Michał Pilipczuk
Treść zadania, Opracowanie

Jakub Radoszewski
Program

OI, Etap III, dzień próbny, 01.04.2008

Lampki

Mały Jaś dostał na święta nietypowy prezent. Po odpakowaniu kolorowego kartonu jego oczom ukazał się napis „Nieskończony łańcuch lampek choinkowych". Zainteresowany chłopiec od razu rozłożył nową zabawkę na podłodze.

Łańcuch Jasia ma formę kabla posiadającego początek, ale nieposiadającego końca. Do kabla są podłączone lampki ponumerowane (zgodnie z kolejnością na kablu) kolejnymi liczbami naturalnymi, poczynając od 0. Kabel podłączony jest do panelu sterowania. Na panelu znajduje się pewna liczba przycisków, każdy w innym kolorze i przy każdym znajduje się inna dodatnia liczba naturalna. Liczby umieszczone przy różnych przyciskach są parami względnie pierwsze.

W momencie rozpakowania prezentu żadna lampka się nie świeciła. Jaś, nie myśląc wiele, przycisnął po kolei wszystkie przyciski na panelu sterującym, od pierwszego aż do ostatniego. Ze zdumieniem zaobserwował, że naciśnięcie i-tego przycisku powoduje zapalenie się wszystkich lampek o numerach podzielnych przez liczbę p_i znajdującą się przy tym przycisku. Co więcej, wszystkie te lampki zaczynają świecić kolorem k_i, takim jak kolor przycisku. W szczególności, wszystkie lampki o numerach podzielnych przez p_i, które były już uprzednio zapalone, zmieniają swój kolor na kolor k_i.

Teraz Jaś patrzy urzeczony na nieskończony wielobarwny łańcuch i zastanawia się, jaka część lampek pali się poszczególnymi kolorami. Oznaczmy przez $L_{i,r}$ liczbę lampek palących się na kolor k_i, spośród lampek o numerach $0, 1, \ldots, r$. Formalnie, ułamek C_i lampek, które palą się kolorem k_i, to:

$$C_i = \lim_{r \to \infty} \frac{L_{i,r}}{r}$$

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia opisy przycisków na panelu sterowania,*
- *dla każdego koloru k_i obliczy ułamek C_i, mówiący jaka część lampek pali się kolorem k_i,*
- *wypisze wyniki na standardowe wyjście.*

Wejście

Pierwszy wiersz wejścia zawiera jedną liczbę całkowitą n ($1 \leqslant n \leqslant 1\,000$), oznaczającą liczbę przycisków znajdujących się na panelu sterowania. Kolejne n wierszy zawiera po jednej liczbie całkowitej p_i ($1 \leqslant p_i \leqslant 1\,000\,000\,000$), oznaczającej, że przyciśnięcie i-tego przycisku powoduje zapalenie się na kolor k_i lampek o numerach podzielnych przez p_i. Liczby p_i są podane w kolejności naciskania przycisków przez Jasia. Liczby p_i są parami względnie pierwsze i różne.

Wyjście

Twój program powinien wypisać na wyjście dokładnie n wierszy. W i-tym z nich powinien się znaleźć ułamek C_i, mówiący jaka część lampek pali się kolorem k_i, zapisany w formie nieskracalnego ułamka a/b, gdzie a jest całkowite, b całkowite dodatnie oraz a i b są względnie pierwsze. Jeśli $C_i = 0$, to ułamek ten powinien być wypisany jako $0/1$.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*

`3`
`2`
`3`
`5`
	`4/15`
	`4/15`
	`1/5`

Rozwiązanie

Wprowadzenie

Rozwiązanie zadania składa się z dwóch części. Pierwszą z nich jest analiza problemu od strony teoretycznej, w celu wyprowadzenia wzoru na średnie liczby lampek poszczególnych kolorów, które będą świecić się po naciśnięciu wszystkich przycisków. Potem należy zastanowić się, jak efektywnie obliczać wartości wyznaczonej formuły tak, by zmieścić się w limitach czasowych nawet dla największych danych.

Analiza teoretyczna

Dla uproszczenia opisu przyjmiemy, że lampka zgaszona też ma pewien kolor, a dokładniej, wszystkie zgaszone lampki są tego samego koloru. Ponadto będziemy stosować oznaczenie $a \perp b$, gdy liczby a i b są względnie pierwsze.

