

Algoritmi. Limbaj algoritmic

SD 2014/2015

Algoritmi. Introducere

Limbaj algoritmic

Tipuri de date

Tablouri și structuri

Exemplu

- ▶ o secvență de numere: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Exemplu

► o secvență de numere: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

► secvența Fibonacci

definiția matematică: $F_n = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{if } n > 1 \end{cases}$

Exemplu

- ▶ o secvență de numere: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

- ▶ secvența Fibonacci

definiția matematică:
$$F_n = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{if } n > 1 \end{cases}$$

- ▶ implemetare C++

```
int F(int n) {  
    if (n == 0) return 0;  
    else if (n == 1) return 1;  
    else  
        return F(n-1) + F(n-2);  
}
```

Algoritmi: etimologie



Muhammad ibn Musa **al-Khwarizmi** - matematician persan; a scris prima carte de algebră (cca. 830).

- metode pentru adunarea, înmulțirea și împărțirea numerelor.

Algoritmi: definiție

- ▶ Nu există o definiție standard pentru noțiunea de algoritm.
- ▶ Cambridge Dictionary:
"A set of mathematical instructions that must be followed in a fixed order, and that, especially if given to a computer, will help to calculate an answer to a mathematical problem."
- ▶ Schneider and Gersting 1995 (Invitation for Computer Science):
"An algorithm is a well-ordered collection of unambiguous and effectively computable operations that when executed produces a result and halts in a finite amount of time."
- ▶ Gersting and Schneider 2012 (Invitation for Computer Science, 6th edition):
"An algorithm is ordered sequence of instructions that is guaranteed to solve a specific problem."

► Wikipedia:

"In mathematics and computer science, an algorithm is a step-by-step procedure for calculations. Algorithms are used for calculation, data processing, and automated reasoning. An algorithm is an effective method expressed as a finite list of well-defined instructions for calculating a function. Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states, eventually producing "output" and terminating at a final ending state. The transition from one state to the next is not necessarily deterministic; some algorithms, known as randomized algorithms, incorporate random input."

Algoritmi: model de calcul, problema rezolvată

Toate definițiile au ceva în comun:

- ▶ datele/informația și procesarea acestora/acesteia în pași. Acestea sunt descrise în general de un model de calcul.

Un **model de calcul** este format din:

- ▶ **memorie** - modul de reprezentare a datelor.
- ▶ **instrucțiuni**
 - sintaxă* - descrie sintactic pașii de procesare;
 - semantică* - descrie pașii de procesare realizați de execuția unei instrucțiuni; în general este dată de o relație de tranziție peste configurații (sistem tranzițional).

- ▶ un algoritm trebuie să producă un rezultat, adică un algoritm trebuie să **rezolve o problemă**.

O problemă este în general reprezentată de o pereche (**input, output**), unde input reprezintă descrierea datelor de intrare (instanță) iar output descrierea datelor de ieșire (rezultatul).

Algoritmi și Structuri de date

- ▶ Algoritm: metodă de rezolvare a unei probleme.
- ▶ Structuri de date: metodă de a păstra/reprezenta informația.

- ▶ Algoritm: metodă de rezolvare a unei probleme.
- ▶ Structuri de date: metodă de a păstra/reprezenta informația.

Algorithms + Data Structures = Programs. — Niklaus Wirth

I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships. — Linus Torvalds

Algoritmi: proprietăți

- ▶ **input** (intrare) – zero sau mai multe entități de date furnizate din exterior.
- ▶ **output** (ieșire) – algoritmul produce informație.
- ▶ **terminare** – pentru orice intrare, algoritmul execută un număr finit de pași.
- ▶ **corectitudine** – algoritmul se termină și produce ieșirea corectă pentru orice intrare; spunem că algoritmul **rezolvă** problema dată.

- ▶ Un algoritm trebuie să folosească un volum rezonabil de resurse de calcul: [spațiu de] memorie și timp [de execuție].
- ▶ Avem nevoie de algoritmi eficienți pentru:
 - ▶ a salva timp de așteptare, spațiu de depozitare, consum energie, etc.;
 - ▶ scalabilitate: putem rezolva probleme de dimensiuni mari cu aceleași resurse (CPU, memorie, disc, etc.);
 - ▶ soluții optimizate.

