AVANSAT · OLIMPIADĂ · LECȚIA 12

Strategie la concurs și simulare de olimpiadă

Durată: 120 min Nivel: olimpiadă / informatică intensiv Înainte: lecțiile co-01, co-04, co-05, co-06

CE VEI ȘTI DUPĂ LECȚIE

- Citești un enunț și alegi algoritmul pornind de la limitele de timp și de la constrângeri.
- Verifici soluția cu teste proprii și cu un program de forță brută (test aleator).
- Îți împarți timpul pe o probă de 3–4 ore și aduni puncte parțiale.
- Rezolvi o simulare cu 4 probleme în ordine crescătoare a dificultății.

1. Un concurs nu este un examen de teorie

La olimpiadă nu primești puncte pentru a ști multe, ci pentru **programe care trec testele** într-un timp limitat. Aceeași persoană poate lua 100 sau 30 de puncte pe aceeași problemă, în funcție de cum citește, de cum își verifică ideea și de ce face când se blochează. Lecția aceasta nu conține algoritmi noi: conține **obiceiurile** care transformă ce ai învățat în puncte.

2. Citirea enunțului (primele 5 minute pe problemă)

1. **Citește tot**, inclusiv exemplele și restricțiile. Cele mai multe puncte pierdute vin dintr-o condiție ratată: „distincte", „strict", „fiecare cel mult o dată", „indexare de la 1".
2. **Subliniază cuvintele-cheie** și **reformulează** problema în două propoziții.
3. **Rulează exemplul pe hârtie**. Dacă rezultatul tău nu coincide cu cel din enunț, nu ai înțeles problema.
4. **Notează restricțiile:** n, valorile maxime, timpul, memoria. Ele decid algoritmul.

Constrângeri → complexitate

Un calculator face aproximativ 10^8 operații simple pe secundă. Dacă limita este 1 secundă, ai nevoie de cel mult ~10^8 operații.

n maxim	Complexitate acceptabilă	Idei posibile
≤ 10	O(n!)	permutări, backtracking
≤ 20	$O(2^n \cdot n)$	submulțimi, măști de biți
≤ 500	$O(n^3)$	Roy-Floyd, DP cu trei parametri
≤ 5 000	$O(n^2)$	DP cu două dimensiuni, perechi
≤ 10^5 – 10^6	O(n log n)	sortare, heap, căutare binară, DSU, Fenwick, arbore de intervale
≤ 10^7 – 10^8	O(n)	ciur, sume parțiale, doi pointeri, DP liniar

n maxim	Complexitate acceptabilă	Idei posibile
$\leq 10^{18}$	O(log n)	formulă, putere rapidă, căutare binară

> **DE ȘTIUT**
>
> Tabelul este o **busolă**, nu o regulă: un $O(n^2)$ cu constantă mică poate trece la $n = 10^5$ (10^{10} operații simple nu trec, dar $5 \cdot 10^9$ poate, cu optimizări). Dar dacă $n = 10^6$ și ideea ta este $O(n^2)$, caută altă idee.

3. De la idee la program

Forța brută mai întâi (pe hârtie)

Înainte de orice idee deșteaptă, spune-ți: **„cum aș rezolva dacă n ar fi 5?"** Forța brută îți arată ce se calculează de fapt. Apoi întrebi: ce se repetă? Ce se poate precalcula (sume parțiale)? Ce se poate sorta? Ce se poate căuta binar?

Întreabă-te, în ordine

Întrebarea	Posibilă tehnică
e vorba de un interval, un prefix, o sumă?	sume parțiale, doi pointeri, Fenwick
pot sorta și lucrurile devin simple?	sortare + parcurgere, greedy
răspunsul crește monoton cu o valoare?	căutare binară pe răspuns
„cel mai mic / mai mare / câte moduri" cu alegeri repetate?	programare dinamică
apar noduri și legături?	BFS, DFS, DSU, Dijkstra
n ≤ 20 sau „toate submulțimile"?	măști de biți
se cere restul modulo $10^9 + 7$?	aritmetică modulară

Scrie curat, nu repede

- Folosește **nume clare** și **puține variabile globale**.
- Declară matricele cu **dimensiunea maximă + puțin** (`100005` , nu `100000`).
- **Citește rapid** când intrarea este mare: `ios::sync_with_stdio(false); cin.tie(nullptr);` .
- `long long` la sume și produse: dacă ai un dubiu, folosește-l.
- Scrie funcția cea mai mică întâi, testeaz-o, apoi continuă.

