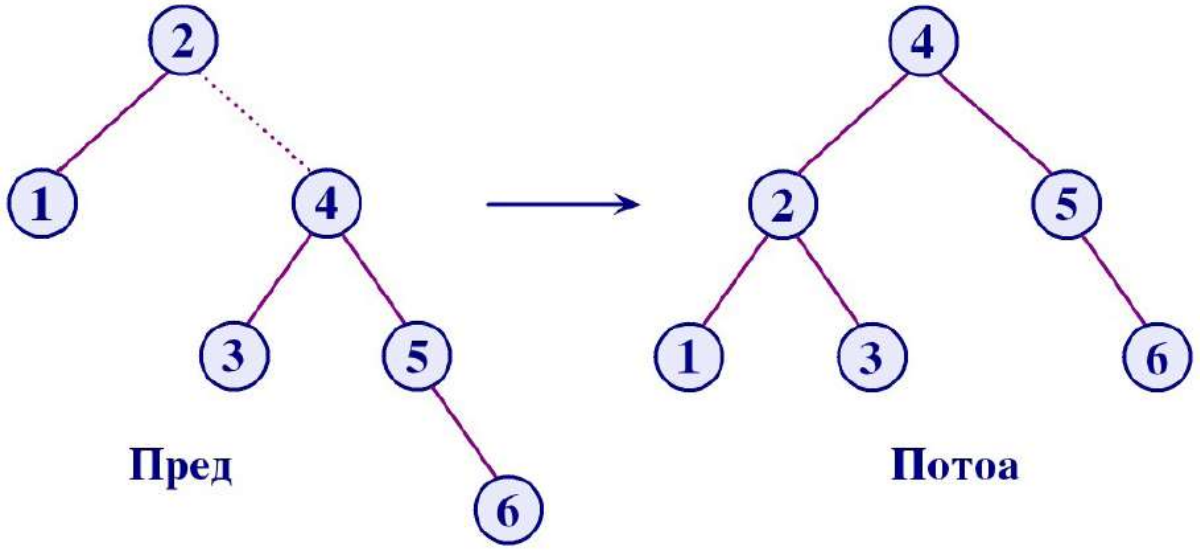


AVL дрва

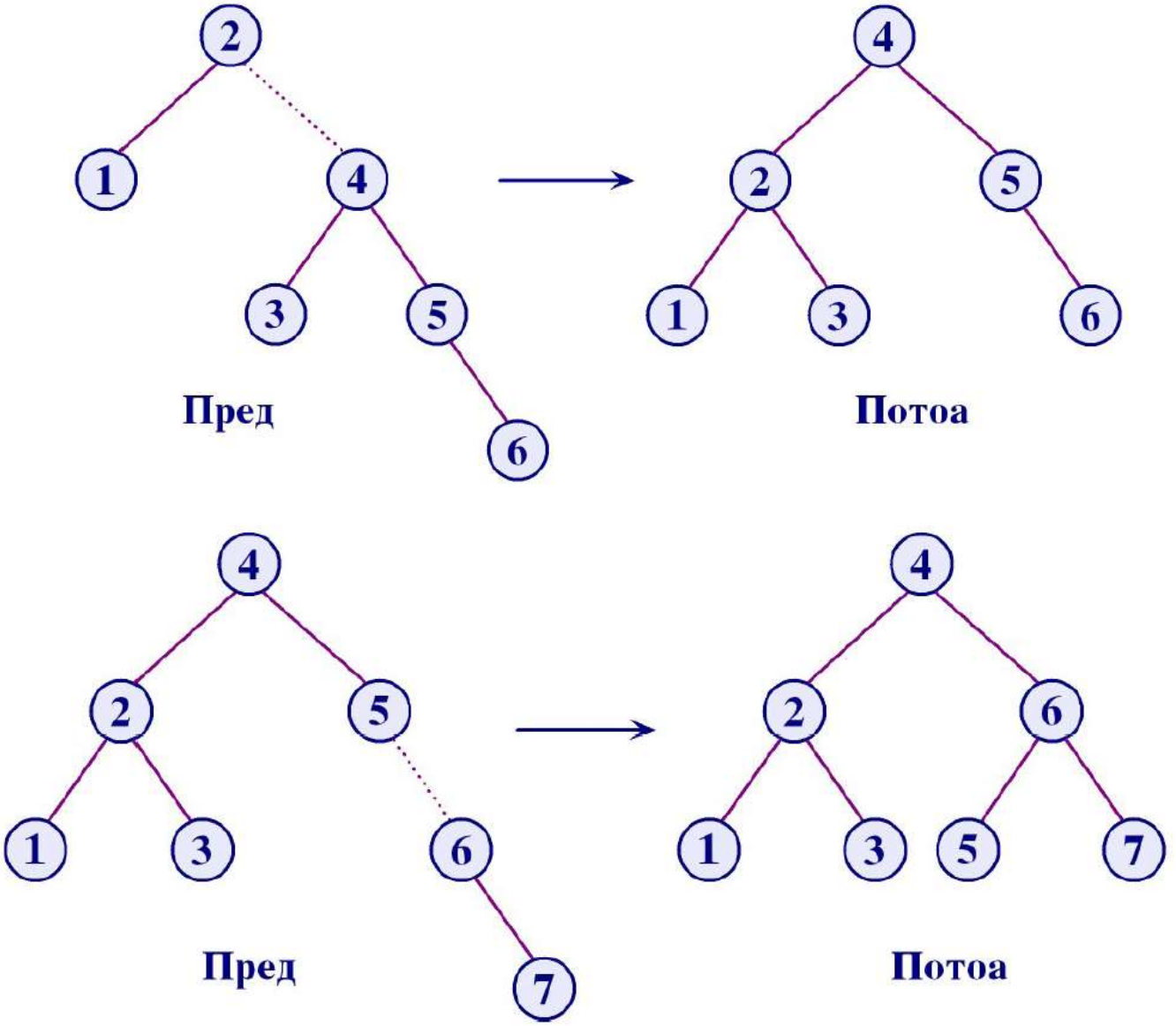
- Пример: Градење на AVL дрво од вредностите 1, 2, 3, 4, 5, 6 и 7



