INFORMATICĂ · CLASA A XI-A · METODA BACKTRACKING · LECȚIA 8

# Backtracking (3): partiții, plata unei sume, paranteze și backtracking în plan

Durată: 60 min  Nivel: mate-info  Înainte: lecțiile xi-07

**CE VEI ȘTI DUPĂ LECȚIE**

- Generezi descompunerile unui număr în sumă (partiții), cu condiții suplimentare.
- Rezolvi problema plății unei sume cu bancnote date.
- Generezi șirurile de paranteze corecte.
- Folosești backtracking pe matrice: drumuri într-un labirint, săritura calului.
- Tai ramurile inutile din timp (optimizare prin condiții de validare).

## 1. Același schelet, condiții mai bogate

În lecția anterioară nu aveam condiții între elemente (sau doar „distincte", „crescătoare"). Acum apar condiții **din problemă**: suma să fie `n`, parantezele să fie corecte, drumul să nu intre în zid. Schema rămâne aceeași; se schimbă doar `valid` și condiția de soluție.

| Problemă | Ce reprezintă `x[k]` | `valid(k)` | Soluție când... |
|---|---|---|---|
| partițiile lui n | al k-lea termen al sumei | `x[k] ≥ x[k−1]` și suma ≤ n | suma = n |
| plata unei sume S | câte bancnote din tipul k | suma ≤ S | k = nr. tipuri și suma = S |
| paranteze corecte | `(` sau `)` pe poziția k | nu închidem mai mult decât am deschis | k = 2n |
| drum în labirint | celula a k-a a drumului | în hartă, liberă, nevizitată | celula = destinația |

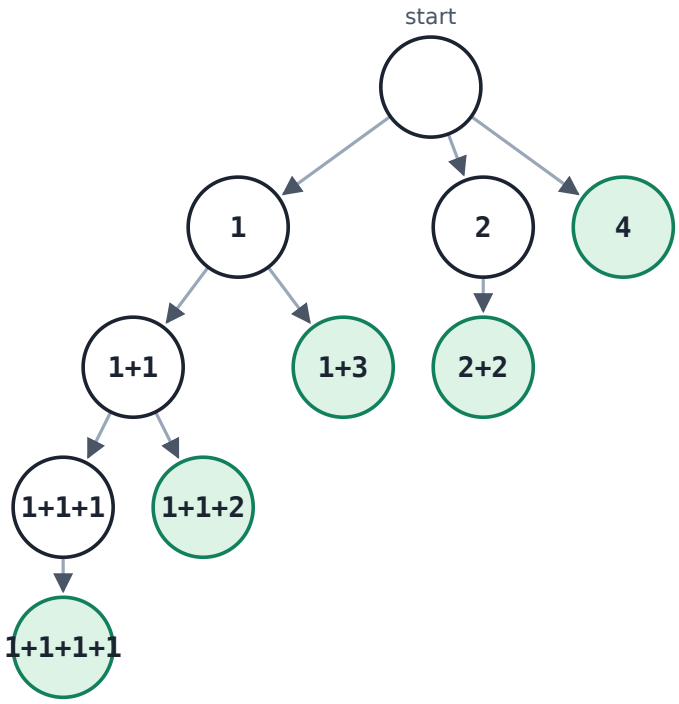
**SFAT**

**Tăierea ramurilor (pruning):** cu cât alegi mai devreme că o ramură nu poate duce la soluție (`valid` fals), cu atât programul face mai puțini pași. Toate cele patru probleme folosesc asta.

## 2. Partițiile unui număr

O **partiție** a lui `n` este o scriere ca sumă de numere naturale nenule. Ca să nu repetăm aceeași partiție în altă ordine (1+3 și 3+1), cerem termenii **în ordine crescătoare**.

Pentru `n = 4`: 1+1+1+1, 1+1+2, 1+3, 2+2, 4 (5 partiții).




fiecare nod = suma scrisă până acum; ramurile care depășesc 4 sunt tăiate de v ≤ n − s

*Arborele partițiilor*

Reținem suma `s` a termenilor puși până acum. Pe poziția `k` încercăm `v` de la `x[k-1]` (nu mai mic) până la `n - s` (nu depășim suma).

