Jakub Radoszewski
Treść zadania, Opracowanie

Jakub Radoszewski, Bartłomiej Wołowiec
Program

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

Najdzielniejszy dzielnik

Dana jest liczba całkowita $N > 1$. Powiemy, że liczba całkowita $d > 1$ jest dzielnikiem N z krotnością $k > 0$ (k całkowite), jeżeli $d^k \mid N$ oraz $d^{k+1} \nmid N$. Dla przykładu, liczba $N = 48 = 16 \cdot 3$ ma następujące dzielniki: 2 z krotnością 4, 3 z krotnością 1, 4 z krotnością 2, 6 z krotnością 1 itd.

Powiemy, że liczba d jest **najdzielniejszym dzielnikiem** *liczby N, jeżeli d jest dzielnikiem N z krotnością k i N nie ma dzielników z krotnościami większymi niż k. Przykładowo, najdzielniejszym dzielnikiem liczby 48 jest 2 (z krotnością 4), a najdzielniejszymi dzielnikami liczby 6 są: 2, 3 i 6 (każdy z krotnością 1).*

Twoim zadaniem jest wyznaczenie krotności najdzielniejszego dzielnika liczby N oraz wyznaczenie liczby wszystkich najdzielniejszych dzielników N.

Wejście

Na standardowym wejściu znajduje się trochę nietypowy opis liczby N. Pierwszy wiersz zawiera jedną liczbę całkowitą n ($1 \leqslant n \leqslant 600$). Drugi wiersz zawiera n liczb całkowitych a_i ($2 \leqslant a_i \leqslant 10^{18}$) pooddzielanych pojedynczymi odstępami. Opis ten oznacza, że $N = a_1 \cdot a_2 \cdot \ldots \cdot a_n$.

Wyjście

Pierwszy wiersz standardowego wyjścia powinien zawierać największą liczbę całkowitą dodatnią k, dla której istnieje dzielnik d liczby N, taki że $d^k \mid N$. Drugi wiersz powinien zawierać jedną liczbę całkowitą dodatnią D będącą liczbą (najdzielniejszych) dzielników N o krotności k.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`3`	`4`
`4 3 4`	`1`
natomiast dla danych:	*poprawnym wynikiem jest:*
`1`	`1`
`6`	`3`

Ocenianie

Jeżeli Twój program wypisze poprawną krotność k najdzielniejszego dzielnika liczby N, natomiast nie wypisze w drugim wierszu liczby najdzielniejszych dzielników D lub wypisana przez niego liczba tych dzielników będzie niepoprawna, to uzyska 50% punktów za dany test (oczywiście, odpowiednio przeskalowane w przypadku przekroczenia połowy limitu czasowego).

Rozwiązanie

Wprowadzenie

Oznaczmy przez $kr(d, N)$ zdefiniowaną w treści zadania krotność dzielnika d w ramach liczby N. Naszym zadaniem jest, wśród wszystkich dzielników danej (w pośredni sposób) liczby N, wskazanie takich dzielników, których krotność w N jest największa, i obliczenie pewnych statystyk dotyczących tych dzielników. Ponieważ liczba N może być bardzo duża (po wymnożeniu wszystkich liczb a_i możemy otrzymać nawet liczbę rzędu $10^{10\,800}$), a także może mieć bardzo wiele dzielników, więc widać wyraźnie, że musimy wymyślić rozwiązanie istotnie sprytniejsze niż bezpośrednie przeglądanie wszystkich dzielników N.

Intuicja może nam podpowiadać, że coś wspólnego z rozwiązaniem mają *liczby pierwsze*, gdyż są one „najmniejszymi” możliwymi dzielnikami, a zatem ich krotności w ramach N powinny być duże. Jednakże drugi przykład z treści zadania wyraźnie pokazuje, że wśród najdzielniejszych dzielników liczby mogą pojawiać się także liczby złożone. Warto więc nieco dokładniej przyjrzeć się temu zagadnieniu.