Algoritmi: eficiență

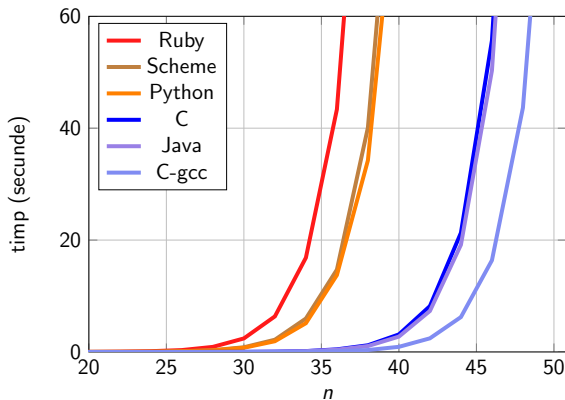


Figura : Execuția algoritmului recursiv F (Fibonacci).

Observație: Comportamentul este diferit în funcție de tipul implementării; totuși diferențele nu sunt atât de substanțiale \Rightarrow Problema e algoritmul! (complexitate exponențială)

Proiectarea algoritmilor

Rezolvarea algoritmică a problemelor presupune următoarele etape:

- ▶ definirea problemei
 - ▶ abstractizează detaliile irelevante;
- ▶ identificarea clasei din care face parte problema și a unui algoritm de construcție a soluției;
- ▶ analiza corectitudinii și a eficienței algoritmului;
- ▶ implementarea algoritmului;
- ▶ (optimizare și generalizare).

Descrierea algoritmilor

- ▶ **informal:** limbaj natural.
- ▶ **formal:**
 - ▶ notație matematică (mașini Turing, lambda-calcul (Church), funcții recursive, etc.);
 - ▶ limbaje de programare: de nivel înalt, de nivel jos, declarative (e.g., programare funcțională, programare logică). Acesta poate fi și un model informal dacă nu există o semnificație formală pentru limbaj.
- ▶ **semiformal:**
 - ▶ pseudo-cod: combină notația formală a limbajelor de programare cu limbajul natural;
 - ▶ notație grafică: scheme logice, automate (state machines), diagrame de activități.

Algoritmi. Introducere

Limbaj algoritmic

Tipuri de date

Tablouri și structuri

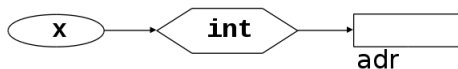
Limbaaj algoritmic

Avem nevoie de un limbaj care este

- ▶ **simplic**: pentru a fi ușor de înțeles;
- ▶ **expresiv**: pentru a descrie algoritmi;
- ▶ **abstract**, în descrierea algoritmului accentul cade pe gândirea algoritmică și nu pe detaliile de implementare;
- ▶ un model de calcul adecvat pentru analiza complexității algoritmilor, în special complexitatea timp.

Variabilă

- ▶ Nume
- ▶ Adresă
- ▶ Atribute (tip de date asociat valorilor memorate)

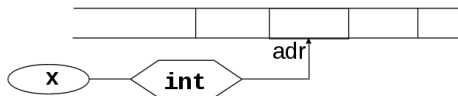


- ▶ Instanță a variabilei

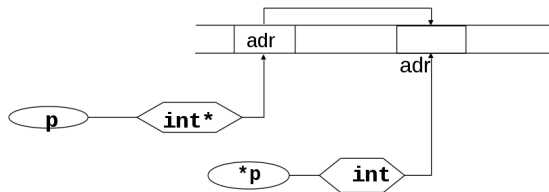
Modelul de calcul

▶ Memoria: structură liniară de celule

▶ variabile



▶ pointeri



Algoritmi. Introducere

Limbaj algoritmic

Tipuri de date

Tablouri și structuri

- ▶ Domeniu (colecția de obiecte)
- ▶ Operații
- ▶ Categoriile de tipuri de date:
 - ▶ Tipuri de date elementare
 - ▶ Tipuri de date structurate de nivel jos
 - ▶ operațiile la nivel de componentă
 - ▶ Tipuri de date de nivel înalt
 - ▶ operațiile implementate de algoritmi utilizator

Tipuri de date elementare

- ▶ Numere întregi
 - ▶ valori: numere întregi
 - ▶ operații: $+$, $-$, ...
- ▶ Numere reale
 - ▶ valori: numere raționale
 - ▶ operații: $+$, $-$, ...
- ▶ Valori booleene
 - ▶ valori: *true*, *false*
 - ▶ operații: *and*, *or*, *not*

