BURSELE AGORA 2004/ 2005. RUNDELE #01-#12

Vă prezentăm în continuare soluțiile problemelor propuse spre rezolvare la primele 12 runde a ediției a șasea a concursului de programare organizat de revista noastră.

P050225: Alegeri

Această problemă face parte din categoria acelora care necesită o prelucrare a datelor pe segmente de lungime variabilă. Lungimea unui astfel de subșir depinde de valoarea elementelor șirului.

De altfel, datorită lungimii maxime posibile este exclus să citim toate datele și să le păstrăm în memorie. În concluzie, vom acționa pe baza valorii unui singur număr.

Numărul curent îl notăm prin *nr*, iar candidatul curent cu *candidat*. Dacă un număr citit din fișier este egal cu candidat, mărim contorul *voturi* în care ținem evidența aparițiilor candidatului. Dacă numărul citit diferă de candidat, indiferent de valoarea lui, scădem 1 din contorul *voturi*.

După fiecare astfel de scădere verificăm contorul *voturi*, deoarece în momentul în care acesta devine 0, înseamnă că s-a sfârșit un segment și începe altul cu un candidat nou.

În final, va trebui să verificăm dacă elful corespunzător ultimului segment deține îmtr-adevăr mai mult dejumătate din voturi.

Prezentăm în continuare versiunea în pseudocod a algoritmului prezentat.

```
algoritm Alegeri
  citește nr
  candidat ← nr
  voturi ← 1
  cât timp nu urmează marcajul
        de sfârșit de fișier execută
    citește nr
    dacă nr = candidat atunci
      voturi ← voturi + 1
    altfel
      voturi ← voturi - 1
      dacă voturi = 0 atunci
        candidat ← nr
        voturi ← 1
      sfârșit dacă
    sfârșit dacă
  sfârșit cât timp
  voturi ← 0
  n ← 0
  cât timp nu urmează marcajul
        de sfârșit de fișier execută
    citește nr
    n ← n + 1
    dacă nr = candidat atunci
      voturi ← voturi + 1
    sfârșit dacă
  sfârșit cât timp
  dacă voturi > [n/2] atunci
    scrie candidat
  altfel
    scrie -1
  sfârșit dacă
sfârșit algoritm
```

P050226: Paradă

Pentru a rezolva această problemă este suficient să determinăm, la fiecare pas, perechile de elfi care se află față în față.

Doi elfi, aflați pe pozițiile *i* și *i* + 1 se vor afla față în față dacă cel de pe poziția *i* privește spre dreapta și cel de pe poziția *i* + 1 privește spre stânga.

După fiecare pas vom realiza întoarcerile necesare și ne vom opri în momentul în care, după un pas, nu avem nevoie de nici o întoarcere.

P050227: Aliniere

Pentru fiecare războinic vom determina cel mai lung subșir strict crescător (din punct de vedere al înălțimii) de războinici care se termină cu el, respectiv cel mai lung subșir strict descrescător de războinici care urmează după el.

După această operație, vom determina soldatul pentru care suma lungimilor celor două șiruri este maximă.

Chiar dacă s-ar părea că în acest mod am găsit soluția problemei, mai există o posibilitate de a mări numărul războinicilor care rămân în șir.

Să considerăm războinicul cel mai înalt în șirul rămas (cel căruia îi corespunde suma maximă). Acesta poate privi fie spre stânga, fie spre dreapta șirului.

Din aceste motive, la stânga sau la dreapta sa poate să se afle un războinic de aceeași înălțime; unul dintre cei doi va privi spre dreapta, iar celălalt spre stânga.

Totuși, nu putem alege orice războinic cu aceeași înălțime, ci doar unul pentru care lungimea șirului strict crescător (dacă se află spre stânga) sau a celui strict descrescător (dacă se află spre dreapta) este aceeași cu lungimea corespunzătoare șirului strict crescător, respectiv strict descrescător, corespunzătoare celui mai înalt războinic dintre cei care au rămas în șir.

După identificarea celor doi războinici de înălțimi egale (sau determinarea faptului că nu există o pereche de acest gen care să respecte condițiile date) se marchează toți războinicii din cele două subșiruri.