Fakt 1. *Jeżeli liczba d jest najdzielniejszym dzielnikiem N, to również każdy jej dzielnik pierwszy p jest najdzielniejszym dzielnikiem N.*

Dowód: Faktycznie, jeśli $d^a \mid N$ oraz $d = p \cdot r$, to także $p^a \mid N$, a więc $kr(p, N) \geqslant kr(d, N)$. ∎

Fakt 2. *Jeżeli d jest najdzielniejszym dzielnikiem N, to d nie jest podzielne przez kwadrat żadnej liczby całkowitej większej niż 1, równoważnie: w rozkładzie d na czynniki pierwsze każda liczba pierwsza występuje co najwyżej raz.*

Dowód: Zauważmy na wstępie, iż $kr(d, N) > 0$. Gdyby zatem jakaś liczba pierwsza p wystąpiła w rozkładzie d na czynniki pierwsze co najmniej dwukrotnie, to mielibyśmy:

$$kr(p, N) \;\geqslant\; 2 \cdot kr(d, N) \;>\; kr(d, N),$$

czyli d nie byłby najdzielniejszym dzielnikiem N. ∎

Fakt 3. *Jeśli parami różne liczby pierwsze $p_1, p_2, \ldots, p_a$ są najdzielniejszymi dzielnikami N, to także ich iloczyn $p_1 p_2 \ldots p_a$ jest najdzielniejszym dzielnikiem N.*

Dowód: Oczywisty. ∎

Na podstawie spostrzeżeń zawartych w Faktach 1-3 możemy już dużo lepiej wyobrazić sobie, jak wygląda zbiór $ndz(N)$ wszystkich najdzielniejszych dzielników N. Jeżeli $p_1, p_2, \ldots, p_c$ są wszystkimi liczbami pierwszymi zawartymi w $ndz(N)$, to

$$ndz(N) = \left\{ \prod_{\substack{j>0 \\ 1 \leqslant i_1 < i_2 < \ldots < i_j \leqslant c}} p_{i_1} p_{i_2} \ldots p_{i_j} \right\},$$

czyli jest to zbiór o liczności $2^c - 1$ złożony ze wszystkich (różnych od 1) iloczynów pewnych spośród liczb p_i, które to iloczyny zawierają każdą z liczb p_i co najwyżej raz.

Aby rozwiązać nasze zadanie, wystarczy zatem rozważyć wszystkie liczby pierwsze dzielące N, dla każdej z nich obliczyć jej krotność w ramach N, wziąć maksimum z tych krotności i wyznaczyć liczbę liczb pierwszych odpowiadających temu maksimum. I tak jak wszystkich dzielników liczba N może mieć wiele, tak dzielników *pierwszych* ma już zdecydowanie mniej — łatwo widać, że każda z liczb a_i ma co najwyżej $\log_2 a_i$ dzielników pierwszych (pytanie kontrolne: dlaczego?), a różnych dzielników pierwszych a_i jest w rzeczywistości jeszcze mniej.

Pozostaje pytanie, jak wyznaczyć wszystkie te dzielniki pierwsze. Jest to tzw. problem *faktoryzacji* liczby, który w ogólności jest, niestety, trudny — nie jest znany żaden algorytm rozwiązujący ten problem w złożoności czasowej wielomianowej względem *długości zapisu* rozkładanej liczby (czyli, innymi słowy, względem logarytmu z tej liczby). Stąd kiepskim pomysłem byłoby wymnożenie wszystkich liczb a_i i operowanie na otrzymanej, bardzo dużej liczbie N. Zamiast tego będziemy operować na reprezentacji liczby N z treści zadania, czyli właśnie na ciągu a.

Rozwiązanie wzorcowe

Faza I: pierwiastek sześcienny

Oznaczmy:

$$M \stackrel{\text{def}}{=} \max\{a_1, a_2, \ldots, a_n\}.$$

Jest to liczba nieprzekraczająca 10^{18}. Najpopularniejszy algorytm rozkładu dowolnej liczby z na czynniki pierwsze polega na rozważaniu kolejnych liczb pierwszych do pierwiastka z z i dzieleniu przez nie liczby z do skutku (trochę mniej efektywne rozwiązanie otrzymujemy, rozważając wszystkie liczby naturalne do $\sqrt{z}$, a nie tylko te pierwsze). Na końcu tego procesu pozostanie albo jedynka, albo jakaś liczba pierwsza większa od $\sqrt{z}$ (pytanie kontrolne do Czytelnika: dlaczego nie może pozostać liczba złożona?). Gdyby zastosować taką właśnie metodę do wszystkich liczb a_i, otrzymalibyśmy algorytm o złożoności czasowej $O(n\sqrt{M})$. To trochę za wolno jak na ograniczenia z zadania.

