AVANSAT · OLIMPIADĂ · LECȚIA 4

Căutare binară pe răspuns

Durată: 75 min Nivel: olimpiadă / informatică intensiv Înainte: lecțiile co-02, x-05

CE VEI ȘTI DUPĂ LECȚIE

- Recunoști o problemă de tip „cea mai mare / cea mai mică valoare pentru care se poate ...".
- Scrii predicatul ok(x) și verifici că este monoton.
- Implementezi căutarea binară pe răspuns fără greșeli la limite (lo, hi, mid, ans).
- Rezolvi și cazul real (cu zecimale), cu un număr fix de pași.

1. Căutarea binară: nu doar în vectori

Căutarea binară clasică găsește o valoare într-un vector sortat înjumătățind mereu intervalul. Dar ideea este mai generală: **oriunde ai un șir de „DA... DA, NU... NU"** (sau invers), poți găsi granița dintre ele în O(log n) pași.

Problemă-model. Ai n cabluri de lungimi date. Vrei să tai cel puțin k bucăți, **toate de aceeași lungime întreagă L**. Care este cea mai mare lungime L posibilă?

Nu avem vreun vector sortat în față, dar putem pune întrebarea: *„se pot tăia k bucăți de lungime L?"*



căutăm ultimul DA: L = 3 (cu L = 4 se taie doar 2 + 1 + 1 = 4 bucăți)

Predicat monoton

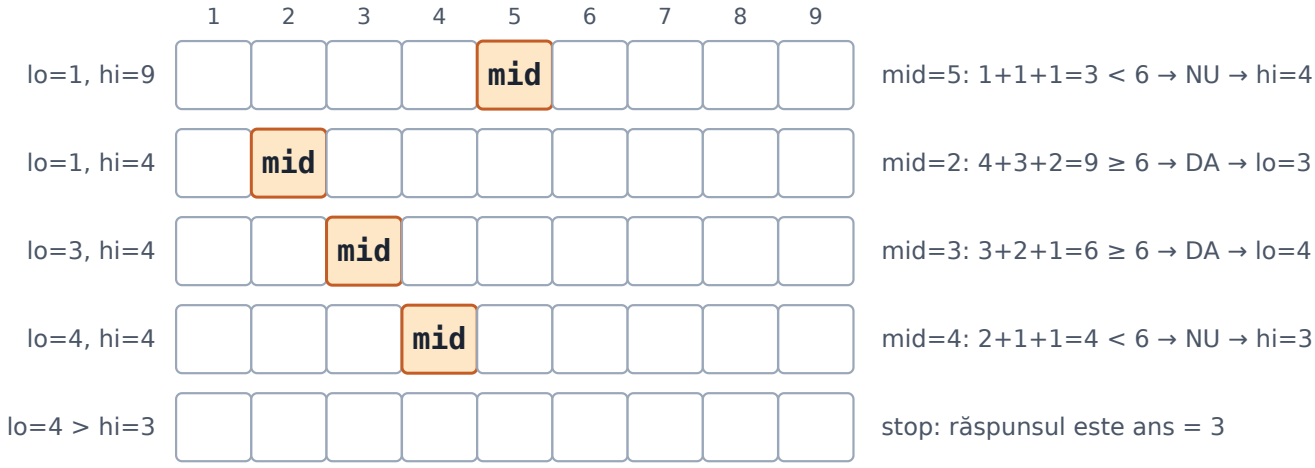
Dacă L = 3 se poate, atunci și L = 2 și L = 1 (bucăți mai mici, mai multe). Dacă L = 4 nu se poate, atunci nici L = 5, 6 ... nu se poate. Răspunsurile formează un șir **monoton**: DA, DA, DA, NU, NU, ... Căutăm **ultimul DA**.

DE CE AȘA?

De ce funcționează? Căutarea binară are nevoie de un singur lucru: o întrebare cu răspuns DA/NU care este **monotonă** (odată ce devine NU, rămâne NU). Atunci, la fiecare pas, o singură întrebare (pentru `mid`) elimină **jumătate** din valorile posibile. Nu contează că nu avem un vector: valorile posibile ale lui L (1, 2, ..., maximul) formează „vectorul" nostru virtual.

