AVANSAT · OLIMPIADĂ · LECȚIA 3

Sume parțiale pe matrice, vectori de diferențe și subsecvențe cu sumă dată

Durată: 75 min Nivel: olimpiadă / informatică intensiv Înainte: lecțiile co-01, co-02, x-07, x-09

CE VEI ȘTI DUPĂ LECȚIE

- Calculezi suma oricărui dreptunghi dintr-o matrice în O(1) după o precalculare O(n·m).
- Aplici mii de actualizări pe intervale (adaugă v pe [l, r]) în O(1) fiecare, cu vectorul de diferențe.
- Numeri subsecvențele cu suma exact K (și cu alte proprietăți) folosind sume parțiale și map.
- Recunoști când „precalculez ca să răspund repede" bate recalcularea la fiecare întrebare.

1. Ideea: precalculezi o dată, răspunzi de q ori

Ai un vector cu n = 10^5 numere și q = 10^5 întrebări „care este suma elementelor de la l la r?". Dacă aduni de fiecare dată, faci până la n·q = 10^{10} operații: prea mult. Dacă **precalculezi** o singură dată sumele parțiale, fiecare întrebare costă O(1) și programul face doar n + q = 2·10^5 operații.

Recapitulare (clasa a X-a). `sp[i] = v[1] + … + v[i]`, cu `sp[0] = 0`. Atunci suma de la l la r este `sp[r] − sp[l−1]`.

i	0	1	2	3	4	5
v[i]		4	1	3	2	5
sp[i]	0	4	5	8	10	15

Suma elementelor 2..4 este `sp[4] − sp[1] = 10 − 4 = 6` (1 + 3 + 2).

DE CE AȘA?

De ce `sp[0] = 0` și nu începem de la 1? Pentru l = 1 formula cere `sp[0]`: dacă acesta nu există (sau nu e 0), prefixele care încep de la primul element ar fi un caz special. Cu `sp[0] = 0` formula este aceeași pentru orice interval.

2. Sume parțiale pe matrice (2D)

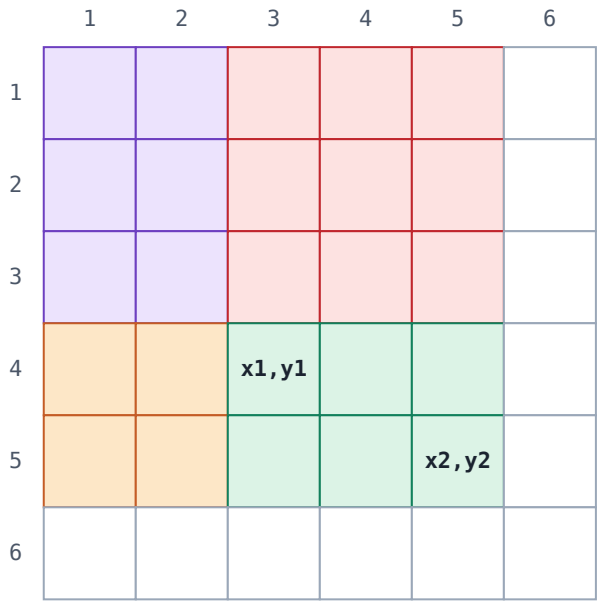
Problemă. O matrice cu n linii și m coloane și q întrebări „care este suma dreptunghiului cu colțurile (x1, y1) și (x2, y2)?". Același raționament: precalculăm `sp[i][j]` = suma dreptunghiului cu colțurile (1,1) și (i,j).

Cum se construiește `sp[i][j]`. Adunăm cele două dreptunghiuri vecine și scădem zona comună, numărată de două ori:

```
sp[i][j] = a[i][j] + sp[i−1][j] + sp[i][j−1] − sp[i−1][j−1]
```

Cum se interoghează. Suma dreptunghiului (x1, y1) – (x2, y2) se obține scăzând din „tot” cele două fâșii din afară, apoi adăugând colțul scăzut de două ori:

`sp[x2][y2] − sp[x1−1][y2] − sp[x2][y1−1] + sp[x1−1][y1−1]`



sp[x2][y2] − sp[x1−1][y2] − sp[x2][y1−1] + sp[x1−1][y1−1] (mov se adaugă la loc)

