INFORMATICĂ · CLASA A XI-A · STRUCTURI DE DATE (INTENSIV) · LECȚIA 20

Arbori binari și arbori binari de căutare

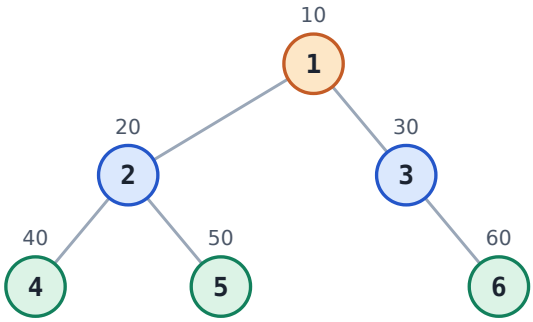
Durată: 60 min Nivel: intensiv informatică Înainte: lecțiile xi-13, xi-18

CE VEI ȘTI DUPĂ LECȚIE

- Reprezinți un arbore binar cu vectori (st, dr) sau alocat dinamic.
- Implementezi parcurgerile preordine, inordine, postordine și pe niveluri.
- Calculezi înălțimea, numărul de frunze, suma valorilor.
- Inserezi și cauți într-un arbore binar de căutare (BST).

1. Ce este un arbore binar

La clasa a X-a/XI-a ai văzut arbori cu rădăcină oarecare (lecția XI.13). Un **arbore binar** este un arbore cu rădăcină în care fiecare nod are **cel mult doi fii**: unul **stâng** și unul **drept** (pot lipsi). Ordinea contează: „doar fiu stâng" este alt arbore decât „doar fiu drept".



portocaliu: rădăcina · albastru: noduri interne · verde: frunze (fără fii) · fiecare nod are cel mult 2 fii

Arbore binar

Vocabular: **rădăcina** (nodul din vârf), **frunze** (fără fii), **nod intern** (are cel puțin un fiu), **subarbore stâng/drept** (arborele care pornește din fiul respectiv), **înălțime** (numărul de muchii pe cel mai lung drum de la rădăcină la o frunză; aici 2), **nivelul** unui nod (distanța până la rădăcină; rădăcina e pe nivelul 0).

Fapt util (la grile)	Formulă
noduri maxime într-un arbore binar de înălțime h	2^(h+1) − 1 (arborele „plin")
frunze maxime la înălțimea h	2^h
arbore în care fiecare nod are 0 sau 2 fii	frunze = noduri interne + 1
înălțimea minimă pentru n noduri	$\lfloor \log_2 n \rfloor$

> **DE ȘTIUT**
>
> În unele probleme pbinfo înălțimea se măsoară în **noduri** (arborele cu un singur nod are înălțimea 1). Citește definiția din enunț: diferența este 1 și strică răspunsul.

2. Reprezentarea: trei vectori

Pentru fiecare nod i: `v[i]` = informația, `st[i]` = fiul stâng, `dr[i]` = fiul drept; `0` = nu există.

	1	2	3	4	5	6
v[i] valoare	10	20	30	40	50	60
st[i] fiu stâng	2	4	0	0	0	0
dr[i] fiu drept	3	5	6	0	0	0

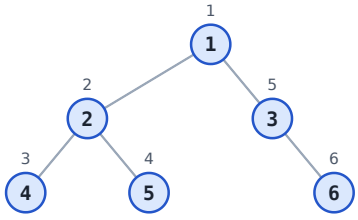
rădăcina este nodul 1; frunzele (4, 5, 6) au st = dr = 0

Vectori

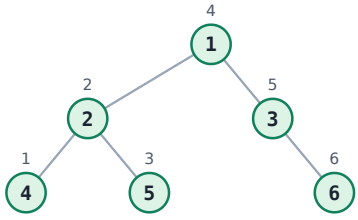
3. Parcurgeri: RSD, SRD, SDR

Cum vizitezi toate nodurile? Cu recursivitate: **R** = rădăcina, **S** = subarborele stâng, **D** = subarborele drept.

