INFORMATICĂ · CLASA A XI-A · RECURSIVITATE · LECȚIA 4

Recursivitate (2): pe vectori, pe șiruri și algoritmul de umplere (fill)

Durată: 60 min Nivel: mate-info Înainte: lecțiile xi-03, x-08

CE VEI ȘTI DUPĂ LECȚIE

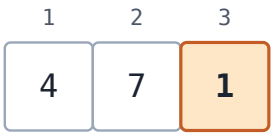
- Scrii funcții recursive care parcurg un vector (sumă, maxim, căutare, afișare inversă).
- Scrii funcții recursive pe șiruri de caractere.
- Folosești algoritmul de umplere (fill) pe matrice: zone, insule, labirinturi.
- Știi să alegi parametrii potriviți pentru apelul recursiv (indice, lungime).

1. Recursivitate pe vectori

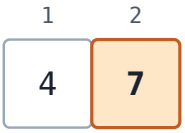
Același gând ca la numere: „suma vectorului de n elemente = suma primelor n − 1 elemente + ultimul”. Problema se micșorează cu un element la fiecare apel.



suma(a, 4) = suma(a, 3) + a[4]



suma(a, 3) = suma(a, 2) + a[3]



suma(a, 2) = suma(a, 1) + a[2]



suma(a, 1) = suma(a, 0) + a[1] = 0 + 4

Suma recursivă

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

int a[101], n;

int suma(int a[], int n)                // suma a[1..n]
{
    if (n == 0) return 0;                   // caz de bază: vectorul vid
    return suma(a, n - 1) + a[n];
}

int maxim(int a[], int n)               // maximul din a[1..n]
{
    if (n == 1) return a[1];                // un singur element
    int m = maxim(a, n - 1);
    return (a[n] > m) ? a[n] : m;
}

bool apare(int a[], int n, int x)       // există x în a[1..n]?
{
    if (n == 0) return false;
    if (a[n] == x) return true;
    return apare(a, n - 1, x);
}

int main()
{
    int x;
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    cin >> x;
    cout << suma(a, n) << ' ' << maxim(a, n) << ' ' << (apare(a, n, x) ? "DA" : "NU");
    return 0;
}
```

DE CE AȘA?

De ce `maxim` are cazul de bază `n == 1` și nu `n == 0`? Maximul unui vector fără elemente nu există. Cel mai mic vector pentru care răspunsul are sens are un element, iar atunci maximul este chiar `a[1]`. **De ce `apare` poate să se oprească înainte de caz de bază?** Dacă `a[n] == x`, răspunsul este deja „da”; nu mai căutăm în rest.

Urmărire pentru `suma(a, 4)` cu `a = (4, 7, 1, 8)`:

apel	n	a[n]	returnează
`suma(a, 4)`	4	8	`suma(a, 3) + 8` = 12 + 8 = **20**
`suma(a, 3)`	3	1	`suma(a, 2) + 1` = 11 + 1 = 12
`suma(a, 2)`	2	7	`suma(a, 1) + 7` = 4 + 7 = 11
`suma(a, 1)`	1	4	`suma(a, 0) + 4` = 0 + 4 = 4
`suma(a, 0)`			0

Intrare / ieșire

Intrare	Ieșire

`5`
`4 7 1 8 3`
`8`
	`23 8 DA`

`5`
`4 7 1 8 3`
`9`
	`23 8 NU`

Afișare în ordine directă și inversă

Ca la cifre: `cout` înainte de apel dă ordinea directă, după apel dă ordinea inversă.

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

int a[101], n;

void afisare(int a[], int i, int n)            // afișează a[i..n]
{
    if (i > n) return;
    cout << a[i] << ' ';                       // înainte de apel: ordine directă
    afisare(a, i + 1, n);
}

void afisareInversa(int a[], int i, int n)
{
    if (i > n) return;
    afisareInversa(a, i + 1, n);
    cout << a[i] << ' ';                       // după apel: ordine inversă
}

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

DE CE AȘA?

De ce aici recursivitatea merge pe indice (`i + 1`) și la sumă pe lungime (`n - 1`)? Sunt două moduri de a micșora problema: fie scoatem ultimul element (rămâne `a[1..n-1]`), fie ne mutăm începutul (rămâne `a[i+1..n]`). Alegem modul care face codul mai simplu pentru problema respectivă.

2. Recursivitate pe șiruri de caractere

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

char s[256];

int nrVocale(char s[], int i)            // vocalele din s[i...]
{
    if (s[i] == '\0') return 0;                              // am ajuns la terminator
    return (strchr("aeiou", s[i]) != NULL) + nrVocale(s, i + 1);
}

int main()
{
    cin >> s;
    cout << nrVocale(s, 0);
    return 0;
}
```

DE CE AȘA?

