Problema 3: Confuzie – soluție

Stud. Andrei Pârvu – Universitatea Politehnica București

1. **40p – O(N^2)**

Se face o parcurgere DFS a arborelui, începând din rădăcină, și pentru fiecare nod se reține părintele acestuia. Când avem de procesat un query, urcăm pe arbore din tată în tată, reținând ultimul nod colorat în negru găsit.

2. **50p – O(N^2) optimizat**

Soluția este asemănătoare cu cea de 40 de puncte, numai ca se face pacurgerea drumului începând cu nodul x, coborând în arbore. Pentru a ști pe care din fii trebuie coborât, pentru fiecare nod vom reține, când facem parcurgerea DFS, timpul de intrare, respectiv timpul de ieșire dintr-un nod. Astfel, când dorim să vedem daca un nod anume este strămoș al lui y, este suficient să verificăm daca timpul de intrare in y este cuprins între timpul de intrare și cel de ieșire al nodului pentru care facem verificarea. Cu această tehnică, putem parcurge drumul de sus în jos, de fiecare dată ducându-ne în fiul care este strămoș al lui y, oprindu-ne atunci când gasim un nod colorat în negru (astfel nefiind nevoie să parcurgem tot drumul ânte x și y).

O altă soluție ce poate obține acest punctaj presupune folosirea unui arbore echilibrat (containerul *set* din STL), în care vom reține doar înalțimile nodurilor colorate în negru. Atunci când avem o interogare pe drumul dintre x și y, selectăm din set, folosind metodele *lower_bound* si *upper_bound*, doar intervalul cuprins între înalțimile celor două noduri. Primul nod găsit ce este strămos al lui y, va fi răspunsul la query.

3. **70p – O($N\log^2 N$)**

Fiecărui nod din arbore colorat cu negru la un moment îi putem asocia valoarea 1, iar fiecărui nod colorat în alb valoarea 0. Vom nota cu suma[x] suma tuturor nodurilor de pe drumul de la rădăcină la nodul x. Pentru un drum de la x la y, răspunsul căutat va fi cel mai de sus nod pentru care suma[tata[x]] + 1 = suma[nod]. Se poate observa că acest nod poate fi căutat binar pe intervalul de noduri dintre x si y. Pentru a executa eficient operațiile de sumă (cu tot cu eventualele update-uri) trebuie folosit un arbore indexat binar, iar pentru a afla al i-lea strămoş al nodului y, trebuie menținut, pentru fiecare nod din arbore, al 2^k-lea strămoș al său.

4. **100p – O(NlogN)**

Pentru soluția optimă, trebuie realizată o împărțire a arborelui pe drumuri, numită heavy-path decomposition. Dacă ne aflăm într-un nod x, atunci path-ul în care este el prezent va continua în fiul cu subarborele cel mai greu (cu cele mai multe noduri), rămânând ca din ceilalți fii să înceapă un path nou. Astfel, se garantează că drumul de la y la x, dintr-un query, va trece prin maxim logN drumuri.

Pentru fiecare drum vom reține un arbore echilibrat (se poate folosi containerul *set* din STL) pentru a reține înalțimile nodurilor colorate cu negru. Atunci când parcurgem path-urile de la y la x, pentru fiecare path, mai puțin ultimul, este suficient să ne uităm la primul element din set (cel cu înalțimea cea mai mică) și să verificăm dacă acesta se află mai sus decât nodul din care s-a intrat în path-ul curent. Pentru ultimul path, nu mai putem să luăm elementul cu înalțimea cea mai mică, deoarece s-ar putea să fie mai sus în arbore decât x. De aceea, vom folosi metoda *lower_bound* a containerului *set* pentru a afla nodul cu înalțimea minimă mai mare sau egală decât înălțimea lui x (operația se implementează într-un arbore echilibrat de căutare folosind o parcurgere din rădăcină). Din toate path-urile parcurse, răspunsul va fi nodul selectat din cel mai de sus path (bineinteles, se poate ca unele pathuri să nu aibă niciun astfel de nod, fie că în path nu există niciun nod colorat cu negru, fie că cel mai de sus nod este mai jos decât punctul de intrare in path).

Pentru fiecare path, mai puțin ultimul, inspectarea minimului are complexitatea O(1), iar pentru ultimul path complexitatea este O(logN).

5. **100p – O(NlogN)**

O soluție alternativă sa bazează pe o liniarizare a arborelui. Daca facem un dfs din rădăcina şi scriem fiecare nod o dată când întrăm în el (înainte să parcurgem fii) şi o dată când ieşim din el vom obtine un şir cu exact 2 * N elemente. Dacă un nod este selectat atunci pe prima poziție a acestui nod(să o notăm begin[nod]) în sub acest şir vom avea valoarea -1, iar pe a doua poziție(să o notăm end[nod]) vom avea valoarea -1, altfel ambele valori vor fi 0.

Acum dacă avem două noduri x şi y, x stramoş al lui y, avem end[y] < end[x] si totodata că suma numerelor scrise sub liniariyare pe intervalul (end[y], end[x]), inclusive capetele, reprezintă cate noduri sunt selectate pe drumul de la x la y. Nodul pe care îl cautam este chiar nodul scris in liniarizare pe prima poziție t (end[y] <= t <= end[x]) astfel încât suma pe intervalul (end[y], t) este egala cu suma pe intervalul (end[y],end[x]).

Ca sa găsim acest t vom ține un arbore de intervale pe valorile de sub liniarizare(-1, 0 sau 1), iar in fiecare nod vom tine suma numerelor din interval şi prefixul de suma maxima din acel interval. Update-ul doar adaugă sau scade unu intr-o poziție. Pentru query vom parcurge cele logN intervale in care se imparte intervalul de query de la stanga la dreapta si vom cauta primul interval cu proprietatea ca suma pe intervalele de dinainte + bestul din intervalul curent da exact valoarea cautata. Apoi vom cobora in acest interval pe aceeasi idée.

Complexitatea este O(logN) pe query si O(N) in memorie.