Ceilalți războinici vor trebui să părăsească formația.

Analiza complexității

Citirea datelor de intrare se realizează în timp liniar, deci ordinul de complexitate al acestei operații este $O(n)$.

Chiar dacă există algoritmi eficienți (care rulează în timp liniar-logaritmic) de determinare a celui mai lung subșir ordonat, timpul de execuție admis ne permite folosirea unui algoritm simplu, cu ordinul de complexitate $O(n^2)$.

Acesta va fi aplicat de două ori, după care se va căuta valoarea maximă a sumei lungimilor șirurilor corespunzătoare unui soldat; așadar identificarea soldatului care poate privi în ambele direcții este o operație cu ordinul de complexitate $O(n^2) + O(n^2) + O(n) = O(n^2)$.

Urmează eventuala identificare a unui alt războinic de aceeași înălțime care respectă condițiile referitoare la lungimile subșirurilor.

În vederea acestei identificări se parcurge șirul războinicilor de la războinicul identificat anterior spre extremități; operația necesită un timp liniar.

Determinarea celor două subșiruri se realizează în timp liniar dacă, în momentul construirii celor două subșiruri, se păstrează predecesorul, respectiv succesorul fiecărui războinic. În timpul parcurgerii subșirurilor sunt marcați războinicii care rămân în formație.

Pentru scrierea datelor de ieșire se parcurge șirul marcajelor și sunt identificați războinicii care părăsesc formația. Ordinul de complexitate al acestei operații este $O(n)$.

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

P050228: Culori

Vom construi un graf bipartit în care n noduri vor reprezenta pătrățelele, iar alte m noduri vor reprezenta cerculețele.

Va exista o muchie de la un nod corespunzător unui pătrățel la un nod corespunzător unui cerculeț dacă există cel puțin o culoare care apare atât pe cerculeț, cât și pe pătrățel.

După construirea grafului bipartit vom determina un cuplaj maxim în acest graf și pe baza sa vom stabili pătrățelul ales pentru fiecare cerculeț.

În final vom scrie rezultatele în fișierul de ieșire.

Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate $O(m + n)$, deoarece se citesc date referitoare la n pătrățele și m cerculețe. Considerăm că citirea datelor referitoare la un pătrățel sau un cerculeț se realizează în timp constant, deoarece pe acestea sunt desenate cel mult zece benzi.

După citirea datelor va trebui să creăm graful bipartit. Acesta va conține cel mult $m \cdot n$ muchii. Verificarea existenței unei muchii între un nod corespunzător unui pătrățel la un nod corespunzător unui cerculeț se realizează în timp constant, deoarece va trebui să efectuăm cel mult o sută de comparații. Așadar, operația de creare a grafului bipartit va avea ordinul de complexitate $O(m \cdot n)$.

După determinarea grafului bipartit va trebui să determinăm cuplajul cu ajutorul unui flux. Valoarea maximă a fluxului va fi cel mult egală cu minimul valorilor m și n.

Rețeaua de transport va conține cel mult $m \cdot n + m + n$ arce deoarece sunt introduse n arce care pleacă de la sursă și m arce care ajung la destinație.

Ca urmare, ordinul de complexitate al operației de determinare a cuplajului va fi $O(\mathbf{min}(m, n) \cdot (m \cdot n + m + n + m + m)) = O(\mathbf{min}(m, n) \cdot (m \cdot n))$. În cazul în care avem mai multe pătrățele decât cercuri, ordinul de complexitate va fi $O(m^2 \cdot n)$, iar în caz contrar va fi $O(m \cdot n^2)$.

Pentru afișarea soluției va trebui să parcurgem cele cel mult $m \cdot n$ muchii ale grafului bipartit, deci această operație are ordinul de complexitate $O(m \cdot n)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(m + n) + O(m \cdot n) + O(\mathbf{min}(m, n) \cdot (m \cdot n)) + O(m \cdot n) = O(\mathbf{min}(m, n) \cdot (m \cdot n))$.

P050229: Copaci

Problema se reduce la a determina o pereche de puncte aflate la distanță minimă. După determinarea perechii de puncte vom scrie în fișierul de ieșire coordonatele unuia dintre cele două puncte.

