AVANSAT · OLIMPIADĂ · LECȚIA 8

Arbori de intervale (segment tree) și propagare leneșă

Durată: 90 min Nivel: olimpiadă / informatică intensiv Înainte: lecțiile co-07, xi-10

CE VEI ȘTI DUPĂ LECȚIE

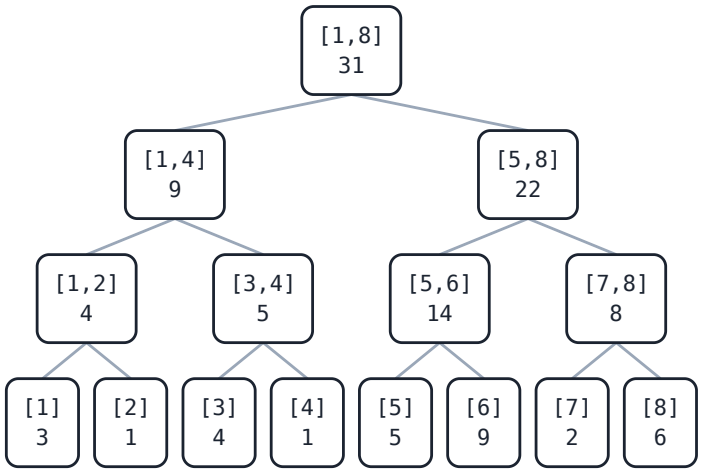
- Construiești un arbore de intervale și explici ce reține fiecare nod.
- Scrii recursiv actualizarea unui punct și interogarea unui interval, în O(log n).
- Adaptezi arborele pentru alte operații: minim, maxim, cmmdc.
- Folosești propagarea leneșă (lazy) pentru actualizări pe interval.

1. De ce încă o structură?

Arborele Fenwick (co-07) merge de minune pentru **sume**. Dar nu poate răspunde, de exemplu, la „care este **minimul** pe `[l, r]`?" cu actualizări. **Arborele de intervale** (*segment tree*) este mai general: orice operație **asociativă** pe un interval se poate pune în el (sumă, minim, maxim, cmmdc, produs modulo ...), iar cu **propagare leneșă** acceptă și actualizări pe interval.

2. Ideea: împărțim șirul în jumătăți

Rădăcina reprezintă tot șirul `[1, n]`. Fiecare nod `[l, r]` cu `l < r` are doi copii: `[l, m]` și `[m + 1, r]`, cu `m = (l + r) / 2`. Frunzele sunt elementele. În fiecare nod reținem **rezultatul pe intervalul lui** (aici, suma).



rădăcina acoperă tot șirul; fiecare nod se împarte în două jumătăți

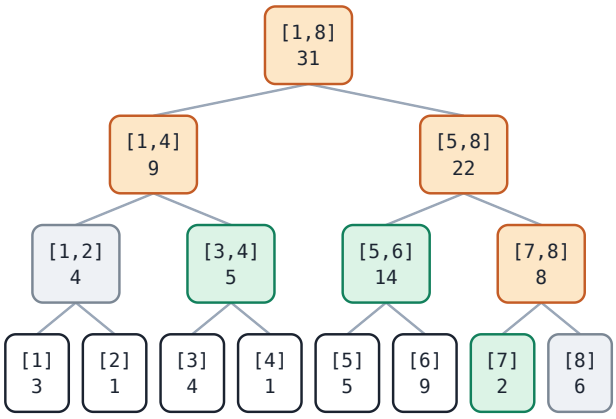
Arbore de intervale

Înălțimea arborelui este $\lceil \log_2 n \rceil + 1$. Numerotăm nodurile ca la un heap: rădăcina 1, copiii lui `nod` sunt `2·nod` și `2·nod + 1`. Vectorul trebuie să aibă mărimea **4·n** (ca să fie sigur de ajuns).

