Laborator 5: Backtracking

Responsabili:

- Darius Neațu
- Ștefan Popa

Autori:

- Radu Stochitoiu (2018)
- Razvan Chitu (2018)

Obiective laborator

- Intelegerea notiunilor de baza despre backtracking;
- Insusirea abilitatilor de implementare a algoritmilor bazati backtracking;
- Rezolvarea unor probleme NP-complete în timp exponential.

Precizari initiale

Toate exemplele de cod se gasesc in demo-lab05.zip. Acestea apar incorporate si in textul laboratorului pentru a facilita parcurgerea cursiva a laboratorului.

Ce este Backtracking?

Backtracking este un algoritm care cauta **una sau mai multe solutii** pentru o problema, printr-o cautare exhaustiva, mai eficienta insa in general decat o abordare „genereaza si testeaza”, de tip „forta bruta”, deoarece un candidat partial care nu duce la o solutie este abandonat. Poate fi folosit pentru orice problema care presupune o cautare in **spatiul starilor**. In general, in timp ce cautam o solutie e posibil sa dam de o infundatura in urma unei alegeri gresite sau sa gasim o solutie, dar sa dorim sa cautam in continuare alte solutii. In acel moment trebuie sa ne intoarcem pe pasii facuti (**backtrack**) si la un moment dat sa luam alta decizie. Este relativ simplu din punct de vedere conceptual, dar complexitatea algoritmului este exponentiala.

Importanţa – aplicaţii practice

Exista foarte multe probleme (de exemplu, probleme NP-complete sau NP-dificile) care pot fi rezolvate prin algoritmi de tip backtracking mai eficient decat prin „forta bruta” (adica generarea tuturor alternativelor si selectarea solutiilor). Atentie insa, complexitatea computationala este de cele mai multe ori exponentiala. O eficientizare se poate face prin combinarea cu tehnici de propagare a restrictiilor. Orice problema care are nevoie de parcurgerea spatiului de stari se poate rezolva cu backtracking.