Skoro nie możemy pozwolić sobie na sprawdzenie tak dużej liczby liczb pierwszych, spróbujemy ją ograniczyć. Okazuje się, że niezłym pomysłem jest rozważenie jedynie liczb pierwszych nieprzekraczających

$$m \stackrel{\text{def}}{=} \sqrt[3]{M}.$$

Dla każdej z tych liczb pierwszych wykonamy operację *skrócenia*, polegającą na wyznaczeniu jej krotności w ramach każdej z liczb a_i, zsumowaniu tych krotności i porównaniu wyniku sumowania (będącego zarazem krotnością tej liczby pierwszej w ramach N) z najlepszym dotychczas otrzymanym — patrz poniższy pseudokod.

```
{ Zmienne globalne (obie początkowo równe 0): }
{   k — maksymalna znaleziona krotność }
{   c — liczba wykrytych liczb pierwszych odpowiadających aktualnemu k }
```

```
4:
5: procedure skrócenie(p : pierwsza)
6: begin
7:   kr := 0;
8:   for i := 1 to n do
9:     while a_i mod p = 0 do begin
10:      a_i := a_i div p;
11:      kr := kr + 1;
12:    end
13:  if kr > k then begin
14:    k := kr; c := 1;
15:  end else if kr = k then c := c + 1;
16: end
```

Można by zapytać, dlaczego tak a nie inaczej dobraliśmy górną granicę zakresu rozważanych liczb pierwszych, tj. m. Odpowiedź na to pytanie uzyskamy, jeśli zastanowimy się, jak będą wyglądać liczby a_i po wszystkich wykonanych skróceniach. Otóż każda będzie iloczynem co najwyżej *dwóch* liczb pierwszych — faktycznie, gdyby po skróceniach było $a_i = p \cdot q \cdot r$ dla liczb naturalnych $p, q, r > 1$, to mielibyśmy $p, q, r > m$, czyli $a_i > m^3 = M$, co nie jest możliwe. Stąd w dalszej części rozwiązania musimy już tylko rozważyć te pozostałe liczby pierwsze. Zanim to uczynimy, oszacujmy jeszcze złożoność czasową Fazy I.

Zacznijmy od tego, że wszystkie liczby pierwsze nie większe niż m możemy wyznaczyć za pomocą sita Eratostenesa[1], którego złożoność czasowa to

$$O(m \log \log m). \tag{1}$$

Liczba rozważanych liczb pierwszych jest rzędu $O\left(\frac{m}{\log m}\right)$ [2]. Dla każdej z tych liczb wykonujemy skrócenie:

```
Faza I:
  P := sito(1, 2, ..., ⌊m⌋);
  foreach p ∈ P do skrócenie(p);
```

Koszt czasowy tych wszystkich skróceń zależy od liczby obrotów pętli **for** oraz **while**, z których ta druga odpowiada wykonywaniu dzieleń przez p: sekwencji udanych dzieleń zakończonej jednym nieudanym. Udanych dzieleń (wiersze 10-11 w pseudokodzie funkcji *skrócenie*) jest łącznie co najwyżej

$$n \log M = n \log (m^3) = 3n \log m = O(n \log m), \tag{2}$$

[1] Więcej o tym algorytmie można przeczytać np. w opisie rozwiązania zadania *Zapytania* z XIV Olimpiady Informatycznej [14].

[2] To oszacowanie można znaleźć w książce [23], a także w różnych książkach poświęconych teorii liczb.

gdyż tyle maksymalnie (niekoniecznie różnych) dzielników pierwszych ma liczba N. Z kolei dzieleń nieudanych jest co najwyżej

$$O(|P| \cdot n) = O\left(\frac{m}{\log m} \cdot n\right). \tag{3}$$

Łączny koszt czasowy Fazy I otrzymujemy jako sumę kosztów częściowych (1), (2) oraz (3):

$$O\left(m \log\log m + n \log m + \frac{m}{\log m} \cdot n\right) = O(m \log\log m + mn/\log m). \tag{4}$$

Faza II: wspólne dzielniki

Po wykonaniu pierwszej fazy każda z liczb a_i jest jedynką, liczbą pierwszą lub iloczynem dwóch liczb pierwszych. Niestety, wszystkie występujące liczby pierwsze są duże, więc musimy poszukiwać ich jakoś sprytniej niż dotychczas. W tej fazie pozbędziemy się *wspólnych* dzielników pierwszych występujących w różnych liczbach a_i. Skorzystamy z tego, że największy wspólny dzielnik dwóch liczb umiemy obliczać bardzo efektywnie, za pomocą algorytmu Euklidesa.

