ძებნის ორობითი ხეები
BST (Binary Search Trees)

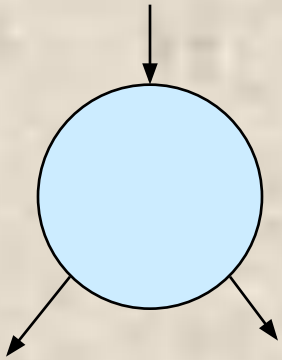
ავტორი: ზაზა გამეზარდაშვილი

ძებნის ორობითი ხეები წარმოადგენენ მონაცემთა სტრუქტურას, რომლის საშუალებითაც ხის სიმაღლის პროპორციულ $O(h)$ დროში სრულდება შემდეგი ოპერაციები:

- ძებნა;
- მინიმუმის პოვნა;
- მაქსიმუმის პოვნა;
- წინა ელემენტის პოვნა;
- მომდევნო ელემენტის პოვნა;
- ელემენტის ჩასმა;
- ელემენტის წაშლა.

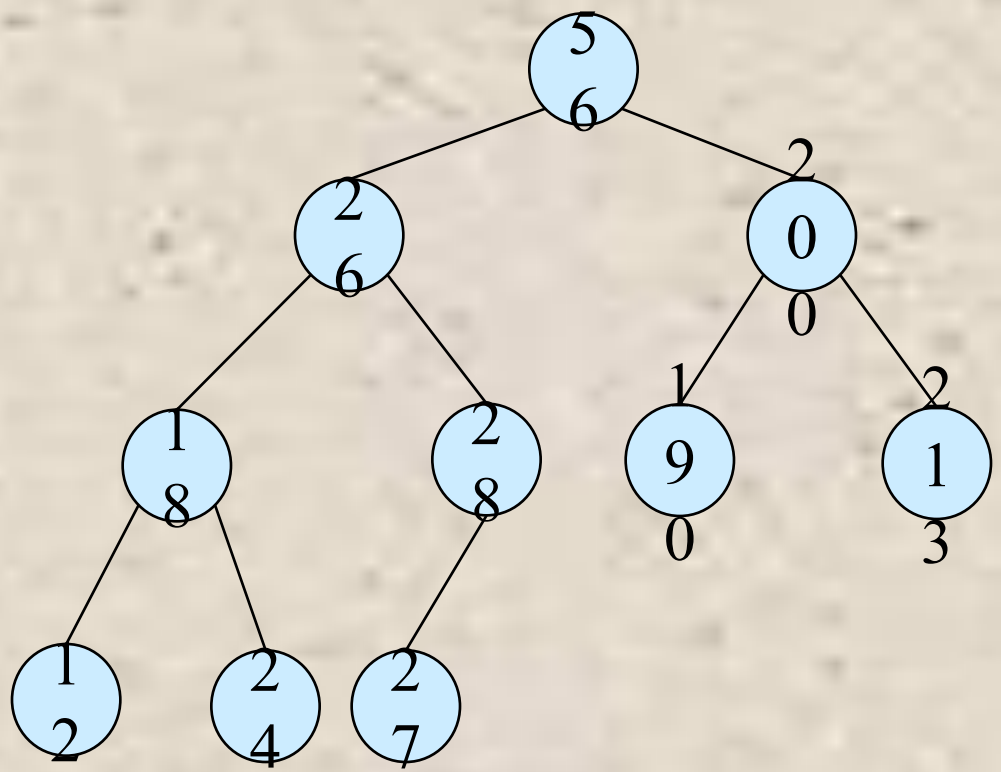
ძებნის ორობითი ხეების წარმოდგენა

- წარმოადგენს მიმთითებლებით დაკავშირებულ ხეს.
- root(T) კვანძი არის T ხის სათავე.
- ყოველი კვანძი შედგება ველებისაგან:
  - გასაღები
  - მარცხენა შვილი: მარცხენა ქვეხის სათავე.
  - მარჯვენა შვილი: მარჯვენა ქვეხის სათავე.
  - p - მშობლის მიმთითებელი. p[root[T]] = NIL

