INFORMATICĂ · PYTHON · CLASA A IX-A · PRINCIPII DE ELABORARE A UNUI PROGRAM · LECȚIA 3

Testare, depanare și eficiența algoritmilor

Durată: 100 min Nivel: mate-info · Python Înainte: lecțiile py9-01, py9-02

CE VEI ȘTI DUPĂ LECȚIE

- Construiești un set de teste: cazuri obișnuite, cazuri limită, date nevalide și câte un test pentru fiecare ramură.
- Urmărești valorile variabilelor într-un tabel de trasare și găsești rândul unde programul o ia razna.
- Încadrezi o eroare în categoria corectă: de sintaxă, de execuție sau de logică.
- Estimezi numărul de operații al unui algoritm și îl scrii în notația O: O(1), O(√n), O(n), O(n²).

1. Cum alegi testele

Un program care merge pe exemplul din enunț **nu** este neapărat corect (ai văzut asta în lecția py9-01, la autobuze). Testele bune nu se aleg la întâmplare, ci după criterii.

Problema: biletul la muzeu costă 0 lei pentru copiii sub 7 ani și pentru cei de cel puțin 65 de ani, 5 lei pentru vârstele 7–18 și 15 lei în rest. Vârsta trebuie să fie între 0 și 120.

```
v = int(input())
if v < 0 or v > 120:            # validarea datelor
    print("vârstă invalidă")
elif v < 7 or v >= 65:
    print(0)
elif v <= 18:
    print(5)
else:
    print(15)
```

Criteriu	Ce verifici	Teste (intrare → ieșire)
caz obișnuit	programul face ce trebuie, în mare	`30 → 15`
fiecare ramură	fiecare `if` / `elif` / `else` se execută măcar o dată	`3 → 0`, `12 → 5`, `30 → 15`, `-3 → vârstă invalidă`
cazuri limită	valorile de pe granițe și vecinii lor	`6 → 0`, `7 → 5`, `18 → 5`, `19 → 15`, `64 → 15`, `65 → 0`
extremele intervalului	cea mai mică și cea mai mare valoare permisă	`0 → 0`, `120 → 0`
date nevalide	programul nu acceptă orice	`-3`, `121 → vârstă invalidă`

> **SFAT**
>
> Majoritatea greșelilor stau pe granițe: `<` în loc de `<=`, `range(1, n)` în loc de `range(1, n + 1)`. Pentru fiecare comparație din program, testează valoarea de pe graniță **și** pe cea imediat de lângă ea.

Validarea datelor înseamnă că programul verifică datele înainte să le folosească. Mai sus, `-3` este respins. Dacă însă utilizatorul scrie `abc`, programul se oprește chiar la `int(input())` cu `ValueError`. Cum previi și asta vom vedea mai târziu; deocamdată, notează în analiză **ce date sunt acceptate**.

2. Trasarea: urmărești valorile variabilelor

Când un program dă rezultate greșite, nu ghici: **urmărește valorile**. Programul de mai jos află cea mai mare putere a lui 2 care nu-l depășește pe `n`.

```
n = int(input())
p = 1
k = 0
while p * 2 <= n:
    p *= 2
    k += 1
print(k, p)            # pentru 20 afișează 4 16
```

Tabelul de trasare pentru `n = 20` are câte o coloană pentru fiecare variabilă și pentru condiție:

pas	`p`	`k`	`p * 2 <= n` ?
început	1	0	2 ≤ 20, da
după pasul 1	2	1	4 ≤ 20, da
după pasul 2	4	2	8 ≤ 20, da
după pasul 3	8	3	16 ≤ 20, da
după pasul 4	16	4	32 ≤ 20, **nu**: ieșire

Trei feluri de a urmări valorile pe calculator:

- un `print` temporar în buclă, de exemplu `print("DEBUG", k, p)`, pe care îl ștergi apoi;
- depanatorul din mediul de lucru (în Thonny: *Debug*, apoi pas cu pas cu F7);
- **Python Tutor** (pythontutor.com), care desenează variabilele la fiecare pas.

3. Trei tipuri de erori

Tip	Când apare	Exemplu	Mesajul Python
de sintaxă	înainte de rulare: programul nu pornește deloc	`if x > 3` (lipsește `:`)	`SyntaxError: expected ':'`

Tip	Când apare	Exemplu	Mesajul Python
de execuție	în timpul rulării, la un anumit rând; programul se oprește	`print(10 // 0)`	`ZeroDivisionError: integer division or modulo by zero`
		`"a" + 5`	`TypeError: can only concatenate str (not "int") to str`
		`int("abc")`	`ValueError: invalid literal for int() with base 10: 'abc'`
de logică	programul rulează până la capăt, dar rezultatul e greșit	`range(1, n)` în loc de `range(1, n + 1)`	**niciun mesaj**

Erorile de logică sunt cele mai periculoase, pentru că Python nu te anunță. Doar testele bune și trasarea le găsesc.

> **DE ȘTIUT**
>
> Ce faci, în ordine, când un program nu merge: 1) citești mesajul de eroare (tipul și rândul); 2) găsești cel mai mic test pe care greșește; 3) faci trasarea pe acel test; 4) repari; 5) rulezi din nou **toate** testele, nu doar pe cel reparat.

