Descrierea soluției - caps

Autor, prof. Gheorghe Manolache,
Colegiul Național de Informatică, Piatra Neamț

Problema construiește un șir după regula precizată. Se observă că se poate construi șirul cu o operație mai simplă și anume prin adăugarea la sfârșitul șirului precedent a șirului obținut cu operația CAPS. Se poate observa că litera de pe poziția N este dată de paritatea numărului de biți din reprezentarea binara a lui N, dacă vom considera prima poziție zero. Pentru a calcula numărul de apariții al acestei cifre se observă că pe fiecare secvență de lungime dublă a șirului inițial aven mereu șirul inițial și șirul CAPS. Se precalculează numărul de apariții al fiecărei litere în șirul inițial și în șirul CAPS. Pentru fiecare interogare, numărul de apariții al literei de pe poziția N se obține calculând câte grupe duble (formate șir și șir CAPS avem până la poziția N. Dacă mai rămâne un rest atunci se determina din ce șir face parte acesta și până la ce poziție se caută litera cu ajutorul valorilor precalculate.
Se pot obține punctaje diferite după complexitatea implementării(descriere **Adrian Budău**).

1) Pentru 15 puncte (+10 din oficiu), $Q \leq 1.000$ $N \leq 3.000$

Se poate pentru fiecare query să se construiască efectiv șirul de caractere, și să afișeze caracterul de pe poziția N și apoi sa se numere în O(N) de cate ori apare acel caracter până la poziția N.

2) Pentru cele încă 20 de puncte ($N \leq 100.000$).

Se poate face inițial de la începutul șirului de caractere până la primele 100.000 de caractere și apoi folosind sume parțiale pentru fiecare caracter se poate afla în O(1) pentru fiecare query al câtelea este. Complexitatea este O(1) pe query și O(N * SIGMA) unde SIGMA este mărimea alfabetului (în problema aceasta 52 = 26 * 2).

3) Pentru celelalte 20 de puncte ($K \leq 3000$ si $Q \leq 1000$).

Se poate afla cel mai mic număr de forma X = K*4^P astfel încât $N \leq X$. De aici se poate afla cu o soluție recursiva litera corespunzătoare poziției N.

La fiecare pas consideram cele 4 intervale [1, K * 4^(P - 1)], [K * 4^(P - 1) + 1, 2K * 4^(P - 1)] [2K * 4^(P - 1), 3K * 4^(P - 1)] si [3K * 4^(P - 1), K*4^P]. N se poate încadra numai în unul din aceste 4 intervale, și dacă se încadrează în primul sau ultimul putem continua recursiv cu N relativ la acel interval, iar dacă se încadrează în al doilea sau al treilea se tine minte că trebuie inversat răspunsul final (din litera mica in litera mare și viceversa) și se continua recursiv cu N relativ la acel interval.

De exemplu daca N = 39 și K = 7.

Găsim X = 7 * 4^2. Obținem cele 4 intervale [1, 28] [29, 56] [57, 84] [85, 112]. 39 se încadrează în [29, 56], se tine minte ca trebuie inversat răspunsul și se continuă recursiv cu N = 39 - 29 + 1 = 11 și INVERSIUNI = 1

Găsim X = 7 * 4. Obținem cele 4 intervale [1, 7] [8, 14] [15, 21] [22, 28]. 11 se încadrează în [8, 14], se tine minte că trebuie inversat răspunsul și se continua recursiv cu N = 11 - 8 + 1 = 4 și INVERSIUNI = 2

X = 4, deci vrem al 4-lea caracter inversat de 2 ori, deci răspunsul ar fi S[4] unde S este șirul de litere inițial.

Daca se și precalculează de cate ori apare fiecare caracter în fiecare string de lungime K * 4^P până când K*4^P depășește 10^18 se poate de fiecare dată când se coboară în recursivitate să se adune la răspuns de câte ori a apărut intervalele de dinainte (în exemplu daca N = 57, atunci ar fi trebuit să se adauge la răspuns de cate ori apare caracterul găsit în șirul de mărime 7 * 4, adică intervalul [1, 28] și de cate ori apare inversul lui într-un șir de mărime 7 * 4, adică intervalul [29, 56]).

La final când N ≤ K se poate face în O(K), obținând astfel complexitate O(K + log^2 N) per query.

4) Combinând ideile de la 2 si 3 putem sa reduce complexitatea direct la O(log^2 N) per query cu O(K * sigma) precalculare. Pentru a reduce de la O(log^2 N) la O(log N) putem să nu îl căutam la fiecare pas în recursie pe X = K*4^P cu N ≤ X. După ce l-am găsit pe cel pentru N din fișierul de intrare se verifica mai întâi X / 4, apoi X / 4^2 si așa mai departe.

O alat soluție, mai ușoara de codat se bazează pe următoarea observație:

Daca transformam N în 2 valori: PART și INDEX unde PART reprezintă numărul concatenării șirului inițial (cu sau fără inversiune din litera mică în literă mare și viceversa) și INDEX poziția în această concatenare atunci răspunsul este S[INDEX] sau invers(S[index]) în funcție de PART. Mai exact dacă numărul de biți de 1 din descompunerea lui PART este par atunci răspunsul este S[INDEX] altfel este invers(S[index]).

Iar pentru a numără de câte ori apare litera răspuns în toate concatenările 1, 2, ... , PART - 1 se poate face următoarea observație:
- Daca PART - 1 e par atunci oricare 2 concatenări de pe pozițiile 2k + 1 si 2k+2 sunt diferite. Asta înseamnă că jumate din concatenări sunt normale, și jumate sunt inversele lor.
  Deci la răspuns s-ar adăuga (PART - 1) / 2 * APARITII[răspuns] + (PART - 1)/2 * APARITII[invers(răspuns)] + de câte ori apare în ultima parte.
- Daca PART -1 e impar, atunci PART - 2 e par și se aplică același raționament și se mai adaugă ce e in PART - 1, verificând dacă e invers sau nu (folosind trucul de mai sus, care numără numărul de biți de 1).

Soluția obținuta este deci O(K * SIGMA) precalculare și O(log N) pe query. Se poate obține O(1) pe query folosind fie funcții non standard (__builtin_parity) sau o dinamica pe biți, dar așa ceva nu era necesar pentru 100 de puncte.

Iar o alta metoda de a calcula de câte ori apare un caracter până la o poziție în șirul original (pe lângă cea cu sumele parțiale pentru fiecare literă) este să se țină pentru fiecare litera pozițiile pe care se află acea literă și să se folosească căutare binara să se afle câte poziții sunt mai mici ca una anume.

Astfel se obține O(K) precalculare și O(log N + log K) pe query, dar în practică se comporta la fel ca soluția precedentă.