Wojciech Rytter
Treść zadania, Opracowanie

Marcin Pilipczuk
Program

Dostępna pamięć: 32 MB.

OI, Etap III, dzień pierwszy, 06.04.2011

Okresowość

Bajtazar, król Bitocji, zarządził reformę nazwisk swoich poddanych. Nazwiska mieszkańców Bitocji często zawierają powtarzające się frazy, np. w nazwisku `Abiabuabiab` *dwukrotnie występuje fragment* `abiab`*. Bajtazar chce zamienić nazwiska swoich poddanych na ciągi bitów takiej samej długości jak ich oryginalne nazwiska. Chciałby przy tym w jakimś stopniu zachować sposób, w jaki w oryginalnych nazwiskach powtarzają się te same frazy.*

W dalszej części zadania, dla prostoty, będziemy utożsamiać wielkie i małe litery w nazwiskach. Dla dowolnego ciągu znaków (liter lub bitów) $w = w_1 w_1 \ldots w_k$ powiemy, że liczba naturalna p $(1 \leqslant p < k)$ jest okresem w, jeżeli $w_i = w_{i+p}$ dla wszystkich $i = 1, \ldots, k-p$. Przez $Okr(w)$ będziemy oznaczać zbiór wszystkich okresów w. Na przykład, $Okr($`ABIABUABIAB`$) = \{6, 9\}$, $Okr($`01001010010`$) = \{5, 8, 10\}$ *oraz* $Okr($`0000`$) = \{1, 2, 3\}$.

Bajtazar zdecydował, że każde nazwisko ma zostać zamienione na ciąg bitów:

- *tej samej długości co oryginalne nazwisko,*
- *o dokładnie takim samym zbiorze okresów co oryginalne nazwisko,*
- *ma to być najmniejszy (w porządku leksykograficznym[1]) ciąg bitów spełniający powyższe warunki.*

Na przykład, nazwisko `ABIABUABIAB` *powinno zostać zamienione na* `01001101001`, `BABBAB` *na* `010010`*, a* `BABURBAB` *na* `01000010`*.*

Bajtazar poprosił Cię o napisanie programu, który pomógłby w tłumaczeniu dotychczasowych nazwisk jego poddanych na nowe. W nagrodę będziesz mógł zachować swoje oryginalne nazwisko!

Wejście

W pierwszym wejściu standardowego wejścia znajduje się jedna liczba całkowita k — liczba nazwisk do przetworzenia $(1 \leqslant k \leqslant 20)$. Nazwiska są podane w kolejnych wierszach, po jednym w wierszu. Każde z nazwisk składa się z od 1 do 200 000 wielkich liter (alfabetu angielskiego).

W testach wartych łącznie 30% punktów każde nazwisko składa się z co najwyżej 20 liter.

Wyjście

Twój program powinien wypisać na standardowe wyjście k wierszy. W kolejnych wierszach powinny znajdować się ciągi zer i jedynek (niezawierające odstępów) odpowiadające kolejnym nazwiskom z wejścia. W przypadku, gdy dla danego nazwiska nie istnieje ciąg bitów zgodny z warunkami zadania, należy dla tego nazwiska wypisać „`XXX`*" (bez cudzysłowów).*

[1] Ciąg bitów $x_1 x_2 \ldots x_k$ jest mniejszy w porządku leksykograficznym od ciągu bitów $y_1 y_2 \ldots y_k$, jeżeli dla pewnego i, $1 \leqslant i \leqslant k$, mamy $x_i < y_i$ oraz dla wszystkich $j = 1, \ldots, i-1$ mamy $x_j = y_j$.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`3`	`01001101001`
`ABIABUABIAB`	`010010`
`BABBAB`	`01000010`
`BABURBAB`	

