## Informatyczny kącik olimpijski (134): *Beautiful Password*

Tym razem omówimy zadanie „Beautiful Password”, które pojawiło się na konkursie *Cuprum 2018* organizowanym przez firmę Codility.

**Zadanie:** *Dane jest słowo $S = s_1 s_2 \dots s_n$ o długości $n$, zawierające małe litery alfabetu angielskiego. Ustal długość najdłuższego podsłowa, zawierającego wyłącznie wystąpienia dwóch różnych liter w tych samych krotnościach. Przykładowo dla $S =$ ”bbbcaccaacdd” odpowiedzią jest* 6.

**Rozwiązanie** $O(n^3)$

Pierwsze rozwiązanie, które przychodzi do głowy, polega na niezależnym sprawdzeniu każdego z $O(n^2)$ podsłów – czy ma ono dokładnie dwie różne litery występujące w tych samych krotnościach. Aby to zrobić, wystarczy przejść po sprawdzanym podsłowie i zliczyć wystąpienia każdej litery. Czas potrzebny na sprawdzenie jednego podsłowa jest proporcjonalny do jego długości. Zatem całe rozwiązanie działa w czasie $O(n^3)$. Alfabet ma stały rozmiar (26 liter), więc został pominięty w obliczeniu złożoności czasowej.

**Rozwiązanie** $O(n^2)$

Szybsze rozwiązanie polega na tym, aby każdą literę słowa $S$ rozważyć jako pierwszą literę poszukiwanego podsłowa. Mając ustaloną pozycję początkową, przeglądamy kolejne litery, zliczając liczbę wystąpień każdej litery od `a` do `z`. Jeśli w którymś momencie dokładnie dwie litery będą miały taką samą dodatnią liczbę wystąpień oraz żadna inna litera się nie pojawiła, to znaleźliśmy podsłowo o poszukiwanych własnościach. Spośród znalezionych słów wybieramy najdłuższe i jego długość wypisujemy. Pozycji początkowych jest $n$, dla każdej z nich przeglądamy wszystkie litery na prawo, co daje złożoność czasową $O(n^2)$.

**Warunek: dokładnie dwie litery**

W procesie rozwiązywania zadania dobrze jest pomyśleć o różnych wariantach zadania macierzystego. W tym zadaniu możemy osłabić jeden z warunków. Otóż znajdźmy najdłuższe podsłowo, które ma dokładnie dwie różne litery (pomijamy warunek o tej samej krotności wystąpień). Takie zadanie można rozwiązać za pomocą metody gąsienicy. Na początku ustawiamy prawy i lewy koniec gąsienicy na pierwszej literze. Następnie rozszerzamy gąsienicę w prawo, dopóki pokrywa ona wystąpienia co najwyżej dwóch różnych liter. W przeciwnym przypadku skracamy gąsienicę, przesuwając lewy koniec w prawo. Pozostało nam jeszcze opisać, w jaki sposób zapisywać w pamięci liczbę różnych liter, które pokrywa gąsienica. W tym celu dla każdej litery od `a` do `z` zapamiętajmy, ile razy ta litera występuje. Gdy przesuwamy prawy koniec gąsienicy, to zwiększamy liczbę wystąpień litery, którą właśnie dodaliśmy do gąsienicy. Kiedy zaś skracamy gąsienicę, to zmniejszamy liczbę wystąpień litery, która została usunięta z gąsienicy.

Prawy i lewy koniec gąsienicy wykonają po $n-1$ ruchów. Aktualizacji przechowywanych wartości dokonujemy w czasie stałym. Zatem otrzymaliśmy rozwiązanie, które działa w czasie $O(n)$.

**Warunek: ta sama liczba wystąpień**

Weźmy na warsztat wersję zadania, w którym założymy, że $S$ zawiera wystąpienia co najwyżej dwóch różnych liter. Bez straty ogólności możemy założyć, że te litery to `a` i `b`. Chcemy znaleźć najdłuższe podsłowo, które ma tyle samo wystąpień `a` co `b`. W tym celu literze `a` przypiszmy wartość 1, zaś `b` wartość $-1$. Intuicyjnie chcemy, żeby suma wartości dwóch różnych liter sumowała się do 0. Niech zatem $a = (a_1, a_2, \dots, a_n)$ oznacza wartości przypisane kolejnym literom.

Wówczas szukamy najdłuższego fragmentu w $a$ o sumie 0. Niech teraz $p = (p_0, p_1, p_2, \dots, p_n)$ oznacza sumy prefiksowe $a$, gdzie $p_i = \sum_{j=1}^{i} a_j$. Podsłowo $a_x, a_{x+1}, \dots, a_y$ ma sumę 0, jeśli $p_y - p_{x-1} = 0$, czyli $p_{x-1} = p_y$. Zatem w ciągu sum prefiksowych szukamy dwóch takich samych wartości, pomiędzy którymi odległość jest najmniejsza. Aby to zrobić, wystarczy dla każdej wartości znaleźć jej pierwsze i ostatnie wystąpienie (można to zrobić za pomocą metody zliczania) i wziąć różnicę tych indeksów. Spośród wyników dla poszczególnych wartości wybieramy ten największy.

Wartości, które pojawią się w ciągu sum prefiksowych, są z przedziału $[-n; n]$, zatem jest ich $O(n)$. Wszystkie pozostałe kroki rozwiązania również wykonują $O(n)$ operacji, a więc całe rozwiązanie działa w czasie $O(n)$.

**Rozwiązanie** $O(n)$

Wróćmy do oryginalnej wersji zadania. Powyższe rozważania zastosujemy w rozwiązaniu wzorcowym. Otóż podzielmy słowo na bloki złożone z tych samych liter. Np. $S =$ ”*bbbcaccaacdd*” ma następujący podział *bbb*|*c*|*a*|*cc*|*aa*|*c*|*dd*. Następnie zastosujmy metodę gąsienicy (podobnie jak w podzadaniu „Warunek: dokładnie dwie litery”) na wyznaczonych blokach. Przesuwamy prawy koniec gąsienicy dopóki występują co najwyżej dwie różne litery. W przeciwnym przypadku przesuwamy lewy koniec aż do uzyskania wystąpień dokładnie jednej litery. Na rysunku zostały przedstawione fragmenty słowa pokrywane przez gąsienicę po kolejnych fazach rozszerzania, a przed skracaniem:

bbb|c|a|c|aa|cc|dd

Na otrzymanych fragmentach uruchamiamy algorytm opisany w sekcji „Warunek: ta sama liczba wystąpień”. Zauważmy, że każdy blok liter należy do co najwyżej dwóch takich fragmentów. Stąd złożoność czasowa rozwiązania wynosi $O(n)$.

*Bartosz ŁUKASIEWICZ*