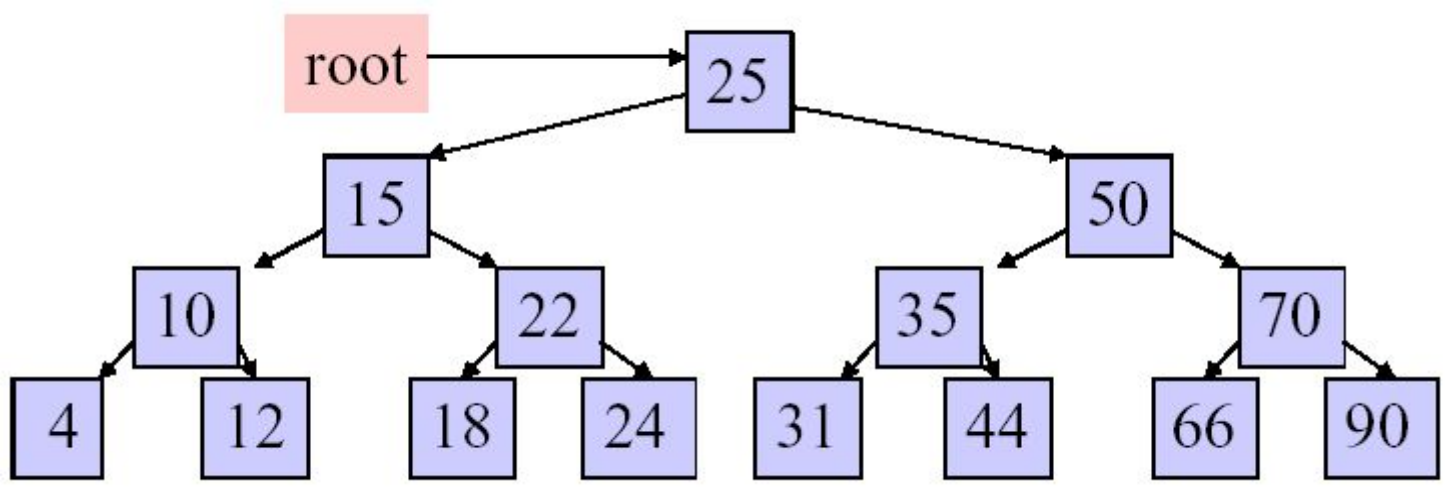


ძებნის ორობითი ხის თვისება

- ძებნის ორობითი ხის გასაღებები უნდა აკმაყოფილებდნენ შემდეგ პირობას:
  - ∀ *y*-სათვის *x*-ის მარცხენა ქვეხიდან, *key*[*y*] ≤ *key*[*x*].
  - ∀ *y*-სათვის *x*-ის მარჯვენა ქვეხიდან *key*[*y*] ≥ *key*[*x*].



ძებნის ხის შემოვლის ალგორითმები



Inorder – 4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

Preorder – 25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

Postorder – 4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25

ძებნის ხის შემოვლის ალგორითმები

```
void print_Tree_Inorder(struct
    node* node) {
if (node == NULL) return;
printTree(node->left);
printf("%d ", node->data);
printTree(node->right);
}
```

