INFORMATICĂ · CLASA A X-A · TABLOURI UNIDIMENSIONALE · LECȚIA 6

Vectori caracteristici, vectori de frecvență și Ciurul lui Eratostene

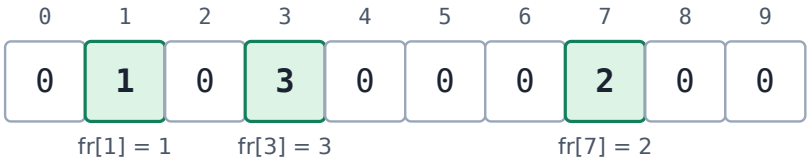
Durată: 60 min Nivel: toate profilurile Înainte: lecțiile x-02

CE VEI ȘTI DUPĂ LECȚIE

- Folosești valoarea unui element ca indice într-un vector (fr[x]++).
- Numeri aparițiile cifrelor / valorilor mici fără să sortezi.
- Afișezi valorile distincte în ordine crescătoare (sortare prin numărare).
- Generezi toate numerele prime până la N cu Ciurul lui Eratostene.
- Recunoști problemele de Bac III.3 care se rezolvă eficient cu vector de frecvență.

1. Ideea: valoarea devine indice

Avem multe numere, dar **mici** (de exemplu cifre 0–9, sau numere până la 1000). În loc să le memorăm, ținem un vector `fr` în care `fr[v]` = de câte ori a apărut valoarea `v`. Ca la un sondaj: nu păstrezi lista tuturor votanților, ci doar câte voturi are fiecare candidat.



indicele este valoarea citită; conținutul este de câte ori a apărut

Frecvențe

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

int fr[1001], n;           // global → inițializat cu 0

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        int x;
        cin >> x;
        fr[x]++;           // încă o apariție a valorii x
    }
    int best = 0;
    for (int v = 1; v <= 1000; v++)
        if (fr[v] > fr[best]) best = v;
    cout << best;          // valoarea cu cele mai multe apariții (cea mai mică la
egalitate)
    return 0;
}
```

Pentru numerele `3, 7, 3, 1, 7, 3`: `fr[1] = 1`, `fr[3] = 3`, `fr[7] = 2`, restul 0.

x citit	operația	fr nenule
3	`fr[3]++`	fr[3] = 1
7	`fr[7]++`	fr[3] = 1, fr[7] = 1
3	`fr[3]++`	fr[3] = 2, fr[7] = 1
1	`fr[1]++`	fr[1] = 1, fr[3] = 2, fr[7] = 1
7	`fr[7]++`	fr[1] = 1, fr[3] = 2, fr[7] = 2
3	`fr[3]++`	fr[1] = 1, **fr[3] = 3**, fr[7] = 2

Tip de vector	Ce reține	Exemplu
caracteristic	`1` dacă valoarea a apărut, `0` altfel	`apare[x] = 1;`
de frecvență	de câte ori a apărut	`fr[x]++;`

DE CE AȘA?

De ce nu căutăm `x` printre valorile deja citite? Pentru fiecare număr nou ar trebui să parcurgem tot ce s-a citit ($O(n^2)$ în total). Cu `fr[x]++` ajungem direct la „căsuța" lui x, într-un singur pas: O(1). Schimbăm **memorie** (un vector de mărimea valorilor) pe **timp**.

2. Ce poți face rapid cu un vector de frecvență

```
// valorile distincte, crescător (fără sortare!)
for (int v = 0; v <= VMAX; v++)
    if (fr[v] > 0) cout << v << ' ';

// toate valorile, crescător (sortare prin numărare)
for (int v = 0; v <= VMAX; v++)
    for (int k = 1; k <= fr[v]; k++) cout << v << ' ';

// valoarea care apare de cele mai multe ori
int best = 0;
for (int v = 1; v <= VMAX; v++)
    if (fr[v] > fr[best]) best = v;
```

LA BAC

Problemă clasică III.3: „Fișierul conține cel mult 10^6 numere naturale din intervalul [10, 99]. Afișați numerele de două cifre care **nu** apar în fișier." Soluția eficientă: un vector de frecvență cu 100 de poziții, o singură parcurgere a fișierului, apoi o parcurgere a vectorului. Memorie: 100 de întregi, oricât de mare ar fi fișierul.

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

bool apare[100];

int main()
{
    ifstream fin("bac.txt");
    int x;
    while (fin >> x)
        apare[x] = true;
    for (int v = 10; v <= 99; v++)
        if (!apare[v])
            cout << v << ' ';
    return 0;
}
```

DE CE AȘA?

De ce vector caracteristic aici? Întrebarea este doar „a apărut sau nu?", deci nu ne interesează de câte ori. `bool` ocupă mai puțin și exprimă mai clar ideea.