Pozbądźmy się nieskończoności

Oznaczmy

$$\Pi = p_1 p_2 \ldots p_n.$$

Zauważmy, że kolory lampek w ciągu powtarzają się z okresem Π. Istotnie, dla każdego $q \in \{0, 1, 2, \ldots, \Pi - 1\}$, lampki q, $q + \Pi$, $q + 2\Pi$ itd. są tego samego koloru, gdyż ich numery dzielą się przez te same spośród czynników $p_1, p_2, \ldots, p_n$, jako że Π dzieli się przez nie wszystkie. Stąd ułamek C_i, oznaczający jak w treści zadania część lampek świecących na końcu i-tym kolorem, jest równy stosunkowi liczby takich lampek o numerach od 0 do $\Pi - 1$ do liczby Π. Intuicyjnie jest to oczywiste — jeśli nieskończenie wiele razy powtarzamy ten sam cykl, to lampki w danym kolorze występują w nieskończonym ciągu średnio z taką samą częstotliwością, z jaką występują w cyklu.

Aby formalnie potwierdzić nasze podejrzenia, oznaczmy przez c_i liczbę lampek w kolorze k_i spośród lampek o numerach od 0 do $\Pi-1$. Spróbujmy oszacować $L_{i,r-1}$. Jeśli $r=a\Pi+b$, gdzie $0\leqslant b<\Pi$, to $ac_i\leqslant L_{i,r-1}\leqslant(a+1)c_i$, gdyż lampek w rozważanym kolorze jest co najmniej tyle, ile w a powtórzeniach cyklu, a nie więcej niż w $a+1$ powtórzeniach. Stąd mamy, że $\frac{a}{r}c_i\leqslant\frac{L_{i,r-1}}{r}\leqslant\frac{a+1}{r}c_i$, czyli

$$\frac{1}{\Pi+\frac{b}{a}}c_i=\frac{a}{a\Pi+b}c_i\leqslant\frac{L_{i,r-1}}{r}\leqslant\frac{a+1}{a\Pi+b}c_i=\frac{1+\frac{1}{a}}{\Pi+\frac{b}{a}}c_i.$$

Wraz z $r\to\infty$ również $a\to\infty$, zaś b pozostaje ograniczone, stąd zarówno górne, jak i dolne ograniczenie ułamka $\frac{L_{i,r-1}}{r}$ zbiega do $\frac{c_i}{\Pi}$. Zatem:

$$\lim_{r\to\infty}\frac{L_{i,r}}{r}=\lim_{r\to\infty}\frac{r+1}{r}\frac{L_{i,r}}{r+1}=\lim_{r\to\infty}\frac{L_{i,r}}{r+1}=\lim_{r\to\infty}\frac{L_{i,r-1}}{r}=\frac{c_i}{\Pi}.$$

Sprowadziliśmy zatem problem do rozważania jedynie Π początkowych lampek. Zobaczmy teraz, które spośród nich zapalają się po naciśnięciu odpowiednich przycisków na panelu. Ze względu na to, że liczy się ostateczny kolor lampki, będziemy rozważać przyciski od ostatniego do pierwszego. Po wciśnięciu n-tego przycisku kolorem k_n zaświeci co p_n-ta lampka, stąd $C_n=\frac{1}{p_n}$. Po wciśnięciu $(n-1)$-szego przycisku zaświeci się co p_{n-1}-sza lampka, ale potem niektóre z nich zmienią kolor na k_n. W ogólności, przy wyznaczaniu ułamka lampek, które ostatecznie będą świeciły kolorem k_i, musimy uwzględnić, że niektóre z nich zmienią potem swój kolor. Nie jest to jednak skomplikowane — naciśnięcie i-tego przycisku zapala bowiem lampki o numerach podzielnych przez p_i, ale tylko te z nich, których numery nie dzielą się przez żadne p_j dla $j>i$, pozostaną w tym kolorze do końca. Dokładne wyliczenie ułamka lampek C_i wymaga zastosowania pewnego prostego aparatu matematycznego.

