



# Faza LOCALĂ a olimpiadei de INFORMATICĂ

Vă prezentăm în continuare soluțiile celor nouă probleme propuse spre rezolvare la ediția 2003 a fazei locale din județul Cluj a olimpiadei de informatică.

## P020307: Ecuație

Ecuația  $x^2 - y^2 = k$  poate fi scrisă sub forma  $(x + y) \cdot (x - y) = k$ . Datorită faptului că  $x$  și  $y$  sunt valori întregi,  $x + y$  și  $x - y$  sunt, de asemenea, valori întregi.

Vom considera toate posibilitățile de a scrie valoarea  $k$  ca produs de numere întregi  $k = p \cdot q$ . Pentru fiecare posibilitate va trebui să rezolvăm sistemul de ecuații:

$$\begin{cases} x + y = p \\ x - y = q \end{cases}$$

Soluțiile acestui sistem de ecuații sunt:

$$\begin{cases} x = \frac{p+q}{2} \\ y = \frac{p-q}{2} \end{cases}$$

Datorită faptului că soluțiile trebuie să fie numere întregi, se observă că valorile  $p$  și  $q$  trebuie să aibă aceeași paritate.

Produsul a două numere de aceeași paritate are întotdeauna paritatea celor două numere; așadar, pentru identificarea numerelor  $p$  și  $q$  vom considera toate numerele mai mici decât  $k$  care au aceeași paritate cu  $k$ .

Vom considera toate variantele posibile pentru valoarea  $p$  și vom determina, pentru fiecare dintre acestea, valoarea  $q$  folosind formula  $q = k / p$ .

Se observă că dacă  $x$  și  $y$  sunt soluții ale ecuației, atunci perechile  $(-x, y)$ ,  $(x, -y)$  și  $(-x, -y)$  sunt și ele soluții. Așadar, va trebui să determinăm doar soluțiile pozitive și în momentul identificării unei astfel de soluții să le generăm și pe celelalte trei.

O situație specială este cazul în care una dintre valorile  $x$  și  $y$  este 0. Nu există posibilitatea ca ambele valori să fie 0, deoarece  $k$  este întotdeauna un număr nenul.

În această situație, doar două dintre cele patru perechi duc la obținerea unor soluții distincte. Din acest motiv vom genera doar perechea  $(-x, -y)$  care este, cu siguranță,

diferită de perechea  $(x, y)$  indiferent care dintre cele două valori este 0.

Se observă că este suficient să luăm în considerare doar divizorii  $p$  pentru care  $p \leq \sqrt{k}$  deoarece perechea  $(p, q)$  duce la obținerea acelorași soluții ca și perechea  $(q, p)$ . Așadar, vom lua în considerare doar divizorii lui  $k$  care sunt mai mici sau egali cu rădăcina pătrată a acestui număr.

## Analiza complexității

Intrarea constă în citirea unui singur număr, așadar ordinul de complexitate al operației de citire a datelor este  $O(1)$ .

Datorită faptului că pentru fiecare divizor determinat efectuăm un număr constant de operații și vom verifica cel mult  $\sqrt{k} / 2$  divizori, ordinul de complexitate al operației de determinare a soluțiilor ecuației este  $O(\sqrt{k})$ .

Datele de ieșire sunt scrise pe parcursul determinării soluțiilor, deci operația de scriere a acestora nu consumă timp suplimentar.

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate  $O(1) + O(\sqrt{k}) = O(\sqrt{k})$ .

Dificultate

10

Algoritmi	★ ★ ★ ★ ★
Structuri de date	★ ★ ★ ★ ★
Originalitate	★ ★ ★ ★ ★
Cunoștințe	★ ★ ★ ★ ★
Implementare	★ ★ ★ ★ ★

[www.ginfo.ro/revista/13\\_3/surse/ecuatie.cpp](http://www.ginfo.ro/revista/13_3/surse/ecuatie.cpp)

## P020308: Număr

Pentru a determina cel mai mic număr par care este format din anumite cifre date vom ordona crescător aceste cifre.

Pentru ca numărul obținut să nu înceapă cu zerouri nesemnificative vom muta cea mai mică cifră pe prima poziție.

Pentru ca numărul să fie par vom muta cea mai mare cifră pară pe ultima poziție cu excepția situației în care aceasta a fost mutată pe prima poziție. În cazul în care cifra respectivă a ajuns pe prima poziție știm că, cu excepția ei, numărul nu mai conține decât cifre impare și, cu siguranță, zerouri. În acest caz vom muta un zero pe ultima poziție.

