Tomasz Idziaszek
Wojciech Śmietanka — Treść zadania
Wojciech Śmietanka — Opracowanie
Zbigniew Wojna — Program

Dostępna pamięć: 128 MB. *OI, etap I, 17.10–14.11.2011*

Odległość

W tym zadaniu rozważamy **odległość** *między dodatnimi liczbami całkowitymi, zdefiniowaną w następujący sposób. Przez pojedynczą* **operację** *rozumiemy pomnożenie danej liczby przez liczbę pierwszą*[1] *lub podzielenie jej przez liczbę pierwszą (o ile dzieli się ona bez reszty). Odległość między liczbami a i b, oznaczana $d(a,b)$, to minimalna liczba operacji potrzebnych do przekształcenia liczby a w liczbę b. Na przykład, $d(69,42) = 3$.*

Zauważmy, że faktycznie funkcja d ma cechy odległości — dla dowolnych dodatnich liczb całkowitych a, b i c zachodzi:

- *odległość liczby od niej samej wynosi 0: $d(a,a) = 0$,*
- *odległość od a do b jest taka sama, jak od b do a: $d(a,b) = d(b,a)$, oraz*
- *spełniona jest nierówność trójkąta: $d(a,b) + d(b,c) \geqslant d(a,c)$.*

Dany jest ciąg n dodatnich liczb całkowitych $a_1, a_2, \ldots, a_n$. Dla każdej liczby a_i należy wskazać takie j, że $j \neq i$ oraz $d(a_i,a_j)$ jest minimalne.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się liczba całkowita n ($2 \leqslant n \leqslant 100\,000$). W kolejnych wierszach znajdują się liczby całkowite $a_1, a_2, \ldots, a_n$ ($1 \leqslant a_i \leqslant 1\,000\,000$), po jednej w wierszu.

W testach wartych łącznie 30% punktów zachodzi dodatkowy warunek $n \leqslant 1\,000$.

Wyjście

Twój program powinien wypisać na standardowe wyjście dokładnie n wierszy, a w każdym z nich po jednej liczbie całkowitej. W i-tym wierszu należy wypisać **najmniejsze** *takie j, że: $1 \leqslant j \leqslant n$, $j \neq i$ oraz $d(a_i,a_j)$ jest minimalne.*

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
6	2
1	1
2	1
3	2
4	1
5	2
6	

[1] Przypomnijmy, że liczba pierwsza to taka liczba całkowita dodatnia, która ma dokładnie dwa różne dzielniki: jedynkę i siebie samą.

Rozwiązanie

Analiza problemu

Zacznijmy od przejścia na język teorii grafów. Rozważmy nieskończony graf nieskierowany, w którym wierzchołkami są wszystkie liczby naturalne, a krawędzie odpowiadają pomnożeniu (równoważnie, podzieleniu) danej liczby przez liczbę pierwszą. Problem z zadania formułuje się teraz następująco: dla każdego z n zaznaczonych wierzchołków chcemy znaleźć w grafie najbliższy inny zaznaczony wierzchołek.

Wygodnie będzie, jeśli na wstępie wyeliminujemy z podanego na wejściu ciągu liczb wszystkie powtórzenia. Faktycznie, dla każdej z powtarzających się liczb, jako wynik możemy od razu wskazać dowolne inne wystąpienie takiej samej liczby w ciągu. Odtąd będziemy zakładać, że nasz ciąg nie zawiera powtórzeń (wystarczy pozostawić po jednym egzemplarzu każdej wartości z ciągu).

Jak duży jest nasz graf?

Na początek musimy poradzić sobie z faktem, że nasz graf jest nieskończony. Pokażemy, że w istocie interesować nas będzie jedynie skończony fragment tego grafu.