Rozwiązanie

Informacją wejściową jest słowo w długości n, pierwszym krokiem jest obliczenie zbioru okresów $Okr(w)$ tego słowa, po czym właściwie zapominamy o słowie w. Zbiór ten można łatwo obliczyć, korzystając z algorytmu na wyznaczanie tablicy tzw. prefikso-sufiksów związanej z algorytmem Knutha-Morrisa-Pratta (szukania wzorca). Tablica ta wielokrotnie pojawiała się w zadaniach z Olimpiady Informatycznej, patrz także opracowania zadań *Szablon* z XII Olimpiady i *Palindromy* z XIII Olimpiady. W drugim z podanych opisów jest uzasadniona własność, że słowo w ma okres p wtedy i tylko wtedy, kiedy ma prefikso-sufiks długości $n-p$, tzn. prefiks słowa długości $n-p$ jest również sufiksem słowa. Zakładamy zatem odtąd, że znamy zbiór okresów słowa wejściowego. Inaczej niż w treści zadania przyjmiemy, że $n \in Okr(w)$.

Zbiory okresów mają wiele ciekawych własności, w szczególności zachodzi następująca implikacja (*nwd* oznacza tutaj *najmniejszy wspólny dzielnik*).

Lemat 1 (o okresowości). $p, q \in Okr(w)$ *oraz* $p + q \leqslant n \;\Rightarrow\; nwd(p, q) \in Okr(w)$.

Lemat ten wystąpił już poprzednio w rozwiązaniach zadań olimpijskich, np. w drugim z wyżej wymienionych.

Tekst u nazywamy **pierwotnym** albo *nierozkładalnym*, gdy u nie ma okresu będącego jego właściwym dzielnikiem (mniejszym od długości u), zapisujemy to jako funkcję logiczną $Pierwotny(u)$. Na przykład $Pierwotny($`1010`$) =$ **false**, $Pierwotny($`1011`$) =$ **true**. Pozostawiamy Czytelnikowi jako proste ćwiczenie wyprowadzenie z lematu o okresowości następującego faktu.

Lemat 2 (o słowach pierwotnych). *Jeśli słowo nie jest pierwotne, to zmiana pojedynczego symbolu na dowolnej pozycji zamienia to słowo na pierwotne.*

Zamiast przetwarzać okresy, wygodniej robić to dla niepustych prefikso-sufiksów tekstu. Zakładamy, że cały tekst też jest swoim prefikso-sufiksem.

Oznaczmy ciąg długości kolejnych prefikso-sufiksów słowa w, posortowany rosnąco, przez $PS(w) = (p_1, p_2, \ldots, p_k)$, w szczególności $p_k = n$. Ciąg $PS(w)$ łatwo obliczyć, znając zbiór okresów, zachodzi bowiem: $Okr(w) = Okr(v) \;\Leftrightarrow\; PS(w) = PS(v)$.

Przykład 1. $Okr(w) = \{10,\ 20,\ 25,\ 27\} \;\Longrightarrow\; PS(w) = (2,\ 7,\ 17,\ 27)$.

Oznaczmy leksykograficznie minimalne słowo, którego ciągiem długości prefikso-sufiksów jest $(p_1, p_2, \ldots, p_i)$, przez:

$$P_i \;=\; MinLex(p_1, p_2, \ldots, p_i).$$

Słowo to jest prefikso-sufiksem wyniku długości p_i.

Opis algorytmu

Zasadnicza koncepcja algorytmu jest naturalna: ze względu na leksykograficzną minimalność narzuca się dopychanie tekstu jak największą liczbą zer. Takie podejście łatwo wymyślić na intuicję i zastosować, pomimo tego, że poprawność nie jest oczywista.

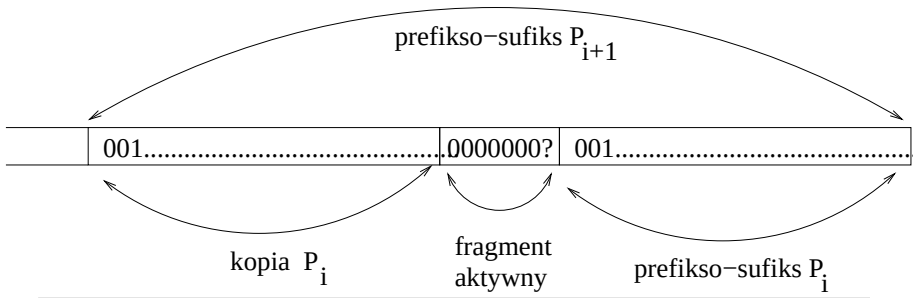
Algorytm opiera się na *zachłannym leksykograficznie* wpisywaniu fragmentów postaci 0^j lub $0^{j-1}1$ w te fragmenty tekstu, które nie są jednoznacznie wyznaczone przez prefikso-sufiksy w następującym sensie: każdy P_{i+1} musi zaczynać się od słowa P_i i kończyć się słowem P_i.