3. Interogare: suma pe [ql, qr]

Coborâm din rădăcină. Pentru un nod `[st, dr]` avem trei cazuri:

Caz	Ce facem
`[st, dr]` **în afara** lui `[ql, qr]`	întoarcem 0 (elementul neutru pentru sumă)
`[st, dr]` **complet în interiorul** lui `[ql, qr]`	întoarcem valoarea din nod
acoperire parțială	coborâm în ambii copii și adunăm rezultatele



verde = noduri incluse complet (se adună); portocaliu = acoperire parțială (coborâm); gri = în afara intervalului

Interogare

Pentru `[3, 7]`: coborâm din `[1,8]`, apoi `[1,4]` și `[5,8]`; de acolo, nodurile complet incluse sunt `[3,4]` (suma 5), `[5,6]` (14) și `[7,7]` (2): 5 + 14 + 2 = **21**.

> **DE CE AȘA?**
>
> **De ce O(log n) și nu O(n)?** Pe fiecare nivel, interogarea vizitează **cel mult 4** noduri, iar cel mult 2 dintre ele sunt cu acoperire parțială (cele de la capetele intervalului). Intervalul `[ql, qr]` se descompune în cel mult $2 \cdot \log_2 n$ noduri complet incluse, iar acelea sunt singurele care se adună.

4. Actualizare de punct

Coborâm până la frunza dorită, o schimbăm, apoi, la întoarcerea din recursivitate, **refacem** valorile părinților din copii.

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

int n, v[100005];
long long arb[400020];                      // 4 * n noduri

void construieste(int nod, int st, int dr)
{
    if (st == dr) { arb[nod] = v[st]; return; }
    int mij = (st + dr) / 2;
    construieste(2 * nod, st, mij);
    construieste(2 * nod + 1, mij + 1, dr);
    arb[nod] = arb[2 * nod] + arb[2 * nod + 1];   // suma copiilor
}

void actualizeaza(int nod, int st, int dr, int poz, int valoare)   // v[poz] = valoare
{
    if (st == dr) { arb[nod] = valoare; return; }
    int mij = (st + dr) / 2;
    if (poz <= mij) actualizeaza(2 * nod, st, mij, poz, valoare);
    else actualizeaza(2 * nod + 1, mij + 1, dr, poz, valoare);
    arb[nod] = arb[2 * nod] + arb[2 * nod + 1];   // refacem părintele
}

long long suma(int nod, int st, int dr, int ql, int qr)
{
    if (qr < st || dr < ql) return 0;              // în afara intervalului
    if (ql <= st && dr <= qr) return arb[nod];     // complet în interior
    int mij = (st + dr) / 2;
    return suma(2 * nod, st, mij, ql, qr) + suma(2 * nod + 1, mij + 1, dr, ql, qr);
}

int main()
{
    int q;
    cin >> n >> q;
    for (int i = 1; i <= n; i++) cin >> v[i];
    construieste(1, 1, n);
    while (q--)
    {
        int tip, a, b;
        cin >> tip >> a >> b;
        if (tip == 1) actualizeaza(1, 1, n, a, b);        // v[a] = b
        else cout << suma(1, 1, n, a, b) << '\n';         // suma v[a..b]
    }
    return 0;
}
```

Operația	Rezultat
`2 3 7`	4 + 1 + 5 + 9 + 2 = 21
`1 3 10`	v[3] devine 10 (părinții `[3,4]`, `[1,4]`, `[1,8]` se refac)
`2 3 7`	10 + 1 + 5 + 9 + 2 = 27

Operația	Rezultat
`2 1 8`	3 + 1 + 10 + 1 + 5 + 9 + 2 + 6 = 37
`1 8 0`	v[8] devine 0
`2 5 8`	5 + 9 + 2 + 0 = 16

DE CE AȘA?

De ce întoarcem 0 pentru un nod în afara intervalului? Zero este elementul **neutru** pentru adunare: adunarea lui nu schimbă nimic. Pentru **minim** elementul neutru este „infinit" (`INT_MAX`), pentru **maxim** este „minus infinit", pentru **cmmdc** este 0. **De ce se poate pune orice operație?** Doar dacă este **asociativă** (`(a ∘ b) ∘ c = a ∘ (b ∘ c)`), ca să poți combina rezultatele copiilor în orice grupare.

5. Altă operație: minimul

Schimbi **două** lucruri: combinarea (`min` în loc de `+`) și elementul neutru (`INT_MAX` în loc de 0):