Problema nu are soluție decât în situația în care toate cifrele numărului dat sunt impare.

În final vom scrie în fișierul de ieșire numărul obținut sau mesajul NU EXISTA.

### Analiza complexității

Vom considera că numărul citit are  $k$  cifre; așadar, ordinul de complexitate al operației de citire a datelor de intrare este  $O(k)$ .

Operația de sortare a cifrelor poate fi realizată în timp liniar-logaritmice dar, datorită numărului mic de cifre, este suficientă folosirea unui algoritm de sortare care are ordinul de complexitate  $O(k^2)$ .

Identificarea și mutarea pe prima poziție a primei cifre diferite de zero se realizează în timp liniar.

Operația de identificare și mutare pe ultima poziție a cifrei pare se realizează tot în timp liniar.

Scrierea datelor de ieșire implică scrierea a  $k$  cifre, deci și această operație are ordinul de complexitate  $O(k)$ .

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este  $O(k) + O(k^2) + O(k) + O(k) + O(k) = O(k^2)$ .

Un număr  $N$  are întotdeauna  $\lceil \lg N \rceil + 1$  cifre; așadar ordinul de complexitate al algoritmului, exprimat în funcție de  $N$ , este  $O(\log^2 N)$ .

### Dificultate

10

Algoritmi	★ ★ ★ ★ ★
Structuri de date	★ ★ ★ ★ ★
Originalitate	★ ★ ★ ★ ★
Cunoștințe	★ ★ ★ ★ ★
Implementare	★ ★ ★ ★ ★

[www.ginfo.ro/revista/13\\_3/surse/numar.cpp](http://www.ginfo.ro/revista/13_3/surse/numar.cpp)

### P020309: Suma

Vom calcula mai întâi suma  $S_0$  a cifrelor numărului dat. Restul împărțirii la 10 al acestei sume este ultima cifră a sumei căutate. Câtul împărțirii la 10 al valorii  $S_0$  este adunat la valoarea  $S_0$ , obținându-se valoarea  $S_1$ . Restul împărțirii la 10 al acestei sume este penultima cifră a sumei căutate. Câtul împărțirii la 10 al valorii  $S_1$  este adunat la valoarea  $S_0$ , obținându-se valoarea  $S_2$ .

În general, câtul împărțirii la 10 a valorii  $S_i$  este adunat la valoarea  $S_0$ , obținându-se valoarea  $S_{i+1}$ . Restul împărțirii la 10 a unei valori  $S_i$  reprezintă o cifră a sumei căutate.

Dacă valoarea  $S_0$  este mai mare decât 9 și numărul dat are  $k$  cifre, atunci rezultatul va avea  $k + 1$  cifre; așadar,

dacă ultima sumă  $S_{k-1}$  este mai mare decât 10, atunci câtul împărțirii sale la 10 este prima cifră a rezultatului.

### Analiza complexității

Vom considera că numărul citit are  $k$  cifre; așadar, ordinul de complexitate al operației de citire a datelor de intrare este  $O(k)$ .

Determinarea sumei cifrelor numărului se realizează printr-o simplă parcurgere a acestora, deci ordinul de complexitate al acestei operații este  $O(k)$ .

Determinarea cifrelor rezultatului se realizează tot printr-o singură parcurgere, deci și această operație are ordinul de complexitate  $O(k)$ .

Scrierea datelor de ieșire implică scrierea a  $k$  cifre; așadar și această operație are ordinul de complexitate  $O(k)$ .

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este  $O(k) + O(k) + O(k) + O(k) = O(k)$ .

Un număr  $N$  are întotdeauna  $\lceil \lg N \rceil + 1$  cifre; așadar ordinul de complexitate al algoritmului, exprimat în funcție de  $N$ , este  $O(\log N)$ .

### Dificultate

06

Algoritmi	★ ★ ★ ★ ★
Structuri de date	★ ★ ★ ★ ★
Originalitate	★ ★ ★ ★ ★
Cunoștințe	★ ★ ★ ★ ★
Implementare	★ ★ ★ ★ ★

[www.ginfo.ro/revista/13\\_3/surse/suma.cpp](http://www.ginfo.ro/revista/13_3/surse/suma.cpp)

### P020310: Numere

Definirea recursivă a problemei ilustrează foarte clar modul de rezolvare a acesteia. Vom crea două rutine recursive, una care generează numerele din prima jumătate a șirului și una care generează numerele din a doua jumătate a șirului.