Includere-excludere

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

long long sp[1005][1005];                // sp[i][j] = suma dreptunghiului (1,1)-(i,j)

int main()
{
    int n, m, q;
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
        {
            int x;
            cin >> x;
            sp[i][j] = x + sp[i - 1][j] + sp[i][j - 1] - sp[i - 1][j - 1];
        }
    cin >> q;
    while (q--)
    {
        int x1, y1, x2, y2;
        cin >> x1 >> y1 >> x2 >> y2;
        cout << sp[x2][y2] - sp[x1 - 1][y2] - sp[x2][y1 - 1] + sp[x1 - 1][y1 - 1] << '\n';
    }
    return 0;
}
```

Pe matricea 3×4 din test: întreaga matrice are suma 78 (1 + 2 + … + 12); dreptunghiul (2,2)–(3,3) conține 6, 7, 10, 11, deci suma 34; dreptunghiul (2,3)–(2,4) conține 7 și 8, suma 15.

DE CE AȘA?

De ce `+ sp[x1−1][y1−1]` în interogare? Când scazi fâșia de sus (`sp[x1−1][y2]`) și fâșia din stânga (`sp[x2][y1−1]`), colțul din stânga-sus (mov în desen) a fost scăzut **de două ori**, fiindcă face parte din ambele fâșii. Îl adaugi o dată, ca să-l scazi efectiv o singură dată. **De ce `long long` ?** Chiar dacă fiecare număr încape în `int` , suma pe 10^6 celule poate depăși $2 \cdot 10^9$.

3. Vectorul de diferențe: actualizări pe interval în O(1)

Problemă. Un vector cu n zerouri și m actualizări „adaugă val la toate elementele de la l la r". La final, afișează vectorul. Fiecare actualizare naivă costă O(n); în total O(n·m).

Idee. Nu modificăm elementele, ci ținem **diferențele** `d[i]` . Adaugi `val` pe [l, r] înseamnă: de la poziția l în sus valoarea crește cu `val` (`d[l] += val`) și după r crește înapoi cu `−val` (`d[r+1] −= val`). La final, `a[i] = d[1] + … + d[i]` (sumele parțiale ale lui `d`).



Diferențe

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

long long d[100005];                        // vectorul de diferențe

int main()
{
    int n, m;
    cin >> n >> m;
    while (m--)
    {
        int l, r;
        long long val;
        cin >> l >> r >> val;
        d[l] += val;                        // de la l în sus se adaugă val
        d[r + 1] -= val;                    // după r se anulează
    }
    long long a = 0;                        // valoarea curentă = suma diferențelor până aici
    for (int i = 1; i <= n; i++)
    {
        a += d[i];
        cout << a << ' ';
    }
    return 0;
}
```

i	1	2	3	4	5	6
actualizări care acoperă i	(1,3)	(1,3), (2,4)	(1,3), (2,4)	(2,4), (4,6)	(4,6)	(4,6)
a[i]	2	2 + 5 = 7	7	5 − 1 = 4	−1	−1

DE CE AȘA?

De ce funcționează `d[r+1] -= val` ? Fiecare `d[i]` spune „cu cât diferă `a[i]` de `a[i−1]` ". Adaugi `val` pe [l, r]: `a[l]` se ridică cu `val` față de `a[l−1]` (diferența `d[l]` crește) și `a[r+1]` coboară cu `val` față de `a[r]` (`d[r+1]` scade). Între l și r diferențele rămân neschimbate. **Care e legătura cu sumele parțiale?** Sunt operații inverse: sumele parțiale transformă diferențele în valori; vectorul de diferențe transformă valorile în diferențe. **Dimensiunea:** `d` trebuie să aibă `n + 2` locuri, fiindcă `r + 1` poate fi `n + 1` .

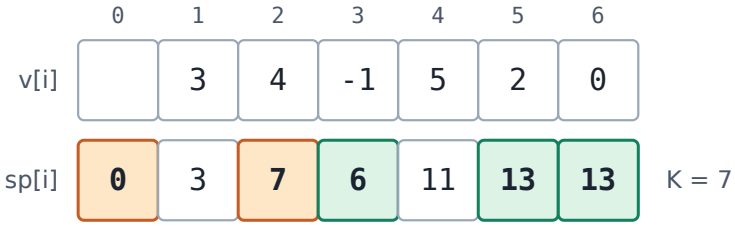
DE ȘTIUT

Merge doar când **toate actualizările vin înaintea întrebărilor** (offline). Dacă actualizările și interogările se amestecă („adaugă pe [l,r]", apoi „cât este a[i]?", apoi altă actualizare), ai nevoie de un arbore indexat binar (lecția 7).

