Focșani, 18-24 aprilie 2003

Olimpiada NAȚIONALĂ de INFORMATICĂ

Mihai Stroe

Vă prezentăm în continuare soluțiile problemelor propuse la ONI 2003, la clasele a IX-a și a X-a. Aceste soluții au fost realizate de redacția GInfo pe baza soluțiilor oficiale prezentate de autorii problemelor și au fost verificate folosind seturile de date cu care au fost testate soluțiile concurenților.

Clasa a IX-a

P050313: Seti

Rezolvarea problemei constă în realizarea unei metode de trecere de la cuvintele de 4 litere la numerele de cel mult 4 cifre. Având o astfel de metodă, toate cuvintele se vor transforma în numere cuprinse între 0 și 9999. Numerele nu vor fi memorate, în schimb se va folosi un tablou cu 10.000 de elemente, fiecare element, reprezentând numărul de apariții al unui cuvânt din fișier. În final, pentru fiecare număr de la 0 la 9999 se va afișa cuvântul reprezentat, de atâtea ori de câte indică numărul său de apariții.

Pentru a trece de la cuvânt la număr și invers se poate folosi ca etapă intermediară un tablou de patru cifre, transformările realizându-se foarte ușor.

Analiza complexității

Fie N numărul de cuvinte din fișier care poate avea valoarea maximă 200.000.

Operația de citirea a datelor are ordinul de complexitate $O(N)$. Metodele care transformă cuvintele în numere și invers au ordinul de complexitate $O(1)$. Actualizarea tabloului care contorizează numărul de apariții se face concomitent cu citirea, deci această operație are ordinul de complexitate $O(N)$. Operația de afișarea a datelor are ordinul de complexitate $O(N)$.

În final, ordinul de complexitate al algoritmului de rezolvare este $O(N) + O(N) + O(N) = \boldsymbol{O(N)}$.

P050314: Circular

O rezolvare foarte bună a acestei probleme se bazează pe faptul că există foarte puține numere circulare între 10 și 10.000.000. Acestea pot fi generate în timpul concursului, obținându-se un fișier cu toate numerele circulare, câte 10 pe linie și separate prin virgulă. Acest fișier poate fi transformat ușor într-un vector de constante care poate fi folosit de un program *Pascal* sau *C/C++* care, astfel conține toate numerele circulare. Acest program va citi valoarea lui N, apoi va rezolva cerințele parcurgând vectorul de constante. Deci, concurentul va elabora două programe: cel de generare a numerelor circulare și cel care, folosindu-le, rezolvă cerințele problemei.

Pentru generarea numerelor circulare se poate folosi un ciclu **`for`** până la 10.000.000, în cadrul căruia se testează, pe rând, pentru fiecare număr dacă este circular. Un mod simplu de a realiza această testare este de a transforma numărul într-un vector de cifre, prin împărțiri succesive la 10 cu salvarea restului.

Se observă că se vor obține cifrele numărului în ordine inversă; este foarte ușor să se restabilească ordinea corectă prin inversarea primei cifre cu ultima, celei de-a doua cu penultima etc.

Pentru un număr reprezentat ca un vector de cifre se poate verifica foarte ușor dacă este format din cifre distincte și nu conține cifra 0; pentru condiția rămasă se va număra spre dreapta, iar pozițiile pe care se oprește numărarea vor fi marcate cu 0. Dacă la un moment dat se va ajunge pe o poziție marcată cu 0, fără a se fi marcat în prealabil toate pozițiile, atunci numărul nu este circular.

O metodă mai bună și mai rapidă ar consta în generarea tuturor vectorilor de cifre distincte nenule de cel mult 7 cifre, folosind metoda *backtracking*.

O altă metodă, și mai rapidă, ar consta în generarea directă a numerelor circulare, folosind o altă variantă a metodei *backtracking*.

Astfel, după încercarea de a folosi cifra c_1 pe prima poziție, se va trece la poziția $1 + c_1$ (de fapt se vor face c_1 avansări spre dreapta) și se va continua de acolo; dacă acolo se va folosi cifra c_2, se avansează încă c_2 poziții spre dreapta etc.