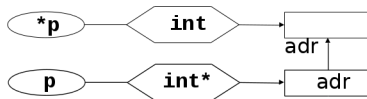
Tipuri de date elementare

► Caractere

- valori: 'a', 'b', ...
- operații: —

► Pointeri

- valori: adrese de variabile aparținând altui tip, valoarea NULL
- operații: —
- referire indirectă: *p



Tipuri de date elementare

- ▶ Operatori pentru numere întregi:
 - ▶ aritmetici: $a+b$, $a-b$, $a*b$, a/b , $a\%b$
 - ▶ relaționali: $a==b$, $a!=b$, $a<b$, $a\leq b$, $a>b$, $a\geq b$

operatie	timp(operatie)	
	cost uniform	cost logaritmic
$a + b$	$O(1)$	$O(\max(\log a, \log b))$
...		

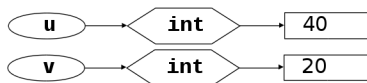
- ▶ Expresii
- ▶ Compuse (bloc): {instrucțiuni}
- ▶ Condiționale: **if if-else**
- ▶ Iterative: **while repeat for**
- ▶ Întreruperea secvenței: **return**

► Atribuirea

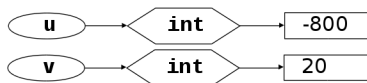
- **Sintaxa:** $\langle \text{variabila} \rangle \leftarrow \langle \text{expresie} \rangle$
- **Semantica:**
 - se evaluează $\langle \text{expresie} \rangle$ și rezultatul obținut se memorează în locația desemnată de $\langle \text{variabila} \rangle$
 - este singura instrucțiune cu ajutorul căreia se poate modifica conținutul memoriei
- cost uniform $O(1)$, cost logaritm $O(\log \langle \text{expresie} \rangle)$

Exemplu:

- Înainte de atribuire:



- După atribuirea $u \leftarrow -v * u$:



► Atribuirea în cazul pointerilor

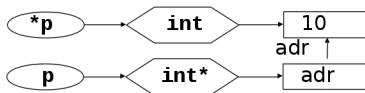
► Sintaxa:

$* < \text{variabila_pointer} > \leftarrow < \text{expresie} >$

► Semantica:

- se evaluează $< \text{expresie} >$ și rezultatul obținut se memorează în locația de la adresa stocată în $< \text{variabila_pointer} >$

► Exemplu: $*p \leftarrow 10$



► if

► Sintaxa:

if $\langle \text{expresie} \rangle$ **then**
 $\langle \text{secventa} - \text{instructiuni}_1 \rangle$
else
 $\langle \text{secventa} - \text{instructiuni}_2 \rangle$

if $\langle \text{expresie} \rangle$ **then**
 $\langle \text{secventa} - \text{instructiuni}_1 \rangle$

Observație: $\langle \text{expresie} \rangle$ este o expresie cu rezultat boolean după evaluare

► Semantica:

- Se evaluează $\langle \text{expresie} \rangle$. Dacă rezultatul este *true*, atunci se execută $\langle \text{secventa} - \text{instructiuni}_1 \rangle$ iar dacă rezultatul este *false*, atunci se execută $\langle \text{secventa} - \text{instructiuni}_2 \rangle$ după care instrucțiunea if se termină

Instructiunea if

- ▶ cost uniform $O(1)$, cost logaritmic $O(1)$
- ▶ Exemplu: calcululul minimului a două numere:

if $a < b$ **then**

$min \leftarrow a$

else

$min \leftarrow b$

sau

$min \leftarrow a$

if $b < a$ **then**

$min \leftarrow b$

▶ while

▶ Sintaxa:

while < *expresie* > **do**
 < *secventa – instructiuni* >

▶ Semantica:

- ▶ Se evaluează < *expresie* >
- ▶ Dacă rezultatul este *true* atunci se execută < *secventa – instructiuni* > după care se reia procesul începând cu pasul 1. Dacă rezultatul este *false* atunci execuția instrucțiunii while se termină.

