

Problema cli - descriere soluție

Autori: Andrei Bălțatu, Lucian Bicsi

Soluție 1 (Lucian Bicsi)

O primă observație în rezolvarea problemei constă în faptul că, având fixate cele K cuvinte, ordinea optimă de parcurgere a acestora este cea **lexicografică**. Cu alte cuvinte, putem sorta lexicografic în prealabil cele N cuvinte, astfel soluția devenind un subșir în ordinea sortată.

Putem folosi programarea dinamică: $DP[i][j]$ = soluția pentru primele i cuvinte (în ordinea sortată), din care am ales j și al j -lea ales este cel de pe poziția i .

Pentru a o calcula, fixăm $i' < i$ și “augmentăm soluția” $DP[i][j] = \min(DP[i][j], DP[i'][j-1] + 2 * (\text{len}(i) - \text{LCP}(i', i)))$ ($\text{len}(i)$ este lungimea cuvântului i , iar $\text{LCP}(i', i)$ este lungimea celui mai lung prefix comun între i și i').

Formula este rezultatul următoarei strategii: dacă decidem să scriem încă un cuvânt, este optim să îl scriem fix înainte să apucăm să ștergem din prefixul comun ultimelor două cuvinte.

Bineînțeles, vom calcula pentru perechea (i', i) o singură dată $\text{LCP}(i', i)$ în complexitate $O(\min(\text{len}(i), \text{len}(i')))$ (brut) sau $O(\log(\min(\text{len}(i), \text{len}(i'))))$ (cu hashuri) sau $O(\text{input_size})$ amortizat (incremental).

Soluția are complexitate $O(\text{input_size} * \log(N) + N^2 * K)$.

Soluție 2 (Lucian Bicsi)

Cheia rezolvării complete a problemei este observația că, având fixate cele K cuvinte, soluția este egală cu **dublul numărului de noduri din tria** formată de cele K cuvinte. Cu alte cuvinte, trebuie să găsim tria de dimensiune minimă care “încadrează” K cuvinte.

Aici sunt foarte importante două observații:

1. Orice trie formată dintr-o submulțime a celor N cuvinte este în sinea ei conținută în tria celor N cuvinte
2. Pornind de la o trie arbitrară, **soluția efectivă se obține printr-o parcurgere Euler a triei** (pornind, bineînțeles, din rădăcină)

Cele două observații ne determină să concluzionăm că, **pentru orice submulțime de cuvinte, le putem scrie în mod optim alegând o parcurgere Euler a triei asociată lor care este subșir în parcurgerea Euler a triei tuturor cuvintelor**. Cu alte cuvinte, soluția este un subșir al șirului obținut parcurgând Euler tria-mamă.

Noul insight ne permite să folosim următoarea dinamică: $DP[i][j]$ = soluția pentru cazul în care șirul rămas corespunde nodului i din trie și am scris cu succes j cuvinte. În final, în $DP[root][*]$ se vor afla cele K răspunsuri.

Soluția elegantă este în felul următor:

Considerăm $root$ rădăcina triei-mamă și $Ans[]$ un vector global de care ne vom folosi ca să ne construim incremental soluția și care NU este dependent de nodul curent. (inițial, $Ans[]$ are 0 pe poziția 0 și infinit pe celelalte poziții). Algoritmul în pseudocod este următorul:

```
procedură DFS(nod) :  
    • Pentru fiecare cuvânt care se termină în nod,  $Ans[i + 1] = Ans[i]$   
    • Aplicăm  $DP[nod] = Ans$  (element cu element)  
    • Pentru fiecare fiu  $v$  al lui nod :  
        ◦ incrementăm vectorul  $Ans$  (element cu element)  
        ◦ DFS( $v$ )  
        ◦ Incrementăm vectorul  $Ans$  (element cu element)  
        ◦ Aplicăm  $DP[nod] = Ans = \min(DP[nod], Ans)$  (element cu element)  
  
DFS( $root$ )  
Output  $Ans$ 
```

(De ce) este corectă?