La fiecare pas din *backtracking* se folosesc cifrele cuprinse între 1 și 9, care nu au fost folosite anterior; dacă avansarea spre dreapta duce într-o poziție deja completată și mai sunt poziții libere, numărul în construcție nu poate fi circular și se revine la cifra precedentă.

Analiza complexității

Pentru această problemă, complexitatea rezolvării este dată de numărul *M* de numere circulare, deci soluția rulează instantaneu, deci ordinul de complexitate al acesteia ar fi $O(\log_2 M)$, în cazul în care se folosește o căutare binară. Deoarece *M* este o constantă, ordinul de complexitate al soluției este $O(1)$.

Complexitatea metodei de generare a numerelor circulare depinde de metoda folosită.

P050315: Scaune

Rezolvarea acestei probleme constă în simularea deplasării copiilor, conform regulilor din enunț.

Deoarece limita pentru numărul de scaune este destul de mică, avansarea copiilor se poate realiza pas cu pas; astfel, în momentul în care un scaun se eliberează, fiecare copil avansează câte o poziție până când unul dintre ei ajunge în dreptul scaunului și îl ocupă.

O metodă mai rapidă constă în calcularea, pentru fiecare copil, a numărului de mutări până la ajungerea la scaunul liber. Fie un copil situat în poziția *C*, iar scaunul liber în poziția *S*; atunci copilul va avansa $S - C$ poziții, dacă $C \le S$, respectiv $NS + S - C$ poziții, dacă $C > S$. Astfel, la fiecare ridicare a unui copil se poate afla cel mai apropiat copil, se calculează câte poziții se va deplasa acesta (să spunem *P*), apoi toți copiii vor avansa *P* poziții. Un copil care avansează *P* poziții din poziția *C* va ajunge în poziția $C + P$ (dacă $C + P \le NS$), respectiv $C + P - NS$ (dacă $C + P > NS$).

Analiza complexității

Deoarece sunt *NC* copii în așteptare, vor fi analizate cel mult *NC* ridicări de pe scaun, indiferent dacă mai urmează ceva în fișierul de intrare. Dacă sunt mai puțin de *NC* copii care se ridică, atunci o parte dintre cei care așteaptă vor rămâne în picioare.

La fiecare moment în care se ridică un copil, avansarea pas cu pas a copiilor aflați în așteptare până la așezarea pe scaun a unuia dintre ei nu poate necesita mai mult de *NS* pași; fiecare pas necesită avansarea tuturor copiilor, deci a cel mult *NC* copii. Complexitatea ocupării unui scaun prin avansare pas cu pas este deci $O(NS \cdot NC)$.

În cazul metodei mai rapide, calcularea distanței pentru toți copiii până la scaunul liber are complexitatea $O(NC)$. Aceeași complexitate au și operațiile de alegere a minimului *P* dintre aceste distanțe, respectiv de avansare a tuturor copiilor cu *P* poziții.

Ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(NC \cdot NS \cdot NC)$ pentru prima variantă, respectiv $O(NC \cdot NC)$ pentru cea de-a doua.

P050316: Criptare

Pentru decriptarea mesajului se utilizează algoritmul de criptare descris în sens invers. Se determină ordinea literelor din cuvântul cheie. Considerând că acest cuvânt are lungimea *L*, iar textul are lungimea *N*, el va fi împărțit în *L* grupe de lungimea *N*/*L*. Grupele sunt așezate într-o matrice cu *L* coloane pe coloanele corespunzătoare ordinii literelor din cuvântul cheie. În final, matricea este parcursă pe linii și se afișează mesajul decriptat.

Analiza complexității

Operațiile de citirea și afișare a datelor au ordinul de complexitate $O(N)$. Deoarece lungimea *L* a cuvântului cheie este mică, operațiile cu acesta (determinarea ordinii literelor) pot fi neglijate la calculul complexității. Scrierea cuvântului în matrice are ordinul de complexitate $O(N)$.

