A. Pentru subalgoritmul f, timpul mediu și timpul defavorabil au aceeași valoare (neexistând în sine un caz fav. și unul defav.).

Complexitate: $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{j} \dots$ -> algoritmul rulează la infinit (f(n) se va apela întotdeauna în continuu).

B. $d(c) = c \bmod 10$ -> adresare deschisă cu verificare liniară

$d'(c,i) = (d(c)+i) \bmod 10$

Tabela inițială:

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Adăugam 35 -> 5

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	35	-1	-1	-1	-1

Adăugam 2 -> 2

0	1	2	3	4	5	6	7	8	9
-1	-1	2	-1	-1	35	-1	-1	-1	-1

Adăugam 18 -> 8

0	1	2	3	4	5	6	7	8	9
-1	-1	2	-1	-1	35	-1	-1	18	-1

Adăugăm 6 -> 6

0	1	2	3	4	5	6	7	8	9
-1	-1	2	-1	-1	35	6	-1	18	-1

Adăugăm 3 -> 3 și 10 -> 0

0	1	2	3	4	5	6	7	8	9
10	-1	2	3	-1	35	6	-1	18	-1

Adăugăm 8

< 8, 9, 0, 1, 2, 3, 4, 5, 6, 7 >

poziția 8 e ocupată => stocăm pe poz. 9

0	1	2	3	4	5	6	7	8	9
10	-1	2	3	-1	35	6	-1	18	8

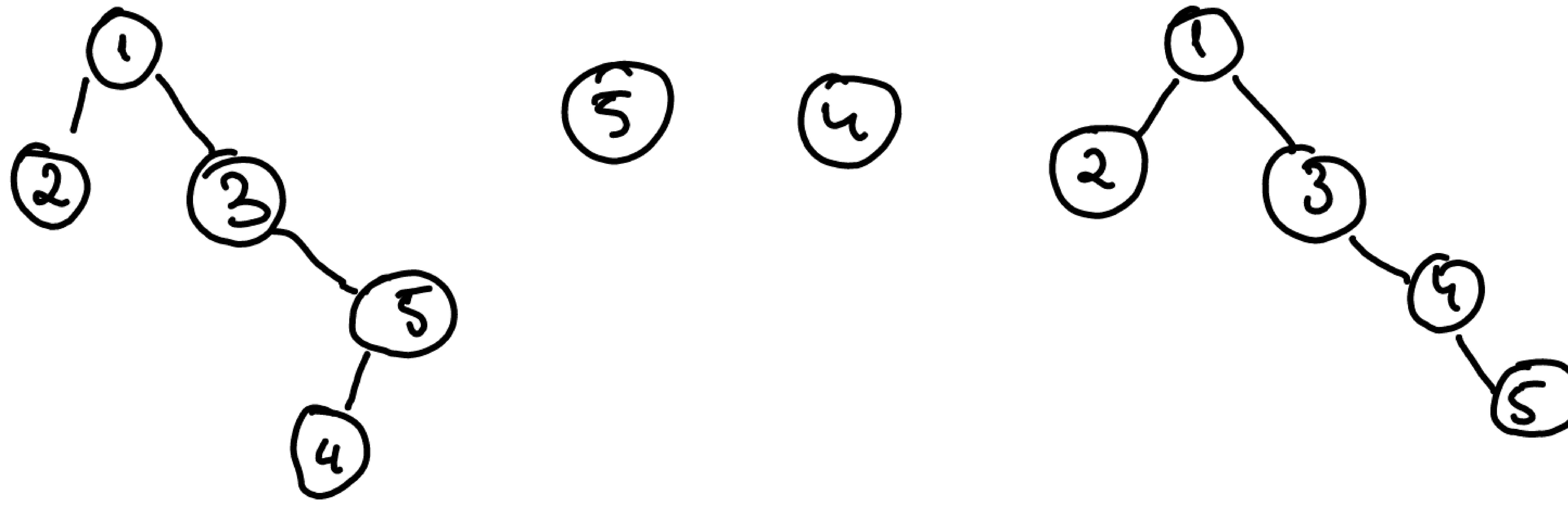
Adăugăm 5

< 5, 6, 7, 8, 9, 0, 1, 2, 3, 4 >

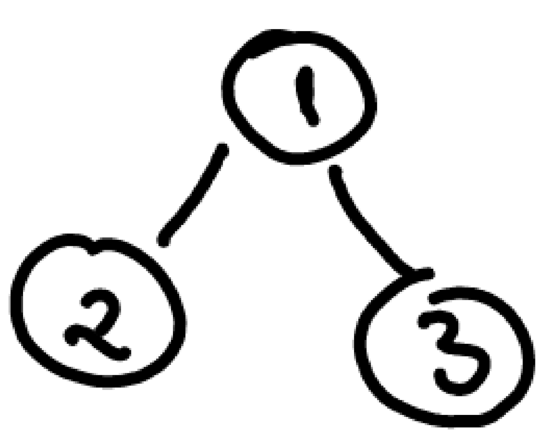
poz. 5 și 6 sunt ocupate => stocăm pe poz. 7

0	1	2	3	4	5	6	7	8	9
10	-1	2	3	-1	35	6	5	18	8

c1.

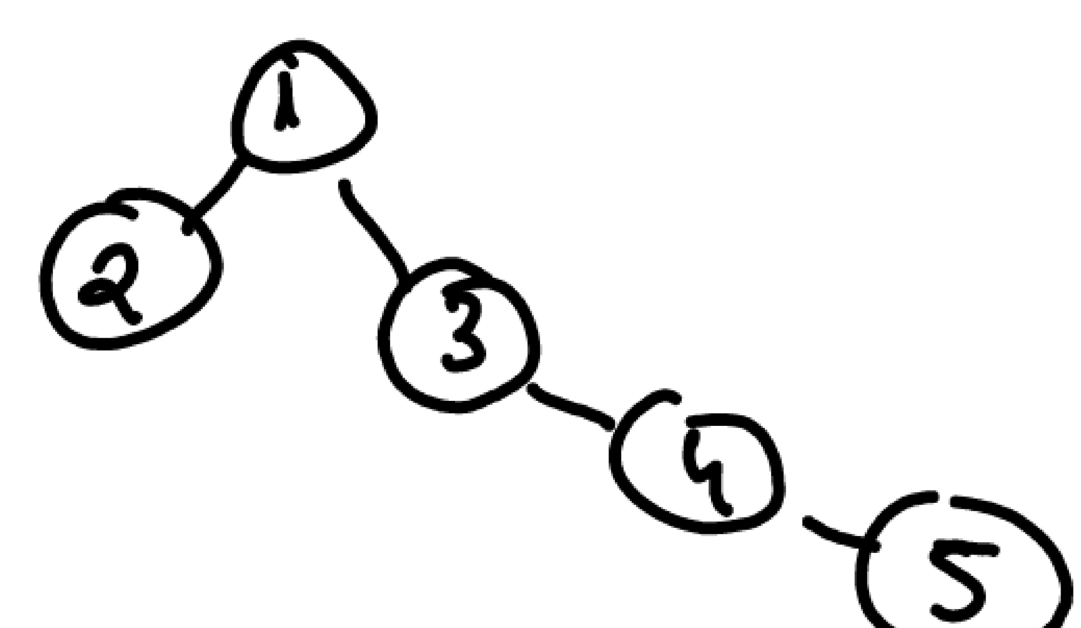


Afirmația este falsă, deoarece ordinea de inserare poate determina: părinți diferiți și poziții diferite. Spre exemplu, în cazul în care avem arborele:

