

Informatyczny kącik olimpijski (90): Zapis wieżowy

Tym razem omówimy rozwiązanie zadania *Zapis wieżowy* z Akademickich Mistrzostw Polski w Programowaniu Zespołowym z 2006 roku. Dla ciągu n liczb naturalnych a_1, a_2, \dots, a_n oraz liczby naturalnej m należy wyznaczyć resztę z dzielenia przez m wieży potęgowej, w której liczby z ciągu są kolejnymi wykładnikami. Innymi słowy, mamy znaleźć wartość wyrażenia

$$a_1^{a_2^{\dots^{a_n}}} \pmod{m}.$$

Przykładowo, $3^{2^3} \pmod{7} = 2$, gdyż $3^{2^3} = 3^8 = 6561 = 937 \cdot 7 + 2$. Oczywiście, bezpośrednie obliczanie potęg nie wchodzi w grę, gdyż ich wartości rosną coraz szybciej z każdym dodatkowym wykładnikiem i już liczba $4^{3^{2^3}}$ ma 3951 cyfr. Dla uproszczenia zapisu wieżę potęgową będziemy oznaczali przez $a_{1 \uparrow n}$.

W rozwiązaniu zadania pomoże nam twierdzenie Eulera, które mówi, że dla względnie pierwszych liczb naturalnych a i m spełniona jest kongruencja

$$(*) \quad a^{\varphi(m)} \equiv 1 \pmod{m},$$

gdzie $\varphi(m)$ oznacza liczbę liczb względnie pierwszych z m i nie większych niż ta liczba. Jeśli zatem liczby a_1 i m byłyby względnie pierwsze, to moglibyśmy:

- (1) wyznaczyć $\varphi(m)$,
- (2) rekurencyjnie obliczyć rozwiązanie mniejszego problemu dla ciągu a_2, \dots, a_n i modułu $\varphi(m)$, czyli $A_2 = a_{2 \uparrow n} \pmod{\varphi(m)}$,
- (3) korzystając z twierdzenia Eulera, wyznaczyć rozwiązanie

$$a_{1 \uparrow n} = a_1^{a_{2 \uparrow n}} = a_1^{\gamma \cdot \varphi(m) + A_2} \pmod{\varphi(m)} = a_1^{\gamma \cdot \varphi(m) + A_2} \equiv a_1^{A_2} \pmod{m}.$$

Przypomnijmy, że jeśli znamy rozkład modułu na czynniki pierwsze $m = p_1^{\alpha_1} \cdot \dots \cdot p_\ell^{\alpha_\ell}$, to do obliczenia $\varphi(m)$ możemy wykorzystać następujący wzór:

$$(**) \quad \varphi(m) = m \cdot \frac{p_1 - 1}{p_1} \cdot \dots \cdot \frac{p_\ell - 1}{p_\ell}.$$

Rozkład ten możemy znaleźć w czasie $O(\sqrt{m})$, przeglądając wszystkie potencjalne dzielniki liczby m nie większe niż \sqrt{m} . Zatem w takim też czasie wykonamy krok 1 algorytmu. Z kolei potęgowanie $a_1^{A_2} \pmod{m}$ z kroku 3 możemy wykonać w czasie $O(\log m)$, stosując metodę wielokrotnego podnoszenia do kwadratu. W sumie wykonamy n wywołań rekurencyjnych, w i -tym wywołaniu obliczając $A_i = a_{i \uparrow n} \pmod{m_i}$, gdzie $m_1 = m$ oraz $m_i = \varphi(m_{i-1})$ dla $i \geq 2$, co da algorytm o złożoności czasowej $O(n\sqrt{m})$.

