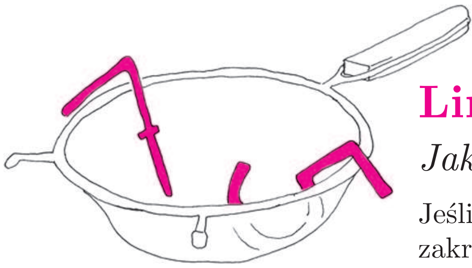


Liniowe sito

Jakub RADOSZEWSKI

Jeśli potrzebne nam są do czegoś liczby pierwsze z pewnego początkowego zakresu, zazwyczaj wyznaczamy je za pomocą sita Eratostenesa. Zaczynamy od wypisania kolejno wszystkich liczb od 2 do n. Następnie zaznaczamy 2 i wykreślamy wszystkie jej wielokrotności, zaznaczamy 3 i wykreślamy jej wielokrotności, dalej to samo z kolejnymi niewykreślonymi liczbami: 5, 7 itd. Jest to bardzo efektywna metoda; wykonujemy w niej rzędu $O(n \log\log n)$ operacji, o czym można przekonać się, czytając artykuł pt. „Jak szybko działa sito?" w *Delcie* 4/2012. $O(n \log\log n)$ to *prawie* $O(n)$, ale jednak nie. Większa niż liniowa złożoność czasowa związana jest z tym, że w sicie Eratostenesa pozwalamy sobie na pewną *rozrzutność*, gdyż niektóre liczby złożone wykreślamy wielokrotnie. Zależnie od szczegółów implementacyjnych, pierwszą taką liczbą złożoną jest 6 albo 12. Zastanówmy się jednak, czy nie dałoby się każdej liczby złożonej wykreślić dokładnie raz?

Tak jak w zwykłym sicie, na początku tworzymy listę wszystkich liczb od 2 do n. Znów w pierwszym kroku ustalamy liczbę pierwszą $p = 2$. Dalej będzie trochę inaczej niż poprzednio, ale nie tak znowu skomplikowanie. Rozważamy kolejne (niewykreślone i nie mniejsze niż p) liczby q na liście i dla każdej z nich wykreślamy wszystkie liczby postaci $p^i \cdot q$ dla $i = 1, 2, \ldots$

Zobaczmy to na przykładzie; niech $n = 40$.

Na początku mamy $p = 2$ i listę przeglądamy, począwszy od $q = 2$. W pierwszym kroku wykreślamy liczby postaci $2^i \cdot 2$ dla $i \geqslant 1$, czyli potęgi dwójki:

	2	3	~~4~~	5	6	7	~~8~~	9	10
11	12	13	14	15	~~16~~	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	~~32~~	33	34	35	36	37	38	39	40

Przyszła pora na $q = 3$; wykreślamy wszystkie liczby postaci $2^i \cdot 3$:

	2	3	~~4~~	5	~~6~~	7	~~8~~	9	10
11	~~12~~	13	14	15	~~16~~	17	18	19	20
21	22	23	~~24~~	25	26	27	28	29	30
31	~~32~~	33	34	35	36	37	38	39	40

Kolejną nieskreśloną liczbą jest $q = 5$. Wykreślamy więc liczby postaci $2^i \cdot 5$:

	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~
11	~~12~~	13	14	15	~~16~~	17	18	19	~~20~~
21	22	23	~~24~~	25	26	27	28	29	30
31	~~32~~	33	34	35	36	37	38	39	~~40~~

Nietrudno zgadnąć, co będzie dalej. Czytelnik zechce sprawdzić, że po pełnym rozpatrzeniu $p = 2$ na liście pozostaną po prostu wszystkie liczby nieparzyste.

	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~
11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~
21	~~22~~	23	~~24~~	25	~~26~~	27	~~28~~	29	~~30~~
31	~~32~~	33	~~34~~	35	~~36~~	37	~~38~~	39	~~40~~

Przyszła wreszcie pora na $p = 3$. Rozważamy wszystkie dotychczas nieskreślone wartości q nie mniejsze niż p. Zaczynamy od $q = p = 3$, czyli najpierw wykreślamy potęgi trójki:

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~
21	~~22~~	23	~~24~~	25	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	33	~~34~~	35	~~36~~	37	~~38~~	39	~~40~~

Następnie mamy $q = 5$ i wykreślamy tylko 15; $q = 7$ i znika 21; dalej wykreślimy jeszcze 33 i 39:

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	25	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	35	~~36~~	37	~~38~~	~~39~~	~~40~~

Rozwiązanie zadania F 824.
Zanurzoną część góry możemy otrzymać z całej góry przez jednokładność o skali ξ względem wierzchołka stożka. Równowaga siły ciężkości i wyporu prowadzi do warunku

$$\rho\pi R^2 hg = \rho_w \pi R^2 h \xi^3 g,$$

gdzie R jest promieniem podstawy stożka, h jego wysokością, a ρ i ρ_w gęstościami odpowiednio lodu i wody. Stąd $\xi = \sqrt[3]{\rho/\rho_w}$; podstawiając $\rho = 0{,}92\ \mathrm{g/cm^3}$ i $\rho_w = 1\ \mathrm{g/cm^3}$, stwierdzamy, że zanurzona część wysokości to $\xi \approx 0{,}97$ wysokości góry, a zatem nad powierzchnię wystaje zaledwie około 3% wysokości góry.