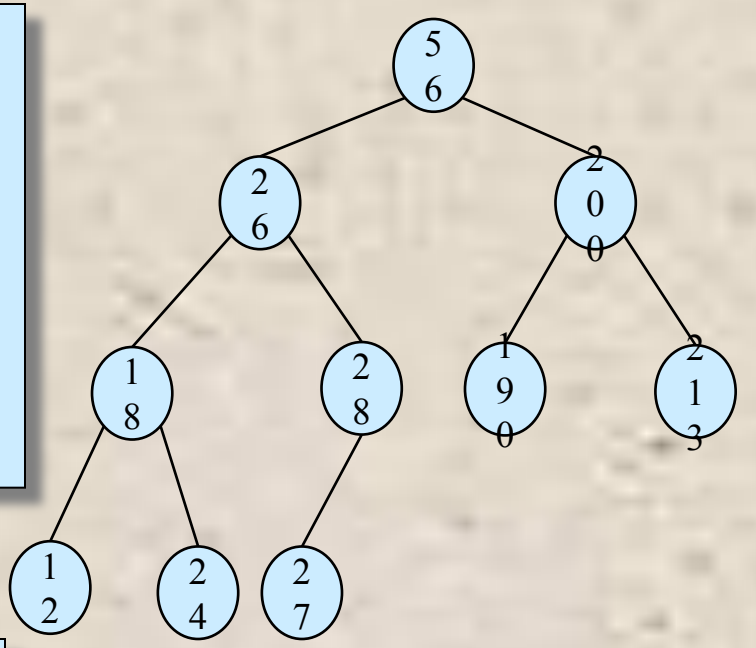
```
void print_Tree_Preorder(struct
    node* node) {
if (node == NULL) return;
printf("%d ", node->data);
printTree(node->left);
printTree(node->right);
}
```

```
void print_Tree_Postorder(struct node* node) {
if (node == NULL) return;
printTree(node->left);
printTree(node->right);
printf("%d ", node->data);
}
```

ელემენტის ძებნა

Recursive-Tree-Search(x, k)

1. **if** x = NIL *or* k = *key*[x]
2. **then** return x
3. **if** k < *key*[x]
4. **then** return Tree-Search(*left*[x], k)
5. **else** return Tree-Search(*right*[x], k)



Iterative-Tree-Search(x, k)

1. **while** $x \neq NIL$ **and** $k \neq$ *key*[x]
2. **do if** k < *key*[x]
3. **then** $x \leftarrow$ *left*[x]
4. **else** $x \leftarrow$ *right*[x]
5. **return** x

მუშაობის დრო: $O(h)$

Min & Max-ის პოვნა

- ძებნის ორობითი ხის თვისება განაპირობებს:
  - » მინიმუმი არის სათავიდან ყველაზე მარცხენა კვანძში.
  - » მაქსიმუმი არის სათავიდან ყველაზე მარჯვენა კვანძში.

Tree-Minimum(*x*)

1. **while** *left*[*x*] ≠ *NIL*
2. **do** *x* ← *left*[*x*]
3. **return** *x*

Tree-Maximum(*x*)

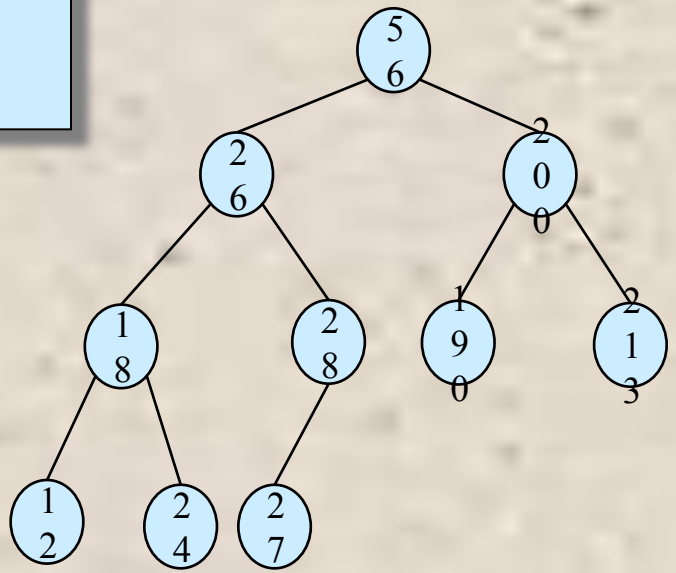
1. **while** *right*[*x*] ≠ *NIL*
2. **do** *x* ← *right*[*x*]
3. **return** *x*