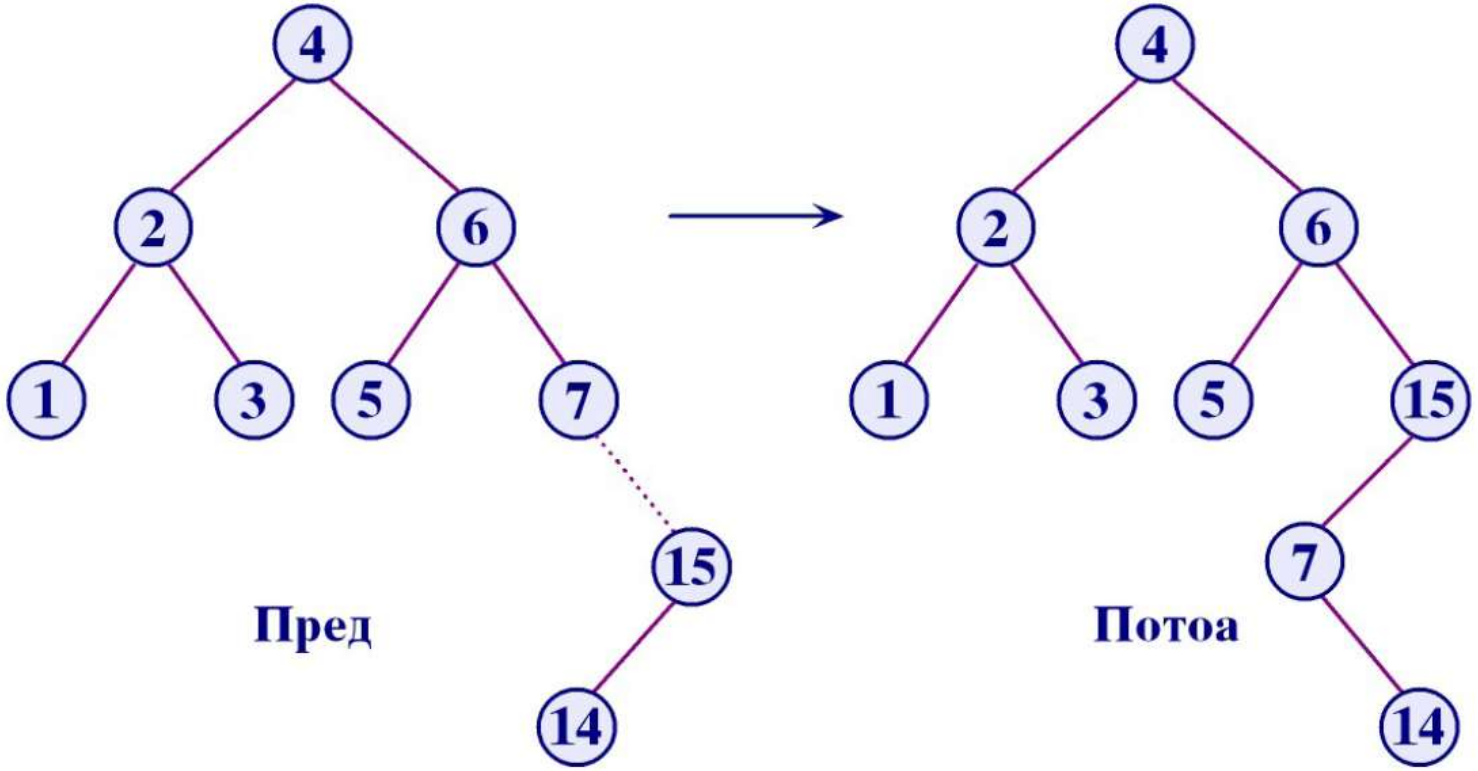
AVL дрва



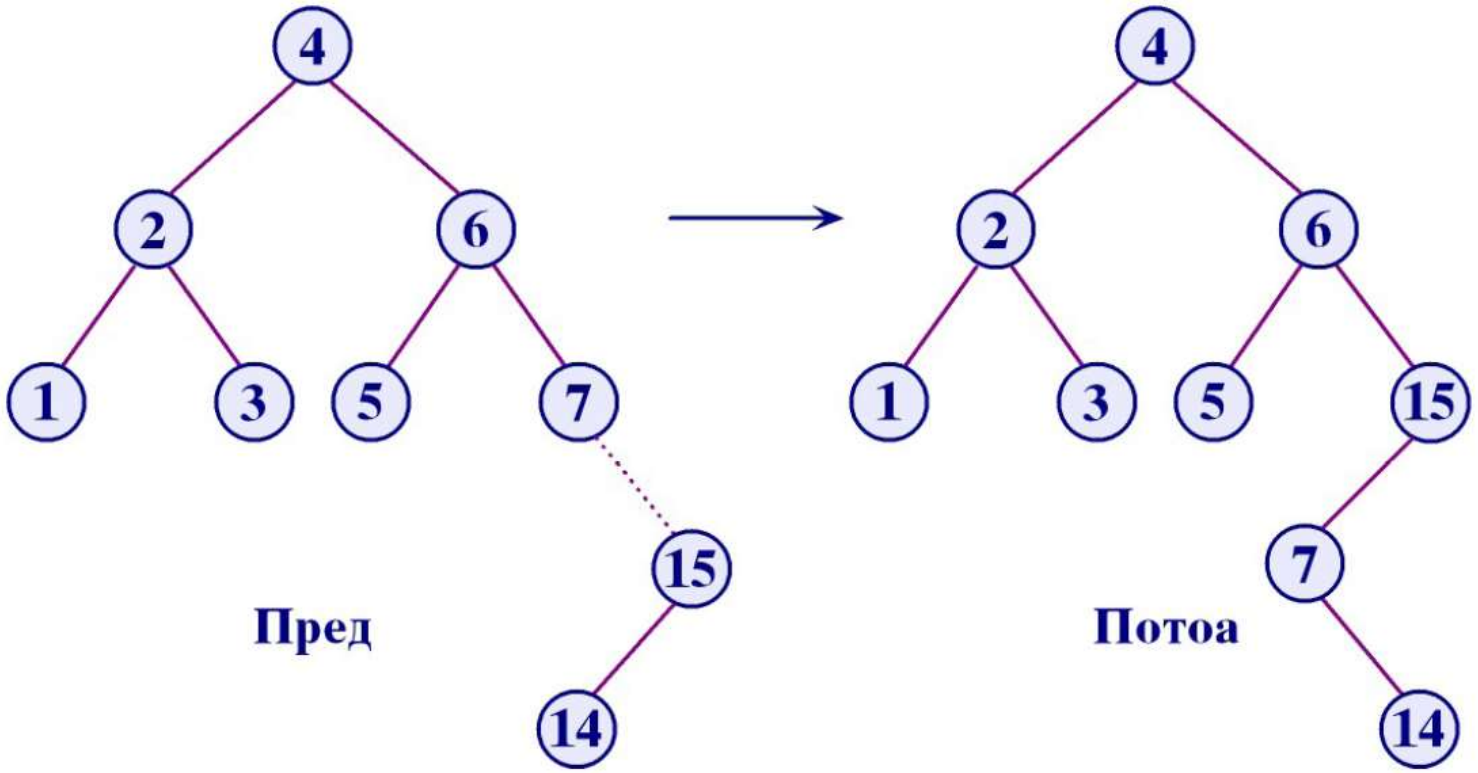
AVL дрва



AVL дрва



AVL дрва



Проблем: Еднократната ротација не помага во балансирањето на дрвото во овој случај!

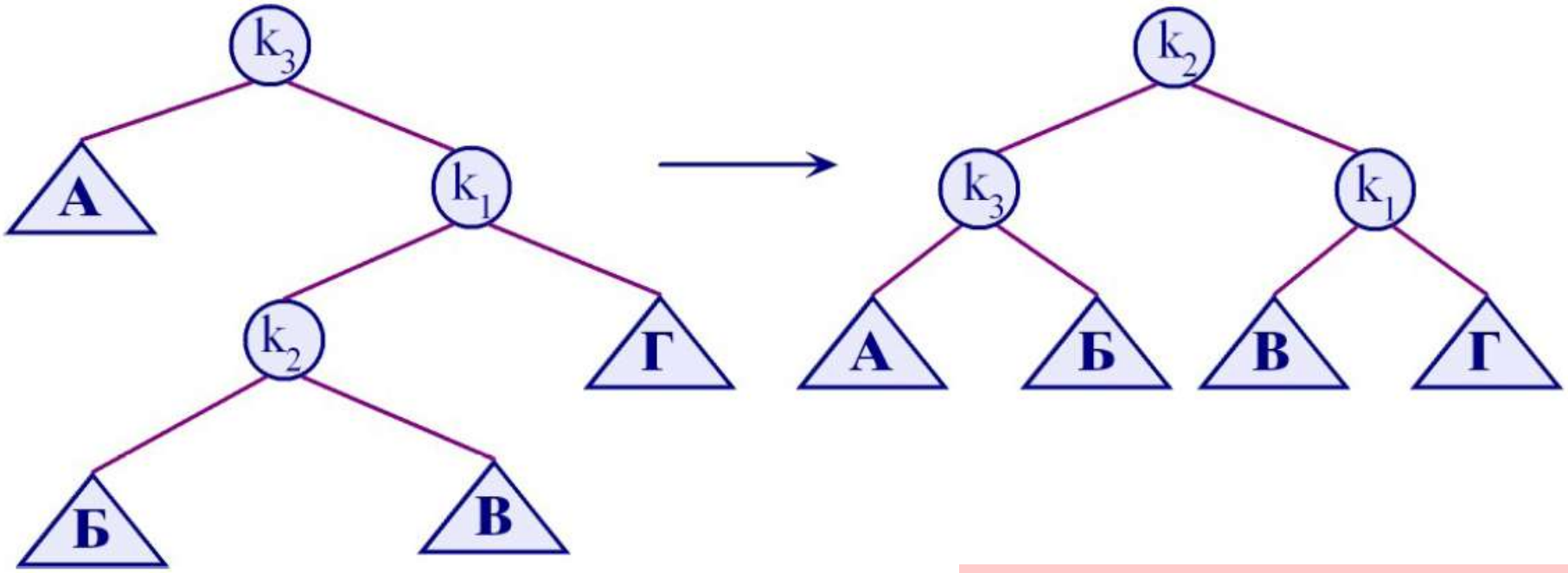
AVL дрва

- Внесувањето на клучот 14 предизвикува нарушување на балансираноста, но со примена на еднократна ротација на лево таа не се разрешува

