INFORMATICĂ · CLASA A X-A · TABLOURI UNIDIMENSIONALE · LECȚIA 3

Vectori: ștergeri, inserări și verificarea proprietăților

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

CE VEI ȘTI DUPĂ LECȚIE

- Ștergi un element de pe o poziție dată (deplasând elementele spre stânga).
- Inserezi o valoare pe o poziție dată (deplasând elementele spre dreapta).
- Ștergi toate elementele cu o proprietate, eficient, într-o singură parcurgere.
- Verifici proprietăți de tip „toate”, „există”, „ordonat”, „munte”.

1. Ștergerea elementului de pe poziția k

Imaginează-ți un rând de oameni. Dacă cineva pleacă din mijloc, ceilalți se strâng: fiecare din dreapta face un pas spre stânga. La vector, **elementele de după poziția `k` se mută o poziție la stânga**, apoi `n` scade.



Ștergere

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

int a[1001], n, k;

int main()
{
    cin >> n >> k;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = k; i < n; i++)
        a[i] = a[i + 1];
    n--;
    for (int i = 1; i <= n; i++)
        cout << a[i] << ' ';
    return 0;
}
```

Intrare / ieșire:

Intrare	Ieșire
`5 2` / `4 7 2 9 5`	`4 2 9 5`
`3 3` / `1 2 3`	`1 2` (se șterge ultimul: bucla nu se execută)

DE CE AȘA?

De ce bucla merge până la `i < n` și nu `i <= n`? Ultima asignare este `a[n-1] = a[n]`. Dacă ar merge până la `n`, am citi `a[n + 1]`, în afara datelor. Iar `n--` la final face ca ultima căsuță (care acum duplică penultimul element) să nu mai conteze.

2. Inserarea valorii x pe poziția k

Elementele de la `k` încolo se mută **o poziție la dreapta**, începând **de la capăt** (altfel le suprascriem):



Inserare

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

int a[1001], n, k, x;

int main()
{
    cin >> n >> k >> x;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = n; i >= k; i--)
        a[i + 1] = a[i];
    a[k] = x;
    n++;
    for (int i = 1; i <= n; i++)
        cout << a[i] << ' ';
    return 0;
}
```

ATENȚIE — GREȘELI FRECVENTE

La inserare, bucla merge **de la dreapta la stânga**. Dacă ai merge de la stânga, ai copia `a[k]` peste tot restul vectorului. Și nu uita că vectorul trebuie declarat cu loc pentru elementul nou.

DE CE AȘA?

De ce dreapta → stânga la inserare, dar stânga → dreapta la ștergere? Regula: muți mereu **în direcția în care nu ai nevoie de ce acoperi**. La ștergere, `a[k]` e cel șters, deci poate fi acoperit; la inserare, tot ce e în dreapta trebuie păstrat, deci începi din capătul liber. Dacă ai dubii, desenează 3 elemente și mută-le.

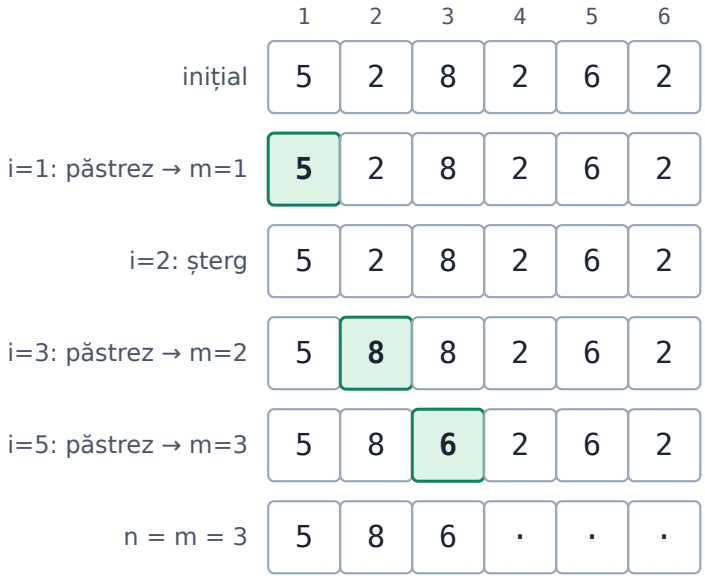
Urmărire pentru `(1, 2, 3)`, `k = 1`, `x = 9`, mergând **greșit** de la stânga:

i	operația	vector
1	`a[2] = a[1]`	(1, **1**, 3)
2	`a[3] = a[2]`	(1, 1, **1**)
—	`a[1] = 9`	(9, 1, 1): am pierdut 2 și 3