წინა და მომდევნო ელემენტების პოვნა

- *x* ელემენტის მომდევნო ელემენტი (Successor) არის ისეთი *y*, რომლის მნიშვნელობაც უმცირესია x-ზე მეტ ელემენტებს შორის.
- უდიდესი ელემენტის მომდევნო არის NIL.
- ძებნის დროს განიხილება ორი შემთხვევა:
  - თუ *x* ელემენტს აქვს არაცარიელი მარჯვენა ქვეხე, მაშინ მისი მომდევნო ელემენტია ამ ქვეხის მინიმუმი.
  - თუ *x* ელემენტის მარჯვენა ქვეხე ცარიელია: თუკი მარჯვენა ქვეხე ცარიელია, მაშინ ჩვენ ვმოძრაობთ x-დან ზემოთ, ვიდრე არ ვიპოვით წვეროს, რომელიც თავისი მშობლის მარცხენა შვილია. სწორედ ეს მშობელი (თუკი ის არსებობს) წარმოადგენს საძებნ ელემენტს.

მომდევნო ელემენტის პოვნის ფსევდოკოდი

- Tree-Successor(*x*)
- **if** *right*[*x*] ≠ *NIL*
- 2. **then** return Tree-Minimum(*right*[*x*])
- 3. y ← *p*[*x*]
- 4. **while** *y* ≠ *NIL* **and** *x* = *right*[*y*]
- 5. **do** *x* ← *y*
- 6. *y* ← *p*[*y*]
- 7. **return** *y*

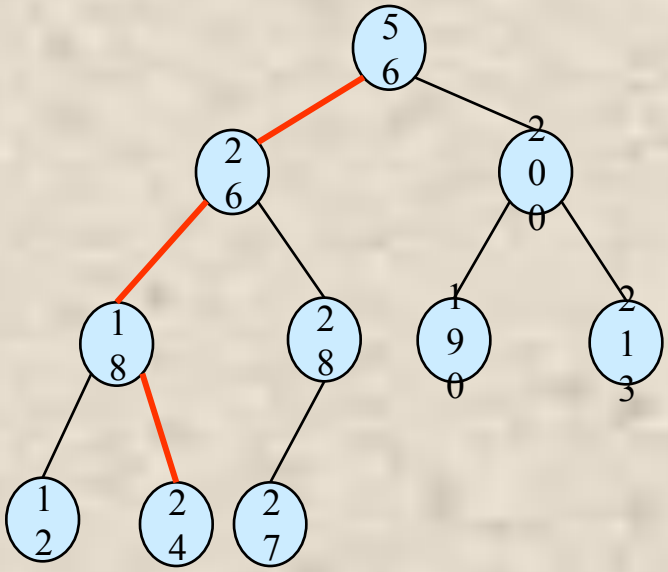


წინა ელემენტის პოვნა სიმეტრიულია.

მუშაობის დრო: $O(h)$.

მუშაობის დრო O(h)-ის ტოლია, რადგან მოძრაობა ხდება ან მხოლოდ ზემოთ, ან მხოლოდ ქვემოთ.

ელემენტის ჩასმა ძებნის ორობით ხეში



- ინიციალიზაცია: $O(1)$.
- While ციკლი 3-7 სტრიქონებში ეძებს z-ის ჩასასმელ ადგილს. სჭირდება $O(h)$ დრო.
- 8-13 სტრიქონებში ხდება ელემენტის ჩასმა: $O(1)$

⇒ სულ: $O(h)$ დრო ელემენტის ჩასასმელად.

Tree-Insert(T, z)

```
1  y ← NIL
2  x ← root[T]
3  while x ≠ NIL
4      do y ← x
5          if key[z] < key[x]
6              then x ← left[x]
7              else x ← right[x]
8  p[z] ← y
9  if y = NIL
10     then root[t] ← z
11     else if key[z] < key[y]
12         then  left[y] ← z
13         else right[y] ← z
```

