

A.

$$R \rightarrow \Theta(n)$$

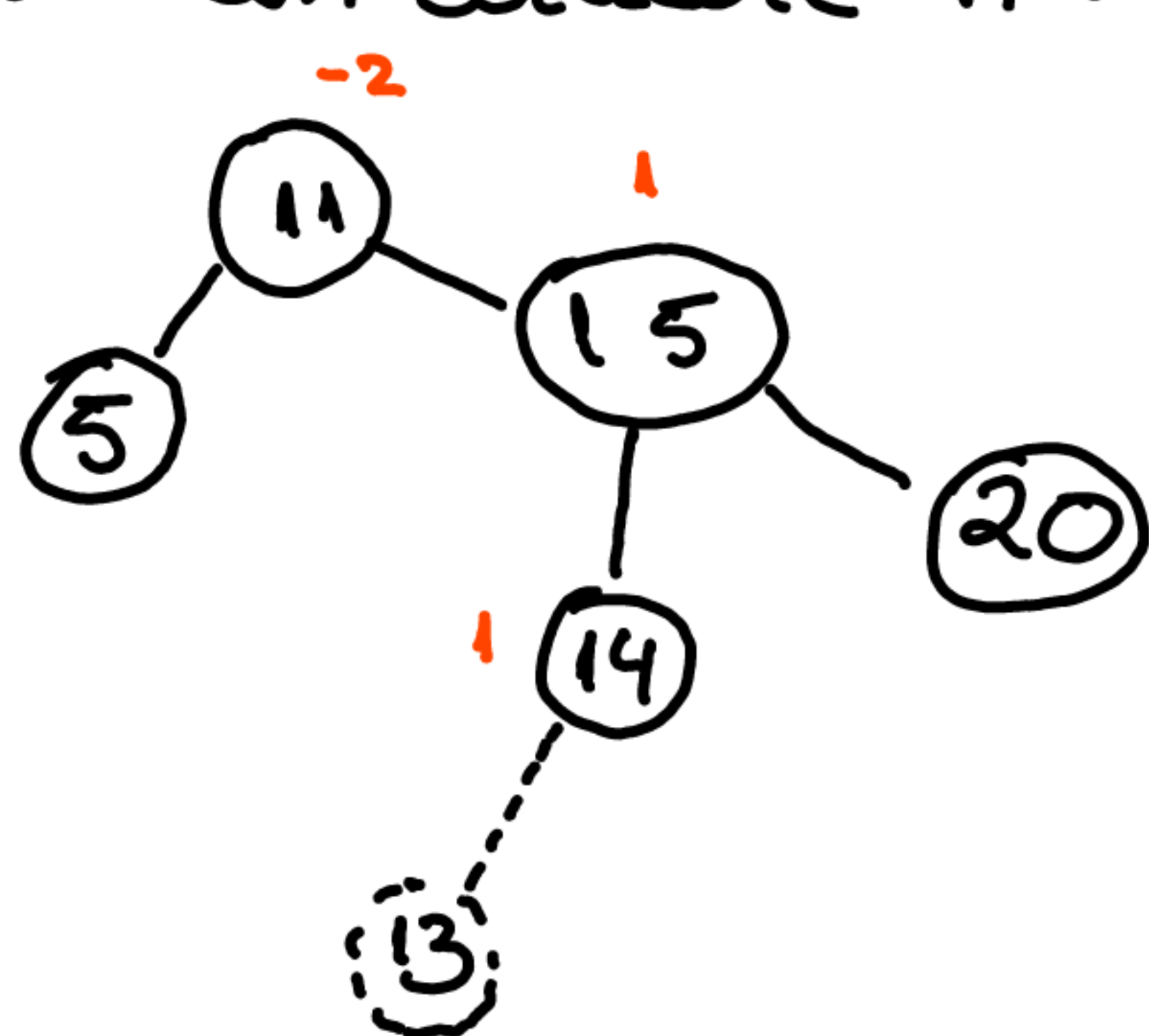
Subalgoritmul prelucrare nu are caz favorabil / defavorabil  $\Rightarrow$  timp mediu și defavorabil vor fi egali

$$T(n) = \sum_{i=1}^n \sum_{j=1}^n 1 = \sum_{i=1}^n n = n^2 \Rightarrow \text{complexitate de}$$

