Satu Mare, 29 martie 2003

Concursul INTERJUDEȚEAN de programare GRIGORE MOISIL

Silvestru-Cosmin Negrușeri

Vă prezentăm în continuare soluțiile problemelor propuse la concursul interjudețean de programare Grigore Moisil. Aceste soluții au fost realizate de redacția Ginfo pe baza soluțiilor oficiale prezentate de autorii problemelor.

Clasele a IX-a

P060319: Excursie

Pentru această problemă se poate observa foarte ușor că numărul de posibilități în care se poate urca până în vârful muntelui este egal cu F_{n-3}, unde n este numărul de serpentine și F_i, i = 1, 2, 3, ..., n este șirul lui *Fibonacci* care respectă următoarele relații:
- $F_0 = 0$;
- $F_1 = 1$;
- $F_i = F_{i-1} + F_{i-2}$, $i > 1$.

Deci, dacă avem o singură serpentină, există trei trasee distincte care respectă condițiile impuse în enunțul problemei. Dacă avem două serpentine, există cinci trasee.

Dacă nu ar fi fost impusă restricția $n \leq 40$ asupra numărului de serpentine n, ar fi apărut necesitatea implementării operației de adunare cu numere mari.

Analiza complexității

Operațiile de citire a datelor de intrare și scriere a rezultatelor are ordinul de complexitate $O(1)$.

Calcularea celui de-al $(n - 3)$-lea element din șirul lui *Fibonacci* folosind relația de recurență are ordinul de complexitate $O(n)$.

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

P060320: Loto

Prima cerință a problemei este de a determina numărul total de variante pe care le poate juca *Vasile* în noul joc creat de el. Acest număr reprezintă numărul de submulțimi de y elemente ale unei mulțimi cu x elemente și este egal cu C_x^y (combinări de x elemente luate câte y).

Există mai multe metode de determinare a acestui număr. Datorită faptului că datele de test sunt mici chiar și o rezolvare care generează toate variantele posibile s-ar încadra în timp.

Soluția oficială a problemei folosește șase structuri repetitive `for` imbricate datorită faptului că numărul de numere prezente într-o variantă nu depășește valoarea 6 și generează toate variantele posibile. Un astfel de algoritm are ordinul de complexitate $O(C_x^y)$.

Această cerință se mai poate rezolva folosind formula de calcul pentru combinări care este $C_x^y = x!/(y! \cdot (x - y)!)$, dar la început va trebui să simplificăm cu cea mai mare valoare de la numitor și apoi, la fiecare pas trebuie efectuate împărțiri, sau folosind următoarea relație de recurență:
- $C_n^0 = 1$;
- $C_n^n = 1$;
- $C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$.

Cea de-a doua cerință a problemei este să determinăm numărul variantelor care rămân nejucate dintre cele n posibile, după ce am extras variantele care le vom juca pe baza regulii impuse în enunț. Considerăm toate variantele posibile ca făcând parte dintr-o listă circulară și jucăm variantele din k în k până când ajungem la o variantă pe care am jucat-o deja.

Deci, pentru a rezolva această cerință, trebuie să determinăm valoarea minimă x având proprietatea că $x \cdot k$ este multiplu al lui n. Valoarea minimă pentru x, care respectă proprietate cerută, este $cmmmc(n, k) / k$, unde $cmmmc(n, k)$ este cel mai mic multiplu comun al numerelor n și k.

Valoarea x astfel determinată reprezintă chiar numărul de variante jucate, deci numărul de variante care nu s-au jucat încă este egal cu $n - x$.

Pentru a putea calcula *cmmmc*(n, k) se folosește relația *cmmmc*$(n, k) = m \cdot n$ / *cmmdc*(n, k), unde *cmmdc*(n, k) este cel mai mare divizor comun al numerelor n și k și poate fi calculat folosind algoritmul lui *Euclid*.

Analiza complexității

Operațiile de citire a datelor de intrare și scriere a rezultatelor au ordinul de complexitate $O(1)$.

Ordinul de complexitate pentru calculul combinărilor pentru prima cerință a problemei, dacă se folosește relația de recurență, este $O(x \cdot y)$.

Ordinul de complexitate al algoritmului de rezolvare pentru cea de-a doua cerință este dat de ordinul de complexitate al algoritmului de calcul pentru *cmmdc*(n, k) care este $O(\log (n + k))$, restul operațiilor se realizează în timp constant.

În concluzie, ordinul de complexitate a soluției pentru această problemă este $O(1) + O(x \cdot y) + O(\log (n + k)) + O(1) = \boldsymbol{O(x \cdot y) + O(\log (n + k))}$.

Clasele a X-a

P060321: Împărțeală

Problema se reduce la determinarea numărului de partiții a numărului n în k numere care să fie ordonate crescător.