- Тоа е поради тоа што нововнесениот клуч е елемент (елемент кој визуелно оди кон средината на поддрвото) кој е:
  - поголем од некое лево поддрво
  - помал од некое десно поддрво

AVL дрва

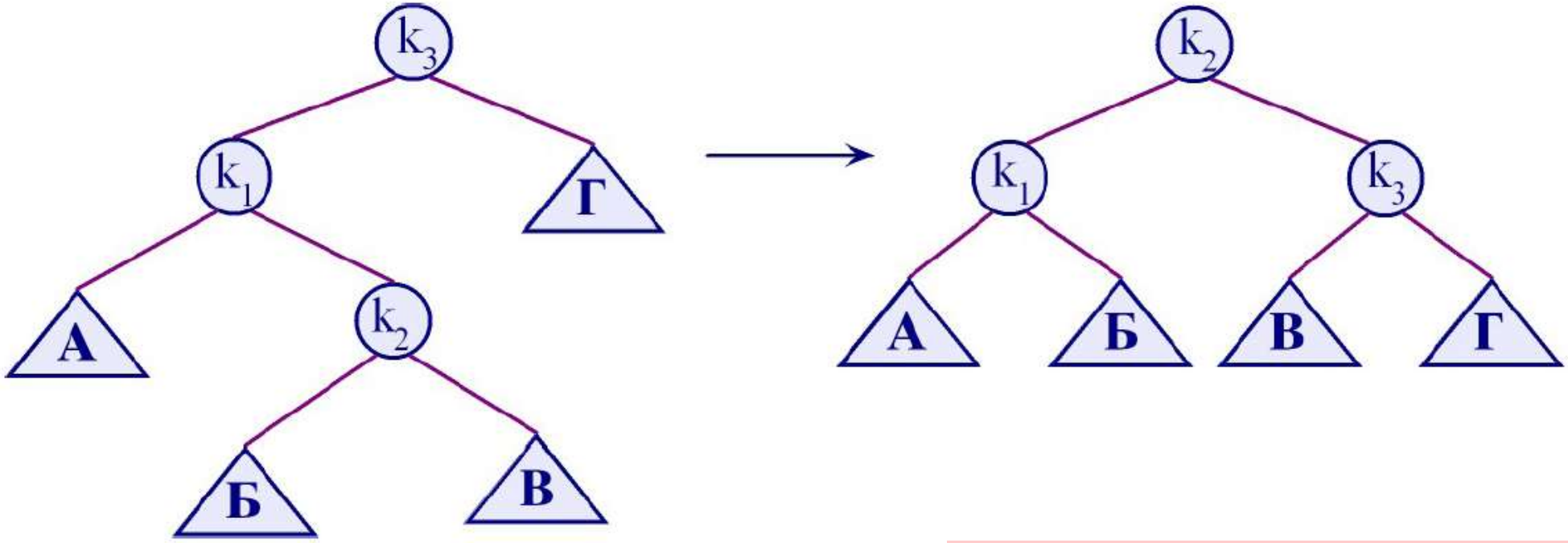
- Двократна ротација на јазел



двократна ротација
десно-лево

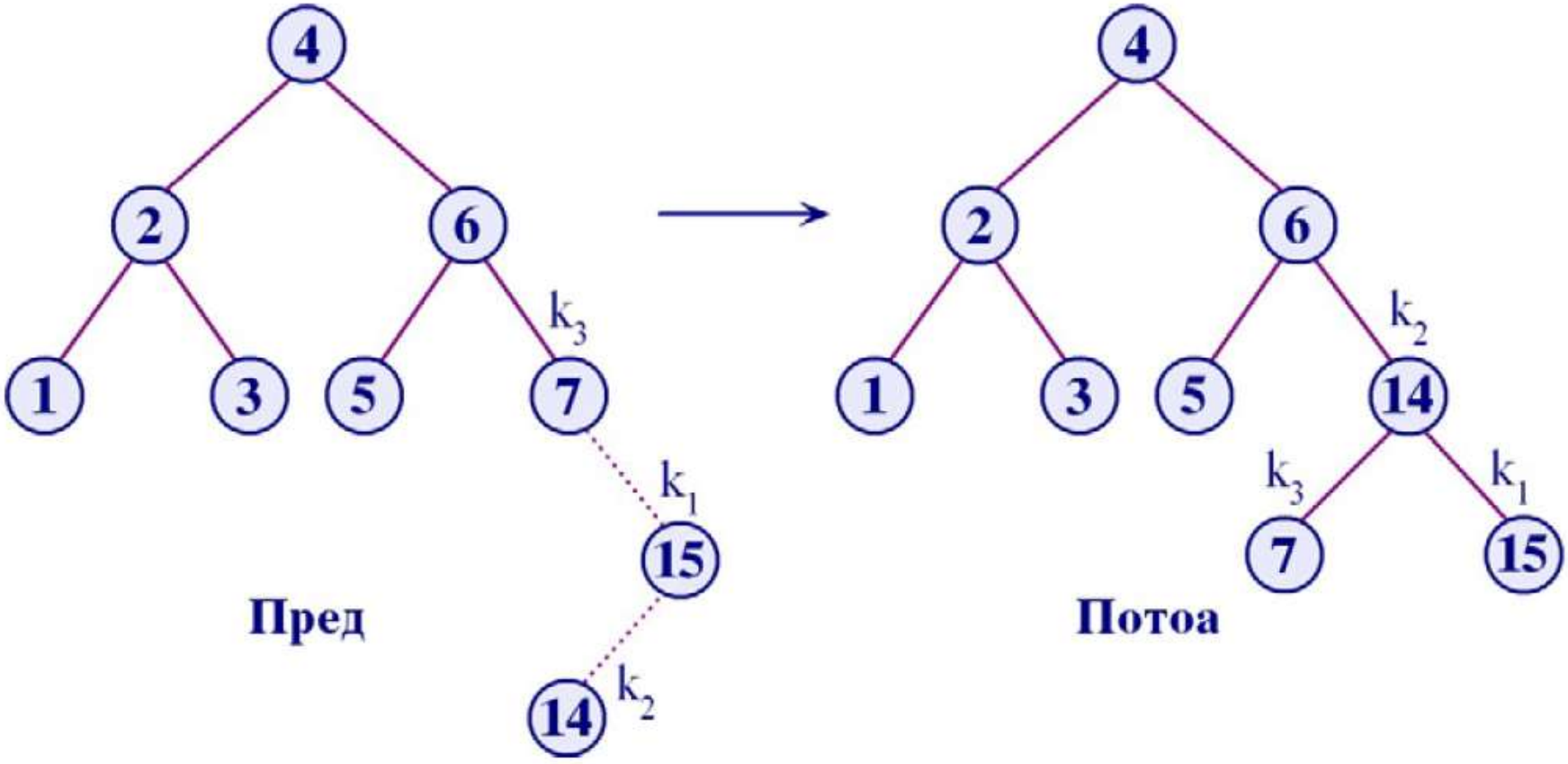