2. Șablonul: cea mai mare valoare cu `ok(x) = DA`

1. Stabilești intervalul posibil `[lo, hi]` (aici: de la 1 la cea mai lungă bucată).
2. Scrii funcția `ok(x)`: răspunde „se poate cu valoarea x?" (de obicei, un parcurs liniar al datelor).
3. Cauți binar, memorând ultimul DA în `ans`.



Pașii căutării

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

int n;
long long k, v[100005];

bool ok(long long L)                    // se pot tăia cel puțin k bucăți de lungime L?
{
    long long bucati = 0;
    for (int i = 0; i < n; i++) bucati += v[i] / L;
    return bucati >= k;
}

int main()
{
    cin >> n >> k;
    long long mx = 0;
    for (int i = 0; i < n; i++) { cin >> v[i]; mx = max(mx, v[i]); }
    long long lo = 1, hi = mx, ans = 0;    // ans = cea mai mare lungime care merge
    while (lo <= hi)
    {
        long long mid = (lo + hi) / 2;
        if (ok(mid)) { ans = mid; lo = mid + 1; }   // merge: încercăm mai mare
        else hi = mid - 1;                          // nu merge: încercăm mai mic
    }
    cout << ans;
    return 0;
}
```

Pas	lo	hi	mid	bucăți (9, 7, 5)	ok?	acțiune
1	1	9	5	1 + 1 + 1 = 3	NU	hi = 4
2	1	4	2	4 + 3 + 2 = 9	DA	ans = 2, lo = 3
3	3	4	3	3 + 2 + 1 = 6	DA	ans = 3, lo = 4
4	4	4	4	2 + 1 + 1 = 4	NU	hi = 3
–	4	3				lo > hi: stop; ans = 3

Dacă nu se poate nicio lungime (de exemplu k = 100), `ans` rămâne 0, valoarea cu care l-am inițializat (al doilea test).

DE CE AȘA?

De ce `lo = mid + 1` și `hi = mid - 1`, cu `ans` separat? După testul lui `mid` știm sigur răspunsul pentru `mid`, deci îl excludem din interval. Memorăm în `ans` ultimul `mid` care a fost DA: acela este candidatul curent. Nu trebuie să ghicim la final dacă `lo` sau `hi` este răspunsul. **De ce `while (lo <= hi)`** și nu `<`? Cu `<=` mai testăm cazul în care intervalul s-a redus la o singură valoare. **Câți pași?** Intervalul se înjumătățește: aproximativ $\log_2(mx) \approx 30$ de pași chiar și pentru $mx = 10^9$; fiecare costă O(n): total O(n log mx).

ATENȚIE — GREȘELI FRECVENTE

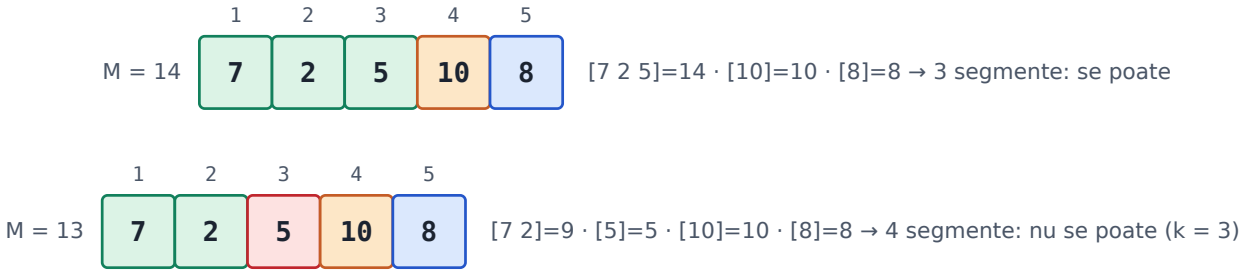
`(lo + hi) / 2` poate **depăși** `int` când lo și hi sunt mari (de exemplu $1{,}5 \cdot 10^9 + 1{,}5 \cdot 10^9$). Folosește `long long` pentru `lo`, `hi`, `mid`, sau scrie `lo + (hi - lo) / 2`.

3. Varianta „cea mai mică valoare cu `ok(x) = DA`"

Uneori șirul este „NU... NU, DA... DA" și căutăm **primul DA**. Atunci, când `ok(mid)` este DA, salvezi `ans = mid` și cobori (`hi = mid − 1`); când este NU, urci (`lo = mid + 1`).

Problemă. Un vector de n numere pozitive trebuie împărțit în **cel mult k segmente consecutive**. Care este cea mai mică valoare posibilă a **sumei maxime** dintr-un segment?

Predicat. `ok(M)` = „se poate împărți în cel mult k segmente, fiecare cu suma ≤ M?". Verificare **greedy**: parcurgi vectorul și pui elemente în segmentul curent cât timp suma nu depășește M; când ar depăși, începi un segment nou. Dacă ai folosit cel mult k segmente, M este suficient.



Segmente

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

int n, k;
long long v[100005];

bool ok(long long M)                    // se poate împărți în cel mult k segmente cu suma <= M?
{
    int segmente = 1;
    long long sum = 0;
    for (int i = 0; i < n; i++)
    {
        if (sum + v[i] > M) { segmente++; sum = 0; }   // segment nou
        sum += v[i];
    }
    return segmente <= k;
}

int main()
{
    cin >> n >> k;
    long long mx = 0, total = 0;
    for (int i = 0; i < n; i++) { cin >> v[i]; mx = max(mx, v[i]); total += v[i]; }
    long long lo = mx, hi = total, ans = total;   // M nu poate fi mai mic decât cel mai mare element
    while (lo <= hi)
    {
        long long mid = (lo + hi) / 2;
        if (ok(mid)) { ans = mid; hi = mid - 1; } // merge: încercăm mai mic
        else lo = mid + 1;
    }
    cout << ans;
    return 0;
}
```