4. Verificarea: teste proprii și test aleator

Teste de colț

Înainte de a trimite, rulează programul pe: **cel mai mic caz** (n = 1), **toate elementele egale**, **valori maxime** (depășiri?), **șir deja sortat / invers sortat**, **răspuns 0 sau „nu există"**.

Testul aleator cu forță brută (stress test)

Scrii două programe: soluția ta `sol.cpp` și o soluție **simplă și sigură**, chiar dacă lentă, `brut.cpp`. Un al treilea program generează intrări mici la întâmplare. Compari ieșirile până găsești o diferență:

```
g++ -O2 -o sol sol.cpp
g++ -O2 -o brut brut.cpp
g++ -O2 -o gen gen.cpp
for i in $(seq 1 500); do
    ./gen $i > in.txt                    # generatorul primește numărul testului ca
sămânță
    ./sol < in.txt > out1.txt
    ./brut < in.txt > out2.txt
    if ! cmp -s out1.txt out2.txt; then
        echo "DIFERENTA la testul $i"
        break
    fi
done
```

Intrarea care diferă este un **contraexemplu mic**: îl rezolvi pe hârtie și afli cine are dreptate.

> **DE CE AȘA?**
>
> **De ce funcționează?** Greșelile din soluții rapide apar de obicei pe cazuri mici și ciudate (n = 1, valori egale, margini), pe care exemplele din enunț nu le acoperă. Un generator produce sute de astfel de cazuri într-un minut. Este cea mai bună investiție de 10 minute din concurs, pentru problemele pe care ești sigur că „merg" dar primești 40 de puncte.

5. Timpul unei probe de 3–4 ore

Etapa	Ce faci
minutele 0–10	citești **toate** problemele, notezi ideea posibilă și dificultatea estimată
apoi	începi cu **cea mai ușoară**: rezolv-o complet, apoi testează
apoi	treci la următoarea ca dificultate
ultima oră	revii la problema pe care ai un **scor parțial** și îl crești
ultimele 10 min	**nu** schimba nimic mare; verifică că ai trimis ultima versiune care trece

Puncte parțiale

Majoritatea probelor acordă puncte pe **teste**, nu totul-sau-nimic. Strategia:

1. **Forța brută corectă** ia testele mici (de obicei 20–40% din punctaj). O ai de la început.
2. **Cazuri speciale** (toate valorile egale, n mic, valori mici) iau încă puncte.
3. **Soluția completă** ia restul.

> **SFAT**
>
> O soluție lentă, dar **corectă**, valorează mai mult decât una rapidă și greșită. Trimite forța brută înainte de a căuta soluția optimă: așa ai puncte garantate.

Când te blochezi

1. Rulează exemplul din nou, pe hârtie. Ai înțeles bine problema?
2. Încearcă **cazuri mici** și caută un tipar.
3. Dacă ai petrecut 30 de minute fără progres, lasă problema pentru mai târziu și treci la alta; revenirea după o pauză ajută.
4. Nu rescrie totul când un test cade: caută **un** exemplu mic pe care greșește.

6. Greșeli frecvente la concurs

Greșeala	Cum o eviți
depășire de `int`	`long long` la sume și produse
index în afara vectorului	dimensiune cu rezervă, verifică limitele cu `at` la depanare
variabile neinițializate sau rămase de la testul anterior	inițializezi/șterge structurile la fiecare test
citire/afișare lentă	`sync_with_stdio(false)`, `'\n'` în loc de `endl`
formatul de ieșire (spații, rânduri)	copiază exact formatul din enunț
`endl` în buclă de 10^5 afișări	`'\n'`
fișiere (`infoarena`: `freopen`) greșite	verifică numele fișierelor din enunț
recursivitate prea adâncă (10^6 niveluri)	variantă iterativă sau stivă proprie
`a / b` cu întregi când vrei real	`(double) a / b`
ai testat doar exemplul din enunț	teste de colț + test aleator

7. Simulare de olimpiadă (2 ore, 4 probleme)

Rezolvă problemele în ordinea dificultății, cronometrat. Soluțiile sunt la sfârșitul paginii, în **Răspunsuri**: nu te uita înainte să ai o încercare serioasă. Fiecare problemă are limită de timp 1 secundă.