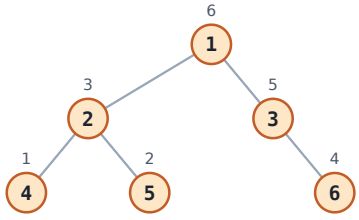
Parcurgere	Ordinea	Exemplu
preordine	R, S, D	10 20 40 50 30 60
inordine	S, R, D	40 20 50 10 30 60
postordine	S, D, R	40 50 20 60 30 10



preordine R S D: 10 20 40 50 30 60



inordine S R D: 40 20 50 10 30 60



postordine S D R: 40 50 20 60 30 10

Parcurgeri

Urmărirea apelurilor pentru **preordine** pe arborele de sus (apelurile cu 0 nu fac nimic, nu le mai scriem):

Apel	Ce afișează	Apoi apelează
pre(1)	10	pre(2), pre(3)

Apel	Ce afișează	Apoi apelează
pre(2)	20	pre(4), pre(5)
pre(4)	40	(fii 0)
pre(5)	50	(fii 0)
pre(3)	30	pre(0), pre(6)
pre(6)	60	(fii 0)

```cpp
#include <iostream>
using namespace std;

int n, v[101], st[101], dr[101];

void preordine(int x)
{
    if (x == 0) return;                 // arbore vid: oprire
    cout << v[x] << ' ';                // R
    preordine(st[x]);                   // S
    preordine(dr[x]);                   // D
}
void inordine(int x)
{
    if (x == 0) return;
    inordine(st[x]);
    cout << v[x] << ' ';
    inordine(dr[x]);
}
void postordine(int x)
{
    if (x == 0) return;
    postordine(st[x]);
    postordine(dr[x]);
    cout << v[x] << ' ';
}
int inaltime(int x)
{
    if (x == 0) return -1;              // arbore vid: -1, ca frunza să aibă 0
    return 1 + max(inaltime(st[x]), inaltime(dr[x]));
}
int frunze(int x)
{
    if (x == 0) return 0;
    if (st[x] == 0 && dr[x] == 0) return 1;
    return frunze(st[x]) + frunze(dr[x]);
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> v[i] >> st[i] >> dr[i];
    preordine(1);  cout << '\n';
    inordine(1);   cout << '\n';
    postordine(1); cout << '\n';
    cout << inaltime(1) << ' ' << frunze(1);
    return 0;
}
```

DE CE AȘA?

De ce se termină recursivitatea? Fiecare apel coboară un nivel; la `x == 0` (nu există nod) nu mai apelezi nimic: este cazul de bază. **De ce `inaltime(0) = -1`?** Ca o frunză (ale cărei două apeluri dau -1) să aibă înălțimea 1 + max(−1, −1) = 0. **Ce spun cele trei nume?** Numele arată **când** afișezi rădăcina: înainte (pre), între (in) sau după (post) cei doi subarbori.

4. Reprezentarea dinamică (cu pointeri)

La fel ca la liste (lecția XI.18), dar nodul are **doi** pointeri:

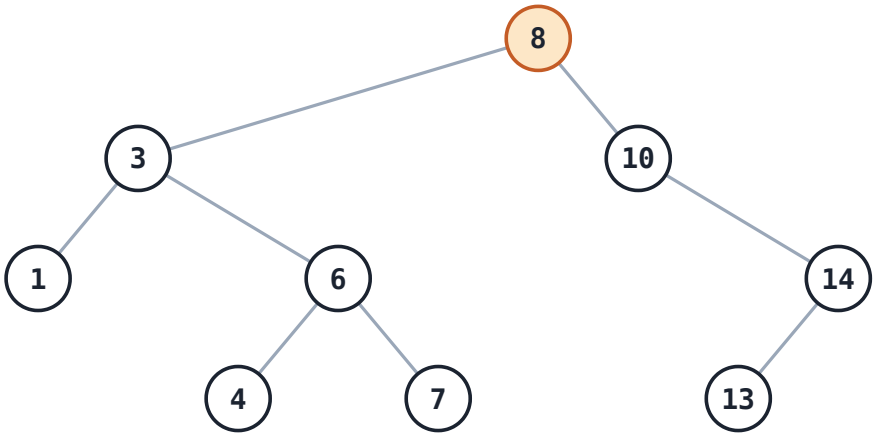
```
struct nod { int info; nod *st, *dr; };
```

Parcurgerea inordine devine:

```
void inordine(nod *r)
{
    if (r == NULL) return;
    inordine(r->st);
    cout << r->info << ' ';
    inordine(r->dr);
}
```

5. Arbore binar de căutare (BST)

Un **BST** respectă regula: pentru **fiecare** nod, toate valorile din subarborele stâng sunt **mai mici**, iar cele din subarborele drept **mai mari**.