Rozważmy najkrótszy ciąg mnożeń i dzieleń przez liczby pierwsze, który przekształca liczbę a w liczbę b. Zauważmy, że jeśli w tym ciągu wykonalibyśmy zarówno operację pomnożenia przez p, jak i operację podzielenia przez p, to, usuwając te operacje, uzyskalibyśmy krótszy ciąg, który również przekształca a w b, co byłoby sprzeczne z założeniem, że nasz ciąg jest najkrótszy. Możemy zatem założyć, że dla każdej liczby pierwszej p wykonujemy albo operacje mnożenia przez p, albo dzielenia przez p. W tym drugim przypadku widzimy, że liczba a musi być podzielna przez odpowiednią potęgę p. Widać więc, że kolejność wykonywania operacji nie ma znaczenia, zatem wszystkie operacje dzielenia mogą zostać wykonane przed mnożeniami.

To prowadzi nas do wniosku, że pewna najkrótsza ścieżka pomiędzy a i b w grafie nie zawiera wierzchołków większych niż $\max(a, b)$. Jeśli zatem oznaczymy przez M największą liczbę z wejścia, to wystarczą nam wierzchołki dla liczb od 1 do M.

Oszacujemy teraz liczbę krawędzi w naszym grafie. Wydaje się, że może ich być dużo. Najprostsze oszacowanie daje $M \cdot \pi(M)$ krawędzi, gdzie $\pi(M)$ jest liczbą liczb pierwszych mniejszych od M. Zauważmy jednak, że jeśli dla pewnego a mamy k krawędzi reprezentujących podzielenie a przez liczbę pierwszą, to a musi mieć co najmniej k różnych dzielników pierwszych. Przy założeniu, że $M \leqslant 10^6$, mamy $k \leqslant 7$, gdyż rozważając iloczyn ośmiu najmniejszych liczb pierwszych, dostajemy

$$2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 = 9\,699\,690 > 10^6.$$

Czyli wszystkich krawędzi jest co najwyżej $7 \cdot 10^6$, gdyż każda z krawędzi w grafie reprezentuje, w szczególności, podzielenie liczby przez jakąś liczbę pierwszą.

Można również zauważyć, że liczba pierwsza p będzie generowała krawędzie dla co najwyżej $\lfloor M/p \rfloor$ wierzchołków, gdyż każda taka krawędź łączy wierzchołek

$a \in \{1, \ldots, \lfloor M/p \rfloor\}$ z wierzchołkiem $a \cdot p$. Wiedząc, że

$$\sum_{p \leqslant M,\, p \in \mathbb{P}} \frac{M}{p} \approx M \ln \ln M,$$

wnioskujemy, że liczba krawędzi grafu jest rzędu $O(M \ln \ln M)$. To dostatecznie mało, aby można było przejrzeć cały graf.

Wyznaczanie najbliższych wierzchołków w grafie

Gdy już wiemy, że możemy pozwolić sobie na przejrzenie całego grafu, to w dalszych rozważaniach szczególna postać naszego grafu będzie nieistotna. Okazuje się bowiem, że istnieje efektywny algorytm, który wyznacza najbliższe wierzchołki w dowolnym grafie nieskierowanym.

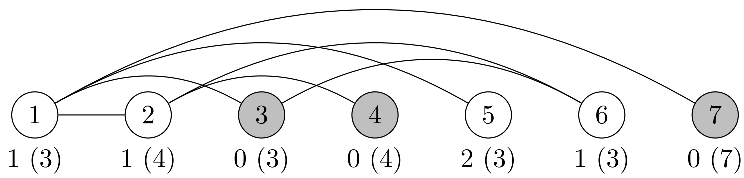
Na początek ustalmy oznaczenia. Mamy dany graf $G = (V, E)$ oraz pewien podzbiór wierzchołków $S \subseteq V$. Dla każdego wierzchołka $v \in S$ chcemy znaleźć inny wierzchołek $w \in S$, taki że odległość między v i w jest jak najmniejsza. Jeśli istnieje więcej niż jeden kandydat na w, wybieramy tego o mniejszym numerze.

