Adam Malinowski
autor zadania, opracowanie

Tomasz Waleń
program wzorcowy

Powtórzenia

Dany jest ciąg słów nad alfabetem $\{'a', \ldots, 'z'\}$*. Należy znaleźć długość najdłuższego słowa występującego jako spójny fragment w każdym z danych słów.*

Zadanie

Napisz program, który:

- *wczyta ciąg słów z pliku tekstowego* `POW.IN`*,*
- *obliczy długość najdłuższego słowa występującego jako spójny fragment w każdym z podanych słów,*
- *zapisze wynik w pliku tekstowym* `POW.OUT`*.*

Wejście

W pierwszym wierszu pliku tekstowego `POW.IN` *zapisano liczbę* n*, gdzie* $1 \leqslant n \leqslant 5$*, oznaczającą liczbę słów. W każdym z* n *kolejnych wierszy znajduje się jedno słowo utworzone z małych liter alfabetu angielskiego* $'a', \ldots, 'z'$*. Każde ze słów ma długość przynajmniej 1, ale nie większą niż 2 000.*

Wyjście

Plik tekstowy `POW.OUT` *powinien zawierać dokładnie jeden wiersz zawierający pojedynczą liczbę całkowitą równą długości najdłuższego słowa występującego jako spójny fragment w każdym z danych słów.*

Przykład

Dla pliku wejściowego `POW.IN`*:*

```
3
abcb
bca
acbc
```

poprawną odpowiedzią jest plik tekstowy `POW.OUT`*:*

```
2
```

ROZWIĄZANIE

Do rozwiązania zadania posłuży nam adaptacja struktury danych zwanej *słownikiem podsłów bazowych*. Słownik podsłów bazowych to z grubsza rzecz biorąc tablica

nazw wszystkich podsłów danego słowa o długościach będących potęgami dwójki, przy czym takie same podsłowa mają te same nazwy, a różne podsłowa o tej samej długości — różne nazwy (nazwa to po prostu liczba całkowita z zakresu od 1 do długości słowa; $\texttt{num}[i,k]$ to nazwa podsłowa o długości 2^k zaczynającego się na pozycji i w danym słowie). Taka informacja umożliwia nam sprawdzenie w czasie stałym, czy dwa podsłowa tej samej długości są jednakowe, nawet jeśli ich długości nie są potęgami dwójki: nazwą identyfikującą jednoznacznie podsłowo długości r ($2^k < r < 2^{k+1}$) zaczynające się na pozycji i jest para $\langle \texttt{num}[i,k], \texttt{num}[i+r-2^k+1,k] \rangle$. Rozmiar tej struktury danych, to oczywiście $\Theta(d)$, gdzie d jest długością słowa, i można ją skonstruować w takim właśnie czasie *metodą wstępującą* (ang. bottom-up):

- Najpierw, na podstawie kodów poszczególnych znaków, nadajemy nazwy słowom jednoliterowym.
- Jeśli nadaliśmy już nazwy wszystkim podsłowom długości 2^k (czyli wypełniliśmy wiersz $\texttt{num}[k,*]$ w naszej tablicy), to teraz podsłowu długości 2^{k+1} zaczynającemu się na pozycji i, gdzie $1 \leqslant i \leqslant d - 2^{k+1} + 1$, nadajemy tymczasową nazwę $\langle \texttt{num}[i,k], \texttt{num}[i+2^k,k] \rangle$. Jest to dobra, unikalna nazwa — tyle, że nie jest liczbą z zakresu od 1 do d, lecz parą takich liczb.
- Zamieniamy teraz nazwy będące parami na nazwy liczbowe. W tym celu sortujemy wszystkie pary leksykograficznie (kubełkowo, zaczynając od mniej znaczącej współrzędnej, zob. [7], [13]) i wykonujemy przenumerowanie (takie same wektory będą po posortowaniu leżały koło siebie). Nazwy po przenumerowaniu umieszczamy w kolejnych kolumnach tablicy `num` (oczywiście wraz z każdym wektorem musimy przechowywać numer pozycji początku odpowiadającego mu podsłowa).

Czas wypełniania każdego wiersza jest liniowy, a liczba wierszy do wypełnienia to $\lceil \log d \rceil$. Szczegóły konstrukcji można znaleźć w książce [10].

W rozwiązaniu wzorcowym obliczamy *wspólny* słownik dla wszystkich zadanych słów, sklejonych w jedno długie słowo z separatorami (specjalnymi znakami występującymi tylko jednokrotnie) na granicach słów wejściowych. Separatory wprowadzamy po to, żeby uniknąć odrębnego analizowania podsłów „przechodzących przez granice" słów wejściowych. Żeby zaoszczędzić pamięć, przechowujemy tylko jeden, ostatnio obliczony wiersz tablicy `num`. Schemat postępowania wygląda następująco:

- Obliczamy nazwy podsłów o długościach będących kolejnymi potęgami dwójki. Robimy to dopóty, dopóki istnieje podsłowo długości 2^k występujące w każdym ze słów wejściowych.
- Kiedy już wiemy, że najdłuższe podsłowo wspólne dla wszystkich słów wejściowych ma długość mieszczącą się w zakresie $2^k, \ldots, 2^{k+1} - 1$, to jej dokładną wartość wyznaczamy metodą bisekcji w k krokach. (Zachodzi tu potrzeba wyznaczenia nazw dla podsłów o długości p na podstawie nazw dla podsłów o długości r, gdzie $r < p < 2r$. Robimy to, posługując się opisaną na początku sztuczką, czyli jako nazwę tymczasową biorąc parę nazw zachodzących na siebie podsłów długości r pokrywających podsłowo długości p.)

Inne rozwiązania

Przedstawione tu rozwiązanie nie jest optymalne. Postawiony w zadaniu problem można rozwiązać w czasie *liniowym* względem sumy długości słów wejściowych. Wymaga to jednak użycia dość wyrafinowanych struktur danych (np. *drzew sufiksowych*), raczej trudnych do zaimplementowania podczas pięciogodzinnej sesji. W dodatku stopień komplikacji tych struktur mógłby spowodować na tyle duże stałe narzuty czasowe, że przy określonych w zadaniu ograniczeniach na rozmiar danych wejściowych, rozwiązanie asymptotycznie optymalne byłoby przy testowaniu trudne do odróżnienia lub wręcz gorsze, od znacznie prostszego rozwiązania opisanego powyżej.

TESTY

Do sprawdzenia rozwiązań zawodników użyto 14 testów, niekiedy łączonych w grupy.

- POW0.IN — test z treści zadania,
- POW1.IN — prosty test poprawnościowy,
- POW2.IN — test poprawnościowy, słowa składające się z róznych liter,
- POW3.IN — test poprawnościowy, trzy identyczne słowa,
- POW4.IN — prosty test poprawnościowy, 4 krótkie słowa losowe,
- POW5.IN — test poprawnościowy, trzy kolejne słowa Fibonacciego,
- POW6.IN — test poprawnościowy, 1 krótkie i 4 długie słowa,
- POW7.IN — test wydajnościowy, 3 słowa o długości 100,
- POW8.IN — test wydajnościowy, 5 słów o długości 400,
- POW9.IN — test wydajnościowy, 3 słowa o długości 400,
- POW10.IN — test wydajnościowy, 4 słowa o długości 1000,
- POW11.IN — test wydajnościowy, 2 słowa o długości 1200,
- POW12.IN — test wydajnościowy, 5 słów o długości 2000,
- POW13.IN — test wydajnościowy, 5 słów o długości 2000.