ელემენტის წაშლა ძებნის ორობით ხეში

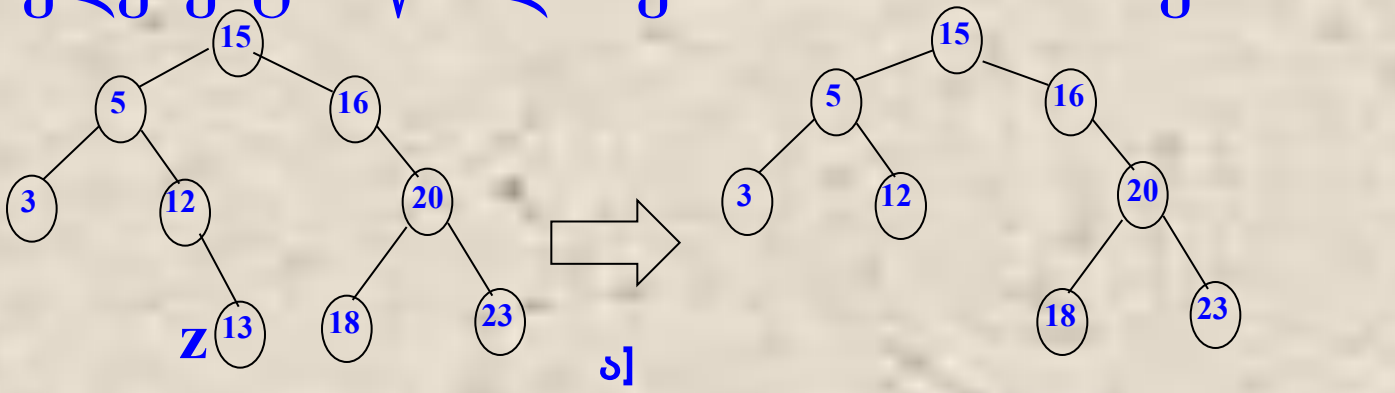
ელემენტის წაშლისას შესაძლებელია სამი შემთხვევა:

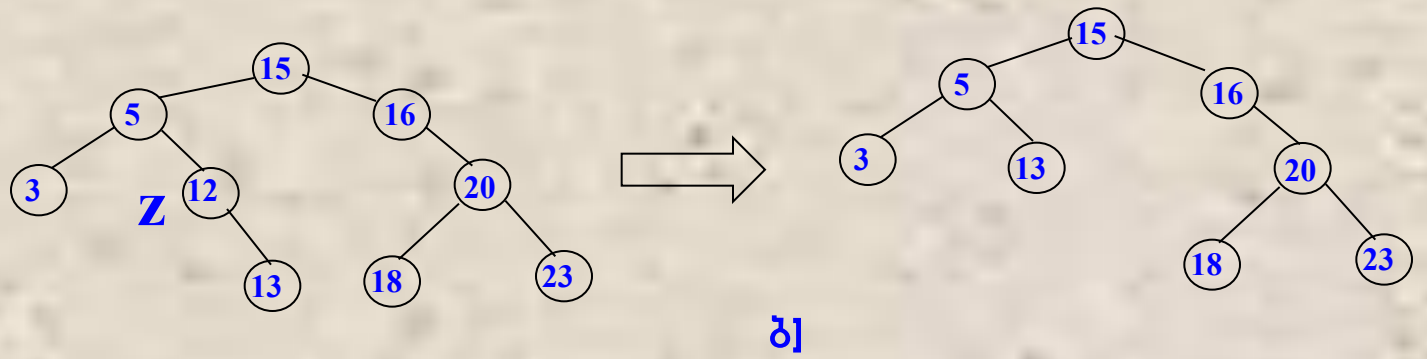
ა) z-ს არ გააჩნია შვილები, ამ დროს z-ის წასაშლელად საკმარისია მისი მშობლის შესაბამის მინდორში მოვათავსოთ nil;

