Structuri de Date și Algoritmi
Programare dinamică - Aprofundare

Dr. Stefania Loredana Nita – stefania.nita@mta.ro

Programare dinamică

- Care este principiul de funcționare pentru programarea dinamică?

- În ce tip de probleme este folosită programarea dinamică?

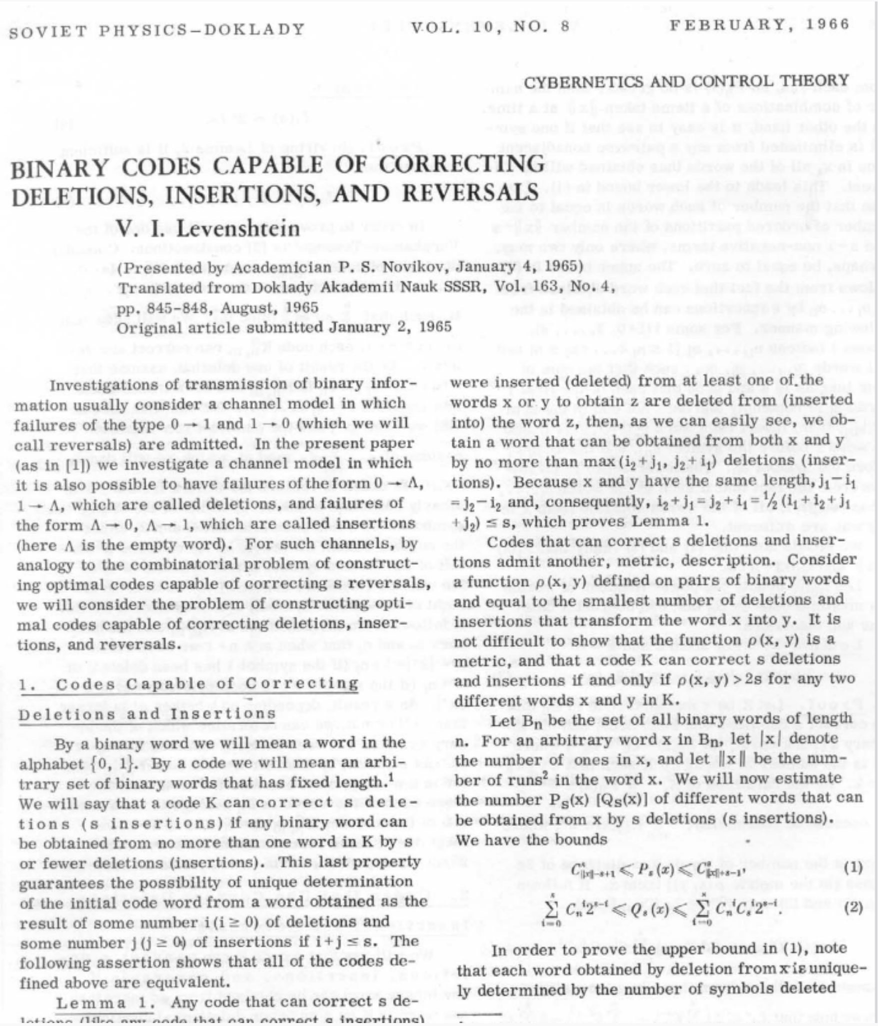
Programare dinamică

- Pași:
  1. Caracterizăm structura unei soluții optime.
  2. Definim recursiv valoarea unei soluții optime.
  3. Calculăm valoarea unei soluții optime (de obicei, cu abordarea bottom-up).
  4. Reconstruim soluția optimă din informațiile anterioare.
- Pașii 1-3 = baza programării dinamice
- Pasul 4 se poate omite dacă se dorește o soluție optimă (și nu soluția în sine).

Programare dinamică vs. Greedy

	Greedy	**Programare dinamică**
Concept	La momentul curent alege opțiunea care pare cea mai bună (optimă); subproblema rezultată este rezolvată după ce se face alegerea.	La momentul curent face o alegere, dar aceasta poate depinde de soluțiile subproblemelor.
Optimalitate	Conduce la o soluție optimă dacă putem demonstra că optimul local => optimul global.	Conduce la o soluție optimă.
Complexitate timp	Polinomial	Polinomial, dar de obicei mai puțin eficient decât greedy.
Complexitate spațiu	Mai eficient, pentru că nu caută înapoi pentru alte soluții.	Folosește un tabel pentru a stoca răspunsurile pentru stările anterior calculate.
Exemple	Problema rucsacului (fracțional)	Problema rucsacului (0/1).

