Structuri de Date
Laboratorul 5: Arbori binari

Mihai Nan

29 martie 2021

1. Introducere

Un arbore combină avantajele oferite de alte două structuri: tablourile și listele înlănțuite. Arborii permit, pe de o parte, executarea unor căutări rapide, la fel ca tablourile ordonate, iar, pe de altă parte, inserarea și ștergerea elementelor sunt rapide, la fel ca la o listă simplu înlănțuită.

Dacă ne-am uitat la un arbore genealogic, sau la o ierarhie de comandă într-o firmă, am observat informațiile aranjate într-un arbore. Un arbore este compus dintr-o colecție de **noduri**, care sunt unite prin **arce**, unde fiecare nod are asociată o anumită informație, și o colecție de copii. Vom reprezenta nodurile prin cercuri, iar arcele, prin linii care unesc cercurile. **Copiii** unui nod sunt acele noduri care urmează imediat sub nodul respectiv. **Părintele** unui nod este acel nod care se află imediat deasupra. **Rădăcina** unui arbore este acel nod unic, care nu are niciun părinte.

Nodurile reprezintă, de regulă, entități din lumea reală, cum ar fi valori numerice, persoane, părți ale unei mașini, rezervări de bilete de avion. Nodurile sunt elemente obișnuite pe care le putem memora în orice altă structură de date. Arcele dintre noduri reprezintă modul în care nodurile sunt conectate. Este ușor și rapid să ajungem de la un nod la altul dacă acestea sunt conectate printr- un arc. Arcele sunt reprezentate în C prin **pointeri**.

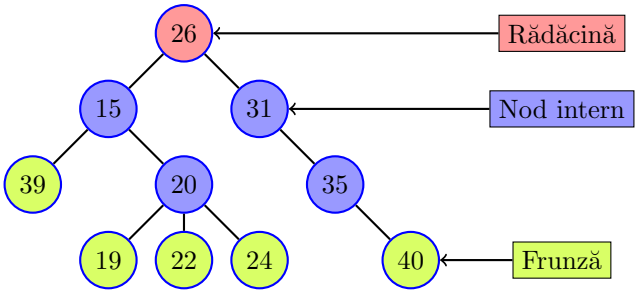


Figura 1: Un exemplu de arbore

Toți arborii au următoarele proprietăți:

- Există o singură rădăcină.

- Toate nodurile, cu excepția rădăcinii, au exact un părinte.
- Nu exista cicluri. Cu alte cuvinte, pornind de la un anumit nod, nu există un anumit traseu pe care îl putem parcurge astfel încât să ajungem înapoi la nodul de plecare.

Înălțimea unui arbore este, de fapt, valoarea maximă de pe nivelurile nodurilor terminale. Numărul de descendenți direcți ai unui nod reprezintă **ordinul** sau **gradul nodului**. **Ordinul** sau **gradul arborelui** este valoarea maximă luată de gradul unui nod component al arborelui.

2. Arborele binar

Arborii binari sunt un tip aparte de arbori, în care fiecare nod are maxim 2 copii. Pentru un nod dat, într-un arbore binar, vom avea fiul din stânga și fiul din dreapta. Deci, un nod dintr-un arbore binar poate avea 2 copii (stânga și dreapta), un singur copil (doar stânga sau doar dreapta) sau niciun copil. Nodurile care nu au niciun copil se numesc **noduri frunză**, iar cele care au 1 sau 2 copii se numesc **noduri interne**.

Între numărul **N** de noduri al unui arbore binar și înălțimea sa **H** există relațiile:

$$H \leq N \leq 2^H - 1 \tag{1}$$

$$\log_2 N \leq H \leq N \tag{2}$$

2.1. Arborele binar de căutare

Într-un **arbore binar de căutare**, pentru fiecare nod cheia acestuia are o valoare mai mare decât cheile tuturor nodurilor din sub-arborele stâng și mai mică decât cheile nodurilor ce compun sub-arborele drept. Arborii binari de căutare permit menținerea datelor în ordine și o căutare rapidă a unei valori, ceea ce îi recomandă pentru implementarea de mulțimi și dicționare ordonate.