AVL дрва

- Двократна ротација на јазел



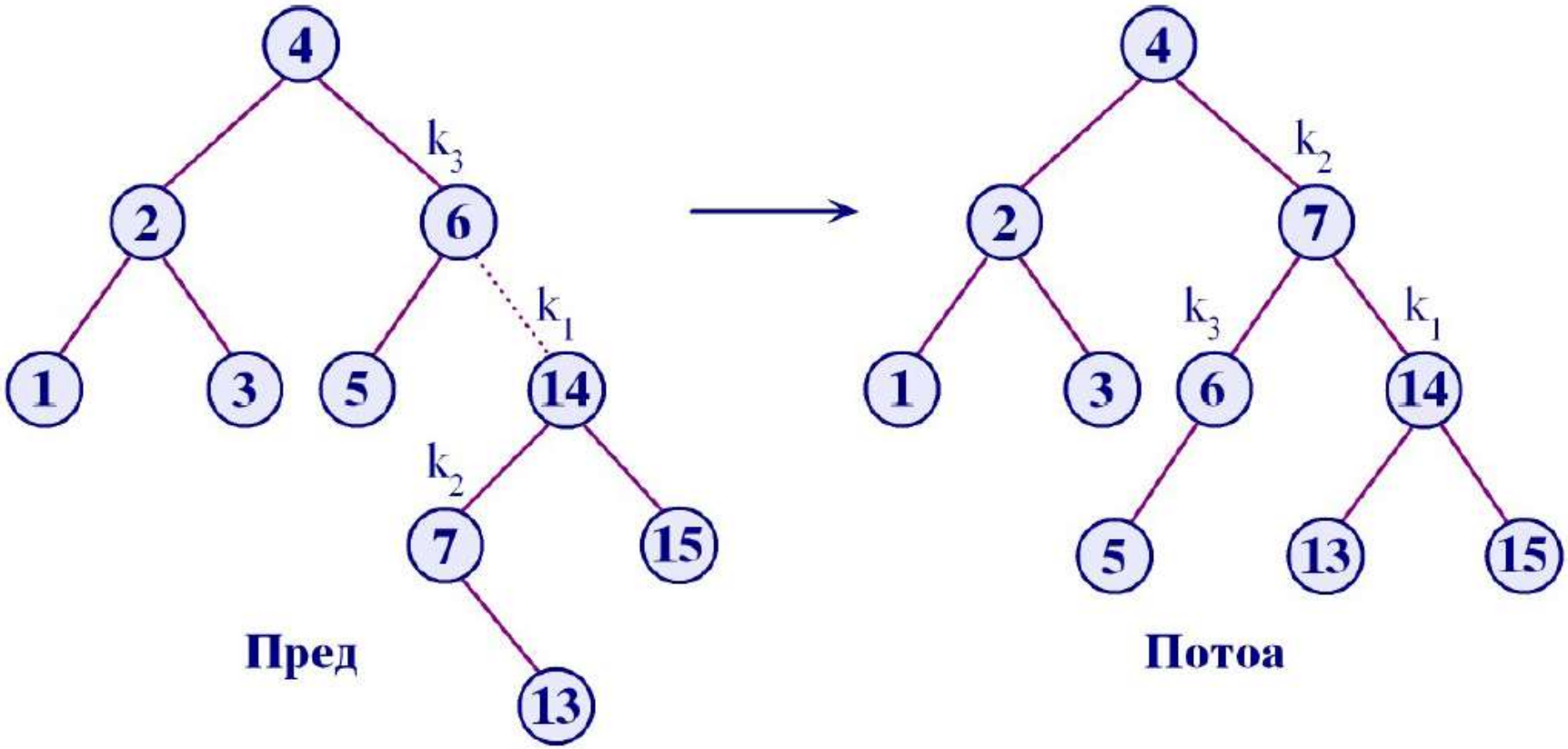
двократна ротација
лево-десно

AVL дрва

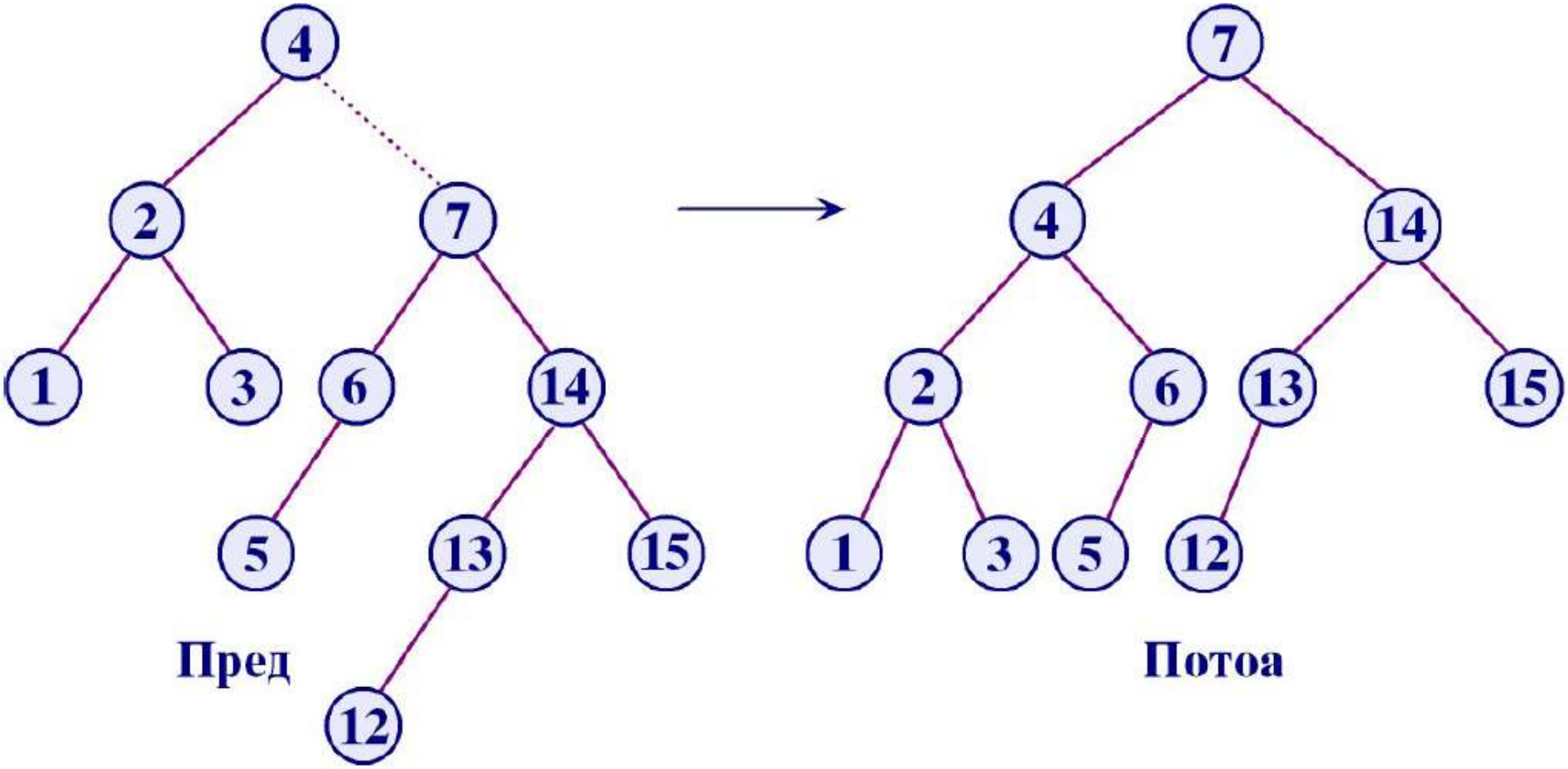


AVL дрва

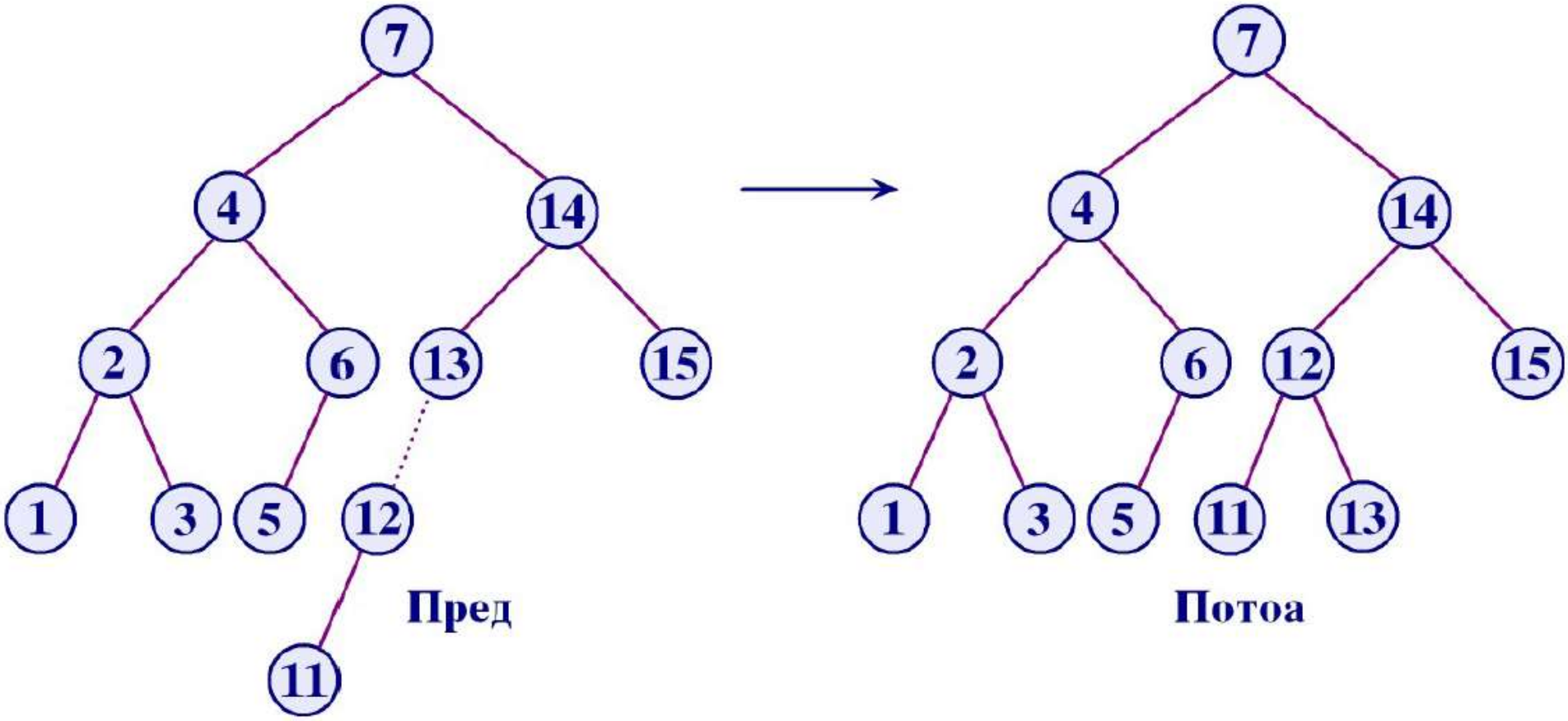
- Нека последователно продолжиме со внесување на јазлите со информации 13, 12, 11, 10, 9 и 8



