Algorytmy i Struktury Danych, 13. ćwiczenia

2023-01-25 (wersja 1.01)

Zadanie 13.1

Dana jest tablica $P[0, \ldots, n]$ nieujemnych liczb całkowitych.

a) Zaproponuj algorytm, który efektywnie sprawdzi, czy P jest tablicą prefiksów-sufiksów z algorytmu KMP dla pewnego słowa nad alfabetem $\{a, b\}$?

   Uwaga: kolejne symbole słowa są indeksowane od 1.

   $P[0]$ odpowiada słowu pustemu.

b) Czy istnieje słowo nad alfabetem $\{a, b\}$, dla którego $P = [0, 0, 1, 0, 1, 2, 3, 4, 1, 2]$ jest tablicą prefiksów-sufiksów?

Rozwiązanie: a) Bez straty ogólności możemy założyć, że rekonstruowane słowo rozpoczyna się od litery a. Dla dowolnego $i > 1$, jeśli $P[i] = 0$ to dodajemy literę b, wpp. $P[i] < i$ i możemy użyć litery o indeksie $P[i]$ z rekonstruowanego słowa. Następnie obliczamy tablicę P dla tak otrzymanego słowa i porównujemy z wejściową tablicą.

b) używając algorytmu otrzymujemy słowo $S = aabaabaaa$, które niestety nie przechodzi weryfikacji ($P = [0, 0, 1, 0, 1, 2, 3, 4, \mathbf{5}, 2]$)

Zadanie 13.2

W tym zadaniu rozważamy słowa na alfabetem binarnym.

a) Ile wynosi suma elementów tablicy P dla słowa $(01)^{2020}$?

b) Ile wynosi wysokość drzewa sufiksowego (liczona liczbą krawędzi) dla słowa $(01)^{2020}$?

c) Zaprojektuj efektywny algorytm, który dla dodatniej liczby całkowitej k oraz słów x, y takich, że $|x| = k$, $|y| = k^2$ sprawdzi, ile jest w y podsłów o długości k, z których każde różni się od x na dokładnie jednej pozycji.

Rozwiązanie: a) $\sum_{i=1}^{4038} i$, tablica P ma postać $[0, 0, 0, 1, 2, 3, 4, 5, \ldots, 4038]$.

b) 2020

c) [rozwiązanie proste dla $|y| = k^2$] Budujemy zbiór słów $X = \{x_i : 1 \leq i \leq k\}$ gdzie x_i oznacza słowo x z zanegowanym i-tym znakiem. Uruchamiany algorytm

wyszukiwania wielu wzorców X w tekście y (alg. Aho-Corasick) działający w czasie $O(\sum_{i=1}^{k} |x_i| + |y|) = O(k^2)$.

c) [rozwiązanie bardziej ogólne dla dowolnej długości y] Budujemy drzewa sufiksowe dla $x\$y\#$ i $\textsc{Rev}(x)\$\textsc{Rev}(y)\#$. Przygotowujemy drzewa do zapytań LCA (Longest common ancestor) — preprocessing $O(n)$, zapytania $O(1)$. Dla każdej pozycji i w y możemy w czasie $O(1)$ wyznaczyć $l' = LCP(x, y[i, \ldots])$ (longest common prefix) oraz $l'' = LCS(x, y[1, \ldots, i+k-1])$ (longest common suffix). Jeśli $l' + l'' = k - 1$ to $y[i, \ldots, i+k-1]$ różni się od x na dokładnie jednej pozycji.

Zadanie 13.4

Powiemy, że dwa ciągi liczb wymiernych $a_1, a_2, \ldots, a_n$ i $b_1, b_2, \ldots, b_n$ są podobne, gdy istnieje takie c, że $b_i = c \cdot a_i$, dla każdego $i = 1, 2, \ldots, n$. Zaprojektuj wydajny algorytm, który dla danych dwóch ciągów dodatnich liczb wymiernych $x_1, x_2, \ldots, x_n$ oraz $y_1, y_2, \ldots, y_m$, $n \leq m$, wyznacza w ciągu y wszystkie pozycje i takie, że podciąg $y_i, y_{i+1}, \ldots, y_{i+n-1}$ oraz ciąg x są podobne. Możesz przyjąć, że operacje arytmetyczne na liczbach wymiernych wykonywane są w czasie stałym.

