

CNER CODE, IAȘI
CLASA A X-A
DESCRIEREA SOLUȚIILOR

COMISIA ȘTIINȚIFICĂ

PROBLEMA A: SUMXOR

Propusă de: Andrei Boacă, Colegiul Național "Emil Racoviță", Iași

Subtask-urile 1 și 2. $1 \leq N \leq 10$

Pentru rezolvarea acestor subtask-uri este suficientă generarea tuturor permutărilor de lungime N și calcularea pentru fiecare permutare care este valoarea expresiei E . Generarea permutărilor poate fi realizată printr-un algoritm de tip succesor sau poate fi o generare recursivă.

Subtask-ul 3. $C = 1$

Să considerăm $A = [1, 2, 3, 4, \dots, N]$. Pentru $N = 1$, permutarea B va fi formată doar din elementul 1, deci valoarea expresiei va fi obligatoriu 2. Altfel, putem obține mereu o permutare B , pentru care expresia va avea valoarea 0, care este în mod evident minimă. Construcția soluției se poate realiza în următorul mod:

- Dacă N este par, atunci permutarea B poate fi $[2, 1, 4, 3, 6, 5, \dots, N, N - 1]$. Astfel, pentru orice pereche de indici de forma $(2 \cdot K - 1, 2 \cdot K)$, $1 \leq K \leq N/2$, vom avea $A[2 \cdot K - 1] + B[2 \cdot K - 1] = A[2 \cdot K] + B[2 \cdot K]$. Prin urmare, termenii consecutivi din expresie vor fi egali, deci se vor anula atunci când vom aplica operația XOR.
- Dacă N este impar, atunci permutarea B poate fi $[1, 2, 3, 5, 4, 7, 6, \dots, N, N - 1]$. În practică, soluția rămâne la fel, doar că primii 3 termeni vor rămâne neschimbați, deoarece $(1 + 1) \oplus (2 + 2) \oplus (3 + 3) = 2 \oplus 4 \oplus 6 = 0$, iar $N - 3$ este par, deci folosim aceeași abordare ca la punctul precedent.

Subtask-ul 4. $C = 2$

Vom considera P ca fiind cel mai mare număr, astfel încât $2^P \leq N$. Cu alte cuvinte, $P = \lfloor \log_2(N) \rfloor$. Putem spune că în baza 2, numărul N are P biți. Deci, orice număr de forma $A[i] + B[i]$ va avea maxim $P + 1$ biți. Prin urmare, și expresia E va avea maxim $P + 1$ biți. Numărul maxim ce se poate obține folosind $P + 1$ biți este $2^{P+2} - 1$. Dar, se observă că suma numerelor din cele două șiruri este $2 \cdot (1 + 2 + 3 + \dots + N)$, care este pară. Prin urmare, suma numerelor $A[1] + B[1]$, $A[2] + B[2]$, ..., $A[N] + B[N]$ este de asemenea pară de unde rezultă că cel mai nesemnificativ bit va fi setat de un număr par de ori, lucru care ne duce la concluzia că E va avea mereu o valoare pară. Deci, E nu poate fi $2^{P+2} - 1$, dar poate fi $2^{P+2} - 2$ de fiecare dată folosind o construcție după cum urmează:

- Dacă $N - P$ este par, atunci, dacă $A[i]$ este putere de 2, vom seta $B[i] = A[i]$. Vom rămâne cu un număr par de elemente nesetate, pe care le vom împerechea două câte două pentru a obține XOR-ul 0, la fel ca la cerința 1. Astfel, vom obține valoarea $E = 2^1 + 2^2 + 2^3 + \dots + 2^{P+1} = 2^{P+2} - 2$.
- Dacă $N - P$ este impar, atunci pentru $N = 1$ avem o singură soluție posibilă, iar pentru $N \leq 3$ vom putea aplica o strategie asemănătoare celei de la punctul precedent, cu diferența că în loc să setăm $A[i] = B[i]$ pentru $A[i] = 1$ și $A[i] = 2$, vom seta $B[i] = A[i]$ pentru $A[i] = 3$, iar valorile 1 și 2 vor fi împerecheate pentru a obține XOR-ul 0. Se observă ușor că în urma acestei modificări numărul de elemente neîmperecheate va fi par, deci vom reuși să obținem rezultatul dorit.

PROBLEMA B: DRAR

Propusă de: Răileanu Alin-Gabriel, Colegiul Național "Emil Racoviță", Iași

Subtask-ul 1. $X = Y = 1$

Pentru rezolvarea acestui subtask este suficientă observația că inițial, Stresu' este în aceeași cameră cu d-ra R. Astfel răspunsul va fi mereu 0.

Subtask-ul 2. $1 \leq N, M \leq 6, 0 \leq Q \leq 3$

Pentru rezolvarea acestui subtask este suficientă utilizarea unei soluții de tip Backtracking care generează toate drumurile posibile ale lui Stresu' și stochează pentru fiecare cameră momentele de timp în care acesta poate ajunge acolo. În final este necesară doar construirea traseului d-rei R și găsirea timpului minim, utilizând datele generate anterior.

Subtask-urile 3 și 4. $7 \leq N, M \leq 50, 51 \leq N, M \leq 200$

Pentru rezolvarea acestor subtask-uri se poate utiliza metoda programării dinamice.