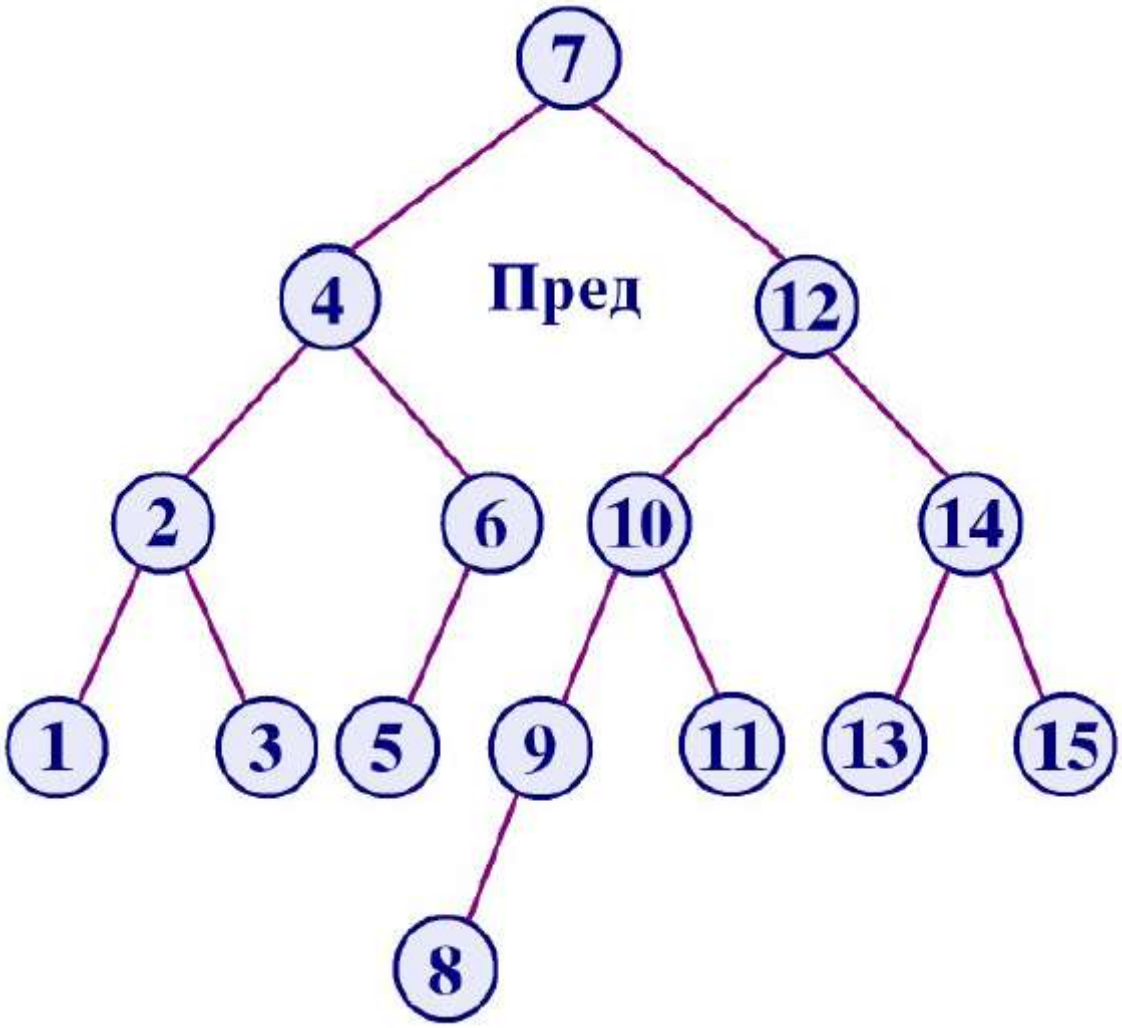
AVL дрва



AVL дрва



AVL дрва



AVL дрва

- Операцијата бришење на јазел е многу покомплицирана од операцијата вметнување на јазел
- Најчесто се применува таканаречената “мрзливо“ бришење кое ги означува јазлите кои не се користат, но физички не ги отстранува

