# Informatyczny kącik olimpijski (135): *Little Elephant and Array*

**Zadanie:** *Dany jest $n$-elementowy ciąg liczb naturalnych $A = (a_1, a_2, \ldots, a_n)$ oraz $m$ zapytań. Każde zapytanie jest opisane za pomocą dwóch liczb naturalnych $(l, p)$, gdzie $1 \leqslant l \leqslant p \leqslant n$, i brzmi: „Ile jest dobrych liczb w podsłowie $a_l, a_{l+1}, \ldots, a_p$?" Liczba $x$ jest dobra, jeśli występuje dokładnie $x$ razy. Przykładowo, wynikiem dla podkreślonego fragmentu $A = (3, \underline{1, 2, 3, 2, 3, 3, 2}, 2)$ jest 2, gdyż dobrymi liczbami są 1 (występuje raz) oraz 3 (występuje trzy razy). Napisz program, który odpowiada na wszystkie $m$ zapytań.*

Niech $a_{l:p}$ oznacza podsłowo $a_l, a_{l+1}, \ldots, a_p$.

**Rozwiązanie** $O(m(n + \max(A))$
W pierwszym podejściu każde zapytanie rozważymy niezależnie. Załóżmy, że szukamy wyniku dla $a_{l:p}$. Na początku zliczmy wystąpienia każdej wartości w tym podsłowie. Niech $Z$ będzie tablicą zliczającą i $Z[x]$ oznacza liczbę wystąpień $x$. Taka tablica ma rozmiar rzędu $O(\max(A))$ i jej wygenerowanie zajmuje czas $O(n + \max(A))$. Wówczas wynikiem jest liczba takich $x$, że $Z[x] = x$, co możemy obliczyć w czasie $O(\max(A))$, przeglądając $Z$. Znalezienie odpowiedzi na jedno zapytanie zajmuje czas $O(n + \max(A))$, więc całe rozwiązanie działa w czasie $O(m(n + \max(A)))$.

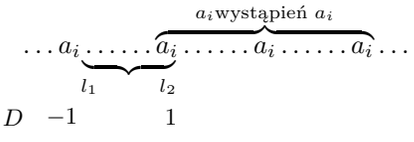
**Rozwiązanie** $O(mn)$
Zauważmy, że dobre liczby są nie większe niż $n$. Żadna liczba większa niż $n$ nie może być dobra, ponieważ długość ciągu (liczba wszystkich wystąpień) wynosi $n$. Zatem zliczanie wystąpień możemy ograniczyć do wartości nie większych niż $n$. Teraz tablica zliczająca ma rozmiar $O(n)$. Odpowiedź na jedno zapytanie realizujemy w czasie $O(n)$, a całe rozwiązanie działa w czasie $O(mn)$.

**Rozwiązanie** $O((n + m)\sqrt{n})$
W tym podejściu, przed przystąpieniem do odpowiadania na zapytania, znajdziemy zbiór $S$ zawierający kandydatów na dobre liczby. Kandydatami mogą być tylko takie liczby $x$, które występują przynajmniej $x$ razy w całym ciągu. Tak jak wcześniej zauważyliśmy, możemy ograniczyć zliczanie do wartości nie większych niż $n$, zatem $S$ generujemy w czasie $O(n)$. Okazuje się, że wszystkich kandydatów jest nie więcej niż $\lfloor \frac{-1+\sqrt{1+8n}}{2} \rfloor$.

Dlaczego? $S$ zawiera różne liczby, a jego suma jest nie większa niż $n$. Zbiór ma największą moc, kiedy zawiera kolejne liczby $1, 2, \ldots, k$. Niech $k$ oznacza największą spośród nich. Oczywiście musi zachodzić $1 + 2 + \ldots + k = \frac{(1+k)k}{2} \leqslant n$. Wystarczy skorzystać ze standardowych metod rozwiązywania nierówności drugiego stopnia, aby otrzymać, że największe $k$ wynosi $\lfloor \frac{-1+\sqrt{1+8n}}{2} \rfloor$. □