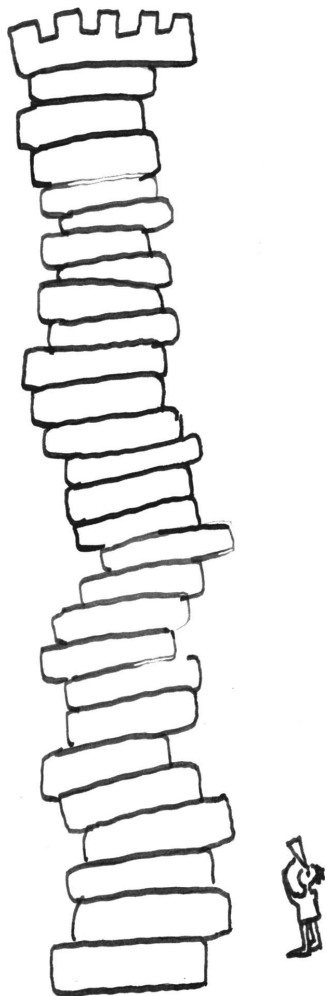
Zauważmy jednak, że ze wzoru $(**)$ widać, iż $\varphi(x)$ dla nieparzystego x jest liczbą parzystą, natomiast dla parzystego x mamy $\varphi(x) \leq \frac{x}{2}$. Wynika z tego, że ciąg kolejnych modułów $m_1, m_2, m_3, \dots, 1$ ma co najwyżej $2 \log_2 m$ wyrazów, zatem wystarczy, że wykonamy $\min(n, 2 \log_2 m)$ wywołań rekurencyjnych. A po drugie, pracę wykonaną we wszystkich wywołaniach kroku 1 można sumarycznie oszacować przez $O(\sqrt{m})$. Zatem lepszym oszacowaniem czasu działania naszego algorytmu jest $O(\sqrt{m} + \min(n, \log m) \log m)$, czyli po prostu $O(\sqrt{m})$.

Niestety, abyśmy mieli pewność, że algorytm działa poprawnie, nie tylko liczby a_1 i m muszą być względnie pierwsze, ale również liczby a_i i m_i dla wszystkich $i \geq 2$. Na szczęście twierdzenie Eulera można uogólnić, aby działało również bez założenia o względnej pierwszości. Nowa wersja brzmi następująco: dla dowolnych liczb naturalnych a , m i k spełniona jest kongruencja

$$(***) \quad a^{s+k \cdot \varphi(m)} \equiv a^s \pmod{m},$$

gdzie s jest pewną liczbą zależną od m , jednak nie większą niż $\log_2 m$.

Zanim udowodnimy to twierdzenie, zobaczymy, jak dzięki niemu naprawić nasz algorytm. Zmodyfikujemy go tak, aby nie tylko obliczał wartości A_i , ale również dodatkowy bit e_i równy 1 wtedy, gdy $a_{i \uparrow n} \geq m_i$. Znowu pokażemy, jak wykonać krok (2) algorytmu: załóżmy zatem, że $m \geq 2$ i obliczyliśmy już A_2 oraz e_2 .



Rozwiązanie zadania M 1484.
Dla $m = 0$ otrzymujemy $x = -2$, co nie spełnia warunków zadania.

W przeciwnym przypadku otrzymujemy równanie kwadratowe, niech x_1 i x_2 będą jego rozwiązaniami. Korzystając ze wzorów Viète'a, otrzymujemy

$$\begin{aligned} (x_1 + 2)(x_2 + 2) &= \\ &= x_1 x_2 + 2(x_1 + x_2) + 4 = \\ &= \frac{9m + 2}{m} + \frac{-2 + 8m}{m} + 4 = 21. \end{aligned}$$

Ponieważ $x_1 + 2, x_2 + 2$ są liczbami naturalnymi większymi od 1 o iloczynie 21, to są one równe 3 i 7, a stąd rozwiązaniami naszego równania są liczby 1 i 5. Z zależności

$$9m + 2 = 1 \cdot 5 \cdot m \text{ mamy } m = -\frac{1}{2}$$

i łatwo sprawdzić, że ta wartość parametru faktycznie spełnia warunki zadania.

Wyznamy A_1 . Jeśli $e_2 = 0$, to $a_{2\uparrow n} < \varphi(m)$, więc $A_2 = a_{2\uparrow n}$ i wystarczy przyjąć $A_1 = a_1^{A_2} \pmod m$. Z kolei jeśli $e_2 = 1$, to $a_{2\uparrow n} \geq A_2 + \varphi(m) \geq \log_2 m \geq s$, czyli

$$a_{1\uparrow n} = a_1^{a_{2\uparrow n}} = a_1^{(\gamma-1)\varphi(m)+A_2+\varphi(m)} \stackrel{(***)}{\equiv} a_1^{A_2+\varphi(m)} \pmod m.$$

Zatem w obu przypadkach mamy $A_1 = a_1^{A_2+e_2\varphi(m)} \pmod m$. Z kolei wyznaczyć e_1 można następująco: jeśli $a_1 < 2$, to $e_1 = 0$, a w przeciwnym przypadku możemy wykonać potęgowanie $a_1^{A_2+e_2\varphi(m)}$, w każdej iteracji domnażając jedno a_1 i sprawdzając, czy wynik osiągnął już m (wykonamy co najwyżej $\log_2 m$ takich iteracji). Ostatecznie złożoność czasowa całego algorytmu nie zmienia się.

