INFORMATICĂ · CLASA A XI-A · METODA BACKTRACKING · LECȚIA 7

Backtracking (2): aranjamente, combinări, submulțimi, produs cartezian

Durată: 60 min Nivel: mate-info Înainte: lecțiile xi-06

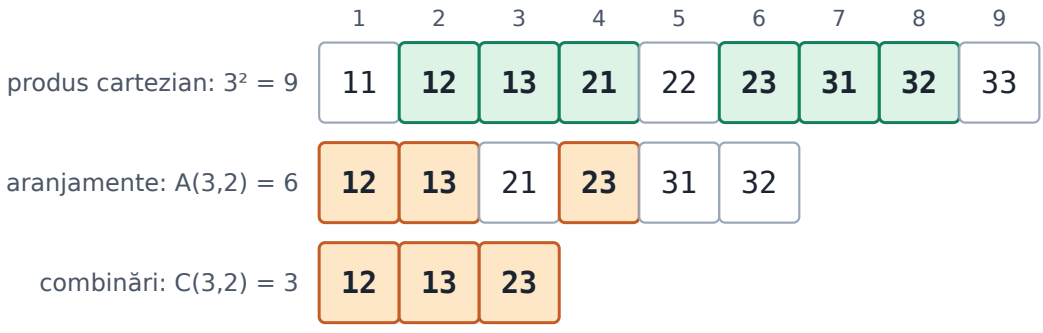
CE VEI ȘTI DUPĂ LECȚIE

- Adaptezi schema de backtracking pentru aranjamente, combinări, submulțimi și produs cartezian.
- Știi formulele de numărare (A_n^k, C_n^k, 2^n, n^k) și le folosești la grile.
- Generezi de mână soluțiile în ordine lexicografică și găsești a k-a soluție.
- Reformulezi o problemă „cu povești” ca una dintre generările standard.

1. Patru generări, o singură schemă

Sunt patru probleme-tip care apar iar și iar, la clasă și la Bac. Toate se rezolvă cu aceeași schemă din lecția anterioară; se schimbă doar **ce valori încercăm** și **când avem soluție**.

Generare	Ce înseamnă	Exemplu (n = 3, K = 2)	Număr de soluții
produs cartezian	K poziții, fiecare din {1..n}, cu repetiții	11 12 13 21 22 ... 33	n^K
aranjamente	K valori distincte, ordinea contează	12 13 21 23 31 32	A(n, K) = n!/(n−K)!
combinări	K valori distincte, ordinea NU contează	12 13 23	C(n, K) = n!/(K!(n−K)!)
submulțimi	orice număr de elemente	{1} {1,2} {1,2,3} {1,3} ...	2^n (cu mulțimea vidă)



verde: fără repetiții; portocaliu: și în ordine crescătoare

Comparație

SFAT

Cum le deosebești într-un enunț: „**contează ordinea?**" (12 ≠ 21 → aranjamente) și „**se pot repeta?**" (11 permis → produs cartezian). Combinările se generează **strict crescător**, ca să nu apară aceeași grupă de două ori (12 și 21).

2. Produs cartezian

Nicio condiție: orice valoare este bună pe orice poziție. `K` este numărul de poziții.

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

int x[20], n, K;

void afisare()
{
    for (int i = 1; i <= K; i++) cout << x[i] << ' ';
    cout << '\n';
}

void back(int k)
{
    for (int v = 1; v <= n; v++)
    {
        x[k] = v;                       // orice valoare e validă
        if (k == K) afisare();
        else back(k + 1);
    }
}