Problema 1: Perechi cu suma dată (≈ 15 min)

Se dă un șir de n numere întregi și un număr S. Să se afișeze **câte perechi de poziții** `i < j` au `v[i] + v[j] = S`.

Intrare: pe prima linie `n S`; pe a doua linie cele n numere. **Ieșire:** numărul de perechi. **Restricții:** 1 ≤ n ≤ $2 \cdot 10^5$, |v[i]|, |S| ≤ 10^9. **Exemplu:** `6 10` și `3 7 5 5 2 8` → `3` (perechile `3+7`, `5+5`, `2+8`).

Problema 2: Moduri de plată (≈ 25 min)

Ai monede de k tipuri (cu valori diferite, în număr nelimitat) și trebuie să plătești exact suma S. Câte **combinații** de monede dau suma S? Două combinații sunt aceleași dacă folosesc aceleași monede în aceleași cantități (ordinea nu contează). Afișează rezultatul modulo $10^9 + 7$.

Intrare: `k S`, apoi valorile monedelor. **Restricții:** 1 ≤ k ≤ 100, 1 ≤ S ≤ 10^5, 1 ≤ valoare ≤ 10^5. **Exemplu:** `3 5` și `1 2 5` → `4` (`5`; `2+2+1`; `2+1+1+1`; `1+1+1+1+1`).

Problema 3: Cabluri (≈ 30 min)

Ai n cabluri cu lungimile întregi `L[i]`. Vrei să le tai în bucăți **de aceeași lungime întreagă x** (o bucată se obține doar dintr-un singur cablu; resturile se aruncă) ca să obții cel puțin k bucăți. Care este cea mai mare lungime x posibilă? Dacă nu se poate nici cu x = 1, afișează 0.

Intrare: `n k`, apoi cele n lungimi. **Restricții:** 1 ≤ n ≤ 10^5, 1 ≤ k ≤ 10^9, 1 ≤ L[i] ≤ 10^9. **Exemplu:** `3 5` și `10 7 4` → `3` (cu x = 3 obții 3 + 2 + 1 = 6 bucăți ≥ 5; cu x = 4 doar 2 + 1 + 1 = 4).

Problema 4: Insule și poduri (≈ 40 min)

Un arhipelag are n insule. Se pot construi m poduri; podul i leagă insulele `a` și `b` și se termină în ziua `t`. Care este **prima zi** în care din orice insulă se poate ajunge în orice alta folosind podurile gata până în acea zi? Dacă niciodată, afișează `-1`.

Intrare: `n m`, apoi m linii `a b t`. **Restricții:** 1 ≤ n ≤ 10^5, 1 ≤ m ≤ $2 \cdot 10^5$, 1 ≤ t ≤ 10^9. **Exemplu:** `4 5` și podurile `1 2 3`, `2 3 1`, `3 4 4`, `1 4 2`, `1 3 5` → `3` (în ziua 3 sunt gata `2–3`, `1–4` și `1–2`: toate insulele sunt legate).

> **IMPORTANT**
>
> Înainte să scrii cod, pentru fiecare problemă răspunde pe hârtie: **(1)** ce înseamnă restricțiile pentru complexitate? **(2)** care este forța brută? **(3)** ce tehnică din lecțiile anterioare se potrivește? Abia apoi scrie.

8. Exersează pe hârtie

1. Un enunț are n ≤ 10^5 și cere „cel mai lung subșir crescător". Ce complexitate ai nevoie? Ce tehnică folosești?

2. Un enunț are n ≤ 18 și cere „numărul de submulțimi cu suma S". Ce complexitate acceptă? Ce tehnică?

3. O problemă cu n = 10^6 și un algoritm $O(n^2)$. Trece? De ce?

4. Ai o soluție care dă răspunsuri greșite pe testele mari, dar trece exemplul. Ce faci primul?

5. De ce se compară rezultatele cu o forță brută, nu cu „ce ar trebui să iasă"?

6. De ce se folosește `'\n'` în loc de `endl` când afișezi 10^5 linii?

7. Estimează: O(n log n) cu n = 10^6. Trece în 1 secundă?

8. Ai 3 ore și 4 probleme. Cum împarți timpul?

9. O soluție obține 60 de puncte; ce testezi ca să afli ce lipsește?

10. Ce înseamnă să „citești rapid" în C++? Ce două linii folosești?

9. Mini-grile

1. Pentru n = $2 \cdot 10^5$, un algoritm acceptabil este: a) $O(n^2)$ b) O(n log n) c) $O(2^n)$ d) O(n!)

2. Un stress test compară: a) două soluții rapide b) soluția ta cu una simplă și sigură, pe intrări aleatoare mici c) doar exemplul din enunț d) timpul de rulare