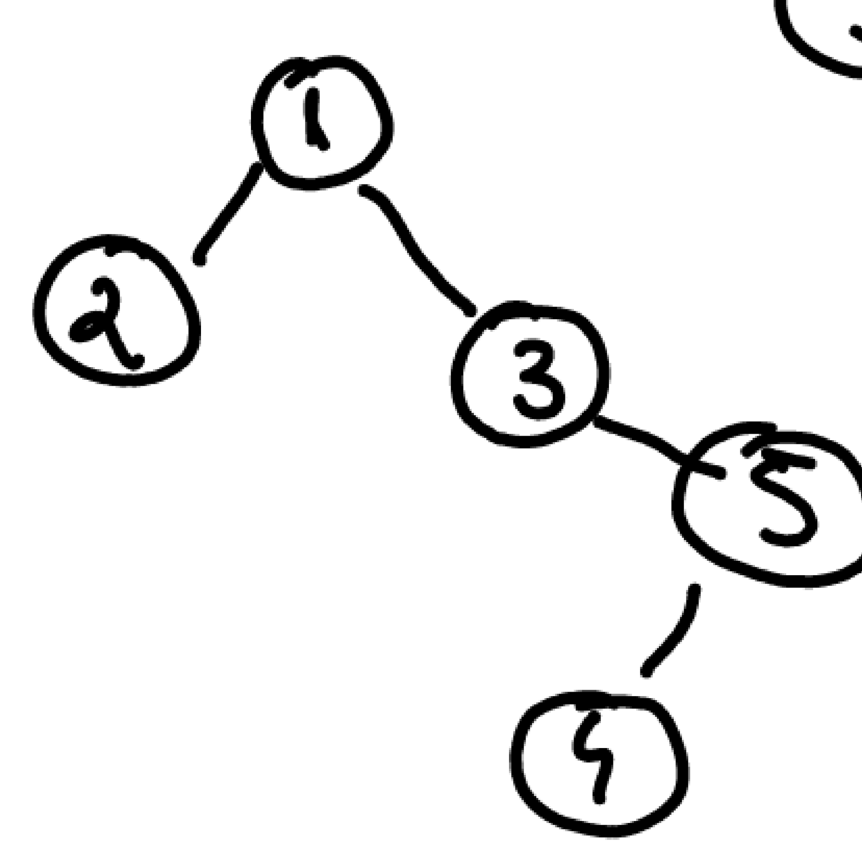


și dorim să inserăm valorile 4 și 5, ordinea în care alegem să fie făcută inserarea va determina cum arată subarborele a cărui rădăcină este 3.

Dacă introducem 4 și apoi 5:



Dar dacă introducem 5 și apoi 4:



c2.

$10 \cdot n^2 < 5 \cdot 2^{n-1} \mid :5$

$2n^2 < 2^{n-1} \mid :2$

$n^2 < 2^{n-2}$

$25 < 8$

$36 < 16$

$49 < 32$

$64 < 64$

$81 < 128$

Cea mai mică valoare a lui n a.î. timpul de ex $10 \cdot n^2$ să fie mai rapid decât un algoritm cu timpul de ex $5 \cdot 2^{n-1}$ este 9, deoarece 9 este cel mai mic nr. care verifică inegalitatea strictă $10 \cdot n^2 < 5 \cdot 2^{n-1}$. Nr. 8 ar fi verificat egalitatea, dar acest lucru nu ar fi însemnat că primul algoritm este mai rapid față de cel de-al doilea.

D. SRD - parcurgere inordine

Nod:

v : TElement

s : ↑Nod

d : ↑Nod

Vector:

r : ↑Nod (rădăcina arborelui)

Subalgoritm find(l, e) este

{ pre: l ∈ Vector

e ∈ TElement

post: se returnează nr. asociat lui e, în urma unei parcurgeri inordine

în cazul în care nu se găsește -> -1 }

count <- 0

ok <- 0

{ se construiește o stivă }

creează (S)

curent ← e.r

câtTimp ¬vida(S) ∨ curent ≠ NIL execută

câtTimp curent ≠ NIL execută

{ adăugăm în stivă subarborele stâng }

adauga(S, curent)

curent ← [curent].s

sfcâtTimp

șterge (S, curent)

count ← count + 1 { am ajuns la un element }

Dacă [curent].e = e atunci

ok ← 1

@ stop { dacă găsim elementul → stop }

sfDacă

curent ← [curent].d

SfcâtTimp

Dacă ok = 0 atunci

curent ← -1 { în cazul în care nu se găsește elem. trebuie ret. -1 }

Rind ← curent

SfSubalgoritm

Complexitate: de timp $\Theta(n)$

de sp. adițional $\Theta(n)$