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

int x[30], n;

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

void back(int k, int s)                    // s = suma x[1..k-1]
{
    for (int v = x[k - 1] > 0 ? x[k - 1] : 1; v <= n - s; v++)
    {
        x[k] = v;
        if (s + v == n) afisare(k);        // suma completă: o partiție
        else back(k + 1, s + v);
    }
}

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

**DE CE AȘA?**

**De ce `s` e parametru și nu variabilă globală?** La revenire din recursivitate, `s` trebuie să aibă iar valoarea de dinainte. Dacă o transmitem ca parametru, fiecare apel își are propria copie și nu trebuie să o „dăm înapoi" manual. **De ce `v ≤ n − s`?** Dacă adăugăm `v > n − s`, suma depășește `n` și ramura nu poate duce la o soluție: o tăiem înainte să intrăm în ea. **Ordinea afișării:** primul termen e cel mai mic, deci apar întâi partițiile cu termeni mici (`1 1 1 1`) și ultima e `4`.

**Intrare / ieșire**

| Intrare | Ieșire |
|---|---|
| `4` | `1 1 1 1`<br>`1 1 2`<br>`1 3`<br>`2 2`<br>`4` |

## 3. Plata unei sume

Avem bancnote de `m` tipuri (`b[1..m]`) și vrem să plătim exact `S`, cu oricâte bancnote din fiecare tip. `x[k]` = câte bancnote de tipul k folosim.

| | x[1]: de 1 leu | x[2]: de 5 lei | x[3]: de 10 lei |
|---|---|---|---|
| modul 1 | 2 | 0 | 1 |
| modul 2 | 2 | 2 | 0 |
| modul 3 | 7 | 1 | 0 |
| modul 4 | 12 | 0 | 0 |

exemplu: modul 2 = 2·1 + 2·5 + 0·10 = 12

*Plata*

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

int x[20], b[20], m, S;

void back(int k, int s)                          // s = suma plătită cu tipurile 1..k-1
{
    for (int v = 0; s + v * b[k] <= S; v++)       // câte bancnote de tipul k
    {
        x[k] = v;
        int ns = s + v * b[k];
        if (k == m)
        {
            if (ns == S)
            {
                for (int i = 1; i <= m; i++) cout << x[i] << ' ';
                cout << '\n';
            }
        }
        else back(k + 1, ns);
    }
}

int main()
{
    cin >> m;
    for (int i = 1; i <= m; i++) cin >> b[i];
    cin >> S;
    back(1, 0);
    return 0;
}
```

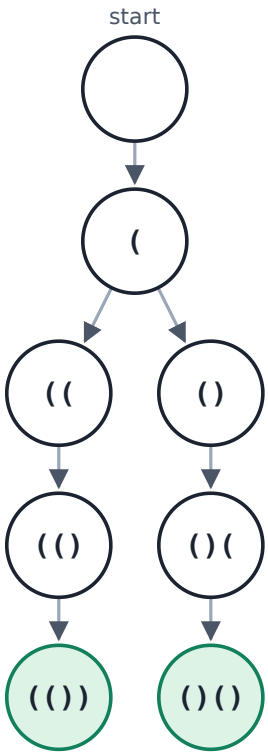
**DE CE AȘA?**

**De ce `v` pornește de la 0?** Putem să nu folosim deloc un tip de bancnotă. **De ce verificăm suma abia la `k == m`?** Până când am hotărât numărul de bancnote pentru **toate** tipurile, nu știm dacă suma a ieșit exactă; pe parcurs doar ne asigurăm că nu o depășim. **Tăiere:** buclă `s + v·b[k] ≤ S` oprește încercările care depășesc suma.

| Intrare | Ieșire |
|---|---|
| `3`<br>`1 5 10`<br>`12` | `2 0 1`<br>`2 2 0`<br>`7 1 0`<br>`12 0 0` |

## 4. Paranteze corecte

Un șir de paranteze este **corect** dacă la fiecare prefix nr. de `)` ≤ nr. de `(`, iar la final sunt egale. Pentru `n` perechi, șirul are lungime `2n`.




la fiecare pas: „(” dacă mai avem de deschis; „)” dacă sunt deschise nepotrivite

*Arbore paranteze*

Reținem `deschise` (câte `(` am pus) și `inchise`. Putem pune `(` dacă `deschise < n`, și `)` dacă `inchise < deschise`.

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

int n;
char s[30];

void back(int k, int deschise, int inchise)
{
    if (k == 2 * n)
    {
        s[k] = 0;
        cout << s << '\n';
        return;
    }
    if (deschise < n)
    {
        s[k] = '(';
        back(k + 1, deschise + 1, inchise);
    }
    if (inchise < deschise)
    {
        s[k] = ')';
        back(k + 1, deschise, inchise + 1);
    }
}

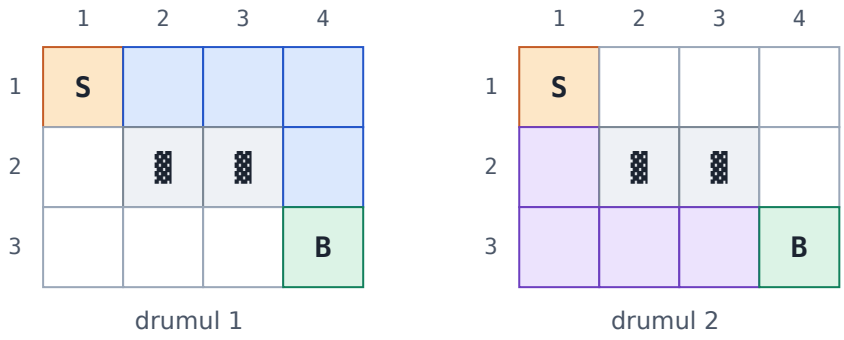
int main()
{
    cin >> n;
    back(0, 0, 0);
    return 0;
}
```