Se va reține o structură de date tetra-dimensională:

- Definiție: $dp[t][i][j][st]$ - (*true/false*) dacă Stresu' poate ajunge în camera $[i, j]$, din celula la care se referă starea st , în t unități de timp.
- Recurență: $dp[t][i][j][st] = \vee(dp[t-1][i'][j'][st']), (1 \leq t \leq lg), (i', j' \text{ sunt coordonatele camerei la care face referire starea } st), (st' \text{ este orice stare care nu se referă la camera } [i, j]), \text{ unde cu } "\vee" \text{ s-a notat operația logică "sau", fiind vorba de o structură de tip bool.}$
- Inițializare: ținând cont că în momentul de timp 0, Stresu' se află în camera $[x, y]$, trebuie să marcăm $dp[0][x][y][st] = \text{true}, (\forall st)$.

Observăm că pentru această soluție:

- t poate lua valori între 0 și lg (lungimea traseului d-rei R);
- i poate lua valori între 1 și N ;
- j poate lua valori între 1 și M ;
- st poate lua valori între 4 și $Q+4$ (deoarece într-o cameră se poate ajunge din fiecare cameră adiacentă + camerele legate prin ași)

Soluția se poate implementa atât iterativ, cât și recursiv, folosind memoizare.

În cele din urmă, având datele calculate în tabloul "dp", sunt cunoscute pentru fiecare cameră toate momentele de timp în care Stresu' poate ajunge acolo. Astfel, în timpul construirii traseului d-rei R pe baza șirului dat în input, se poate face și găsirea timpului minim în care cei doi vor ajunge în aceeași cameră, dacă acest aspect este posibil.

Complexitatea în spațiu va fi $O(LGMAX \cdot NMAX \cdot MMAX \cdot QMAX)$, iar cea în timp va fi $O(LGMAX \cdot (NMAX \cdot MMAX + QMAX))$, unde am folosit notațiile:

- NMAX-numărul maxim de linii,
- MMAX-numărul maxim de coloane,
- LGMAX-lungimea maximă a traseului d-rei R,
- QMAX-numărul maxim de ași.

În plus, pentru subtask-ul 4, sunt necesare câteva optimizări.

Observații:

- În implementarea iterativă nu este necesară stocarea timpului în una din dimensiuni, deoarece soluțiile pentru un timp t , depind doar de timpul $t-1$. Așadar în loc de lg valori, t va putea lua doar 2.
- Nu este necesară reținerea tuturor stărilor pentru camera din care am venit în $[i, j]$, deoarece se garantează în enunț că dacă există un as în mână de la camera $[L1, C1]$ la camera $[L2, C2]$, nu va exista un alt as de la camera $[L2, C2]$ la camera $[L1, C1]$. Așadar din $Q+4$ stări, numărul se va reduce la 5 (cele 4 camere adiacente și a 5-a care face referire la venirea printr-un as).

Așadar, complexitatea în spațiu se va reduce la $O(NMAX \cdot MMAX)$, iar cea în timp va rămâne aceeași.

Mențiuni.

Problema se poate rezolva și prin soluții de tip Greedy în complexități chiar mai bune, dar demonstrația acelor soluții depășește limitele materiei pentru Olimpiada Națională de Informatică la clasa a X-a.

Așadar, rămâne ca exercițiu pentru concurenți rezolvarea problemei utilizând acele abordări cu restricția: $1 \leq N, M \leq 2000$.

PROBLEMA C: MINOTAUR

Propusă de: Loghinoia Ilinca-Ioana, Colegiul Național "Emil Racoviță", Iași

Problema este bazată pe curba lui Hilbert, un fractal, de aceea vom folosi metoda Divide et Impera. Vom considera că fiecare matrice este împărțită în 4 cadrane, numerotate astfel: cadranul 1 este cel din stânga jos, cadranul 2 este cel din stânga sus, cadranul 3 este cel din dreapta sus, iar cadranul 4 este cel din dreapta jos.

Subtask-ul 1. $N \leq 7$

Putem calcula pentru fiecare celulă din matricea $2^N * 2^N$ numărul de pași cu care se poate ajunge în ea. Analizăm în ce cadran se află celula și putem adăuga la răspuns numărul de celule din cadranele precedente. Apoi, întoarcem cadranul analizat în poziție normală și aplicăm același procedeu până când cadranul este constituit dintr-o singură celulă.

Deoarece complexitatea în timp este $O(2^{2*N} * N)$, programul nu se va încadra în timp decât pentru acest subtask. Memoria pe stivă utilizată este $O(N)$, iar cea alocată este neglijabilă, nefiind nevoie să ținem minte răspunsul pentru fiecare celulă din matrice.

Subtask-urile 2 și 3. $N \leq 15$

Trebuie să aflăm care este ultimul cadran în care numărul de pași dat ne permite să ajungem. Vom lua 2 perechi de numere, reprezentând colțul din stânga-sus și din dreapta-jos al ariei din matrice în care se află poziția căutată. Pentru fiecare iterație, din numărul de pași vom scădea numărul de celule din cadranele parcurse complet. De exemplu, dacă putem ajunge în cadranul 3, vom scădea numărul de celule din cadranele 1 și 2. De asemenea, vom reduce aria în care căutăm poziția la cea a ultimului cadran în care se poate ajunge cu ajutorul celor două perechi de indici stabilite. Acest lucru se întâmplă până când numărul de pași ajunge la 0 și aria în care căutăm este de doar o singură celulă.

Complexitatea în timp a acestui program este $O(N)$, memoria pe stivă este $O(N)$, iar memoria alocată este neglijabilă.