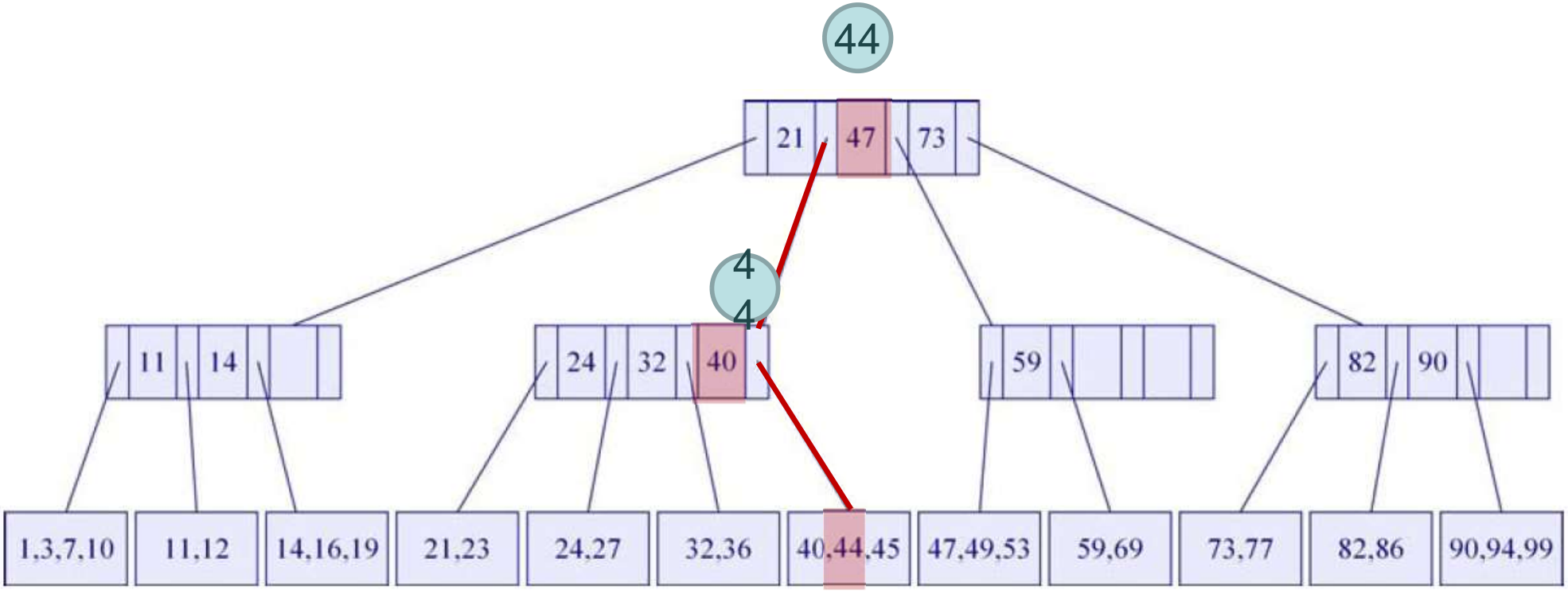
B - дрва

- Небинарни пребарувачки дрва (B, B+, B*, R…)
- B-дрво од ред *m* е дрво со следните структурни својства:
  - Коренот е или лист или има помеѓу 2 и m деца
  - Сите внатрешни јазли (освен коренот) имаат помеѓу $\lceil m/2 \rceil$ и *m* деца
  - Сите листови се на исто ниво

B - дрва

- Карактеристични операции кај B-дрвата:
  - пребарување
  - внесување на вредност
  - бришење на вредност

ПРИМЕР: B – дрво од ред 4



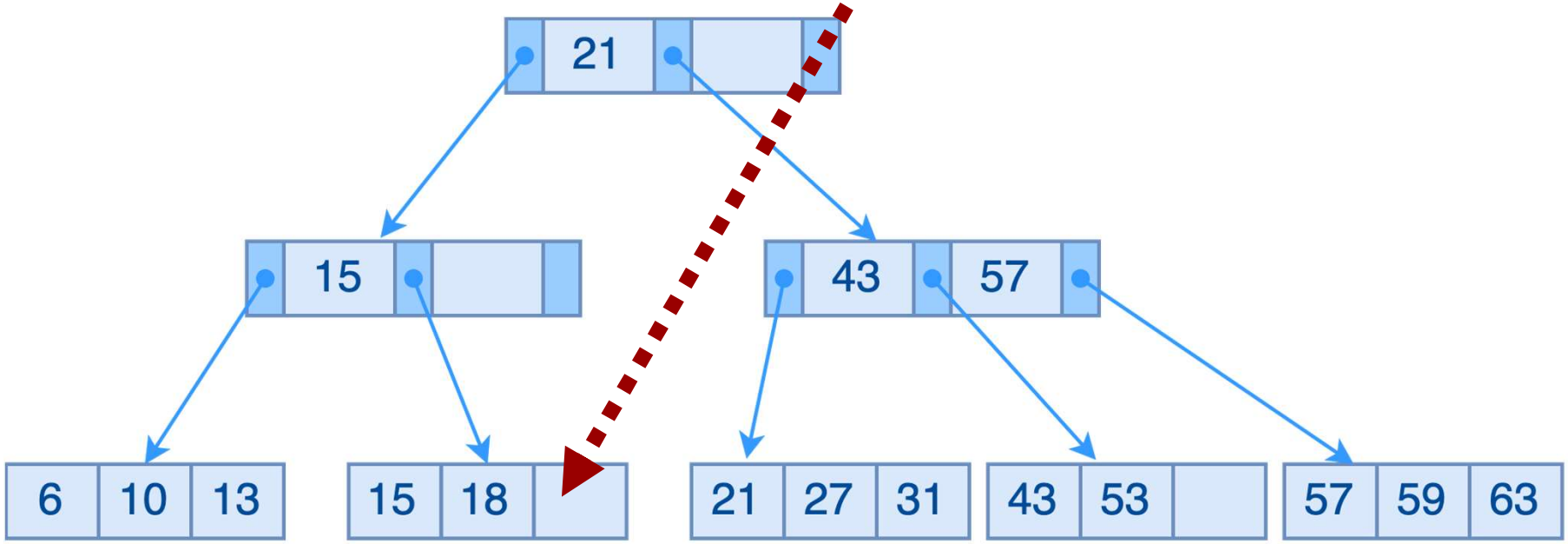
Пребарување: 44

B - дрва

- Внесување на вредност од B-дрво:
  - Пронаоѓање на јазелот каде треба да се вметне вредноста
  - Разделување на јазелот
    - ако не се зачувани принципите на B-дрво
    - јазелот има повеќе вредности од дозволеното - во овој случај се применува процесот на **разделување** на јазли
      - може да предизвика и разделување на јазли и во погорните нивоа