```
arb[nod] = min(arb[2 * nod], arb[2 * nod + 1]);      // în construieste și actualizeaza
if (qr < st || dr < ql) return INT_MAX;              // în interogare (include <climits>)
return min(minim(2 * nod, st, mij, ql, qr), minim(2 * nod + 1, mij + 1, dr, ql, qr));
```

SFAT

Dacă ai nevoie de **poziția** minimului, reții în nod perechea `(valoare, poziție)` . Dacă ai nevoie de **două** rezultate (de exemplu minim și număr de apariții), reții o structură cu doi membri și o funcție de combinare.

6. Actualizări pe interval: propagare leneșă (lazy)

Dacă trebuie „adună x la toate elementele din `[l, r]` " cu query pe sume, a actualiza fiecare element costă O(n). Ideea **leneșă**: nu coborâm până la frunze. Când un nod este **complet** acoperit de actualizare, îi **actualizăm suma** și lăsăm în el o **notă** `lz[nod] = x` : „copiii tăi încă trebuie să adune x, dar nu ți-am spus încă". Nota coboară doar când trecem prin acel nod mai târziu.

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

int n, v[100005];
long long arb[400020], lz[400020];          // lz[nod] = ce mai trebuie adunat copiilor

void aplica(int nod, int st, int dr, long long x)   // nodul întreg primește +x
{
    arb[nod] += x * (dr - st + 1);                  // suma crește cu x pentru fiecare
element
    lz[nod] += x;                                   // nota pentru copii
}

void propaga(int nod, int st, int dr)               // trimite nota la copii
{
    if (lz[nod] == 0) return;
    int mij = (st + dr) / 2;
    aplica(2 * nod, st, mij, lz[nod]);
    aplica(2 * nod + 1, mij + 1, dr, lz[nod]);
    lz[nod] = 0;
}

void construieste(int nod, int st, int dr)
{
    if (st == dr) { arb[nod] = v[st]; return; }
    int mij = (st + dr) / 2;
    construieste(2 * nod, st, mij);
    construieste(2 * nod + 1, mij + 1, dr);
    arb[nod] = arb[2 * nod] + arb[2 * nod + 1];
}

void adauga(int nod, int st, int dr, int ql, int qr, long long x)   // v[ql..qr] += x
{
    if (qr < st || dr < ql) return;
    if (ql <= st && dr <= qr) { aplica(nod, st, dr, x); return; }   // nu coborâm: lăsăm
nota
    propaga(nod, st, dr);                           // înainte de a coborî, copiii trebuie
la zi
    int mij = (st + dr) / 2;
    adauga(2 * nod, st, mij, ql, qr, x);
    adauga(2 * nod + 1, mij + 1, dr, ql, qr, x);
    arb[nod] = arb[2 * nod] + arb[2 * nod + 1];
}

long long suma(int nod, int st, int dr, int ql, int qr)
{
    if (qr < st || dr < ql) return 0;
    if (ql <= st && dr <= qr) return arb[nod];
    propaga(nod, st, dr);                           // și interogarea trebuie să vadă
copiii la zi
    int mij = (st + dr) / 2;
    return suma(2 * nod, st, mij, ql, qr) + suma(2 * nod + 1, mij + 1, dr, ql, qr);
}

int main()
{
    int q;
```

```
    cin >> n >> q;
    for (int i = 1; i <= n; i++) cin >> v[i];
    construieste(1, 1, n);
    while (q--)
    {
        int tip, l, r;
        cin >> tip >> l >> r;
        if (tip == 1)
        {
            long long x;
            cin >> x;
            adauga(1, 1, n, l, r, x);
        }
        else cout << suma(1, 1, n, l, r) << '\n';
    }
    return 0;
}
```

Operația	Ce se întâmplă	Rezultat
`2 1 5`	suma tuturor	15
`1 2 4 10`	`[2,4]` primește +10 (câteva noduri primesc notă)	–
`2 1 3`	1 + 12 + 13	26
`2 4 5`	14 + 5	19

DE CE AȘA?