3. Dacă n ≤ 20, probabil se folosește: a) sortare b) măști de biți / backtracking c) Fenwick d) DSU

4. Pentru puncte parțiale, trimiți mai întâi: a) soluția optimă doar b) forța brută corectă c) nimic d) o soluție greșită rapidă

10. Mai departe (pentru cine vrea mai mult)

- **Concursuri online** regulate: Codeforces, AtCoder, infoarena, CodeChef. Participă la cel puțin unul pe lună.
- **Upsolving:** după concurs, rezolvă problemele pe care nu le-ai rezolvat în timp. Aici înveți cel mai mult.
- **Caiet de șabloane:** DSU, Fenwick, arbore de intervale, Dijkstra, putere rapidă, ciur: scrise o dată, curat, în fișiere separate.
- **Antrenament pe teme:** una sau două teme pe săptămână, cu 5–10 probleme pe temă.
- **Citirea soluțiilor oficiale** după fiecare problemă: compară ideea ta cu a lor.

Rezumat

- Constrângerile aleg algoritmul: n ≤ 20 → 2^n; n ≤ 5 000 → n^2; n ≤ 10^6 → n log n.
- Citește tot, rulează exemplul pe hârtie, notează restricțiile.
- Forța brută corectă dă puncte parțiale și servește la testul aleator.
- Teste de colț (n = 1, egale, maxime) și stress test cu un generator.
- Timp: citește toate problemele, începe cu cea mai ușoară, nu schimba mult în ultimele minute.

Probleme de lucru (concurs)

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

- [ ] CSES **Two Sum (CSES: Sum of Two Values)** — ușoară
  cuplul cu suma dată, cu sort + doi pointeri sau map
  https://cses.fi/problemset/task/1640

- [ ] CSES **Coin Combinations II** — medie
  numărul de moduri de plată (ordinea nu contează)
  https://cses.fi/problemset/task/1636
- [ ] CSES **Factory Machines** — medie
  căutare binară pe răspuns
  https://cses.fi/problemset/task/1620
- [ ] CSES **Road Reparation** — medie
  Kruskal și DSU
  https://cses.fi/problemset/task/1675

Temă (concurs)

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

- [ ] CSES **Stick Lengths** — ușoară
  median și sume de diferențe
  https://cses.fi/problemset/task/1074
- [ ] CSES **Movie Festival II** — dificilă
  greedy cu multiset
  https://cses.fi/problemset/task/1632

Citește și exersează în plus

- infoarena — arhiva de probleme — probleme românești, grupate pe tehnici
  https://www.infoarena.ro/arhiva-educationala
- CSES Problem Set — 300 de probleme standard, ordonate pe teme (engleză)
  https://cses.fi/problemset/
- USACO Guide — traseu de pregătire de la începător la avansat (engleză) https://usaco.guide/
- Codeforces — concursuri online săptămânale (engleză) https://codeforces.com/problemset

Răspunsuri

Exersează pe hârtie

1. O(n log n): LIS cu căutare binară (rețiI, pentru fiecare lungime, cel mai mic final posibil), nu O(n^2) din lecția XI.16. Cu Fenwick, tot O(n log n).

2. $2^{18} \cdot 18 \approx 4{,}7 \cdot 10^6$: foarte bine. Enumerare de submulțimi cu măști de biți.

3. Nu: 10^{12} operații, de sute de ori peste limită. Trebuie O(n log n) sau O(n).

4. Caută un contraexemplu: scrie o forță brută și un generator de teste mici, apoi compară (stress test).

5. „Ce ar trebui să iasă" nu se știe pe intrări aleatoare; forța brută este simplă și sigură, deci joacă rolul de judecător.

6. `endl` golește bufferul de ieșire la fiecare linie, ceea ce este lent; `'\n'` doar trece pe rândul următor.

7. $10^6 \cdot 20 = 2 \cdot 10^7$ operații: trece cu mare marjă.

8. Primele 10 minute: citești toate. Apoi probleme în ordinea dificultății, cu 15–20, 30, 40, 60 de minute (de exemplu); ultima oră revii la scoruri parțiale; ultimele 10 minute doar verifici.