Dla każdej liczby $x$ ze zbioru $S$ przygotujmy tablicę $L_x$, gdzie $L_x[i] = 1$, jeśli $a_i = x$ lub $L_x[i] = 0$ w przeciwnym przypadku. Intuicyjnie mówiąc, skopiowaliśmy ciąg $A$, zamieniając wystąpienia $x$ na 1, zaś pozostałe liczby na 0. Dodatkowo, niech $P_x$ będzie tablicą sum prefiksowych $L_x$. Wówczas sprawdzenie, czy $x$ jest dobrą liczbą $a_{l:p}$, sprowadza się do obliczenia $L_x[l] + L_x[l+1] + \ldots + L_x[p] = P_x[p] - P_x[l-1]$. Za pomocą tak przygotowanej struktury danych potrafimy w czasie $O(1)$ sprawdzić, czy kandydat jest dobrą liczbą w podsłowie. Aby znaleźć dobre liczby w $a_{l:p}$, wystarczy sprawdzić kandydatów ze zbioru $S$, których jest $O(\sqrt{n})$. Wyznaczenie odpowiedzi na $m$ zapytań zajmuje czas $O(m\sqrt{n})$. Przygotowanie opisanych struktur danych zajmuje czas $O(n\sqrt{n})$, zatem całe rozwiązanie działa w czasie $O((n + m)\sqrt{n})$.

**Rozwiązanie** $O((n + m)\log(n))$
W tym rozwiązaniu odpowiemy na zapytania *offline*. Oznacza to, że najpierw wczytamy wszystkie zapytania, następnie obliczymy wyniki (być może w innej kolejności niż ta podana na wejściu) i na końcu wypiszemy odpowiedzi w pierwotnej kolejności zapytań. Najpierw pogrupujmy zapytania według ich końców. Niech $zap[i]$ oznacza zapytania, których koniec znajduje się w $i$. Przejdźmy teraz do przeglądania kolejnych elementów ciągu według rosnących indeksów (od 1 do $n$). Załóżmy, że rozważamy indeks $i$. Jeśli wartość $a_i$ występowała wcześniej przynajmniej $a_i$ razy, to $a_i$ jest dobrą liczbą dla niektórych fragmentów. Dokładniej, niech $l_1$ oznacza najmniejszy taki indeks, że $a_{l_1:i}$ zawiera $a_i$ wystąpień $a_i$, oraz niech $l_2$ oznacza największy taki indeks, że $a_{l_2:i}$ zawiera $a_i$ wystąpień $a_i$. Te indeksy możemy wyznaczyć, mając dla każdej wartości zapamiętane pozycje, na których ta wartość występuje. Wówczas dla każdego $l_1 \leqslant j \leqslant l_2$ fragment $a_{j:i}$ również zawiera $a_i$ jako dobrą liczbę.




Zachowajmy tę informację w tablicy $D$. Niech $D[l_1 - 1] = -1$, zaś $D[l_2] = 1$. Warto nadmienić, że jeśli wcześniej mieliśmy wyznaczony przedział $[l'_1; l'_2]$, gdzie mogło zaczynać się podsłowo z dobrą wartością $a_i$, to należy usunąć ten przedział przed nowym przypisaniem, czyli $D[l'_1 - 1] = 0$ oraz $D[l'_2] = 0$. Teraz możemy już odpowiedzieć na zapytania $zap[i]$. Załóżmy, że rozważamy zapytanie $(l, i)$. Wtedy wynikiem jest $D[l] + D[l+1] + \ldots + D[i]$. Na $D$ możemy rozpiąć drzewo przedziałowe, aby w czasie $O(\log(n))$ obliczać sumę i aktualizować wartości.

Grupowanie zapytań według ich końca realizujemy w czasie $O(n + m)$, jeśli wykorzystamy metodę zliczania. Odpowiedź na zapytanie odbywa się w czasie $O(\log(n))$, co w sumie dla $m$ zapytań daje $O(m\log(n))$. Wszystkie aktualizacje $D$ zajmują $O(n\log(n))$. Całkowita złożoność czasowa rozwiązania wynosi $O((n + m)\log(n))$.

*Bartosz ŁUKASIEWICZ*