Odrobina teorii liczb

Wpierw zauważmy, że jeżeli $a\perp b$ oraz $a|c$ i $b|c$, to również $ab|c$. Jest to jasne — skoro a i b nie mają w rozkładzie na czynniki pierwsze żadnych wspólnych czynników, zaś w rozkładzie c występują zarówno czynniki liczby a, jak i b, to wszystkie czynniki iloczynu ab także występują w rozkładzie liczby c. Analogicznie, jeśli $a_1,a_2,\ldots,a_n$ są parami względnie pierwsze oraz $a_i|c$ dla każdego i, to c jest podzielne przez $a_1a_2\ldots a_n$.

Weźmy teraz wszystkie liczby ze zbioru $\{0,1,\ldots,\Pi-1\}$ i dla każdej z nich wyliczmy ciąg reszt z dzielenia jej kolejno przez $p_1,p_2,\ldots,p_n$. Załóżmy, że dla dwóch różnych liczb $a,b\in\{0,1,\ldots,\Pi-1\}$ otrzymamy takie same ciągi reszt. Co to oznacza? Przyjmijmy bez straty ogólności, że $a\leqslant b$, i popatrzmy na liczbę $b-a$. Skoro a i b dają taką samą resztę z dzielenia przez p_i, to p_i dzieli $b-a$. Skoro zachodzi to dla każdego p_i, to Π dzieli $b-a$. Jednak $b-a$ jest liczbą ze zbioru $\{0,1,\ldots,\Pi-1\}$, więc $a=b$.

W ten sposób dochodzimy do wniosku, że każda z liczb od 0 do $\Pi-1$ ma inny ciąg reszt. Dodatkowo zauważmy, że jest dokładnie $p_1p_2\ldots p_n=\Pi$ różnych ciągów reszt, gdyż pierwszą resztę można wybrać na p_1 sposobów, drugą na p_2 sposobów itd. To oznacza, że ciągów reszt jest tyle samo, co rozważanych liczb, więc każdy ciąg jest ciągiem reszt dokładnie jednej liczby. Czytelnik, który spotkał się wcześniej z podstawami teorii liczb, z pewnością zauważył, że właśnie udowodniliśmy Chińskie twierdzenie o resztach, o którym

można też przeczytać w opracowaniu zadania *Permutacja* w niniejszej książeczce czy w książce [20].

Powróćmy teraz do problemu znalezienia liczby elementów zbioru $\{0,1,\ldots,\Pi-1\}$, które dzielą się przez p_k i nie dzielą się przez żadne p_j dla $j>k$. Dzięki udowodnionemu twierdzeniu, zamiast zliczać liczby z rozważanego zbioru, możemy zliczać ciągi reszt. Te, które nas interesują, muszą mieć następujące własności:

- reszta z dzielenia przez p_j dla $j>k$ musi być niezerowa — możemy więc wybrać ją na p_j-1 sposobów,
- reszta z dzielenia przez p_k musi być równa 0, czyli mamy tylko 1 możliwość,
- reszta z dzielenia przez p_j dla $j<k$ może być dowolna, co daje p_j możliwości.

Jest więc $c_k = p_1\cdot p_2\cdot\ldots\cdot p_{k-1}\cdot 1\cdot(p_{k+1}-1)\cdot\ldots\cdot(p_n-1)$ poszukiwanych ciągów reszt, zatem:

$$C_k=\frac{c_k}{\Pi}=\frac{p_1p_2\ldots p_{k-1}(p_{k+1}-1)\ldots(p_n-1)}{p_1p_2\ldots p_n}=\frac{(p_{k+1}-1)(p_{k+2}-1)\ldots(p_n-1)}{p_kp_{k+1}\cdots p_n}.$$

Algorytm wzorcowy

Wzór jest gotowy, ale czy to koniec zadania? Już na pierwszy rzut oka widać, że występujące w nim liczby (nawet po skróceniu liczników i mianowników) będą iloczynami około 1 000 liczb rzędu 1 000 000 000, a więc będą duże. Stąd na pewno trzeba będzie zaimplementować własną arytmetykę. Przy obliczaniu wyniku będziemy musieli uporać się z dwoma problemami:

- jak zaplanować obliczenia tak, by algorytm był odpowiednio efektywny (w sensie złożoności czasowej),
- jak zapewnić względną pierwszość liczników i mianowników.

