Igor Adamski
Treść zadania, Opracowanie

Lech Duraj
Program

Dostępna pamięć: 128 MB.

OI, etap II, dzień drugi, 9.02.2012

Okropny wiersz

Bajtek musi nauczyć się na pamięć fragmentu pewnego wiersza. Wiersz, zgodnie z najlepszymi regułami sztuki współczesnej, jest długim napisem składającym się wyłącznie z małych liter alfabetu angielskiego. Brzmi oczywiście okropnie, ale nie to jest największym problemem Bajtka: przede wszystkim zapomniał on, który właściwie fragment był zadany. Wszystkie fragmenty wydają się zresztą zbyt trudne do nauczenia...

Jest jednak cień nadziei: niektóre partie wiersza wykazują pewne prawidłowości. W szczególności, czasem fragment A jest wielokrotnym powtórzeniem pewnego innego fragmentu B (innymi słowy, $A = BB\ldots B$, tzn. $A = B^k$, gdzie $k \geqslant 1$ jest liczbą całkowitą). Powiemy wtedy, że B jest **pełnym okresem** *A (w szczególności, każdy napis jest swoim pełnym okresem). Jeśli zadany fragment ma jakiś krótki pełny okres, Bajtka czeka znacznie mniej roboty. Tylko... właściwie który to był fragment?*

Zrób Bajtkowi prezent — napisz program, który wczyta pełen tekst wiersza oraz listę fragmentów, o których Bajtek podejrzewa, że mogły być tym zadanym do nauczenia, i dla każdego z nich obliczy, jaki jest jego najkrótszy pełny okres.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedna liczba całkowita n ($1 \leqslant n \leqslant 500\,000$). W drugim wierszu znajduje się napis długości n złożony z małych liter alfabetu angielskiego — jest to tekst wiersza. Pozycje kolejnych liter numerujemy od 1 do n.

W następnym wierszu znajduje się jedna liczba całkowita q ($1 \leqslant q \leqslant 2\,000\,000$) określająca liczbę fragmentów. W kolejnych q wierszach zapisane są zapytania, po jednym w wierszu. Każdy z tych wierszy zawiera dwie liczby całkowite a_i, b_i ($1 \leqslant a_i \leqslant b_i \leqslant n$), oddzielone pojedynczym odstępem, reprezentujące zapytanie o długość najkrótszego pełnego okresu fragmentu wiersza zaczynającego się na pozycji a_i i kończącego się na pozycji b_i.

W testach wartych łącznie 42% punktów zachodzi dodatkowy warunek $n \leqslant 10\,000$. W pewnych spośród tych testów, wartych łącznie 30% punktów, zachodzi także warunek $q \leqslant 10\,000$.

Wyjście

Twój program powinien wypisać na standardowe wyjście q wierszy. W wierszu numer i powinna znaleźć się jedna liczba całkowita — odpowiedź na i-te zapytanie.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`8`	`1`
`aaabcabc`	`3`
`3`	`5`
`1 3`	
`3 8`	
`4 8`	

Rozwiązanie

Wprowadzenie

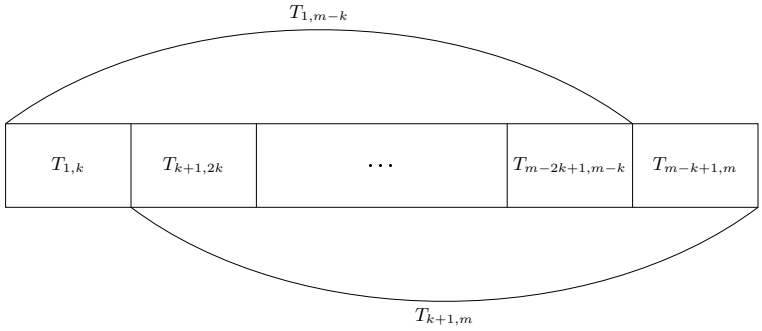
W zadaniu dane jest pewne słowo $S = s_1 s_2 \dots s_n$. Celem jest odpowiadanie na serię zapytań postaci:

$$P(i,j)\text{: } \textit{Jaki jest najkrótszy pełny okres podsłowa } S_{i,j} = s_i s_{i+1} \dots s_j\textit{?}$$

Wstępne obserwacje

Zastanówmy się na początek, w jaki sposób można szybko sprawdzić, czy dane słowo ma pełny okres o zadanej długości. Okazuje się, że zachodzi następujący fakt:

Obserwacja 1. Słowo T o długości m ma pełny okres długości k wtedy i tylko wtedy, gdy $k \mid m$ oraz $T_{1,m-k} = T_{k+1,m}$.



