Descrierea soluţiilor

Problema 1. Robot

Autor: stud. Eduard-Lucian Pîrţac, Facultatea de Informatică, Universitatea "Al. I. Cuza" Iaşi

Soluţia Medium

Soluția pentru versiunea *Medium* a robotului constă în calcularea următorilor vectori:

- `dr1[i]` = cel mai mare `j` (cu $i \le j \le n$) astfel încât secvența $v_{i..j}$ conține doar valori cu frecvența `1`;
- `dr2[i]` = cel mai mare `j` (cu $i \le j \le n$) astfel încât secvența $v_{i..j}$ conține doar valori cu frecvența ≤ 2.

Aceștia se pot calcula separat folosind tehnica *two pointers*. Să calculăm spre exemplu vectorul `dr1`. Vom porni cu două variabile `l=1` și `r=0` și mereu păstrăm într-un map valorile din secvența $v_{l,r}$ (deci și frecvența acestora).

Pentru a calcula `dr1[l]`, extindem variabila `r` spre dreapta cât timp toate valorile rămân cu frecvența `1`. În final, am determinat valoarea lui `dr1[l]` care este `r`.

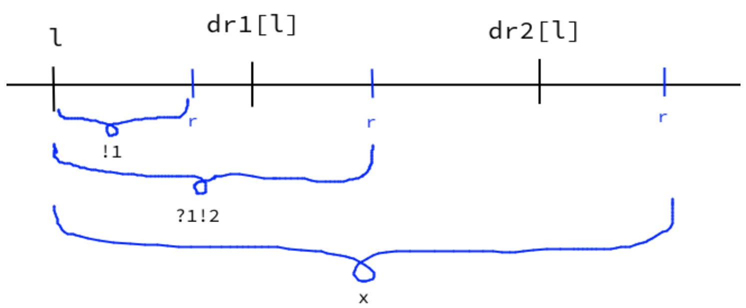
După, trecem la următorul pas cu `l=l+1` și repetăm din nou. Observați că putem refolosi valoarea lui `r` de la pasul anterior, deoarece vectorii `dr1` și `dr2` sunt *monotoni* (adică `dr1[i] ≤ dr1[i+1]`, pentru orice $1 \le i \le n-1$).

Vom rula același algoritm și pentru a construi vectorul `dr2`, singura schimbare e că variabila `r` o vom extinde spre dreapta cât timp toate valorile rămân cu frecvența ≤ 2.

Complexitatea acestui algoritm este `O(n)` sau `O(n*log(n))` (depinde de structura de date folosită - `unordered_map` sau `map`), deoarece variabilele `l` și `r` se deplasează numai spre dreapta, prin urmare se fac exact `2n` pași. Complexitatea `O(n*log(n))` se poate obține și normalizând valorile, astfel că se putea folosi un simplu vector de frecvență în loc de structuri de date din STL.

Pentru a răspunde acum la întrebarea `l r`, ne vom folosi de valorile `dr1[l]` și `dr2[l]`. Dacă `r ≤ dr1[l]`, știm că secvența $v_{l..r}$ conține doar valori cu frecvența `1`. Dacă `dr1[l] < r ≤ dr2[l]`, știm că secvența $v_{l..r}$ conține valori cu frecvența ≤ 2 (și cel puțin o valoare cu frecvența exact `2`). Dacă `dr2[l] < r`, atunci știm că există valori cu frecvența ≥ 3.

Complexitatea de a răspunde la o întrebare este `O(1)`. Prin urmare, complexitatea finală a acestei soluții este `O(n+q)` sau `O(n*log(n)+q)` și obține 80 de puncte.



Soluţia High — Varianta optimă

Vom extinde soluția de la versiunea Medium a robotului, adică încă vom folosi tehnica *two pointers* ca să găsim răspunsurile corecte cu `!1` și `x`. Dar, când suntem în cazul `dr1[l] < r ≤ dr2[l]`, cum decidem care răspuns dintre `!2` și `!1!2` trebuie să alegem?

Haideți să vorbim puțin despre operatorul *XOR*. Acest operator, notat în continuare cu $\oplus$, este un operator logic care primește două valori la intrare și returnează o valoare.

În cel mai simplu caz, intrările/ieșirile sunt valori booleene, adică adevărat sau fals. Returnează adevărat dacă și numai dacă intrările diferă, adică una este falsă și cealaltă este adevărată. Prin urmare, returnează fals dacă și numai dacă intrările sunt identice.