Distanța Levenshtein (Distanța de editare)



SOVIET PHYSICS–DOKLADY VOL. 10, NO. 8 FEBRUARY, 1966

CYBERNETICS AND CONTROL THEORY

BINARY CODES CAPABLE OF CORRECTING DELETIONS, INSERTIONS, AND REVERSALS

V. I. Levenshtein

(Presented by Academician P. S. Novikov, January 4, 1965)
Translated from Doklady Akademii Nauk SSSR, Vol. 163, No. 4, pp. 845–848, August, 1965
Original article submitted January 2, 1965

Investigations of transmission of binary information usually consider a channel model in which failures of the type $0 \to 1$ and $1 \to 0$ (which we will call reversals) are admitted. In the present paper (as in [1]) we investigate a channel model in which it is also possible to have failures of the form $0 \to \Lambda$, $1 \to \Lambda$, which are called deletions, and failures of the form $\Lambda \to 0$, $\Lambda \to 1$, which are called insertions (here Λ is the empty word). For such channels, by analogy to the combinatorial problem of constructing optimal codes capable of correcting s reversals, we will consider the problem of constructing optimal codes capable of correcting deletions, insertions, and reversals.

1. Codes Capable of Correcting Deletions and Insertions

By a binary word we will mean a word in the alphabet $\{0, 1\}$. By a code we will mean an arbitrary set of binary words that has fixed length.[1] We will say that a code K can correct s deletions (s insertions) if any binary word can be obtained from no more than one word in K by s or fewer deletions (insertions). This last property guarantees the possibility of unique determination of the initial code word from a word obtained as the result of some number i ($i \geq 0$) of deletions and some number j ($j \geq 0$) of insertions if $i + j \leq s$. The following assertion shows that all of the codes defined above are equivalent.

Lemma 1. Any code that can correct s deletions (like any code that can correct s insertions)

were inserted (deleted) from at least one of the words x or y to obtain z are deleted from (inserted into) the word z, then, as we can easily see, we obtain a word that can be obtained from both x and y by no more than $\max(i_2 + j_1, j_2 + i_1)$ deletions (insertions). Because x and y have the same length, $j_1 - i_1 = j_2 - i_2$ and, consequently, $i_2 + j_1 = j_2 + i_1 = \frac{1}{2}(i_1 + i_2 + j_1 + j_2) \leq s$, which proves Lemma 1.

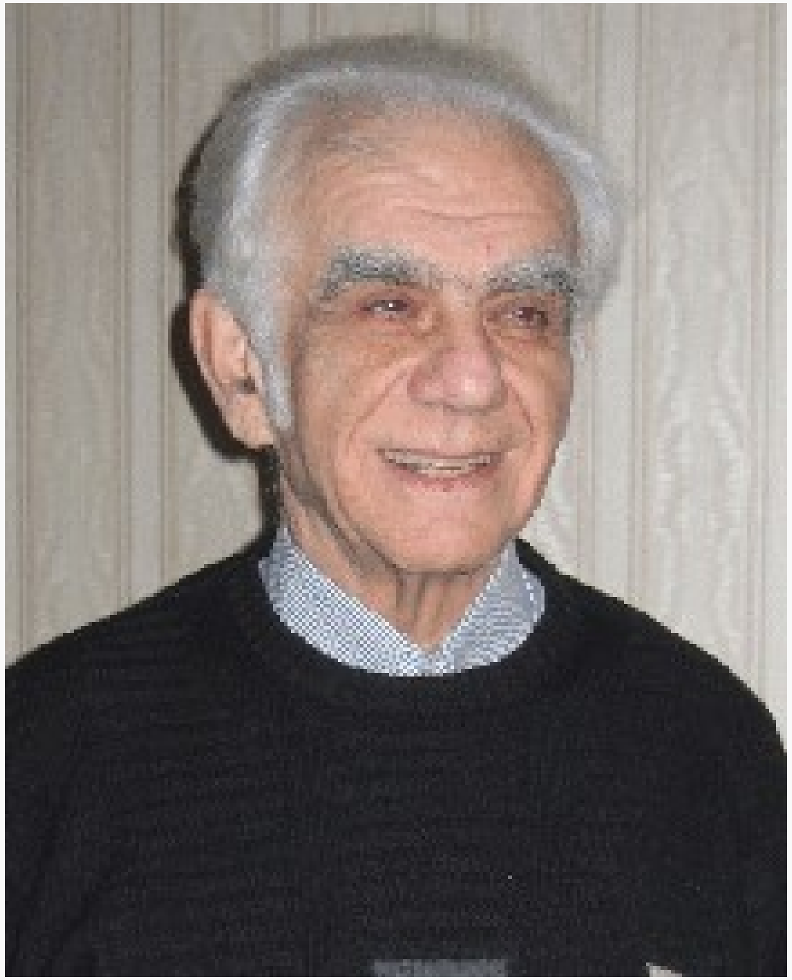
Codes that can correct s deletions and insertions admit another, metric, description. Consider a function $\rho(x, y)$ defined on pairs of binary words and equal to the smallest number of deletions and insertions that transform the word x into y. It is not difficult to show that the function $\rho(x, y)$ is a metric, and that a code K can correct s deletions and insertions if and only if $\rho(x, y) > 2s$ for any two different words x and y in K.