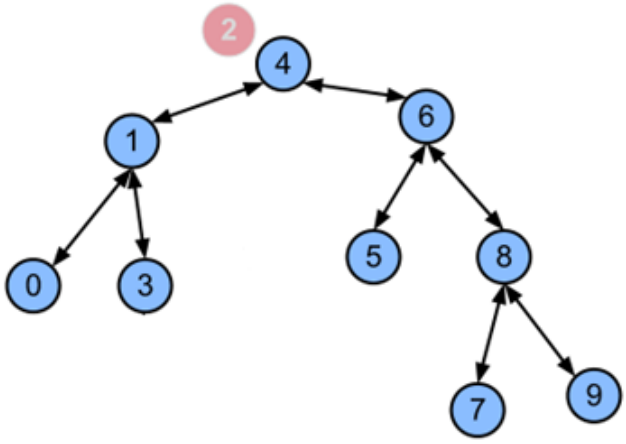
Observație

O importantă proprietate a arborilor de căutare este aceea că **parcurgerea în inordine** produce o secvență ordonată crescător a cheilor din nodurile arborelui.
Valoarea maximă dintr-un arbore binar de căutare se află în nodul din **extremitatea dreapta** și se determina prin coborârea pe sub-arborele drept, iar **valoarea minimă** se află în nodul din **extremitatea stânga**.

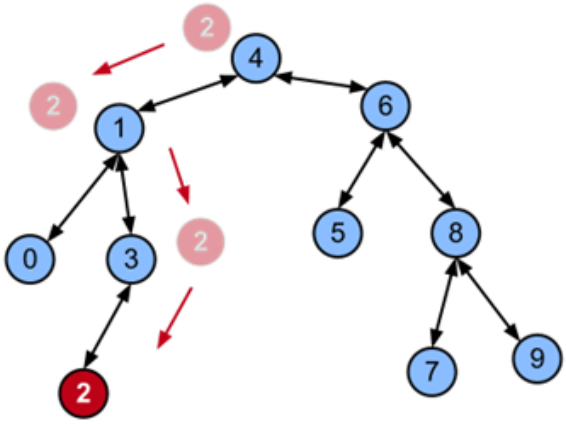
Pentru reținerea informației, vom folosi alocarea dinamică. Pentru fiecare nod în parte este necesar să se rețină, pe lângă informația utila, și legăturile cu nodurile copii (adresele acestora), iar pentru aceasta uzităm pointeri. În definirea de mai jos, ***left*** reprezintă adresa nodului copil din stânga, ***value*** reprezinta câmpul cu informație utilă, iar ***right*** este legătura cu copilul din dreapta.

```c
typedef struct node {
    Item value;
    struct node *left;
    struct node *right;
} TreeNode, *Tree;
```

2.1.1. Inserarea unui nod



Să considerăm arborele de mai sus și să presupunem că dorim să inserăm nodul cu valoarea 2. Acesta se va insera ca nod frunză. Pentru a-l insera, va trebui să căutăm o poziție în arbore care respectă regula de integritate a arborilor binari de căutare.



Vom începe prin compararea nodului de inserat (2) cu rădăcina arborelui dat (4). Observăm că este mai mic decât ea, deci va trebui inserat undeva în sub-arborele stâng al acesteia. Vom compara apoi 2 cu 1. Din moment ce 2 este mai mare decât 1, nodul cu valoarea 2 va trebui plasat undeva în sub-arborele drept al lui 1. Se compară apoi 2 cu 3. Deoarece 3 este mai mare decât 2, nodul cu valoarea 2 trebuie să se afle în sub-arborele din stânga al nodului 3. Dar nodul 3 nu are niciun copil în partea stângă. Asta înseamnă că am găsit locația pentru nodul 2. Tot ceea ce mai trebuie făcut este să modificăm în nodul 3 adresa către fiul sau stâng astfel încât să indice spre 2. (Vezi fig. de mai sus!)

Algorithm 1 Insert

```
1: procedure INSERT(root, value)
2:     if root = NULL then
3:         initTree(root, value)
4:         return root
5:     else if root → value = value then
6:         print(Nodul exista)
7:         return root
8:     else
9:         if root → value > value then
10:            if root → left = NULL then
11:                initTree(root → left, value)
12:                return root
13:            else
14:                root → left ← insert(root → left, value)
15:                return root
16:        else
17:            if root → right = NULL then
18:                initTree(root → right, value)
19:                return root
20:            else
21:                root → right ← insert(root → right, value)
22:                return root
```