Pentru aceasta va trebui doar să utilizăm algoritmul de determinare a celei mai apropiate perechi de puncte.

Analiza complexității

Citirea datelor de intrare implică citirea celor n perechi de coordonate ale copacilor care vor fi plantați, așadar ordinul de complexitate al operației de citire a datelor de intrare este $O(n)$.

Algoritmul de determinare a celei mai apropiate perechi de puncte are ordinul de complexitate $O(n \cdot \log n)$.

Operația de scriere a datelor în fișierul de ieșire are ordinul de complexitate $O(1)$ deoarece vom scrie doar două numere.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(n) + O(n \cdot \log n) + O(1) = O(n \cdot \log n)$.

P050230: Risc

În vederea rezolvării acestei probleme vom crea o rețea de transport astfel: introducem artificial o sursă și o

destinație; pe lângă aceste două noduri, rețeaua va mai conține alte $2 \cdot n$ noduri (dublul numărului de tipuri de fructe).

Vom uni pe rând sursa de fiecare dintre primele n noduri (reprezentând tipurile de fructe, numerotate de la 1 la n), considerând că arcele introduse au capacitatea 1 și costul 0. Celelalte n noduri (care reprezintă cele n grămezi, și care sunt numerotate de la $n + 1$ la $2 \cdot n$) vor fi unite cu destinația prin arce de capacitate 1 și cost 0.

Va exista un arc de la un nod i ($1 \le i \le n$) la un nod j ($n + 1 \le j \le 2 \cdot n$) dacă și numai dacă fructul de tipul i se află în grămada $j - n$ după amestecare. Acest arc va avea capacitatea 1 și costul egal cu valoarea absolută dintre $j - n$ și i (valoarea n este scăzută datorită faptului că am numerotat grămezile începând cu $n + 1$).

După construirea rețelei este suficient să determinăm un flux maxim de cost minim pentru a obține soluția problemei.

Fluxul va avea întotdeauna valoarea n, iar un fruct va fi extras dintr-o anumită grămadă doar dacă există flux pe arcul care leagă nodul corespunzător fructului de nodul corespunzător grămezii.

După determinarea fluxului se afișează costul acestuia, precum și alegerile efectuate.

Analiza complexității

Ordinul de complexitate al operație de citire a datelor de intrare este $O(m \cdot n)$ deoarece sunt citite m linii, fiecare conținând n numere.

Pentru crearea rețelei de transport va trebui, mai întâi, să construim cele $2 \cdot n + 2$ noduri ale acesteia, operație al cărei ordin de complexitate este $O(n)$.

Pentru a lega primele n noduri de sursă avem nevoie de un timp de ordinul $O(n)$; același ordin are și timpul necesar legării celorlalte n noduri de destinație.

De fiecare dintre ultimele n noduri vor fi legate m dintre primele n; ordinul de complexitate al operației pentru un nod este $O(m)$, deci pentru toate nodurile obținem un timp de ordinul $O(m \cdot n)$. Așadar, ordinul de complexitate al operației de construire a rețelei de transport este $O(n) + O(n) + O(n) + O(m \cdot n) = O(m \cdot n)$.

Ordinul de complexitate al algoritmului de determinare al fluxului maxim de cost minim depinde de modul în care se determină drumurile de cost minim.

Pentru a scrie datele de ieșire va trebui doar să verificăm care dintre arce conțin fluxuri nenule.

Deoarece avem $m \cdot n$ astfel de arce, ordinul de complexitate al operației de creare a fișierului de ieșire va fi $O(m \cdot n)$.

Putem trage concluzia că ordinul de complexitate al algoritmului de rezolvare a acestei probleme depinde doar de modul în care se determină fluxul, deoarece această operație va avea un ordin de complexitate superior față de cel al celorlalte operații (toate celelalte operații au ordinul de complexitate cel mult $O(m \cdot n)$).

P050231: Divizori

Pentru rezolvarea acestei probleme vom aplica metoda programării dinamice. Vom începe cu demonstrarea unui adevăr matematic: dacă n este un număr natural și a este un șir care conține puterile la care apar factorii primi din descompunerea lui n, atunci numărul divizorilor lui n este:

$$\prod_{i=0}^{\infty}(a_i+1).$$

Demonstrația acestei teoreme este foarte simplă. Numărului n i se asociază șirul a_i al puterilor la care apar factorii primi din descompunerea sa.