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

P050317: Mașină

Având în vedere că orașele sunt așezate și parcurse circular, putem dubla orașele, iar apoi putem să le considerăm așezate liniar:

$$a_n + 1 = a_1,\ a_n + 2 = a_2,\ \ldots,\ a_n + k = a_k,\ \ldots,\ a_{2n} = a_n$$
$$b_n + 1 = b_1,\ b_n + 2 = b_2,\ \ldots,\ b_n + k = b_k,\ \ldots,\ b_{2n} = b_n.$$

Cantitatea de benzină câștigată în urma opririi în orașul *i* și parcurgerii drumului către orașul $i + 1$ este $c_i = a_i - b_i$. Evident, această cantitate poate fi negativă.

Definim șirul sumelor parțiale ale vectorului *c* astfel:

$$s_i = \sum_{k=1}^{i} c_i.$$

Pentru ca *PAM* să poată efectua turul, este necesar ca suma cantității de benzină acumulate să fie mai mare sau egală cu 0, deci $s_n \ge 0$.

Rezolvarea problemei constă în a găsi un *x* astfel încât $c_x \ge 0$, $c_x + c_{x-1} \ge 0, \ldots, c_x + \ldots c_{x+n-1} \ge 0$, adică $s_x - s_{x-1} \ge 0$, $s_{x+1} - s_{x-1} \ge 0, \ldots, s_{x+n-1} - s_{x-1} \ge 0$. Un candidat viabil este poziția $m \le n$ pentru care s_m este minim. În continuare se demonstrează că această alegere conduce la o soluție.

Fie $m < k \le n$; avem $s_k \ge s_m$ (din definiția lui *m*).

Fie $n < k$; avem $s_k = c_1 + \ldots + c_n + c_{n+1} + \ldots + c_k = s_n + c_1 + \ldots + c_k = s_n + s_k \ge s_k \ge s_m$ (am folosit faptul că $s_n \ge 0$). În concluzie $x = m + 1$ este o soluție pentru problemă.

Implementarea constă în citirea datelor, calcularea șirului sumelor parțiale și găsirea minimului dintre $s_1, s_2, \ldots, s_n$.

Analiza complexității

Operația de citire a datelor are ordinul de complexitate $O(n)$.

Calculul șirului sumelor parțiale are ordinul de complexitate $O(n)$ pentru că $s_i = s_{i-1} + a_i$. Minimul din acest șir se calculează cu aceeași complexitate. Operația de scriere a rezultatului are ordinul de complexitate $O(1)$.

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

P050318: Operații

Pentru rezolvarea problemei este necesară descompunerea în binar a celor două numere și înțelegerea efectului pe care îl au cele două operații asupra acestei descompuneri.

Se observă că operația O_1 inserează o cifră 1 în descompunere, pe o anumită poziție, în timp ce operația O_2 șterge un 0, la alegere. Privind astfel cele două operații, algoritmul de rezolvare începe cu descompunerea numerelor în baza 2 și ștergerea tuturor cifrelor 0 din ambele numere. Se vor obține astfel două numere, nu neapărat egale, ale căror descompuneri în baza 2 sunt formate numai din cifre 1 (deci numerele sunt de forma $2 \cdot P - 1$). Cum cifrele 1 nu pot fi șterse, singura variantă de a ajunge la același număr constă în inserarea de cifre 1 în cadrul numărului mai mic.

La efectuarea operațiilor O_2 pe fiecare număr, acestea se memorează pentru a fi afișate. Pentru operațiile O_1 nu este necesară memorarea; ele se vor efectua numai asupra numărului care rămâne mai mic, valoarea pentru K putând fi 1 pentru toate aceste operații.

Analiza complexității

Operația de citire a datelor are ordinul de complexitate $O(1)$. De asemenea, ordinul de complexitate al operațiilor de descompunere a numerelor în baza 2, de detectare a zerourilor din descompunere, de memorare a pozițiilor operațiilor de tip 2, de detectare a numărului mai mic și afișarea mutărilor este $O(1)$.