De ce `propaga` înainte de coborâre? Dacă nodul are o notă neînaintată și coborâm în copii fără s-o trimitem, copiii ar avea valori vechi, iar apoi, când recalculăm `arb[nod]` din copii, am pierde actualizarea. **De ce `x * (dr - st + 1)` ?** Fiecare dintre cele `dr - st + 1` elemente ale nodului crește cu x, deci suma crește cu produsul. **Cost:** tot O(log n) pe operație: fiecare operație atinge O(log n) noduri și propagă doar pe drumul parcurs.

ATENȚIE — GREȘELI FRECVENTE

Pentru lazy, tipul trebuie să fie `long long` și atât `arb` cât și `lz` trebuie să aibă mărimea 4·n. Cea mai frecventă greșeală este să uiți `propaga` în **interogare**: actualizările par să funcționeze, dar query dă valori vechi.

7. Comparație: când ce folosești?

Problema	Structura potrivită
sumă pe interval, fără modificări	sume parțiale
sumă pe interval, cu actualizări de punct	Fenwick (scurt) sau arbore de intervale
minim / maxim / cmmdc pe interval cu actualizări	arbore de intervale
adunare pe interval + sumă pe interval	arbore de intervale cu lazy
minim pe interval, **fără** modificări	tablou rar (sparse table), O(1) pe query

8. Greșeli frecvente

Greșeala	Ce se întâmplă	Corect
vector prea mic (`2·n`)	acces în afara vectorului	`4·n`
neutrul greșit	rezultate incorecte	0 la sumă, `INT_MAX` la minim, `-INT_MAX` la maxim
`mij = (st + dr) / 2` cu intervale calculate greșit	noduri suprapuse sau sărite	copiii sunt `[st, mij]` și `[mij + 1, dr]`
uiți să reconstruiești părintele după actualizare	sume incorecte	`arb[nod] = combinare(copii)`
uiți `propaga` în query	valori vechi	`propaga` la fiecare coborâre
`int` la sume sau la produsul `x * lungime`	depășire	`long long`
rescrii arborele la fiecare query	O(n)	construiește o singură dată

9. La olimpiadă

Arborele de intervale este structura cea mai des folosită la nivel de olimpiadă județeană și națională: „maxim pe interval", „al k-lea element", „numărul de elemente distincte" (offline), „cel mai lung subșir crescător cu Fenwick sau arbore", „drumuri în arbori (după descompunere)". Memorează scheletul din lecție: construiește, actualizeaza, interogheaza și, la nevoie, `propaga`. Pe orice problemă alegi doar combinarea și elementul neutru.

10. Exersează pe hârtie

1. Câte noduri are arborele (cu frunze) pentru n = 8? Dar pentru n = 6?

2. Pentru `v = 2 7 1 8 2 8 1 8`, scrie sumele nodurilor `[1,8]`, `[1,4]`, `[5,8]`, `[1,2]`, `[7,8]`.

3. La interogarea `[2, 5]` pe n = 8, care sunt nodurile complet incluse?

4. Ce întoarce `suma` pentru un nod în afara intervalului? De ce?

5. Ce schimbi pentru **maxim** pe interval?

6. Ce schimbi pentru **cmmdc** pe interval (`__gcd`)?

7. În `[1,8]` cu `v` ca la exercițiul 2: după `actualizeaza(poz = 4, valoare = 0)`, ce noduri își schimbă valoarea?

8. De ce, după o actualizare pe interval cu lazy, trebuie `propaga` înainte să coborâm?

9. Cât timp costă o actualizare de punct? Dar una pe interval cu lazy?

10. Pentru n = 10^5 și 10^5 operații, de ce o soluție O(n) pe operație nu trece?

11. Mini-grile

1. Mărimea sigură a vectorului unui arbore de intervale este: a) n b) 2n c) 4n d) n^2

2. Elementul neutru pentru minim este: a) 0 b) −1 c) `INT_MAX` d) `INT_MIN`

3. Complexitatea unei interogări este: a) O(1) b) O(log n) c) O(n) d) O(n log n)

4. `propaga` trimite: a) valoarea nodului la părinte b) nota nodului către copii c) nota către frunze direct d) nimic

12. Mai departe (pentru cine vrea mai mult)

- **Arbore de intervale cu structură** în nod (de exemplu „cel mai lung subșir de 1" sau „suma maximă a unui subșir").
- **Coborâre în arbore** („prima poziție cu valoare ≥ x") în O(log n).
- **Arbore persistent** (păstrezi versiunile vechi).
- **Descompunere în lanțuri grele** (HLD): arbore de intervale pe drumuri în arbori.
- **Tablou rar (sparse table):** minim pe interval în O(1) fără modificări.