Zastanówmy się, czy wystarczające byłoby podejście bezpośrednie — wyznaczenie wszystkich liczników i mianowników przez proste wymnożenie czynników i następnie skrócenie wyniku przez *NWD* licznika i mianownika znalezione za pomocą algorytmu Euklidesa. Powiedzmy, że występująca w obliczeniach liczba jest dla nas *duża*, jeśli potencjalnie może mieć $9n$ cyfr (zagadka dla Czytelnika: skąd wzięła się magiczna stała 9?). Liczbę uznamy za *małą*, jeśli mieści się w typie 32-bitowym ze znakiem.

Zauważmy, że dla każdego ułamka musimy wykonać:

- $O(n)$ mnożeń dużej liczby przez małą (domnażając wynik kolejno przez małe czynniki), każde domnożenie ma złożoność $O(n)$,
- wyznaczenie *NWD* dwóch dużych liczb w standardowej złożoności $O(n^2)$,
- podzielenie dwóch dużych liczb przez ich *NWD*, każde dzielenie w standardowej złożoności $O(n^2)$.

Daje to nam całkowitą złożoność $O(n^3)$, gdyż powyższe operacje wykonujemy dla wszystkich n ułamków. Nie jest to zatem rozwiązanie wystarczająco szybkie.

Pierwszy pomysł na przyspieszenie obliczeń to skorzystanie z wartości C_{k+1} w trakcie obliczania C_k. Pozwala na to następująca reguła rekurencyjna, wynikająca prosto ze wzoru na C_k:

$$C_k = \frac{p_{k+1}-1}{p_k} C_{k+1}$$

dla $k < n$ oraz $C_n = \frac{1}{p_n}$. Możemy więc obliczać szukane ułamki, poczynając od ostatnich, domnażając licznik i mianownik uzyskanego już C_{k+1} przez pewne małe wartości. W ten sposób wykonujemy w sumie $O(n)$ mnożeń dużej liczby przez małą, każdorazowo w złożoności $O(n)$. Niestety, aby skrócić wynikowe ułamki, nadal musimy stosować kosztowne obliczanie *NWD* oraz dzielenie dużych liczb.

W tym miejscu pojawia się kolejny pomysł, pozwalający jeszcze lepiej wykorzystać C_{k+1} w trakcie obliczania C_k. Jeśli mamy już C_{k+1} przedstawione jako $\frac{a_{k+1}}{b_{k+1}}$, gdzie $a_{k+1} \perp b_{k+1}$, to analogiczny ułamek reprezentujący $C_k = \frac{a_k}{b_k}$ spełnia następującą zależność:

$$\frac{a_k}{b_k} = \frac{p_{k+1}-1}{p_k} \frac{a_{k+1}}{b_{k+1}}.$$

Obliczenie go przez zwykłe wymnożenie może sprawić, że licznik i mianownik przestaną być względnie pierwsze z kilku powodów:

1. $p_{k+1}-1$ i p_k nie są względnie pierwsze,
2. $p_{k+1}-1$ i b_{k+1} nie są względnie pierwsze,
3. p_k i a_{k+1} nie są względnie pierwsze.

Warto więc przed wymnożeniem poskracać przez największe wspólne dzielniki kolejno:

1. ułamek $\frac{p_{k+1}-1}{p_k}$,
2. licznik (skrócony) powyższego ułamka i b_{k+1},
3. mianownik (skrócony) powyższego ułamka i a_{k+1}.

Wykonane operacje spowodują, że po prostym wymnożeniu skróconych liczb dostaniemy wartość C_k w formie ułamka nieskracalnego.

Powyższy algorytm wymaga skrócenia dwóch małych liczb (czyli wykonania algorytmu Euklidesa i dzielenia), co robimy w czasie stałym, oraz dwukrotnie skrócenia liczby małej i dużej, co robimy w czasie $O(n)$. W ten sposób dla każdego ułamka C_k wykonujemy operacje o złożoności $O(n)$, co prowadzi do wzorcowego algorytmu o złożoności $O(n^2)$.

