Tomasz Waleń
Treść zadania, Opracowanie

Wojciech Tyczyński
Program

Dostępna pamięć: 64 MB. *OI, Etap I, 19.10–16.11.2009*

Korale

Bajtazar postanowił zająć się produkcją naszyjników. Udało mu się okazyjnie kupić bardzo długi sznur różnokolorowych korali. Dysponuje też maszyną, która dla zadanego k ($k > 0$) potrafi pociąć sznur na odcinki składające się z k korali (czyli pierwszy odcinek składa się z korali o numerach $1, \ldots, k$, drugi $k+1, \ldots, 2k$, itd.). Jeśli sznur korali ma długość, która nie jest wielokrotnością k, to ostatni odcinek, który ma długość mniejszą niż k, jako niepełnowartościowy, nie jest wykorzystywany. Kolory korali oznaczamy dalej dodatnimi liczbami całkowitymi.

Bajtazar lubi różnorodność i zastanawia się, jak dobrać liczbę k, tak by otrzymać jak najwięcej różnych sznurów korali. Sznur korali, który ma zostać pocięty, ma wyraźnie określony koniec, od którego zaczynamy odcinać krótsze sznury. Jednak każdy odcięty sznur może zostać obrócony — inaczej mówiąc, sznury $(1, 2, 3)$ i $(3, 2, 1)$ są dla Bajtazara takie same. Napisz program, który pomoże mu wyznaczyć optymalną wartość parametru k.

Przykładowo, dla sznura korali postaci:

$$(1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 3, 1, 2, 2, 1, 3, 3, 2, 1),$$

- *stosując $k = 1$, możemy otrzymać 3 różne sznury korali: (1), (2), (3),*
- *stosując $k = 2$, możemy otrzymać 6 różnych sznurów korali: $(1, 1)$, $(1, 2)$, $(2, 2)$, $(3, 3)$, $(3, 1)$, $(2, 3)$,*
- *stosując $k = 3$, możemy otrzymać 5 różnych sznurów korali: $(1, 1, 1)$, $(2, 2, 2)$, $(3, 3, 3)$, $(1, 2, 3)$, $(3, 1, 2)$,*
- *stosując $k = 4$, możemy otrzymać 5 różnych sznurów korali: $(1, 1, 1, 2)$, $(2, 2, 3, 3)$, $(3, 1, 2, 3)$, $(3, 1, 2, 2)$, $(1, 3, 3, 2)$,*
- *dla większych wartości k możemy uzyskać co najwyżej 3 różne sznury korali.*

Wejście

W pierwszym wierszu standardowego wejścia znajduje się liczba całkowita n ($1 \leqslant n \leqslant 200\,000$), oznaczająca długość sznura korali. W drugim wierszu znajduje się n dodatnich liczb całkowitych a_i ($1 \leqslant a_i \leqslant n$), pooddzielanych pojedynczymi odstępami i oznaczających kolory kolejnych korali w sznurze Bajtazara.

Wyjście

W pierwszym wierszu standardowego wyjścia należy wypisać dwie liczby całkowite oddzielone pojedynczym odstępem: liczbę różnych sznurów korali, które można uzyskać przy optymalnym wyborze parametru k, oraz liczbę l różnych wyborów parametru k prowadzących do uzyskania takiej liczby sznurów. Drugi wiersz powinien zawierać l liczb pooddzielanych pojedynczymi odstępami: wartości parametru k, dla których uzyskujemy optymalne rozwiązanie — mogą one zostać wypisane w dowolnej kolejności.

Przykład

Dla danych wejściowych:
```
21
1 1 1 2 2 2 3 3 3 1 2 3 3 1 2 2 1 3 3 2 1
```
poprawnym wynikiem jest:
```
6 1
2
```

Rozwiązanie

Pierwsze rozwiązanie

Pierwszym pomysłem, jaki nasuwa się w związku z zadaniem, jest bezpośrednie wykonanie poleceń podanych w treści zadania.

Musimy sprawdzić wszystkie możliwe długości podsłów k ($k = 1, 2, \dots, n$). Przez *podsłowa* rozumiemy tu spójne fragmenty wyjściowego ciągu korali, który to ciąg będziemy w związku z tym nazywać *słowem*. Dla danej długości k przeglądamy kolejne podsłowa tej długości, wybrane zgodnie z warunkami zadania. W trakcie przeglądania musimy sprawdzać, czy aktualnie analizowane podsłowo występowało (w identycznej formie, lub też odwrócone) wcześniej w podziale.