int main()
{
    cin >> n >> K;
    back(1);
    return 0;
}
```

DE CE AȘA?

De ce nu există `valid` ? Produsul cartezian nu impune nicio relație între elemente: fiecare poziție poate fi aleasă independent. Numărul de soluții este n^K (n variante pentru fiecare dintre cele K poziții). Parolele de 4 cifre sunt 10^4.

3. Aranjamente

Ca la permutări (valori distincte), dar soluția are K elemente: `if (k == K)`.

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

int x[20], n, K;

bool valid(int k)
{
    for (int i = 1; i < k; i++)
        if (x[i] == x[k]) return false;
    return true;
}

void back(int k)
{
    for (int v = 1; v <= n; v++)
    {
        x[k] = v;
        if (valid(k))
        {
            if (k == K)
            {
                for (int i = 1; i <= K; i++) cout << x[i] << ' ';
                cout << '\n';
            }
            else back(k + 1);
        }
    }
}

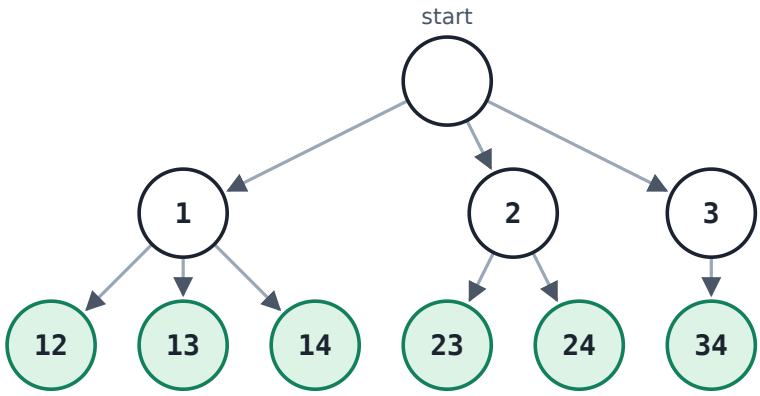
int main()
{
    cin >> n >> K;
    back(1);
    return 0;
}
```

DE CE AȘA?

De ce aranjamentele cu K = n sunt chiar permutări? Dacă luăm toate cele n elemente și le așezăm pe n poziții, fără repetiții, obținem permutările (lecția XI.6). Permutările sunt cazul particular A(n, n) = n!.

4. Combinări

Dacă ordinea nu contează, `{1, 2}` și `{2, 1}` sunt aceeași grupă. Ca să o generăm o singură dată, cerem ca valorile să fie **strict crescătoare**: bucla pornește de la `x[k − 1] + 1` (cu `x[0] = 0`).



x[k] pornește de la x[k−1] + 1

Arborele combinărilor

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

int x[20], n, K;

void back(int k)
{
    for (int v = x[k - 1] + 1; v <= n; v++)       // mai mare decât precedentul
    {
        x[k] = v;
        if (k == K)
        {
            for (int i = 1; i <= K; i++) cout << x[i] << ' ';
            cout << '\n';
        }
        else back(k + 1);
    }
}

int main()
{
    cin >> n >> K;
    back(1);                                      // x[0] = 0 (global)
    return 0;
}
```

DE CE AȘA?

De ce `x[k − 1] + 1`? Dacă fiecare valoare este mai mare decât precedenta, fiecare grupă apare într-o singură ordine (cea crescătoare). Pentru {1, 2}: doar `1 2`, niciodată `2 1`. Nu mai avem nevoie nici de `valid`: valorile strict crescătoare sunt automat distincte. **De ce `x[0] = 0`?** La prima poziție nu avem „precedent”: cu `x[0] = 0`, prima valoare încercată este 1.

DE ȘTIUT

Optimizare. Pe poziția `k` nu are rost să încercăm valori mai mari decât `n − K + k`: nu ar mai rămâne destule valori pentru pozițiile următoare. Pentru n = 5, K = 3: `x[1] ≤ 3`, `x[2] ≤ 4`, `x[3] ≤ 5`.

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

int x[20], n, K, optim, solutii, apeluri;

void back(int k)
{
    apeluri++;
    int sus = optim ? n - K + k : n;            // limita superioară pentru x[k]
    for (int v = x[k - 1] + 1; v <= sus; v++)
    {
        x[k] = v;
        if (k == K) solutii++;
        else back(k + 1);
    }
}

int main()
{
    cin >> n >> K >> optim;
    back(1);
    cout << solutii << ' ' << apeluri;
    return 0;
}
```

Același număr de soluții (C(5, 3) = 10), dar **16** apeluri fără optimizare și **10** cu ea: prin tăierea ramurilor fără ieșire, căutarea se face mai repede.