Kilka słów o dużych liczbach

Opisując rozwiązanie wzorcowe, stwierdziliśmy, że potrafimy wykonać następujące operacje na dużych liczbach:

- pomnożenie dużej liczby przez małą w złożoności liniowej względem długości zapisu (liczby cyfr) dużej liczby,

- podzielenie dużej liczby przez małą w takiej samej złożoności,
- obliczenie *NWD* dużej liczby i małej w takiej samej złożoności,
- podzielenie dużej liczby przez dużą w złożoności kwadratowej od długości ich zapisu,
- obliczenie *NWD* dwóch dużych liczb w takiej samej złożoności.

Warto się chwilę zastanowić, czy rzeczywiście umiemy to zrobić. Mnożenie dużej liczby przez małą w czasie liniowym można zrealizować, naśladując szkolne mnożenie pisemne — łatwo zauważyć, że jest to algorytm liniowy, jeśli liczba, przez którą mnożymy, ma ustaloną, małą liczbę cyfr. Podobnie efektywne jest dzielenie dużej liczby przez małą metodą „pod kreską".

Oczywiście powyższe dzielenie możemy wykorzystać do obliczania reszt z dzielenia dużej liczby przez małą[1]. Z kolei umiejętność obliczania reszty przydaje się do wyznaczenia *NWD*. Jeśli a jest dużą liczbą, zaś b małą, to $NWD(a,b) = NWD(a \bmod b, b)$. Od tego momentu mamy już do czynienia tylko z małymi liczbami, gdyż $a \bmod b < b$, więc ich *NWD* liczymy w czasie stałym. Stąd *NWD* dużej i małej liczby wyznaczamy w czasie liniowym.

Czwartą operację, czyli dzielenie dużej liczby przez inną dużą liczbę, wykonujemy także podobnie do dzielenia pisemnego, które tym razem ma jednak złożoność kwadratową względem długości dzielnej.

Euklides nie zadbał o naprawdę duże liczby

Ostatnia, piąta operacja jest nieco trudniejsza. Standardowa implementacja algorytmu Euklidesa, polegająca na redukcji $NWD(a,b) = NWD(a \bmod b, b)$, może wymagać wykonania tylu kroków, ile cyfr ma krótsza liczba. W każdym kroku liczymy resztę modulo dla dwóch dużych liczb. Wykonanie tego poprzez dzielenie liczb zajmuje czas kwadratowy względem długości tych liczb. Zatem cały algorytm może działać nawet w czasie proporcjonalnym do sześcianu długości liczb.

Okazuje się jednak, że można osiągnąć złożoność kwadratową liczenia *NWD*. Trzeba się jednak przy tym wykazać sporym sprytem! Niech a i b będą dużymi liczbami, których *NWD* chcemy obliczyć. Przedstawmy a i b w systemie dwójkowym — potrafimy wykonać to w czasie kwadratowym: po prostu dzielimy liczbę a czy b wielokrotnie przez dwa, za każdym razem sprawdzając parzystość (czyli ostatni bit) wyniku i ustalając w ten sposób kolejne bity przekształcanej liczby. Następnie skorzystajmy z kilku prostych obserwacji:

1. Jeśli $b = 0$, to $NWD(a,b) = a$ (i symetrycznie w przypadku $a = 0$).
2. Jeśli a i b są parzyste, to $NWD(a,b) = 2 \cdot NWD(\frac{a}{2}, \frac{b}{2})$.
3. Jeśli a jest parzyste, a b nie, to $NWD(a,b) = NWD(\frac{a}{2}, b)$.
4. Jeśli b jest parzyste, a a nie, to $NWD(a,b) = NWD(a, \frac{b}{2})$.
5. Jeśli a i b są nieparzyste i $a > b$, to $NWD(a,b) = NWD(a-b, b)$.

[1] Nieco bardziej elegancko można to zrealizować, traktując dużą liczbę jak wielomian i wyznaczając jej wartość modulo mała za pomocą schematu Hornera.