De ce verificăm `s[i] == '\0'` înaintea lui `strchr`? `strchr("aeiou", '\0')` nu returnează `NULL` (terminatorul face parte din șir) și am număra o „vocală" în plus (am spus asta și la lecția X.11).

3. Algoritmul de umplere (fill)

Avem o hartă (matrice) cu `1` = pământ și `0` = apă. Câte **insule** sunt (zone de 1 vecine pe orizontală sau verticală)? Care este cea mai mare?

	1	2	3	4	5
1	1	1	0	0	0
2	1	0	0	1	1
3	0	0	0	1	0
4	1	0	0	0	0
5	1	1	0	0	1

insule = zone de 1 vecine pe orizontală / verticală: aria 3, 3, 3, 1

Harta

Ideea. Când găsim un `1` nevizitat, am descoperit o insulă nouă. O „scufundăm" complet (marcăm toate celulele ei) cu o funcție recursivă care se extinde către cei 4 vecini.

	1	2	3
1	**2**	1	0
2	1	0	0

1) marcăm (1,1)

	1	2	3
1	**2**	1	0
2	**2**	0	0

2) jos: marcăm (2,1)

	1	2	3
1	**2**	**2**	0
2	**2**	0	0

3) dreapta: marcăm (1,2)

Pașii

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

int a[102][102], n, m;

int fill(int i, int j)                              // întoarce aria insulei care conține
(i, j)
{
    if (i < 1 || i > n || j < 1 || j > m) return 0; // în afara hărții
    if (a[i][j] != 1) return 0;                     // apă sau deja vizitat
    a[i][j] = 2;                                    // marcăm ca vizitat, ÎNAINTE de
apeluri
    return 1 + fill(i - 1, j) + fill(i + 1, j) + fill(i, j - 1) + fill(i, j + 1);
}

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> a[i][j];

    int insule = 0, amax = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            if (a[i][j] == 1)                       // insulă nouă
            {
                insule++;
                int aria = fill(i, j);
                if (aria > amax) amax = aria;
            }
    cout << insule << ' ' << amax;
    return 0;
}
```

DE CE AȘA?

De ce marcăm `a[i][j] = 2` înainte de apelurile recursive? Dacă nu am marca, `fill(1, 1)` ar apela `fill(2, 1)`, care ar apela înapoi `fill(1, 1)` (vecinul de sus), și tot așa la infinit. Marcarea spune „aici am fost deja". **De ce verificăm mai întâi dacă (i, j) e în hartă?** Altfel am citi `a[0][j]` sau `a[n+1][j]`, în afara matricei date. **De ce `1 + suma celor 4 apeluri`?** Celula curentă contribuie cu 1 la arie, iar fiecare vecin adaugă aria părții de insulă pe care o descoperă el.

Urmărire (harta din figură, parcurgere pe linii):

celulă (i, j)	valoare	acțiune	insule	arie
(1, 1)	1	insulă nouă; `fill` marchează (1,1), (2,1), (1,2)	1	3
(1, 2)	2	deja vizitat	1	

celulă (i, j)	valoare	acțiune	insule	arie
(2, 4)	1	insulă nouă: (2,4), (2,5), (3,4)	2	3
(4, 1)	1	insulă nouă: (4,1), (5,1), (5,2)	3	3
(5, 5)	1	insulă nouă: doar (5,5)	4	1

La final: 4 insule, aria maximă 3.

Intrare / ieșire

Intrare	Ieșire
harta din figură (5 × 5)	`4 3`
`1 1` `1`	`1 1`

Variante care se rezolvă la fel:

- **8 vecini** (și pe diagonală): încă 4 apeluri, cu `(i ± 1, j ± 1)`;
- **labirint**: există drum de la start la ieșire? Pornești `fill` din start și verifici dacă ieșirea a fost marcată;
- **numărul de camere** într-un plan de clădire (ziduri = 1, spațiu liber = 0).

> **CAPCANĂ**
>
> Pentru hărți foarte mari (de exemplu 1000 × 1000), recursivitatea poate umple stiva: o insulă de 10^6 celule înseamnă 10^6 cadre imbricate. Atunci se folosește o coadă (algoritmul lui Lee, lecția XI.17) sau o stivă proprie.

4. Greșeli frecvente

Greșeala	Efect	Corect
nu marchezi celula înainte de apeluri	recursie infinită	`a[i][j] = 2;` mai întâi
verifici `a[i][j]` înainte de a verifica marginile	citire în afara matricei	întâi `i`, `j` în hartă
`maxim` cu caz de bază `n == 0` și `return a[0]`	valoare irelevantă	caz de bază `n == 1`
`suma(a, n - 1) + a[n - 1]` pentru vector indexat de la 1	element greșit	potrivește indexarea
uiți să resetezi harta între teste	rezultate amestecate	citește harta din nou
`strchr(..., s[i])` fără oprire la `'\0'`	o vocală în plus	verifică întâi terminatorul

5. La BAC

> **LA BAC**
>
> Subiectul I: subprograme recursive cu vectori (`f(st, dr, v)` , `f(v, n)`): „ce valoare întoarce apelul?" Se rezolvă cu tabelul apelurilor. Subiectul II/III: fill apare rar la Bac, dar este frecvent la olimpiadă; ideea de „marcare ca vizitat" se regăsește la parcurgerea grafurilor (XI.11).

6. Exersează pe hârtie

Răspunsurile sunt la final: încearcă întâi singur.

1. Pentru `a = (2, 5, 1)` , scrie apelurile și valoarea lui `suma(a, 3)` .

2. Care este valoarea lui `maxim(a, 4)` pentru `a = (3, 9, 2, 7)` ? Scrie apelurile.

3. Ce afișează `afisareInversa(a, 1, 3)` pentru `a = (5, 6, 7)` ?

4. Scrie o funcție recursivă `bool palindrom(char s[], int st, int dr)` .

5. Scrie o funcție recursivă care returnează numărul de elemente pare dintr-un vector.

6. Pe harta `1 0 1 / 0 1 0 / 1 0 1` , câte insule sunt cu 4 vecini? Dar cu 8 vecini?

7. De ce în `fill` marcăm `a[i][j] = 2` înainte de apelurile recursive?

8. Câte apeluri ale lui `fill` se fac pentru o insulă formată dintr-o singură celulă (inclusiv primul)?

9. (BAC) Ce întoarce `f(1, 5, v)` pentru `v = (3, 8, 1, 9, 4)` (indexat de la 1)?