Intrare / ieșire

Intrare	Ieșire

`2 2` (produs cartezian)	`1 1`
	`1 2`
	`2 1`
	`2 2`

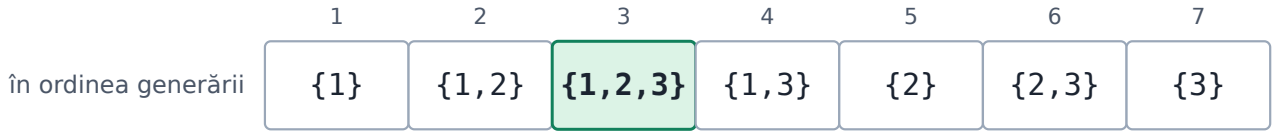
`3 2` (aranjamente)	`1 2`
	`1 3`
	`2 1`
	`2 3`
	`3 1`
	`3 2`

Intrare	Ieșire

`4 2` (combinări)	`1 2`
	`1 3`
	`1 4`
	`2 3`
	`2 4`
	`3 4`

5. Submulțimi

Fiecare prefix strict crescător `x[1..k]` este o submulțime. Așa că afișăm la **fiecare** pas, nu doar la final.



Submulțimi

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

int x[20], n;

void back(int k)
{
    for (int v = x[k - 1] + 1; v <= n; v++)
    {
        x[k] = v;
        for (int i = 1; i <= k; i++) cout << x[i] << ' ';     // x[1..k] este o submulțime
        cout << '\n';
        back(k + 1);                                          // și încercăm să o prelungim
    }
}

int main()
{
    cin >> n;
    back(1);
    return 0;
}
```

DE CE AȘA?

De ce afișăm la fiecare pas? O submulțime poate avea oricâte elemente (1, 2, ..., n), iar fiecare prefix crescător este una. Programul nu are „caz de bază" explicit: când `v` depășește `n`, bucla nu mai rulează și funcția se termină. Mulțimea vidă (ignorată aici) se poate afișa o dată, în `main`. **Varianta „cu 0 și 1":** `x[i] = 1` dacă `i` e în submulțime și `0` altfel: produs cartezian cu valorile {0, 1} și `K = n`; pentru {1, 3} din {1, 2, 3} se obține `1 0 1`.

6. Grile de Bac: cum le rezolvi

„Utilizând metoda backtracking se generează toate numerele de 3 cifre distincte, formate cu cifre din mulțimea {1, 3, 5, 7}. Primele numere generate sunt 135, 137, 153. Care este al 8-lea număr generat?"

1. Recunoaște tipul: cifre distincte, ordinea contează → **aranjamente** de 4 luate câte 3 (24 de soluții).
2. Scrie soluțiile în ordine până la a 8-a: 135, 137, 153, 157, 173, 175, **315**, **317** → al 8-lea este **317**.



a 8-a soluție: a doua din blocul lui 3 → 317

Blocuri

LA BAC

Scurtătură: pe prima poziție, fiecare cifră „acoperă" un bloc de A(3, 2) = 6 soluții. Soluțiile 1–6 încep cu 1, 7–12 cu 3. Deci a 8-a este a doua din blocul lui 3 → 317. Folosește blocurile ca să ajungi repede la soluția cerută, fără să le scrii pe toate.

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

int c[] = {0, 1, 3, 5, 7};             // cifrele disponibile (indice de la 1)
int x[10], n = 4, K = 3, nr, tinta;

bool valid(int k)
{
    for (int i = 1; i < k; i++)
        if (x[i] == x[k]) return false;
    return true;
}

void back(int k)
{
    for (int v = 1; v <= n; v++)
    {
        x[k] = v;
        if (valid(k))
        {
            if (k == K)
            {
                nr++;
                if (nr == tinta)
                {
                    for (int i = 1; i <= K; i++) cout << c[x[i]] << ' ';
                }
            }
            else back(k + 1);
        }
    }
}

int main()
{
    cin >> tinta;
    back(1);
    return 0;
}
```