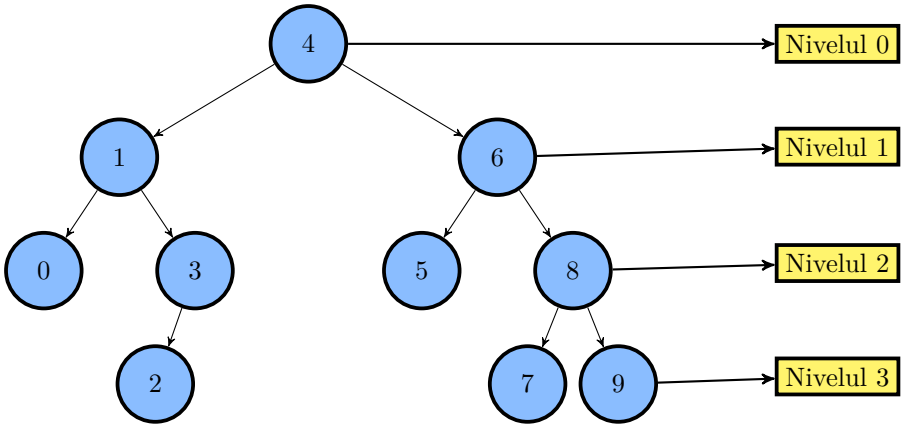
2.1.2. Înălțimea arborelui

Înălțimea unui arbore reprezintă nivelul maxim din arbore (adâncimea frunzei de nivel maxim).

Înălțimea unui arbore binar cu N noduri interne este minim $\log_2 N$ şi maxim $N - 1$.

Înălțimea unui arbore binar complet cu M noduri este cel mult $O(\log M)$.

Pentru a determina cum se calculează înălțimea unui arbore binar, pornim de la următorul exemplu:

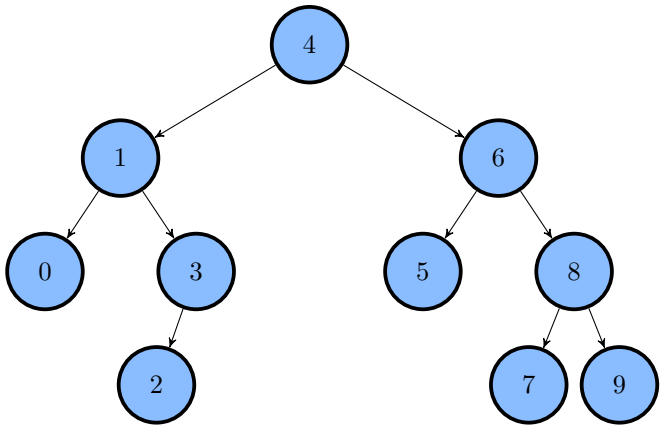


Algorithm 2 Height

```
1: procedure HEIGHT(root)
2:     if root = NULL then
3:         return 0
4:     else
5:         tmp ← root
6:         left ← height(tmp → left)
7:         right ← height(tmp → right)
8:         if left ≥ right then
9:             left ← left + 1
10:            return left
11:        else
12:            right ← right + 1
13:            return right
```

2.1.3. Parcurgerea arborelui

Există 3 tipuri de parcurgere a unui arbore: **inordine**, **preordine** și **postordine**. Aceste denumiri corespund modului cum este vizitată rădăcina.

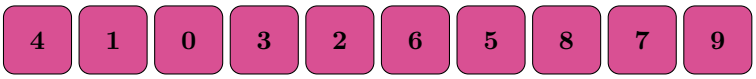


Parcurgerea în inordine

Algorithm 3 Inordine

```
1: procedure INORDINE(root)
2:     if root ≠ NULL then
3:         inordine(root → left)
4:         print(root → value)
5:         inordine(root → right)
```

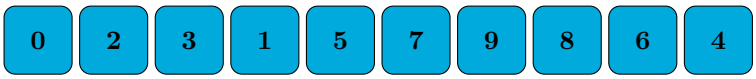
Parcurgerea în preordine



Algorithm 4 Preordine

```
1: procedure PREORDINE(root)
2:     if root ≠ NULL then
3:         print(root → value)
4:         preordine(root → left)
5:         preordine(root → right)
```

Parcurgerea în postordine



Algorithm 5 Postordine

```
1: procedure POSTORDINE(root)
2:     if root ≠ NULL then
3:         postordine(root → left)
4:         postordine(root → right)
5:         print(root → value)
```

2. Cerințe

În acest laborator dispuneți de mai multe fișiere, inclusiv scheletul de cod, după cum urmează:

- `tree.h` – fișierul în care este definită structura de date cu care vom lucra și antetele funcțiilor care trebuie implementate.
- `tree.c` – fișierul în care se vor implementa funcțiile ce rezolvă cerințele laboratorului.
- `main.c` – checker pentru validarea implementării cerințelor laboratorului.
- `Makefile` – fișierul pe baza căruia se vor compila și rula testele.