3. Ștergerea tuturor elementelor cu o proprietate

Varianta lentă: pentru fiecare element de șters, apelezi ștergerea de mai sus → $O(n^2)$.

Varianta eficientă (doi indici): păstrăm doar elementele „bune”, pe care le mutăm în față.



Doi indici

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

int a[1001], n;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    int m = 0;                          // câte elemente păstrăm
    for (int i = 1; i <= n; i++)
        if (a[i] != 2)                  // condiția de PĂSTRARE
            a[++m] = a[i];
    n = m;
    for (int i = 1; i <= n; i++)
        cout << a[i] << ' ';
    return 0;
}
```

i	a[i]	se păstrează?	m	vector (primele m)
1	5	da	1	(5)
2	2	nu	1	(5)
3	8	da	2	(5, 8)
4	2	nu	2	(5, 8)
5	6	da	3	(5, 8, 6)
6	2	nu	3	(5, 8, 6)

DE CE AȘA?

De ce funcționează pe același vector, fără vector nou? `m ≤ i` mereu: scriem pe poziția `m` ceva citit de la `i ≥ m`, deci nu stricăm niciodată un element pe care îl mai avem de citit. Doi „degete”: `i` citește, `m` scrie. Timpul e O(n) în loc de O(n^2).

SFAT

Același tipar rezolvă și inserările multiple dacă construiești rezultatul într-un **vector nou** `b`: pentru fiecare `a[i]`, pui în `b` elementul și, dacă e cazul, valoarea inserată.

```
// după fiecare element par inserăm dublul său
int m = 0;
for (int i = 1; i <= n; i++)
{
    b[++m] = a[i];
    if (a[i] % 2 == 0) b[++m] = 2 * a[i];
}
```

4. Verificarea proprietăților

„Toate elementele au proprietatea P”

Presupunem că da, căutăm un contraexemplu:

```
bool toate = true;
for (int i = 1; i <= n && toate; i++)
    if (a[i] % 2 != 0) toate = false;
```

„Există cel puțin un element cu P”

```
bool exista = false;
for (int i = 1; i <= n && !exista; i++)
    if (a[i] % 2 == 1) exista = true;
```

DE CE AȘA?

De ce „toate” începe cu `true`, iar „există” cu `false`? „Toate” se dărâmă printr-un singur contra-exemplu (deci pornim optimiști). „Există” se confirmă printr-un singur exemplu (deci pornim pesimiști). Condiția `&& toate` (respectiv `&& !exista`) oprește bucla imediat ce răspunsul e cunoscut.

Vector ordonat crescător

```
bool cresc = true;
for (int i = 1; i < n; i++)             // atenție: până la n-1, comparăm cu a[i+1]
    if (a[i] > a[i + 1]) cresc = false;
```

Toate elementele distincte

```
bool distincte = true;
for (int i = 1; i < n; i++)
    for (int j = i + 1; j <= n; j++)
        if (a[i] == a[j]) distincte = false;
```

(O(n²): mai eficient cu sortare sau vector de frecvență, în lecțiile următoare.)

Șir „munte”



vârful nu e primul și nici ultimul element

Șir munte

Crește strict până la un vârf, apoi scade strict (vârful nu e la capete):

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

int a[1001], n;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    int i = 1;
    while (i < n && a[i] < a[i + 1]) i++;      // urcăm
    int varf = i;
    while (i < n && a[i] > a[i + 1]) i++;      // coborâm
    cout << ((varf > 1 && varf < n && i == n) ? "DA" : "NU");
    return 0;
}
```