Początkowo mamy tekst długości n, którego wszystkie pola są „puste". Wypełniamy kolejne puste pola od końca, wpisując prefikso-sufiksy, poczynając od najkrótszego z nich, tzn. P_1.

Dla $PS = (p_1, p_2, \ldots, p_k)$, słowo P_{i+1} konstruujemy, startując od P_i i dopisując na początku P_i lub jego prefiks, jeśli jest za mało miejsca na całe P_i, patrz rysunek 1. Niezapełnione pola tekstu zastępujemy leksykograficznie pierwszym ciągiem bitów Δ, który nie wygeneruje nadmiarowego prefikso-sufiksu.

Obserwacja 1. Najdziwniejszą własnością algorytmu jest to, że wystarczy zawsze sprawdzić tylko dwie możliwości na brakujący fragment: $\Delta \in \{0^j,\ 0^{j-1}1\}$ dla $j = p_{i+1} - 2p_i$.

Dzięki tej obserwacji możliwy jest algorytm działający w czasie liniowym. Podstawowym elementem algorytmu jest sprawdzenie, którą z opcji na Δ wybrać. Można to zrealizować na wiele sposobów, w naszej wersji korzystamy z funkcji *Pierwotny*, inną możliwością jest sprawdzenie dla każdej alternatywy wprost, czy nie generuje się nadmiarowy prefikso-sufiks.



Rys. 1: Konstrukcja kolejnego prefikso-sufiksu w sytuacji, gdy $2p_i < p_{i+1}$. Końcową częścią tekstu jest P_i, zapełniamy P_{i+1}, z definicji prefikso-sufiksu prefiks P_{i+1} długości p_i jest równy P_i, wolne pola (fragment aktywny) zapełniamy jak największą liczbą zer, w miejsce '?' wstawiamy 0 lub 1.

Przykład 2. Opiszemy, w jaki sposób konstruujemy minimalne leksykograficznie słowo $MinLex(PS(w))$ dla naszego przykładowego ciągu prefikso-sufiksów $PS(w) = (2, 7, 17, 27)$:

Krok 1. Konstruujemy $MinLex(2) = \texttt{01}$.

Krok 2. Wiemy, że $MinLex(2,7) =$ `01???01`. Próbujemy wstawić minimalny leksykograficznie ciąg `000`, zastępując znaki '`?`'. Otrzymujemy $MinLex(2,7) =$ `0100001`, tekst ten jest zgodny z ciągiem prefikso-sufiksów $2, 7$.

Krok 3. $MinLex(2,7,17) = MinLex(2,7)$`???`$MinLex(2,7)$. Jeśli zastąpimy znaki '`?`' przez `000`, to otrzymany tekst `01000 01000 01000 01` ma nadmiarowy prefikso-sufiks długości 12. Zatem próbujemy zastąpić '`???`' przez następny w kolejności minimalnej ciąg `001`. Tym razem otrzymany tekst jest zgodny z ciągiem prefikso-sufiksów $2, 7, 17$. Zatem $MinLex(2,7,17) =$ `01000010010100001`.

Krok 4. Ponieważ $27 - 17 < 17$, więc wiemy, że wynikiem jest tekst, którego prefiksem i sufiksem jest $MinLex(p_1, \ldots, p_{i+1})$. Zatem końcowym wynikiem jest:

$$\begin{aligned} MinLex(2,7,17,27) &= \texttt{0100001001}\ MinLex(2,7,17) \\ &= \texttt{0100001001 01000010010100001}. \end{aligned}$$

