A.

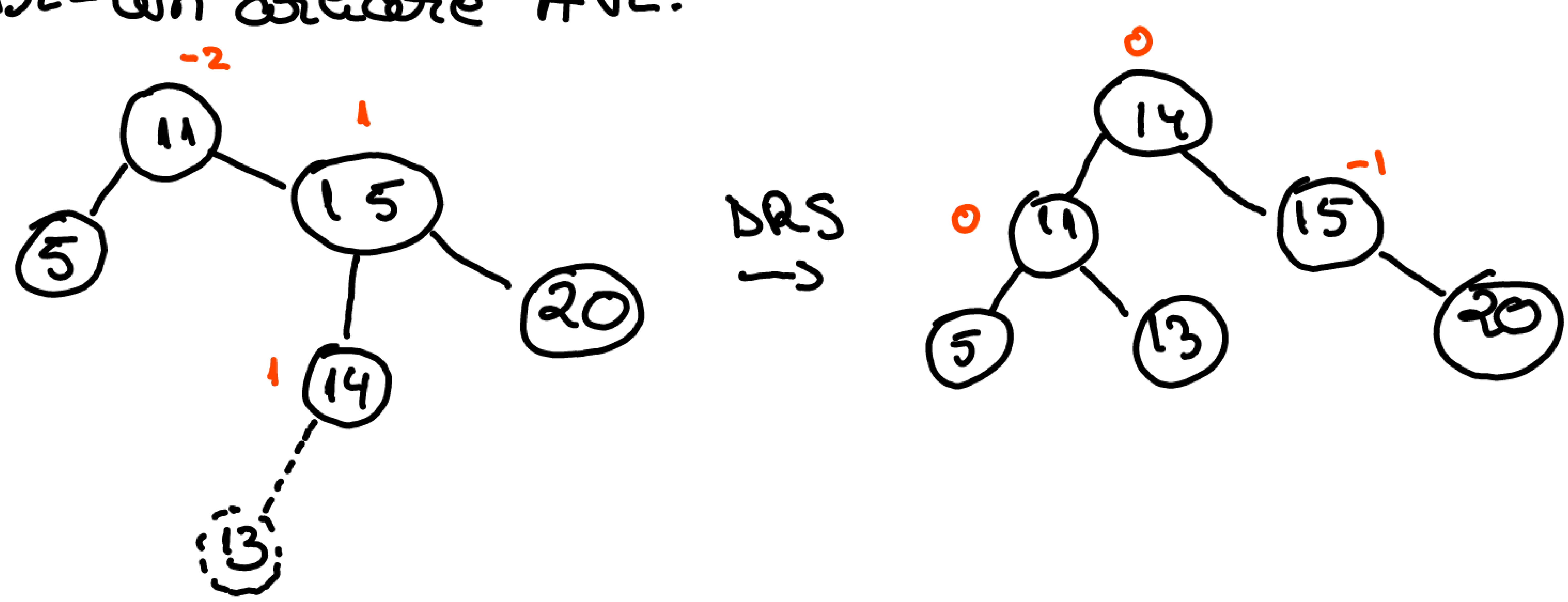
f -> $\Theta(n)$

Subalgoritmul prelucrare nu are caz favorabil / defavorabil => timpii mediu și defavorabil vor fi egali

$T(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} 1 = \sum_{i=1}^{n} n = n^2$ => complexitate de timp : $\Theta(n^2)$

↓ în funcția f, i-ul va lua întotdeauna valori între 1 și n

B. Să se ilustreze op. de dublă rot. spre stânga într-un arbore AVL.



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 => dublă rotație spre st

=> 14 devine rădăcină (deoarece am făcut o singură rotație dreapta)

- părintele lui 14 este mai mare => sub. drept îl are pe 15 rădăcină pe 15
- rădăcina inițială 11 este mai mică decât 14 => 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 -> arbore echilibrat => înălțimea = $\log_2 n$, unde n = nr. de noduri ale arborelui. "adâncimea

Răspuns, răspunsurile sunt: a), b), c), d)