În final ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(1)$.

Clasa a X-a

P050319: Asediu

Acestă problemă se rezolvă matematic. Considerând poligonul format din cele n puncte fără a trasa diagonalele, fiecare latură formează cu circumferința cercului n regiuni, plus o regiune interioară poligonului.

Trasând pe rând fiecare diagonală, se adaugă un număr de regiuni egal cu unu plus numărul de intersecții cu diagonalele duse deja. Astfel, fiecare intersecție adaugă o regiune la numărul total de regiuni.

Considerând fiecare grup de 4 puncte, se observă că acesta determină exact o intersecție. În total sunt C_4^n grupuri de câte 4 puncte. Numărând și diagonalele, se ajunge la:

$$nr.\ total\ de\ regiuni = n + 1 + \frac{n \cdot (n-3)}{2} + C_n^4 = 1 + C_n^2 + C_n^4.$$

În implementare se ține cont de faptul că se ajunge la numere mari. Dacă se folosește formula finală, se poate scăpa de implementarea împărțirilor pe numere mari, simplificând fracțiile rezultate înainte de a efectua produsele.

De exemplu: $(18 \cdot 17 \cdot 16 \cdot 15) / (2 \cdot 3 \cdot 4 \cdot 5)$ devine $3 \cdot 17 \cdot 4 \cdot 15$ etc.

Analiza complexității

Deoarece formula finală este un polinom de gradul 4 în funcție de N, calcularea rezultatului necesită un număr constant de operații, deci ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(1)$.

P050320: Muzeu

Se observă că numărul variantelor de cumpărare a biletelor (cumpăr / nu cumpăr biletul 1, biletul 2,... biletul 10) este $2^{10} = 1024$.

Se generează fiecare din aceste variante. Pentru o astfel de variantă, se calculează costul și se transformă matricea inițială într-o matrice cu 0 și 1, în care 0 reprezintă o cameră pentru care se plătește biletul (sau nu e nevoie de bilet) și 1 reprezintă o cameră pentru care nu se cumpăra bilet (deci în care nu se intră).

Pentru o astfel de matrice, problema determinării celui mai scurt drum de la (1, 1) la (M, N) se rezolvă cu algoritmul lui *Lee*.

Se rezolvă această problemă pentru toate cele 1024 variante. Eventual, dacă la un moment dat există o soluție cu un cost mai bun decât al variantei curente, aceasta nu mai este abordată.

Evident, problema determinării celui mai scurt drum se poate rezolva și pe matricea inițială, ținând cont la fiecare pas de biletele cumpărate; cum algoritmul lui *Lee* este folosit de obicei pentru o matrice cu 0 și 1, am folosit inițial convenția respectivă pentru a ușura înțelegerea

Se alege soluția de cost minim și, în caz de egalitate, cea cu număr minim de camere.

Analiza complexității

Operația de citire a datelor are ordinul de complexitate $O(M \cdot N)$.

Fie C numărul de numere de marcaj diferite din matrice. Pentru fiecare dintre cele 2^C variante, găsirea drumului minim cu algoritmul lui *Lee* are complexitatea $O(M \cdot N)$.

Operația de scriere a soluției are ordinul de complexitate $O(M \cdot N)$ pentru cazul cel mai defavorabil.

Ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(M \cdot N \cdot 2^C)$.

Algoritmul funcționează datorită restricției impuse pentru C (cel mult 10 numere de marcaj). Considerând 2^C o constantă, ordinul de complexitate devine $O(M \cdot N)$.

P050321: Munte

Problema se rezolvă prin metoda *programării dinamice*.

Practic, problema se poate împărți în două subprobleme. Primul pas constă în determinarea perechilor de vârfuri care pot fi unite printr-un cablu. Acest pas se rezolvă folosind cunoștințe elementare de geometrie plană (ecuația dreptei, $y = a \cdot x + b$). Se va obține o matrice OK, unde $OK_{i,j}$ are valoarea 1 dacă vârfurile i și j pot fi unite și 0 în caz contrar. Folosind această matrice, se vor conecta vârfurile 1 și N cu un lanț de K stații, astfel încât oricare două stații consecutive să aibă OK-ul corespunzător egal cu 1.