Exemplu **while**

- ▶ cel mai mic k astfel încât $7^k \geq n$ pentru un n dat

$k \leftarrow 0$

$sapte_la_k \leftarrow 1$

while $sapte_la_k < n$ **do**

$k \leftarrow k + 1$

$sapte_la_k \leftarrow sapte_la_k * 7$

► repeat

► Sintaxa:

repeat

< secventa – instructiuni >

until *< expresie >*;

► Semantica:

Instrucțiunea:

repeat

S

until *e*;

simulează execuția următorului program:

S

while *not e* **do**

S

Exemplu **repeat**

- ▶ cel mai mic k astfel încât $7^k \geq n$ pentru un n dat

$k \leftarrow 0$

$sapte_la_k \leftarrow 1$

repeat

$k \leftarrow k + 1$

$sapte_la_k \leftarrow sapte_la_k * 7$

until $sapte_la_k \geq n$;

► for

► Sintaxa:

for < *variabila* > ← < *expresie*₁ > **to** < *expresie*₂ > **do**
 < *secvența – instrucțiuni* >

sau

for < *variabila* > ← < *expresie*₁ > **downto** < *expresie*₂ > **do**
 < *secvența – instrucțiuni* >

Observație: < *variabila* > este o variabilă de tip întreg, iar
< *expresie*₁ > și < *expresie*₂ > sunt expresii cu rezultat întreg după
evaluare

► for

► Semantica:

for $i \leftarrow e1$ **to** $e2$ **do**
 S

este echivalentă cu:

$i \leftarrow e1$
 $temp \leftarrow e2$
while $i \leq temp$ **do**
 S
 $i \leftarrow i + 1$

► for

► Semantica:

for $i \leftarrow e1$ **downto** $e2$ **do**
 S

este echivalentă cu:

$i \leftarrow e1$
 $temp \leftarrow e2$
while $i \geq temp$ **do**
 S
 $i \leftarrow i - 1$

- ▶ Limbajul este modular: un program conține un număr de module
- ▶ Un modul în limbajul prezentat este identificat cu un subprogram
- ▶ Subprograme:
 - ▶ Proceduri
 - ▶ Funcții

► Proceduri:

► Sintaxa:

```
Procedure nume (lista-parametri-formali)  
begin  
    secventa-instructiuni  
end
```

► **Apel:** NUME(lista-parametri-actuali)

- interfața între o procedură și modulul care o apelează se realizează doar prin intermediul parametrilor și a variabilelor globale

► Exemplu:

Procedure *SWAP* (x,y)

begin

$aux \leftarrow x$

$x \leftarrow y$

$y \leftarrow aux$

end

Apel:

SWAP(a, b)

SWAP(b, c)

- ▶ **Funcții:**

- ▶ **Sintaxa:**

Function *nume (lista-parametri-formali)*

begin

 secventa-instructiuni

end

secventa-instructiuni conține măcar o instrucțiune *return < expr >*

- ▶ **Apel:** NUME(lista-parametri-actuali)

utilizat într-o expresie: valoarea întoarsă de funcție este cea obținută prin evaluarea *< expr >*

► Exemplu:

```
Function max3(x,y,z)  
begin  
    temp  $\leftarrow$  x  
    if y > temp then  
        temp  $\leftarrow$  y  
    if z > temp then  
        temp  $\leftarrow$  z  
    return temp  
end
```

Apel:

max3(*a, b, c*)

$2 * \text{max3}(a, b, c) > 5$

Algoritmi. Introducere

Limbaj algoritmic

Tipuri de date

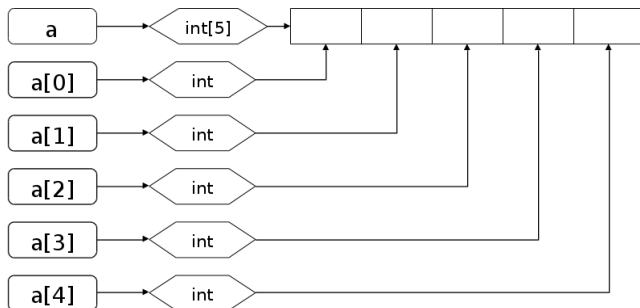
Tablouri și structuri

- ▶ Ansamblu omogen de variabile numite componentele tabloului
- ▶ Toate componentele aparțin aceluiași tip
- ▶ Componentele sunt identificate cu ajutorul indicilor

- ▶ Tablourile sunt utilizate pentru a reprezenta mulțimi, secvențe (ordinea elementelor este importantă), matrici

- ▶ Tablourile pot fi:
 - ▶ unidimensionale (1-dimensionale)
 - ▶ bidimensionale (2-dimensionale)