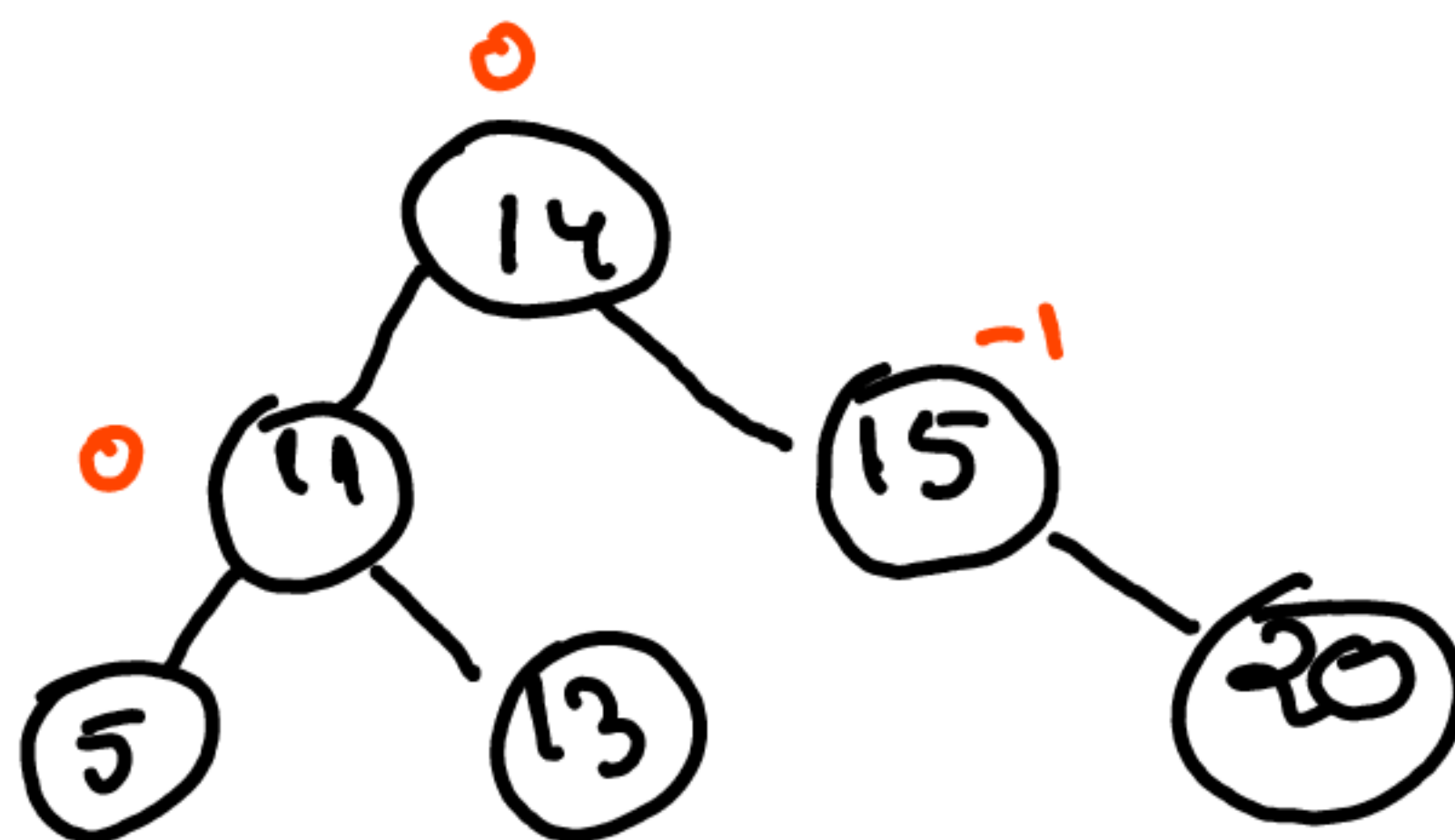
$$\text{timp} : \Theta(n^2)$$

în funcția  $R$ ,  $i$ -le va lua întreaga valoare între  $1$  și  $n$

B. Să se realizeze op. de dubla rot. spre stânga într-un arbore AVL.



DRS  $\rightarrow$



↑ dorim să adăugăm nodul 13 în arborele AVL

- totuși, prin adăugarea lui 13, arborele nu ar mai fi echilibrat (fact. de echilibru al rădăcinii este -2)
- având fact. de echilibru al rădăcinii -2, știm că este necesară o rotație spre stânga, însă o singură rotație nu este suficientă  $\Rightarrow$  dublă rotație spre st



$\Rightarrow$  14 devine rădăcină (dacă am fi efectuat o singură rotație era 15)

- părintele lui 14 este mai mare  $\Rightarrow$  sub. drept îl are ca rădăcină pe 15
- rădăcina inițială 11 este mai mică decât 14  $\Rightarrow$  sub. stâng îl are ca rădăcină pe 11
- 11 își păstrează descendentul stâng, iar ca descendent drept îl capătă pe 13
- 15 își păstrează descendentul drept

PARALELĂ CU TEORIA:

$A = 11$	$A_1 = 5$
$B = 15$	$A_2 = 13$
$C = 14$	$A_4 = 20$

c1.