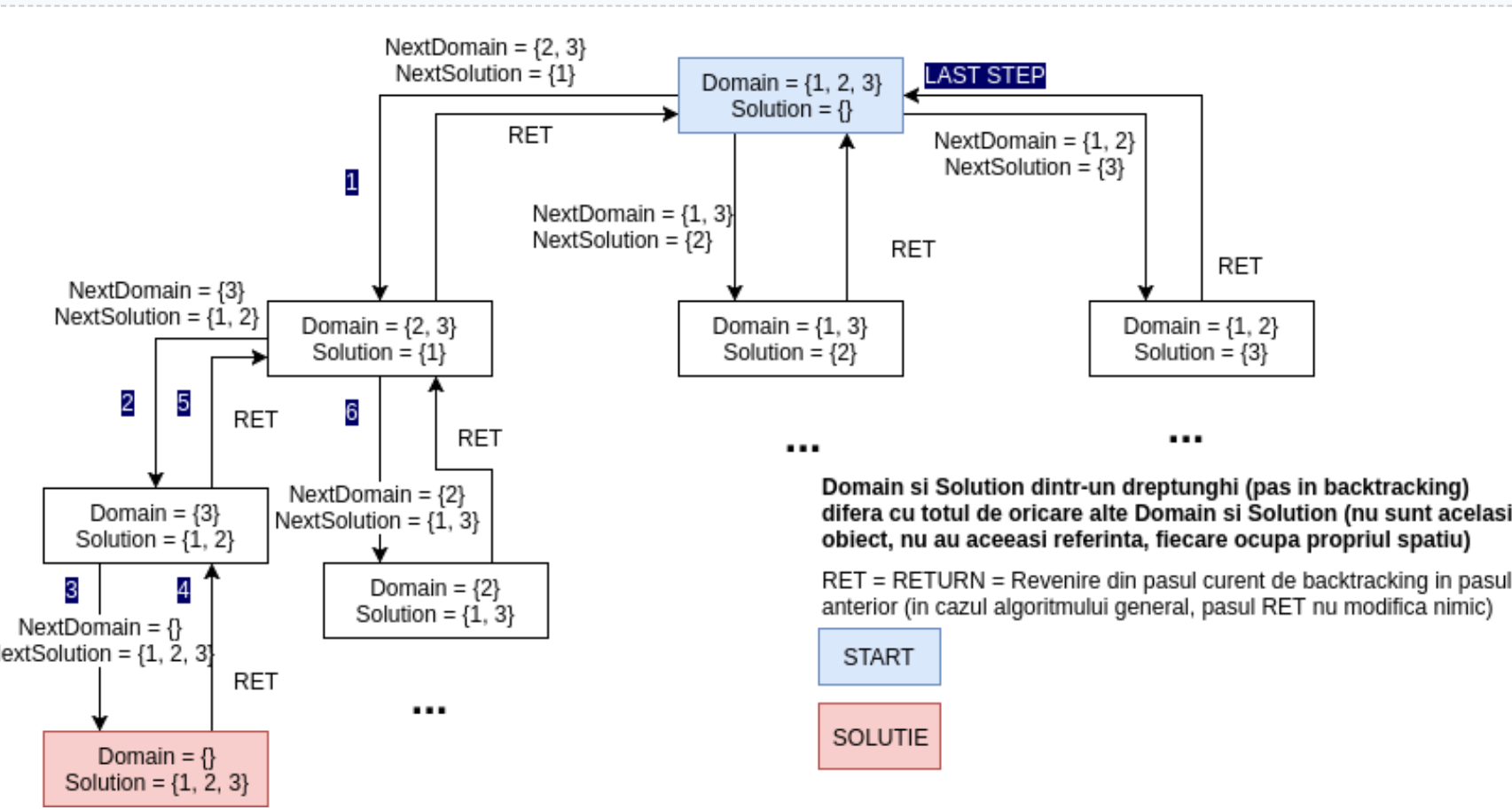
Descrierea problemei si a rezolvarilor

Pornind de la strategiile clasice de parcurgere a **spatiului de stari**, algoritmii de tip backtracking practic enumera un set de candidati partiali, care, dupa completarea definitiva, pot deveni solutii potentiale ale problemei initiale. Exact ca strategiile de **parcurgere în latime/adancime** si backtracking-ul are la baza expandarea unui nod curent, iar determinarea solutiei se face într-o maniera incrementala. Prin natura sa, backtracking-ul este recursiv, iar în arborele expandat top-down se aplica operatii de tipul pruning (taiere) daca solutia partiala nu este valida.

Algoritm de baza

```
/* Domain - domeniul curent (cu un cardinal mai mic decat la pasul trecut)
Solution - solutia curenta pe care o extindem catre cea finala */
back(Domain, Solution):
    if check(Solution):
        print(Solution)
        return

    for value in Domain:
        NextSolution = Solution.push(value)
        NextDomain = Domain.erase(value)
        back(NextDomain, NextSolution)
```