Pozostaje udowodnić dane wzorem (***) uogólnienie twierdzenia Eulera. Zdefiniujmy ciąg d_i następująco:

$$d_0 = \text{nwd}(a, m), \quad d_i = \text{nwd}\left(a, \frac{m}{d_0 \cdots d_{i-1}}\right) \quad \text{dla } i \geq 1,$$

oraz niech s będzie najmniejszą liczbą, taką że $d_s = 1$. Oznaczmy też $D = d_0 \cdots d_{s-1}$.

Liczba m/D jest liczbą powstałą po usunięciu z m wszystkich czynników pierwszych występujących w a , zatem liczby a i m/D są względnie pierwsze. Z twierdzenia Eulera wynika zatem, że

$$a^{k \cdot \varphi(m/D)} \equiv 1 \pmod{m/D}.$$

Ponadto każda z liczb a/d_i jest całkowita, więc liczba a^s/D również. Mnożąc powyższe równanie przez a^s/D , dostajemy

$$a^{s+k \cdot \varphi(m/D)} / D \equiv a^s / D \pmod{m/D}.$$

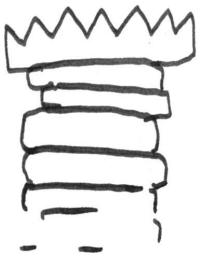
Korzystając z faktu, że kongruencja $x \equiv y \pmod w$ jest spełniona wtedy i tylko wtedy, gdy spełniona jest $xD \equiv yD \pmod wD$, możemy przemnożyć przez D obie strony i moduł powyższego równania:

$$a^{s+k \cdot \varphi(m/D)} \equiv a^s \pmod m.$$

Liczby D i m/D są względnie pierwsze, zatem z multiplikatywności funkcji φ dostajemy $\varphi(m) = \varphi(m/D)\varphi(D)$. Ponadto wykładniki w rozkładach na czynniki pierwsze liczb $m/(d_0 \cdots d_{i-1})$ zmniejszają się, więc $s \leq \log_2 m$. Zatem ostatecznie dostajemy tezę twierdzenia:

$$a^{s+k \cdot \varphi(m)} \equiv a^s \pmod m.$$

Tomasz IDZIASZEK



Odwracamy, obracamy...

Dana jest n -elementowa tablica $a[1..n]$, którą chcemy odwrócić, czyli spowodować, że jej elementy będą zapisane w kolejności $a[n], a[n-1], \dots, a[1]$. Najłatwiej to zrobić *w miejscu* (czyli używając jedynie stałej liczby komórek pamięci dla zmiennych pomocniczych), korzystając z instrukcji $\text{swap}(a[i], a[j])$ zamieniającej miejscami wartości dwóch elementów:

```
for i := 1 to [n/2] do
  swap(a[i], a[n+1-i]);
```

Nietrudno się przekonać, że powyższy kod wywołuje instrukcję zamiany jedynie $\lfloor \frac{n}{2} \rfloor$ razy, co w przypadku odwrócenia tablicy jest wynikiem optymalnym.

A teraz trudniejsze zadanie: chcemy tę samą tablicę obrócić o k komórek w lewo, czyli ustawić jej elementy w kolejności $a[k+1], a[k+2], \dots, a[n], a[1], \dots, a[k]$. I tym razem spróbujmy to zrobić, używając jedynie instrukcji zamiany.

Można w tym celu trzykrotnie wywołać omówioną przed chwilą procedurę odwracania tablicy:

```
rev(a[1..k]); rev(a[k+1..n]); rev(a[1..n]);
```

Ten kod wykona $\lfloor \frac{k}{2} \rfloor + \lfloor \frac{n-k}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$ instrukcji zamiany, co w zależności od parzystości liczb n i k da nam n lub $n-1$ instrukcji. Pytanie: czy da się mniej?

Poniższy kod obraca tablicę, dzieląc ją na $\text{nwd}(n, k)$ cykli o długości $n/\text{nwd}(n, k)$ i wykonując obrót każdego z nich niezależnie, do czego potrzebuje $n/\text{nwd}(n, k) - 1$ instrukcji zamiany:

```
for i := 1 to nwd(n, k) do
  for j := 1 to n/nwd(n, k) - 1 do
    swap(a[i+n(j-1)·k], a[i+n·j·k]);
```

Operacja $i+n·j$ oznacza tu $(i+j-1) \pmod{n+1}$. Zatem w tym rozwiązaniu użyjemy w sumie $n - \text{nwd}(n, k)$ instrukcji zamiany. A czy ten wynik da się poprawić?

T.I.