Descrierea Soluțiilor
Olimpiada Societății pentru Excelență și Performanță în Informatică, Baraj Juniori

1 Problema Intergalactic

Propunători: Pop Ioan-Cristian, Universitatea Politehnica din București
Tulbă-Lecu Theodor-Gabriel, Universitatea Politehnica din București

Soluție pentru cazul în valoare de 7 puncte, în care pe piață există fie un singur reactant stabil, fie un singur reactant instabil

Problema se reduce la identificarea unicului reactant (fie stabil, fie instabil), respectiv la sortarea crescătoare a celorlalți reactanți. Rezultatul va fi suma dintre reactantul unic și al k-lea reactant din vectorul sortat.

Pentru determinarea numerelor prime, se pot folosi atât ciurul lui Eratostene, cât și algoritmul de determinare în complexitate temporală $\mathcal{O}(\sqrt{N})$.

Soluție pentru cazul în valoare de 11 puncte, în care $K \leq 100$

Se observă că este nevoie strict de primii 100 de reactanți stabili, respectiv instabili (sortați în ordine crescatoare). Se construiesc sumele alcauite dintr-un reactant stabil și reactant instabil, iar apoi se sortează aceste sume. Astfel, rezultatul este a K-a suma.

Pentru determinarea numerelor prime, doar cirul lui Eratostene va obține punctaj maxim.

Soluție pentru cazul în valoare de 21 de puncte, în care $N \leq$1000

Se implementează aceeași solutie ca la cerința precedentă, cu precizarea că va trebui să procesam toți reactanții stabili, respectiv instabili.

Pentru determinarea numerelor prime, se pot folosi atât ciurul lui Eratostene, cât și algoritmul de determinare în complexitate temporală $\mathcal{O}(\sqrt{N})$.

Soluție pentru cazul în valoare de 28 de puncte, în care $N \leq 30000$

Aceste 28 de puncte se vor obtine pentru o solutie ineficienta, insa foarte asemanatoare cu soluția de 100 de puncte.

Determinăm numerele prime și construim doi vectori sortați: unul cu numere prime, unul cu numere neprime.

Observația cu care se pot obține cele 100 de puncte este aceea că soluția trebuie căutată binar. Fie valoarea propusă de cautarea binara (la un pas) X. Atunci, rămane să verificăm cate sume au valoarea strict mai mică decât X. Dacă numarul de sume este mai mic, atunci înseamnă că există o posibilitate ca X să fie soluția problemei, și căutăm o valoare mai mare. În caz contrar, înseamnă că soluția este mai mică decât X, deci căutăm o valoare mai mică. Soluția este ultima valoare propusă validă.

Pentru a calcula numarul de sume mai mici decât X, putem cauta binar, pentru fiecare număr prim (notăm cu A), care este cel mai mare număr neprim (notăm cu B) pentru care $A + B < X$. Având în vedere că vectorul este sortat, toate numerele din stânga lui B vor respecta și ele restricția (suma să fie mai mică decat X). Deci, vom aduna într-un contor, pentru fiecare valoare A, cate numere din celalalt vector pot fi alese.

O implementare a soluției de 100 de puncte, însă care determină în complexitate temporală $\mathcal{O}(\sqrt{N})$ numerele prime și nu cu ciurul lui Eratostene va obține aceste 28 de puncte.

Soluție oficială – 100 de puncte

Vom proceda la fel ca la soluția anterioară, însă schimbam metoda prin care determinăm câte sume sunt mai mici decat X. Plecăm de la observația că, trecând de la un numar prim la urmatorul, valoarea maximă a numarului neprim este cel mult egală cu valoarea maximă de la pasul precedent. Astfel, vom folosi un iterator prin vectorul de numere neprime. La trecerea la urmatorul număr prim, mutăm interatorul la stânga până când suma devine mai mică sau egală cu X (dacă este cazul).

O implementare a acestei soluții, însa fară ciurul lui Eratostene, nu va obține punctaj maxim.

2 Problema Cârtița

Propunător: Daniela Lica, Centrul Județean de Excelență Prahova, Ploiești

Problema de față constă în prelucrarea eficientă a celor U *update*-uri asupra întregului șir de valori, și a celor Q *query*-uri de aflare a minimului dintr-un interval dat.

Subtask 1 (12 puncte):

Fiecare *update* va fi efectuat parcurgând întreg șirul de N valori și modificând corespunzător înălțimile.

Fiecare *query* va fi efectuat parcurgând întreg șirul de N valori și reținând înălțimea minimă întâlnită.