Astfel, fiecărui divizor îi va corespunde un element al produsului cartezian $A_1 \times A_2 \times \ ... \times A_\infty$, unde $A_i = \{0, 1, ..., a_i\}$, iar numărul elementelor acestui produs cartezian este exact:

$$\prod_{i=0}^{\infty}(a_i+1).$$

De fapt, nu trebuie să luăm în considerare toate numerele prime, fiind suficiente doar cele mai mici.

În continuare vom presupune că am ales primele 15 numere prime. La început determinăm vectorul d al divizorilor numărului k. Considerăm că acest vector are m elemente.

Construim o matrice A cu 15 linii și m coloane, A_{ij} având valoarea egală cu cel mai mic număr care are ca factori primi primele numere prime și având d_j divizori. Rezultatul cerut va fi valoarea minimă de pe ultima coloană a matricei.

Primul element al fiecărei linii (A_{i1}) va avea valoarea 1, deoarece 1 este cel mai mic număr cu un singur divizor. Vom avea $A_{0j} = \infty$ deoarece nu există numere cu i divizori și nici un factor prim.

Elementul A_{ij} poate fi calculat folosind valorile $A_{i-1,k}$ pentru $k < j$ și d_k este divizor al lui d_j. $A_{i-1,k}$ are d_k divizori, deci o valoare A_{ij} cu d_j divizori poate fi obținută calculând valoarea:

$$A_{i-1,k} \cdot p_i^{d_j / d_k + 1},$$

unde p_i este al i-lea număr prim.

Valoarea A_{ij} va fi aleasă ca fiind minimul acestor valori.

Dacă această valoare minimă este mai mare decât $A_{i-1,j}$, atunci A_{ij} va primi valoarea $A_{i-1,j}$.

Deoarece ar trebui să lucrăm cu numere foarte mari, în program determinăm logaritmii naturali ai valorilor A_{ij}, iar în final calculăm valoarea reală a minimului de pe ultima coloană a matricei folosind operații cu numere mari.

Analiza complexității

Citirea datelor de intrare se realizează în timp constant deoarece acestea constau într-un singur număr; așadar ordinul de complexitate al acestei operații este $O(1)$.

Operația de determinare a divizorilor numărului n are ordinul de $O(n)$ deoarece vom verifica toate numerele cuprinse între 1 și n.

Există și variante mai rapide, dar această operație are un timp de execuție nesemnificativ față de restul operațiilor, motiv pentru care putem folosi un algoritm puțin mai lent.

Pentru a determina valorile elementelor matricei A va trebui ca,

pentru fiecare element, să luăm în considerare toate numerele aflate deasupra sa, pe aceeași coloană.

Așadar, pentru un element ordinul de complexitate al operației va fi $O(n)$, deoarece numărul liniilor matricei este egal cu numărul divizorilor, iar acest număr variază liniar în funcție de n.

Deoarece numărul coloanelor matricei este constant, ordinul de complexitate al operației de determinare a elementelor matricei va fi $O(n^2)$.

Va trebui acum să determinăm valoarea minimă de pe ultima coloană, operație al cărei ordin de complexitate este $O(n)$.

Această valoare reprezintă doar logaritmul rezultatului; pentru determinarea valorii reale sunt necesare operații cu numere mari.

În timp liniar vom determina divizorii și puterile acestora și apoi, dacă presupunem că o operație cu numere mari se realizează în timp constant, tot în timp liniar vom determina valoarea exactă a rezultatului.

Afișarea celui mai mic număr cu n divizori este realizată în timp constant, deoarece se afișează un singur număr.

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

Observație

Este evident faptul că, chiar dacă am fi folosit o metodă mai rapidă pentru determinarea divizorilor unui număr, ordinul de complexitate ar fi fost tot $O(n^2)$.

P050232: Spioni

Rețeaua de spioni a elfilor poate fi reprezentată printr-un graf neorientat ale cărui vârfuri reprezintă spioii elfi. Între două vârfuri va exista o muchie dacă și numai dacă cei doi spioni pot comunica direct între ei.