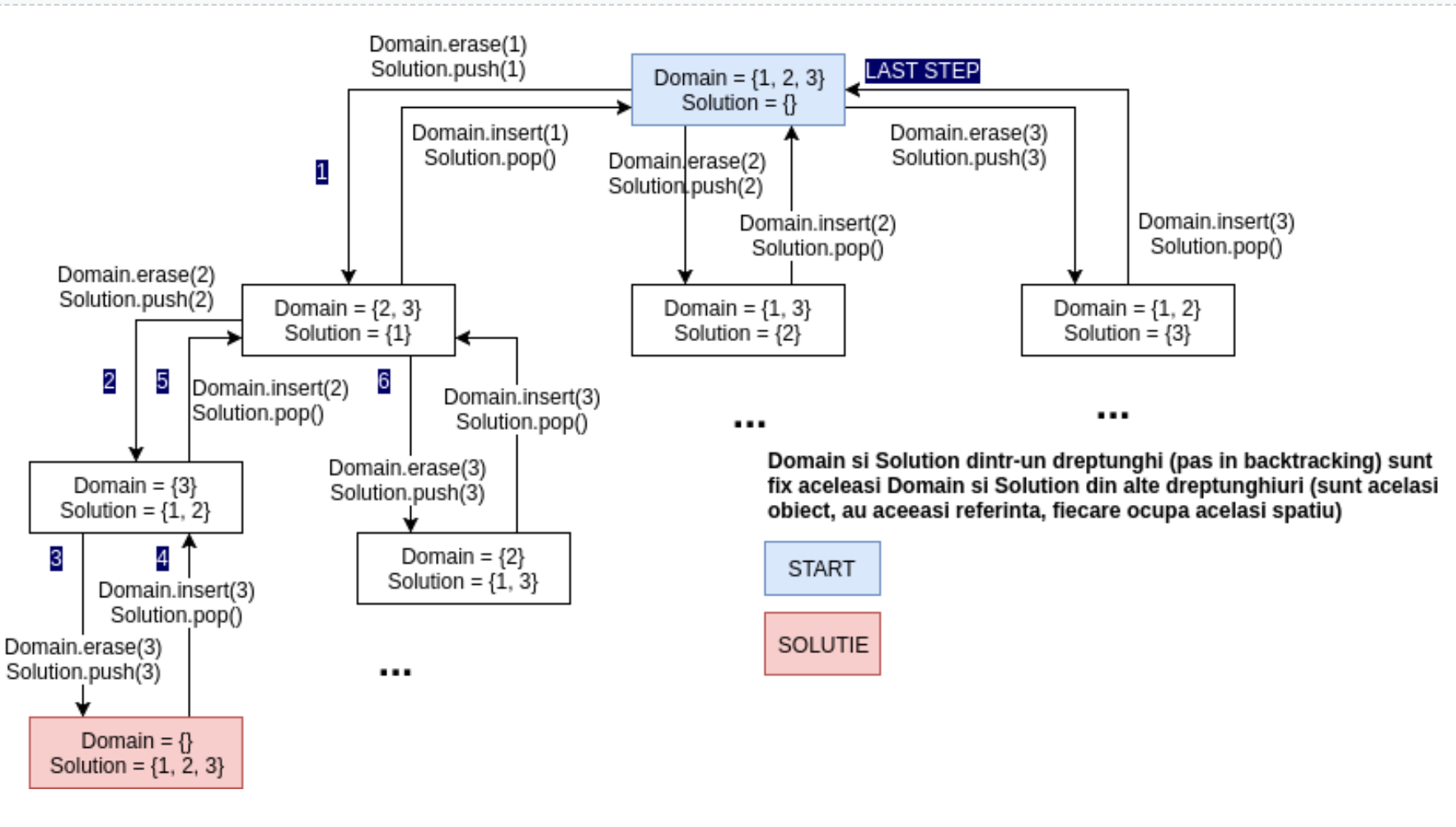
Algoritm de baza (modificat pentru transmitere prin referinta)

```
/* Domain - domeniul curent (cu un cardinal mai mic decat la pasul trecut)
Solution - solutia curenta pe care o extindem catre cea finala */
back(Domain, Solution):
    if check(Solution):
        print(Solution)
        return

    for value in Domain:
        /* DO */
        Solution = Solution.push(value)
        Domain = Domain.erase(value)

        /* RECURSION */
        back(Domain, Solution)

        /* UNDO */
        Solution = Solution.pop()
        Domain = Domain.insert(value)
```



Exemple clasice

Ne vom ocupa in continuare de urmatoarele probleme:

- Permutari
- Combinari
- Aranjamente
- Submultimi
- Genererare de siruri
- Problema damelor
- Problema soricelului
- Tic-Tac-Toe
- Sudoku
- Ultimate Tic-Tac-Toe

Sudoku si Ultimate Tic-Tac-Toe sunt probleme foarte grele. In general nu putem explora tot spatiul starilor pentru un input arbitrar dat.