Tablouri unidimensionale



Tablouri unidimensionale

- ▶ Memoria este o secvență contiguă de locații
- ▶ Ordinea de memorare – ordinea indicilor
- ▶ Operațiile se realizează prin intermediul componentelor

Exemple:

```
for  $i \leftarrow 0$  to  $n - 1$  do  
     $a[i] \leftarrow 0$ 
```

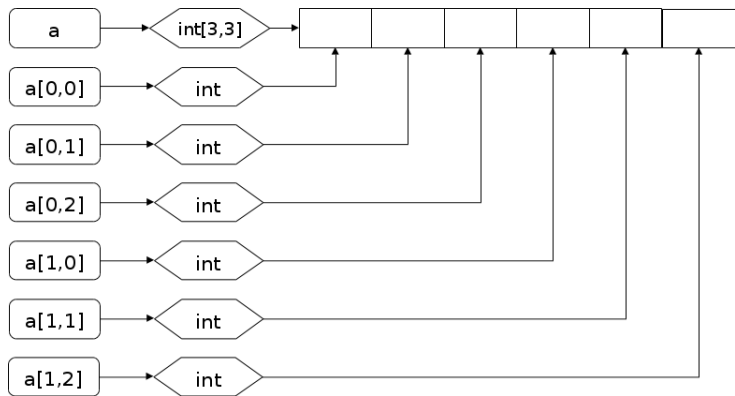
```
for  $i \leftarrow 0$  to  $n - 1$  do  
     $c[i] \leftarrow a[i] + b[i]$ 
```

Costul operațiilor:

operatie	timp(operatie)	
	cost uniform	cost logaritmic
$a[i]$	$O(1)$	$O(i + \log a_i)$
$a[i] \leftarrow v$	$O(1)$	$O(i + \log v)$

unde a este un tablou de dimensiune n , cu valorile $a[0], \dots, a[n-1]$

Tablouri bidimensionale

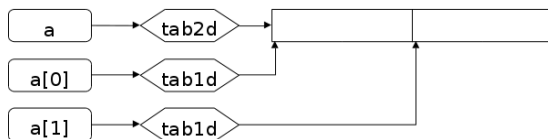


Tablouri bidimensionale

- ▶ Memorie contiguă de $m \times n$ locații
- ▶ Componentele sunt identificate cu ajutorul a 2 indici:
 - ▶ primul indice are valori $\{0, 1, \dots, m - 1\}$
 - ▶ al doilea indice are valori $\{0, 1, \dots, n - 1\}$
 - ▶ variabilele componente :
 $a[0, 0], a[0, 1], \dots, a[0, n - 1], a[1, 0], a[1, 1], \dots, a[1, n - 1], \dots, a[m - 1, 0], a[m - 1, 1], \dots, a[m - 1, n - 1]$
- ▶ Ordinea de memorare a componentelor este dată de ordinea lexicografică a indicilor

Tablouri bidimensionale

- ▶ Cu analogia de la matrici, un tablou 2-dimensional poate fi privit ca un tablou 1-dimensional în care fiecare componentă este un tablou 1-dimensional.
- ▶ Notăție: $a[0][0], a[0][1], \dots, a[0][n-1], \dots, a[m-1][0], a[m-1][1], \dots, a[m-1][n-1]$



Tablouri bidimensionale

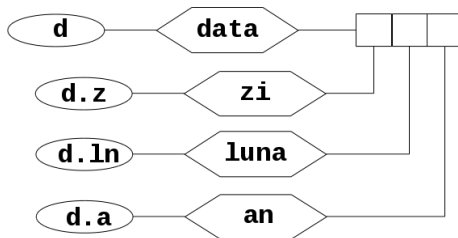
- Operațiile cu tablouri 2-dimensionale se realizează prin intermediul componentelor

```
for  $i \leftarrow 0$  to  $m - 1$  do  
    for  $j \leftarrow 0$  to  $n - 1$  do  
         $c[i, j] \leftarrow 0$   
        for  $k \leftarrow 0$  to  $p - 1$  do  
             $c[i, j] \leftarrow c[i, j] + a[i, k] * b[k, j]$ 
```