Rys. 1: Ilustracja równości $T_{1,m-k} = T_{k+1,m}$.

Obserwację tę można łatwo samemu uzasadnić. Jej dowód można także znaleźć np. w opracowaniu zadania *Palindromy* z XIII Olimpiady Informatycznej [13]. Na mocy obserwacji, zamiast sprawdzać, czy słowo ma pełny okres danej długości, możemy sprawdzać, czy dwa podsłowa tego słowa są sobie równe. Zajmijmy się najpierw efektywnym rozwiązaniem tego drugiego problemu.

Sprawdzanie równości dwóch podsłów

Haszowanie

Sposób ten polega na wybraniu małej liczby całkowitej p oraz dużej liczby całkowitej M i obliczeniu dla każdego sufiksu $S_{k,n}$ wartości

$$h[k] = \left(\sum_{i=k}^{n} s_i \cdot p^{i-k} \right) \bmod M.$$

Można to zrobić w czasie liniowym, korzystając z tzw. schematu Hornera:

$$h[n+1] = 0, \quad h[k] = (s_k + p \cdot h[k+1]) \bmod M \quad \text{dla} \quad k = n, n-1, \ldots, 1.$$

Teraz możemy każdemu podsłowu $S_{i,j}$ przyporządkować wartość (tzw. hasz):

$$\left(\sum_{k=i}^{j} s_i \cdot p^{k-i} \right) \bmod M = (h[i] - h[j+1] \cdot p^{j-i+1}) \bmod M.$$

Jeśli dwa podsłowa tej samej długości mają ten sam hasz, to z dużym prawdopodobieństwem podsłowa te są równe, jeśli natomiast hasze są różne, to podsłowa na pewno są różne. Po wstępnych obliczeniach w czasie $O(n)$ (spamiętanie potęg liczby p oraz obliczenie haszy sufiksów) uzyskujemy probabilistyczne rozwiązanie, które odpowiada na pytania o równość podsłów w czasie stałym. W praktyce, przy odpowiednim doborze wartości p (najczęściej mała liczba pierwsza większa od rozmiaru alfabetu) oraz M (duża liczba pierwsza), „złapanie" haszowania na błędnej odpowiedzi jest skrajnie trudne. (Za to w opisie rozwiązania zadania *Prefiksufiks* w tej książeczce można znaleźć kilka informacji o tym, w jaki sposób haszowania *nie* należy używać).

Algorytm Karpa-Millera-Rosenberga (KMR)

W tym algorytmie indeksujemy wszystkie podsłowa, których długości są potęgami dwójki, przypisując im liczby całkowite. Zachowana jest przy tym własność, że słowa są równe wtedy i tylko wtedy, gdy mają przypisane jednakowe wartości. Słowa o dowolnej długości k można porównać, znajdując najpierw takie d, dla którego $2^d \leqslant k < 2^{d+1}$, a następnie porównując identyfikatory prefiksów i sufiksów słów o długości 2^d. Tablicę indeksów buduje się w czasie $O(n \log n)$ albo $O(n \log^2 n)$, w zależności od użytego algorytmu sortowania. Wówczas odpowiedzi na pytanie o równość podsłów można udzielać w czasie stałym. Dokładniejszy opis działania oraz konstrukcji tej struktury danych można znaleźć np. w opracowaniu zadania *Powtórzenia* z VII Olimpiady Informatycznej [7] albo zadania *Korale* z XVII Olimpiady [17].

Odpowiedź na zapytanie

Umiemy już szybko odpowiadać na pytania postaci:

$P(i, j, k)$: *Czy słowo $S_{i,j}$ ma pełny okres długości k?*

Wykorzystamy to do udzielania odpowiedzi na oryginalne zapytania $P(i, j)$. Dla wygody oznaczmy $W = S_{i,j}$ i $m = |W|$, tj. $m = j - i + 1$.

Rozwiązanie wolne $O(\sqrt{m})$

Wiedząc, że długość każdego pełnego okresu musi dzielić długość całego słowa, możemy sprawdzić wszystkie dzielniki m i wybrać najmniejszy, dla którego istnieje pełny okres o tej długości. Wszystkie dzielniki liczby m możemy znaleźć w czasie $O(\sqrt{m})$, podobną metodą jak proste sprawdzanie, czy liczba jest pierwsza. Stąd otrzymujemy złożoność tego rozwiązania. Na zawodach otrzymywało około 65-75 punktów. W plikach `okrs0.cpp` i `okrs5.pas` można znaleźć implementację wykorzystującą haszowanie, a w pliku `okrs1.cpp` wykorzystany został algorytm Karpa-Millera-Rosenberga.