șir	vârful	după coborâre i	rezultat
1 3 5 7 4 2	4	6 = n	DA
1 2 3	3 (= n)	3	NU (vârf la capăt)
1 3 3 2	2	2 ≠ n	NU (platou, nu strict)

5. Greșeli frecvente

Greșeala	Efect	Corect
inserare de la stânga	elemente pierdute	`for (i = n; i >= k; i--)`

Greșeala	Efect	Corect
uiți `n--` / `n++`	afișezi un element în plus/minus	actualizează `n`
ștergi cu bucla `i <= n`	citești `a[n+1]`	`i < n`
ștergi în timp ce parcurgi, fără ajustare	sari elemente	metoda cu doi indici
vector declarat exact `n`, apoi inserare	depășire	declară cu loc în plus

6. La BAC

> **LA BAC**
>
> II.3 cere adesea o secvență de ștergere/inserare („ștergeți elementele de pe pozițiile pare", „inserați după fiecare element par…"). Soluția sigură: desenează 4–5 elemente, fixează direcția mutărilor, apoi scrie bucla. Pentru ștergeri multiple, metoda cu doi indici dă și punctajul de eficiență.

7. Exersează pe hârtie

Răspunsurile sunt la final: încearcă întâi singur.

1. Vectorul `(5, 2, 8, 2, 6, 2)`. Ce conține după secvența de mai jos (primele elemente și valoarea lui `m`)?

```
m = 0;
for (i = 1; i <= 6; i++)
    if (v[i] != 2) v[++m] = v[i];
```

2. Scrie o secvență care inserează valoarea 0 între oricare două elemente consecutive de aceeași paritate.

3. Verifică dacă un vector este palindrom (`a[i] == a[n-i+1]` pentru toți `i`).

4. Din `(4, 7, 2, 9, 5)` se șterge elementul de pe poziția 1. Ce conține vectorul și cât e `n`?

5. Care este complexitatea ștergerii unui singur element? Dar a ștergerii tuturor elementelor pare prin ștergeri repetate, față de metoda cu doi indici?

6. Câte elemente are vectorul după ce inserezi un element pe fiecare poziție impară dintr-un vector cu `n = 6`? (Insertii independente, în vectorul original.)

7. Pentru ce șiruri de lungime 3 se afișează `DA` la programul „munte"?

8. Mai departe (pentru cine vrea mai mult)

`erase` și `insert` din `vector`. În C++ standard: `v.erase(v.begin() + k)` și `v.insert(v.begin() + k, x)` fac exact mutările din lecție, în O(n). Știind ce fac în spate, înțelegi de ce o ștergere repetată în mijloc e scumpă.

Lazy deletion (ștergere „de formă”). Când ai multe ștergeri și puține parcurgeri, nu muți elemente: marchezi ștergerile într-un vector `sters[i] = true` și le ignori la parcurgere. Este tehnica din baze de date și din sisteme de fișiere (coșul de gunoi).

Rotații. Permutarea circulară cu k poziții se poate face în O(n) fără vector auxiliar prin trei inversări: `reverse(a, a+k); reverse(a+k, a+n); reverse(a, a+n);`. Este un truc clasic de interviu.

Eliminarea duplicatelor. Dacă vectorul e ordonat, duplicatele sunt vecine, iar tiparul cu doi indici le elimină într-o parcurgere: `if (i == 1 || a[i] != a[i-1]) a[++m] = a[i];`. Folosit în `unique()` din biblioteca standard.

Rezumat

- Ștergere la poziția k: mută la stânga de la k; inserare: mută la dreapta de la n spre k.
- Ștergere multiplă eficientă: `a[++m] = a[i]` doar pentru elementele păstrate (doi indici, O(n)).
- „Toate” → presupui `true`, cauți contraexemplu; „există” → presupui `false`.
- Verificări cu vecini (`a[i+1]`): bucla merge până la `n - 1`.