Cifrele unui număr: vector de frecvență local

Două numere au **aceleași cifre** (eventual în altă ordine) dacă au aceeași frecvență pentru fiecare cifră:

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

int main()
{
    int a, b;
    cin >> a >> b;
    int f1[10] = {0}, f2[10] = {0};
    while (a > 0) { f1[a % 10]++; a /= 10; }
    while (b > 0) { f2[b % 10]++; b /= 10; }
    bool aceleasi = true;
    for (int c = 0; c <= 9; c++)
        if (f1[c] != f2[c]) aceleasi = false;
    cout << (aceleasi ? "DA" : "NU");
    return 0;
}
```

Cel mai mare număr format cu cifrele lui n: parcurgi `fr` de la 9 la 0 și „lipești” fiecare cifră de `fr[c]` ori.

> **ATENȚIE — GREȘELI FRECVENTE**
>
> Vectorul de frecvență merge doar dacă valorile sunt **mici** (cel mult câteva milioane). Pentru valori până la 10^9 nu poți declara `fr[1000000001]` : ar ocupa 4 GB. Atunci folosești sortarea.

3. Ciurul lui Eratostene

Vrem toate numerele prime ≤ N (N până la câteva milioane). În loc să testăm fiecare număr, **tăiem multiplii**:

1. presupunem că toate numerele de la 2 la N sunt prime;
2. luăm cel mai mic număr netăiat `i` (e prim) și tăiem toți multiplii lui;
3. repetăm până la `i · i > N`.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

e: prime; portocaliu: multipli de 2; albastru: de 3; mov: de 5; roșu: de 7 (

Ciurul pentru N = 50

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

const int NMAX = 1000000;
bool ciur[NMAX + 1];     // ciur[x] = true  →  x NU este prim („tăiat”)

int main()
{
    int N, cnt = 0;
    cin >> N;
    ciur[0] = ciur[1] = true;
    for (int i = 2; i * i <= N; i++)
        if (!ciur[i])
            for (int j = i * i; j <= N; j += i)
                ciur[j] = true;
    for (int x = 2; x <= N; x++)
        if (!ciur[x]) cnt++;
    cout << cnt;         // câte numere prime sunt ≤ N
    return 0;
}
```

i	prim?	taie începând de la	taie
2	da	4	4, 6, 8, 10, …
3	da	9	9, 12, 15, … (6 era tăiat)
4	nu (tăiat de 2)	—	—
5	da	25	25, 30, 35, …
6	nu	—	— (`6·6 > 30` : stop)

SFAT

Începem tăierea de la `i * i`, pentru că multiplii mai mici (2i, 3i, …) au fost deja tăiați de numere prime mai mici. Complexitatea este aproximativ O(N log log N), practic liniară.

DE CE AȘA?

De ce ne oprim la `i * i <= N`? Un număr compus ≤ N are un divizor prim ≤ √N, deci a fost tăiat sigur de unul dintre primii până la √N. Același argument ca la verificarea unui singur număr prim cu `d * d <= n`.

Ciurul merită când ai de verificat **multe** numere (de exemplu 10^5 numere, toate ≤ 10^6): îl construiești o dată, apoi fiecare verificare e instantanee.

4. Greșeli frecvente

Greșeala	Efect	Corect
`fr` prea mic (valori până la 1000, `fr[100]`)	în afara vectorului	`fr[1001]`

Greșeala	Efect	Corect
vector local de 10^6	blocarea programului	vector global
ciur: `j` pornește de la `2*i`	corect, dar mai lent	`i * i`
`i * i` cu `i` de tip `int` la N foarte mare	depășire	`long long`
uiți să marchezi 0 și 1 ca neprime	numeri 0 și 1 ca prime	`ciur[0] = ciur[1] = true`

5. La BAC

> **LA BAC**
>
> Vectorul de frecvență/caracteristic este soluția standard la III.3 când valorile sunt mici (cifre, numere de 2–3 cifre) și enunțul cere „elementele care nu apar", „care apare de cele mai multe ori" sau „valorile distincte". Justificarea eficienței: o parcurgere, memorie constantă (nu depinde de numărul de valori din fișier).

6. Exersează pe hârtie

Răspunsurile sunt la final: încearcă întâi singur.

1. Ce valoare are `fr[3]` după citirea șirului `3, 7, 3, 1, 7, 3`?

2. Se citesc `n` cifre. Afișează cifrele care apar de un număr impar de ori.

3. Se citesc `n` numere ≤ 1000. Afișează valoarea care apare cel mai des (cea mai mică, la egalitate).

4. Câtă memorie ocupă `int fr[1000000001]`? Ce faci dacă valorile sunt până la 10^9?

5. Care sunt numerele prime ≤ 30? Câte sunt ≤ 100?

6. Numerele `1233` și `3123` au aceleași cifre? Dar `112` și `122`?

7. Fișierul conține numere de două cifre. Scrie secvența care afișează câte numere **distincte** conține.

7. Mai departe (pentru cine vrea mai mult)

Sortare prin numărare. Secțiunea 2 conține deja o sortare O(n + V) fără nicio comparație: pentru valori mici bate sortarea clasică. Este baza algoritmilor de tip *radix sort*, folosiți pentru milioane de numere.