Rozwiązanie trochę szybsze

Powyższe rozwiązanie można usprawnić, generując i zapamiętując wcześniej wszystkie dzielniki dla każdej liczby od 1 do n. Dzięki temu odpowiedź na pojedyncze zapytanie wykonujemy w czasie zależnym dokładnie od liczby dzielników długości słowa, zamiast w pesymistycznym czasie $\Theta(\sqrt{m})$. Jest to znaczące usprawnienie, ponieważ liczba dzielników jest zazwyczaj istotnie mniejsza niż pierwiastek (zainteresowany Czytelnik więcej informacji na ten temat znajdzie np. w opracowaniu zadania *Sejf* z XVIII Olimpiady Informatycznej [18]). Implementacja rozwiązania znajduje się w pliku `okrs9.cpp`. Na zawodach za tego typu rozwiązanie można było otrzymać nawet ponad 90 punktów, gdyż bardzo ciężko ułożyć testy, które odcinałyby je od rozwiązania wzorcowego.

Rozwiązanie wzorcowe $O(\log m)$

Do rozwiązania wzorcowego potrzebujemy jeszcze dwóch prostych obserwacji:

Obserwacja 2. Jeśli słowo długości m nie ma pełnego okresu długości k, to nie ma też żadnego pełnego okresu długości l, gdzie $l \mid k$.

Dowód jest trywialny (np. przez sprzeczność).

Obserwacja 3. Jeśli słowo U jest pełnym okresem słowa T, to najkrótszy pełny okres T jest równy najkrótszemu pełnemu okresowi U.

Dowód tej obserwacji można znaleźć we wspomnianym już omówieniu zadania *Palindromy* z XIII Olimpiady [13].

Niech k będzie czynnikiem pierwszym, który dzieli m w maksymalnej potędze α. Wykorzystajmy wcześniejsze obserwacje:

- jeśli W ma pełny okres długości $\frac{m}{k}$, to z obserwacji 3 możemy obliczyć najkrótszy pełny okres słowa $W_{1,\frac{m}{k}}$; będzie on także najkrótszym pełnym okresem W;
- jeśli natomiast W nie ma pełnego okresu długości $\frac{m}{k}$, to z obserwacji 2 wiemy, że nie ma też żadnego pełnego okresu długości l, gdzie $l \mid \frac{m}{k}$. Zauważmy, że dzielniki m, które nie są dzielnikami $\frac{m}{k}$, muszą być podzielne przez k^α.

Możemy już skonstruować algorytm wzorcowy. Będziemy trzymać dwie wartości: m — długość aktualnie rozpatrywanego słowa, oraz liczbę l — iloczyn czynników, przez które musi być podzielna długość najkrótszego pełnego okresu. Zaczynamy z $m = |S_{i,j}|$ oraz $l = 1$ i wykonujemy kroki zgodnie z powyższymi punktami:

1. Znajdujemy czynnik pierwszy k, który dzieli $\frac{m}{l}$, oraz maksymalną potęgę α, w której dzieli.

2. Jeśli aktualne słowo ma pełny okres długości $\frac{m}{k}$, to zmniejszamy długość słowa do tej wartości.

3. Jeśli nie, to domnażamy do l wartość k^α.

Algorytm zatrzymuje się w momencie, gdy $m = l$. Jest to wówczas długość najkrótszego pełnego okresu wyjściowego słowa.

Podczas działania potrzebujemy często znajdować jakiś dzielnik pierwszy danej liczby. Możemy to zrealizować w czasie stałym, jeśli na samym początku użyjemy sita Eratostenesa do stablicowania po jednym dzielniku pierwszym każdej liczby od 2 do n, w czasie $O(n \log\log n)$. Opis tej metody można znaleźć np. w opracowaniu zadania *Zapytania* z XIV Olimpiady Informatycznej [14].

Złożoność czasowa rozwiązania wynika z faktu, że w każdym kroku albo wartość m jest dzielona przez co najmniej 2, albo wartość l jest przemnażana przez co najmniej 2, więc liczba kroków wykonanych do momentu, w którym wartości te będą równe, nie przekroczy $\log m$.

W plikach `okr.cpp` oraz `okr2.pas` znajduje się implementacja tego rozwiązania używająca haszowania, działająca w całkowitym czasie $O(n \log\log n + q \log n)$. W pliku `okr1.cpp` znajduje się rozwiązanie korzystające z algorytmu Karpa-Millera-Rosenberga, działające w czasie $O(n \log^2 n + q \log n)$. Oba rozwiązania uzyskiwały na zawodach komplet punktów.