```
int f(int st, int dr, int v[]) { if (st == dr) return v[st]; int m = f(st + 1, dr, v);
return v[st] > m ? v[st] : m; }
```

7. Mai departe (pentru cine vrea mai mult)

Flood fill în programe de desen. Unealta „găleată de vopsea" din orice editor de imagini este exact `fill` : pornește de la pixelul pe care ai dat clic și colorează toți pixelii vecini de aceeași culoare.

Număr de insule pe hărți reale. Problema „Number of Islands" este una dintre cele mai frecvente la interviurile de angajare în IT. Variantele ei: insule cu 8 vecini, insulă maximă, „lacuri" (apă înconjurată de pământ).

Backtracking pe matrice. Dacă la `fill` nu marchezi definitiv ci „demarchezi" la întoarcere, obții căutarea tuturor drumurilor dintr-un labirint: un exemplu de backtracking (lecțiile XI.6–XI.8).

De la recursivitate la stivă proprie. Orice funcție recursivă poate fi scrisă iterativ cu o stivă pe care o gestionezi singur (lecția XI.17). Așa se evită depășirea stivei la date mari.

Rezumat

- Pe vectori: recursie pe lungime (`n - 1`) sau pe indice (`i + 1`); cazul de bază: vector vid sau cu un element.
- Ordinea afișării: înainte de apel = directă, după apel = inversă.
- Fill: marchezi celula, apelezi pentru vecini, te oprești la margine sau la celule deja vizitate/apă.

- Aria unei zone: `1 + suma apelurilor pentru vecini`.

Probleme de lucru (pbinfo)

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

- [ ] #835 **afisarevectorrec** — ușoară
  https://www.pbinfo.ro/probleme/835/afisarevectorrec
- [ ] #836 **afisarevectorrec1** — ușoară
  https://www.pbinfo.ro/probleme/836/afisarevectorrec1
- [ ] #1843 **fsumvecrec** — ușoară
  https://www.pbinfo.ro/probleme/1843/fsumvecrec
- [ ] #837 **fill** — ușoară
  Prima problemă de umplere: numărul de zone.
  https://www.pbinfo.ro/probleme/837/fill

Temă (pbinfo)

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

- [ ] #938 **fsumrec** — ușoară
  https://www.pbinfo.ro/probleme/938/fsumrec
- [ ] #1752 **nr-aparitii-rec** — ușoară
  https://www.pbinfo.ro/probleme/1752/nr-aparitii-rec
- [ ] #1654 **nrvocrec** — ușoară
  https://www.pbinfo.ro/probleme/1654/nrvocrec
- [ ] #843 **puteri2** — ușoară
  https://www.pbinfo.ro/probleme/843/puteri2
- [ ] #872 **puteri3** — ușoară
  https://www.pbinfo.ro/probleme/872/puteri3
- [ ] #1511 **fcautarerec** — ușoară
  https://www.pbinfo.ro/probleme/1511/fcautarerec
- [ ] #840 **croco** — medie
  Fill + cea mai mare zonă.
  https://www.pbinfo.ro/probleme/840/croco

Pentru cei care vor mai mult

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

- [ ] #838 **iesire** — medie
  https://www.pbinfo.ro/probleme/838/iesire
- [ ] #1507 **grupuri** — ușoară
  https://www.pbinfo.ro/probleme/1507/grupuri

Grile din subiectele de BAC (pbinfo)

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

- [ ] #628 **Exercițiul #628**
  tablou unidimensional global și subprogram recursiv: ce valoare se obține