Pentru compilarea tuturor aplicațiilor, folosiți comanda "`make build`". Aceasta are următorul output pentru un program fără erori de sintaxă sau warning-uri:

```
$ make build
gcc -c tree.c -g
gcc main.c tree.o -o testTree -g -Wall
```

Iar pentru ștergerea automată a fișierelor generate prin compilare folosiți comanda "`make clean`":

```
$ make clean
rm -f *.o *~ testTree *.dot *.png
```

Pentru testarea completă (inclusiv memory leaks) puteți folosi "`make test`".

Cerința 1 (6 puncte) Implementați următoarele funcții de lucru cu arbori binari în fișierul "`tree.c`".

a) `void init(Tree *root, Item value)` – Funcție care inițializează un nod de arbore: îi alocă memorie, îi setează câmpul valoare, setează left și right să pointeze către `NULL`.

b) `Tree insert(Tree root, Item value)` – Funcție care inserează o valoare într-un arbore binar, respectând proprietățile unui arbore binar de căutare. Funcția va returna arborele nou creat.

> **Atenție**
>
> Funcția poate primi ca parametru arborele vid (`NULL`) și va trebui să returneze un arbore format dintr-un singur nod.

c) `void printPostorder(Tree root)` – Funcție care afișează nodurile folosind parcurgerea în postordine.

d) `void printPreorder(Tree root)` – Funcție care afișează nodurile folosind parcurgerea în preordine.

e) `void printInorder(Tree root)` – Funcție care afișează nodurile folosind parcurgerea în inordine.

Observație

Testarea acestor funcții se va face manual.

f) `void freeTree(Tree *root)` – Funcție care dealocă întreaga memorie alocată pentru un arbore binar. `root` va pointa către `NULL` după ce se va apela funcția.

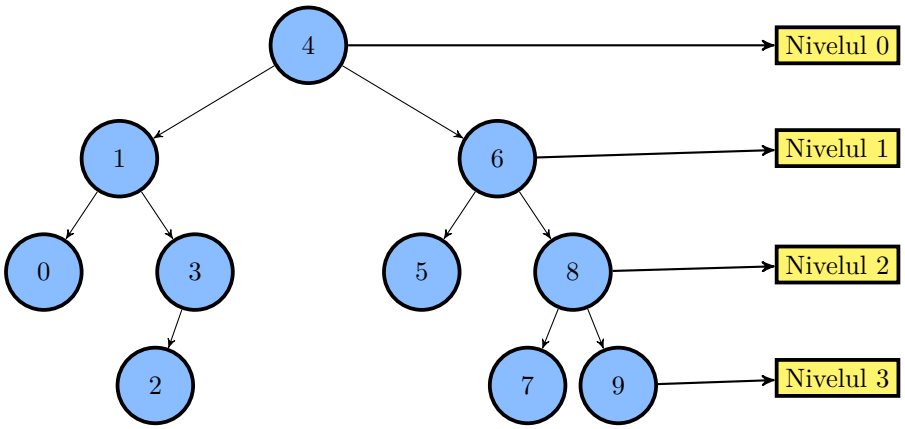
g) `int size(Tree root)` – Funcție care determină numărul de noduri dintr-un arbore binar.

h) `int maxDepth(Tree root)` – Funcție care returnează adâncimea maximă a arborelui.

Atenție

Nivelul sau **adâncimea** unui nod reprezintă numărul de arce de la rădăcină la nod (rădăcina are nivel 0).

Exemplu: Adâncimea maximă a arborelui de mai jos este 3.