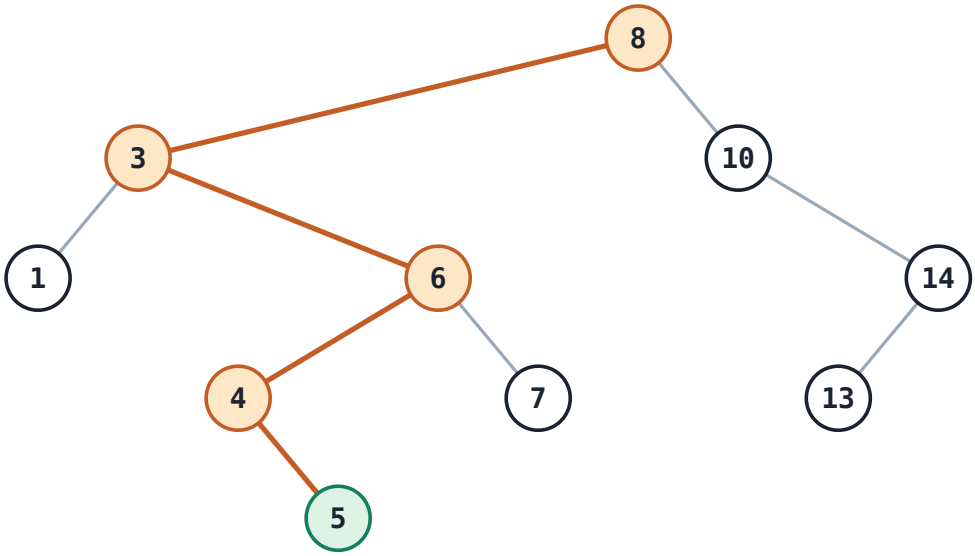


stânga < nod < dreapta, pentru FIECARE nod · inserare în ordinea 8, 3, 10, 1, 6, 14, 4, 7, 13

BST

Inserarea

Cobori de la rădăcină: dacă x e mai mic, mergi stânga, altfel dreapta, până găsești un loc liber; acolo pui nodul nou.

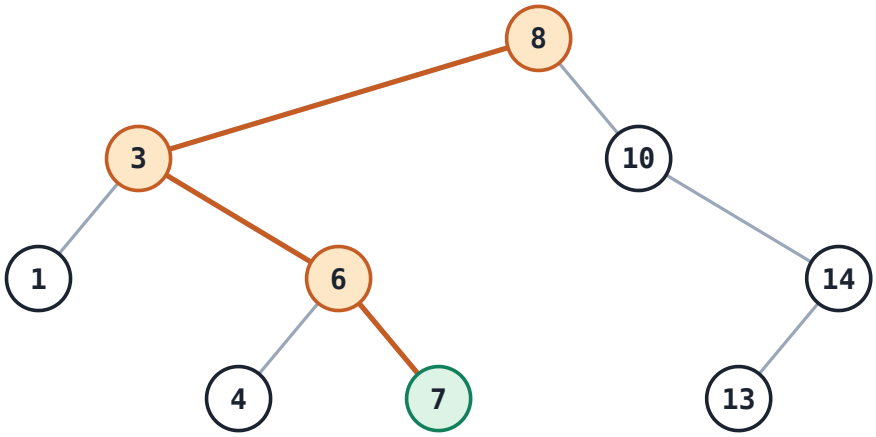


inserez 5: 5 < 8 stânga · 5 > 3 dreapta · 5 < 6 stânga · 5 > 4 dreapta → loc liber: nodul nou