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=628
- [ ] #630 **Exercițiul #630**
  tablou unidimensional global: valoarea întoarsă de un subprogram recursiv
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=630
- [ ] #1277 **Exercițiul #1277**
  subprogram recursiv cu un vector ca parametru: valoarea lui f
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1277
- [ ] #1350 **Exercițiul #1350**
  f(st, dr, v[]): valoarea întoarsă pe un vector dat
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1350
- [ ] #1739 **Exercițiul #1739**
  f(s, d, v[]) recursivă pe un vector: valoarea apelului
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1739
- [ ] #616 **Exercițiul #616**
  valorile componentelor unui tablou v și un subprogram recursiv: ce se obține
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=616

Citește și exersează în plus

- Recursivitate https://www.pbinfo.ro/articole/3873/recursivitate
- Algoritmi de umplere https://www.pbinfo.ro/articole/18890/algoritmi-de-umplere
- USACO Guide — Flood Fill — algoritmul de umplere explicat, cu probleme graduale (engleză) https://usaco.guide/silver/flood-fill
- VisuAlgo — Recursion — animații pentru arborele de apeluri https://visualgo.net/en/recursion
- cppreference — Pointer and arrays as parameters — de ce un vector se transmite ca adresă (engleză) https://en.cppreference.com/w/cpp/language/array

Răspunsuri

Exersează pe hârtie

1. `suma(a, 3) = suma(a, 2) + 1`; `suma(a, 2) = suma(a, 1) + 5`; `suma(a, 1) = suma(a, 0) + 2 = 2`. Deci `suma(a, 2) = 7`, iar `suma(a, 3) =` **8**.

2. `maxim(a, 1) = 3`; `maxim(a, 2) = 9`; `maxim(a, 3) = 9`; `maxim(a, 4)` compară `a[4] = 7` cu 9 și returnează **9**.

3. `7 6 5`.

4.

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

char s[256];

bool palindrom(char s[], int st, int dr)
{
    if (st >= dr) return true;                  // 0 sau 1 caractere: palindrom
    if (s[st] != s[dr]) return false;
    return palindrom(s, st + 1, dr - 1);
}

int main()
{
    cin >> s;
    cout << (palindrom(s, 0, strlen(s) - 1) ? "DA" : "NU");
    return 0;
}
```

5.

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

int a[101], n;

int nrPare(int a[], int n)
{
    if (n == 0) return 0;
    return (a[n] % 2 == 0) + nrPare(a, n - 1);
}

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

6. Cu 4 vecini: **5** insule (fiecare `1` e izolat pe orizontală/verticală). Cu 8 vecini: **1** insulă (toate cele cinci celule se leagă prin diagonale de celula din centru).

7. Fără marcare, două celule vecine s-ar apela una pe alta la nesfârșit (`fill(1, 1)` → `fill(2, 1)` → `fill(1, 1)` ...) și stiva s-ar umple.

8. 5 apeluri: cel inițial (care marchează celula) și 4 pentru vecini, toate întorcându-se imediat (fie sunt în afara hărții, fie au valoarea 0).

9. `9` : funcția întoarce maximul din `v[st..dr]` . `f(5,5) = 4` ; `f(4,5) = 9` ; `f(3,5) = 9` ; `f(2,5) = 9` ; `f(1,5)` : `v[1] = 3 < 9` → **9**.