Rozwiązanie to, mimo iż wygląda na niezbyt optymalne, w rzeczywistości nie jest najgorsze. Sprawdzenie, ile jest różnych podsłów zadanej długości k wybranych zgodnie z treścią zadania, ma bowiem łączny koszt czasowy rzędu:

$$T_k = \sum_{i=1}^{\lfloor n/k \rfloor} (i-1) \cdot k = k \cdot \sum_{i=1}^{\lfloor n/k \rfloor} (i-1) = k \cdot O\left(\frac{n^2}{k^2}\right) = O\left(\frac{n^2}{k}\right).$$

Powyższe oszacowanie bierze się stąd, że dla i-tego z kolei podsłowa musimy sprawdzić, czy wystąpiło wśród dotychczas rozpatrzonych $i-1$ podsłów, a dla każdej pary podsłów sprawdzenie, czy są one równe zgodnie z naszymi kryteriami, zajmuje czas $O(k)$. Zatem złożoność czasowa całego algorytmu wynosi:

$$T = \sum_{k=1}^{n} T_k = O\left(\sum_{k=1}^{n} \frac{n^2}{k}\right) = O(n^2) \cdot O\left(\sum_{k=1}^{n} \frac{1}{k}\right) = O(n^2) \cdot O(\log n) = O(n^2 \log n).$$

W tym oszacowaniu wykorzystujemy znaną własność tzw. liczb harmonicznych, patrz np. książka [23]:

$$\sum_{k=1}^{n} \frac{1}{k} = O(\log n). \tag{1}$$

Implementację przedstawionego rozwiązania można znaleźć w pliku `kors1.cpp`. Na zawodach tego typu programy zdobywały około 40 punktów.

Rozwiązanie wzorcowe

Zastanówmy się teraz, w jaki sposób można ulepszyć poprzednie rozwiązanie. Najbardziej pracochłonną częścią tego algorytmu było porównywanie podsłów długości k. Aby usprawnić ten fragment rozwiązania, możemy zastosować algorytm Karpa-Millera-Rosenberga (KMR), który buduje tzw. *słownik podsłów bazowych*[1]. W ten sposób dla zadanego słowa s, w czasie $O(n \log n)$ (lub $O(n \log^2 n)$, w zależności od tego, czy w algorytmie używamy sortowania pozycyjnego, czy jakiegoś o złożoności czasowej liniowo-logarytmicznej) przygotowujemy strukturę danych, która pozwala na porównywanie dowolnych dwóch podsłów słowa s w czasie $O(1)$.

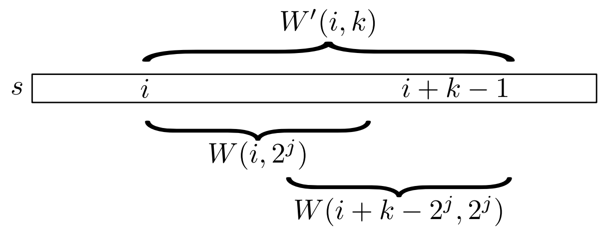
Struktura ta, dla wejściowego słowa s ($s = s[1..n]$) oraz dowolnych liczb całkowitych $0 \leqslant j \leqslant \log n$ oraz $1 \leqslant i \leqslant n + 1 - 2^j$, generuje liczbowy identyfikator $W(i, 2^j)$ dla podsłowa $s[i..i + 2^j - 1]$. Identyfikatory te są niewielkimi liczbami całkowitymi (z zakresu $[0..n - 1]$) oraz mają tę własność, że

$$W(i, 2^j) = W(p, 2^j) \quad \Leftrightarrow \quad s[i..i + 2^j - 1] = s[p..p + 2^j - 1].$$

Dzięki takiej informacji możemy również jednoznacznie identyfikować podsłowa *dowolnej* długości k. Faktycznie, niech j oznacza największą liczbę całkowitą, taką że $2^j \leqslant k$. Wtedy podsłowo $s[i..i + k - 1]$ możemy jednoznacznie zidentyfikować z parą

$$W'(i, k) \stackrel{\text{def}}{=} (W(i, 2^j), W(i + k - 2^j, 2^j)),$$

zobacz rys. 1. Oczywiście, nadal ma miejsce własność, że dwa podsłowa długości k są identyczne wtedy i tylko wtedy, gdy odpowiadające im identyfikatory są równe.



Rys. 1: Funkcja $W'(i, k)$.

Zadanie wymaga od nas utożsamiania słów postaci w oraz w^R (odwrócone słowo w). Można pokonać to utrudnienie, stosując algorytm KMR dla słowa $s\#s^R$ (# to dowolny symbol różny od wszystkich symboli wejściowych). Musimy również zmienić definicję funkcji $W'(i, k)$. Każde podsłowo $w = s[i..i + k - 1]$ występuje w słowie $s\#s^R$ w dwóch miejscach:

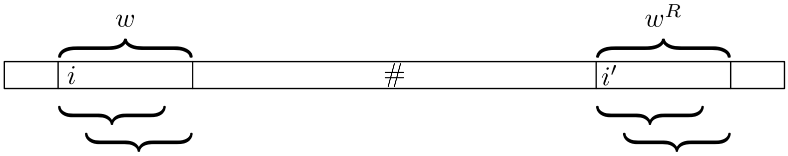
- na pozycji i — jako w,
- oraz na pozycji $i' = 2n + 2 - (i + k - 1)$ — jako w^R.

