Solutie arbfind

Pentru fiecare arbore construim o reprezentare sub forma de sir cu ajutorul unei parcurgeri euleriane : la vizitarea pt prima data a unui nod adaugam la reprezentare un simbol S (start), la coborarea pe o muchie punem un simbol D, la intoarcere punem U si la parasirea finala a unui nod punem F.
Avand reprezentarile a doi arbori e usor de observat ca similaritatea celor doi este egala cu numarul de S-uri ale prefixului lor comun.
Reprezentarile subarborilor fiecarui nod din arborele de evaluat sunt incluse in reprezentarea intregului arbore.
Calculam si concatenam reprezentarile celor doi arbori, retinand pentru fiecare nod din arborele de evaluat pozitia de unde incepe reprezentarea subarborelui corespunzator acestuia si sortam toate sufixele folosind suffix arrays. Putem calcula pentru fiecare nod prefixul comun maximal dintre reprezentarea subarborelui acestuia si cea a modelului in timp logN.
Pentru a calcula numarul de S-uri din bucata comuna ne trebuie conform descrierii de mai sus sa retinem si un arbore de intervale. Putem optimiza pastrand doar simbolurile D si U, iar lungimea prefixului comun va fi 2*numarul de muchii din partea comuna, astfel nu mai e necesar arborele de intervale.

Timpul total de executie este O((N + M) * log(N + M))

Solutie – Cosmin
Folosim 2 biti pentru reprezentare 1 daca mergem in jos pe o muchie, si 0 cand urcam in parcurgerea arborilor in adancime. Acum practic din nou trebuie sa calculam cel mai lung prefix comun a doua stringuri. In loc sa facem aceasta operatie naiva in O(n) putem face in O(n / log n) daca facem potrivire pe biti. Alta idee ar fi sa ne folosim de tabele de dispersie. Pentru un sir S[i] codul lui hash va fi (s[0] + s[1] * 3 + s[2] * 3^2 + ... + s[n] * 3^n) % numar_prim_mare . Acum vom calcula pentru cel de-al doilea arbore, codul hash pentru fiecare prefix. Pentru a determina codul hash pentru o subsecventa S[i..j] a unui sir daca stim codurile hp[i] pentru toate prefixele, atunci putem obtine codul hash al subsecventei in O(1) h(i..j) = ((h[j] – h[i-1]) / (q^i)) % numar_prim_mare.
Acum problema lcp-ului poate fi rezolvata printr-o cautare binara.

Timpul total de executie este O((N + M) * log(N + M) * log (numar_prim_mare)) pentru ca inversul fata de un numar prim mare il facem in log (numar_prim_mare).