Această subproblemă se rezolvă prin metoda *programării dinamice*, după cum urmează: se construiește o matrice A cu K linii și N coloane, unde $A_{i,j}$ reprezintă lungimea totală minimă a unui lanț cu i stații care conectează vârfurile 1 și j. Inițial $A_{1,1} = 0$; $A_{1,j} = +\infty$ și $A_{i,1} = +\infty$ pentru $i > 1$. Pentru i cuprins între 2 și N, componentele matricei se calculează astfel:

$Ai,j = \min(Ai\text{-}1,v + \text{dist}v,j)$, unde $v < j$ și $OKv,j = 1$.

Concomitent cu calculul elementelor $A_{i,j}$ se construiește o matrice T, unde $T_{i,j}$ reprezintă valoarea lui v care minimizează expresia de mai sus. Matricea T este folosită pentru reconstituirea soluției.

Lungimea totală minimă este regăsită în $A_{K,N}$.

Subproblemele puteau fi tratate și simultan, ceea ce complica implementarea.

Analiza complexității

Operația de citire a datelor are ordinul de complexitate $O(N)$. Calculul matricei OK are ordinul de complexitate $O(N^3)$. Algoritmul bazat pe metoda *programării dinamice* are ordinul de complexitate $O(N^2 \cdot K)$. Reconstituirea soluției și afișarea au ordinul de complexitate $O(N)$.

Deoarece $K \le N$, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $\boldsymbol{O(N^3)}$.

P050322: Partiție

Problema se poate rezolva în mai multe moduri.

O idee bună de rezolvare a problemei constă în folosirea metodei *programării dinamice*. Se poate construi o matrice A, unde $A_{i,k}$ reprezintă numărul de partiții ale numărului i cu numere impare, din care ultimul este cel mult egal cu k.

Un element din A se calculează observând că o partiție a lui i cu numere impare, cel mult egale cu k, este formată dintr-un un număr M, mai mic sau egal cu k, și alte numere, mai mici sau egale cu M. De aici rezultă relația:

$$A_{i,k} = \sum_{\substack{M=1,k;M\le i;\\ M \text{ impar}}} A_{i-M,M}.$$

Inițial $A_{0,0}$ este 1. La implementare, pentru a economisi spațiu, se poate alege o variantă în care $A_{i,k}$ să reprezinte numărul de partiții al lui i cu numere impare, din care ultimul este cel mult egal cu al k-lea număr impar, adică $2 \cdot k - 1$.

După calcularea elementelor matricei, soluția pentru numărul i se găsește în $A_{i,i}$. Aceste valori pot fi salvate într-un vector de constante, care este transformat într-un nou program, în același mod ca la problema *Circular* de la clasa a *IX*-a. Această metodă conduce la o rezolvare instantanee a testelor date în concurs.

O altă metodă, bazată pe vectorul de constante, ar fi însemnat generarea soluțiilor folosind metoda *backtracking*. Un *backtracking* lăsat să se execute în timpul concursului ar fi putut genera soluțiile pentru toate valorile lui N, după care se putea folosi metoda vectorului de constante, în timp ce un *backtracking* folosit ca soluție a problemei ar fi obținut punctajul maxim doar pentru testele mici și medii (pentru valori ale lui N mai mici decât 130). Limitele problemei și timpul de execuție de 3 secunde permit rezolvării prin backtracking obținerea unui punctaj mulțumitor.

Analiza complexității

Pentru o rezolvare care se bazează pe metoda vectorului de constante, ordinul de complexitate al soluției finale ar fi fost $O(1)$; soluția constă în citirea valorii lui N și afișarea rezultatului memorat.

Soluția descrisă anterior, bazată pe metoda *programării dinamice*, are ordinul de complexitatea $\boldsymbol{O(n^3)}$. Invităm cititorul să caute metode mai eficiente.