Cum AVL  $\rightarrow$  arbore echilibrat  $\Rightarrow$  înălțimea =  $\log_2 n$ , unde  $n$  = nr. de noduri ale arborelui. "adaucimea" Arborelui, răspunsurile sunt: a), c), d)

Totuși, operațiile au această complexitate pentru că nu este necesară parcurgerea tuturor elementelor pt. efectuarea operațiilor de Căutare (adaugare, ștergere, căutare), ci doar parcurgerea unui "drum", datorită existenței relației dintre elemente.

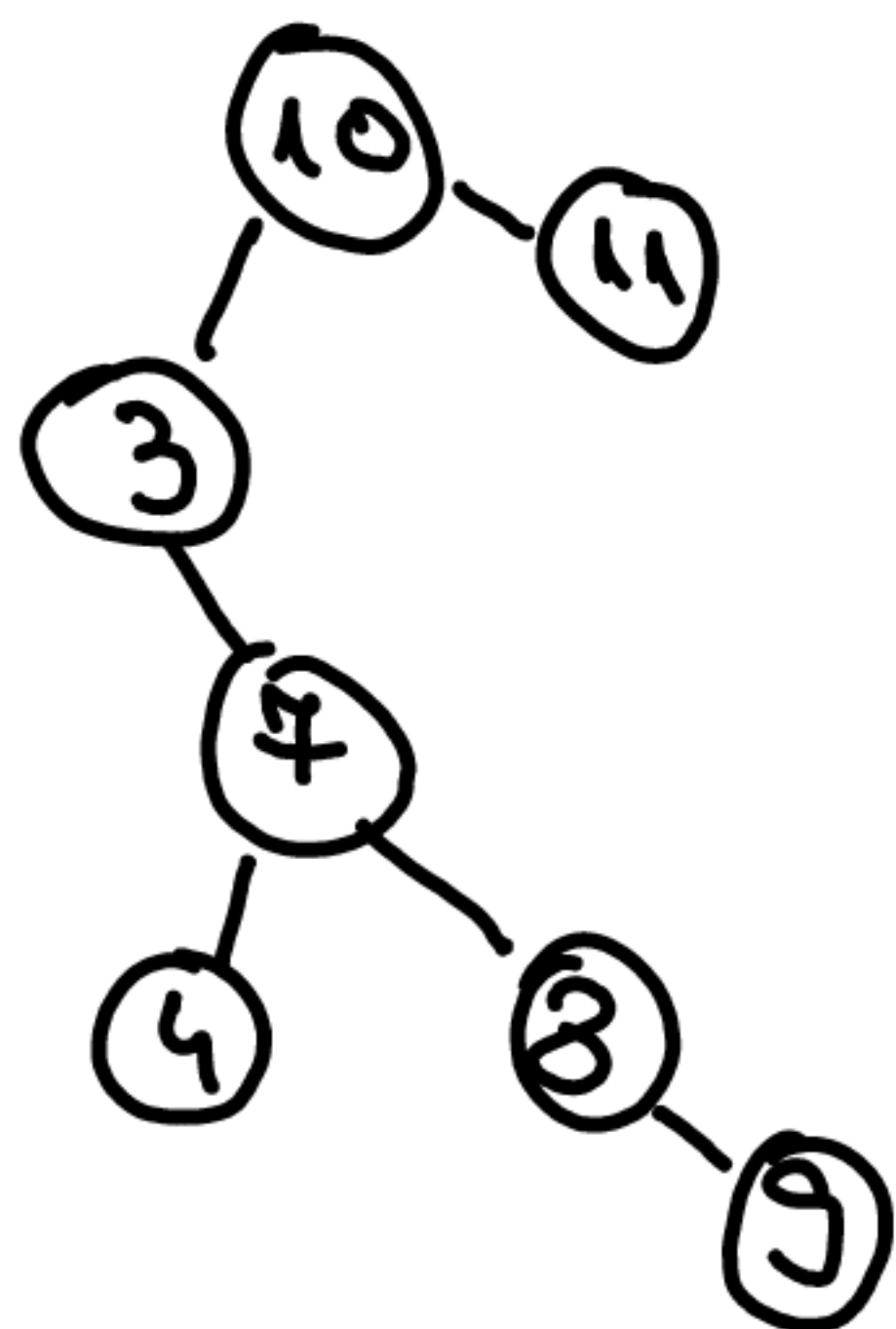
Spre exemplu, în cazul căutării:

