**Problema Constant**

**Autor : Florin Chirica - Colegiul National de Informatica "Tudor Vianu" Bucuresti**

Vom incepe prin a rezolva problema pentru un singur query. Sa presupunem ca avem foarte multe unitati de timp la dispozitie. Presupunand ca putem sta intr-o pozitie cat vrem, o strategie ar fi sa mergem pana la pozitia de valoare maxima, iar restul timpului sa stam acolo. Din pacate, restrictiile problemei ne obliga sa ne mutam fie in stanga, fie in dreapta la urmatoarea unitate de timp. Putem generaliza ideea: sa ne mutam minimul necesar de pozitii, adica o pozitie, apoi sa ne intoarcem in pozitia dorita. Astfel, cand avem mult timp la dispozitie, ar fi optim sa ne oprim in prima pozitie i pentru care x[i] + x[i – 1] e maxim (si sa facem drumul 1 -> 2 -> ... -> i -> i – 1 -> i -> i – 1 etc)

Daca ne-am putea “teleporta” pana la pozitia i, aceasta ar fi strategia optima. Din pacate, trebuie sa platim un “pret” pentru a ajunge la pozitia i (trebuie sa parcurgem intai x[1] + x[2] + .... + x[i]). In unele cazuri, este optim sa nu mergem chiar pana in pozitia i cu x[i] + x[i – 1] maxim, pentru ca am colecta prea putin ajungand pana acolo. In schimb, ar fi optim sa ne oprim intr-o pozitie j < i si sa facem traseul 1 -> 2 -> 3 -> ... –> j -> j – 1 -> j etc.

Cum putem remedia asta? Pentru fiecare pozitie i, putem calcula costul maxim astfel incat folosesc doar primele i pozitii. Daca iteram i-ul, atunci singurul lucru care ar da costul maxim ar fi sa mergem pana la suma maxima. Daca prin absurd am obtine un cost mai bun oprindu-ne inainte de pozitia sumei maxime, atunci acea posibilitate ar fi tratata la un i mai mic, deja iterat. Deci localizam prima pozitie j <= i astfel incat x[j] + x[j – 1] sa fie maxim, valoarea optima va fi x[1] + x[2] + ... + x[j] + ((t – (j – 1)) % 2) \* x[j – 1], unde cu t am notat cate unitati de timp avem la dispozitie. Pentru a scapa de pozitia j, e suficient sa iteram toate pozitiile i si sa calculam maximul valorilor x[1] + x[2] + ... + x[i] + ((t – (i – 1)) % 2) \* x[i – 1]. Normal, pozitiile i astfel incat x[i] + x[i – 1] nu este maximul primelor i pozitii sunt inutile, dar aceasta formula ne da un algoritm simplu in O(n) per query.

Pentru a rezolva mai multe query-uri vom folosi “Divide et Impera”. Termenul ((t – i + 1) % 2) \* x[i] este cel care ne incurca, deoarece formula este diferita pentru diferite valori ale lui t. Daca am grupa query-urile dupa paritate (tratam intai query-urile cu t par, apoi cele cu t impar), aceasta problema dispare, iar formulele raman identice indiferent de valoarea lui t.

In momentul de fata, avem doar functii liniare, asa ca putem rezolva cu “Convex Hull Trick” problema. In continuare voi descrie o metoda mai simpla de implementat. Vom renumerota query-urile in ordine crescatoare a unitatii de timp. Fie f(i) = care este pozitia j din drumul 1->2->...->j->j-1->j->j-1 etc pentru al i-lea query, in ordinea sortarii crescatoare dupa valoare.

Observam ca f(1) <= f(2) <= ... <= f(numar\_queryuri). Motivul e simplu: cand creste numarul unitatilor de timp, avem 2 posibilitati: ramanem in pozitia de dinainte sau ne miscam la stanga/dreapta. Trebuie sa aratam ca nu are rost sa te misti la stanga, atat timp cat pozitia nu e egala cu f(T). Singurul motiv pentru care ne-am misca de pe pozitia curenta (x[i] + x[i – 1]) ar fi sa gasim un alt maxim (x[j] + x[j – 1]). Acest maxim se poate afla doar in dreapta pozitiei i. Daca j < i, atunci pozitia i nu ar mai fi maxima. Deci, cand creste timpul, fie ramanem in pozitia i, fie mergem intr-o pozitie j >= i. Astfel, daca t1 <= t2, f(t1) <= f(t2) pentru ca i <= j.

Imediat ce observam inegalitatea asta, putem reduce timpul de executie folosind divide et impera (nu numai la problema asta, ci la orice problema in care se observa inegalitatea!!!).

Fie o functie f(st, dr, i, j) o functie care rezolva query-urile cuprinse intre st si dr, folosind doar pozitii cuprinse intre i si j. Calculam med = (st + dr) / 2. Vom rezolva al med-lea query prin brute force. Iteram k intre i si j si retinem pozitia poz = k care maximizeaza expresia pt al med-lea query, impreuna cu valoarea maxima.

Pentru query-urile din intervalul [st, med – 1], valorile indicilor care merita verificati vor fi cuprinse intre i si poz (conform inegalitatii de mai sus). Pentru query-urile din intervalul [med + 1, dr], valorile indicilor vor fi cuprinse intre poz si j.

Astfel, apelam f(st, med, i, poz) si f(med + 1, dr, poz, j).

Cat e complexitatea? La prima vedere, nu am facut decat sa optimizam brute force-ul. In schimb, vom grupa functiile f dupa lungimea intervalelor. Lungimile posibile vor fi numar\_query-uri, numar\_query-uri / 2, ..., numar\_queryuri / (2^i), deci vor fi aproximativ log(numar\_query-uri) lungimi. Observam ca indicii lui f sparg sirul in multe intervale disjuncte. Deci, pentru o lungime L, indicii i si j din apel reuniti vor da sirul de numere. Fiecare element al sirului apare de maxim 2 ori in reuniune. Cum sunt O(n) elemente si O(log(nr\_queryuri)) lungimi distincte, complexitatea devine O(n \* log(nr\_queryuri)).