Rezolvarea problemei se bazează pe *metoda programării dinamice*. Un algoritm de rezolvare care se bazează pe metoda *backtracking* nu s-ar încadra în timp pentru toate testele.

Notăm cu $num_{i,j}$ numărul de partiții a numărului i în j numere. Observăm că $num_{i,j}$ respectă următoarea relație de recurență:

$$num_{i,j} = \sum_{p=0}^{\lfloor i/j \rfloor} num_{i-j \cdot p, j-1}.$$

În continuare vom explica de unde a apărut această relație. Observăm că numărul de soluții care au primul termen egal cu p este egal cu numărul de partiții crescătoare de k termeni care au fiecare termen mai mare sau egal cu p. Dacă scădem din fiecare element al partiției numărul p obținem numărul de partiții crescătoare cu primul termen 0 al numărului $n - k \cdot p$.

În momentul implementării soluției algoritmului trebuie ținut cont de faptul că elementele aflate pe prima coloană a matricei *num* respectă următoarea relație: $num_{i,1} = 1$, $1 \le i \le n$, deoarece numărul de partiții cu un element pentru numărul i este egal cu 1. Rezultatul pentru această problemă îl va constitui valoarea $num_{n-k,k}$ deoarece fiecare termen al partițiilor trebuie să fie mai mare sau egal cu 1.

Analiza complexității

Operațiile de citire a datelor de intrare, respectiv de scriere a rezultatelelor au ordinul de complexitate $O(1)$.

Determinarea numărului de partiții are ordinul de complexitate $O(k \cdot n \cdot \log n)$, deoarece calculul tuturor sumelor la pasul curent are ordinul de complexitate $O(n \cdot \log n)$, ordinul mediu de complexitate al operației de determinare a unei singure sume fiind $O(\log n)$.

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

P060322: Cutii

Această problemă a fost propusă în altă formă la o selecție a lotului olimpic din 1999 și o rezolvare a fost publicată în paginile revistei ***GInfo.***

Pentru a rezolva problema putem folosi *metoda greedy*, astfel: punem cutia curentă în cea mai mică cutie, care este mai mare decât ea, existentă printre cutiile deja poziționate.

Șirul dimensiunilor cutiilor deschise este crescător. Această afirmație poate fi demonstrată ușor prin inducție și prezentăm în continuare această demonstrație.

Pentru un singur pachet afirmația este adevarată. În cazul în care avem un șir de n pachete ordonate crescător și adăugăm o cutie nouă, atunci ea fie formează un nou pachet (deci șirul este crescător), fie intră într-un pachet deja format, și din modul de alegere a pachetului rezultă că șirul dimensiunilor rămâne crescător.

În concluzie, dacă folosim proprietatea că șirul este crescător, pentru a găsi locul cutiei curente vom folosi o căutare binară în locul unei căutări secvențiale.

Analiza complexității

Operațiile de citire a datelor de intrare și scriere a rezultatelor au ordinul de complexitate $O(n)$.

Operația de împachetare a cutiilor are ordinul de complexitate $O(n \cdot \log n)$, deoarece pentru fiecare cutie care vine pe bandă căutăm locul unde va fi poziționată folosind o căutare binară.

În concluzie, ordinul de complexitate a soluției acestei probleme este $O(n) + O(n \cdot \log n) + O(n) = \boldsymbol{O(n \cdot \log n)}$.

Clasele a XI-a și a XII-a

P060323: Ali Baba și cei 40 de hoți

Acestă problemă se reduce la determinarea elementului majoritar dintr-un șir. Restricțiile impuse în enunțul problemei ne permit să sortăm șirurile și să determinăm elementul majoritar. Pentru restricții mai puțin severe, o metodă de rezolvare a fost tratată într-un articol mai vechi din *GInfo*. O altă rezolvare a fost dată de studenții de la *Universitatea București* în cadrul concursului regional *ACM* din anul 1998, și această rezolvare fiind publicată în paginile *GInfo*.

Pentru o a treia rezolvare observăm că, dacă șirul mărgelelor ar fi sortat, atunci culoarea mărgelei majoritare (dacă există o mărgea majoritară) ar fi chiar la mijlocul șirului. Însă pentru determinarea elementului median dintr-un șir nu este necesar să sortăm șirul ci putem să folosim abordarea randomizată care are ordinul de complexitate $O(n)$.