Probleme de lucru (pbinfo)

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

- [ ] `#1452` **stergere-element** — ușoară
  https://www.pbinfo.ro/probleme/1452/stergere-element
- [ ] `#158` **inserare** — ușoară
  https://www.pbinfo.ro/probleme/158/inserare
- [ ] `#288` **verifpare** — ușoară
  https://www.pbinfo.ro/probleme/288/verifpare
- [ ] `#500` **verificareordonat** — ușoară
  https://www.pbinfo.ro/probleme/500/verificareordonat

Temă (pbinfo)

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

- [ ] `#1453` **stergere1** — medie
  Ștergerea tuturor elementelor pare — metoda cu doi indici.
  https://www.pbinfo.ro/probleme/1453/stergere1
- [ ] `#163` **stergere** — medie
  https://www.pbinfo.ro/probleme/163/stergere
- [ ] `#159` **inseraredupa** — medie
  https://www.pbinfo.ro/probleme/159/inseraredupa

- [ ] #289 **existaimpare** — ușoară
  https://www.pbinfo.ro/probleme/289/existaimpare
- [ ] #502 **verifdiferite** — ușoară
  https://www.pbinfo.ro/probleme/502/verifdiferite
- [ ] #505 **identice** — ușoară
  https://www.pbinfo.ro/probleme/505/identice
- [ ] #1321 **sir-munte** — ușoară
  https://www.pbinfo.ro/probleme/1321/sir-munte
- [ ] #1320 **ordonat-neordonat** — ușoară
  https://www.pbinfo.ro/probleme/1320/ordonat-neordonat

Pentru cei care vor mai mult

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

- [ ] #162 **permcirc** — medie
  https://www.pbinfo.ro/probleme/162/permcirc
- [ ] #309 **permutare** — medie
  https://www.pbinfo.ro/probleme/309/permutare

Grile din subiectele de BAC (pbinfo)

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

- [ ] #1036 **Exercițiul #1036**
  care secvență afișează corect produsul elementelor pare
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1036
- [ ] #1039 **Exercițiul #1039**
  ce afișează secvența după modificări 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
- [ ] #1038 **Exercițiul #1038**
  care secvență ordonează descrescător un tablou
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1038

Citește și exersează în plus

- Ștergeri și inserări de elemente https://www.pbinfo.ro/articole/5584/stergeri-si-inserari-de-elemente

- Verificarea unor proprietăți https://www.pbinfo.ro/articole/5586/verificarea-unor-proprietati
- Python Tutor pentru C++ — urmărește mutarea elementelor la ștergere și inserare https://pythontutor.com/cpp.html
- cppreference — std::vector (insert, erase) — ce fac gata scrise `insert` și `erase` pentru vectorii standard https://en.cppreference.com/w/cpp/container/vector
- VisuAlgo — List / Array — animație pentru inserare și ștergere (alege Array) https://visualgo.net/en/list

Răspunsuri

Exersează pe hârtie

1. După secvență `m = 3`, iar primele 3 elemente sunt `(5, 8, 6)`. Restul vectorului (4–6) nu a fost șters: conține `2, 6, 2`, dar nu mai contează pentru că `n = m = 3`.

2.

```
int m = 0;
for (int i = 1; i <= n; i++)
{
    b[++m] = a[i];
    if (i < n && a[i] % 2 == a[i + 1] % 2)
        b[++m] = 0;
}
```

Condiția `i < n` evită citirea lui `a[n + 1]`.

3.

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

int a[1001], n;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    bool pal = true;
    for (int i = 1; i <= n / 2 && pal; i++)
        if (a[i] != a[n - i + 1]) pal = false;
    cout << (pal ? "DA" : "NU");
    return 0;
}
```

4. `(7, 2, 9, 5)` și `n = 4` (toate elementele s-au mutat cu o poziție la stânga).

5. O(n) pentru un element. Prin ștergeri repetate: până la n/2 ștergeri de câte O(n) = **O(n²)**; cu doi indici: **O(n)**.

6. Vectorul original are poziții impare 1, 3, 5 (trei poziții), deci **9 elemente** (6 + 3).

7. Doar șirurile cu vârful la mijloc: `a < b > c`, de exemplu `1 5 2`. Șirurile crescătoare, descrescătoare sau cu valori egale dau `NU`.