W ten sposób rozważyliśmy $p = 3$, przechodzimy do $p = 5$. Za pomocą $q = 5$ wykreślimy już tylko 25, a za pomocą $q = 7$ wykreślimy 35.

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~

Dla każdego kolejnego kandydata na p liczba p^2 jest większa niż n, możemy więc śmiało stwierdzić, że na liście pozostały nam już tylko liczby pierwsze (i to wszystkie w badanym zakresie).

W tym przykładzie rzeczywiście każdą liczbę złożoną skreśliliśmy dokładnie raz. Nie jest to przypadek; liczbę złożoną

$$k = p_1^{a_1} p_2^{a_2} \ldots p_l^{a_l} \qquad (p_1 < p_2 < \ldots < p_l)$$

skreślamy w przebiegu algorytmu, w którym $p = p_1$ i $q = p_2^{a_2} \ldots p_l^{a_l}$ (a jeśli $l = 1$, to, oczywiście, dla $p = q = p_1$). To, w szczególności, oznacza, że dla każdej liczby złożonej obliczamy przy okazji jej najmniejszy dzielnik pierwszy.

Możemy więc pokusić się o stwierdzenie, że cała metoda wykonuje wymarzone $O(n)$ operacji. Aby jednak dało się ją przekuć na algorytm o liniowym koszcie czasowym, musimy przyjrzeć się dokładniej używanej strukturze danych. Mamy tu do czynienia z dwoma typami operacji: znajdowaniem następnego elementu na liście (potrzebne do rozpatrywania kolejnych p i q) oraz wykreśleniem danej liczby z listy. O ile pierwszą z tych operacji wykonuje się na liście standardowo w czasie stałym, o tyle druga z nich sprawia pewien kłopot. W przypadku listy nie mamy bowiem swobodnego dostępu do jej elementów. Taki dostęp daje np. tablica, która to struktura nie pozwala z kolei na proste usuwanie bądź wykreślanie elementów…

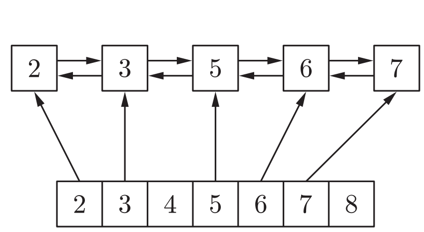
Odpowiednią strukturą danych może być także para tablic $prev[2..n]$ i $next[2..n]$, w których trzymamy dowiązania listowe: $prev[x]$ oznacza element poprzedzający x na liście, zaś $next[x]$ oznacza element następujący po x. Dla uproszczenia implementacji nasza lista będzie cykliczna (tzn. po elemencie n następuje 2). Sytuację, gdy x został wykreślony z listy, oznaczymy przez $next[x] =$ **nil**. Oto pseudokod takiego algorytmu:

```
for i := 2 to n do
  prev[i] := i − 1;
  next[i] := i + 1;
next[n] := 2; prev[2] := n;
p := 2;
while p² ⩽ n do
  q := p;
  while p · q ⩽ n do
    r := p · q;
    while r ⩽ n do
      next[prev[r]] := next[r];
      prev[next[r]] := prev[r];
      next[r] := nil;
      r := r · p;
    q := next[q];
  p := next[p];
```

Co ciekawe, mimo trzech zagnieżdżonych pętli **while** algorytm jest liniowy.

Klucz do rozwiązania tej ostatniej trudności stanowi połączenie dwóch wspomnianych struktur danych, czyli listy i tablicy. Dokładniej, oprócz *dwukierunkowej* listy wszystkich niewykreślonych elementów utrzymujemy tablicę wskaźników do poszczególnych liczb od 2 do n na liście. Jeśli danej liczby nie ma już na liście, w odpowiednim polu tablicy możemy wstawić np. **nil**. Za pomocą wskaźników zapisanych w tablicy łatwo znajdujemy elementy listy, które chcemy wykreślić, a potem usuwamy je już standardowo – podpinając do siebie wzajemnie następny i poprzedni element listy.

Przykładowo, opisana struktura danych dla $n = 8$ po wykreśleniu potęg dwójki wygląda tak:



Teraz możemy już z całą pewnością powiedzieć, że otrzymaliśmy liniową metodę wykrywania wszystkich liczb pierwszych w danym początkowym zakresie liczb. Czy w praktyce jest ona lepsza od klasycznego sita Eratostenesa? Nie sądzę. Za to ma ona pewne ciekawe z teoretycznego punktu widzenia zastosowanie, o czym można przeczytać w kolejnym artykule w tym numerze *Delty*.

Opisany algorytm pochodzi z 1978 roku i jest autorstwa Davida Griesa i Jayadeva Misry. Co ciekawe, wszystkie liczby pierwsze z zakresu od 2 do n można znaleźć nawet w czasie $O(n/\log\log n)$ (Paul Pritchard, 1981).