4. Câte subsecvențe au suma exact K?

Problemă. Un vector de n numere (pot fi și negative) și un număr K. Câte subsecvențe `[i+1 .. j]` au suma exact K?

Idee. Suma `v[i+1] + … + v[j]` este `sp[j] − sp[i]`. Vrem `sp[j] − sp[i] = K`, adică `sp[i] = sp[j] − K`. Pentru fiecare j, ne întrebăm: **câte prefixe anterioare** au suma parțială `sp[j] − K`? Reținem într-un `map` de câte ori a apărut fiecare sumă parțială.



sp[0]=0 → sp[2]=7 (secvența 3 4) · sp[3]=6 → sp[5]=13 și sp[6]=13 (secvențele 5 2 și 5 2 0)

Perechi de sume parțiale

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

int main()
{
    int n;
    long long K;
    cin >> n >> K;
    map<long long, int> aparitii;          // de câte ori a apărut fiecare sumă parțială
    aparitii[0] = 1;                       // suma parțială a prefixului gol
    long long sp = 0, rez = 0;
    for (int i = 1; i <= n; i++)
    {
        long long x;
        cin >> x;
        sp += x;
        auto it = aparitii.find(sp - K);  // câte prefixe anterioare au suma sp - K
        if (it != aparitii.end()) rez += it->second;
        aparitii[sp]++;
    }
    cout << rez;
    return 0;
}
```

j	v[j]	sp[j]	sp[j] − 7	prefixe cu această sumă	rez
1	3	3	−4	0	0
2	4	7	0	1 (sp[0])	1
3	−1	6	−1	0	1
4	5	11	4	0	1
5	2	13	6	1 (sp[3])	2

j	v[j]	sp[j]	sp[j] − 7	prefixe cu această sumă	rez
6	0	13	6	1 (sp[3])	3

Cele trei subsecvențe sunt 3 4 · 5 2 · 5 2 0.

> **DE CE AȘA?**
>
> **De ce `aparitii[0] = 1` la început?** Subsecvențele care pornesc de la primul element corespund prefixului **gol** (suma 0). Fără această inițializare le-ai pierde: aici, subsecvența 3 4 are `sp[2] − K = 0` și trebuie să găsească prefixul gol. **De ce inserăm `sp` în map abia după căutare?** Pentru K = 0, `sp − K = sp`: dacă am insera întâi, fiecare prefix s-ar „număra" cu el însuși. **De ce nu fereastra glisantă?** Pentru că numerele pot fi negative (lecția 2); sumele parțiale + `map` funcționează oricum, în O(n log n).

Aceeași schemă, altă proprietate. Câte subsecvențe au suma **divizibilă cu k**? Suma `sp[j] − sp[i]` este divizibilă cu k exact când `sp[j]` și `sp[i]` dau **același rest** la împărțirea cu k. Numeri, pentru fiecare rest, câte prefixe îl au: dacă un rest apare `c` ori, el aduce `c·(c−1)/2` perechi (atenție la resturile negative în C++: `((sp % k) + k) % k`).

5. Care tehnică pentru ce întrebare?

Întrebarea	Tehnica	Cost
suma pe un interval, fără modificări	sume parțiale	O(n + q)
suma pe un dreptunghi, fără modificări	sume parțiale 2D	O(n·m + q)
„adaugă val pe un interval", apoi citești totul	vector de diferențe	O(n + m)
subsecvențe cu suma K / divizibile cu k	sume parțiale + map / resturi	O(n log n)
modificări și interogări amestecate	arbore indexat binar / arbore de intervale	O(log n) fiecare
minimul pe interval (nu se „scade")	sparse table / arbore de intervale	altă lecție

> **DE ȘTIUT**
>
> Sumele parțiale merg pentru operații **inversabile** (adunarea se anulează prin scădere). Pentru minim sau maxim nu poți „scădea" un prefix, de aceea pentru acestea ai nevoie de alte structuri.

6. Greșeli frecvente

Greșeala	Ce se întâmplă	Corect
`sp[r] - sp[l]` în loc de `sp[r] - sp[l-1]`	pierzi `v[l]`	`l-1`