**DE CE AȘA?**

**De ce nu mai generăm toate cele $2^{2n}$ șiruri și le verificăm la final?** Pentru n = 10 ar fi peste un milion de șiruri, din care doar 16 796 sunt corecte. Dacă interzicem de la început alegerile care nu pot duce la soluție, nu se mai ating ramurile moarte. Numărul de șiruri corecte este **numărul lui Catalan**: 1, 2, 5, 14, 42, ... pentru n = 1, 2, 3, 4, 5.

## 5. Drumuri în labirint

Avem o hartă `a[i][j]` (0 = liber, 1 = zid) și vrem toate drumurile simple de la `(is, js)` la `(if, jf)`, mișcându-ne în cele 4 direcții. Marcăm celulele din drum ca să nu le repetăm.




*Labirint*

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

int a[20][20], n, m, is, js, ifin, jfin, nr;
int di[] = {-1, 0, 1, 0}, dj[] = {0, 1, 0, -1};

void back(int i, int j)
{
    if (i == ifin && j == jfin) { nr++; return; }
    a[i][j] = 2;                                    // marcăm: celula e în drum
    for (int d = 0; d < 4; d++)
    {
        int ni = i + di[d], nj = j + dj[d];
        if (ni >= 1 && ni <= n && nj >= 1 && nj <= m && a[ni][nj] == 0)
            back(ni, nj);
    }
    a[i][j] = 0;                                    // demarcăm la întoarcere
}