Pentru prima parte a șirului, la fiecare pas  $i$  se vor genera numere cuprinse între  $2^i$  și  $3 \cdot 2^{i-1} - 1$ . Pentru a doua parte a șirului, la fiecare pas  $i$  se vor genera numere cuprinse între  $3 \cdot 2^{i-1}$  și  $2^{i+1} - 1$ .

Diferența principală a celor două rutine recursive este poziția apelului recursiv față de generarea elementelor. Pentru prima parte a șirului, apelul va apărea după generare, iar pentru a doua parte a șirului, apelul va apărea înaintea generării.

### Analiza complexității

Datele de intrare constau într-un singur număr, deci ordinul de complexitate al operației de citire este  $O(1)$ .

Întotdeauna vor fi generate  $2^N$  numere, deci ordinul de complexitate al operației de generare a acestora este  $O(2^N)$ .

Scrierea datelor în fișierul de ieșire este realizată pe parcursul determinării acestora, deci operația de scriere nu consumă timp suplimentar.



Soluții



În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este  $O(1) + O(2^N) = O(2^N)$ .

**Dificultate** 08

Algoritmi	☆☆☆☆☆
Structuri de date	☆☆☆☆☆
Originalitate	☆☆☆☆☆
Cunoștințe	☆☆☆☆☆
Implementare	☆☆☆☆☆

[www.ginfo.ro/revista/13\\_3/surse/numere.cpp](http://www.ginfo.ro/revista/13_3/surse/numere.cpp)

### P020311: Triunghi

Pentru fiecare punct dat va trebui să verificăm dacă se află sau nu în interiorul triunghiului considerat și să numărăm punctele din interiorul triunghiului.

O variantă mai simplă de rezolvare se bazează pe faptul că dacă un punct se află în interiorul triunghiului, pe una din laturi sau este un vârf, atunci suma ariilor celor trei triunghiuri determinate de punctul respectiv și perechi de vârfuri ale triunghiului dat este egală cu aria triunghiului dat (vezi figura 1).

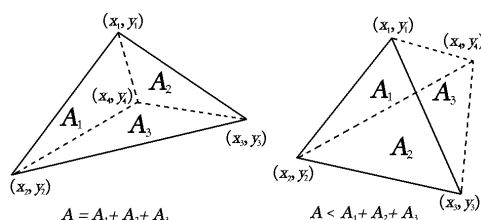


Figura 1

Așadar, pentru fiecare punct în parte vom determina ariile triunghiurilor determinate de punctul respectiv și două vârfuri ale triunghiului. Dacă suma celor trei arii este egală cu aria triunghiului, atunci punctul se află în interiorul triunghiului.

Se observă că dacă punctul este un vârf al triunghiului, două dintre cele trei arii vor fi egale cu 0, iar cea de-a treia arie va fi chiar aria triunghiului dat.

În cazul în care vârful se află pe una dintre laturi (dar nu este un vârf al triunghiului), atunci una dintre arii va fi egală cu 0.

Dacă se cunosc coordonatele vârfurilor unui triunghi, atunci formula de calcul a ariei acestuia este:

$$\left| \frac{1}{2} \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \right|$$

În final vom scrie în fișierul de ieșire numărul punctelor aflate în interiorul triunghiului.

### Analiza complexității

Citirea coordonatelor vârfurilor triunghiului constă în citirea a șase numere întregi, deci se realizează în timp constant.

Verificarea faptului dacă un punct se află sau nu în interiorul triunghiului considerat se realizează cu ajutorul unui număr constant de operații aritmetice. Această operație poate fi realizată pe măsura citirii coordonatelor punctelor. Așadar, ordinul de complexitate al operațiilor de citire a datelor și determinare a soluției este  $O(N)$ .

Datele de ieșire constau într-un singur număr, așadar ordinul de complexitate al operației de scriere a acestora este  $O(1)$ .

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate  $O(1) + O(N) + O(1) = O(N)$ .

**Dificultate** 12

Algoritmi	☆☆☆☆☆
Structuri de date	☆☆☆☆☆
Originalitate	☆☆☆☆☆
Cunoștințe	☆☆☆☆☆
Implementare	☆☆☆☆☆

[www.ginfo.ro/revista/13\\_3/surse/triunghi.cpp](http://www.ginfo.ro/revista/13_3/surse/triunghi.cpp)

### P020312: Găuri

Vom determina toate regiunile formate din zerouri și vom eticheta elementele acestor regiuni folosind numere naturale, începând cu 1.