`sp` în `int`	depășire la sume mari	`long long`
`d` cu `n` locuri	`d[r+1]` iese din vector pentru r = n	`n + 2` locuri
la 2D uiți `+ sp[x1-1][y1-1]`	rezultat prea mic	formula completă, cu 4 termeni
`aparitii[0]` neinițializat	pierzi subsecvențele de la început	`aparitii[0] = 1`
`aparitii[sp]++` înainte de căutare	greșit pentru K = 0	întâi căutarea, apoi inserarea
rest negativ: `sp % k` cu `sp < 0`	rest negativ, perechi pierdute	`((sp % k) + k) % k`

7. La olimpiadă

„Sume pe dreptunghiuri", „actualizări pe intervale" și „numără subsecvențele cu proprietatea P" apar în aproape fiecare concurs. Fraza-cheie: *„q întrebări"* cu q și n mari: dacă fiecare întrebare nu poate costa O(n), ai nevoie de precalculare. Subsecvențe cu proprietate pe sume: gândește-te la `sp[j] - sp[i]`.

8. Exersează pe hârtie

1. Pentru v = 4 1 3 2 5, calculează `sp` și suma elementelor 3..5.

2. Pentru matricea 1 2 3 / 4 5 6 / 7 8 9, calculează `sp` (3×3) și suma dreptunghiului (2,2)–(3,3) cu formula din lecție.

3. Ce memorie ocupă `long long sp[1001][1001]`? Încape în 64 MB?

4. n = 5, actualizările: `[1,5] +1` și `[2,3] +2`. Completează vectorul de diferențe și vectorul final.

5. Câte subsecvențe cu suma 5 are v = 2 3 0 2 3? Listează-le.

6. Câte subsecvențe au suma divizibilă cu 3 în v = 3 1 4 1 5 9 2 6? (Indiciu: resturile sumelor parțiale.)

7. De ce în programul din secțiunea 4 este nevoie de `aparitii[0] = 1`? Dă un exemplu mic în care răspunsul ar fi greșit fără el.

8. Pentru n = 10^6 și valori până la 10^9, ce tip alegi pentru sumele parțiale? De ce?

9. Poți folosi vectorul de diferențe dacă actualizările și întrebările sunt amestecate? Justifică.

10. Formulează, prin analogie cu cazul 1D, cum ai adăuga `val` pe un dreptunghi (x1,y1)–(x2,y2) cu diferențe 2D (indiciu: 4 modificări în colțuri).

9. Mini-grile

1. `sp[r] − sp[l−1]` reprezintă: a) suma elementelor de la l la r b) elementul l c) maximul pe [l, r] d) numărul de elemente

2. Pentru 2D, câți termeni are formula sumei unui dreptunghi? a) 1 b) 2 c) 3 d) 4

3. Adaugi `val` pe [l, r] cu vector de diferențe. Ce faci? a) `d[l] += val; d[r] -= val;` b) `d[l] += val; d[r+1] -= val;` c) `d[l..r] += val;` d) `d[r] += val;`

4. Câte subsecvențe cu suma 0 are v = 1 −1 1 −1? (Perechile cu același `sp`.) a) 2 b) 3 c) 4 d) 5

10. Mai departe (pentru cine vrea mai mult)

- **XOR pe prefixe:** `px[i] = v[1] ^ … ^ v[i]` și `v[l] ^ … ^ v[r] = px[r] ^ px[l−1]` (XOR se anulează singur).
- **Sume parțiale pe 3 dimensiuni** (includere–excludere cu 8 termeni).
- **Difference array 2D** pentru actualizări pe dreptunghiuri.
- **Sparse table** pentru minim pe interval în O(1) după O(n log n).
- **Offline:** când toate întrebările se cunosc dinainte, poți reordona și combina tehnici.

Rezumat

- Precalculezi o singură dată (sume parțiale), apoi răspunzi la fiecare întrebare în O(1).
- 2D: `sp[i][j] = a + sp[i−1][j] + sp[i][j−1] − sp[i−1][j−1]`; interogarea are 4 termeni (includere–excludere).
- Vectorul de diferențe: `d[l] += v; d[r+1] −= v;` apoi sume parțiale: actualizări pe interval în O(1).
- Subsecvențe cu suma K: `sp[j] − sp[i] = K` și un `map` cu apariții (cu `aparitii[0] = 1`).