Dacă elementul cerut  $<$  decât ce căutăm  $\rightarrow$  mergem în dr. stâng, altfel, putem merge în stâng

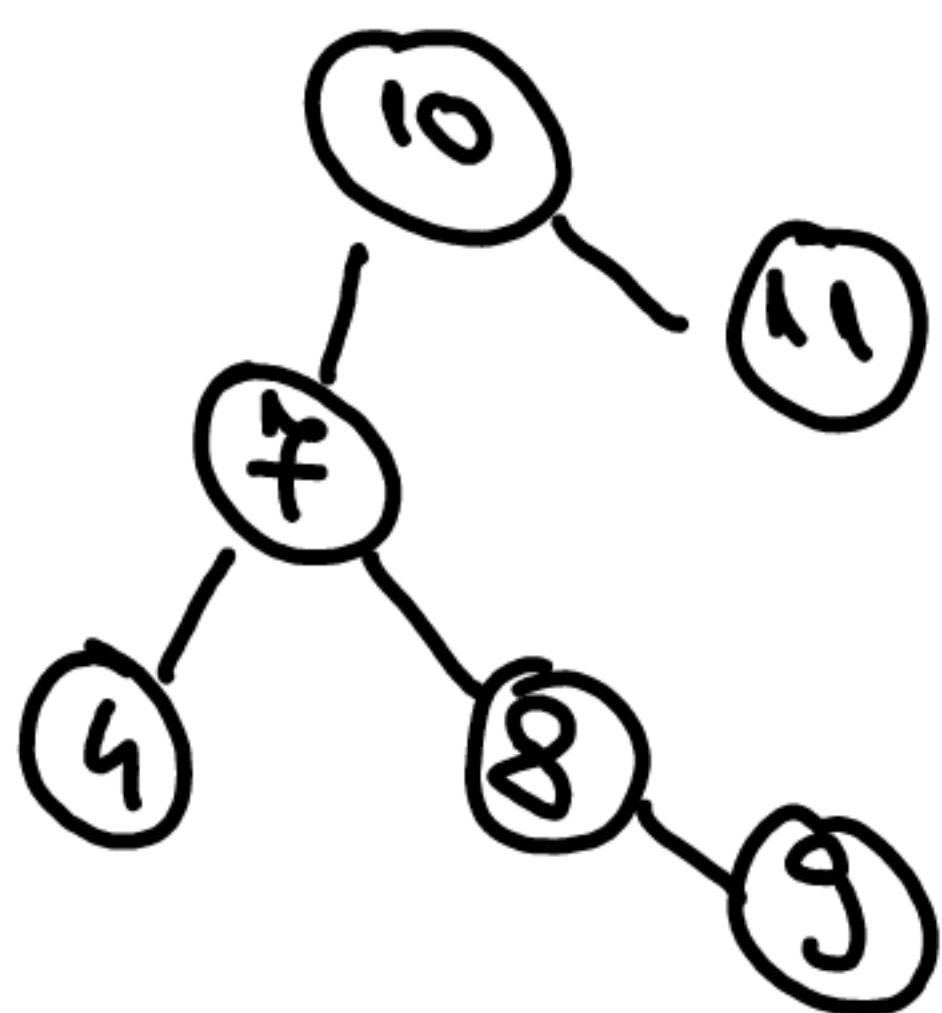
ex. a) adăugare

- deoarece, indiferent de ordinea stăruirii, legăturile se formează în același mod, ajungând la Rădăcină sau la același