Inserare

Căutarea

Același drum: la fiecare nod un singur răspuns (stânga sau dreapta) elimină jumătate din arbore.



caut 7: 4 comparații în loc de 9 — la fiecare pas jumătate din arbore dispare

Căutare

```cpp
#include <iostream>
using namespace std;

struct nod { int info; nod *st, *dr; };

void insereaza(nod *&r, int x)
{
    if (r == NULL)                                  // loc liber: creăm nodul
    {
        r = new nod;
        r->info = x;
        r->st = r->dr = NULL;
    }
    else if (x < r->info) insereaza(r->st, x);
    else if (x > r->info) insereaza(r->dr, x);      // x == info: duplicat, ignorăm
}

bool cauta(nod *r, int x)
{
    if (r == NULL) return false;
    if (x == r->info) return true;
    return x < r->info ? cauta(r->st, x) : cauta(r->dr, x);
}

void inordine(nod *r)
{
    if (r == NULL) return;
    inordine(r->st);
    cout << r->info << ' ';
    inordine(r->dr);
}

int inaltime(nod *r)
{
    if (r == NULL) return -1;
    return 1 + max(inaltime(r->st), inaltime(r->dr));
}

int main()
{
    int n, x;
    cin >> n;
    nod *radacina = NULL;
    for (int i = 1; i <= n; i++) { cin >> x; insereaza(radacina, x); }
    cin >> x;
    inordine(radacina);
    cout << '\n' << (cauta(radacina, x) ? "da" : "nu") << '\n' << inaltime(radacina);
    return 0;
}
```

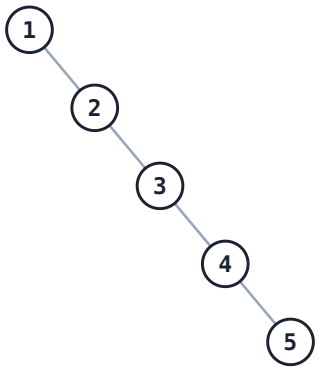
DE CE AȘA?

De ce `nod *&r` (pointer prin referință)? Când `r == NULL` creăm nodul și trebuie să „agățăm" pointerul nou de părintele lui (`r` este chiar câmpul `st` sau `dr` al părintelui). Fără `&`, am modifica o copie și nodul nou s-ar pierde (exact ca la `prim` în lecția XI.18). **De ce inordinea dă valorile sortate?** Pentru fiecare nod afișezi mai întâi tot ce e mai mic (stânga), apoi nodul, apoi tot ce e mai mare (dreapta): sortare „gratis".

LA BAC

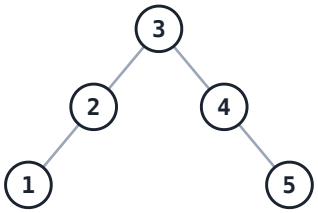
Grilă clasică: „ce se afișează la inordine pe arborele obținut prin inserarea valorilor 50, 30, 70, 20, 40, 60?": răspunsul este valorile **sortate**, 20 30 40 50 60 70. Nu mai desenezi nimic.

Ordinea inserării contează



1, 2, 3, 4, 5 (sortat): o „listă" → înălțime 4, căutare O(n)

Degenerat



3, 2, 4, 1, 5: echilibrat → înălțime 2, căutare O(log n)

Dacă valorile vin deja sortate, BST-ul devine o „listă" și căutarea costă O(n). De aceea bibliotecile folosesc arbori **echilibrați** (`std::set`, `std::map`, arbori AVL / roșu-negru).