Załóżmy, że pewne dwa wyrazy ciągu a, tj. a_i oraz a_j, posiadają wspólny dzielnik pierwszy i nie są równe, czyli, bez straty ogólności, zachodzi $1 < a_i < a_j$. Widać, że mogą one mieć co najwyżej jeden taki wspólny dzielnik pierwszy p. Wówczas $a_i = p \cdot q$, $a_j = p \cdot r$ i każda z liczb q, r, $q \neq r$, jest albo pierwsza, albo równa 1. To oznacza, że $\mathrm{nwd}(a_i, a_j)$ jest równe p, czyli że jest szukanym wspólnym dzielnikiem pierwszym.

Ilustrację tego rozumowania stanowi poniższy pseudokod.

```
Faza II:
  for i := 1 to n − 1 do
    for j := i + 1 to n do begin
      d := nwd(a_i, a_j);
      if (d > 1) and (d < max(a_i, a_j)) then skrócenie(d);
    end
```

Na złożoność czasową powyższego algorytmu składa się obliczanie największych wspólnych dzielników oraz wykonywanie skróceń. Algorytm Euklidesa wywołujemy $O(n^2)$ razy, każde wywołanie ma złożoność czasową $O(\log M) = O(\log m)$, więc łączny koszt obliczania nwd to:

$$O(n^2 \log m). \tag{5}$$

Aby oszacować łączny koszt czasowy wszystkich skróceń, zauważmy, że łączna liczba liczb pierwszych występujących w ramach a_i po wykonaniu Fazy I nie przekracza $2n$. Stąd będzie co najwyżej $2n$ skróceń, z których każde wykonamy w czasie $O(n)$, co daje łączny koszt czasowy wszystkich skróceń rzędu

$$O(n^2). \tag{6}$$

To pokazuje, że złożoność czasowa Fazy II to $O(n^2 \log m)$.

Faza III: kwadraty

Liczby pierwsze mogą powtarzać się także w ramach jednej liczby a_i, tzn. wtedy, gdy jest ona kwadratem liczby pierwszej. W tym przypadku bardzo łatwo zidentyfikować taką liczbę pierwszą i wykonać odpowiednie skrócenie:

```
Faza III:
  for i := 1 to n do
    if a_i > 1 then begin
      d := ⌊√a_i⌋;
      if d² = a_i then skrócenie(d);
    end
```

Łączny koszt czasowy obliczania wartości d to $O(n)$ lub $O(n \log m)$, w zależności od tego, czy mamy do dyspozycji operację pierwiastkowania w czasie stałym, czy też musimy ją sami zaimplementować za pomocą wyszukiwania binarnego. W językach programowania dostępnych na zawodach dostępna jest stosowna funkcja `sqrt`. To pokazuje, że dominujący w tej fazie jest tak naprawdę koszt czasowy operacji skrócenia, który w poprzedniej fazie oszacowaliśmy na $O(n^2)$.

Faza IV, ostatnia

W poprzednich fazach pozbyliśmy się różnych rodzajów dzielników pierwszych liczby N. Zastanówmy się, jak mogą wyglądać otrzymane w rezultacie liczby a_i. Widzimy, że każda z nich jest albo jedynką, albo liczbą pierwszą, albo iloczynem dwóch *różnych* liczb pierwszych. Co więcej, jeśli ustalimy pewne $a_i > 1$, to wszystkie pozostałe elementy ciągu są albo równe a_i, albo względnie pierwsze z a_i (czyli nie posiadają wspólnych dzielników pierwszych z a_i). To oznacza, że każdą grupę równych wartości w ciągu a_i możemy rozważyć oddzielnie. I teraz, jeśli wartość a_i występuje w ciągu b razy, to dostarcza nam ona z krotnością b albo jedną, albo dwie liczby pierwsze, różne od dotychczas rozważonych. Żeby jednak sprawdzić, czy jedną, czy dwie, musimy rozłożyć liczbę a_i na czynniki pierwsze... Niestety, jest to smutna wiadomość, gdyż, jak wcześniej stwierdziliśmy, w ogólności nie jest to problem, który potrafimy łatwo rozwiązać.