Dla ustalonego podzbioru $S \subseteq V$ oraz wierzchołka $v \in V$ oznaczmy przez $d_S[v]$ odległość z v do najbliższego wierzchołka z S (jeśli $v \in S$, to oczywiście $d_S[v] = 0$). Przez $m_S[v]$ oznaczmy taki wierzchołek z S, który realizuje tę odległość. Jeśli istnieje więcej niż jeden kandydat na $m_S[v]$, wybieramy tego o mniejszym numerze.

Najpierw pokażemy, jak obliczyć $d_S[v]$ i $m_S[v]$ dla wszystkich $v \in V$. Można to zrobić, przeszukując graf wszerz (BFS), zaczynając jednocześnie od wszystkich wierzchołków z S. W kolejce Q będziemy trzymać wierzchołki do przetworzenia. Utrzymujemy niezmiennik, że każdy wierzchołek v wrzucany do kolejki ma poprawnie obliczoną wartość $d_S[v]$. Ponadto $m_S[v]$ jest poprawnie obliczone po przetworzeniu wszystkich wierzchołków i, dla których $d_S[i] = d_S[v] - 1$.

```
foreach v ∈ V do d_S[v] := ∞;
foreach v ∈ S do begin
  d_S[v] := 0;
  m_S[v] := v;
  Q.push(v);
end
while not Q.empty do begin
  v := Q.pop();
  foreach (v, w) ∈ E do
    if d_S[w] = ∞ then begin
      d_S[w] := d_S[v] + 1;
      m_S[w] := m_S[v];
      Q.push(w);
    end
    else if d_S[v] + 1 = d_S[w] and m_S[v] < m_S[w] then
      m_S[w] := m_S[v];
end
```

Przykład 1. Poniższy rysunek przedstawia graf zbudowany dla $M = 7$ i zbioru $S = \{3, 4, 7\}$. Liczby pod wierzchołkiem v oznaczają wartości $d_S[v]$ i $m_S[v]$.



Naszym celem jest obliczenie odległości z każdego wierzchołka $v \in S$ do najbliższego wierzchołka ze zbioru $S \setminus \{v\}$. Oznaczmy tę odległość przez $d_{S\setminus\{v\}}[v]$, a wierzchołek, który ją realizuje, przez $m_{S\setminus\{v\}}[v]$.

Ustalmy $v \in S$ i oznaczmy $w = m_{S\setminus\{v\}}[v]$. Niech $v = v_1, v_2, \ldots, v_k = w$ będzie najkrótszą ścieżką z v do w. Niech i będzie najmniejszą liczbą, taką że $m_S[v_{i-1}] = v$ oraz $m_S[v_i] \neq v$ (taka liczba oczywiście istnieje, bo $m_S[v_1] = v$ i $m_S[v_k] \neq v$). Zauważmy, że najkrótsza odległość z v_i do zbioru S jest realizowana przez pewien wierzchołek z $S \setminus \{v\}$, zatem $m_S[v_i] = m_{S\setminus\{v\}}[v_i]$, czyli $m_S[v_i] = w$ wobec wyboru wierzchołka w.

Oznacza to, że w grafie istnieje taka krawędź $(v', w') \in E$, dla której $m_S[v'] = v$, $m_S[w'] = w$ oraz w jest najbliższym do v wierzchołkiem z $S \setminus \{v\}$, odległym od niego o $d_S[v'] + 1 + d_S[w']$. Nie wiemy, która to krawędź, więc sprawdzimy wszystkie:

foreach $v \in S$ **do**
    $d_{S\setminus\{v\}}[v] := \infty;$
foreach $(v', w') \in E$ **do begin**
    $v := m_S[v'];$
    $w := m_S[w'];$
    if $v \neq w$ **then begin**
        $d := d_S[v'] + 1 + d_S[w'];$
        if $d < d_{S\setminus\{v\}}[v]$ **or** $(d = d_{S\setminus\{v\}}[v]$ **and** $w < m_{S\setminus\{v\}}[v])$ **then begin**
            $d_{S\setminus\{v\}}[v] := d;$
            $m_{S\setminus\{v\}}[v] := w;$
        end
    end
end

Przykład 2. W grafie z przykładu 1 dla $v = 4$ mamy $w = 3$ oraz $d_{S\setminus\{v\}}[v] = 3$. Wówczas $v' = 2$ i $w' = 1$ lub $v' = 2$ i $w' = 6$.