Podobnie jak to ma miejsce w klasycznym algorytmie Euklidesa, wśród powyższych mamy własności pozwalające rekurencyjnie zmniejszać a i b (2–5) oraz przypadek brzegowy (1). Z własności 2–4 korzystamy, gdy na końcu zapisu przynajmniej jednej z liczb występuje zero. Z własności 5 — gdy mamy dwie liczby nieparzyste. Redukcje 2–4 polegają na podzieleniu przez 2 liczb a i/lub b (robimy to w czasie stałym, obcinając bit zero na końcu liczby) i, w przypadku drugim, wymnożeniu wyniku wywołania rekurencyjnego przez 2 — robimy to w czasie liniowym. W przypadku piątym potrzebujemy liniowego czasu, by obliczyć różnicę a i b. Oczywiście przy stosowaniu reguł 2-4 łączna liczba cyfr binarnych rozważanych liczb zmniejsza się co najmniej o 1. Gdy jesteśmy zmuszeni zastosować regułę 5 (która nie powoduje wzrostu liczby cyfr binarnych rozważanych liczb), to zaraz po niej wystąpi przypadek trzeci lub czwarty. Zatem najwyżej dwa ruchy są potrzebne, by zmniejszyć sumę długości liczb binarnych a i b co najmniej o 1. Stąd wynika, że algorytm składa się co najwyżej z liniowej liczby kroków. Ponieważ każdy krok potrafimy wykonać w czasie liniowym ze względu na sumę długości liczb, to uzyskany algorytm jest kwadratowy. Praktyka pokazuje, że jest on również stosunkowo prosty w implementacji, dzięki uniknięciu kłopotliwej operacji wyznaczania reszty z dzielenia dla dużych liczb.

Dobre rady na przyszłość

Skoro już jesteśmy przy dużych liczbach, to warto wspomnieć, że zapis binarny pojawia się w ich implementacji raczej rzadko. Wiele algorytmów na dużych liczbach ma złożoności kwadratowe i większe, dlatego w praktyce istotne jest zapisanie liczby w jak najkrótszej postaci (np. skrócenie zapisu c razy przekłada się na skrócenie czasu działania algorytmu kwadratowego o czynnik $\frac{1}{c^2}$). Zazwyczaj stosuje się system pozycyjny o podstawie 10^9, aby „cyfry" liczby można było zapisać za pomocą 32 bitów. Nie wolno wówczas zapomnieć, by mnożenie takich „cyfr" wykonywać na 64 bitach. Czasem, przy bardziej skomplikowanych algorytmach, używa się podstawy reprezentacji równej 10^6, aby pomieścić na 64 bitach wynik mnożenia trzech takich cyfr. Zaleca się przy tym używanie jako podstawy potęgi 10, aby uniknąć dodatkowych przekształceń systemu dziesiętnego w binarny i odwrotnie przy wczytywaniu i wypisywaniu liczb.

Zauważmy także, że w algorytmie wzorcowym zyskaliśmy sporo na czasie, zastępując operacje na dwóch długich argumentach operacjami, w których tylko jedna liczba była długa. O ile bowiem podzielenie dużej liczby przez małą jest stosunkowo proste, o tyle dzielenie dwóch dużych liczb zajmuje sporo czasu (cennego zwłaszcza na zawodach) i często jest mało efektywne. To samo podejście okazuje się skuteczne w wielu innych przypadkach.

Testy

Wszystkie testy zostały wygenerowane losowo przy pomocy dwóch parametrów — liczby lampek **n** i maksymalnej wielkości liczby przy przycisku $\mathbf{max_p}$. Poniższa tabelka zawiera listę wartości tych parametrów w dziesięciu testach użytych do sprawdzania rozwiązań.

Nazwa	n	$\mathbf{max_p}$	Opis
lam1.in	5	20	test poprawnościowy
lam2.in	5	100	test poprawnościowy

Nazwa	n	$\max_p$	Opis
lam3.in	10	200	test poprawnościowy
lam4.in	100	20 000	test poprawnościowy
lam5.in	200	20 000	test poprawnościowy
lam6.in	200	1 000 000	test wydajnościowy
lam7.in	500	1 000 000	test wydajnościowy
lam8.in	1 000	1 000 000	test wydajnościowy
lam9.in	1 000	100 000 000	test wydajnościowy
lam10.in	1 000	1 000 000 000	test wydajnościowy

Ciekawostki

Dzień próbny finału XV OI odbył się pierwszego kwietnia. Z tej okazji uczestnicy przez pierwsze pół godziny zawodów dysponowali jedynie przykładowymi wejściami i wyjściami, bez historyjki stojącej za nimi. Mimo to, jednemu zawodnikowi udało się w tym czasie odgadnąć wzór oraz wysłać rozwiązanie go obliczające, chociaż nie używające dużych liczb. Szacun.