Informatyczny kącik olimpijski (41): Samogenerujący się ciąg

W tym numerze *Delty* dużo uwagi poświęcono ciągowi EKG, który zarówno z matematycznego, jak i z informatycznego punktu widzenia przejawia wiele interesujących własności. W kąciku kontynuujemy temat ciekawych ciągów liczbowych. Zajmiemy się zadaniem *Ciąg* z finału II Olimpiady Informatycznej Gimnazjalistów, w którym poproszono uczestników o wyznaczenie n-tego wyrazu pewnego ciągu, zwyczajowo wiązanego z nazwiskiem matematyka Solomona Golomba.

Ciąg ten jest niemalejącą sekwencją liczb całkowitych dodatnich $(a_k)_{k=1}^{\infty}$, taką że a_k oznacza łączną liczbę wystąpień liczby k w tej sekwencji. To już cała definicja; spróbujmy rozszyfrować, jak wygląda ten tajemniczy ciąg.

Zacznijmy od początku, czyli od wyrazu a_1. Ponieważ $a_1 > 0$, więc w ciągu musi wystąpić co najmniej jedna jedynka, a skoro ciąg jest niemalejący, to musi zaczynać się od jedynki, czyli $a_1 = 1$. Przejdźmy teraz do a_2. Otóż nie może być $a_2 = 1$, gdyż w ciągu mamy dokładnie jedną jedynkę ($a_1 = 1$). Stąd $a_2 \geqslant 2$, czyli – podobnie jak poprzednio – w ciągu jest co najmniej jedna dwójka, więc $a_2 = 2$. To oznacza, że w ciągu mamy łącznie dwie dwójki, czyli $a_3 = 2$. A zatem w ciągu mamy dwie trójki, $a_4 = a_5 = 3$. To z kolei oznacza, że mamy po trzy czwórki i piątki, czyli nasz ciąg zaczyna się tak: 1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5.

Widzimy, że możemy dalej rozumować w ten sposób; co więcej, otrzymujemy algorytm na generowanie wyrazów ciągu Golomba. Startujemy od $a_1 = 1$ i $a_2 = a_3 = 2$, po czym dla każdego kolejnego $k = 3, 4, \ldots$ na koniec ciągu wstawiamy a_k wyrazów równych k (ciąg reprezentujemy w zwykłej tablicy). Za każdym razem dodajemy więcej niż jeden wyraz, więc nigdy nie będziemy musieli „zgadywać” – kiedy dojdziemy do indeksu k, będziemy już znali a_k. Kontynuujemy to postępowanie, aż przypiszemy wartość a_n. Koszt czasowy i, niestety, także pamięciowy to $O(n)$. Za taki naturalny algorytm można było na zawodach uzyskać 40% punktów.

Spróbujmy usprawnić ten algorytm, przede wszystkim pod względem zapotrzebowania na pamięć. Kluczowe spostrzeżenie już właściwie poczyniliśmy poprzednio: kiedy natrafiamy na wyraz a_k, dopisujemy na koniec ciągu pełne a_k nowych wyrazów, a tymczasem przesuwamy się indeksem zaledwie o 1 (do a_{k+1}). A im dalej, tym a_k jest większe. To oznacza, że istotnie rośnie nam „ogon” już obliczonych wartości, które pozostają do przejrzenia. Algorytm może się zakończyć, gdy już przetworzony fragment ciągu wraz z tym ogonem ma co najmniej n wyrazów. Ponieważ ogon jest długi, więc będziemy przechowywać w tablicy tylko pewną początkową liczbę wyrazów ciągu, a ponadto będziemy pamiętać długość pozostałego ogona.