Let B_n be the set of all binary words of length n. For an arbitrary word x in B_n, let $|x|$ denote the number of ones in x, and let $\|x\|$ be the number of runs[2] in the word x. We will now estimate the number $P_s(x)$ [$Q_s(x)$] of different words that can be obtained from x by s deletions (s insertions). We have the bounds

$$C^s_{\|x\|-s+1} \leqslant P_s(x) \leqslant C^s_{\|x\|+s-1}, \tag{1}$$

$$\sum_{i=0}^{s} C_n^i 2^{s-i} \leqslant Q_s(x) \leqslant \sum_{i=0}^{s} C_n^i C_s^i 2^{s-i}. \tag{2}$$

In order to prove the upper bound in (1), note that each word obtained by deletion from x is uniquely determined by the number of symbols deleted



Sursă articol și imagine: https://nymity.ch/sybilhunting/pdf/Levenshtein1966a.pdf

Vladimir Levenshtein, 20 Martie 1935 – 6 Septembrie 2017
Sursă imagine: https://www.ithistory.org/honor-roll/mr-vladimir-iosifovich-levenshtein

Distanța Levenshtein

- Se consideră două cuvinte A și B, cu $|A| = m$, $|B| = n$.
- Cerință: să se transforme cuvântul A în cuvântul B, folosind operațiile:
  - Adăugarea unei litere
  - Ștergerea unei litere
  - Modificarea unei litere
- **Transformarea se va face folosind un număr minim de operații.**

Distanța Levenshtein

- Exemplu: SPATE și PACT.

	#	P	A	C	T
#					
S					
P					
A			cost[i,j]		
T					
E					

cost[i,j] = nr. de operații necesar pentru a transforma SPA în PA.

Distanța Levenshtein

- Se poate folosi programarea dinamică?

Distanța Levenshtein

- Se poate folosi programarea dinamică?
  - **Soluția parțială** = “Acesta este costul pentru a transforma A până la poziția i în B până la poziția j”.
  - **Următorul pas** = “Pentru a transforma A până la poziția x în B până la poziția y, ultima operație ar trebui să fie o adăugare, ștergere sau modificare?”

Distanța Levenshtein

- Exemplu: SPATE și PACT.

	#	P	A	C	T
#	șterge				
S					
P					
A			adaugă		
T					modifică
E					

Operațiile pentru transformarea SPATE și PACT.

Distanța Levenshtein

- Exemplu: SPATE și PACT.

	#	P	A	C	T
#	c_{00}	c_{01}	c_{02}	c_{03}	c_{04}
S	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}
P	c_{20}	c_{21}	c_{22}	c_{23}	c_{24}
A	c_{30}	c_{31}	?		
T					
E					

Operațiile pentru transformarea SPATE și PACT.
Simplificare:
cost[i,j] ≡ c_{ij}

Distanța Levenshtein

- Exemplu: SPATE și PACT.

	#	P	A	C	T
#	c_{00}	c_{01}	c_{02}	c_{03}	c_{04}
S	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}
P	c_{20}	modifică c_{21}	șterge c_{22}	c_{23}	c_{24}
A	c_{30}	adaugă c_{31}	?		
T					
E					

Distanța Levenshtein

- Obs: numărul total de operații folosite în conversie este mai mic sau egal decât $\max(m, n)$.
  - De ce?
- În șirul transformărilor, de la un termen la altul, se folosește o singură operație dintre cele 3.
  - 2 termeni consecutivi ai șirului diferă printr-un singur caracter;
  - Nu contează ordinea efectuării calculelor.