Complexitatea de timp totală: $\mathcal{O}(U \cdot N + Q \cdot N)$.

Subtask-urile 1 și 2 (21 de puncte):

Fiecare *update* va fi efectuat parcurgând întreg șirul de N valori și modificând corespunzător înălțimile.

Pentru a răspunde corect și eficient la cele Q *query*-uri, se pot folosi următoarele tehnici și metode de aflare a minimului dintr-un interval compact: *Arbori de Intervale*, *Range Minimum Query*, respectiv *Șmenul lui Batog*.

Complexitatea de timp totală: $\mathcal{O}(U \cdot N + Q \cdot log_2 N)$, sau $\mathcal{O}(U \cdot N + N \cdot log2N + Q)$, respectiv $\mathcal{O}(U \cdot N + Q \cdot \sqrt{N})$.

Subtask-urile 1, 2, 3 și 4 (63 de puncte):

Pentru a aborda eficient operațiile tip *update*, vom folosi tehnica *Sqrt Decomposition* sau *Șmenul lui Batog*. Astfel, fiecare *update* va fi prelucrat într-o manieră de tip *lazy*, modificând valoarea a cel mult $2 \cdot \sqrt{N}$ elemente propriu-zise din șirul dat, în complexitatea de timp $\mathcal{O}(\sqrt{N})$.

Mai exact, vom împărți șirul de N elemente în *bucket-uri* (compartimente, subsecvențe) compacte de mărime $\lfloor \sqrt{N} \rfloor$. Așadar, vom avea cel mult $(\sqrt{N} \pm 2)$ *bucket-uri*. Dacă notăm cu $DIM = \lfloor \sqrt{N} \rfloor$, împărțirea celor N indici după metoda descrisă mai sus va fi de forma:

$$[1, DIM][DIM + 1, 2 \cdot DIM] \dots [(\lfloor \frac{N + DIM - 1}{DIM} \rfloor - 1) \cdot DIM + 1, N].$$

Pentru un *update* de forma: $i\ x\ k$, înseamnă că valorile elementelor din fiecare poziție pos ($1 \leq pos \leq N$) se vor modifica cu $modif_{pos} = (x + (pos - i) \cdot k)$; dacă $modif_{pos} \geq 0$, putem spune că valoarea elementului de la poziția pos crește, altfel aceasta scade. Poziția i se află în *bucket*-ul cu indicele (indexat de la 1): $y = \lfloor \frac{i + DIM - 1}{DIM} \rfloor$. În cadrul acestui *bucket*, cu indicele y, vom efectua modificările într-o manieră de tip *brute-force*, parcurgând cel mult DIM valori. Pentru acele *bucket*-uri care sunt indexate cu o valoare $> y$, vom modifica doar valoarea primului element (de indice minim) din cadrul *bucket*-ului; pentru cele care sunt indexate cu o valoare $< y$, vom modifica doar valoarea ultimului element (de indice maxim) din cadrul *bucket*-ului.

După ce am prelucrat *lazy* cele U *update*-uri, dorim ca în $\mathcal{O}(N)$ operații să modificăm întregul șir inițial, pentru a reține exact valorile reale care rezultă din urma *update*-urilor. Pentru a actualiza corect valorile din șir, vom reține suma tuturor k-urilor care apar în cele U *update*-uri. Pe baza acestei sume și a valorilor modificate din capetele *bucket*-urilor, putem deduce corect și restul valorilor.

Pentru procesarea eficientă *online*, în $\mathcal{O}(1)$, a fiecărui *query*, vom alege să efectuăm o precalculare (în $\mathcal{O}(N \cdot log_2 N)$), în cadrul căreia vom folosi tehnica *Range Minimum Query*. De asemenea, soluții ce folosesc tehnicile *Arborilor de Intervale* și *Range Minimum Query* vor fi destul de eficiente pentru a satisface task-urile amintite mai sus.

O soluție ce tratează în $\mathcal{O}(U \cdot \sqrt{N})$ *update*-urile și într-o manieră *brute* *query*-urile ($\mathcal{O}(Q \cdot N)$), poate obține $12 + 25 = 37$ de puncte (Subtask-urile 1 și 3).

Prin urmare, complexitățile de timp dorite pentru a obține punctajul maxim, de 63 de puncte în acest caz, sunt: $\mathcal{O}(U \cdot \sqrt{N} + N \cdot log_2 N + Q)$, $\mathcal{O}(U \cdot \sqrt{N} + N + Q \cdot log_2 N)$, sau $\mathcal{O}(U \cdot \sqrt{N} + Q \cdot \sqrt{N})$.