Na szczęście możemy trochę „oszukać”. Zauważmy, że do obliczenia wyniku (parametry k oraz c) wystarczy nam tylko informacja, *czy* dane a_i jest liczbą pierwszą, czy iloczynem dwóch liczb pierwszych, a w ogóle nie interesuje nas to, *jakie* są to liczby! To z kolei można sprowadzić do pytania, czy liczba a_i jest pierwsza czy złożona. Zamiast problemu faktoryzacji otrzymujemy zatem problem badania *pierwszości* liczby, a to już jest dużo lepsza sytuacja. Otóż istnieją wielomianowe (znów, względem długości zapisu liczby) algorytmy testujące pierwszość liczb. Najpopularniejsze z nich to algorytmy *randomizowane*[3]: algorytm Millera-Rabina i Solovaya-Strassena. Więcej o tych

[3]Od niedawna (2002 r.) znany jest także deterministyczny (czyli nierandomizowany) algorytm testujący pierwszość liczb, a mianowicie algorytm AKS (nazwa pochodzi od pierwszych liter nazwisk twórców: Agrawal, Kayal, Saxena). Algorytm ten korzysta jednak z zaawansowanych narzędzi i ma bardzo dużą, choć wielomianową, złożoność czasową, więc w praktyce nie jest obecnie wykorzystywany.

algorytmach można przeczytać w Internecie (patrz np. Wikipedia), a dokładny opis algorytmu Millera-Rabina znajduje się m.in. w książce [21]. W tym miejscu ograniczymy się do przedstawienia skróconego opisu wspólnej struktury tych algorytmów.

W algorytmie dysponujemy pewną losową procedurą, której przekazujemy rozważaną liczbę i która zwraca jeden z komunikatów: „na pewno złożona" albo „nie wiem, czy pierwsza czy złożona". Wiemy, że procedura ta nigdy nie zwraca nieprawdy, a jeśli liczba faktycznie jest złożona, to procedura zwraca wynik „na pewno złożona" z pewnym dodatnim prawdopodobieństwem p (w teście Millera-Rabina mamy $p \geqslant 3/4$). I teraz uruchamiamy wspomnianą procedurę wielokrotnie (s razy); jeśli kiedykolwiek orzeknie, że testowana liczba jest złożona, to tak też jest w rzeczywistości, a w przeciwnym przypadku zakładamy, że liczba jest pierwsza. Zauważmy, że prawdopodobieństwo tego, że po wykonaniu s prób zakończonych odpowiedzią „nie wiem" mamy wciąż do czynienia z liczbą złożoną, to $(1-p)^s$, zakładając, że wyniki zwracane przez rozważaną procedurę są niezależne (jako zmienne losowe). Dla algorytmu Millera-Rabina i $s = 50$ powtórzeń mamy konkretnie:

$$(1-p)^s \leqslant \frac{1}{4^{50}} \approx 0.00000000000000000000000000000079,$$

czyli to prawdopodobieństwo jest naprawdę znikome.

Wystarczy teraz użyć dowolnego z wspomnianych testów pierwszości (funkcja *pierwsza*) i mamy gotowe rozwiązanie Fazy IV:

```
1:  Faza IV:
2:    for i := 1 to n do
3:      if a_i > 1 then begin
4:        kr := 1;
5:        for j := i + 1 to n do
6:          if a_j = a_i then begin
7:            a_j := 1;  kr := kr + 1;
8:          end
9:        if pierwsza(a_i) then ile := 1 else ile := 2;
10:       a_i := 1;  { dla porządku }
11:       if kr > k then begin
12:         k := kr;  c := ile;
13:       end else if kr = k then c := c + ile;
14:     end
```