Pentru 7 2 5 10 8 și k = 3: cu M = 14 ies segmentele [7 2 5], [10], [8] (3 segmente: se poate); cu M = 13 ies [7 2], [5], [10], [8] (4 segmente: nu se poate). Răspunsul este 14.

DE CE AȘA?

De ce `lo = mx` și `hi = total` ? Niciun segment nu poate avea suma mai mică decât cel mai mare element (acesta trebuie să încapă undeva). Iar cu M = suma totală, tot vectorul este un singur segment: sigur se poate. Răspunsul real este între cele două. **De ce este corect greedy în `ok` ?** Dacă tai cât mai târziu (segmentul cât mai lung), ai nevoie de cele mai puține segmente: nu poți pierde. **De ce `ok` este monoton?** Dacă se poate cu M, se poate și cu M + 1 (aceleași segmente rămân bune).

4. Căutarea pe răspuns real

Când răspunsul este un număr real (de exemplu o rădăcină), nu mai există „valoarea următoare”: intervalul se înjumătățește un **număr fix de ori** (sau până când `hi - lo` este mic).

Problemă. Găsește $x \geq 0$ astfel încât $x^3 + x = N$. Funcția crește odată cu x, deci „$x^3 + x < N$" este DA... DA, NU... NU.

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

int main()
{
    double N;
    cin >> N;                                  // găsim x >= 0 cu x^3 + x = N
    double lo = 0, hi = N + 1;
    for (int pas = 0; pas < 100; pas++)    // 100 de înjumătățiri: precizie mai mult decât suficientă
    {
        double mid = (lo + hi) / 2;
        if (mid * mid * mid + mid < N) lo = mid;   // prea mic
        else hi = mid;
    }
    cout << fixed << setprecision(6) << lo;
    return 0;
}
```

DE CE AȘA?

De ce un număr fix de pași și nu `while (lo != hi)`? Cu numere reale, `lo` și `hi` pot să nu devină niciodată **exact** egale (rotunjiri). După 100 de înjumătățiri intervalul are lungimea $(N + 1) / 2^{100}$, adică mult sub precizia lui `double`. **De ce `hi = N + 1`?** Pentru $x = N + 1$ avem $x^3 + x > N$ (orice $N \geq 0$), deci răspunsul este sigur în interval.

5. Cum recunoști problemele „pe răspuns"

Cuvinte din enunț	Întrebarea `ok(x)`
„cel mai mare ... posibil astfel încât ..."	se poate cu x? (de obicei mai mic e mai ușor)
„cel mai mic ... astfel încât ..."	ajunge x? (mai mare e mai ușor)
„minimizează maximul" / „maximizează minimul"	ok(x) cu greedy
n până la 10^5 – 10^6, răspuns până la 10^9 – 10^{18}	O(n · log răspuns)
răspunsul se verifică ușor, dar nu se calculează direct	căutare pe răspuns

DE ȘTIUT

Tiparul în doi pași: (1) **verifici** o valoare (greedy O(n)); (2) **găsești** valoarea cu căutare binară. Verificarea este de obicei mult mai simplă decât calcularea directă a răspunsului.

6. Greșeli frecvente

Greșeala	Ce se întâmplă	Corect
`ok` nu este monotonă	căutarea binară dă răspunsuri aleatorii	verifică pe exemple mici: DA...DA, NU...NU
`lo + hi` în `int`	depășire	`long long` sau `lo + (hi-lo)/2`
`ans` neinițializat	valoare nedefinită când nu există răspuns	inițializează cu valoarea „imposibil”
`lo = mid` / `hi = mid` fără ±1	buclă infinită	`mid ± 1` și `ans` separat
intervalul inițial prea mic	răspunsul real nu e în interval	`hi` suficient de mare (suma, N+1...)
numere reale cu `while (lo != hi)`	buclă infinită	număr fix de pași