Subtask-urile 5, 6, 7 și Soluția de 100 de puncte:

Se dorește ca prelucrarea celor U *update*-uri și modificarea corectă a șirului inițial de valori să se realizeze în complexitatea de timp totală: $\mathcal{O}(U + N)$.

Vom utiliza 3 variabile, după cum urmează: Sum_X = suma tuturor x care apar în cele U *update*-uri; Sum_K = suma tuturor K care apar în cele U *update*-uri; $Diff$ = suma tuturor termenilor de forma $i \cdot K$ care apar în cele U *update*-uri.

Ne putem imagina că toate cele U *update*-uri ar avea $i = 1$, și, astfel, putem modifica în $\mathcal{O}(1)$ fiecare înălțime dintre cele N, folosind o metodă similară cu $SumeParțiale$ sau $ȘmenulluiMars$: $final_i = initial_i + Sum_X + (i \cdot Sum_K - Diff)$, pentru fiecare înățime i, cu $1 \leq i \leq N$.

Pentru tratarea eficientă a celor U *update*-uri, în $\mathcal{O}(1)$ fiecare, și aplicarea metodei *brute* pentru cele Q *query*-uri ($\mathcal{O}(N \cdot Q)$), se poate satisface Subtask-ul 5. Îmbunătățind prelucrarea query-urilor folosind *Șmenul lui Batog*, se poate trece și Subtask-ul 6.

Pentru 100 de puncte, în plus față de cele descrise anterior, se vor folosi *Arbori de Intervale* sau *Range Minimum Query* pentru cele Q query-uri.

3 Problema Inno

Propunător: prof. Dan Pracsiu, Liceul Teoretic Emil Racoviță Vaslui

Soluție parțială – 12 puncte

Pentru fiecare pereche de indici (i, j) cu $1 \leq i \leq j \leq N$ se elimină secvența $a[i], a[i+1], \ldots, a[j]$ și se verifică dacă $a[1]\&a[2]\&\ldots\&a[i-1]\&a[j+1]\&\ldots\&a[N]$ are cel puțin K biți de 1. Complexitatea temporală a acestei soluții este $\mathcal{O}(N^3 \cdot log(N))$.

Soluție parțială – 35 de puncte

Construim vectorii *st* și *dr* de lungime N în care:

- $st[i] = a[1]\&a[2]\&\ldots\&a[i]$
- $dr[i] = a[i]\&a[i+1]\&\ldots\&a[N]$

Ca și la soluția anterioară, se verifică dacă în urma eliminării unei secvențe $a[i], a[i+1], \ldots, a[j]$, șirul rămas este un șir *inno*.

Trebuie să verificăm dacă $a[1]\&a[2]\&\ldots\&a[i-1]\&a[j+1]\&\ldots\&a[N]$ are cel puțin K biți de 1. Dar $a[1]\&a[2]\&\ldots\&a[i-1] = st[i-1]$, iar $a[j+1]\&\ldots\&a[N] = dr[j+1]$. Deci se testează dacă $st[i-1]\&dr[j+1]$ are cel puțin K biți de 1.

Complexitatea temporală va fi în acest caz $\mathcal{O}(N^2 \cdot log(N))$.

Soluție oficială – 100 de puncte

Ne vom folosi in continuare de vectorii *st* și *dr* definiți anterior:

Notăm cu:

- x = cea mai din dreapta poziție cu proprietatea că $st[x]$ are cel puțin K biți de 1 (dacă $a[1]$ are mai puțin de K biți de 1, atunci $x = 0$)
- y = cea mai din stânga poziție cu proprietatea că $dr[y]$ are cel puțin K biți de 1 (dacă $a[N]$ are mai puțin de K biți de 1, atunci $y = N + 1$)

Apar cazurile:

1. $x = N$, adică $st[n]$ are cel puțin N biți de 1, deci întregul șir este șir *inno*. În acest caz, orice secvență poate fi eliminată (mai puțin întreg șirul, dar se poate elimina în schimb secvența vidă), deci soluția problemei este în acest caz $N \cdot (N + 1)/2$.
2. $x = 0$ și $y = N + 1$. În acest caz nu există nicio secvență
3. $1 \leq x < N$ și $y = N + 1$. În acest caz șirurile *inno* se obțin eliminând orice secvență de forma $a[i], a[i+1], \ldots, a[N]$, cu $i = 2, \ldots x + 1$, deci răspunsul la problemă este x.
4. $x = 0$ și $1 < y \leq N$. Atunci șirurile *inno* se obțin prin eliminarea secvențelor de forma $a[1], a[2], \ldots, a[i]$, cu $i = y - 1, \ldots, N - 1$, deci răspunsul la problemă este $N - y + 1$.