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) cin >> a[i][j];
    cin >> is >> js >> ifin >> jfin;
    back(is, js);
    cout << nr;
    return 0;
}
```

**DE CE AȘA?**

**De ce demarcăm la întoarcere ( `a[i][j] = 0` )?** Marcarea împiedică drumul să treacă de două ori prin aceeași celulă. Dar celula poate face parte dintr-un **alt** drum: dacă am lăsa marcajul, drumul 2 (care trece prin ea) ar fi pierdut. Marchezi la intrare, demarchezi la ieșire. **De ce nu marcăm destinația?** Ajungem la ea și ne oprim imediat ( `return` ).

**ATENȚIE — GREȘELI FRECVENTE**

Pe hărți mari numărul de drumuri crește exponențial. Pentru „există un drum?" sau „drumul cel mai scurt" se folosesc **BFS** / algoritmul Lee, nu backtracking.

## 6. Cifre în ordine crescătoare cu suma dată

Se dau `k` și `s`. Să se afișeze numerele cu `k` cifre, **strict crescătoare**, cu suma cifrelor `s`.

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

int x[10], K, S;

void back(int k, int s)
{
    for (int v = x[k - 1] + 1; v <= 9; v++)
    {
        if (s + v > S) break;                    // cifrele cresc: mai mari nu ajută
        x[k] = v;
        if (k == K)
        {
            if (s + v == S)
            {
                for (int i = 1; i <= K; i++) cout << x[i];
                cout << '\n';
            }
        }
        else back(k + 1, s + v);
    }
}

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

**DE CE AȘA?**

**De ce `break` și nu `continue`?** Valorile încercate cresc; dacă `s + v > S`, orice valoare mai mare depășește și ea suma. Oprim bucla complet, nu doar saltul peste `v`.

## 7. Greșeli frecvente

| Greșeala | Efect | Corect |
|---|---|---|
| uiți să demarchezi în labirint | se pierd drumuri | `a[i][j] = 0` după bucla de direcții |
| `s` global, modificat și nerestaurat | sume greșite | parametru, sau restaurat manual |
| partiții: `v` de la 1 la fiecare nivel | apar și 1+3 și 3+1 | `v` de la `x[k−1]` |
| paranteze: verifici abia la final | foarte lent | condiții `deschise < n`, `inchise < deschise` |
| plată: verifici suma la fiecare pas | pierzi soluții valide | verifici egalitatea doar la `k == m` |

## 8. La BAC

> **LA BAC**
>
> Subiectul cu backtracking cere de obicei să **completezi** o funcție `valid` sau condiția de soluție pentru un enunț dat („numere cu cifre impare în ordine crescătoare și suma cifrelor 10"). Alte grile: „câte soluții are problema pentru datele X", „care este soluția următoare". Scrie condiția pe baza enunțului, apoi verific-o pe un exemplu mic.

## 9. Exersează pe hârtie

Încearcă întâi singur; răspunsurile sunt la final.

**1.** Scrie toate partițiile lui 5 în ordinea generării. Câte sunt?

**2.** În câte moduri se poate plăti 12 lei cu bancnote de 1, 5 și 10 lei?

**3.** Scrie toate șirurile corecte de paranteze pentru n = 3.

**4.** Cu bancnote de 2 și 5 lei, în câte moduri plătești exact 11 lei?

**5.** Câte partiții ale lui 6 au termeni distincți (strict crescători)?

**6.** Care sunt numerele de 3 cifre strict crescătoare cu suma cifrelor 10?

**7.** Câte șiruri corecte de paranteze are n = 4?

**8.** Pentru partițiile lui 6, care partiție urmează după `1 1 4`?

**9.** (BAC) Într-un program de generare a partițiilor lui 6 se afișează, în ordine, `1 1 1 1 1 1`, `1 1 1 1 2`, `1 1 1 3`, … Care este a 5-a partiție?

**10.** Ce modificare transformă programul din secțiunea 5 într-unul care afișează drumul de lungime minimă?

## 10. Mai departe (pentru cine vrea mai mult)

**Problema damelor și Sudoku.** Același tipar: pui o valoare, verifici condițiile, avansezi sau te întorci. La Sudoku, `valid` verifică linia, coloana și pătratul 3×3.

**Memoizare peste backtracking.** Dacă aceeași stare (suma rămasă, tipul curent) apare de multe ori, rezultatul se poate reține într-un tablou. Obții programarea dinamică (lecția XI.16): de la „toate modurile de a plăti S" la „numărul de moduri", în timp proporțional cu `S · m`.

**Numere Catalan.** Același număr numără: șirurile corecte de paranteze, arborii binari cu `n` noduri, triangulările unui poligon cu `n + 2` laturi. Formula: `C(2n, n) / (n + 1)`.

**Ordonare lexicografică.** Dacă vrei „a t-a soluție" fără a le genera pe toate, numeri soluțiile pe blocuri (ca în lecția anterioară) și coboară direct în ramura bună.

## Rezumat

- Un `valid` bun taie ramurile moarte și transformă backtracking-ul dintr-o căutare brută într-una rapidă.