Operatorul XOR poate fi aplicat și pentru numere, nu doar pentru valori booleene. Astfel, atunci când îl aplicăm între două numere, operația se efectuează pe fiecare bit corespunzător: bitul rezultat va fi `1` doar dacă biții de la aceeași poziție în cele două numere sunt diferiți, altfel va fi `0`.

Din acest lucru rezultă două proprietăți interesante ale XOR-ului:

- $x \oplus x = 0$, pentru orice $x \in \mathbb{Z}$;
- $x \oplus 0 = x$, pentru orice $x \in \mathbb{Z}$.

Rețineți că suntem în cazul `dr1[l] < r ≤ dr2[l]`, deci valorile au frecvența ≤ 2. Fie $S = v_l \oplus v_{l+1} \oplus \ldots \oplus v_r$. Să analizăm următoarea afirmație:

➢ Toate valorile au frecvența 2 $\Rightarrow$ $S=0$.

Această afirmație este corectă și se poate demonstra cu ușurință folosind proprietățile de mai sus. Totuși, afirmația inversă nu este corectă — dacă știm că $S=0$, nu putem deduce că toate valorile au frecvența 2. Un exemplu e secvența `1,2,3`; aceasta are $S=0$, și totuși valorile nu au frecvența 2.

Putem face în așa fel încât și afirmația inversă să fie corectă, **cu probabilitate extrem de mare**. Să luăm fiecare valoare din șirul inițial o singură dată și să-i asociem un număr întreg, generat aleator uniform, din intervalul $[0,\ 2^x)$, unde $x \ge 0$. Să înlocuim acum fiecare valoare din șirul inițial cu cea asociată. Să analizăm următoarea afirmație pe șirul nou, cu echivalență în loc de implicație:

➢ Toate valorile au frecvența 2 $\Leftrightarrow$ $S=0$.

Implicația inițială rămâne în continuare adevărată. Probabilitatea ca implicația inversă să fie adevărată crește odată cu variabila `x`. Pentru $n,q \le 200000$, alegerea $x=32$ are șanse aproape 0 să dea un răspuns eronat.

Prin urmare, soluția este să calculăm sumele XOR parțiale ale noului șir de numere, și să răspundem la întrebări în complexitate `O(1)`. Complexitatea finală a acestei soluții este `O(n+q)` sau `O(n*log(n)+q)` și obține 100 de puncte.

Soluţia High — Varianta eficientă

Problema se poate rezolva și folosind algoritmul lui Mo. Întrebările sunt sortate în așa fel încât să se poată face trecerea de la o întrebare la alta într-un timp cât mai bun. Nu mai putem păstra într-un unordered_map/map valorile și frecvențele lor, ca la celelalte soluții, deoarece complexitatea este deja destul de mare și soluția nu va primi punctaj maxim. Suntem nevoiți să normalizăm valorile din șir, astfel încât să folosim un simplu vector de frecvență. Complexitatea finală a acestei soluții este O((n+q)*sqrt(n)) și obține 100 de puncte.

Problema 2. Patru

Autori:
stud. Ştefan Palcu, Facultatea de Informatică, Universitatea "Al. I. Cuza" Iaşi
stud. Robert Popa, Facultatea de Informatică, Universitatea "Al. I. Cuza" Iaşi
stud. George Tănăsucă, Facultatea de Informatică, Universitatea "Al. I. Cuza" Iaşi

1. Soluția pentru cazul când numerele din șir sunt până la 10^6 și $n \le 10^4$

Trebuie să determinăm de câte ori găsim patru indici i<j<k<p astfel încât a[i]*a[j]*a[k]=a[p]. Pentru aceasta parcurgem cu p elementele șirului începând de la poziția 4 și până la n. La fiecare pas p, elementele a[1], a[2], ..., a[p-1] le avem deja memorate în vectorul de frecvențe fr (deci fr[i] reține numărul de apariții ale valorii i, i=1..10^6). Aflăm divizorii lui a[p] și îi memorăm în vectorul d de lungime k (pentru numerele de la 1 la 10^6 numărul maxim de divizori este 240 – îl are 720720). Ordonăm crescător șirul de divizori și acum alegem orice pereche (d[i], d[j]) cu proprietatea că i ≤ j. Verificăm dacă a[p] este divizibil cu produsul d[i]* d[j] și în acest caz putem determina care este al treilea număr din produs, anume x= a[p]/(d[i] * d[j]). În acest moment avem trei numere (d[i], d[j], x) care au produsul egal cu a[p]. Pentru nu a obține aceleași soluții de mai multe ori, trebuie să avem d[i] ≤d[j] ≤x. Apar cazurile:

