Podciągi

Autor zadania i opisu: Karol Pokorski

Zacznijmy od tego, że rozwiązanie zawsze istnieje, więc uwaga o wypisywaniu `NIE` to jest pic na wodę.

Problem odwrotny

Zadajmy sobie naturalne pytanie: mając dane słowo, jak sprawdzić, ile ono ma różnych podciągów. Załóżmy, że policzyliśmy już wynik dla słowa s. Jeśli dopiszemy do tego słowa literkę a, która nie występowała w s, to wynik przemnoży się przez 2. Jeśli jednak literka a już wcześniej występowała w s, to tak licząc pewne słowa policzymy podwójnie. Wypadałoby odjąć te słowa, które już wcześniej policzyliśmy i kończyły się na a. Potrzeba i wystarczy odjąć wynik dla prefiksu s kończącego się na ostatnie a.

Dostajemy więc proste programowanie dynamiczne:

$$DP[n] = 2 \cdot DP[n-1] \ \ (\ \ - DP[i] \ ),$$

gdzie i to pozycja ostatniego wystąpienia znaku $s[i]$ w słowie $s[1..n-1]$.

Problem właściwy

No dobrze, ale co to pomaga? Umiemy napisać sprawdzaczkę wyjść, ale chcemy wzorcówkę. Zauważmy, że jeśli oczekiwana liczba podciągów n jest parzysta, to możemy trywialnie sprowadzić problem do mniejszego ($\frac{n}{2}$) poprzez dopisanie do rozwiązania mniejszego problemu nowej, unikalnej literki (w dowolnym miejscu – na początku, na końcu lub w środku).

Super, ale co gdy n jest nieparzyste? Niestety nie ma już takiego ładnego gadżetu, który by w takim przypadku sprowadził problem rozmiaru n do problemu istotnie mniejszego... czy na pewno? Trochę tak, ale nie do końca. Pójdźmy na razie mikroskopijny krok do przodu i zauważmy, że ten pomysł da się nieco uogólnić – jeśli n jest podzielne przez (małe) $k > 2$ to możemy do rozwiązania problemu dla $\frac{n}{k}$ podciągów dopisać (znowu w dowolnym miejscu) $k-1$ takich samych literek, które nie wystąpiły w rozwiązaniu $\frac{n}{k}$ i dostaniemy dokładnie n różnych podciągów.

Świetnie, czyli potrafimy rozwiązać każdy przypadek, w którym mamy n składające się jedynie z małych czynników pierwszych. Gorzej, gdy n zawiera jakiś duży czynnik, albo co gorsza jest liczbą pierwszą. Widać, że brakuje nam jakiejś redukcji addytywnej (jak na razie mamy tylko multiplikatywne).

Zanim do tego przejdziemy, zauważmy że dość trudno jest przewidzieć jak zmieni się wartość dynamika, gdy wstawimy do środka słowa jakąś literkę wcześniej już występującą. Dlatego pomysł na redukcję problemu oprzemy o obudowywanie wyniku zredukowanego problemu z lewej oraz prawej strony. Jak się okaże, mamy wystarczająco dużo „luzu", żeby sobie na to pozwolić.

Weźmy znowu przykład: Niech wynik dla s jest równy r. Wówczas jeśli napiszemy słowo asa (jak ostatnio zakładamy, że a jest znakiem niewystępującym w s) to wynik

przemnaża się przez 4... ale należy jeszcze odjąć 1 (dostaliśmy dokładnie jeden nowy podciąg powtórzony – pojedynczą literkę a). To jest kolejny sukces, bo pozwala nam trywialnie przekształcić problem rozmiaru $4n-1$ do problemu n. A zatem w kombinacji z poprzednimi pomysłami mamy już rozwiązane przypadki, gdy $n \equiv 0 \pmod 4$, $n \equiv 2 \pmod 4$ oraz $n \equiv 3 \pmod 4$. Niestety ponownie okazuje się, że nie jest tak prosto, gdy $n \equiv 1 \pmod 4$.

Ale kto nam kazał dopisywać tylko po jednej literce z każdej strony: równie dobrze możemy sobie dodać jakiś losowy ciąg x z lewej, losowy ciąg y z prawej. Zakładamy oczywiście nadal, że x i y nie współdzielą liter z s, ale nie zakładamy nic o powtarzalności/niepowtarzalności x względem y i na odwrót. Niech $r(x)$ oraz $r(y)$ są liczbami różnych podciągów słów x, y oraz s. Policzmy ile wynosi $r(xy) = A$ i porównajmy to z wynikiem $r(x) \cdot r(y) = B$. Dostaliśmy właśnie gadżet (dopisz x z lewej oraz y z prawej), który możemy użyć do przejścia z problemu rozmiaru $An - B$ do problemu rozmiaru n. Takich gadżetów można sobie wygenerować całą masę, jeśli tylko napiszemy dynamika, o którym mowa była wyżej. Słowa x, y można generować losowo (najlepiej, żeby były oczywiście krótkie – wtedy A jest małe, a każdy gadżet możemy zastosować tylko w $\frac{1}{A}$ przypadków). Okazuje się, że dla niektórych A dostajemy gadżety na mniej więcej połowę różnych możliwych wartości B, więc jeśli będziemy mieli tych gadżetów dostatecznie dużo, to wygenerujemy odpowiedź dla każdego testu.