6. Parcurgerea pe niveluri (cu coadă)

Nivel cu nivel: rădăcina, apoi fiii ei, apoi nepoții... Este BFS (lecția XI.17) pe arbore: o coadă în care punem fiii fiecărui nod scos.

```cpp
#include <iostream>
#include <queue>
using namespace std;

struct nod { int info; nod *st, *dr; };

void insereaza(nod *&r, int x)
{
    if (r == NULL) { r = new nod; r->info = x; r->st = r->dr = NULL; }
    else if (x < r->info) insereaza(r->st, x);
    else if (x > r->info) insereaza(r->dr, x);
}

int main()
{
    int n, x;
    cin >> n;
    nod *radacina = NULL;
    for (int i = 1; i <= n; i++) { cin >> x; insereaza(radacina, x); }
    queue<nod*> q;
    q.push(radacina);
    while (!q.empty())
    {
        int cat = q.size();                    // câte noduri are nivelul curent
        for (int i = 0; i < cat; i++)
        {
            nod *p = q.front(); q.pop();
            cout << p->info << ' ';
            if (p->st) q.push(p->st);
            if (p->dr) q.push(p->dr);
        }
        cout << '\n';
    }
    return 0;
}
```

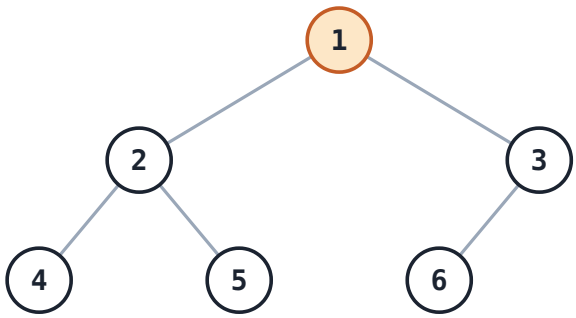
DE CE AȘA?

De ce `cat = q.size()` înainte de buclă? În timp ce procesezi nivelul, adaugi în coadă fiii (nivelul următor); dimensiunea măsurată la început separă exact nodurile nivelului curent.

7. Reconstituirea arborelui din două parcurgeri

Din **preordine singură** nu poți reconstitui arborele, dar din **preordine + inordine** da:

1. Primul element din preordine este **rădăcina**.
2. În inordine, tot ce este **la stânga** rădăcinii formează subarborele stâng, tot ce este la dreapta, cel drept.
3. Repeți pe fiecare parte.



preordine 1 2 4 5 3 6 → rădăcina 1 · inordine 4 2 5 | 1 | 6 3 → stânga {4,2,5}, dreapta {6,3}

Reconstituire

```cpp
#include <iostream>
using namespace std;

int n, pre[101], in[101];

// subarborele cu preordinea pre[a..b] și inordinea in[c..d]
void postordine(int a, int b, int c, int d)
{
    if (a > b) return;
    int rad = pre[a];
    int k = c;
    while (in[k] != rad) k++;                    // poziția rădăcinii în inordine
    int nrStanga = k - c;
    postordine(a + 1, a + nrStanga, c, k - 1);   // stânga
    postordine(a + nrStanga + 1, b, k + 1, d);   // dreapta
    cout << rad << ' ';
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> pre[i];
    for (int i = 1; i <= n; i++) cin >> in[i];
    postordine(1, n, 1, n);
    return 0;
}
```

8. Greșeli frecvente

Greșeala	Efect	Corect
uiți cazul `x == 0` / `r == NULL`	recursivitate infinită / blocare	prima linie a funcției
`inaltime(0) = 0`	frunza are înălțime 1	alege și păstrează o convenție
confunzi **inordine** cu **preordine**	ordine greșită	R între S și D (in), înainte (pre), după (post)

Greșeala	Efect	Corect
`insereaza(nod *r, ...)` fără `&`	arborele rămâne gol	`nod *&r`
presupui că BST-ul e echilibrat	O(n) la valori sortate	desenează arborele
condiția BST doar „fiu stâng < părinte < fiu drept”	arbore greșit	regula este pentru **tot subarborele**

9. La BAC

> **LA BAC**
>
> La arbori binari BAC-ul cere: **parcurgeri** pe un arbore desenat, **înălțime / număr de frunze / noduri** (formulele din tabelul de la început), **BST** (inserare + inordine). Metoda sigură: desenezi arborele, apoi aplici regula (R, S, D) cu un deget pe desen.