Permutari

Enunt

Se da un numar N. Sa se genereze toate permutarile multimii formate din toate numerele de la 1 la N.

Exemple

Exemplu 1

Solutii

Backtracking (algoritmul in cazul general)

Implementare

Complexitate

Solutia va avea urmatoarele complexitati:

- complexitate temporala : $T(n) = O(n * n!) = O(n!)$
  - explicatie : Complexitate generarii permutarilor, $O(n!)$, se inmulteste cu complexitatea copierii vectorilor solutie si domeniu si a stergerii elementelor din domeniu, $O(n)$
- complexitate spatiala : $S(n) = O(n^2)$
  - explicatie : Fiecare nivel de recursivitate are propria lui copie a solutiei si a domeniului. Sunt n nivele de recursivitate, deci complexitatea spatiala este $O(n * n) = O(n^2)$

Backtracking (date transmise prin referinta)

Implementare

Complexitate

Solutia va avea urmatoarele complexitati:

- complexitate temporala : $T(n) = O(n * n!)$
  - explicatie : Complexitate generarii permutarilor, $O(n!)$, se inmulteste cu complexitatea stergerii elementelor din domeniu, $O(n)$
- complexitate spatiala : $S(n) = O(n)$
  - explicatie : Spre deosebire de solutia anterioara, toate nivelele de recursivitate folosesc aceeasi solutie si acelasi domeniu. Complexitatea spatiala este astfel redusa la $O(n)$

Abordarea aceasta este mai eficienta decat cea generala prin evitarea folosirii memoriei auxiliare.

Backtracking (taierea ramurilor nefolositoare)

Implementare

Complexitate

Solutia va avea urmatoarele complexitati:

- complexitate temporala : $T(n) = O(n * n!) = O(n!)$
  - explicatie : Complexitate generarii permutarilor, O(n!), se inmulteste cu complexitatea

iterarii prin domeniu, $O(n)$

- complexitate spatiala : $S(n) = O(n)$
  - explicatie : Toate nivelele de recursivitate folosesc aceeasi solutie si acelasi domeniu.

Aceasta solutie este optima si are complexitate temporala $T(n) = O(n!)$. Nu putem sa obtinem o solutie mai buna, intrucat trebuie sa generam n! permutari.

De asemenea, este optima si din punct de vedere spatial, intrucat trebuie sa avem $S(n) = O(n)$, din cauza stocarii permutarii generate.

Combinari

Enunt

Se dau numerele N si K. Sa se genereze toate combinarile multimii formate din toate numerele de la 1 la N, luate cate K.

Exemple

Exemplu 1

Solutii

Backtracking (taierea ramurilor nefolositoare)

Implementare

Complexitate

Solutia va avea urmatoarele complexitati:

- complexitate temporala : $T(n) = O(Combinari(n, k))$
- complexitate spatiala : $S(n) = O(n + k) = O(n)$
  - explicatie : $k <= n$, deci $O(n + k) = O(n)$

Problema soricelului

Enunt

Se da un numar N si o matrice patratica de dimensiuni N x N in care elementele egale cu 1 reprezinta ziduri (locuri prin care nu se poate trece), iar cele egale cu 0 reprezinta spatii goale. Aceasta matrice are un soricel in celula (0, 0) si o bucata de branza in celula (N - 1, N - 1). Scopul soricelului e sa ajunga la bucata de branza. Afisati toate modurile in care poate face asta stiind ca acesta poate merge doar in dreapta sau in jos cu cate o celula la fiecare pas.