Subalgoritmul de determinare a numărului median dintr-un șir care are ordinul de complexitate $O(n)$ este similar subalgoritmului de sortare rapidă în care pivotul se alege aleator și este folosit pentru a determina al i-lea cel mai mic element. Acest algoritm este:

```
Subalgoritm DeterminăMediana(a,p,r,i)
// a - șirul de numere
// p, r - indicele elementelor între care se caută al i-lea cel
          mai mic element
dacă p = r atunci
   returnează a_p
q ← PartiţieAleatoare(a, p, r)
k ← q - p + 1
dacă i ≤ k atunci
   returnează DeterminăMediana(a,p,q,i)
altfel
   returnează DeterminăMediana(a,q,r,i - k)
sfârșit subalgoritm
```

Acest subalgoritm se apelează cu parametrii a, 1, n, respectiv $[n / 2]$, pentru un șir de numere a cu n elemente.

Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate $O(n \cdot nr)$, unde n este numărul de boluri, iar nr este numărul mediu de mărgele care se află în boluri.

Operația de determinare a elementului majoritar are ordinul de complexitate $O(n \cdot nr)$.

Operația de scriere a rezultatelor de intrare are ordinul de complexitate $O(k \cdot n)$, unde k este numărul maxim de mărgele identice care se extrag din fiecare bol.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(n \cdot nr) + O(n \cdot nr) + O(k \cdot n) = \mathbf{O(n \cdot (k + nr))}$.

P060324: Profi și studenți

Structura profesorilor formează un arbore, iar procedeul, prin care aceștia părăsesc sala de ședințe, explicat în enunțul problemei, reprezintă algoritmul de determinare al codului *Prufer* pentru un arbore.

Se observă că o rezolvare cu ordinul de complexitate $O(n^2)$ s-ar încadra în timpul acordat pentru execuție.

În continuare vom încerca o abordare al cărei ordin de complexitate este $O(n \cdot \log n)$ pentru a determina ordinea de ieșire a profesorilor.

Vom folosi un șir g cu n elemente, unde n reprezintă numărul de noduri din arbore care este egal cu numărul de profesori, iar g_i, $1 \le i \le n$, reprezintă numărul de fii ai nodului i. Vom mai folosi o structură *min-heap* (*heap* în care cel mai mic element este vârful) în care vom introduce nodurile pentru care $g_i = 0$, care reprezintă frunzele arborelui.

În continuare, analizăm codul *Prufer* secvențial și, la fiecare pas, eliminăm frunza care are asociat numărul cel mai mic, decrementăm gradul nodului tată, iar dacă gradul a devenit 0, îl inserăm în *min-heap*-ul existent.

Pentru decodificarea codului *Prufer* în $O(n)$ se poate studia ***Problema 8*** din cartea *Psihologia concursurilor de informatică*, Cătălin Frâncu, Editura L&S, București, 1997.

Analiza complexității

În continuare vom analiza complexitatea algoritmului de rezolvare pentru o singură universitate.

Operațiile de citire a datelor de intrare și scriere a rezultatelor au ordinul de complexitate $O(n)$.

Complexitatea algoritmului de decodificare a codului *Prufer* prezentată este $O(n \cdot \log n)$.

În concluzie, ordinul de complexitate al algoritmului de decodificare a codului *Prufer* este $O(n) + O(n \cdot \log n) + O(n) = \mathbf{O(n \cdot \log n)}$.

Dacă numărul de universități este m, atunci ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $\mathbf{O(m \cdot n \cdot \log n)}$.

P060325: Hacker

Această problema reprezintă o aplicație directă a *metodei programării dinamice*.

Vom nota cu s șirul care reprezintă cheia de criptare, cu a matricea citită din fișierul de intrare și cu n dimensiunea acesteia.

Vom folosi un tablou auxiliar bidimensional c care are aceleași dimensiuni cu cele ale matricei citite, iar c_{ij} reprezintă numărul de apariții al subșirului lui s care începe de pe prima poziție din șir și se termină pe poziția pe care se află elementul a_{ij} în s. La început, elementele matricei pentru care elementele corespunzătoare din matricea a sunt egale cu primul caracter din cheia s vor fi inițializate cu valoarea 1, iar restul cu valoarea 0.

În continuare, vom parcurge secvențial cheia s, iar elementele din c, pentru care elementul corespunzător din matricea a este egal cu caracterul curent din cheia s, vor primi ca valoare suma vecinilor lor.

În final, rezultatul problemei îl constituie suma tuturor elementelor din matricea c al căror corespondent din matricea a este egal cu ultimul caracter din s.

Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate $O(n^2)$.

Operația de determinare a numărului de apariții a cheii s în matricea a are ordinul de complexitate $O(n^2 \cdot k)$, unde k reprezintă lungimea lui s.

Operațiile de scriere a rezultatului are ordinul de complexitate $O(1)$.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(n^2) + O(n^2 \cdot k) + O(1) = \mathbf{O(n^2 \cdot k)}$.

Silvestru-Cosmin Negruşeri este student în anul III la Universitatea Babeș-Bolyai din Cluj-Napoca și poate fi contactat prin e-mail la adresa kosmin_2000@yahoo.com.