Jako identyfikator $W'(i, k)$, dla $1 \leqslant i$ oraz $i + k - 1 \leqslant n$, powinniśmy wybrać np. mniejszą (leksykograficznie) z par:

$$(W(i, 2^j), W(i + k - 2^j, 2^j)) \quad \text{oraz} \quad (W(i', 2^j), W(i' + k - 2^j, 2^j)),$$

[1] Patrz np. książka [19] lub opis rozwiązania zadania *Powtórzenia* z VII OI [7].

gdzie $2^j \leqslant k < 2^{j+1}$, patrz także rys. 2.



Rys. 2: Konstrukcja funkcji $W'(i,k)$ dla słowa $s\#s^R$.

Kolejnym krokiem, który musimy zastosować, jest efektywniejsze badanie liczby różnych podsłów w zadanym podziale. Dla ustalonego k, za pomocą poprzedniego kroku utożsamiamy podsłowa z parami liczb całkowitych, czyli otrzymujemy ciąg złożony z $\lfloor \frac{n}{k} \rfloor$ par. Następnie taki ciąg możemy uporządkować leksykograficznie (tzn. posortować pary w pierwszej kolejności względem pierwszej współrzędnej, a w razie remisów względem drugiej). W uporządkowanym ciągu już łatwo wyznaczamy liczbę różnych elementów. Takie postępowanie możemy zaimplementować łatwo w czasie liniowo-logarytmicznym względem liczby elementów ciągu, równej $\lfloor \frac{n}{k} \rfloor$ dla danej długości podsłowa k. Można się także bardziej postarać i zamiast sortowania liniowo-logarytmicznego użyć algorytmu sortowania pozycyjnego. Wówczas należy *wszystkie* ciągi par, skonstruowane dla poszczególnych wartości parametru k, posortować za jednym razem, dodając do każdej z par trzecią współrzędną oznaczającą wartość parametru k, której ona odpowiada. Dzięki temu koszt czasowy sortowania spada do:

$$O\left(\sum_{k=1}^{n} \frac{n}{k}\right),$$

czyli możemy powiedzieć, że na daną wartość parametru k przypada $O(\frac{n}{k})$ operacji.

Podsumowując, zadanie możemy rozwiązać, używając następującego postępowania:

uruchom algorytm KMR dla słowa $s\#s^R$
for $k := 1$ **to** n **do begin**
  wygeneruj ciąg par identyfikujących kolejne podsłowa o długości k:
  $C_k = (W'(i,k) \ : \ i = 1,\ k+1,\ 2k+1, \ldots)$
end
uporządkuj ciągi C_k przy pomocy sortowania pozycyjnego
for $k := 1$ **to** n **do**
  wyznacz liczbę różnych elementów w ciągu C_k

Zastanówmy się, jaka jest złożoność takiego rozwiązania. Czas potrzebny na wstępne obliczenia w algorytmie KMR wynosi $O(n \log n)$. Następnie, wyznaczenie liczby różnych podsłów długości k wybranych zgodnie z wymaganiami zadania zajmuje czas $O(\frac{n}{k})$. Korzystając ponownie z oszacowania (1), otrzymujemy koszt czasowy całego algorytmu:

$$T = O(n \log n) + \sum_{k=1}^{n} O\left(\frac{n}{k}\right) = O(n \log n) + O(n) \cdot O\left(\sum_{k=1}^{n} \frac{1}{k}\right) = O(n \log n).$$

Implementacje rozwiązania wzorcowego używające sortowania liniowo-logarytmicznego, czyli działające w złożoności czasowej $O(n\log^2 n)$, można znaleźć w plikach `kor.cpp` i `kor1.pas`.

W tym miejscu warto dodać, że równie efektywne rozwiązanie można otrzymać, korzystając z metody haszowania, która — podobnie jak algorytm KMR — pozwala na wyznaczanie identyfikatorów zadanej długości podsłów słowa s, jednakże jest obarczona mało prawdopodobną możliwością błędnego stwierdzenia, że dwa różne podsłowa są równe. Więcej o tej metodzie można przeczytać w opisie rozwiązania zadania *Antysymetria* w niniejszej książeczce oraz w podanych tam odnośnikach.

Testy

Rozwiązania były sprawdzane na 12 grupach danych testowych. Poniżej przedstawiona została tabela z podstawowymi parametrami testów, w której n oznacza długość sznura korali, a m — liczbę różnych kolorów korali.

Nazwa	n	m
kor1a.in	20	13
kor1b.in	20	2
kor1c.in	20	1
kor2a.in	100	17
kor2b.in	100	8
kor2c.in	100	1
kor3a.in	200	60
kor3b.in	200	4
kor4a.in	500	252
kor4b.in	500	9
kor5a.in	16 000	726
kor5b.in	16 000	16
kor6a.in	20 000	775
kor6b.in	20 000	48

Nazwa	n	m
kor7a.in	40 000	2 035
kor7b.in	40 000	96
kor8a.in	50 000	12 303
kor8b.in	50 000	120
kor9a.in	80 000	34 267
kor9b.in	80 000	18
kor10a.in	140 000	88 456
kor10b.in	140 000	3
kor11a.in	170 000	101 623
kor11b.in	170 000	13
kor12a.in	200 000	126 290
kor12b.in	200 000	2
kor12c.in	200 000	1