DE CE AȘA?

De ce generăm indici (`x[k]` între 1 și n) și nu cifrele direct? Algoritmul lucrează cu pozițiile 1..n din mulțimea dată; abia la afișare traducem poziția într-o cifră cu `c[x[k]]`. Așa același program merge pentru orice mulțime (litere, nume, valori).

Alte întrebări frecvente: „Câte soluții au proprietatea ...?" (numeri pe blocuri sau cu formule), „Care este soluția generată imediat înaintea / după ...?", „Câte soluții se generează în total?".

7. Probleme „cu povești", generări standard

Enunț	Este de fapt...
toate echipele de 3 elevi dintr-o clasă de 10	combinări C(10, 3)

Enunț	Este de fapt...
toate podiumurile (aur, argint, bronz) din 8 sportivi	aranjamente A(8, 3)
toate parolele de 4 cifre	produs cartezian 10^4
toate parolele de 4 cifre distincte	aranjamente A(10, 4) = 5 040
toate meniurile alese dintr-o listă de preparate	submulțimi 2^n

8. Greșeli frecvente

Greșeala	Efect	Corect
combinări cu `for (v = 1; …)`	apar și 12 și 21	`for (v = x[k−1] + 1; …)`
aranjamente cu `if (k == n)`	soluții de lungime greșită	`if (k == K)`
uiți că `x[0]` trebuie să fie 0	prima valoare nu e 1	global, inițializat cu 0
submulțimi: afișezi doar la `k == K`	lipsesc submulțimile mai scurte	afișezi la fiecare pas
confunzi „ordinea contează”	număr greșit de soluții	întreabă: 12 ≠ 21?
nu optimizezi limita superioară	timp mai mare	`v <= n − K + k`

9. La BAC

> **LA BAC**
>
> Acestea sunt cele mai frecvente grile din Subiectul I de informatică la clasa a XI-a: se dă o generare (cifre distincte, cuvinte din litere, combinări de numere) și se cere „a câta soluție este …”, „care este soluția următoare”, „câte soluții respectă condiția X”. Rezolvarea standard: recunoști tipul, folosești blocurile ca să sari direct, scrii doar soluțiile necesare.

10. Exersează pe hârtie

Răspunsurile sunt la final: încearcă întâi singur.

1. Generează de mână combinările de 5 luate câte 3. Câte sunt? Care este a 7-a?

2. Se generează submulțimile lui {1, 2, 3, 4} (nevide, în ordinea din lecție). Care submulțime urmează după {1, 3, 4}?

3. Câte dintre șirurile de 4 litere din {a, b} conțin exact două litere `a`?

4. Câte soluții au: aranjamentele de 5 luate câte 3? combinările de 5 luate câte 3? produsul cartezian cu 5 valori și 3 poziții? submulțimile lui {1, …, 5}?

5. (BAC) Pentru generarea din secțiunea 6 (cifrele 1, 3, 5, 7), care este al 20-lea număr generat?

6. Se generează numerele de 3 cifre formate cu cifrele 1, 2, 3 (nu neapărat distincte), în ordine crescătoare. Care este al 10-lea?

7. La combinări de 6 luate câte 4, care este valoarea maximă pe care o poate lua `x[2]`, conform optimizării `n - K + k`?