Pentru fiecare regiune determinată vom păstra dimensiunile acestora (numărul de elemente) într-un vector ai cărui indici sunt numerele folosite pentru etichetarea dimensiunilor regiunilor. Pentru a determina aceste regiuni vom folosi un simplu algoritm de umplere.

O gaură poate fi privită ca fiind o regiune formată din zerouri care nu conține nici un element aflat pe marginea matricei. Vom parcurge cele patru margini ale matricei și vom seta la 0 dimensiunile tuturor regiunilor care conțin elemente de pe aceste margini. Astfel, practic am eliminat toate regiunile care nu sunt găuri.

În final vom parcurge vectorul dimensiunilor și vom determina elementul care are valoarea maximă. Valoarea acestui element va fi scrisă în fișierul de ieșire.

### Analiza complexității

Datele de intrare constau în dimensiunile matricei și valorile tuturor elementelor acesteia. Așadar ordinul de complexitate al operației de citire este  $O(M \cdot N)$ .

Operația de determinare a regiunilor formate din zerouri folosind un algoritm de umplere are ordinul de complexitate  $O(M \cdot N)$ .

Pentru a elimina regiunile care nu sunt găuri vom parcurge cele patru margini ale matricei, așadar această operație are ordinul de complexitate  $O(M) + O(N) + O(M) + O(N) = O(M + N)$ .

În total pot exista cel mult  $[M \cdot N / 2] + 1$  regiuni (dacă matricea are aspectul unei table de șah). Așadar, operația de determinare a dimensiunii celei mai mari regiuni are ordinul de complexitate  $O(M \cdot N)$ .

Datele de ieșire constau într-un singur număr, operația de scriere a acestuia având ordinul de complexitate  $O(1)$ .

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate  $O(M \cdot N) + O(M \cdot N) + O(M + N) + O(M \cdot N) + O(1) = O(M \cdot N)$ .

Dificultate						18
Algoritmi	★	★	★	★	★	
Structuri de date	★	★	★	★	★	
Originalitate	★	★	★	★	★	
Cunoștințe	★	★	★	★	★	
Implementare	★	★	★	★	★	

[www.ginfo.ro/revista/13\\_3/surse/gauri.cpp](http://www.ginfo.ro/revista/13_3/surse/gauri.cpp)

### P020313: Exponent

Pentru a determina valoarea  $k$ , va trebui, mai întâi să descompunem numărul  $n$  în factori primi.

Pentru fiecare factor  $p$  care apare în descompunerea lui  $n$ , va trebui să determinăm puterea la care apare acest factor în descompunerea în factori primi a valorii  $m!$ .

Vom considera că  $p$  apare în descompunerea în factori primi a numărului  $n$  de  $u$  ori; cu alte cuvinte  $n$  poate fi scris sub forma  $n = p^u \cdot q$ , unde  $q$  este un număr întreg.

Vom determina acum puterea  $v$  la care apare  $p$  în descompunerea în factori primi a numărului  $m!$ . Valoarea  $v$  poate fi obținută astfel:

- pentru fiecare multiplu al lui  $p$  mai mic sau egal cu  $m$ , numărul  $p$  apare o dată în descompunerea lui  $m!$ ;
- pentru fiecare multiplu al lui  $p^2$  mai mic sau egal cu  $m$ , numărul  $p$  apare încă o dată în descompunerea lui  $m!$ ;
- în general, pentru fiecare multiplu al lui  $p^i$  mai mic sau egal cu  $m$ , numărul  $p$  apare încă o dată în descompunerea lui  $m!$ .

Expresia pentru fiecare multiplu al lui  $p^2$  mai mic sau egal cu  $m$  este echivalentă cu expresia pentru fiecare multiplu al lui  $p$  mai mic sau egal cu  $[m / p]$ .

În general, expresia pentru fiecare multiplu al lui  $p^i$  mai mic sau egal cu  $m$  este echivalentă cu expresia pentru fiecare multiplu al lui  $p$  mai mic sau egal cu  $[m / p^{i-1}]$ .