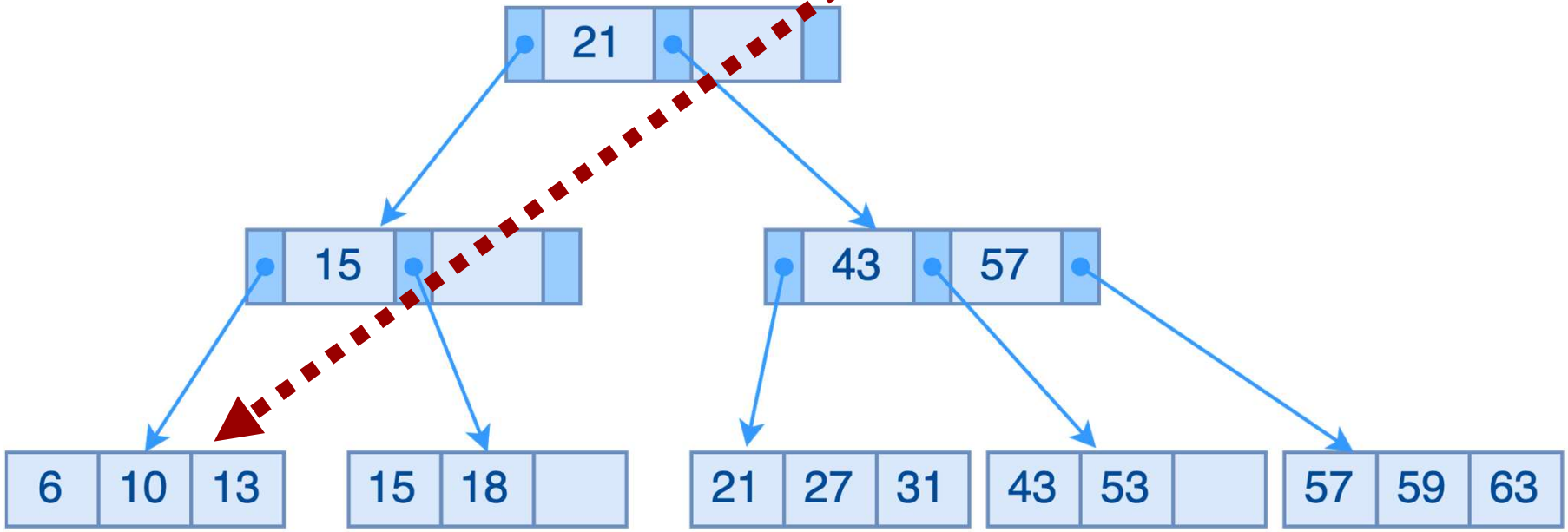
ПРИМЕР: B - дрва

- Внесување на вредности: 19, 2 , 20 и 29



ПРИМЕР: B - дрва

- Внесување на вредности: 19, 2 , 20 и 29

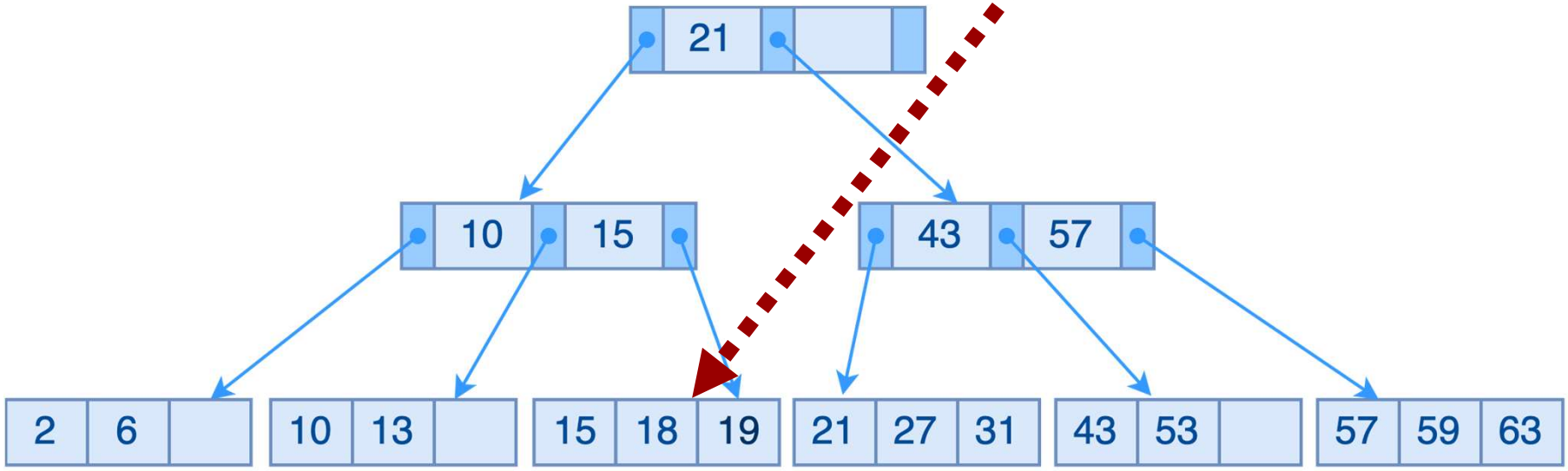


Проблем:
Овој терминален јазел е полн!

Решение:
Разделување на јазелот!

ПРИМЕР: B - дрва

- Внесување на вредности: 19, 2 , 20 и 29



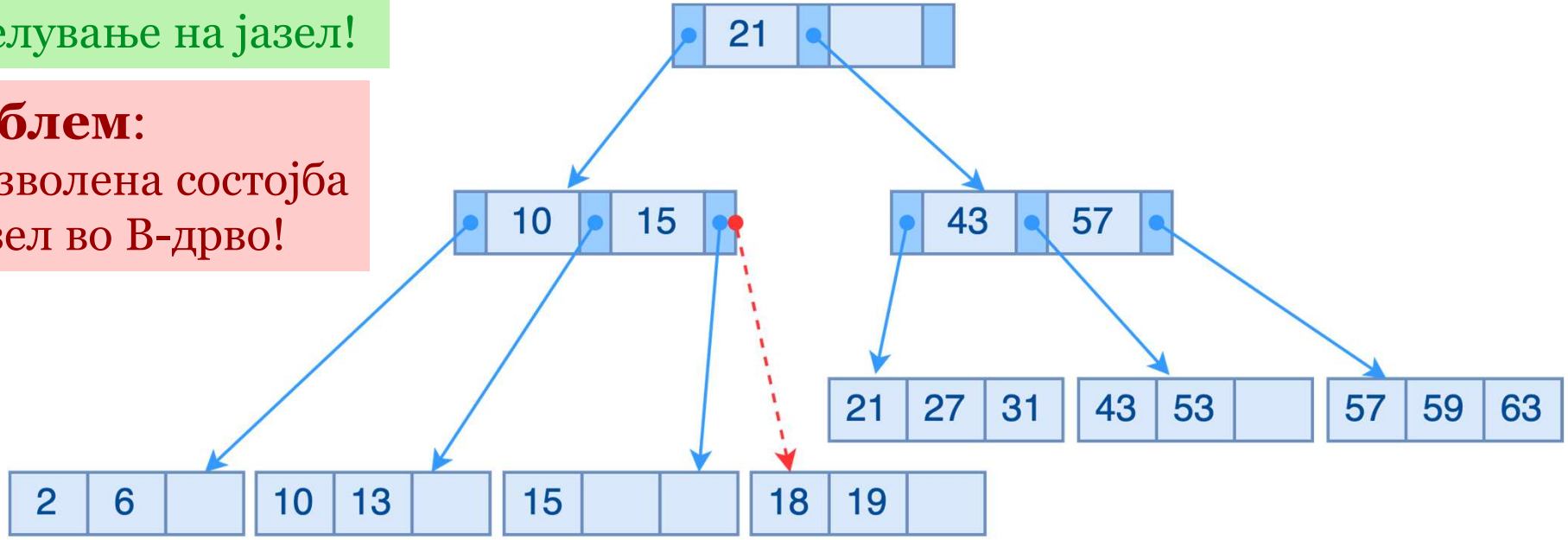
Проблем:
Овој терминален јазел е полн!

ПРИМЕР: B - дрва

- Внесување на вредности: 19, 2 , 20 и 29

Решение:
Разделување на јазел!