greedy greșit în `ok`	răspuns greșit chiar dacă căutarea e corectă	demonstrează sau verifică greedy-ul separat

7. La olimpiadă

Enunțuri de tip „minimizați valoarea maximă” sau „cel mai mare k pentru care ...” aproape sigur se rezolvă pe răspuns. Pentru 100 de puncte, construiești `ok` în O(n) sau O(n log n) și cauți binar: O(n log R). Pentru punctaj parțial se testează valorile pe rând (O(n · R)), dar aceasta este prea lentă la limite mari: tocmai de aceea se folosește căutarea binară.

8. Exersează pe hârtie

1. Pentru cablurile 10 4 6 și k = 5, completează tabelul de pași al căutării binare (lo, hi, mid, ok) și află L maxim.

2. Pentru v = 1 2 3 4 5 și k = 2 segmente, care este cea mai mică sumă maximă? Verifică M = 8, 9, 10 cu `ok`.

3. Câți pași face căutarea binară pentru un interval de 10^9 valori? (Aproximativ.)

4. Cum ai modifica `ok(L)` din secțiunea 2 dacă bucățile pot avea lungime **reală** (nu doar întreagă)? Ce se schimbă în căutare?

5. De ce șablonul de la secțiunea 3 folosește `hi = mid - 1` când `ok(mid)` este DA?

6. Dă un exemplu de predicat **ne**monoton pe care căutarea binară nu poate fi aplicată.

7. Un grup de k muncitori trebuie să construiască n ziduri, fiecare zid cu timp dat; un zid se dă unui singur muncitor. Care este predicatul și monotonia pentru „timpul minim în care se termină totul”? (Aici zidurile se dau în ordine, consecutiv.)

8. Cu 100 de pași de înjumătățire și `hi - lo = 10^9` la început, cât este lungimea finală a intervalului?

9. Ce valori inițiale pentru `lo` și `hi` alegi la „minimizează suma maximă” dacă v = 5 5 5 și k = 3? Ce răspuns primești?

10. Pentru o problemă cu n = $2 \cdot 10^5$ și valori până la 10^9, estimează numărul de operații ale soluției O(n log R).

9. Mini-grile

1. Condiția necesară pentru căutarea binară pe răspuns: a) vectorul este sortat b) predicatul este monoton c) răspunsul este întreg d) n este mic

2. „Cea mai mare lungime L pentru care se pot tăia k bucăți”: când `ok(mid)` este DA, ce faci? a) `hi = mid - 1` b) `ans = mid; lo = mid + 1` c) `ans = mid; hi = mid - 1` d) te oprești

3. Pentru un interval de lungime 10^9 câți pași face căutarea binară? a) 10 b) aproximativ 30 c) 1000 d) 10^9

4. De ce la numere reale se folosește un număr fix de pași? a) pentru că e mai scurt b) pentru că lo și hi pot să nu devină exact egale c) pentru că reale nu au monotonie d) nu se face așa

10. Mai departe (pentru cine vrea mai mult)

- **Căutarea ternară** pentru funcții cu un singur maxim sau minim (nu monotone).
- **Căutare binară paralelă** (probleme cu mai multe întrebări, rezolvate simultan).
- **`lower_bound` și `upper_bound`** din STL: căutarea binară clasică, gata scrisă (lecția 1).
- **Căutare pe răspuns + alt algoritm în `ok`:** de exemplu, `ok(x)` rulează un BFS (drum cu muchii ≤ x), un DSU sau un flux.
- **Binary lifting** pentru strămoși în arbori (căutare binară pe biți).

Rezumat

- Căutarea binară merge peste orice șir monoton DA... DA, NU... NU, nu doar peste vectori sortați.
- Șablon: `ok(x)` + `lo`, `hi`, `mid` + `ans` memorat; actualizezi `lo = mid + 1` sau `hi = mid - 1`.