Ca urmare, valoarea  $v$  la care apare  $p$  în descompunerea în factori primi a numărului  $m!$  poate fi determinată folosind următorul algoritm:

```
v ← 0
cât timp m > 0 execută
    v ← v + [m / p]
    m ← [m / p]
sfârșit cât timp
```

În aceste condiții, pentru ca  $n^k = p^{u \cdot k} \cdot q^k$  să fie divizor al numărului  $m!$ , trebuie să avem  $k \leq [v / u]$ .

Vom determina valorile maxime ale numărului  $k$  pentru toți divizorii  $p$  și o vom alege pe cea mai mică dintre acestea.

### Analiza complexității

Intrarea constă în citirea a două numere, așadar ordinul de complexitate al operației de citire a datelor este  $O(1)$ .

Operația de descompunere în factori primi a unui număr natural  $n$  are ordinul de complexitate  $O(\sqrt{n})$ .

Numărul factorilor primi ai valorii  $n$  poate fi cel mult  $[\log_2 n]$ . Pentru un număr  $m$  dat, folosirea algoritmului prezentat pentru determinarea puterii la care apare  $p$  în descompunerea în factori primi a valorii  $m!$  necesită parcurgerea a  $[\log_p m]$  pași.

Ca urmare, operația de determinare a valorii  $k$  are ordinul de complexitate  $O(\log m) \cdot O(\log n) = O(\log m \cdot \log n)$ .

Ieșirea constă într-un singur număr, așadar ordinul de complexitate al operației de scriere a datelor este  $O(1)$ .

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate  $O(1) + O(\sqrt{n}) + O(\log m \cdot \log n) + O(1) = O(\sqrt{n} + \log m \cdot \log n)$ .

Dificultate						18
Algoritmi	★	★	★	★	★	
Structuri de date	★	★	★	★	★	
Originalitate	★	★	★	★	★	
Cunoștințe	★	★	★	★	★	
Implementare	★	★	★	★	★	

[www.ginfo.ro/revista/13\\_3/surse/expo.cpp](http://www.ginfo.ro/revista/13_3/surse/expo.cpp)

### P020314: Bastioane

Se observă că pentru primul bastion va trebui să alegem o submulțime a soldaților care pot duce un număr total de pachete divizibil cu valoarea  $n$ . Pentru al doilea bastion, dintre cei  $k$  soldați rămași va trebui să alegem o submulțime a soldaților care pot duce un număr total de pachete divizibil cu valoarea  $k$ . Așadar, pentru al doilea bastion avem aceeași problemă pentru o valoarea  $k$  strict mai mică decât  $n$ .

Practic, la fiecare pas vom rezolva aceeași problemă pentru valori din ce în ce mai mici până în momentul în care nu mai există nici un soldat.

Așadar, la fiecare pas va trebui să determinăm o submulțime a unei mulțimi cu  $k$  elemente a cărei sumă să fie divizibilă cu  $k$ . În acest scop vom folosi principiul cutiei lui Dirichlet.

Presupunem că, la un moment dat, cei  $k$  soldați rămași pot duce  $a_1, a_2, \dots$ , respectiv  $a_k$  pachete. Vom calcula sumele  $S_i = a_1 + a_2 + \dots + a_i$  folosind formula recurentă  $S_1 = a_1$  și  $S_i = a_i + S_{i-1}$  pentru toate valorile  $i$  cuprinse între 2 și  $k$ .

Potrivit principiului cutiei lui Dirichlet, dacă resturile împărțirii valorilor  $S_i$  la  $k$  sunt distincte, atunci unul dintre aceste resturi este 0.

Potrivit aceluiași principiu, dacă resturile nu sunt distincte, atunci două dintre acestea sunt egale. Dacă resturile împărțirii valorilor  $S_u$  și  $S_v$  sunt egale, atunci restul împărțirii valorii  $S_v - S_u$  la  $k$  este 0. Așadar, valoarea  $a_{u+1} + a_{u+2} + \dots + a_v$  va fi divizibilă cu  $k$ .





În concluzie, va exista întotdeauna posibilitatea de a alege o submulțime a cărei sumă să fie divizibilă cu  $k$ . După alegerea acestei submulțimi, dacă ea nu conține toate cele  $k$  elemente, va trebui să rezolvăm aceeași problemă pentru o valoare strict mai mică decât  $k$  (elementele mulțimii respective sunt eliminate).

### Analiza complexității

Intrarea constă în citirea valorilor corespunzătoare celor  $n$  soldați, deci ordinul de complexitate al operației de citire a datelor este  $O(n)$ .