Obie fazy algorytmu działają w czasie $O(|V|+|E|)$. Jako ćwiczenie dla Czytelnika proponujemy modyfikację tego algorytmu, aby działał również dla grafów z wagami na krawędziach (należy wtedy użyć algorytmu Dijkstry zamiast BFS).

Złożoność

Złożoność czasowa podanego dwufazowego przeszukiwania grafu jest liniowa względem rozmiaru grafu, czyli rzędu $O(M \ln\ln M)$. Warto zauważyć, że nie trzeba trzymać w pamięci całego grafu G. Wystarczy dla każdej liczby mieć obliczony pewien jej dzielnik pierwszy (do generowania krawędzi odpowiadających dzieleniu) i osobno pamiętać listę wszystkich liczb pierwszych (do generowania krawędzi odpowiadających mnożeniu). Zarówno listę wszystkich liczb pierwszych nieprzekraczających M, jak i przykładowe dzielniki pierwsze poszczególnych liczb od 2 do M można obliczyć w czasie $O(M \ln\ln M)$ za pomocą sita Eratostenesa[1]. Dodajmy dla pełności, że początkowe usuwanie powtórzeń z wyjściowego ciągu możemy wykonać w czasie i pamięci $O(M)$, wykorzystując M-elementową tablicę kubełków (list). Ostatecznie, całe rozwiązanie ma złożoność czasową $O(M \ln\ln M)$, a pamięciową $O(M)$.

Implementacje rozwiązania wzorcowego można znaleźć w plikach `odl.cpp` i `odl1.pas`.

Testy

Przygotowanych zostało 10 zestawów testowych. Jeśli dana grupa składa się z więcej niż jednego testu, to pierwszy test jest testem poprawnościowym, co może np. oznaczać, że odległości do najbliższych liczb są większe aniżeli w losowym teście. W tych testach parametr n jest zazwyczaj stosunkowo niewielki.

Nazwa	n	M	Opis
odl1.in	50	95	test losowy
odl2.in	345	698	potęgi liczb pierwszych
odl3a.in	584	3 499	trochę maksymalnych potęg liczb pierwszych, trochę losowy
odl3b.in	1 000	3 500	losowy test wydajnościowy
odl4a.in	11 884	80 649	liczby o nieparzystej i większej niż 4 sumie wykładników w rozkładzie
odl4b.in	21 234	47 288	losowy test wydajnościowy
odl5a.in	13 459	150 000	kilka liczb, od których jest daleko do najbliższego elementu, reszta losowa
odl5b.in	50 001	150 001	losowy test wydajnościowy
odl6a.in	34 209	399 989	mniejsze liczby: losowe, większe liczby: maksymalne potęgi liczb pierwszych
odl6b.in	73 900	234 564	losowy test wydajnościowy

[1] Przykład takiego wykorzystania sita Eratostenesa można znaleźć w opracowaniu zadania *Zapytania* z XIV Olimpiady Informatycznej [14].

Nazwa	**n**	**M**	**Opis**
odl7a.in	36 118	756 432	dużo liczb, od których odległość do najbliższej to 2, trochę mniej tych, od których odległość to 3
odl7b.in	86 023	803 819	losowy test wydajnościowy
odl8a.in	63 759	999 999	dużo liczb o odległości co najmniej 2, nieliczne o dużo większych odległościach do im najbliższych
odl8b.in	98 837	853 983	losowy test wydajnościowy
odl9a.in	1 482	999 810	losowe odległości między 1 a 5
odl9b.in	99 048	1 000 000	losowy test wydajnościowy
odl10a.in	1 176	1 000 000	potęgi liczb naturalnych o wykładnikach większych niż 1
odl10b.in	100 000	999 996	losowy test wydajnościowy
odl10c.in	100 000	999 999	maksymalne wejście