Ordinul de complexitate al unei soluții care se bazează pe metoda *backtracking* este exponențial.

P050323: Rubine

Prima parte a problemei constă în determinarea, pentru fiecare hoț, a numărului maxim de rubine pe care le poate strânge, respectând cerințele enunțului. Acest număr se poate determina, pentru fiecare hoț, plecând din poziția de start a acestuia și folosind un algoritm de complexitate $O(L \cdot C)$.

Putem optimiza această metodă dacă se pornește din poziția destinație (comună) a tuturor hoților; astfel acest algoritm se va executa o singură dată, obținându-se după executare rezultatul cerut pentru fiecare poziție de start posibilă.

În prima fază a algoritmului (amintit mai sus) se obține pentru fiecare poziție (i, j) numărul minim de mutări $M(i, j)$ până la ajungerea la destinație, pornind din poziția respectivă. Pentru aceasta putem folosi algoritmul lui *Lee*, de complexitate $O(L \cdot C)$.

O examinare mai atentă a numerelor rezultate conduce la un algoritm mult mai simplu, de aceeași complexitate. Astfel, $M[L, C] = 0$, $M[L - 1, C] = M[L - 1, C - 1] = M[L, C - 1] = 1$, ..., $M[L - d_1, C - d_2] = \max(d_1, d_2)$.

Folosind această informație vom calcula, pentru fiecare poziție (i, j) din matrice, numărul maxim de rubine $R(i, j)$ care poate fi strâns pornind spre destinație din poziția respectivă. Astfel, pentru o poziție (i, j), $R(i, j)$ este maximul dintre R-urile vecinilor care au M-ul mai mic cu 1 decât $M(i, j)$, la care se adaugă cantitatea de rubine din poziția (i, j).

Pozițiile pot fi parcurse în ordinea numerelor $M(i, j)$; astfel, pentru fiecare poziție, R-urile vecinilor care ne interesează vor fi calculate anterior, deci vor fi disponibile la calculul $R(i, j)$.

Deoarece pentru fiecare poziție din matrice se vor examina cel mult trei vecini, calcularea matricei R are complexitatea $O(L \cdot C)$.

Se folosește apoi un algoritm *greedy* pentru selectarea perechii (i, j), j fiind un hoț pentru care nu există perechi (j, k).

Ordinul de complexitate al unei implementări pentru acest algoritm este $O(N \cdot N)$; o implementare mai preten-

ţioasă poate avea ordinul de complexitate $O(P)$, unde P este numărul de perechi (i, j) prezente în fişierul de intrare.

Limitele din enunţ permit şi variante mai puţin optime, cum ar fi cea în care algoritmul lui *Lee* este executat pentru fiecare hoţ.

Analiza complexităţii

Ordinul de complexitate al operaţiei de citire a datelor de intrare este $O(L \cdot C + N + P)$. Deoarece P este limitat superior de $N \cdot N$, ordinul de complexitate al acestei operaţii devine $O(L \cdot C + N \cdot N)$.

Calcularea cantităţii de rubine strânse de fiecare hoţ are ordinul de complexitate $O(L \cdot C)$.

Alegerea perechilor pentru repartizarea sacilor are ordinul de complexitate $O(P)$ sau $O(N \cdot N)$, în funcţie de implementarea aleasă.

Afişarea soluţiei are ordinul de complexitate $O(N)$.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(L \cdot C + N \cdot N) + O(N \cdot N) + O(N) = \boldsymbol{O(L \cdot C + N \cdot N)}$.

P050324: Scufiţa

Problema se rezolvă folosind metoda *backtracking* în plan. Se observă că mutările *Lupului* sunt independente de mutările *Scufiţei*, astfel ele pot fi determinate imediat după citirea datelor.

În acest punct al rezolvării, *Scufiţa* trebuie să mute la fiecare moment, astfel încât să rămână în joc, iar în final să strângă cât mai multe ciupercuţe. Restricţiile sunt date de enunţul problemei şi de numărul de ciupercuţe culese de *Lup*, care este cunoscut la fiecare moment.

