**Rozwiązanie zadania F 806.**
Po nawinięciu grubszej taśmy będzie ona zajmować powierzchnię o polu $S_1 = \pi(r_k^2 - r_0^2) = 8\pi r_0^2$. Zatem długość nawiniętej taśmy będzie równa $l = S_1/d = 8\pi r_0^2/d$, gdzie $d$ jest grubością taśmy.

Po nawinięciu cieńszej taśmy będzie ona zajmować powierzchnię o polu $S_2 = \pi(r_x^2 - r_0^2)$, a jej długość wyniesie $l = 2\pi(r_x^2 - r_0^2)/d$. Ponieważ długości taśm są jednakowe, to z porównania wzorów na nie wynika, że $r_x^2 - r_0^2 = 4\pi r_0^2$. Stąd $r_x = r_0\sqrt{5}$.

Liczba obrotów szpulki w pierwszym i drugim przypadku jest równa:

$N_1 = (r_k - r_0)/d = 2r_0/d,$
$N_2 = (r_x - r_0)/(d/2) = 2(\sqrt{5}-1)r_0/d.$

Prędkość kątowa jest stała, zatem

$$t_2 = (\sqrt{5}-1)t_1.$$

# Informatyczny kącik olimpijski (49): Monety

W tym miesiącu opiszemy zadanie *Monety*, które pojawiło się na Potyczkach Algorytmicznych w roku 2010. Dany jest ciąg $n$ rzutów monetą. Należy sprawdzić, jaka jest najdłuższa seria kolejnych rzutów, w której wypadło dokładnie $k$ razy więcej orłów niż reszek. Dla przykładu, w ciągu

RORROOROOROOORO

serie od piątego do dwunastego, jak również od szóstego do trzynastego rzutu zawierają dokładnie 6 orłów i 2 reszki. Zatem dla $k = 3$ odpowiedzią jest 8, gdyż nie istnieje żadna dłuższa seria zawierająca dokładnie 3 razy więcej orłów niż reszek.

Uprośćmy na chwilę nasze zadanie i powiedzmy, że chcemy znaleźć najdłuższą serię spełniającą warunki zadania, która zawiera pierwszy rzut. Użyjemy dwóch zmiennych $r$ i $o$, oznaczających liczby reszek i orłów, które wypadły do tej pory. Za każdym razem, kiedy po rzucie wystąpi $k \cdot r - o = 0$, będzie to oznaczało, że znaleźliśmy dłuższą serię. Zauważmy jednak, że nie potrzebujemy aż dwóch zmiennych: nie interesuje nas dokładna liczba orłów i reszek, a jedynie ich wzajemne zbilansowanie. Możemy więc trzymać jeden licznik $s$, który po każdym rzucie będziemy uaktualniali następująco: wyrzucenie reszki zwiększa $s$ o $k$, natomiast wyrzucenie orła zmniejsza $s$ o 1. Oznaczmy przez $s_i$ wartość licznika po $i$-tym rzucie (przyjmujemy też $s_0 = 0$). Teraz każde $s_i = 0$ odpowiada poprawnej serii kończącej się $i$-tym rzutem.

A co w ogólnym przypadku? Zastanówmy się, jaki warunek musi być spełniony, aby pewna seria spełniająca warunki zadania kończyła się $i$-tym rzutem monetą. Łatwo przekonać się, że jest tak wtedy, gdy istnieje taki rzut $j < i$, że wartość licznika po $i$-tym rzucie jest taka sama jak po $j$-tym rzucie ($s_i = s_j$) – wtedy rzuty od $(j+1)$-ego do $i$-tego tworzą poprawną serię. Chcielibyśmy zatem dla danego $s$ szybko stwierdzać, jaki jest (o ile istnieje) najmniejszy numer rzutu $j$, dla którego $s_j = s$. Skorzystamy w tym celu z drzewa wyszukiwań binarnych, w którym dla klucza $s$ będziemy trzymać odpowiadający mu numer rzutu. Teraz po $i$-tym rzucie, jeśli klucz $s_i$ występuje w drzewie z wartością $j$, to zgłaszamy znalezienie serii o długości $i - j$. W przeciwnym przypadku uaktualniamy drzewo, wstawiając do niego klucz $s_i$ z wartością $i$. Zauważmy, że jeśli w drzewie istniał już klucz $s_i$, to nie musieliśmy go uaktualniać, gdyż interesuje nas jedynie najmniejszy numer rzutu odpowiadający temu kluczowi. Każda operacja na drzewie zajmuje czas $O(\log n)$, zatem całe rozwiązanie działa w czasie $O(n \log n)$.

Czytelnicy, którzy słyszeli o tablicach haszujących, mogą powiedzieć, że bez kłopotu uzyskalibyśmy rozwiązanie działające w oczekiwanym czasie $O(n)$, gdybyśmy zamiast drzewa wyszukiwań binarnych użyli właśnie tablicy haszującej. To prawda, ale okazuje się, że można jeszcze lepiej – wykorzystując specyfikę naszego zadania, można zaprojektować „tablicę haszującą", która da nam rozwiązanie $O(n)$, i to w pesymistycznym przypadku!

Zamiast drzewa będziemy mieli tablicę $t[0..n-1]$. Odwołania do klucza $s$ będą realizowane przez komórkę $t[s \bmod n]$, w której będziemy trzymać listę par postaci (klucz $s$, numer rzutu). Innymi słowy, po $i$-tym rzucie sprawdzamy, czy na liście $t[s_i \bmod n]$ znajduje się para $(s_i, j)$ dla pewnego $j$. Jeśli tak, to zgłaszamy znalezienie serii o długości $i - j$, a w przeciwnym przypadku uaktualniamy listę $t[s_i \bmod n]$, wstawiając do niej nową parę $(s_i, i)$.

I teraz czas na kluczową obserwację. Załóżmy, że nastąpił drugi z powyższych przypadków. To oznacza, że na liście $t[s_i \bmod n]$ mogą znajdować się tylko pary $(s', j)$, gdzie $s' = s_i + an$ dla $a \neq 0$. Zauważmy jednak, że ilekroć nasz licznik osiągnął pewną wartość $s$, to w przyszłości już nigdy nie będzie równy $s - n$ lub mniejszy, ponieważ po każdym rzucie możemy go zmniejszyć co najwyżej o 1. Zatem na liście nie ma żadnej pary dla $a > 0$ (bo wtedy byłoby $s' \geqslant s_i + n$, co przeczy temu, że aktualna wartość licznika to $s_i$). Co więcej, skoro pary, które mogą pojawić się na liście, spełniają $s' \leqslant s_i - n$, to sytuacja, w której licznik byłby równy $s'$, nigdy już nie wystąpi. Tak więc przed dodaniem pary $(s_i, i)$ do listy $t[s_i \bmod n]$ możemy z niej usunąć wszystkie występujące na niej pary. Widać zatem, że na żadnej liście nie będzie nigdy więcej niż jednej pary – stąd czas wyszukiwania na liście faktycznie będzie stały.

*Tomasz IDZIASZEK*

Nasza tablica $t$ to po prostu tablica z haszowaniem modularnym, stosująca metodę łańcuchową do rozwiązywania kolizji. Zauważmy, że używamy tutaj matematycznej definicji operacji modulo:

$$x \bmod n = x - n\left\lfloor \frac{x}{n} \right\rfloor,$$

zatem $0 \leqslant x \bmod n < n$ nawet dla ujemnych wartości $x$. Nie jest to jednak definicja powszechnie obowiązująca w popularnych językach programowania.