La fiecare pas va trebui să determinăm sumele  $S_i$ , operație care are ordinul de complexitate  $O(k)$ , unde  $k = O(n)$  este numărul soldaților rămași la pasul respectiv.

Pe parcursul determinării sumelor vom păstra și resturile împărțirii lor la  $k$ , așadar vom putea identifica elementele care vor fi eliminate pe parcursul determinării sumelor. Eliminarea propriu-zisă este realizată în timp liniar. În cazul cel mai defavorabil la fiecare pas vom elimina un singur element, așadar putem avea cel mult  $n$  pași. Ca urmare, ordinul de complexitate al operației de repartizare a soldaților este  $O(n) \cdot O(n) = O(n^2)$ .

Datele de ieșire sunt scrise pe parcursul determinării soluțiilor, deci operația de scriere a acestora nu consumă timp suplimentar.

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate  $O(n) + O(n^2) = O(n^2)$ .

Dificultate 18					
Algoritmi	★	★	★	★	★
Structuri de date	★	★	★	★	★
Originalitate	★	★	★	★	★
Cunoștințe	★	★	★	★	★
Implementare	★	★	★	★	★

[www.ginfo.ro/revista/13\\_3/surse/soldati.cpp](http://www.ginfo.ro/revista/13_3/surse/soldati.cpp)

### P020315: Graf

Pentru a determina cel mai îndepărtat nod față de cel identificat prin valoarea 1 va trebui să realizăm o parcurgere în lățime (*Breadth First - BF*) a grafului.

O astfel de parcurgere implică atingerea nodurilor mai apropiate de nodul 1 înaintea atingerii nodurilor mai îndepărtate.

Ca urmare, ultimul nod atins de o astfel de parcurgere va fi întotdeauna unul dintre cele mai îndepărtate noduri, chiar dacă există și alte noduri aflate la aceeași distanță.

După determinarea ultimului nod parcurs vom scrie în fișierul de ieșire numărul său de ordine.

### Analiza complexității

Intrarea constă în citirea informațiilor referitoare la cele  $M$  muchii ale grafului, deci ordinul de complexitate al operației de citire a datelor este  $O(m)$ . Pe parcursul citirii este construită și structura de date în care va fi păstrat graful, deci nu este consumat timp suplimentar pentru generarea acestuia.

Operația de parcurgerea în lățime a unui graf are ordinul de complexitate  $O(m)$ .

Datele de ieșire constau într-un singur număr, așadar ordinul de complexitate al operației de scriere este  $O(1)$ .

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate  $O(m) + O(m) + O(1) = O(m)$ .

Dificultate 20					
Algoritmi	★	★	★	★	★
Structuri de date	★	★	★	★	★
Originalitate	★	★	★	★	★
Cunoștințe	★	★	★	★	★
Implementare	★	★	★	★	★

[www.ginfo.ro/revista/13\\_3/surse/graf.cpp](http://www.ginfo.ro/revista/13_3/surse/graf.cpp)

## Punctajele GInfo

Mai mulți cititori și-au exprimat dorința de a afla modul în care sunt acordate punctajele de către redacția GInfo în momentul prezentării soluției unei probleme.

După cum probabil v-ați dat seama, sunt luate în considerare cinci criterii și anume: dificultatea algoritmilor folosiți, complexitatea structurilor de date utilizate, originalitatea problemei, cunoștințele teoretice necesare rezolvării problemei și dificultățile întâlnite în timpul implementării propriu-zise.

Pentru fiecare dintre aceste criterii este acordat un punctaj cuprins între 0 și 10 în funcție

de diverse aspecte pe care nu le putem prezenta din motive de spațiu.

Pentru fiecare criteriu punctajul este un număr real care va fi rotunjit la cel mai apropiat număr întreg. Numărul steluțelor acordate este obținut prin împărțirea la 2 a acestui număr (așadar pentru fiecare "unitate" se acordă o "jumătate de steluță"). În calculul punctajului final se iau în considerare valorile reale, nu cele rotunjite. Formula de calcul este următoarea:

$$\begin{aligned} \text{Dificultate} = & \text{Algoritmi} \cdot 3 + \text{Structuri de date} \cdot 2,5 + \\ & + \text{Originalitate} + \text{Cunoștințe} \cdot 2,5 + \\ & + \text{Implementare} \end{aligned}$$