Oznaczmy przez A_k sumę k początkowych wyrazów ciągu a (niech $A_0 = 0$). W tablicy musimy przechowywać jedynie pierwsze m wyrazów ciągu, takich że $A_m \geqslant n$. Jak duże musi być to m? Sprawdźmy eksperymentalnie! Za pomocą poprzedniego algorytmu obliczamy, że $a_{1\,000\,000} = 6\,137$, co pozwala zgadnąć, iż $a_k \approx \sqrt{k}$. Przy tym założeniu mamy, że

$$A_m = a_1 + \ldots + a_m \approx \sqrt{1} + \ldots + \sqrt{m} \approx$$
$$\approx \frac{2}{3} m\sqrt{m} = \frac{2}{3} m^{3/2}.$$

Stąd $A_m \geqslant n$ dla m rzędu $n^{2/3}$. Jest to istotnie lepiej niż poprzednio, a sam algorytm pozostał prosty: obliczamy początkowy fragment ciągu długości rzędu $n^{2/3}$ (dokładną długość możemy wyznaczyć eksperymentalnie, rozważając górne ograniczenie na n), po czym obliczamy kolejne A_k do momentu, gdy $A_k > n - 1$; wówczas $a_n = k$. Za takie podejście można było otrzymać 70% punktów; pozwala ono obliczyć np. $a_{2\,000\,000\,000} = 673\,365$.

To oznacza, że musi dać się jeszcze lepiej! W poprzedniej metodzie z m-elementowego początkowego fragmentu ciągu wnioskowaliśmy o A_m wyrazach ciągu. Teraz zwiększymy tę liczbę.

Załóżmy, że mamy obliczone $a_1, \ldots, a_m$. Wówczas dla każdego $k = 1, 2, \ldots, m$ wiemy, że na pozycjach $A_{k-1} + 1, \ldots, A_k$ w naszym ciągu pojawia się liczba k. To z kolei oznacza, iż każda z a_k liczb $A_{k-1} + 1, \ldots, A_k$ występuje w ciągu k razy. Nasz ciąg wygląda zatem tak: jedno wystąpienie liczby A_1, po dwa wystąpienia liczb $A_1 + 1, \ldots, A_2$, po trzy wystąpienia liczb $A_2 + 1, \ldots, A_3$ itd. W ten sposób na podstawie pierwszych m wyrazów ciągu jesteśmy w stanie ustalić $B_m = \sum_{i=1}^{m} a_i \cdot i$ początkowych wyrazów ciągu. Liczbę B_m szacujemy podobnie jak poprzednio:

$$B_m = 1 \cdot a_1 + \ldots + m \cdot a_m \approx$$
$$\approx 1 \cdot \sqrt{1} + \ldots + m \cdot \sqrt{m} \approx$$
$$\approx \frac{2}{5} m^2\sqrt{m} = \frac{2}{5} m^{5/2}.$$

Stąd warunek $B_m \geqslant n$ zachodzi już dla m rzędu $n^{2/5}$. Dokładny opis algorytmu opartego na wartościach B_k, podobny do poprzedniego, pozostawiamy Czytelnikowi. To jest rozwiązanie wzorcowe tego zadania; za jego pomocą obliczamy, że np. $a_{10^{18}} = 160\,113\,790\,524$.

Na koniec winni jesteśmy Czytelnikowi drobne sprostowanie: otóż odgadnięte oszacowanie $a_k \approx \sqrt{k}$ jest nieprawdziwe, lecz nie jest zbyt odległe od poprawnego. W rzeczywistości $a_k \approx \phi^{2-\phi} \cdot k^{\phi-1}$, przy czym $\phi = (1 + \sqrt{5})/2$, czyli a_k jest rzędu $k^{0{,}62}$. Po więcej informacji na ten temat odsyłamy do Internetowej Encyklopedii Ciągów Liczbowych na stronie `http://oeis.org`, którą, swoją drogą, polecamy wszystkim miłośnikom ciekawych ciągów.

I jeszcze pytanie do Czytelnika: czy można dalej usprawniać podane algorytmy, wprowadzając odpowiednie ciągi pomocnicze C_k, D_k itd.?

Jakub RADOSZEWSKI