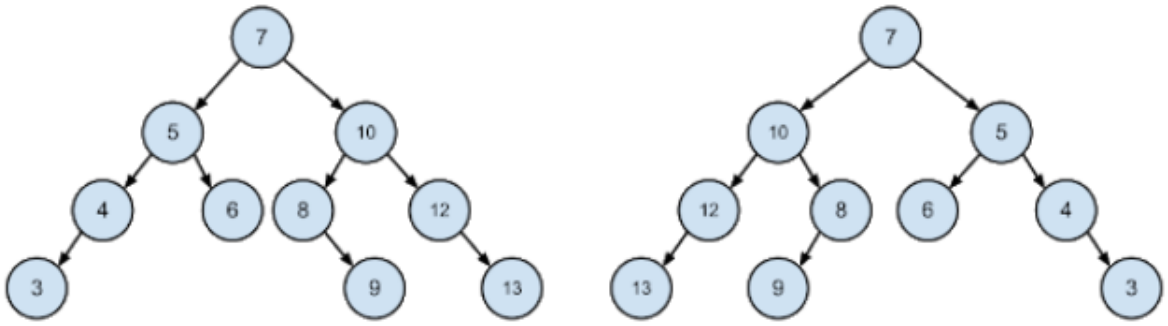


Cerința 2 (2 puncte) Implementați, în fișierul `tree.c`, o funcție care construiește oglinditul unui arbore.

`void mirror(Tree root)`

Observație

Funcția va aplica modificările direct pe arborele primit ca parametru prin pointer la nodul rădăcină.



Cerința 3 (2 puncte) Implementați, în fișierul `tree.c`, o funcție care verifică dacă doi arbori binari sunt identici.

`int sameTree(Tree root1, Tree root2)` – funcția returnează 1 dacă arborii sunt identici și 0 altfel.

3. Informații utile

Pentru toate cerințele, putem testa și individual funcțiile implementate. Pentru acest lucru, va trebui să rulăm executabilul `testTree` utilizând argumente în linia de comandă:

- 1 – pentru a testa `insert` și `init` (1.5 puncte);
- 2 – pentru a testa `printPostorder` (0.75 puncte);
- 3 – pentru a testa `printPreorder` (0.75 puncte);
- 4 – pentru a testa `printInorder` (0.75 puncte);
- 5 – pentru a testa `freeTree` (0.75 puncte);
- 6 – pentru a testa `size` (0.75 puncte);
- 7 – pentru a testa `maxDepth` (0.75 puncte);
- 8 – pentru a testa `mirror` (2 puncte);
- 9 – pentru a testa `sameTree` (2 puncte).

Observație: Punctajul pentru `freeTree` se va acorda manual dacă nu există erori sau pierderi de meorie atunci când rulăm cu `valgrind`!

Exemple de testări:

- Testarea funcției `insert`

```
$ ./testSortedList 1
insert  - Toate testele au trecut!
Toate testele selectate au trecut!
Punctajul obtinut este: 1.50
Teste rulate: 1
```

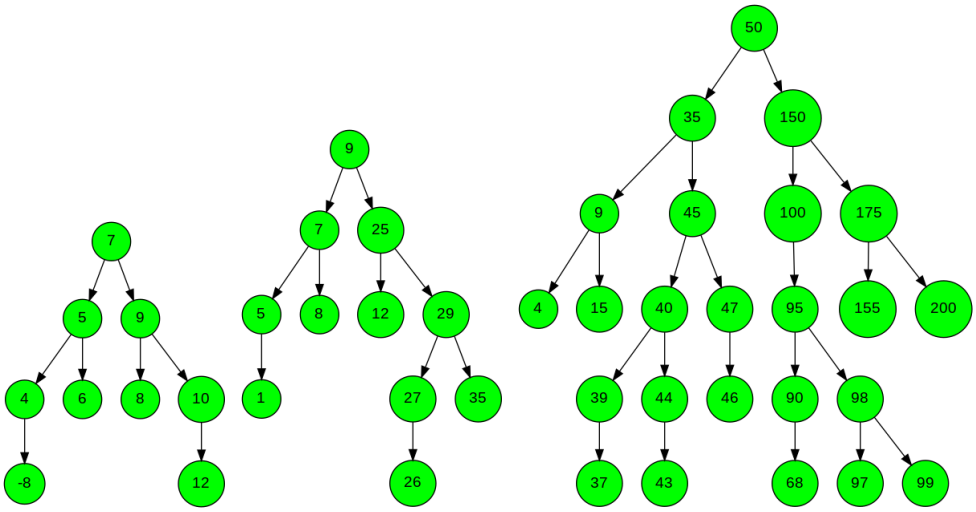
Pentru a fi mai ușor de verificat dacă este corectă implementarea, este implementată o funcție care desenează arborele. Această funcție utilizează utilitarul `graphViz`.