Distanța Levenshtein

- Formula de recurență:

$$cost_{A,B}[i,j] = \begin{cases} \max\{i,j\}, \, dacă \min\{i,j\} = 0 \\ \min\{cost_{A,B}[i-1,j]+1, \\ cost_{A,B}[i,j-1]+1, \\ cost_{A,B}[i-1,j-1]+1_{A[i]\neq B[j]}\}, altfel \end{cases}$$

Distanța Levenshtein

$$cost_{A,B}[i,j] = \begin{cases} \max\{i,j\}, \textit{dacă} \min\{i,j\} = 0 \\ \min\{cost_{A,B}[i-1,j]+1, \\ cost_{A,B}[i,j-1]+1, \\ cost_{A,B}[i-1,j-1]+1_{A[i]\neq B[j]}\}, altfel \end{cases}$$

- Costul transformării caracterului vid, care precede pe *A*, în primele *j* caractere ale cuvântului *B* este *j* (i.e. se fac *j* operații de adăugare).
- Costul transformării primelor *i* caractere ale lui *A* în caracterul vid care îl precede pe *B* este *i* (i.e. se fac *i* operații de ștergeri).
- $cost_{A,B}[i,j] = cost_{A,B}[i-1,j]+1$ => Caracterul A[i] a fost șters
- $cost_{A,B}[i,j] = cost_{A,B}[i,j-1]+1$ => Pe poziția A[i] se inserează caracterul B[j]
- $cost_{A,B}[i,j] = cost_{A,B}[i-1,j-1]+\mathbf{1}_{\boldsymbol{A[i]\neq B[j]}}$=> caracterul A[i] este înlocuit cu caracterul B[j] **($\boldsymbol{cost_{A,B}[i-1,j-1]}$ se adună cu 1 dacă A[i]≠B[j])**

Distanța Levenshtein

- Exemplu: SPATE și PACT.

	#	P	A	C	T
#	0	1	2	3	4
S	1				
P	2				
A	3				
T	4				
E	5				

$$cost_{A,B}[i,j] = \begin{cases} \max\{i,j\}, \, dacă \min\{i,j\} = 0 \\ \min\{cost_{A,B}[i-1,j]+1, \\ \quad cost_{A,B}[i,j-1]+1, \\ \quad cost_{A,B}[i-1,j-1]+1_{A[i]\neq B[j]}\}, altfel \end{cases}$$

Distanța Levenshtein

- Exemplu: SPATE și PACT.

	#	P	A	C	T
#	0	1	2	3	4
S	1	1			
P	2				
A	3				
T	4				
E	5				

$$cost_{A,B}[i,j] = \begin{cases} \max\{i,j\}, \text{dacă} \min\{i,j\} = 0 \\ \min\{cost_{A,B}[i-1,j]+1, \\ \quad cost_{A,B}[i,j-1]+1, \\ \quad cost_{A,B}[i-1,j-1]+1_{A[i]\neq B[j]}\}, altfel \end{cases}$$

Exemplu:

$$c_{1,1} = \min\{c_{1-1,1}+1, c_{1,1-1}+1, c_{1-1,1-1}+1\}$$
$$= \min\{c_{0,1}+1, c_{1,0}+1, c_{0,0}+1\}$$
$$= \min\{0+1, 1+1, 0+1\} = 1$$

Distanța Levenshtein

- Exemplu: SPATE și PACT.

	#	P	A	C	T
#	0	1	2	3	4
S	1	1	2	3	4
P	2				
A	3				
T	4				
E	5				

$$cost_{A,B}[i,j] = \begin{cases} \max\{i,j\}, \text{dacă} \min\{i,j\} = 0 \\ \min\{cost_{A,B}[i-1,j]+1, \\ \quad cost_{A,B}[i,j-1]+1, \\ \quad cost_{A,B}[i-1,j-1]+1_{A[i]\neq B[j]}\}, altfel \end{cases}$$

Distanța Levenshtein

- Exemplu: SPATE și PACT.

	#	P	A	C	T
#	0	1	2	3	4
S	1	1	2	3	4
P	2	1			
A	3				
T	4				
E	5				

$$cost_{A,B}[i,j] = \begin{cases} \max\{i,j\}, dacă \min\{i,j\} = 0 \\ \min\{cost_{A,B}[i-1,j]+1, \\ \quad cost_{A,B}[i,j-1]+1, \\ \quad cost_{A,B}[i-1,j-1]+1_{A[i]\neq B[j]}\}, altfel \end{cases}$$

Distanța Levenshtein

- Exemplu: SPATE și PACT.