Ciur liniar și ciur cu factori. Dacă păstrezi în `cmf[x]` cel mai mic factor prim al lui `x`, poți descompune orice număr ≤ N în factori primi în O(log N). Este tehnica folosită la problemele de concurs cu mii de descompuneri.

Ciur segmentat. Pentru a găsi prime în intervalul [a, b] cu b ≈ 10^{12} și `b - a ≤ 10^6`, folosești primele până la 10^6 (√b) ca să „ciuruiești" doar segmentul. Memoria rămâne mică.

Tabele hash. Când valorile sunt prea mari pentru un vector de frecvență, `unordered_map<int,int> fr;` face același lucru: `fr[x]++`. Este vectorul de frecvență „adaptat" pentru orice valori; vei folosi `map` / `unordered_map` în clasa a XI-a.

Rezumat

- `fr[x]++`: numeri fără să sortezi; merge pentru valori mici.
- Valori distincte / sortare: parcurgi `fr` de la mic la mare.
- Vector caracteristic (`bool`) pentru „a apărut / nu a apărut”.
- Ciurul: tai multiplii lui `i` începând de la `i*i`, pentru `i*i ≤ N`.

Probleme de lucru (pbinfo)

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

- [ ] `#244` **cifreord** — ușoară
  Cifrele în ordine crescătoare: fr[0..9].
  https://www.pbinfo.ro/probleme/244/cifreord
- [ ] `#267` **unice** — ușoară
  https://www.pbinfo.ro/probleme/267/unice
- [ ] `#301` **frecventa1** — medie
  https://www.pbinfo.ro/probleme/301/frecventa1
- [ ] `#303` **eratostene** — medie
  Ciurul o singură dată, apoi răspunsuri în O(1).
  https://www.pbinfo.ro/probleme/303/eratostene

Temă (pbinfo)

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

- [ ] `#187` **ciffrecv** — ușoară
  https://www.pbinfo.ro/probleme/187/ciffrecv
- [ ] `#264` **maxcif** — ușoară
  https://www.pbinfo.ro/probleme/264/maxcif
- [ ] `#239` **nrlipsa** — medie
  https://www.pbinfo.ro/probleme/239/nrlipsa
- [ ] `#276` **pareimpare** — medie
  https://www.pbinfo.ro/probleme/276/pareimpare
- [ ] `#529` **cifre4** — medie
  https://www.pbinfo.ro/probleme/529/cifre4
- [ ] `#302` **cautaprim** — medie
  https://www.pbinfo.ro/probleme/302/cautaprim

- [ ] #1927 **bitsort** — medie
  https://www.pbinfo.ro/probleme/1927/bitsort

Pentru cei care vor mai mult

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

- [ ] #809 **nrlipsa1** — medie
  https://www.pbinfo.ro/probleme/809/nrlipsa1
- [ ] #797 **subnumar** — medie
  https://www.pbinfo.ro/probleme/797/subnumar

Grile din subiectele de BAC (pbinfo)

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

- [ ] #1037 **Exercițiul #1037**
  a[i] = i·i − i: câte componente satisfac o condiție
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1037
- [ ] #1039 **Exercițiul #1039**
  efectul unor modificări succesive pe un vector
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1039
- [ ] #1040 **Exercițiul #1040**
  prima și ultima poziție a unui element
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1040
- [ ] #415 **Exercițiul #415**
  cea mai mare dintre valorile strict negative (Bac 2009)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=415

Citește și exersează în plus

- Vectori caracteristici și de frecvență https://www.pbinfo.ro/articole/5617/vectori-caracteristici-si-de-frecventa
- Ciurul lui Eratostene https://www.pbinfo.ro/articole/2540/ciurul-lui-eratostene
- VisuAlgo — Sieve of Eratosthenes (Math) — caută secțiunea de matematică; animație pentru ciur https://visualgo.net/en
- CSES — Missing Number / Distinct Numbers — probleme simple de frecvență și caracteristic, cu teste mari https://cses.fi/problemset/task/1083
- Algopedia — Ciurul lui Eratostene — variante optimizate ale ciurului, în română https://algopedia.ro/

Răspunsuri

Exersează pe hârtie

1. `3`.

2.

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

int fr[10], n;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        int c;
        cin >> c;
        fr[c]++;
    }
    for (int c = 0; c <= 9; c++)
        if (fr[c] % 2 == 1)
            cout << c << ' ';
    return 0;
}
```

3. Codul din secțiunea 1 (`best` cu `fr[v] > fr[best]`): la egalitate rămâne valoarea mai mică, pentru că `>` este strict.

4. Aproximativ 4 GB (10^9 elemente de 4 octeți), mult peste limita obișnuită. Pentru valori mari folosești sortarea sau `unordered_map`.

5. Primele 10 prime ≤ 30: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29. Sunt 25 de prime ≤ 100.

6. `1233` și `3123`: da (cifrele 1, 2, 3, 3). `112` și `122`: nu (au frecvențe diferite pentru 1 și 2).

7.

```cpp
int distincte = 0;
for (int v = 10; v <= 99; v++)
    if (apare[v]) distincte++;
```