1. d[i] = d[j] = x și în acest caz la numărul total de soluții adăugăm fr[x]*(fr[x]-1)*(fr[x]-2)/6
2. d[i] = d[j] și în acest caz la numărul total de soluții adăugăm fr[d[i]]*(fr[d[i]]-1)/2 *fr[x]
3. d[i]=x, caz asemănător cu al doilea, adăugând la soluție fr[x]*(fr[x]-1)/2 *fr[d[j]]
4. d[j]=x, caz asemănător cu al doilea, adăugând la soluție fr[x]*(fr[x]-1)/2 *fr[d[i]]
5. d[i] ≠ d[j], d[i] ≠ x și d[j] ≠ x, caz în care adăugăm la soluție fr[d[i]]* fr[d[j]]* fr[x]

Complexitatea soluției este O(n sqrt(MAX)), unde MAX≤10^6.

2. Soluția pentru cazul când numerele din șir sunt până la 2^{31}-1 și $n \le 2000$

Expresia v[i] * v[j] * v[k] = v[p] o rescriem echivalent: v[i] * v[j] = v[p] / v[k].

Construim toate perechile (v[i], v[j]) cu i < j și la fel reținem perechile (v[p],v[k]), adică vom reține toate produsele v[i]*v[j] și rapoartele v[p]/v[k] cu proprietatea că v[p] este divizibil cu v[k]. Trebuie deci să identificăm, având grijă la relația i < j < k < p, de câte ori v[i]*v[j] este egal cu v[p]/v[k]. Complexitatea obținută este fie O(n^2), fie O(n^2 * log n). Să explicăm cum se obțin aceste complexități.

A. Soluția de complexitate O(n^2)

Iterăm k de la 3 la n-1. La fiecare pas k, adăugăm într-un *unordered_map* M toate produsele a[i] * a[k-1], cu i=1..k-2 și cu proprietatea că a[i] * a[k-1] este un produs în int. Acum parcurgem toate numerele a[p], cu p=k+1..n și cu a[p] divizibil cu a[k] și practic adăugăm la soluție M[a[i] * a[k-1]].

B. Soluția de complexitate $O(n^2 * \log n)$

Această soluție este utilă atunci când structurile de date *unordered_map* sau *map* nu sunt cunoscute. Soluția de la punctul A funcționează la fel, numai că în loc de *unordered_map* reținem produsele a[i] * a[k-1] într-un vector pe care-l menținem sortat. Acum, pentru orice pereche (a[p], a[k]) unde a[p] este divizibil prin a[k], căutăm prin vectorul sortat de câte ori apare valoarea x=a[p]/a[k]. Efectuăm două căutări binare pentru a determina p1=cea mai din stânga poziție unde se găsește valoarea x și p2= cea mai din dreapta poziție unde se găsește valoarea x. Adăugăm (p2-p1+1) la numărul total de soluții.

3. Soluție alternativă de aproximativ 90 de puncte pentru cazul când numerele din șir sunt până la 2^{31}-1 și n≤2000

Efectuăm o așa-zisă normalizare a valorilor din șir. Ținem cont de faptul că un număr natural nenul de tip int poate avea maximum 9 factori primi în descompunerea sa. Acest lucru înseamnă că numărul total posibil de factori primi distincți din cele n numere este de cel mult 20.000. Ştim că există peste 78.000 de numere prime mai mici sau egale cu 10^6, deci vom asocia celor cel mult L≤20.000 de numere prime valori din lista celor mai mici L numere prime. Să dăm un exemplu.

Să presupunem că şirul de numere este a=(115, 17, 81, 55). Atunci numerele prime distincte care apar ca factori în aceste numere sunt 3,5,11,17,23. Acestora li se vor asocia numerele prime cele mai mici: 2,3,5,7,11, deci șirul inițial a=(115, 17, 81, 55) se transformă în a=(3*11, 7, 2*2*2*2, 2*3) = (33, 7, 16, 6).

Suntem astfel în situația de a obține un șir de lungime n≤2000 și în care valorile sunt acum cel mult 10^6, deci putem aplica soluția de la 1.

Soluția nu obține 100 de puncte, deoarece sunt cazuri în care această normalizare nu funcționează. De exemplu, dacă în șir există valoarea 223.092.870 = 2 * 3 * 5 * 7 * 11 * 13 * 17 * 19 * 23. Valoarea normalizată este aceeași, din acest motiv soluția cu vector de frecvență nu funcționează.