- Partiții: termeni crescători, `v ≤ n − s`; `s` e parametru ca să se restaureze singur.
- Plata: `v` de la 0, suma verificată la ultimul tip; paranteze: `deschise < n`, `inchise < deschise`.
- Labirint: marchezi la intrare, demarchezi la ieșire.

## Probleme de lucru (pbinfo)

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

- [ ] `#320` **partitiinumar** — ușoară
  https://www.pbinfo.ro/probleme/320/partitiinumar
- [ ] `#344` **paranteze** — ușoară
  https://www.pbinfo.ro/probleme/344/paranteze
- [ ] `#2245` **plata1** — ușoară
  https://www.pbinfo.ro/probleme/2245/plata1

## Temă (pbinfo)

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

- [ ] `#319` **suma35** — ușoară
  https://www.pbinfo.ro/probleme/319/suma35
- [ ] `#321` **partitiinumar1** — ușoară
  https://www.pbinfo.ro/probleme/321/partitiinumar1
- [ ] `#1357` **plusminus** — ușoară
  https://www.pbinfo.ro/probleme/1357/plusminus
- [ ] `#1330` **partitiimultime** — ușoară
  https://www.pbinfo.ro/probleme/1330/partitiimultime
- [ ] `#342` **soarece** — ușoară
  Backtracking în plan: toate drumurile.
  https://www.pbinfo.ro/probleme/342/soarece
- [ ] `#1810` **cifregen5** — ușoară
  Subiect de Bac.
  https://www.pbinfo.ro/probleme/1810/cifregen5
- [ ] `#1812` **literegen1** — ușoară
  Subiect de Bac.
  https://www.pbinfo.ro/probleme/1812/literegen1

## Pentru cei care vor mai mult

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

- [ ] #337 **saritura-calului** — ușoară
  https://www.pbinfo.ro/probleme/337/saritura-calului
- [ ] #1518 **sudoku** — medie
  https://www.pbinfo.ro/probleme/1518/sudoku

## Grile din subiectele de BAC (pbinfo)

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

- [ ] #411 **Exercițiul #411**
  partiții ale unui număr: care urmează / câte sunt
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=411
- [ ] #412 **Exercițiul #412**
  sume de termeni egale cu o valoare dată
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=412
- [ ] #434 **Exercițiul #434**
  generare cu condiții suplimentare
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=434
- [ ] #435 **Exercițiul #435**
  generare cu condiții suplimentare, a doua variantă
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=435
- [ ] #496 **Exercițiul #496**
  soluții care respectă o proprietate
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=496
- [ ] #803 **Exercițiul #803**
  șiruri generate cu restricții
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=803
- [ ] #1213 **Exercițiul #1213**
  numere cu cifre în ordine și suma dată
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1213
- [ ] #1763 **Exercițiul #1763**
  partiții și sume: numărarea soluțiilor
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1763

## Citește și exersează în plus

- Metoda backtracking — prezentare generală https://www.pbinfo.ro/articole/16597/metoda-backtracking
- CSES — Grid Paths / Two Sets (backtracking pe tablă) — labirint cu drumuri și tăieri de ramuri (engleză) https://cses.fi/problemset/task/1625

- Numerele lui Catalan — de ce parantezele corecte sunt numărate de Catalan(n) https://ro.wikipedia.org/wiki/Număr_Catalan
- Partiția unui număr — câte partiții are n https://ro.wikipedia.org/wiki/Partiția_unui_număr

# Răspunsuri

## Exersează pe hârtie

**1.** `1 1 1 1 1`, `1 1 1 2`, `1 1 3`, `1 2 2`, `1 4`, `2 3`, `5`: **7** partiții.

**2.** **4** moduri: `(12, 0, 0)`, `(7, 1, 0)`, `(2, 2, 0)`, `(2, 0, 1)` (numărul de bancnote de 1, 5, 10).

**3.** `((()))`, `(()())`, `(())()`, `()(())`, `()()()`: **5** șiruri.

**4.** Cu `x` bancnote de 2 și `y` de 5: `2x + 5y = 11` → `y = 1, x = 3`; `y` impar, `y ≤ 2`, deci doar **1** mod: `(3, 1)`.

**5.** Partiții cu termeni distincți ale lui 6: `1+2+3`, `1+5`, `2+4`, `6` → **4**.

**6.** Cifre strict crescătoare, suma 10: **127, 136, 145, 235**.

**7.** Numărul lui Catalan pentru n = 4: **14**.

**8.** Partițiile lui 6 în ordine: `1 1 1 1 1 1`, `1 1 1 1 2`, `1 1 1 3`, `1 1 2 2`, `1 1 4`, `1 2 3`, … Deci după `1 1 4` urmează **`1 2 3`**.

**9.** Ordinea: 1) `1 1 1 1 1 1`; 2) `1 1 1 1 2`; 3) `1 1 1 3`; 4) `1 1 2 2`; 5) **`1 1 4`**.

**10.** Păstrezi lungimea drumului curent (un parametru `lung`) și, la destinație, reții minimul: `if (lung < best) best = lung;`. Mai simplu și mai rapid: BFS / Lee.