

Informatyczny kącik olimpijski (107): Palindromiczny podciąg

W tym miesiącu omówimy zadanie *Palindromic Subsequence*, które pojawiło się na konkursie SRM 708 na platformie Topcoder.

Palindrom to słowo, które nie zmienia się, gdy czytamy je od prawej do lewej, np. „kajak” lub „abba”.

W zadaniu dane jest słowo s o długości n ($n \leq 3000$) znaków. Dla każdego indeksu i mamy znaleźć liczbę palindromicznych podciągów s , które zawierają i -tą literę s . Wyniki należy wypisać modulo $(10^9 + 7)$.

Innymi słowy: dla ustalonego i mamy 2^{n-1} sposobów na usunięcie niektórych z pozostałych $n - 1$ liter w s . Interesuje nas liczba sposobów, dla których nieusunięte litery utworzą palindrom (i -ta litera jest, oczywiście, nieusunięta).

Zacznijmy od rozwiązania prostszego zadania – policzmy wszystkie palindromiczne podciągi w danym słowie.

Złym pomysłem jest przetwarzanie słowa od lewej do prawej, bo zapewne musielibyśmy dokładnie pamiętać już wybrany podciąg, by później wybrać te same litery w odwróconej kolejności (by powstał palindrom). Zamiast tego policzymy wynik dla każdego przedziału, zaczynając od przedziałów o długości 1.

Niech $in[i][j]$ oznacza liczbę palindromicznych podciągów przedziału $[i, j]$ (tzn. palindromicznych podciągów pod słowa złożonego z liter s_i, s_{i+1}, \dots, s_j). Szczegółem implementacyjnym jest decyzja, czy należy liczyć puste podciągi. Zacznijmy analizę od wstępnego wzoru (wcale nieprzypadkowo podobnego do wzoru na liczenie sum prefiksowych w macierzy!):

$$in[i][j] = in[i+1][j] + in[i][j-1] - in[i+1][j-1].$$

Dodajemy tu liczby palindromicznych podciągów w przedziałach $[i+1, j]$ i $[i, j-1]$, po czym odejmujemy te w przedziale $[i+1, j-1]$, bo policzyliśmy je podwójnie. Policzyliśmy w ten sposób wszystkie palindromiczne podciągi, które występują też w którymś z krótszych przedziałów. Pozostaje jednak nieuwzględniony przypadek, gdy podciąg zawiera i -tą i j -tą literę – taki podciąg nie występuje ani w $[i+1, j]$, ani w $[i, j-1]$. Skoro i -ta i j -ta litera są pierwszą i ostatnią literą w przedziale, muszą one być równe, by powstał palindrom. Jeśli tak rzeczywiście jest, to do $in[i][j]$ powinniśmy jeszcze dodać $in[i+1][j-1]$, bo do każdego z podciągów przedziału $[i+1, j-1]$ możemy teraz dodać i -tą i j -tą literę z przodu i z tyłu, co da nam nowy dłuższy palindrom.

Mówimy, że w palindromie pierwsza litera jest sparowana z ostatnią, druga z przedostatnią itd. Zastanówmy się, jak dla pary indeksów i, j ($i \leq j$) policzyć takie palindromiczne podciągi, które zawierają indeksy i i j , a ponadto w nich te dwie litery są ze sobą sparowane. Musi zatem zachodzić równość $s_i = s_j$, a wzięte indeksy z przedziału $[i+1, j-1]$ muszą tworzyć palindromiczny podciąg. Takich podciągów jest dokładnie $in[i+1][j-1]$. Pozostaje nam jeszcze uwzględnić liczbę możliwych zewnętrznych rozszerzeń, a więc policzyć liczbę takich wyborów, że litery wzięte na lewo od i (te z mniejszymi indeksami) odpowiadają literom wziętym na prawo od j . Tę liczbę oznaczmy przez $out[i-1][j+1]$. Za chwilę pokażemy, jak ją obliczać.

Niech $out[i][j]$ oznacza liczbę palindromicznych podciągów, które zawierają indeksy z przedziałów $[1, i]$ oraz $[j, n]$, a także takich, że litery z $[1, i]$ są sparowane z tymi z $[j, n]$ (czyli te dwa przedziały zawierają po tyle samo wybranych liter). Podobnie jak do liczenia wartości in , możemy zacząć od rekurencyjnego wzoru:

$$out[i][j] = out[i-1][j] + out[i][j+1] - out[i-1][j+1],$$

po czym dodać $out[i-1][j+1]$, jeśli $s_i = s_j$. Uwzględniamy tu podciągi, które zawierają indeksy z $[1, i-1]$ i $[j, n]$ oraz podciągi, które zawierają indeksy z $[1, i]$ i $[j+1, n]$. Do obu tych grup podciągów należą takie, które zawierają indeksy z $[1, i-1]$ i $[j+1, n]$, więc takie policzyliśmy dwukrotnie, co musimy odjąć. Dodatkowo, jeśli litery s_i i s_j są równe, uwzględniamy podciągi, które je zawierają.

Zauważmy, że w każdym palindromicznym podciągu zawierającym i -tą literę, litera ta jest sparowana z dokładnie jedną literą, która musi być, oczywiście, równa s_i . By znaleźć wynik zadania dla i , wystarczy więc przeiterować wszystkie indeksy j ($i \leq j$ i $s_i = s_j$) i zsumować iloczyny $out[i-1][j+1] \cdot in[i+1][j-1]$ (taki iloczyn jest równy liczbie palindromicznych podciągów, w których indeksy i i j są sparowane).

Zarówno obliczenie tablic in i out , jak i końcowe iterowanie par indeksów, ma złożoność czasową $O(n^2)$. Tablice in i out mają rozmiar kwadratowy. Czytelnikowi Z Ograniczoną Pamięcią pozostawiamy zastanowienie się nad implementacją w pamięci liniowej.

Kamil DĘBOWSKI