	#	P	A	C	T
#	0	1	2	3	4
S	1	1	2	3	4
P	2	1	2	3	4
A	3	2	1	2	3
T	4	3	2	2	2
E	5	4	3	3	3

$$cost_{A,B}[i,j] = \begin{cases} \max\{i,j\}\,,\, dacă \min\{i,j\} = 0 \\ \min\{cost_{A,B}[i-1,j]+1, \\ \quad cost_{A,B}[i,j-1]+1, \\ \quad cost_{A,B}[i-1,j-1]+1_{A[i]\neq B[j]}\}, altfel \end{cases}$$

Costul final al transformării A -> B se află pe poziția cost[m,n] în matricea de costuri.

$$best\_score = cost_{A,B}[m,n]$$

Distanța Levenshtein

- Exemplu: SPATE și PACT.

	#	P	A	C	T
#	**0**	1	2	3	4
S	**1**	1	2	3	4
P	2	**1**	2	3	4
A	3	2	**1**	**2**	3
T	4	3	2	2	**2**
E	5	4	3	3	**3**

$$cost_{A,B}[i,j] = \begin{cases} \max\{i,j\}, dacă \min\{i,j\} = 0 \\ \min\{cost_{A,B}[i-1,j]+1, \\ cost_{A,B}[i,j-1]+1, \\ cost_{A,B}[i-1,j-1]+1_{A[i]\neq B[j]}\}, altfel \end{cases}$$

Urma (*trace*) indică modul în care este obținută valoarea minimă și poate fi folosită pentru a determina ordinea operațiilor în transformarea A->B.

Distanța Levenshtein

- Exemplu: SPATE și PACT.

	#	P	A	C	T
#	**0**	1	2	3	4
S	**1**	1	2	3	4
P	2	**1**	2	3	4
A	3	2	**1**	**2**	3
T	4	3	2	2	**2**
E	5	4	3	3	**3**

$$cost_{A,B}[i,j] = \begin{cases} \max\{i,j\}, \, dacă \min\{i,j\} = 0 \\ \min\{cost_{A,B}[i-1,j]+1, \\ \quad cost_{A,B}[i,j-1]+1, \\ \quad cost_{A,B}[i-1,j-1]+1_{A[i]\neq B[j]}\}, altfel \end{cases}$$

- Șterge A[5]
- Adaugă pe poz A[3] B[3]
- Șterge S[1]

adaugă $\rightarrow$

șterge $\downarrow$

modifică $\searrow$

Distanța Levenshtein

- Exemplu: SPATE și PACT.

	#	P	A	C	T
#	**0**	1	2	3	4
S	**1**	1	2	3	4
P	2	**1**	2	3	4
A	3	2	**1**	**2**	3
T	4	3	2	2	**2**
E	5	4	3	3	**3**

1	2	3	4	5
S	P	A	T	E

↓ Șterge A[1]

1	2	3	4	5
	P	A	T	E

↓ Adaugă pe poz A[3] B[3]

1	2	3	4	5
P	A	C	T	E

↓ Șterge A[5]

1	2	3	4	5
P	A	C	T	

1	2	3	4
P	A	C	T

- Șterge A[5]
- Adaugă pe poz A[3] B[3]
- Șterge A[1]

Distanța Levenshtein

- Exemplu: SPATE și PACT.

- Transformarea este unică?

1	2	3	4	5
S	P	A	T	E

1	2	3	4
P	A	C	T

↓ Șterge A[1]

1	2	3	4	5
	P	A	T	E

↓ Adaugă pe poz A[3] B[3]

1	2	3	4	5
P	A	C	T	E

↓ Șterge A[5]

1	2	3	4	5
P	A	C	T	

Distanța Levenshtein - Pseudocode

```
function LevenshteinDistance(char s[1..m], char t[1..n]):
  // for all i and j, d[i,j] will hold the Levenshtein distance between
  // the first i characters of s and the first j characters of t
  declare int d[0..m, 0..n]

  set each element in d to zero

  // source prefixes can be transformed into empty string by
  // dropping all characters
  for i from 1 to m:
      d[i, 0] := i

  // target prefixes can be reached from empty source prefix
  // by inserting every character
  for j from 1 to n:
      d[0, j] := j

  for j from 1 to n:
      for i from 1 to m:
          if s[i] = t[j]:
            substitutionCost := 0
          else:
            substitutionCost := 1

          d[i, j] := minimum(d[i-1, j] + 1,                   // deletion
                             d[i, j-1] + 1,                   // insertion
                             d[i-1, j-1] + substitutionCost)  // substitution

  return d[m, n]
```

Sursa: https://en.wikipedia.org/wiki/Levenshtein_distance

Distanța Levenshtein

- Complexitate?