Gadżety generowane są następująco:

- generujemy wszystkie istotnie różne słowa długości $\leqslant 7$,
- próbujemy każdą parę (x, y) wcześniej wygenerowanych słów jako lewy/prawy dopisek i obliczamy jaki gadżet z tego wychodzi,
- usuwamy duplikaty (tzn. gadżety o tym samym dzielniku i reszcie, a innej parze dopisków).

Generowanie gadżetów trwa na komputerze autora około dwóch sekund i generuje 691 344 gadżetów.

Dowód

Dobra wiadomość jest taka: autor ma dowód, że opisane rozwiązanie działa dla wszystkich $n \leqslant 10^{18}$.

Niestety jest też zła wiadomość: dowód jest z użyciem komputera, a jego wygenerowanie zajmuje na komputerze autora 30 minut.

Dowód działa tak:

- uruchamiamy program na dwóch miliardach losowych testów, żeby uporządkować gadżety w kolejności częstotliwości ich używania – lepiej najpierw sprawdzać te, które częściej się przydają (przyspiesza to czas sprawdzania),
- wybieramy liczbę antypierwszą $M = 10\,475\,665\,200$ (nieco więcej niż $\sqrt{10^{18}}$ – wybór podyktowany jest tym, że dalszy algorytm będzie działał w czasie proporcjonalnym do $M + \frac{10^{18}}{M} \cdot c$, gdzie c to pewna nie za duża stała),

- odrzucamy gadżety o dzielniku niepodzielnym przez M (nie będziemy mogli ich użyć w punkcie poniżej) – właśnie dlatego chcieliśmy, żeby M miało dużo dzielników (żeby jak najwięcej gadżetów móc używać w punkcie poniżej),
- generujemy dowód dla jak największej liczby reszt r modulo M, że wszystkie $n = kM+r$ są pokryte jakimś gadżetem (naiwnie sprawdzamy liczby r od 2 do $M+1$ czy istnieje jakiś gadżet, z wcześniej ograniczonego zbioru, który możemy zastosować – jeśli tak, to możemy go też zastosować do każdego $kM+r$) – dla $M+1$ udać się nam nie może (przystaje do 1 modulo wszystkie dzielniki gadżetów), ale dla wszystkich pozostałych reszt modulo M uda nam się znaleźć uniwersalny dla niej gadżet,
- dalej sprawdzamy naiwnie wszystkie liczby postaci $kM+1$ do 10^{18} czy znajdziemy dla każdej z nich (w już nieograniczonym zbiorze) odpowiedni gadżet,
- ostatecznie sprawdziliśmy każdą liczbę od 1 do 10^{18} – więc uznajemy, że rozwiązanie jest poprawne dla limitów postawionych w zadaniu.

Jeszcze mała uwaga: wbrew pozorom w tym zadaniu nie jest dopuszczalne stwierdzenie „*działa na dwóch miliardach testów, więc działa*" – liczby niestety nie są tak samo trudne, dla rozwiązania autora gdyby zmienić magiczną stałą 7 na 6 to dwa miliardy losowych testów przechodzi z bardzo dużym prawdopodobieństwem, a dzięki zastosowaniu sprawdzenia zgodnie z dowodem udało się znaleźć kontrprzykład (najmniejszy to: 15 738 449 126 401).

Program dowodowy jest w pliku `pod_prover.cpp`.

Uwagi

Wypada jeszcze zastanowić się czy rozwiązanie mieści się w założonym alfabecie oraz limicie znaków. Jeśli chodzi o alfabet to sprawa jest prosta – zastosowanie każdej nowej, wcześniej nie występującej literki mnoży liczbę podciągów przez 2, więc nigdy nie użyjemy ich więcej niż 59 (a założony alfabet pozwala nam na 62). Jeśli chodzi o limit znaków też jest łatwo – każdy gadżet co najmniej dzieli nam n przez 2 i dokłada $\leqslant 14$ znaków (po $\leqslant 7$ z każdej strony), więc łącznie zużywamy $\leqslant 14 \cdot 60 = 840$ znaków. Oczywiście te szacunki są bardzo zgrubne i w praktyce rozwiązanie mieści się w stu znakach.