Podaną metodę ilustruje poniższy pseudokod.

```
1:  function MinLex(p_1, p_2, ..., p_k)
2:  begin
3:    if p_1 = 1 then P_1 := 0 else P_1 := 0^{p_1 - 1}1;
4:    for i := 1 to k - 1 do begin
5:      j := p_{i+1} - 2p_i;
6:      if j ⩽ 0 then P_{i+1} := vP_i  { v jest prefiksem P_i długości p_{i+1} - p_i }
7:      else if Pierwotny(P_i 0^j) then P_{i+1} := P_i 0^j P_i
8:      else P_{i+1} := P_i 0^{j-1}1 P_i;
9:    end
10:   return P_k;
11: end
```

Poprawność algorytmu

Chcemy pokazać indukcyjnie (po i), że zbiorem prefikso-sufiksów skonstruowanego przez nas słowa P_i jest właśnie $\{p_1, p_2, \ldots, p_i\}$. Zauważmy, że w kroku indukcyjnym dla P_{i+1} wystarczy pokazać, że to słowo ma prefikso-sufiks długości p_i oraz że nie ma dłuższych nietrywialnych prefikso-sufiksów.

Na początku rozważmy przypadek $j > 0$. Załóżmy, że wówczas P_i zawiera co najmniej jedną jedynkę — w przeciwnym przypadku $P_i = 0^{p_i}$ i nie ma czego dowodzić. Poprawność algorytmu wynika z dwóch następujących faktów.

Fakt 1. *Niech $j > 0$ oraz P będzie dowolnym binarnym tekstem zawierającym co najmniej jedną jedynkę. Słowo $u = P\,0^j P$ ma (nadmiarowy) prefikso-sufiks o długości q, $|P| < q < |u|$, wtedy i tylko wtedy, gdy $P\,0^j$ nie jest pierwotny.*

Dowód: Przypuśćmy, że słowo u ma nadmiarowy prefikso-sufiks P' długości q. Ten prefikso-sufiks nie może się zacząć wewnątrz aktywnego fragmentu 0^j, bo P zawiera

jedynkę. Faktycznie, wówczas mielibyśmy $P' = P\ 0^i = 0^i P$ dla $i = q - |P|$, czyli P' miałoby okres długości i złożony z samych zer.

Z drugiej strony, jeśli P' zaczyna się w prefiksie P słowa u, to, na mocy lematu o okresowości, słowo u ma (mały) okres, który jest krótszy od $P\,0^j$ i jest dzielnikiem długości tego słowa. Wynika to stąd, że tekst u miałby okresy o długościach $|P\ 0^j|$ oraz $|u| - q \leqslant |P|$. Długość tekstu u jest nie mniejsza od sumy tych okresów i można wtedy zastosować lemat o okresowości.

Zatem jeśli $P\,0^j\,P$ ma nadmiarowy prefikso-sufiks, to $P\,0^j$ nie jest pierwotne. Łatwo widać implikację odwrotną, wtedy u ma nadmiarowy prefikso-sufiks o długości $|u| - p$, gdzie p jest (małym) pełnym okresem $P\,0^j$. To kończy uzasadnienie faktu. ■

Fakt 2. *Jeśli $P\,0^j$ nie jest pierwotny, to $P\,0^{j-1}1$ jest pierwotny.*

Dowód: Wynika to z lematu o tekstach pierwotnych. ■

Aby zakończyć uzasadnienie w tym przypadku, wystarczy zauważyć, że fakt 1 zachodzi także dla słów postaci $u' = P\ 0^{j-1}1\ P$.

Teraz pozostał nam do rozpatrzenia przypadek, że $j \leqslant 0$. W tym celu wystarczy wykazać następujący fakt, implikujący krok indukcyjny w tym przypadku. W jego dowodzie wykorzystujemy to, że wyjściowy zbiór prefikso-sufiksów $\{p_1, p_2, \ldots, p_k\}$ odpowiada jakiemuś istniejącemu słowu, tj. początkowemu słowu w.