Pentru a putea instala acest utilitar pe Linux, putem folosi comanda:

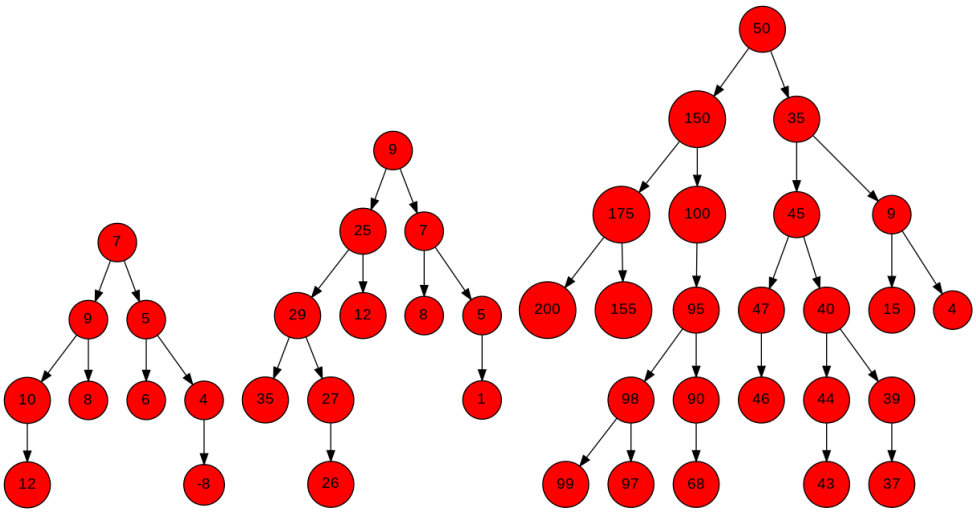
```
sudo apt-get install graphviz
```

Pentru a putea instala acest utilitar pe Windows, putem descărcă executabilul de la adresa: https://graphviz.org/download/.

Exemple de imagini generate:



Arborii rezultați în urma aplicării operației de oglindire sunt următorii:



- Testarea tuturor funcțiilor folosind valgrind:

```
$ make test
valgrind ./testTree
==15577== Memcheck, a memory error detector
==15577== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==15577== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==15577== Command: ./testTree
==15577==
insert      - Toate testele au trecut!
postorder   - Testele sunt verificate manual!
        postorder         - Testul 1: [-8 4 6 5 8 12 10 9 7]
                -8 4 6 5 8 12 10 9 7
        postorder         - Testul 2: [1 5 8 7 12 26 27 35 29 25 9]
                1 5 8 7 12 26 27 35 29 25 9
        postorder         - Testul 3: [4 15 9 37 39 43 44 40 46 47 45 35 68 90 97 99 98 95 100 155 200 175 150 50]
                4 15 9 37 39 43 44 40 46 47 45 35 68 90 97 99 98 95 100 155 200 175 150 50
preorder    - Testele sunt verificate manual!
        preorder         - Testul 1: [7 5 4 -8 6 9 8 10 12]
                7 5 4 -8 6 9 8 10 12
        preorder         - Testul 2: [9 7 5 1 8 25 12 29 27 26 35]
                9 7 5 1 8 25 12 29 27 26 35
        preorder         - Testul 3: [50 35 9 4 15 45 40 39 37 44 43 47 46 150 100 95 90 68 98 97 99 175 155 200]
                50 35 9 4 15 45 40 39 37 44 43 47 46 150 100 95 90 68 98 97 99 175 155 200
inorder     - Testele sunt verificate manual!
        inorder         - Testul 1: [-8 4 5 6 7 8 9 10 12]
                -8 4 5 6 7 8 9 10 12
        inorder         - Testul 2: [1 5 7 8 9 12 25 26 27 29 35]
                1 5 7 8 9 12 25 26 27 29 35
        inorder         - Testul 3: [4 9 15 35 37 39 40 43 44 45 46 47 50 68 90 95 97 98 99 100 150 155 175 200]
                4 9 15 35 37 39 40 43 44 45 46 47 50 68 90 95 97 98 99 100 150 155 175 200
free        - Testele sunt verificate manual!
size        - Toate testele au trecut!
maxDepth    - Toate testele au trecut!
mirror      - Toate testele au trecut!
sameTree    - Toate testele au trecut!
Toate testele au trecut! Felicitari!
Punctajul obtinut este: 10.00
Teste rulate: 9
==15577==
==15577== HEAP SUMMARY:
==15577==     in use at exit: 0 bytes in 0 blocks
==15577==   total heap usage: 497 allocs, 497 frees, 40,528 bytes allocated
==15577==
==15577== All heap blocks were freed -- no leaks are possible
==15577==
==15577== For counts of detected and suppressed errors, rerun with: -v
==15577== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```