

O idee des întâlnită la problemele care necesită să se verifice dacă o valoare poate fi mediană într-o secvență este să se considere un șir în care pe pozițiile unde apar valori mai mare decât aceasta să se pună 1, iar pe cele unde apar valori mai mici -1. Această formă permite diverse interogări și se dovedește a fi utilă și în cazul acestei probleme.

Datorită restricțiilor impuse (secvența pentru care o valoare candidat trebuie să fie mediană + valorile din input sunt distincte), condiția din enunț se rezumă la a verifica pentru o valoare dată dacă există o secvență din șirul cu 1 și -1 care să aibă suma 0 și lungimea cel puțin K. Odată ce s-a verificat condiția pentru o valoare, trebuie găsită și o modalitate de a tranziționa rapid către restul valorilor rămase de verificat.

Este destul de intuitiv că valorile ar trebui considerate drept posibili candidați în ordine crescătoare / descrescătoare, pentru că astfel, la fiecare pas, structura descrisă anterior necesită o singură actualizare.

Pentru a verifica dacă există o secvență dintr-un șir format doar cu valorile 1 și -1 care să aibă suma 0 și lungimea cel puțin K (cum am spus anterior că este necesar), vom face întâi observația că acest lucru este echivalent cu a avea 2 sume parțiale pe prefixe egale, iar indicii acestora să fie la distanță cel puțin K și intervalul determinat de acestea să conțină candidatul curent. Să considerăm în continuare poziția candidatului curent în șirul inițial dat, notată în continuare cu **pos**. Pentru a rezolva restricția dată de K ușor, putem trata 2 cazuri separate: când intervalul determinat de sumele parțiale are capătul drept în range-ul $[pos, pos + K)$, respectiv în $[pos + K, N]$. Primul caz impune din cauza lui K o restricție suplimentară asupra range-ului (prefixului defapt) în care se poate afla capătul stânga, dar dacă am putea să-l rezolvăm în $O(K)$ am rezolva toată problema în $O(N * K)$, cum pare a fi intended din restricții. Pentru al doilea caz, prefixul în care capătul stânga se poate afla este constant, fiind mereu $[1, pos]$, deci acesta nu ridică probleme suplimentare.

Pare totuși că pentru a rezolva ambele cazuri avem nevoie de acces la o structură de date care pentru problema noastră trebuie să ne returneze "rapid" dacă 2 range-uri dintr-un șir au cel puțin o valoare comună. La prima vedere acest lucru ar putea părea imposibil de realizat în timp logaritm, dar, pentru situația noastră specifică se dovedește a fi realizabil cu o observație în plus asupra valorilor cu care operăm defapt. Șirul pe care trebuie să verificăm dacă două intervale disjuncte au cel puțin o valoare comună este un șir de sume parțiale peste un alt șir format doar din valorile 1 și -1. Acest șir al sumelor de 1 și -1 are proprietatea că este o funcție **discret-continuă**, adică pentru situația noastră este de ajuns pentru un interval să îi aflăm minimul și maximum pentru a ști toate valorile întregi care apar în acesta. Astfel, query-ul se reduce la a afla minimul și maximum pentru cele 2 range-uri și apoi să se afle dacă intervalele formate de acestea se intersectează. Pentru a simula update-urile și query-urile dorite, structura de date ideală este astfel un **arbore de intervale**.

Singurul caz la care mai trebuie să fim atenți este cel pe care dorim să-l rezolvăm în $O(K)$. Dacă am face K query-uri pe structura de date, am obține complexitatea totală $O(N * K * \log N)$, care nu obține punctajul maxim. Pentru a optimiza acest proces, putem să obținem toate

cele K valori de interes **deodată**, într-un singur query pe arborele de intervale în complexitate $O(K + \log N)$, obținând astfel complexitatea dorită pentru toată problema, $O(N * K + N * \log N)$.