8. Câte echipe de 5 elevi se pot forma dintr-o clasă de 28? Dar câte echipe de 5 elevi cu un căpitan ales (5 variante)?

9. Scrie submulțimile lui {1, 2, 3} ca șiruri de 0 și 1 (de lungime 3). Ce șir corespunde submulțimii {1, 3}?

10. Pentru ce valori ale lui K aranjamentele A(n, K) și combinările C(n, K) au același număr de soluții?

11. Mai departe (pentru cine vrea mai mult)

Combinările cu repetiții. Alegi K elemente dintre n, cu repetiții, fără să conteze ordinea (de exemplu 3 bile din 4 culori: 112, 123, 333...). Se obțin cu aceeași schemă, dar bucla pornește de la `x[k - 1]` (nu `x[k - 1] + 1`).

Generarea cu biți. Orice submulțime a unei mulțimi de n ≤ 20 elemente se poate reprezenta printr-un număr `mask` de la 0 la $2^n - 1$: bitul `i` este 1 dacă elementul `i` este în submulțime. Parcurgi `for (mask = 0; mask < (1 << n); mask++)` și testezi biții cu `mask & (1 << i)`. Nu e recursiv și e foarte folosit la concursuri.

Triunghiul lui Pascal și C(n, K). Formula `C(n, K) = C(n - 1, K - 1) + C(n - 1, K)` permite să calculezi numărul de combinări fără factoriale (cu o matrice), evitând depășirea lui `int` (vezi lecția X.9).

Numere mari de soluții. Pentru n = 20, K = 10 sunt 184 756 de combinări: încă rezonabil. Pentru n = 30, $2^n \approx 10^9$: backtracking-ul nu mai este fezabil dacă trebuie să afișezi toate soluțiile.

Rezumat

- Produs cartezian: orice valoare (n^K). Aranjamente: valori distincte, K poziții (A(n, K)). Combinări: strict crescătoare, bucla de la `x[k-1] + 1` (C(n, K)). Submulțimi: crescătoare + afișare la fiecare pas (2^n).
- Întrebări-cheie: ordinea contează? Se pot repeta valorile?
- Grile: folosește blocurile (A(n−1, K−1) soluții per prima valoare) ca să sari direct la soluția cerută.
- Optimizare: `v ≤ n - K + k` taie ramuri fără ieșire.

Probleme de lucru (pbinfo)

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

- [ ] #196 **aranjamente** — ușoară
  https://www.pbinfo.ro/probleme/196/aranjamente
- [ ] #197 **combinari** — ușoară
  https://www.pbinfo.ro/probleme/197/combinari
- [ ] #198 **submultimi** — ușoară
  https://www.pbinfo.ro/probleme/198/submultimi

- [ ] #1277 **produscartezian1** — ușoară
  https://www.pbinfo.ro/probleme/1277/produscartezian1

Temă (pbinfo)

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

- [ ] #1278 **produscartezian2** — ușoară
  https://www.pbinfo.ro/probleme/1278/produscartezian2
- [ ] #1286 **submultimi1** — ușoară
  https://www.pbinfo.ro/probleme/1286/submultimi1
- [ ] #3152 **combinari2** — ușoară
  https://www.pbinfo.ro/probleme/3152/combinari2
- [ ] #1287 **cifregen** — ușoară
  https://www.pbinfo.ro/probleme/1287/cifregen
- [ ] #1289 **cifregen2** — ușoară
  https://www.pbinfo.ro/probleme/1289/cifregen2
- [ ] #3162 **cifre-bin-back** — ușoară
  https://www.pbinfo.ro/probleme/3162/cifre-bin-back
- [ ] #3247 **subimp1** — ușoară
  https://www.pbinfo.ro/probleme/3247/subimp1