Distanța Levenshtein

- Complexitate?
  - $O(m \times n)$

Distanța Levenshtein

- Variații
  - Fiecare operație are un cost diferit, cu condiția cost_modificare < cost_adăugare + cost_ștergere
    - De ce?

Distanța Levenshtein

- Aplicații
  - verificatori ortografici
  - Sisteme de corecție pentru recunoașterea optică a caracterelor
  - Procesarea limbajului natural
  - Distanța lingvistică

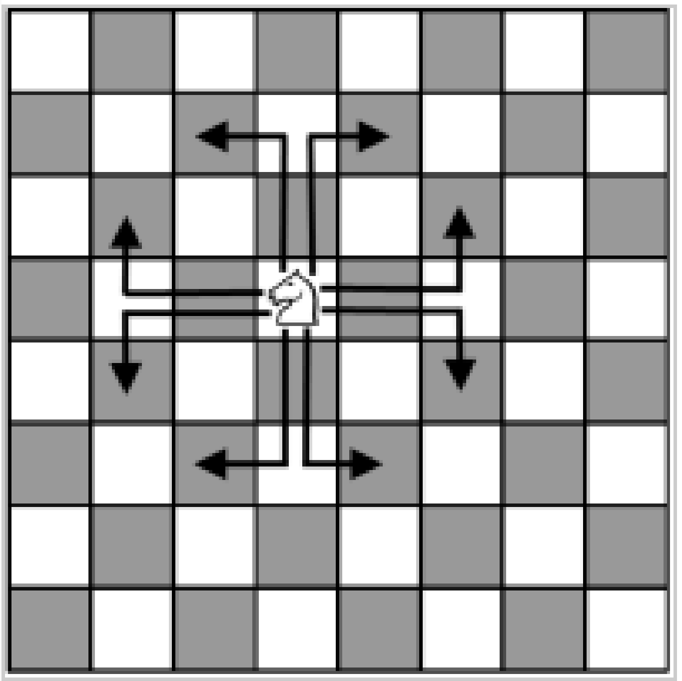
Cal pe tabla de șah

- Fiind date două coordonate pe tabla de șah ($n \times n$), determinați cel mai scurt drum dintre acestea utilizând mutările unui cal.

Sursa exemplu: Doina Hrinciuc Logofătu. *C++.Probleme rezolvate și algoritmi*

Cal pe tabla de șah

- Fiind date două coordonate pe tabla de șah ($n \times n$), determinați cel mai scurt drum dintre acestea utilizând mutările unui cal.



Sursa exemplu “Cal pe tabla de șah”: Doina Hrinciuc Logofătu. *C++.Probleme rezolvate și algoritmi*
Sursa imagine: *https://e4e4.wordpress.com/basics/*

Cal pe tabla de șah

- Fiind date două coordonate pe tabla de șah ($n \times n$), determinați cel mai scurt drum dintre acestea utilizând mutările unui cal.
  - Exemplu:
    - N=5
    - Start = (5,2)
    - Fin = (1,4)

Cal pe tabla de șah

- Fiind date două coordonate pe tabla de șah ($n \times n$), determinați cel mai scurt drum dintre acestea utilizând mutările unui cal.
  - Exemplu:
    - N=5
    - Start = (5,2)
    - Fin = (1,4)

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

Cel mai scurt drum: 2 pași
Drumul parcurs: (5,2), (3,3), (1,4)

Programare dinamică

- Când se aplică programarea dinamică?

Programare dinamică

- Când se aplică programarea dinamică?
- Problema trebuie să aibă următoarele caracteristici:
  - Substructură optimală
  - Suprapunerea problemelor

Programare dinamică

- Substructură optimă
  - O soluție optimă a problemei include soluții optime ale subproblemelor.
  1. Presupunem că există o soluție mai optimă a subproblemei.
  2. Arătăm că presupunerea contrazice optimalitatea problemei inițiale.

Programare dinamică

- Suprapunerea problemelor
  - Când un algoritm recursiv rezolvă mereu o aceeași problemă => subprobleme suprapuse.
  - Spațiul subproblemelor trebuie să fie restrâns.
  - O subproblemă este rezolvată, iar rezultatul este salvat pentru utilizări viitoare => timp de regăsire constant.