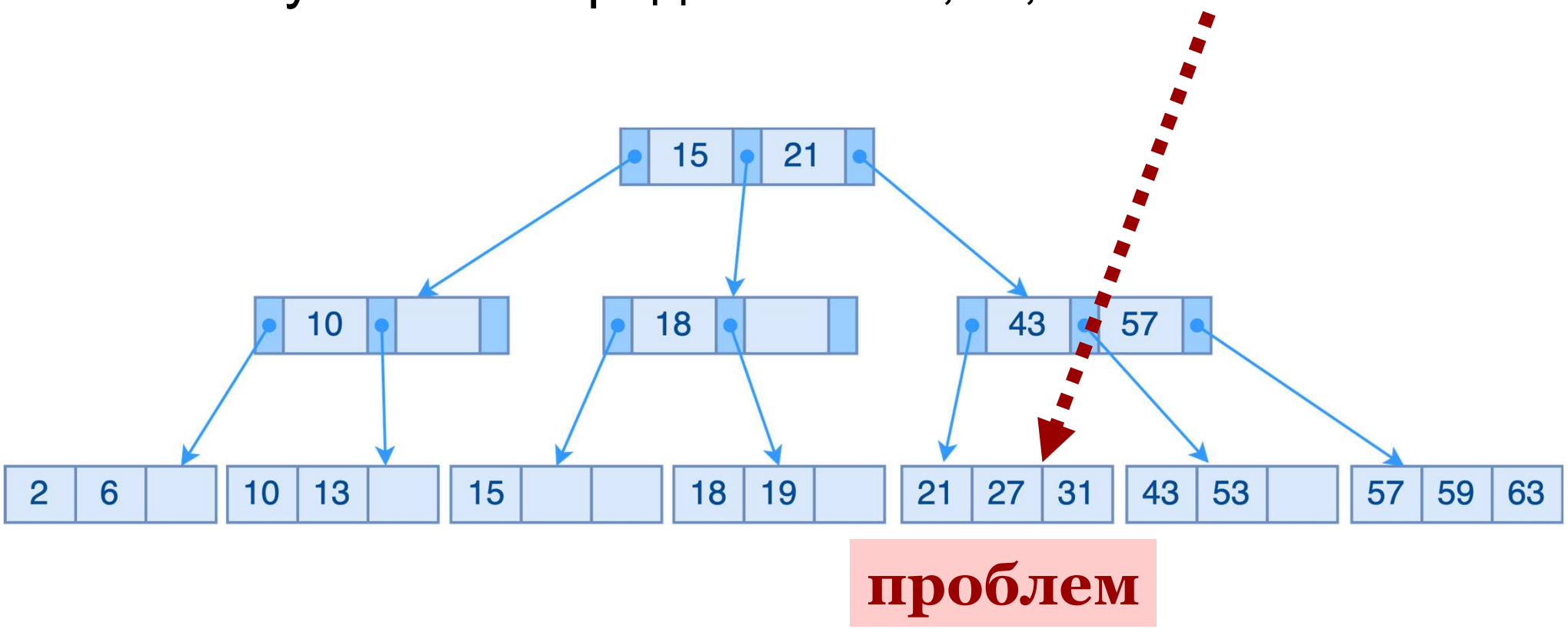
Проблем:
Недозволена состојба на јазел во B-дрво!



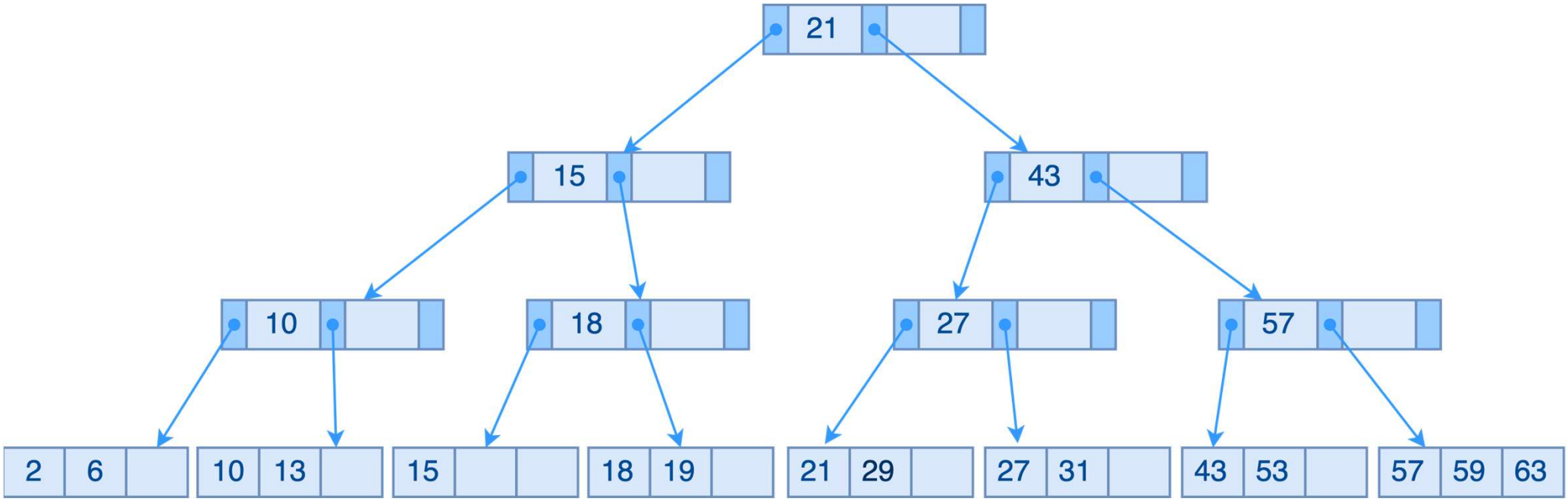
Вредноста на ? се зема да биде најлевата вредност од десното дете, а 15 се качува едно ниво погоре

ПРИМЕР: B - дрва

- Внесување на вредности: 19, 2 , 20 и 29



ПРИМЕР: B - дрва

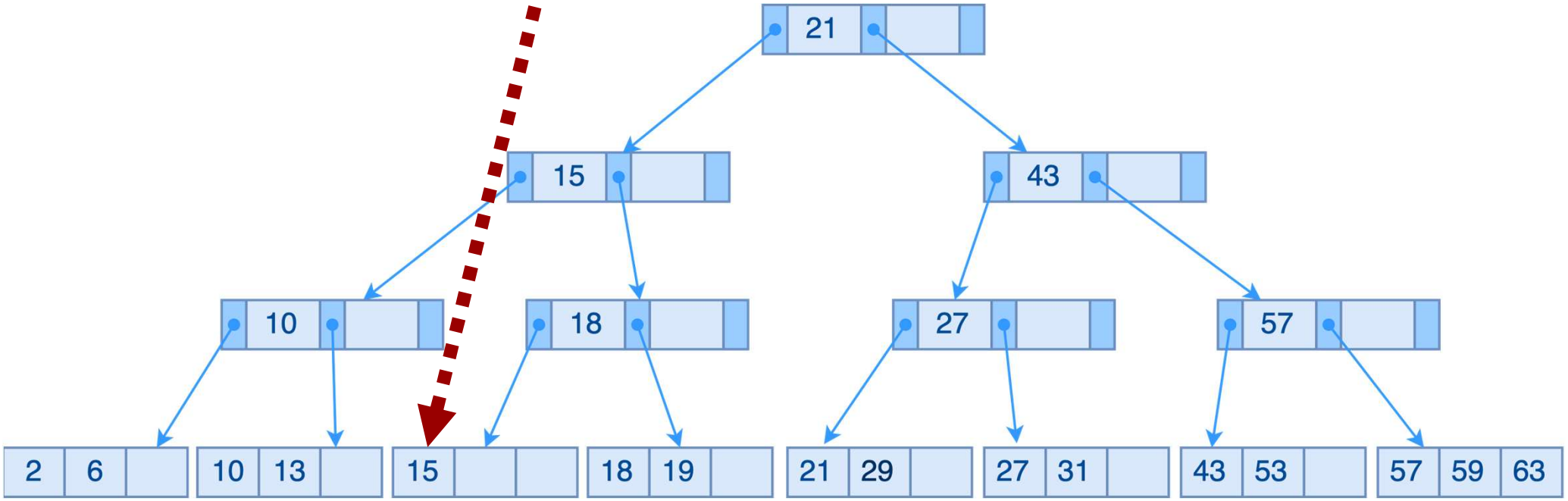


B - дрва

- Бришење на вредност од B-дрво:
  - Пронаоѓање на вредноста
  - Отстранување на вредноста од јазелот
    - зачувани се принципите на B-дрво
    - јазелот има помалку вредности од дозволеното - во овој случај се применува процесот на **спојување** на јазли

ПРИМЕР: B - дрва

- Бришење на 15



Проблем:
Јазелот ја нарушува структурата на Б-дрвото

Решение:
Спојување на соседни јазли

ПРИМЕР: B - дрва

- Бришење на 15