4. Eficiența: timp și memorie

Două programe corecte pot fi foarte diferite ca resurse:

- **timpul de executare**: câte operații face programul, în funcție de mărimea datelor (`n`);
- **memoria**: câte valori trebuie să țină minte în același timp.

Exemplu: suma `1 + 2 + … + n`, calculată în două feluri.

```
s = 0
for i in range(1, n + 1):     # n adunări
    s += i
```

```
s = n * (n + 1) // 2          # câteva operații, oricât de mare e n
```

Pentru `n = 10 000 000`, pe calculatorul pe care am testat, bucla a durat aproximativ **1,3 secunde**, iar formula a dat rezultatul instantaneu. Timpul exact depinde de calculator, dar raportul dintre ele, nu.

Ambele variante folosesc puține variabile (`n`, `s`, `i`), deci memorie constantă. Dacă ai **memora** toate cele `n` numere ca să le aduni la final, ai avea nevoie de memorie proporțională cu `n`. Pentru multe probleme nu e nevoie să păstrezi datele: le prelucrezi pe măsură ce le citești.

5. Notația O

Nu numărăm exact operațiile, ci **cum crește numărul lor când crește** `n`. Păstrăm doar termenul dominant și renunțăm la constante: `3n + 5` operații înseamnă O(n), iar `n(n - 1) / 2` înseamnă $O(n^2)$.

Ordin	Exemplu tipic	Operații pentru n = 10^6
O(1), constant	formula lui Gauss; ultima cifră `n % 10`	câteva
O(√n)	căutarea divizorilor cu `d * d <= n` (lecția py9-05)	aproximativ 1000
O(n)	o buclă `for i in range(n)`	10^6 (sub o secundă)
$O(n^2)$	două bucle una în alta, fiecare până la `n`	10^{12} (peste o zi!)

Orientativ, Python face câteva milioane până la zece milioane de pași simpli pe secundă; de aici estimările din tabel.

Cum determini ordinul:

Structură	Regula	Exemplu
instrucțiuni fără bucle	O(1)	`a = n // 10 % 10`
o buclă care face `n` pași	O(n)	`for i in range(n): …`
bucle **una după alta**	se adună, deci contează cea mai mare	$O(n) + O(n^2) = O(n^2)$
bucle **una în alta**	se înmulțesc	`n` pași × `n` pași = $O(n^2)$
buclă până la `√n`	O(√n)	`while d * d <= n: d += 1`

Exemplu: câte perechi `(i, j)` cu `i < j` putem forma din `n` elevi?

```
c = 0
for i in range(n):
    for j in range(i + 1, n):
        c += 1
print(c)                # pentru n = 10 afișează 45
```

Bucla interioară face `n - 1`, apoi `n - 2`, …, apoi 0 pași: în total `n(n - 1) / 2`, deci **$O(n^2)$**. Același rezultat îl dă formula `n * (n - 1) // 2`, în O(1).

> **LA BAC**
>
> La Bac apar des cerințe de tipul „algoritm **eficient** din punctul de vedere al timpului de executare”. Înainte să scrii codul, întreabă-te: pot evita o buclă printr-o formulă? Pot opri bucla mai devreme (de exemplu la `√n`)? Pot prelucra datele fără să le memorez pe toate?

6. Verifică-te

1. Programul verifică dacă o notă `x` este de trecere (`x >= 5`). Ce trei valori testezi pe graniță?
2. Ce tip de eroare este fiecare: a) `printț(x)` ; b) `print(x` ; c) media calculată cu `(a + b) / 3` în loc de `/ 2` ?
3. Fă trasarea programului din secțiunea 2 pentru `n = 1` . Ce afișează?
4. Care este ordinul de complexitate pentru `5n + 100` operații? Dar pentru `n² + 1000n` ?
5. Un algoritm O(n) și unul O(n²) rezolvă aceeași problemă. Pentru `n = 1000` , de aproximativ câte ori face mai multe operații al doilea?
6. Ce ordin are o buclă `for i in range(1, n + 1)` urmată de o altă buclă `for j in range(1, n + 1)` (nu una în alta)?

Răspunsuri

1. `4` , `5` și `6` (granița și cei doi vecini); plus extremele, de exemplu `1` și `10` .
2. Prima este de execuție (`NameError` : `printț` nu există, iar asta se descoperă abia când se ajunge la acel rând); a doua, de sintaxă (paranteză neînchisă); a treia, de logică.
3. Condiția `2 <= 1` este falsă de la început, deci bucla nu se execută: afișează `0 1` .
4. O(n), respectiv O(n²).
5. De aproximativ 1000 de ori (n² / n = n).
6. O(n): `n + n = 2n` pași.

Probleme de lucru în clasă (pbinfo)

Le rezolvăm împreună în ședință, în ordinea de mai jos.

- [ ] `#1360` **suma-gauss** — ușoară
  https://www.pbinfo.ro/probleme/1360/suma-gauss
- [ ] `#1255` **lipsa** — ușoară
  https://www.pbinfo.ro/probleme/1255/lipsa

Temă pentru acasă (pbinfo)

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

- [ ] `#3609` **countmod10** — ușoară
  https://www.pbinfo.ro/probleme/3609/countmod10
- [ ] `#3978` **sabc** — medie
  https://www.pbinfo.ro/probleme/3978/sabc