- Costul: O(n · log R), cu R mărimea intervalului de răspunsuri.
- Verificarea (`ok`) este adesea un greedy simplu O(n); demonstrează monotonia.
- Pentru răspuns real: un număr fix de pași (≈ 100), nu `while (lo != hi)`.

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` **cautbin** — ușoară
  căutare binară pe tablou: cele trei variante (exact, ≤, ≥)
  https://www.infoarena.ro/problema/cautbin

- [ ] CSES **Factory Machines** — medie
  timpul minim în care k mașini produc t produse: căutare pe răspuns
  https://cses.fi/problemset/task/1620
- [ ] CSES **Array Division** — medie
  exact împărțirea în segmente din lecție
  https://cses.fi/problemset/task/1085

Temă (concurs)

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

- [ ] USACO Guide **Aggressive cows (clasic)** — medie
  distanța minimă maximă dintre k puncte: căutare pe răspuns + greedy
  https://usaco.guide/silver/binary-search
- [ ] CSES **Sum of Three Values** — dificilă
  combină sortarea, doi pointeri și căutarea binară
  https://cses.fi/problemset/task/1641

Citește și exersează în plus

- USACO Guide — Binary Search — căutare binară pe tablou și pe răspuns, cu probleme graduate (engleză) https://usaco.guide/silver/binary-search
- CP-Algorithms — Binary search — articolul de referință, inclusiv căutarea pe răspuns real (engleză) https://cp-algorithms.com/num_methods/binary_search.html
- cppreference — std::lower_bound — varianta din STL pentru tablouri sortate (engleză) https://en.cppreference.com/w/cpp/algorithm/lower_bound

Răspunsuri

Exersează pe hârtie

1. Cabluri 10 4 6, k = 5: L = 1 → 20 bucăți; L = 2 → 5 + 2 + 3 = 10; L = 3 → 3 + 1 + 2 = 6; L = 4 → 2 + 1 + 1 = 4 < 5. Căutare: lo = 1, hi = 10: mid = 5 (2 + 0 + 1 = 3 NU) → hi = 4; mid = 2 (10 DA, ans = 2) → lo = 3; mid = 3 (6 DA, ans = 3) → lo = 4; mid = 4 (4 NU) → hi = 3: stop. **L = 3**.

2. M = 8: [1 2 3] [4] [5] sunt 3 segmente (>2): NU. M = 9: [1 2 3] [4 5] = 2 segmente: DA. M = 10 este DA, dar nu mai mic decât 9. Cea mai mică sumă maximă: **9**.

3. $\log_2 10^9 \approx 30$ de pași.

4. `ok(L)` rămâne `sum(floor(v[i] / L)) ≥ k` dacă tăierea este întreagă; pentru bucăți reale, `floor` dispare din „lungime", iar numărul de bucăți se calculează cu `(int)(v[i] / L)`; căutarea devine pe numere reale, cu număr fix de pași și fără `lo = mid + 1`.

5. Pentru că `mid` este deja DA și memorat în `ans`; căutăm doar valori mai mici decât `mid`.

6. „x este un număr prim?" pentru x = 1, 2, 3, 4, ...: răspunsurile alternează, nu există o graniță unică.

7. `ok(T)` = „zidurile consecutive se pot împărți în cel mult k grupe cu suma timpurilor ≤ T". Dacă se poate cu T, se poate și cu T + 1: monoton. Este exact problema din secțiunea 3.

8. $10^9 / 2^{100} \approx 10^9 / (1{,}27 \cdot 10^{30}) \approx 8 \cdot 10^{-22}$ (mult sub precizia double).

9. `lo = 5` (cel mai mare element), `hi = 15` (suma). `ok(5)` = 3 segmente (DA), deci răspunsul este **5**.

10. $\log_2 10^9 \approx 30$ și fiecare pas costă O(n) = $2 \cdot 10^5$: aproximativ $6 \cdot 10^6$ operații (sub o secundă, cu mare rezervă).

Mini-grile

1. b) predicatul este monoton. **2.** b). **3.** b) aproximativ 30. **4.** b).