- Merge doar pentru operații inversabile și, la diferențe, doar când actualizările preced întrebările.

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 **Static Range Sum Queries** — ușoară
  sume parțiale 1D
  https://cses.fi/problemset/task/1646
- [ ] CSES **Forest Queries** — medie
  sume parțiale 2D, exact ca în lecție
  https://cses.fi/problemset/task/1652
- [ ] CSES **Subarray Sums II** — medie
  subsecvențe cu suma x: sume parțiale + map
  https://cses.fi/problemset/task/1661

- [ ] CSES **Subarray Divisibility** — medie
  subsecvențe cu suma divizibilă cu n: resturile sumelor parțiale
  https://cses.fi/problemset/task/1662

Temă (concurs)

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

- [ ] CSES **Range Update Queries** — dificilă
  actualizări și interogări amestecate: vezi lecția cu arbori indexați (Fenwick)
  https://cses.fi/problemset/task/1651

- [ ] CSES **Static Range Minimum Queries** — dificilă
  minimul pe interval: ce nu merge cu sume parțiale?
  https://cses.fi/problemset/task/1647

Citește și exersează în plus

- CP-Algorithms — Prefix sums (și 2D) — căutare „prefix sum" / „difference array": idei și variante (engleză) https://cp-algorithms.com/
- USACO Guide — Prefix Sums — sume parțiale 1D și 2D, cu probleme de concurs (engleză) https://usaco.guide/silver/prefix-sums
- USACO Guide — Introduction to hashing și map — pagina principală a ghidului: caută „Prefix Sums" și „Sorting & Searching" https://usaco.guide/

Răspunsuri

Exersează pe hârtie

1. sp = 0 4 5 8 10 15; suma elementelor 3..5 este `sp[5] − sp[2] = 15 − 5 = 10` (3 + 2 + 5).

2. sp (rând cu rând): 1 3 6 / 5 12 21 / 12 27 45. Dreptunghiul (2,2)–(3,3): `sp[3][3] − sp[1][3] − sp[3][1] + sp[1][1] = 45 − 6 − 12 + 1 = 28` (5 + 6 + 8 + 9).

3. 1 001 · 1 001 · 8 octeți ≈ 8 MB: încape.

4. d: d[1] = 1, d[2] = 2, d[3] = 0, d[4] = −2, d[5] = 0, d[6] = −1 (indicii 1..6; d[6] vine de la `[1,5]` cu `r+1 = 6`). Vectorul final: 1 3 3 1 1.

5. 5 subsecvențe: (2 3) de la început, (2 3 0), (3 0 2), (0 2 3) și (2 3) de la sfârșit. Verificare cu sumele parțiale 0 2 5 5 7 10: perechile cu diferența 5 sunt (0→5) de două ori [prefixele de pe pozițiile 2 și 3], (2→7) o dată și (5→10) de două ori.

6. Resturile sumelor parțiale 0 3 4 8 9 14 23 25 31 mod 3 sunt 0 0 1 2 0 2 2 1 1: restul 0 apare de 3 ori (3 perechi), restul 1 de 3 ori (3 perechi), restul 2 de 3 ori (3 perechi): **9** subsecvențe.

7. Fără `aparitii[0] = 1` nu se numără subsecvențele care încep la primul element. Exemplu: v = 5, K = 5: `sp[1] − K = 0` trebuie să găsească prefixul gol; răspunsul corect este 1, nu 0.

8. `long long`: suma poate ajunge la $10^6 \cdot 10^9 = 10^{15}$, mult peste $2{,}1 \cdot 10^9$.

9. Nu: vectorul de diferențe produce valorile abia după ce se fac toate actualizările (sumele parțiale ale lui `d`); la întrebări amestecate cu actualizări ai nevoie de un arbore indexat binar sau de un arbore de intervale.

10. `d[x1][y1] += val; d[x1][y2+1] −= val; d[x2+1][y1] −= val; d[x2+1][y2+1] += val;` apoi sume parțiale 2D pe `d`.

Mini-grile

1. a). **2.** d) 4. **3.** b). **4.** c) 4 (sumele parțiale sunt 0 1 0 1 0: valoarea 0 apare de 3 ori → 3 perechi, valoarea 1 de 2 ori → 1 pereche; total 4 subsecvențe cu suma 0).