Rezumat

- Fiecare nod reține rezultatul pe un interval; copiii împart intervalul în jumătăți; mărimea 4·n.
- **Interogare:** în afara → neutru; complet în interior → valoarea nodului; parțial → coboară în ambii copii.
- **Actualizare de punct:** coborâre + reconstruire la întoarcere; **interval:** lazy.
- Se potrivește pentru orice operație asociativă; O(log n) pe operație.
- Lazy: nodul complet acoperit primește valoarea și o notă; `propaga` înainte de coborâre, și în actualizare, și în interogare.

Probleme de lucru (concurs)

Probleme de pe infoarena, CSES, Kilonova sau pbinfo, în ordinea dificultății. Trimite soluțiile pe platformă și bifează ce ai terminat.

- [ ] `infoarena` **arbint** — ușoară
  maximul pe interval cu actualizare de punct: exact programul din lecție
  https://www.infoarena.ro/problema/arbint

- [ ] `CSES` **Dynamic Range Minimum Queries** — ușoară
  minim pe interval cu actualizări
  https://cses.fi/problemset/task/1649

- [ ] `CSES` **Range Update Queries** — medie
  adunare pe interval și citire de punct
  https://cses.fi/problemset/task/1651

- [ ] CSES **Range Queries and Copies** — dificilă
  arbore persistent: gândește-te la copierea nodurilor modificate
  https://cses.fi/problemset/task/1737

Temă (concurs)

Mai grele: încearcă-le după ce le-ai terminat pe cele de mai sus.

- [ ] CSES **Hotel Queries** — medie
  coborâre în arbore pe maxim
  https://cses.fi/problemset/task/1143
- [ ] CSES **Polynomial Queries** — dificilă
  propagare leneșă cu progresii aritmetice
  https://cses.fi/problemset/task/1736

Citește și exersează în plus

- CP-Algorithms — Segment Tree — articol de referință, cu propagare leneșă și variante (engleză)
  https://cp-algorithms.com/data_structures/segment_tree.html
- USACO Guide — Segment Tree — probleme graduate (engleză) https://usaco.guide/gold/segtree
- VisuAlgo — Segment Tree — animație pentru construire, interogare, actualizare (engleză)
  https://visualgo.net/en/segmenttree

Răspunsuri

Exersează pe hârtie

1. n = 8: 8 frunze + 4 + 2 + 1 = **15** noduri. n = 6: [1,6] → [1,3] și [4,6]; [1,3] → [1,2] și [3,3]; [1,2] → 1, 2; [4,6] → [4,5] și [6,6]; [4,5] → 4, 5. Total: **11** noduri (n frunze + n − 1 interne = 2n − 1).

2. `[1,8] = 37`, `[1,4] = 2+7+1+8 = 18`, `[5,8] = 2+8+1+8 = 19`, `[1,2] = 9`, `[7,8] = 9`.

3. `[2,5]`: `[2,2]`, `[3,4]` și `[5,5]`.

4. 0: elementul neutru la adunare; nu schimbă suma.

5. `max` în loc de `+` și neutrul `-INT_MAX` (sau `INT_MIN`).

6. `__gcd` în loc de `+` și neutrul 0.

7. Frunza `[4,4]`, apoi `[3,4]`, `[1,4]` și `[1,8]`. Noile valori: `[3,4] = 1`, `[1,4] = 10`, `[1,8] = 29`.

8. Ca să nu pierdem nota: copiii trebuie să aibă valori la zi înainte ca părintele să fie recalculat din ei sau înainte să citim din ei.

9. Punct: O(log n). Interval cu lazy: tot O(log n).

10. $10^5 \cdot 10^5 = 10^{10}$ operații: de zeci de secunde, cu mult peste limita de 1–2 secunde.

Mini-grile

1. c) 4n. **2.** c) `INT_MAX`. **3.** b) O(log n). **4.** b) nota nodului către copii.