Exemple

Exemplu 1
Exemplu 2
Exemplu 3

Solutii

Backtracking (transmitere prin referinta)

Implementare

Complexitate

Solutia va avea urmatoarele complexitati:

- complexitate temporala : $T(n) = O(Aranjamente(n^2, 2n-1))$
  - explicatie: Initial in domeniu avem n^2 valori. Noi dorim sa generam toate submultimile ordonate de cate $2n-1$ elemente. Acestea sunt tocmai aranjamentele de n^2 luate cate $2n-1$.
- complexitate spatiala : $S(n) = O(n^2)$
  - explicatie: Trebuie sa stocam informatie despre drum, care are $2n-1$ celule; stocam domeniul care are n^2 elemente

Backtracking (taierea ramurilor nefolositoare)

Implementare

Complexitate

Solutia va avea urmatoarele complexitati:

- complexitate temporala : $T(n) = O(2^{2n})$
  - explicatie: avem de urmat un sir de $2n-1$ mutari, iar la fiecare pas avem 2 variante posibile
- complexitate spatiala : $S(n) = O(n)$
  - explicatie: stocam maximum $2n-1$ casute

Exercitii

In acest laborator vom folosi scheletul de laborator din arhiva skel-lab05.zip.

Aranjamente

Fie N si K doua **numere naturale strict pozitive**. Se cere afisarea tuturor aranjamentelor de N elemente luate cate K din multimea {1, 2, …, N}.

Exemplu 1

Se doreste o complexitate $T(n, k) = A(n, k)$.

Hint

Solutiile se vor genera in ordine lexico-grafica!

Checkerul asteapta sa le stocati in aceasta ordine.

Submultimi

Fie N un **numar natural strict pozitiv**. Se cere afisarea tuturor submultimilor multimii {1, 2, …, N}.

Exemplu 1

Se doreste o complexitate $T(n) = O(2^n)$.

Hint

Solutiile se vor genera in ordine lexico-grafica!

Checkerul asteapta sa le stocati in aceasta ordine.

Problema damelor

Problema damelor (sau problema reginelor) trateaza plasarea a 8 regine de sah pe o tablă de șah de dimensiuni 8 x 8 astfel incat sa nu existe doua regine care se ameninta reciproc. Astfel, se cauta **o solutie** astfel incat nicio pereche de doua regine sa nu fie pe acelasi rand, pe aceeasi coloana, sau pe aceeasi diagonala. Problema cu opt regine este doar un caz particular pentru problema generala, care presupune plasarea a N regine pe o tablă de sah N x N în aceleasi conditii. Pentru aceasta problema, există solutii pentru toate numerele naturale N cu excepția lui N = 2 si N = 3.

Exemplu 1
Hint

Se va cauta o singura solutie (**oricare** solutie corecta), care va fi returnata sub format unui vector cu $n+1$ elemente.

Solutia este $sol[0], sol[1], \ldots, sol[n]$, unde $sol[i]$ = coloana unde vom plasa regina de pe linia i.

Elementul 0 este nefolosit, dorim sa pastram conventia cu indexare de la 1.

Generare de siruri

Vi se da o lista de caractere si o lista de frecvente (pentru caracterul de pe pozitia i, frecventa de pe pozitia i). Vi se cere sa generati toate sirurile care se pot forma cu aceste caractere si aceste frecvente stiind ca nu pot fi mai mult de K aparitii consecutive ale aceluiasi caracter.

Exemplu 1
Exemplu 2

Solutiile se vor genera in ordine lexico-grafica!

Checkerul asteapta sa le stocati in aceasta ordine.

Bonus

Problema damelor (AC3)

Aplicati AC3 pe problema damelor.

Algoritmul AC-3 (Arc Consistency Algorithm) este de obicei folosit in probleme de satisfacere a constrangerilor (CSP). Acesta sterge arcele din arborele de stari care sigur nu se vor folosi niciodata.