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

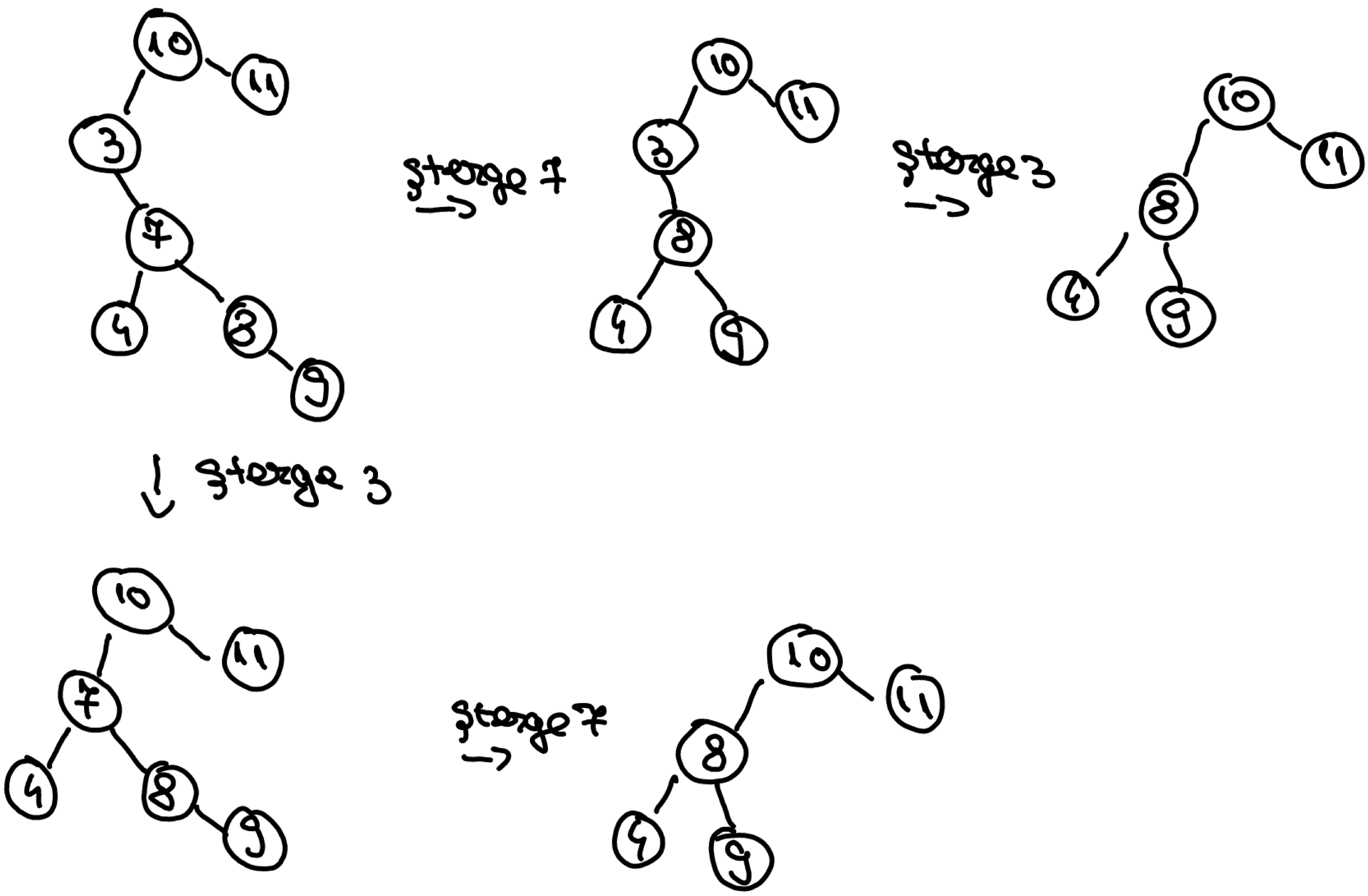
Spre exemplu, în cazul căutării:

Dacă elementul curent < decât ce căutăm -> mergem în dr.
altfel, putem merge în stânga

c2. a) adevărat

- deoarece, indiferent de ordinea ștergerii, legăturile se formează în același mod, ajungând la final să fie aceleași

Exemplu:



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

D.

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

Arbore
e : TElement[]
n : Întreg (dimensiunea arborelui)

vom avea și NULL_TElement pentru pozițiile neocupate

Subalgoritm cod (a, e) este:
{ pre : a ∈ Arbore
e ∈ TElement, e ≠ NULL_TElement
post : se returnează codul elementului dacă se află în arbore, altfel -1
}

creează (c) { creează o coadă }
string ← "r"
Dacă a.e[1] ≠ NULL_TElement atunci
adaugă (c, {1, string})
{ se adaugă în coadă perechi indice -> cod }
sf Dacă

câtTimp ¬vida (c) execută
șterge (c, {indice, s})
{ se scoate o pereche de elemente din coadă }
{ se verifică dacă am găsit elementul }

Daca' a.e[indice] = e atunci

    cod <- S {am gasit elementul -> se returneaza codul}
SRDaca'

Daca' a.e[indice^2] ≠ NULL-TElement ∧ indice*2 ≤ a.nr.
    S1 <- S
    @adaugam "0" la S1
    adauga(e, indice*2, S1)
SRDaca' {se adauga descendentul st. in coada}

Daca' a.e[indice*2+1] ≠ NULL-TElement ∧ indice*2+1 ≤ a.nr.

    S2 <- S
    @adaugam "1" la S2
    adauga(e, indice*2+1, S2)
SRDaca' {se adauga descendentul dr. in coada}

SRcat[imp

{daca s-a gasit elementul, nu s-a gasit elementul in arbore}

cod <- -1

SRSubalgoritm

Complexitatea: → analiza in vedere cu op. de adaugare si stergere a cozii au Θ(1)
- de timp: O(n) unde n reprezinta dimensiunea arborelui
- de spatiu: O(n) (spatiu suplimentar)