9. Rulează programul pe cazuri mici cu forța brută (stress test), apoi pe valori maxime (depășiri, timp), pe cazuri de colț.

10. `ios::sync_with_stdio(false); cin.tie(nullptr);` la începutul lui `main`.

Mini-grile

1. b) O(n log n). **2.** b). **3.** b) măști de biți / backtracking. **4.** b) forța brută corectă.

Simulare: soluții

Problema 1. Parcurgem șirul de la stânga la dreapta; pentru fiecare `x` numărăm câte elemente **anterioare** sunt egale cu `S - x` (într-un `map`), apoi adăugăm `x`. Timp O(n log n). Rezultatul poate depăși `int` ($n^2/2$ perechi): `long long`.

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

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    long long S;
    cin >> n >> S;
    map<long long, int> aparitii;
    long long perechi = 0;
    for (int i = 0; i < n; i++)
    {
        long long x;
        cin >> x;
        auto it = aparitii.find(S - x);
        if (it != aparitii.end()) perechi += it->second;   // perechi cu elementele de
dinainte
        aparitii[x]++;
    }
    cout << perechi;
    return 0;
}
```

Problema 2. DP pe monede, **în ordinea monedelor** (pentru a număra combinații, nu secvențe): `dp[s]` = numărul de combinații care dau suma `s`; pentru fiecare monedă `m`, în ordine crescătoare a lui `s`: `dp[s] += dp[s − m]`. Timp O(k · S).

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

int main()
{
    const long long MOD = 1000000007;
    int k, S;
    cin >> k >> S;
    vector<long long> dp(S + 1, 0);
    dp[0] = 1;                                    // o singură combinație pentru suma 0:
nicio monedă
    for (int i = 0; i < k; i++)
    {
        int m;
        cin >> m;
        for (int s = m; s <= S; s++)
            dp[s] = (dp[s] + dp[s - m]) % MOD;   // folosim moneda m încă o dată
    }
    cout << dp[S];
    return 0;
}
```

Dacă ai inversa ordinea buclelor (întâi `s`, apoi monedele) ai număra **secvențe** (1+2 și 2+1 diferite), adică altă problemă.

Problema 3. Pentru un x dat, numărul de bucăți este `suma(L[i] / x)`. Funcția „se poate cu lungimea x" este **monotonă** (dacă merge cu x, merge cu orice valoare mai mică). Căutare binară pe răspuns, x între 1 și 10^9. Atenție: suma poate depăși `int` ($10^5 \cdot 10^9$).

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

int main()
{
    int n;
    long long k;
    cin >> n >> k;
    vector<long long> L(n);
    for (auto &x : L) cin >> x;

    long long st = 1, dr = 1000000000, rez = 0;
    while (st <= dr)
    {
        long long mij = (st + dr) / 2;
        long long bucati = 0;
        for (long long x : L) bucati += x / mij;
        if (bucati >= k) { rez = mij; st = mij + 1; }   // merge: încercăm mai mare
        else dr = mij - 1;
    }
    cout << rez;
    return 0;
}
```

Problema 4. Sortăm podurile după zi; le adăugăm într-un DSU (lecția co-06); ziua în care numărul componentelor devine 1 este răspunsul. Dacă la final rămân mai multe componente, `-1`. Timp O(m log m).

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

struct Pod { int a, b, t; };
int tata[100005], sz[100005];

int gaseste(int x)
{
    if (tata[x] == x) return x;
    return tata[x] = gaseste(tata[x]);
}

int main()
{
    int n, m;
    cin >> n >> m;
    vector<Pod> p(m);
    for (auto &x : p) cin >> x.a >> x.b >> x.t;
    sort(p.begin(), p.end(), [](const Pod &u, const Pod &v) { return u.t < v.t; });
    for (int i = 1; i <= n; i++) { tata[i] = i; sz[i] = 1; }

    int componente = n, raspuns = -1;
    if (n == 1) raspuns = 0;                        // o singură insulă: deja legată (ziua 0)
    for (auto &x : p)
    {
        int a = gaseste(x.a), b = gaseste(x.b);
        if (a == b) continue;
        if (sz[a] < sz[b]) swap(a, b);
        tata[b] = a;
        sz[a] += sz[b];
        if (--componente == 1) { raspuns = x.t; break; }
    }
    cout << raspuns;
    return 0;
}
```