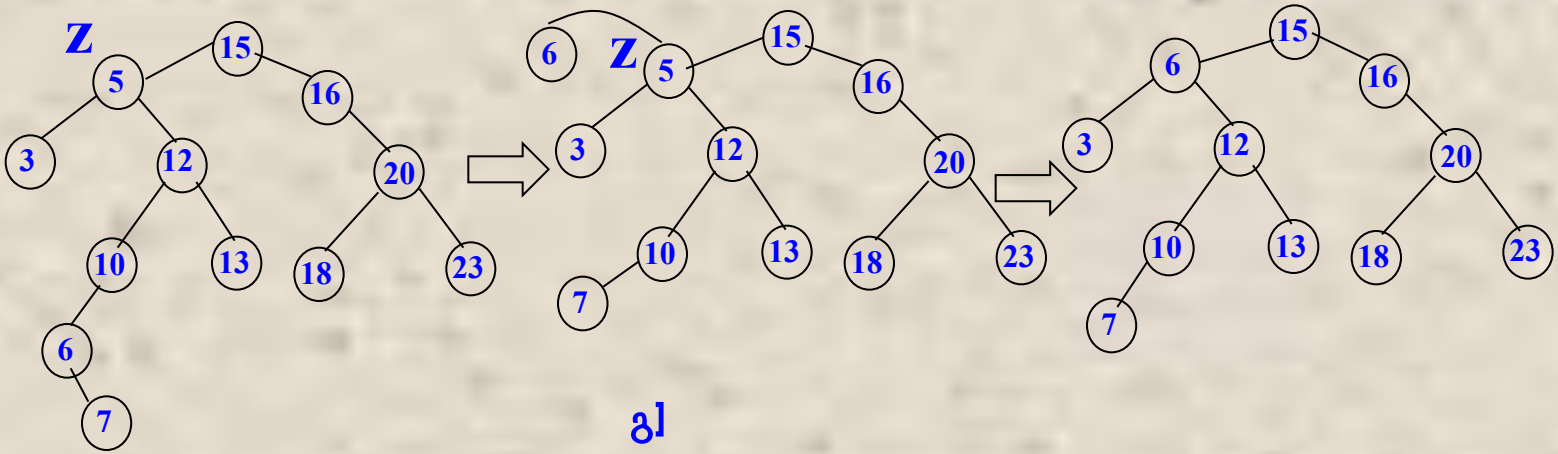
ბ) z-ს გააჩნია ერთი შვილი, ამ შემთხვევაში z "ამოიშლება" მისი მშობლისა და შვილის პირდაპირი შეერთებით;

გ) z-ს გააჩნია ორი შვილი – აქ წვეროს წაშლამდე საჭიროა გარკვეული მოსამზადებელი სამუშაოების ჩატარება: უნდა მოვძებნოთ გასაღების სიდიდის მიხედვით მომდევნო y ელემენტი. მას არ ეყოლება მარცხენა შვილი (ძებნის ორობით ხეში მტკიცდება შემდეგი თვისება: თუ ელემენტს გააჩნია ორი შვილი, მაშინ გასაღების სიდიდის მიხედვით მომდევნო ელემენტს არ გააჩნია მარცხენა შვილი, ხოლო გასაღების სიდიდის მიხედვით წინას – მარჯვენა შვილი). ამის შემდეგ y წვეროს გასაღები და დამატებითი მონაცემები z წვეროს შესაბამის მინდვრებში, ხოლო თავად y წვერო წავშალოთ ზემოთ ნახსენები (ბ) ხერხით.

ელემენტის წაშლა ძებნის ორობით ხეში







ელემენტის წაშლა ძებნის ორობით ხეში

Tree-Delete(*T*, *z*)

- /* Determine which node to splice out: either *z* or *z*'s successor. */
- **if** *left*[*z*] = NIL **or** *right*[*z*] = NIL
- **then** *y* ← *z* **else** *y* ← Tree-Successor[z]
- /* Set *x* to a non-NIL child of *x*, or to NIL if *y* has no children. */
- **if** *left*[*y*] ≠ NIL
- **then** *x* ← *left*[*y*] **else** *x* ← *right*[*y*]
- /* *y* is removed from the tree by manipulating pointers of *p*[*y*] and *x* */
- **if** *x* ≠ NIL
- **then** *p*[*x*] ← *p*[*y*]