AC-3 lucreaza cu:

- constrangeri
- variabile
- domenii de variabile

O variabila poate lua orice valoare din domeniul sau la orice pas. O constrangere este o relatie sau o limitare a unor variabile.

Exemplu AC-3

Consideram A, o variabila ce are domeniul D(A) = {0, 1, 2, 3, 4, 5, 6} si B o variabila ce are domeniul D(B) = {0, 1, 2, 3, 4}. Cunoastem constrangerile: C1 = “A trebuie sa fie impar” si C2 = “A + B trebuie sa fie egal cu 5”.

Algoritmul AC-3 va elimina in primul rand toate valorile pare ale lui A pentru a respecta C1 ⇒ D(A) = {1, 3, 5}. Apoi, va incerca sa satisfaca C2, asa ca va pastra in domeniul lui B toate valorile care adunate cu valori din D(A) pot da 5 ⇒ D(B) = {0, 2, 4}.

AC-3 a redus astfel domeniile lui A si B, reducand semnificativ timpul folosit de algoritmul backtracking.

Extra

Immortal

Enunt

Solutie

Ultimate Tic Tac Toe

The AI Games - Ultimate Tic Tac Toe

Diverse

- Hall of PA
- Regulamente PA 2020
- Proiect
- Catalog
- Test practic
- [TEME] Configuratie vmchecker
- [skel_graph] Precizari laboratoare 07-12

Laboratoare

- 00: Introducere și Relaxare
  - skel-lab00.zip
- 01: Divide et Impera
  - skel-lab01.zip
  - sol-lab01.zip
- 02: Greedy
  - skel-lab02.zip
  - sol-lab02.zip
- 03: Programare Dinamică 1
  - skel-lab03.zip
  - sol-lab03.zip
- 04: Programare Dinamică 2
  - skel-lab04.zip
  - sol-lab04.zip
- 05: Backtracking
  - skel-lab05.zip
  - sol-lab05.zip
- 06: Minimax
  - skel-lab06.zip
  - sol-lab06.zip
- 07: Parcurgerea Grafurilor. Sortare Topologică
  - skel-lab07.zip
  - sol-lab07.zip
- 08: Aplicații DFS
  - skel-lab08.zip
  - sol-lab08.zip
- 09: Drumuri minime
  - skel-lab09.zip
  - sol-lab09.zip
- 10: Arbori minimi de acoperire
  - skel-lab10.zip
- 11: Flux Maxim
  - skel-lab11.zip
- 12: A*
  - skel-lab12.zip

Materiale Suplimentare Test Practic

- Articole
- Probleme

Crash-course Opțional

- Debugging și Structuri de Date
- Schelet: Debugging și Structuri de Date

Table of Contents

- Laborator 5: Backtracking
  - Obiective laborator
  - Precizari initiale
  - Ce este Backtracking?
  - Importanţa – aplicaţii practice
  - Descrierea problemei si a rezolvarilor
  - Algoritm de baza
  - Algoritm de baza (modificat pentru transmitere prin referinta)
  - Exemple clasice
    - Permutari
      - Enunt
      - Exemple
      - Solutii
      - Backtracking (algoritmul in cazul general)
      - Backtracking (date transmise prin referinta)
      - Backtracking (taierea ramurilor nefolositoare)
      - Complexitate
    - Combinari
      - Enunt
      - Exemple
      - Solutii
      - Backtracking (taierea ramurilor nefolositoare)
      - Complexitate
    - Problema soricelului
      - Enunt
      - Exemple
      - Solutii
      - Backtracking (transmitere prin referinta)
      - Complexitate
      - Backtracking (taierea ramurilor nefolositoare)
      - Complexitate
  - Exercitii
    - Aranjamente
    - Submultimi
    - Problema damelor
    - Generare de siruri
  - Bonus
    - Problema damelor (AC3)
      - Exemplu AC-3
  - Extra
    - Immortal
    - Ultimate Tic Tac Toe