Koszt czasowy tej fazy to $O(n^2)$ (wiersze 5-8) plus n wywołań testu pierwszości. Jeżeli, dla przykładu, użyjemy testu Millera-Rabina, którego złożoność czasowa to $O(s \log^3 M)$, przy czym s to liczba powtórzeń losowej procedury testującej, to koszt czasowy Fazy IV wyniesie:

$$O(n^2 + ns \log^3 m). \tag{7}$$

Podsumowanie

Po wykonaniu wszystkich czterech faz znamy wartości dwóch parametrów: k — krotności każdej z liczb ze zbioru $ndz(N)$, oraz c — liczby liczb pierwszych w zbiorze

$ndz(N)$. Na mocy wcześniejszych rozważań, oznacza to, że powinniśmy wypisać kolejno liczby k oraz 2^c-1. Pojawia się tu dodatkowo jeden drobny kłopot: liczba 2^c-1 może być bardzo duża. Aby ją obliczyć, musimy zaimplementować własne operacje arytmetyczne na dużych liczbach. Wystarczą nam zaledwie mnożenie przez 2 i odejmowanie jedynki, a każdą z tych operacji implementujemy jak odpowiednie działanie pisemne. Ponieważ koszt czasowy każdej z tych operacji to $O(\log(2^c-1)) = O(c)$, a musimy ich łącznie wykonać c, więc złożoność czasowa obliczania 2^c-1 wynosi:

$$O(c^2) = O((n \log M)^2) = O(n^2 \log^2 m). \tag{8}$$

Możemy już teraz wyznaczyć łączną złożoność czasową całego algorytmu. Otrzymujemy ją, sumując składniki (4)–(8):

$$O\left(m \log\log m + mn/\log m + ns \log^3 m + n^2 \log^2 m\right).$$

Wygląda to dosyć skomplikowanie, ale wystarczy zauważyć, że każdy ze składników, przy ograniczeniach z zadania, daje rozsądnego rzędu wielkości liczbę wykonywanych operacji.

Implementacje tego rozwiązania można znaleźć w plikach: `naj.cpp`, `naj2.cpp`, `naj3.pas`, `naj4.pas` i `naj5.cpp` (różne implementacje testu Millera-Rabina) oraz `naj1.cpp` (test Solovaya-Strassena).

Warto też na koniec przypomnieć sobie o dodatkowym warunku z treści zadania, że rozwiązanie wypisujące poprawnie wartość k uzyskuje za zadanie 50% punktów. Wbrew pozorom, pominięcie parametru D nie tylko pozwala uniknąć implementacji arytmetyki dużych liczb, lecz także i złożonych algorytmów testowania pierwszości! Faktycznie, wynik testu pierwszości (wiersz 9 w implementacji Fazy IV) jest używany wyłącznie do obliczania wartości parametru c. Implementację tak uproszczonego rozwiązania można znaleźć w pliku `najb4.cpp`.

Rozwiązanie alternatywne

Znajomość jeszcze jednego klasycznego algorytmu teorioliczbowego pozwala skonstruować inne, również całkiem efektywne rozwiązanie. Chodzi tutaj o heurystykę „ro" Pollarda[4], służącą do faktoryzacji liczb. Załóżmy, że dana jest liczba złożona z, której najmniejszy dzielnik pierwszy jest równy p. Wówczas heurystyka Pollarda znajdzie ten dzielnik (lub pewną jego wielokrotność, jednakże niebędącą wielokrotnością z) w oczekiwanej liczbie kroków rzędu $O(\sqrt{p})$, przy czym każdy krok jest wykonywany w czasie $O(\log^2 z)$.

Okazuje się, że można użyć tej heurystyki zamiast Faz II-IV rozwiązania wzorcowego. W tym celu wykonujemy następujące kroki:

- Na początku wykonujemy Fazę I, dokładnie tak samo jak w rozwiązaniu wzorcowym.
- Następnie dla każdej liczby $a_i > 1$ sprawdzamy, czy jest liczbą pierwszą, czy złożoną, używając do tego celu np. testu Millera-Rabina.

[4] Opis tego algorytmu można znaleźć w książce [21].

- Po wykonaniu poprzednich kroków pozostają nam do rozłożenia liczby a_i postaci $p \cdot q$, przy czym $p \leqslant q$ są liczbami pierwszymi. Wówczas zachodzi $p \leqslant \sqrt{M}$, a zatem heurystyka Pollarda pozwala znaleźć nietrywialną wielokrotność pewnego dzielnika pierwszego liczby a_i w oczekiwanym czasie $O(\sqrt{p}) = O(\sqrt[4]{M})$. Na tej podstawie za pomocą operacji nwd rozkładamy a_i na czynniki pierwsze, po czym dokonujemy odpowiednich skróceń w ciągu a.