Fakt 3. *Załóżmy, że słowo P_i ma zbiór prefikso-sufiksów $\{p_1, p_2, \ldots, p_i\}$. Jeśli $p_{i+1} - 2p_i \leqslant 0$, to słowo $P_{i+1} = vP_i$ ma prefiks P_i oraz nie ma nadmiarowych prefikso-sufiksów długości q, $p_i < q < p_{i+1}$.*

Dowód: Najpierw pokażemy pierwszą część tezy, czyli że P_{i+1} ma prefikso-sufiks P_i. Niech w_i i w_{i+1} oznaczają sufiksy słowa w długości, odpowiednio, p_i i p_{i+1}. Wystarczy przeprowadzić następujące rozumowanie. Jeśli chcemy pokazać, że P_{i+1} ma prefikso-sufiks P_i, czyli równoważnie, że P_{i+1} ma okres $p_{i+1} - p_i$, to wystarczy uzasadnić, że P_i ma taki właśnie okres, gdyż P_{i+1} zaczyna się prefiksem słowa P_i długości $p_{i+1} - p_i$. Na mocy założenia faktu, słowo P_i ma dokładnie taki sam zbiór okresów jak w_i, więc wystarczy pokazać, że w_i ma okres $p_{i+1} - p_i$. Tak jednak jest, gdyż w_i jest prefikso-sufiksem słowa w_{i+1}, które całe ma, w związku z tym, okres $p_{i+1} - p_i$. Rozumowanie zakończone sukcesem.

Musimy jeszcze pokazać, że P_{i+1} nie może mieć nadmiarowego prefikso-sufiksu o długości q, $q > p_i$. Wówczas P_{i+1} miałoby okresy $p_{i+1} - q$ oraz $p_{i+1} - p_i$, więc na mocy lematu o okresowości (ponieważ $j \leqslant 0$) P_{i+1} miałoby okres d będący dzielnikiem tych dwóch wartości. Stąd, stosując rozumowanie takie jak w pierwszej części dowodu, pokazujemy, że d byłoby także okresem słowa u_{i+1}, a więc także $p_{i+1} - q$ byłoby okresem tego słowa, czyli q byłoby jego prefikso-sufiksem, co daje sprzeczność. ■

Złożoność algorytmu

Podstawowym problemem jest sprawdzenie, czy tekst $P_i\ 0^j$ jest pierwotny. W miarę konstruowania coraz dłuższych prefikso-sufiksów, a więc jednocześnie prefiksów całego tekstu wynikowego, budujemy on-line tablicę *wszystkich* najdłuższych prefikso-sufiksów z algorytmu Knutha-Morrisa-Pratta. Dzięki tej tablicy znamy najkrótszy

okres i możemy sprawdzić, czy dany prefiks jest pierwotny — za pomocą lematu o okresowości można pokazać, że słowo jest pierwotne wtedy i tylko wtedy, gdy jego najkrótszy okres jest nim samym lub nie dzieli jego długości (patrz także wspomniane już opracowanie zadania *Palindromy* w książce [13]).

Jeśli obliczyliśmy już tablicę prefikso-sufiksów dla $P_i\ 0^{j-1}$, to sprawdzenie, czy $P_i\ 0^j$ jest niepierwotne, wykonujemy w czasie stałym: można pokazać, że jeśli $P_i\ 0^j$ jest niepierwotne, to najdłuższy prefikso-sufiks słowa $P_i\ 0^{j-1}$ przedłuża się do prefikso-sufiksu słowa $P_i\ 0^j$. Jeśli odpowiedź jest pozytywna, to obliczamy prefikso-sufiks on-line dla $P_i\ 0^{j-1}1$, a jeśli negatywna, to dla $P_i\ 0^{j-1}0$. Inaczej mówiąc, koszt „cofania się" jest za każdym razem $O(1)$, w sumie liniowy. Pozostaje „na czysto" koszt liczenia on-line tablicy prefikso-sufiksów dla całego słowa, jest on liniowy.

Testy

Każdy test składał się z 20 przypadków testowych. W poniższej tabeli n to maksymalna długość słowa w teście.

Nazwa	n	Opis
okr1.in	14	mały test poprawnościowy, 10 pkt
okr2.in	17	mały test poprawnościowy, 10 pkt
okr3.in	20	mały test poprawnościowy, 10 pkt
okr4.in	200	średni test, 20 pkt
okr5.in	2 000	średni test, 20 pkt
okr6.in	200 000	duży test, 30 pkt