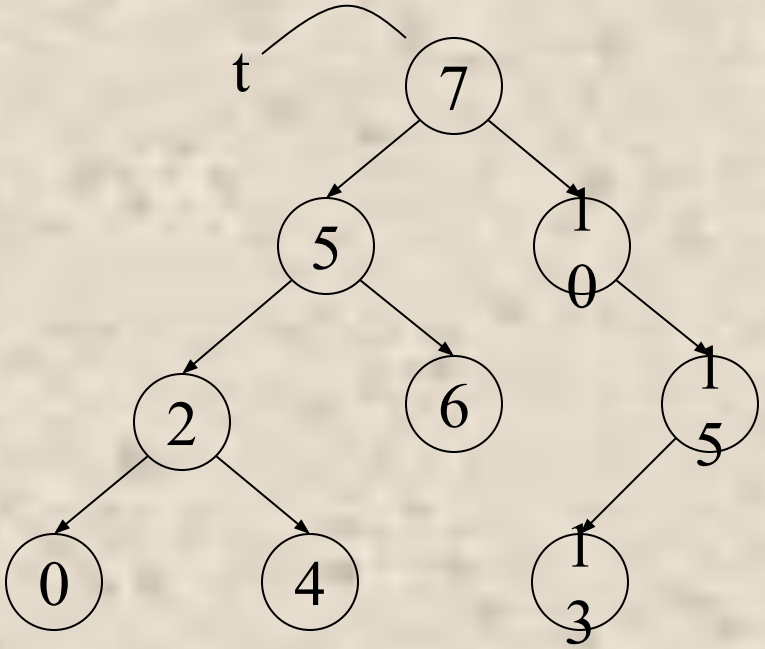
9. **if** *p*[*y*] = NIL
10. **then** *root*[*T*] ← *x*
11. **else if** *y* ← *left*[*p*[*i*]]
12. **then** *left*[*p*[*y*]] ← *x* **else** *right*[*p*[*y*]] ← *x*
13. /* If *z*'s successor was spliced out, copy its data into *z* */
14. **if** *y* ≠ *z*
15. **then** *key*[*z*] ← *key*[*y*]
16. copy *y*'s satellite data into *z*.
17. **return** *y*

მუშაობის დრო: $O(h)$

რიგობრივი სტატისტიკა ძებნის ხეში

მიზანი: მოვძებნოთ k-ური ელემენტი (ზრდადობით) N-ელემენტიან ძებნის ორობით ხეში, სადაც 1 <= k <= N.

მაგალითად, ხშირად გვჭირდება, მოვძებნოთ მედიანა დინამიურად ცვალებად მასივში.



t.size() □ 9

t.elementAt(1) □ 0 // მინიმუმი

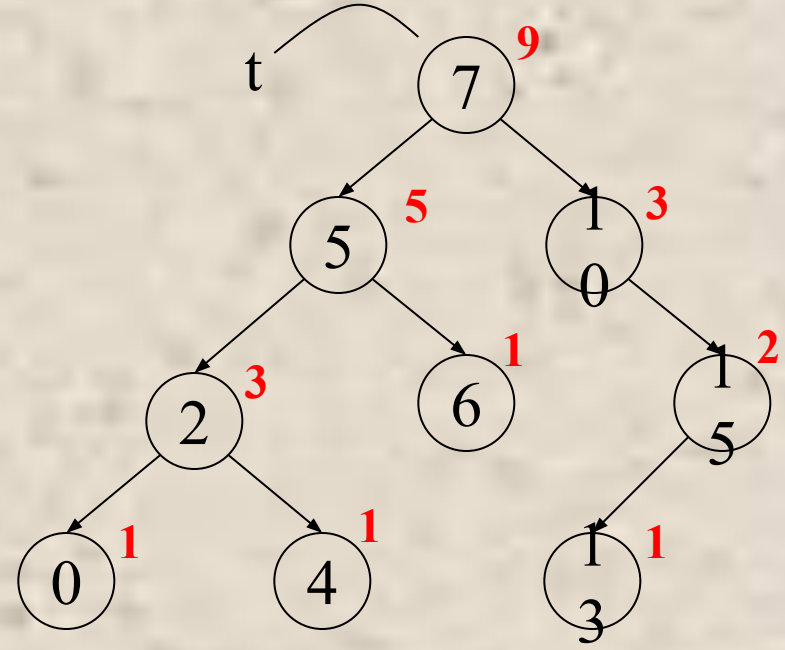
t.elementAt(5) □ 6 // მედიანა

t.elementAt(9) □ 15 // მაქსიმუმი

რიგობრივი სტატისტიკა ძებნის ხეში

თითოეულ კვანძში შევინახოთ შესაბამისი ქვეხის ელემენტების რაოდენობა.