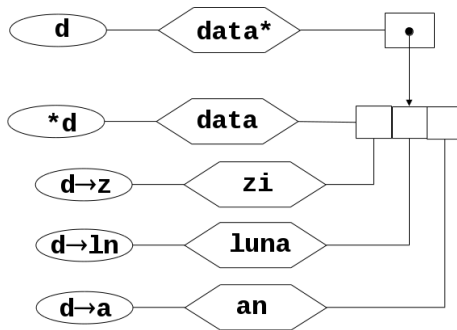
- ▶ Pot fi considerate ca fiind tablouri unidimensionale cu elemente de tip caracter
- ▶ Constantele șir de caracter se notează utilizând “ ”:
“Șir-de-caractere”
- ▶ Operații: Concatenarea, notată cu +:
“un sir” + “alt sir” = “un siralt sir”

- ▶ Structura: ansamblu eterogen de variabile numite câmpuri.
- ▶ Structura are un nume și fiecare câmp are propriul nume și propriul tip.
- ▶ Exemple: o structură pentru a reprezenta puncte în plan are două câmpuri: x și y ; o structură pentru a reprezenta o persoană poate avea trei câmpuri: nume, vârstă, adresă;
- ▶ Numele complet al unui câmp:
punct.x, punct.y
persoana.nume, persoana.varsta, persoana.adresa.strada
dacă p este pointer la persoana: $p \rightarrow varsta$

- ▶ Memoria alocată este o zonă contiguă; elementele sunt memorate în ordinea declarării în structură



Structuri si pointeri



Costul operațiilor:

operatie	timp(operatie)	
	cost uniform	cost logaritmic
$S.x$	$O(1)$	$O(\log S_x)$
$S.x \leftarrow v$	$O(1)$	$O(\log v)$

Execuția unui algoritm

```
x ← 0
i ← 1
while i < 6 do
    x ← x * 10 + i
    i ← i + 2
```

Pasul	Instrucțiunea	i	x
0	$x \leftarrow 0$	—	—
1	$i \leftarrow 1$	—	0
2	$1 < 6$	1	0
3	$x \leftarrow x * 10 + i$	1	0
4	$i \leftarrow i + 2$	1	1
5	$3 < 6$	3	1
6	$x \leftarrow x * 10 + i$	3	1
7	$i \leftarrow i + 2$	3	13
8	$5 < 6$	5	13
9	$x \leftarrow x * 10 + i$	5	13
10	$i \leftarrow i + 2$	5	135
11	$7 < 6$	7	135
12		7	135

Execuția unui algoritm

- **Calcul:** succesiunea de pași elementari determinați de execuția instrucțiunilor ce compun algoritmul.
- **Configurație:** starea memoriei + instrucțiunea curentă;

```
x ← 0  
i ← 1  
while i < 6 do  
    x ← x * 10 + i  
    i ← i + 2
```

Pasul	Instrucțiunea	i	x
0	$x \leftarrow 0$	—	—
1	$i \leftarrow 1$	—	0
2	$1 < 6$	1	0
3	$x \leftarrow x * 10 + i$	1	0
4	$i \leftarrow i + 2$	1	1
5	$3 < 6$	3	1
6	$x \leftarrow x * 10 + i$	3	1
7	$i \leftarrow i + 2$	3	13
8	$5 < 6$	5	13
9	$x \leftarrow x * 10 + i$	5	13
10	$i \leftarrow i + 2$	5	135
11	$7 < 6$	7	135
12		7	135

Execuția unui algoritm

- **Calcul:** succesiunea de pași elementari determinați de execuția instrucțiunilor ce compun algoritmul.
- **Configurație:** starea memoriei + instrucțiunea curentă;
- În exemplul de mai jos calculul este dat de secvența de configurații $(c_0 \mapsto c_1 \mapsto \dots \mapsto c_{12})$.

```
x ← 0
i ← 1
while i < 6 do
    x ← x * 10 + i
    i ← i + 2
```

Pasul	Instrucțiunea	i	x
0	$x \leftarrow 0$	—	—
1	$i \leftarrow 1$	—	0
2	$1 < 6$	1	0
3	$x \leftarrow x * 10 + i$	1	0
4	$i \leftarrow i + 2$	1	1
5	$3 < 6$	3	1
6	$x \leftarrow x * 10 + i$	3	1
7	$i \leftarrow i + 2$	3	13
8	$5 < 6$	5	13
9	$x \leftarrow x * 10 + i$	5	13
10	$i \leftarrow i + 2$	5	135
11	$7 < 6$	7	135
12		7	135