10. Exersează pe hârtie

1. Desenează BST-ul obținut din 8, 3, 10, 1, 6, 14, 4, 7, 13 și scrie preordinea, inordinea și postordinea.

2. Reconstituie arborele din preordinea 1 2 4 5 3 6 și inordinea 4 2 5 1 6 3. Scrie postordinea.

3. Ce afișează inordinea arborelui obținut din 50, 30, 70, 20, 40, 60?

4. Scrie o funcție care returnează valoarea minimă dintr-un BST nevid.

5. Un arbore binar plin are înălțimea 3. Câte noduri și câte frunze are?

6. Care este numărul maxim de frunze al unui arbore binar de înălțime 4?

7. Un arbore în care fiecare nod are 0 sau 2 fii are 7 frunze. Câte noduri are în total?

8. Se inserează în ordine 5, 3, 8, 3, 5 într-un BST care ignoră duplicatele. Ce afișează inordinea?

9. Scrie o funcție `suma(nod *r)` care returnează suma valorilor dintr-un arbore binar.

10. Ce arbore obții inserând 1, 2, 3, 4, 5, 6 într-un BST? Câte comparații costă căutarea lui 6?

11. Mai departe (pentru cine vrea mai mult)

Ștergerea într-un BST. Trei cazuri: nod frunză (îl scoți), un singur fiu (îl înlocuiești cu fiul), doi fii (îl înlocuiești cu cea mai mică valoare din subarborele drept, apoi ștergi acel nod).

```cpp
#include <iostream>
using namespace std;

struct nod { int info; nod *st, *dr; };

void insereaza(nod *&r, int x)
{
    if (r == NULL) { r = new nod; r->info = x; r->st = r->dr = NULL; }
    else if (x < r->info) insereaza(r->st, x);
    else if (x > r->info) insereaza(r->dr, x);
}

nod *minim(nod *r) { while (r->st) r = r->st; return r; }

void sterge(nod *&r, int x)
{
    if (r == NULL) return;
    if (x < r->info) sterge(r->st, x);
    else if (x > r->info) sterge(r->dr, x);
    else if (r->st == NULL || r->dr == NULL)        // 0 sau 1 fiu
    {
        nod *q = r;
        r = (r->st != NULL) ? r->st : r->dr;
        delete q;
    }
    else                                            // 2 fii
    {
        r->info = minim(r->dr)->info;               // succesorul în inordine
        sterge(r->dr, r->info);
    }
}

void inordine(nod *r)
{
    if (r == NULL) return;
    inordine(r->st);
    cout << r->info << ' ';
    inordine(r->dr);
}

int main()
{
    int n, x;
    cin >> n;
    nod *radacina = NULL;
    for (int i = 1; i <= n; i++) { cin >> x; insereaza(radacina, x); }
    cin >> x;
    sterge(radacina, x);
    inordine(radacina);
    return 0;
}
```

Număr de arbori binari cu n noduri. Numerele Catalan: 1, 1, 2, 5, 14, 42... (pentru 3 noduri sunt 5 arbori).

Arbori echilibrați. AVL și roșu-negru fac rotații după fiecare inserare; `std::set` și `std::map` din STL sunt arbori roșu-negru.

Heap (arbore binar complet într-un vector). Fiii lui i sunt 2i și 2i + 1; `priority_queue` este un heap: scoate maximul în O(log n).

Rezumat

- Arbore binar: fiecare nod are cel mult un fiu stâng și unul drept; reprezentare cu `st[]`, `dr[]` sau `struct nod`.
- Preordine R S D, inordine S R D, postordine S D R; caz de bază: nod inexistent.
- Înălțime = 1 + max(înălțime stânga, înălțime dreapta) (cu −1 pentru arborele vid).
- BST: stânga < nod < dreapta; inordinea dă șirul sortat; căutare și inserare pe un singur drum.
- Preordine + inordine reconstituie arborele; pe niveluri se parcurge cu o coadă.

Probleme de lucru (pbinfo)

Începe cu acestea, în ordine: sunt alese să urmeze pașii din lecție.

