Solutie in O(N * Q * L)
Tinem o dinamica, dp[node][suffix]. In cate moduri putem alege un sufix de lungime 'suffix' care sa fie in subarborele lui node.
Raspunsul e in dp[0][1]
Ca sa calculam dinamica
dp[node][suffix] = sum(dp[child][suffix])
Adaugam raspunsurile date de nodul curent, cu label-ul setat.
dp[node][label[node]] += dp[node][label[node] + 1]

Consideram updateurile pe tot sirul o singura data.
Se aplica algoritmul pana se termina updateurile.

Solutie in O(Q * L)
Daca trebuie schimbat label-ul nodului 'node' din 'x' in 'y', dinamica din subarbore se schimba foarte putin.
Putem aborda schimbarea label-ului unui nod in felul urmator. Scadem cate solutii treceau prin label-ul vechi, si adunam cate solutii trec prin label-ul nou.
Putem calcula o dinamica noua, up[node][prefix] = In cate moduri putem alege un prefix care porneste din radacina(0) si se termina cu valoarea 'prefix',
undeva inaintea nodului nostru.

Astfel, ca sa vedem cate solutii am pierde putem aborda problema in felul urmator
#1 Scoatem drumurile adaugate de nodul curent din dinamica
dp[node][label[node]] -= dp[node][label[node] + 1]
#2 Calculam in cate moduri putem sa ne legam in sus, si in jos.
scadem din resultatul total
      up[node][label[node] - 1] * down[node][label[node] + 1]
#3 inlocuim label-ul
label[node] = new_label[node]
#4 adaugam noile solutii
up[node][label[node] - 1] * down[node][label[node] + 1]
#5 updatam dinamica "up"
up[node][label[node]] += up[node][label[node] - 1]

Dinamica din subarbore se calculeaza pe label-urile vechi (cele de dinaintea updateurilor) in O(N * L) cu memorie O(N * L)
Dinamica pe lant-ul pana la radacina tot in O(N * L) cu memorie O(N * L)

Rezolvarea dinamicii catre radacina in O(N) cu O(N) memorie.
Recurenta la dinamica e
up[node][label] = up[father[node]][label]
up[node][label[node]] += up[node][label[node] - 1]

Astfel, nu trebuie sa o copiem pe toata.
Doar modificam valoarea dinamicii in df cand intram in el, iar mai apoi scadem valoarea adaugata cand iesim din traversare.
Astfel, complexitatea devine O(N) cu O(N) memorie.

Rezolvarea dinamicii in subarbore in O(N)
Observatie #1
Pentru fiecare nod, ne intereseaza doar 2 valori din dinamica lui de subarbore.
down[node][label[node] + 1] si down[node][new_label[node] + 1]

Astfel, nu avem nevoie de toata dinamica la fiecare pas, doar de 2 valori. Pe masura ce calculam dinamica, putem tine minte valorile de care o sa avem nevoie, astfel, avand posibilitatea de a scadea memoria.

Observatia #2
Daca privim dinamica pe subarbore al unui nod, va exista o valoare minima a lui 'label' de la care valorile din dinamica o sa inceapa sa fie nenule.
Nu se poate sa avem solutii pentru [label] dar nu sa avem pentru [label + 1], deoarece orice solutie in [label] trebuie sa continue in [label + 1]
Astfel, pentru fiecare nod, putem tine minte 'label'-ul minim pentru care avem drum.

Cand calculam dinamica pentru in nod, putem alege sa 'furam' dinamica unui copil care are valoarea label-ului cat mai mica.
Astfel, luam dinamica lui in O(1) si pentru ceilalti copii doar le aplicam dinamica peste (adunand valorile, stiind ca toate au labeluri mai mari)
Retinem cele 2 valori cautate in observatia #1
Updatam dinamica sa contina si nodul curent cu valoarea lui veche.

Astfel, complexitatea devine O(N) pentru a calcula dinamica.
Daca ne gandim la un copil caruia ii vom aplica dinamica peste tata, acesta avand X valori in dinamica nenule, am folosii O(X) timp.
Daca NU am aplica dinamica lui peste tata, se poate observa ca complexitatea de timp nu ar creste in viitor(nu ar afecta tatal din punct de verede al complexitatii) deoarece tatal avea toate valorile pe care copilul le adauga. Astfel, el nu creea elemente noi care trebuie aplicate poate peste alte dinamica.
Asta nu se aplica la abordarile generale de a merge-ui multimea mai mica in cea mai mare, caz in care solutie se face in O(NlogN) timp.
Faptul ca orice copil care se aplica peste dinamica tatalui nu ii aduce

elemente noi in multime creeaza amortizarea la O(N), deoarece un copil va fi aplicat o singura data.