Łączny oczekiwany koszt czasowy tego rozwiązania jest taki sam jak koszt rozwiązania wzorcowego:

$$O\Big(m \log\log m + mn/\log m + ns \log^3 m + nM^{1/4} \log^2 m + n^2 \log^2 m\Big) =$$
$$= O\big(m \log\log m + mn/\log m + ns \log^3 m + n^2 \log^2 m\big). \quad (9)$$

W praktyce jest ono co najwyżej kilkukrotnie wolniejsze od rozwiązania wzorcowego, ale jest to różnica na tyle nieznaczna, że na zawodach była mu przyznawana maksymalna punktacja. Implementacje tego rozwiązania można znaleźć w plikach `naj6.cpp` i `naj7.pas`.

Inne rozwiązania

W gruncie rzeczy, w tym zadaniu wszystkie istotne rozwiązania powolne tudzież błędne są gorszymi implementacjami rozwiązania wzorcowego: praktycznie każdą z jego faz można wykonać w gorszej złożoności czasowej, niepoprawnie albo w ogóle o niej zapomnieć. Zainteresowany Czytelnik znajdzie rozmaite przykłady takich rozwiązań w plikach `najs[1-5].[cpp|pas]` oraz `najb[1-11].cpp`.

Testy

Zadanie było sprawdzane na 14 zestawach danych testowych, których podstawowe parametry są wymienione w poniższej tabeli. Główną grupę stanowią testy zwane *ogólnymi*, generowane poprzez wyszczególnienie liczby liczb pierwszych w $ndz(N)$ znajdowanych w poszczególnych fazach rozwiązania wzorcowego oraz maski informującej o obecności (lub jej braku) liczb pierwszych z poszczególnych faz w danym ciągu a. Poza tym w zestawie występują: testy maksymalizujące wynikowe k (typ max **k**) i wynikowe D (typ max **c**), testy wydajnościowe dla algorytmów badania pierwszości skonstruowane tylko za pomocą liczb pierwszych z Fazy IV, wreszcie test zawierający pseudolosowe liczby a_i oraz małe testy generowane ręcznie.

Nazwa	n	k	c	Opis
naj1a.in	7	4	2	$N = 941\,412\,010\,017\,164\,400$
naj1b.in	2	3	2	$N = 719\,986\,312\,752\,603\,624$
naj2.in	200	17	5	test ogólny, tylko Faza I
naj3a.in	200	2	89	test ogólny, tylko Faza I

Nazwa	n	k	c	Opis
naj3b.in	300	17 700	1	test max **k**
naj4.in	170	16	9	test ogólny, wszystkie fazy
naj5.in	300	3	63	test ogólny, bez Fazy III
naj6.in	299	24	9	test ogólny, tylko Fazy I i III
naj7a.in	300	14	21	test ogólny, bez Fazy II
naj7b.in	289	284	1	test „zupełnie losowy”
naj8a.in	300	56	4	test ogólny, wszystkie fazy
naj8b.in	300	1	1 346	test max **c**
naj8c.in	100	4	20	test ogólny, wszystkie fazy
naj9a.in	300	4	72	test ogólny, wszystkie fazy
naj9b.in	300	1	1 450	test max **c**
naj9c.in	268	15	16	test ogólny, bez Fazy III
naj9d.in	298	1	305	test wydajnościowy dla badania pierwszości
naj10a.in	300	2	191	test ogólny, wszystkie fazy
naj10b.in	300	12	20	test ogólny, wszystkie fazy
naj10c.in	300	1	304	test wydajnościowy dla badania pierwszości
naj11.in	300	12	18	test ogólny, bez Fazy IV
naj12a.in	300	10	28	test ogólny, wszystkie fazy
naj12b.in	300	22	11	test ogólny, bez Fazy I
naj13.in	300	9	21	test ogólny, wszystkie fazy
naj14a.in	300	3	90	test ogólny, bez Fazy IV
naj14b.in	300	5	57	test ogólny, bez Fazy IV