Pentru cei care vor mai mult

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

- [ ] #1355 **sirab** — ușoară
  https://www.pbinfo.ro/probleme/1355/sirab
- [ ] #3158 **numere123** — ușoară
  https://www.pbinfo.ro/probleme/3158/numere123

Grile din subiectele de BAC (pbinfo)

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

- [ ] #380 **Exercițiul #380**
  modelul combinărilor: numere cu trei cifre distincte, care urmează / câte sunt
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=380

- [ ] #381 **Exercițiul #381**
  numere de 3 cifre cu toate cifrele distincte: a câta soluție este un număr dat
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=381

- [ ] #416 **Exercițiul #416**
  numere formate din câte 3 cifre distincte dintr-o mulțime dată
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=416

- [ ] #417 **Exercițiul #417**
  toate cuvintele de câte 3 litere din mulțimea {a, b, c, ...}: câte sunt / a câta este
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=417

- [ ] #440 **Exercițiul #440**
  toate submulțimile cu 4 elemente ale unei mulțimi: care urmează după o soluție dată
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=440

- [ ] #449 **Exercițiul #449**
  produsul cartezian A×B×C: a câta soluție este un șir dat
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=449

- [ ] #984 **Exercițiul #984**
  seturi de câte 5 elemente: combinări, a câta soluție
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=984

- [ ] #1180 **Exercițiul #1180**
  selecții de câte 4 piese: combinări sau aranjamente
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1180

Citește și exersează în plus

- Generarea aranjamentelor https://www.pbinfo.ro/articole/26111/generarea-aranjamantelor
- Generarea combinărilor https://www.pbinfo.ro/articole/26112/generarea-combinarilor
- Generarea submulțimilor https://www.pbinfo.ro/articole/137/generarea-submultimilor
- Elemente de combinatorică https://www.pbinfo.ro/articole/18941/elemente-de-combinatorica
- CSES — Creating Strings / Apple Division (generări) — permutări de litere cu repetiții și submulțimi, cu teste mari (engleză) https://cses.fi/problemset/task/1622
- Combinatorică — formule (aranjamente, combinări) — formulele și demonstrațiile din spatele numărului de soluții https://ro.wikipedia.org/wiki/Combinări
- VisuAlgo — Recursion (backtracking) — arborele de căutare animat https://visualgo.net/en/recursion

Răspunsuri

Exersează pe hârtie

1. 10 combinări: `123, 124, 125, 134, 135, 145, 234, 235, 245, 345`. A 7-a este **234**.

2. Ordinea: `{1}, {1,2}, {1,2,3}, {1,2,3,4}, {1,2,4}, {1,3}, {1,3,4}, {1,4}, {2}, …`. După `{1,3,4}` urmează **`{1,4}`**.

3. 6 (alegi pozițiile celor două litere `a`: C(4, 2) = 6): `aabb, abab, abba, baab, baba, bbaa`.

4. Aranjamente: 5·4·3 = **60**. Combinări: 60/3! = **10**. Produs cartezian: 5^3 = **125**. Submulțimi: 2^5 = **32** (cu mulțimea vidă; 31 fără ea).

5. 715: blocul lui 7 are soluțiile 19–24 (`713, 715, 731, 735, 751, 753`); a 20-a este a doua din bloc.

6. 211: șirurile încep `111, 112, 113, 121, 122, 123, 131, 132, 133, 211`: a 10-a este `211`.

7. `x[2] ≤ n − K + k = 6 − 4 + 2 =` **4** (pe pozițiile 3 și 4 mai trebuie două valori mai mari).

8. C(28, 5) = 28·27·26·25·24 / 120 = **98 280** echipe. Cu un căpitan ales: 98 280 · 5 = **491 400**.

9. `000, 001, 010, 011, 100, 101, 110, 111` (8 șiruri). Submulțimii {1, 3} îi corespunde **`101`**.

10. `A(n, K) = C(n, K) · K!`, deci sunt egale doar când K! = 1, adică **K = 0 sau K = 1**.