- [ ] #670 **preordine** — ușoară
  https://www.pbinfo.ro/probleme/670/preordine
- [ ] #671 **inordine** — ușoară
  https://www.pbinfo.ro/probleme/671/inordine
- [ ] #672 **postordine** — ușoară
  https://www.pbinfo.ro/probleme/672/postordine

Temă (pbinfo)

Trimite soluțiile pe pbinfo și notează punctajul obținut. Bifează ce ai terminat.

- [ ] #675 **bifrunze** — ușoară
  https://www.pbinfo.ro/probleme/675/bifrunze
- [ ] #761 **inaltime1** — ușoară
  https://www.pbinfo.ro/probleme/761/inaltime1
- [ ] #757 **bimax** — ușoară
  https://www.pbinfo.ro/probleme/757/bimax
- [ ] #677 **nivelebin** — medie
  https://www.pbinfo.ro/probleme/677/nivelebin
- [ ] #3010 **bst** — medie
  https://www.pbinfo.ro/probleme/3010/bst

Pentru cei care vor mai mult

Probleme mai grele sau de concurs, pentru cine vrea să meargă dincolo de BAC.

- [ ] #759 **bilatime** — medie
  https://www.pbinfo.ro/probleme/759/bilatime

Grile din subiectele de BAC (pbinfo)

Exerciții de tip grilă din variantele de Bacalaureat, rezolvate online pe pbinfo, cu verificare automată a răspunsului.

- [ ] #1137 **Exercițiul #1137**
  arbore asociat unei expresii: parcurgeri
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1137
- [ ] #1153 **Exercițiul #1153**
  înălțimea unui arbore binar
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1153
- [ ] #1154 **Exercițiul #1154**
  număr maxim de frunze
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1154
- [ ] #1165 **Exercițiul #1165**
  câți arbori binari cu 3 noduri
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1165
- [ ] #1166 **Exercițiul #1166**
  frunze și înălțime minimă
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1166
- [ ] #1175 **Exercițiul #1175**
  arbore binar plin: număr de noduri
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1175
- [ ] #1569 **Exercițiul #1569**
  nod intern cu doi descendenți: numărare
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1569
- [ ] #1643 **Exercițiul #1643**
  arbore binar complet (0 sau 2 fii)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1643

Citește și exersează în plus

- Arbori binari https://www.pbinfo.ro/articole/25641/arbori-binari
- Arbori binari de căutare https://www.pbinfo.ro/articole/25704/arbori-binari-de-cautare

- VisuAlgo — Binary Search Tree — inserare, căutare, ștergere și parcurgeri animate (engleză)
  https://visualgo.net/en/bst
- USACO Guide — Binary Trees — de la BST la set/map din STL (engleză)
  https://usaco.guide/silver/binary-search-tree
- cppreference — std::set — BST echilibrat gata făcut, cu inserare/căutare O(log n)
  https://en.cppreference.com/w/cpp/container/set

Răspunsuri

Exersează pe hârtie

1. Preordine: **8 3 1 6 4 7 10 14 13**; inordine: **1 3 4 6 7 8 10 13 14**; postordine: **1 4 7 6 3 13 14 10 8**.

2. Rădăcina 1; în inordine: stânga {4, 2, 5}, dreapta {6, 3}. Pe stânga: rădăcina 2, fii 4 și 5. Pe dreapta: rădăcina 3, fiu stâng 6. Postordinea: **4 5 2 6 3 1**.

3. Valorile sortate: **20 30 40 50 60 70**.

4. `int minim(nod *r) { while (r->st != NULL) r = r->st; return r->info; }`

5. Noduri: $2^4 - 1 =$ **15**; frunze: $2^3 =$ **8**.

6. $2^4 =$ **16** frunze.

7. Noduri interne = frunze − 1 = 6, deci în total 7 + 6 = **13** noduri.

8. 5 (rădăcina), 3 (stânga), 8 (dreapta); a doua apariție a lui 3 și a lui 5 este ignorată. Inordinea: **3 5 8**.

9. `int suma(nod *r) { if (r == NULL) return 0; return r->info + suma(r->st) + suma(r->dr); }`

10. Un lanț de fii drepți (fiecare valoare e mai mare decât toate cele dinainte): înălțime 5; căutarea lui 6 face **6 comparații** (O(n)).