5. $x \geq 1$ și $y \leq n$. Din aceste condiții rezultă și că $x < y$, altfel ne-am afla în cazul 1. Deci în mod obligatoriu trebuie eliminată o secvență care conține pe $a[x+1], a[x+2], \ldots, a[y-1]$. Soluția problemei este dată de:

   (a) eliminarea secvențelor de forma $a[1], a[2], \ldots, a[i]$, cu $i = y-1, \ldots, N-1$ ($N-y+1$ soluții)

   (b) pentru fiecare $i = 1, \ldots, x$, se caută cea mai din stânga poziție p, cu $p = y, \ldots, N$, cu proprietatea că $st[i]\&dr[p]$ are cel puțin K biți de 1; se elimină secvențe de forma $a[i+1], a[i+2], \ldots, a[j]$, cu $j = p-1 \ldots, N$ ($N-j+2$ soluții). Pentru fiecare $i = 1, \ldots, x$ determinarea poziției p se poate face fie prin căutare binară, fie prin tehnica *two pointers*.

Complexitatea algoritmului este $\mathcal{O}(NlogN)$, unde logaritmul se obține prin determinarea numărului de biți de 1.

Deși nu este necesar pentru obținerea punctajului maxim, putem îmbunătăți aceasta soluție în continuare.

Funcția de calculare a numărului de biți (dese ori întâlnita sub numbele de *popcount*), poate fi implementată în complexitate temporală $\mathcal{O}(1)$ în diferite moduri:

1. Folosind o tabelă de valori: Știind că numerele noastre sunt numere pe 32 de biți, putem să creăm o tabelă *cnt*, astfel încât $cnt[i]$ reprezintă numărul de biți de 1 ai numărului i, doar pentru numere de maxim 16 biți (numerele de la 0 la 65535). Această tabelă poate fi calculată cu ajutorul recurenței $cnt[i] = cnt[i/2] + (i\&1)$.

   Acum, pentru a calcula pentru orice număr x de 32 de biți căți biți de 1 are aprinși este suficient să calculăm câți biți sunt aprinși În prima jumătate a biților numărului și câți sunt aprinși în a doua jumătate. Pentru a realiza acest lucru ne vom folosi de tabela *cnt* astfel:

   $$\text{popcnt} = cnt[x >> 16] + cnt[x \;\&\; 65536]$$

2. Folosind funcții deja existente: în cazul compilatorului de C/C++ folosit în cadrul concursului – gcc/g++, această funcție se numește `__builtin_popcount`.

3. Implementând funcția *popcount* folosind operații pe biți. Puteți găsi mai multe detalii despre implementarea acestei funții optim *aici*

Folosind o astfel de implementare, complexitatea temporală a soluției devine $\mathcal{O}(N)$.

Echipa

Setul de probleme pentru această rundă a fost pregătit de:

- prof. Boian Dumitru Flavius, Colegiul Național ”Spiru Haret”, Târgu-Jiu
- prof. Bunget Mihai, Colegiul Naţional ”Tudor Vladimirescu”, Târgu-Jiu
- prof. Cheșcă Ciprian, Liceul Tehnologic ”Grigore C. Moisil”, Buzău
- prof. Costineanu Veronica Raluca, Colegiul Național ”Ștefan cel Mare”, Suceava
- prof. Dumitrascu Dan Octavian, Colegiul National Dinicu Golescu, Câmpulung
- prof. Iordaiche Cristina, Liceul Teoretic ” Grigore Moisil” Timisoara, Timișoara
- prof. Lica Daniela, Centrul Județean de Excelență, Prahova
- prof. Nicoli Marius, Colegiul Național ”Frații Buzești”, Craiova
- prof. Nodea Gheorghe-Eugen, Colegiul Național ”Tudor Vladimirescu”, Târgu-Jiu
- prof. Piț-Rada Ionel-Vasile, Colegiul Național ”Traian”, Drobeta-Turnu Severin
- student Pop Ioan-Cristian, Universitatea Politehnica București, Facultatea de automatică și calculatoare.
- prof. Pracsiu Dan, Liceul Teoretic Emil Racoviță, Vaslui
- student Tulbă-Lecu Theodor-Gabriel, Universitatea Politehnica București, Facultatea de automatică și calculatoare.