În aceste condiții, un spion cheie reprezintă un punct de articulație al grafului, o legătură cheie reprezintă o punte a grafului, iar un subserviciu sigur reprezintă o componetă biconexă a grafului.

Ca urmare, problema se reduce la determinarea punctelor critice, a muchiilor critice și a componentelor biconexe ale unui graf neorientat.

Toate acestea pot fi determinate printr-o singură parcurgere în adâncime a grafului dar, pentru aceasta, la fiecare pas al parcurgerii trebuie efectuate toate operațiile corespunzătoare celor trei algoritmi (de determinare a punctelor de articulație, de determinare a muchiilor și de determinare a componentelor biconexe).

Datorită ordinii în care trebuie scrise datele de ieșire, fișierul nu mai poate fi creat pe parcurs. Din acest motiv, datele obținute trebuie memorate și scrise în fișier doar la sfârșit.

Analiza complexității

Citirea datelor de intrare implică citirea celor m muchii ale grafului, așadar ordinul de complexitate al acestui algoritm este $O(m)$.

În paralel cu citirea datelor, se realizează crearea structurii de date în care este memorat graful, ordinul de complexitate al acestei operații fiind tot $O(m)$.

Parcurgerea în adâncime care unifică cei trei algoritmi are ordinul de complexitate $O(m + n)$ deoarece numărul operațiilor suplimentare introduse la fiecare pas este constant.

Practic, la fiecare pas, vom verifica dacă am obținut un punct critic, dacă am obținut o muchie critică și dacă am obținut o nouă componentă biconexă.

Datele care vor fi scrise în fișierul de ieșire vor fi memorate pe măsură ce vor fi determinate; nu se va consuma timp suplimentar pentru memorarea lor.

După determinarea punctelor critice, a muchiilor critice și a componentelor biconexe, datele corespunzătoare trebuie scrise în fișierul de ieșire.

Vom avea cel mult $n - 2$ puncte de articulație, deci ordinul de complexitate al operației de scriere a acestora este $O(n)$.

Numărul punților este de cel mult $n - 1$, deci și operația de scriere a acestora are ordinul de complexitate $O(n)$.

Componentele biconexe reprezintă o partiționare a muchiilor grafului; din acest motiv, în cel mai defavorabil caz, s-ar putea ca pentru descrierea componentelor să fie necesară descrierea fiecărei muchii.

Ca urmare, operația de scriere a datelor referitoare la componentele biconexe are ordinul de complexitate $O(m)$.

Așadar, ordinul de complexitate al operației de scriere a datelor de ieșire este $O(n) + O(n) + O(m) = O(m + n)$.

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

P050233: Criză

Satele elfilor vor reprezenta nodurile unui graf neorientat; între două noduri ale grafului va exista o muchie doar dacă între cele două sate corespunzătoare există o cărare.

Astfel, practic, va trebui doar să identificăm componentele biconexe ale acestui graf și, pe măsura detectării lor, să le scriem în fișierul de ieșire.

Analiza complexității

Citirea datelor de intrare implică citirea celor m muchii ale grafului, așadar ordinul de complexitate al acestui subalgoritm este $O(m)$.

În paralel cu citirea se realizează crearea structurii de date în care este memorat graful, ordinul de complexitate al acestei operații fiind tot $O(m)$.

Algoritmul de determinare a componentelor biconexe ale unui graf are ordinul de complexitate $O(m + n)$. Pe măsura determinării acestora, ele vor fi descrise în fișierul de ieșire; așadar, nu se va consuma timp suplimentar pentru crearea acestui fișier.

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

P050234: Plantare

Rezolvarea acestei probleme este foarte simplă. Practic tot ceea ce trebuie

să facem este să verificăm dacă parcelele date apar, așa cum au fost prevăzute în planul inițial.

P050235: Alți copaci

Vom rezolva această problemă folosind metoda programării dinamice. Pentru aceasta va trebui să o descompunem în subprobleme. Având numărul n al copacilor, în vectorul a vom citi coordonatele acestora.