Rezolvarea prin metoda *backtracking* încearcă la fiecare pas, pe rând, fiecare mutare din cele cel mult patru posibile (teoretic; de fapt, cel mult trei mutări sunt posibile în fiecare moment, deoarece *Scufiţa* nu se va întoarce în poziţia din care tocmai a venit).

Se urmăreşte respectarea restricţiilor impuse. În momentul găsirii unei soluţii, aceasta este comparată cu soluţia optimă găsită până atunci şi dacă este mai bună este reţinută.

Încă nu a fost găsită o rezolvare polinomială pentru această problemă (şi este improbabil ca o astfel de rezolvare să existe), dar limitele mici şi faptul că numărul de mutări disponibile începe să scadă destul de repede, în multe cazuri permit un timp de execuţie foarte bun.

Analiza complexităţii

Fie M numărul de mutări pe care le are de efectuat *Scufiţa*.

Operaţiile de citire şi scriere a datelor au ordinul de complexitate $O(N^2)$, respectiv $O(M)$, deci nu vor fi luate în calcul. Ordinul de complexitate este dat de rezolvarea prin metoda *backtracking*.

Având în vedere că la fiecare pas *Scufiţa* poate alege dintre cel mult 3 mutări (deoarece nu se poate întoarce în poziţia din care a venit), ordinul de complexitate ar fi $O(3^M)$. De fapt, numărul maxim de stări examinate este mult mai mic; de exemplu, primele două mutări oferă două variante de alegere în loc de trei. Alte restricţii apar datorită limitării la o matrice de $10 \cdot 10$ elemente.

Cum numărul M este cunoscut şi mic, s-ar putea considera că ordinul de complexitate este limitat superior, deci constant (constanta introdusă fiind totuşi destul de mare).

AMD64

Pe data de 23 septembrie, compania *AMD* a lansat pe piaţă mult aşteptatele modele de procesoare pe 64 de biţi destinate calculatoarelor de birou. Noile procesoare, *Athlon 64* 3200+ şi *Athlon 64 FX-51* se alătură celorlalte procesoare care folosesc tehnologia AMD64, cele din familia Opteron, care sunt destinate serverelor.

Indicativul 3200+ al procesorului *Athlon 64* arată că performanţele sale sunt comparabile cu cele ale unui procesor realizat de compania *Intel* care are o frecvenţă de 3,2 *GHZ*. *Athlon 64* are o frecvenţă de 2 *GHZ*, foloseşte o magistrală cu frecvenţa de 400 *MHZ*, iar interfaţa sa externă este *Socket 754*.

Procesorul *Athlon 64 FX-51*, spre deosebire de *Athlon 64* 3200+, are frecvenţa de 2,2 *GHZ*, iar performanţele sunt asemănătoare cu cele ale unui procesor Intel Pentium 4 Extreme Edition cu frecvenţa de 3,2 GHz (acesta este un procesor Xeon modificat pentru a fi folosit pentru calculatoarele de birou, performanţele sale superioare fiind datorate unei memorii cache de nivel 3, cu dimensiunea de 2 MB).

Noile modele de procesoare lansate de *AMD* sunt echipate cu diode termosensibile care înregistrează schimbările de temperatură de 16 ori pe secundă şi întrerup funcţionarea procesorului dacă temperatura depăşeşte valorile normale.

Setul de instrucţiuni al celor două modele de procesoare este format din instrucţiunile procesoarelor din familia *Athlon XP* pe 32 de biţi la care s-au adăugat instrucţiuni pe 64 de biţi, precum şi instrucţiunile SSE2.

O altă facilitate de care dispun aceste procesoare este tehnologia *HyperTransport*. Cu ajutorul acesteia, sistemele dotate cu aceste procesoare sunt mult mai eficiente, fiind diminuate pierderile de performanţă datorate transferurilor de date între acestea şi placa de bază.