Inne rozwiązania wolne

W plikach `okrs4.cpp` i `okrs8.pas` zaimplementowano rozwiązanie, które nie używa żadnej struktury danych do porównywania podsłów. Sprawdza po prostu wszystkie dzielniki długości słowa i dla każdego iteruje się po całym słowie do pierwszej niezgodności. Pesymistyczny czas działania wynosi $O(qn\sqrt{n})$. Na zawodach takie rozwiązanie otrzymywało 20-30 punktów.

Istnieją także rozwiązania zadania korzystające z tablicy prefikso-sufiksów[1]. Oba rozwiązania opierają się na pewnym wariancie obserwacji 1: długość najkrótszego pełnego okresu jest równa długości słowa minus długość najdłuższego jego prefikso-sufiksu, jeśli ta liczba dzieli długość słowa, a w przeciwnym razie najkrótszym pełnym okresem jest całe słowo. Wyznaczenie długości najdłuższego prefikso-sufiksu zajmuje czas liniowy względem długości słowa.

Pierwsze rozwiązanie, zaimplementowane w plikach `okrs3.cpp` i `okrs7.pas`, oblicza dla każdego zapytania tablicę prefikso-sufiksową dla podsłowa $S_{i,j}$ i następnie odpowiada zgodnie z powyższą obserwacją. Czas działania wynosi $O(qn)$. Takie rozwiązanie otrzymywało 30-40 punktów.

Drugie rozwiązanie wczytywało wszystkie zapytania i grupowało je po początkach podsłów. Następnie dla każdego sufiksu całego słowa obliczało tablicę prefikso-sufiksów i zapamiętywało wynik dla wszystkich zapytań, których podsłowa zaczynały

[1] O tej tablicy można przeczytać (w związku z algorytmem KMP) np. w książkach [21, 23].

się na tej samej pozycji, co ten sufiks. Dzięki temu grupowaniu złożoność rozwiązania wynosi $O(n^2)$. Takie rozwiązania otrzymywały 40-50 punktów. Implementację można znaleźć w plikach `okrs2.cpp` i `okrs6.pas`.

Testy

Przy konstrukcji testów używano następujących, charakterystycznych fragmentów:

- powtórzony i zaburzony napis losowy: krótki ciąg przypadkowych znaków, powtórzony wielokrotnie (niekoniecznie całkowitą liczbę razy), zmieniony w przypadkowym miejscu (w szczególności, powtarzany ciąg znaków stosunkowo często był jednoelementowy);
- losowa wieża: test o strukturze rekurencyjnej; w zależności od wyniku losowania wieża jest sklejeniem dwóch mniejszych wież lub powtórzoną mniejszą wieżą;
- wysoka wieża: jw., tylko wieża jest zawsze powtórzoną mniejszą wieżą, liczba powtórzeń jest między 2 a 3, aby zapewnić dużą głębokość zagnieżdżenia.
- łańcuch: słowo postaci $(\mathtt{a}^{s_1}\mathtt{b}^{t_1})^{k_1}(\mathtt{b}^{s_2}\mathtt{c}^{t_2})^{k_2}(\mathtt{c}^{s_3}\mathtt{d}^{t_3})^{k_3}\ldots$

Słowa występujące w poszczególnych testach były zazwyczaj zbudowane z kilku sklejonych fragmentów różnych typów. W znacznej części zapytań długość słowa była liczbą o dużej liczbie dzielników, co pozwalało spowolnić rozwiązania sprawdzające wszystkie dzielniki. Wszystkie testy zestawione są w poniższej tabeli.

Nazwa	n	q
okr1a.in	72	8
okr1b.in	1	1
okr1c.in	12	4
okr2.in	186	245
okr3.in	2 714	3 042
okr4a.in	8 200	9 999
okr4b.in	10 000	9 901
okr5a.in	9 900	9 992
okr5b.in	10 000	9 889
okr6a.in	9 001	90 992
okr6b.in	10 000	100 000
okr7a.in	7 691	98 227
okr7b.in	10 000	100 000

Nazwa	n	q
okr8.in	40 000	80 006
okr9.in	63 480	99 119
okr10.in	81 490	84 962
okr11.in	90 006	91 037
okr12.in	240 176	500 300
okr13.in	299 400	590 856
okr14.in	370 488	983 865
okr15a.in	500 000	968 228
okr15b.in	500 000	2 000 000
okr16a.in	466 530	975 537
okr16b.in	500 000	2 000 000