Vom considera un vector c, ale cărui componente trebuie determinate: valoarea c_i va reprezenta numărul maxim de poziții marcate care pot fi luate în considerare, începând de la sfârșitul rândului, până la a i-a poziția inclusiv, având în vedere că aceasta va fi folosită.

Elementele vectorului c vor fi calculate folosind relațiile:

- $c_n = 1$;
- $c_i = 1 + \max\{c_k \mid i < k \le n, d(k, i) \ge x\}$, unde $d(k, i)$ reprezintă distanța dintre marcajul având numărul de ordine i și marcajul având numărul de ordine j și se calculează folosind formula $a_j - a_i$.

Se observă că șirul reprezentat de elementele vectorului c este un șir descrescător. Soluția acestei probleme este dată de elementul c_1.

Prezentăm în continuare versiunea în pseudocod a algoritmului descris:

```
algoritm copaci(n,x,a)
  c_n ← 1
  pentru i ← n - 1, 1 pas -1
                          execută
    max ← 0
    pentru k ← i + 1, n
                          execută
      dacă (x ≤ a_k - a_i) și
           (max < c_k) atunci
        max ← c_k
      sfârșit dacă
    sfârșit pentru
    ci ← max + 1
  sfârșit pentru
  sol ← c_1
sfârșit algoritm
```

Datorită faptului că vectorul c este descrescător, la găsirea primei valori diferite de 0, se știe că acesta este chiar maximul căutat, deci putem întrerupe căutarea.

O altă îmbunătățire poate consta în a ne opri atunci când am găsit un maxim între poziția k și poziția pe care am găsit un maxim la pasul anterior.

P050236: Rețea magică

Putem privi arborii strămoșești ca fiind nodurile unui graf orientat în care există o muchie de la un nod la altul dacă arborele strămoșesc corespunzător primului nod poate trimite un mesaj arborelui strămoșesc corespunzător celui de-al doilea nod.

În aceste condiții problema se reduce la determinarea componentelor tare-conexe ale unui graf orientat.

Datorită numărului mare de muchii, vom păstra graful sub forma unei liste a succesorilor.

Pentru fiecare nod vom cunoaște poziția la care încep succesorii săi în această listă.

Evident, poziția la care se termină succesorii unui nod este dată de poziția la care încep succesorii următorului nod (cel care are numărul de ordine mai mare cu 1).

Folosind această reprezentare a grafului vom putea aplica fără dificultăți deosebite oricare dintre algoritmii rapizi de determinare a componentelor tare-conexe.

Evident, datorită numărului relativ mare de noduri, algoritmul simplu (cel cu ordinul de complexitate $O(n^3)$) nu va putea fi folosit, în schimb algoritmul pătratic va furniza soluții în timpul pus la dispoziție.

Analiza complexității

Citirea datelor de intrare implică citirea celor m muchii ale grafului, așadar ordinul de complexitate al acestui algoritm este $O(m)$.

În paralel cu citirea se determină numărul succesorilor fiecărui nod, ordinul de complexitate al acestei operații fiind tot $O(m)$.

Pentru a determina pozițiile de început ale succesorilor nodurilor, este necesară o parcurgere a vectorului care conține numerele succesorilor, ordinul de complexitate al acestei operații fiind $O(n)$.

Pentru a determina șirul succesorilor este necesară o nouă parcurgere a muchiilor; ca urmare, ordinul de complexitate al acestei operații este $O(m)$.

În continuare vom presupune că am folosit algoritmul eficient de determinare a componentelor tare-conexe. Așadar acestea sunt găsite într-un timp de ordinul $O(m + n)$.

Componentele tare-conexe sunt scrise în fișierul de ieșire pe măsura determinării lor. Datorită faptului că vor fi scrise exact n numere, ordinul de complexitate al operației de scriere a datelor de ieșire este $O(n)$.

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

Clasamentul primelor 12 runde

1.	Adrian Diaconu	757.0
2.	Mircea Pașoi	731.0
3.	Pătcaș Csaba	723.0
4.	Liviu Ciortea	634.0
5.	Mihnea Giurgea	556.0
6.	Constantin Dolghier	502.5
7.	Mugurel-Ionuț Andreica	498.0
8.	Claudiu Rad	451.0
9.	Alexandru Popa	444.0
10.	Sergiu Hlihor	388.0