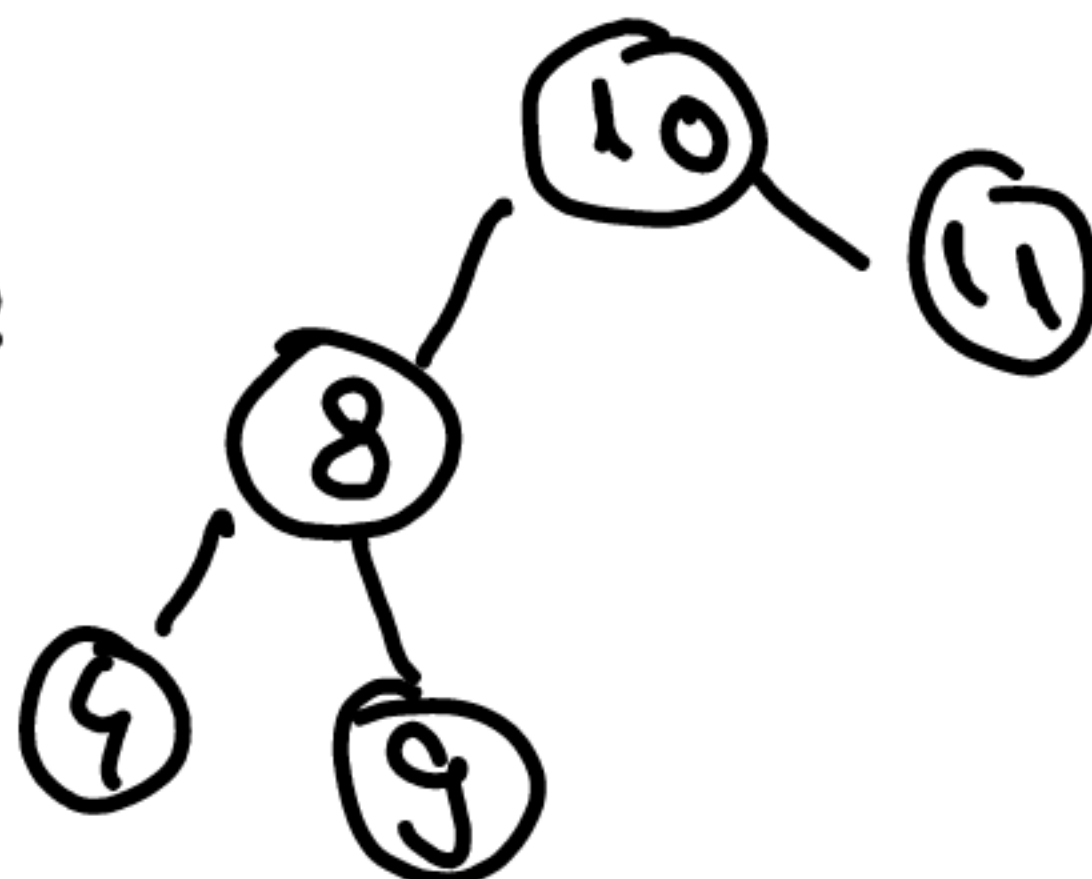
Exemple:



↓ stăruie 3



stăruie 7  
→



- Chiar dacă etapele intermediare sunt diferite, rezultatul final este același



D.

- reprezentare secvențială, pe vector (model  
ansamblu) + recursiv

AzCure

e: TEElement[]

n: Întreg (dimensiunea cozii)

Vom avea și NULL - TEElement pentru pozițiile  
neocupate

Subalgoritm cod(a, e) este:

{ pre: a ∈ AzCure

e ∈ TEElement, e ≠ NULL - TEElement

Post: se returnează codul elementului dacă se  
are în azCure, altfel -1

}

creaza(c) { creaza o coada }

string ← "2"

Dacă a[1] ≠ NULL - TEElement atunci

adauga(c, 1, string)

{ se adauga în coada perechi indice → cod }  
sfîrșit

cît timp 7 via(c) execută

sterge(c, { indice, s })

{ se scoate o pereche de elemente din coada }

{ se verifică dacă am găsit elementul }

Dacă  $a \in \text{indice } 3 = e$  atunci:

cod  $\leftarrow S$  și am găsit elementul  $\rightarrow$  se returnează  
SF Dacă

Dacă  $a \in \text{indice } 2 \neq \text{NULL-Element}$  și  $\text{indice } 2 \leq a.mai.$

$S_1 \leftarrow S$   
@ adăugăm "0" la  $S_1$   
adauga( $c$ , {indice  $2$ ,  $S_1$ })  
și se adaugă descendentul st. în cod  
SF Dacă

Dacă  $a \in \text{indice } 2+1 \neq \text{NULL-Element}$  și  $\text{indice } 2+1 \leq a.mai.$

$S_2 \leftarrow S$   
@ adăugăm "1" la  $S_2$   
adauga( $c$ , {indice  $2+1$ ,  $S_2$ })  
SF Dacă și se adaugă descendentul dr. în cod  
SF Căutăm

SF Căutăm

și dacă s-a ajuns aici, nu s-a găsit elementul în arbori

cod  $\leftarrow -1$

SF Suu algoritmul

Complexitatea: având în vedere că op. de adăugare și  
ștergere ale codului au  $\Theta(1)$

- de timp:  $O(n)$  unde  $n$  reprezintă dimensiunea  
arborului

- de spațiu:  $O(n)$  (spațiu suplimentar)