În adevăr, algoritmul nostru “simulează” o parcurgere Euler a triei-mamă și, la fiecare pas în parcurgerea Euler, actualizează $DP[nod]$ cu Ans , care se dovedește că nu este altceva decât dinamica de la poziția anterioară (în parcurgere). Din considerentele de pe pagina anterioară, concluzionăm că **soluția este corectă**.

Complexitatea soluției este $O(input_size + trie_size * K)$.

Pentru a optimiza soluția de mai sus, putem argumenta că are rost să aplicăm instrucțiunile 2. și 3d. doar în nodurile în care se “ramifică” tria (nodurile cu minim 2 fii) și în nodurile în care se termină măcar un cuvânt și să realizăm incrementările 3a. și 3c. *lazily*. Se poate demonstra ușor că sunt maxim $2 * N - 1$ astfel de noduri.

Complexitate finală: $O(input_size + N * K)$.

Referință: https://en.wikipedia.org/wiki/Euler_tour_technique

Soluție 3 (Bălțatu Andrei)

Soluția se bazează tot pe observațiile făcute mai sus dar ținem următoarea dinamică

$dp[i][j] = 2 * \text{lungimea triei minime dacă am alege } j \text{ noduri care reprezintă sfârșit de cuvânt din subarborele nodului } i.$

Recurența acestei dinamici se aseamănă cu cea de la problema rucsacului: $dp[i][j] = \min(dp[fiu_1][x_1] + dp[fiu_2][x_2] + \dots + dp[fiu_k][x_k] + 2)$ unde $x_1 + x_2 + \dots + x_k = j - 1$ (deoarece nodul i nu a fost luat în calcul la size-ul triei), $x_1, x_2, \dots, x_k \geq 0$.

Complexitatea finala: $O(\text{input_size} + \text{trie_size} * K * K)$

Soluție 4 (Bălțatu Andrei)

Se ține ca în soluția 1 dinamica $dp[i][j] = 2 * \text{lungimea triei minime dacă aș alege cuvinte lexicografic mai mici sau egale cu } i, \text{ al } j\text{-lea fiind cuvântul } i.$

Se observă că dacă pentru fiecare cuvânt am trece prin toți strămoșii nodului său am avea $O(\text{input_size})$. Deci dacă ne-am ține informații în strămoși asupra unor dp-uri și actualizăm $dp[i][j]$ luând în calcul toate informațiile din strămoși, soluția va avea $O(\text{input_size} * K)$.

Ce ne-ar fi de ajutor ar fi că pentru fiecare nod (care e și strămoș a altor noduri) să ne ținem la pasul curent $val[\text{strămos}][j] = \min(dp[x][j])$ unde x e un nod din subarbore și mai mic lexicografic decât nodul i (cel care îl actualizăm acum). După $dp[i][j] = \min(val[\text{strămos}][j - 1] + 2 * (\text{level}[i] - \text{level}[\text{strămos}])),$ unde $\text{level}[i] = \text{adâncimea nodului } i \text{ în tria globala. Desigur } dp[i][j] \text{ poate fi actualizat și cu } dp[\text{strămos}][j - 1] \text{ dacă nodul strămoș este sfârșit de cuvânt.}$

Cum actualizăm $val[\text{nod}]$? Stim că noi mergem pe trie în ordine lexicografică (în ordinea sortată cuvintelor) deci când ieșim din nod în DFS vom putea actualiza nodul "tată" deoarece din acel moment nodul i mereu va fi mai mic lexicografic ca următoarele noduri ce vor fi actualizate. Codul parcurgerii DFS va arată așa:

procedură DFS(nod) :

- Actualizare $dp[\text{nod}]$ cu $val[\text{strămos}]$
- Pentru fiecare fiu apelăm DFS(fiu)
- Actualizăm vectorul $val[\text{tată}], val[\text{tată}][j] = \min(val[\text{tată}][j], dp[\text{nod}][j])$

Complexitatea finala: $O(\text{input_size} + \text{trie_size} * K)$