ინფორმაციის განახლება ხდება ხეში ცვლილების პარალელურად.



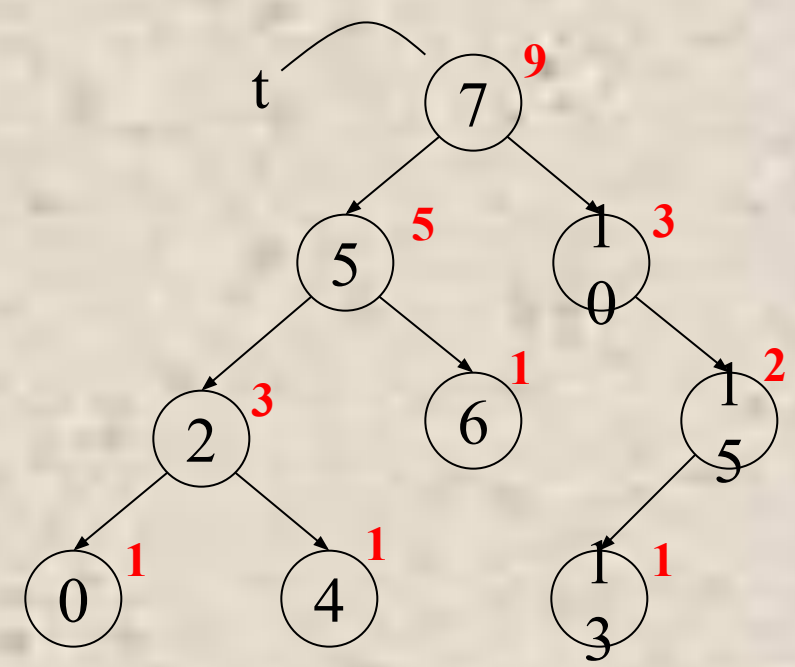
რიგობრივი სტატისტიკა ძებნის ხეში

k-ური ელემენტის ძებნა:

თუ *k* == left.size + 1, return

თუ k < left.size + 1, ვეძებოთ *k*-ური ელემენტი მარცხენა ქვეხეში

თუ K > left.size + 1, ვეძებოთ (*k-left.size-1*)-ური ელემენტი მარჯვენა ქვეხეში



2224. Team Selection

- he Interpeninsular Olympiad in Informatics is coming and the leaders of the Balkan Peninsula Team have to choose the best contestants on the Balkans. Fortunately, the leaders could choose the members of the team among N very good contestants, numbered from 1 to N ($3 \le N \le 500000$). In order to select the best contestants the leaders organized three competitions. Each of the N contestants took part in all three competitions and there were no two contestants with equal results on any of the competitions. We say that contestant A is better than another contestant B when A is ranked before B in all of the competitions. A contestant A is said to be excellent if no other contestant is better than A. The leaders of the Balkan Peninsula Team would like to know the number of excellent contestants.
- Write a program, which for given N and the three competitions results, computes the number of excellent contestants.
- **Input**
- The input data are given as four lines. The first line contains the number N. The next three lines show the rankings for the three competitions. Each of these lines contains the identification numbers of the contestants, separated by single spaces, in the order of their ranking from first to last place.
- **Output**
- The output should contain one line with a single number written on it: the number of the excellent.
- **Example**
- **Input**
- 10
- 2 5 3 8 10 7 1 6 9 4
- 1 2 3 4 5 6 7 8 9 10
- 3 8 7 10 5 4 1 2 6 9
- **Output**
- 4
- *Note: The excellent contestants are those numbered with 1, 2, 3 and 5.*