Rozwiązanie: Jeśli $n = 1$ to jedno-elementowy ciąg X występuje na każdej pozycji w Y.
Jeśli $n > 1$ i elementy ciągów są różne od 0, to obliczamy nowe ciągi:

$$X' = \left[\frac{x_2}{x_1}, \frac{x_3}{x_2}, \ldots, \frac{x_n}{x_{n-1}}\right]$$

$$Y' = \left[\frac{y_2}{y_1}, \frac{y_3}{y_2}, \ldots, \frac{y_m}{y_{m-1}}\right]$$

Za pomocą KMP znajdujemy wystąpienia X' w Y'.

Zadanie 13.5

W tym zadaniu rozważamy słowa zbudowane z cyfr $0, 1, \ldots, 9$. Każde takie słowo można traktować jako zapis w układzie dziesiętnym pewnej nieujemnej liczby całkowitej. Zaprojektuj efektywny algorytm, który dla danego niepustego słowa x:

a) obliczy liczbę wszystkich takich par indeksów (i, j), $1 \leq i \leq j \leq |x|$, że słowo $x[i..j]$ jest zapisem liczby podzielnej przez 3;

b) wyznaczy taką parę indeksów (i, j), $1 \leq i \leq j \leq |x|$, że słowo $x[i..j]$ jest zapisem największej liczby podzielnej przez 3 wśród wszystkich podsłów słowa x;

c) obliczy liczbę wszystkich parami różnych podsłów (różniących się długością lub znakami na odpowiadających sobie pozycjach) słowa x będących zapisami liczb podzielnych przez 3.

Rozwiązanie: Dla ustalenia uwagi, niech S oznacza zadane słowo $c_1, \ldots, c_n$ ($c_i \in \{0, \ldots, 9\}$).

a) Obliczamy tablicę $Ile(i, j)$ dla $0 \leq i \leq n$, $j \in \{0, 1, 2\}$ oznaczającą liczbę prefiksów $S[1, \ldots, i]$ o sumie $\equiv j \pmod 3$. Następnie obliczamy tablicę $Wynik(i)$ dla $0 \leq i \leq n$ oznaczającą liczbę podsłów $S[1, \ldots, i]$ o sumie podzielnej przez 3.

$$Wynik(i) = Wynik(i-1) + Ile(i-1, Suma(i))$$

b) Dla każdej pozycji i zaznaczamy najdłuższe słowo podzielne przez 3 kończące się na danej pozycji. Następnie wybieramy tylko słowa o maksymalnej długości. Za pomocą drzewa sufiksowego wyznaczamy największe leksykograficznie słowo.

c)

Zadanie 13.6

Niech x będzie słowem binarnym o długości co najmniej 2 i zawierającym co najmniej jedno `0` (zero) oraz co najmniej jedną `1` (jedynkę). Zaprojektuj efektywny algorytm, który w słowie binarnym x znajduje dwa podsłowa o maksymalnej długości, które różnią na każdej pozycji.

Rozwiązanie: Obliczamy drzewo sufiksowe dla $S' = x\$\textsc{Neg}(x)\#$. Dla takiego słowa szukamy najdłuższego słowa, które występuje w S' co najmniej dwa razy, raz w x raz w $\textsc{Neg}(x)$. Złożoność czasowa i pamięciowa: $O(n)$.

Zadanie 13.7

Zaprojektuj efektywny algorytm, który dla danych słów x, y nad alfabetem $\{d, i, k, s\}$ obliczy ile różnych słów będących cyklicznymi przesunięciami słowa x jest podsłowami słowa y.

Rozwiązanie: Budujemy drzewo sufiksowe dla słowa $S = y\$xx\#$. Znajdujemy zbiór węzłów drzewa V na głębokości $|x|$ (zbiór powinien zawierać również węzły, które znajdują się wewnątrz skompresowanych krawędzi drzewa). Dla każdego wierzchołka $v \in V$ sprawdzamy, czy w jego poddrzewie istnieje sufiks rozpoczynający się we fragmencie xx.

Zadanie 13.8

Zaproponuj efektywny algorytm, który w słowie S o długości $n \geq 3$ nad alfabetem $\{a, b\}$ znajdzie dwa najdłuższe, takie same podsłowa nie zachodzące na siebie

Przykład
W słowie *aaaaaa* takie dwa najdłuższe podsłowa to *aaa* zaczynające się na pozycji 1 i *aaa* zaczynające się na pozycji 4.
W słowie *abbabbbab* takie słowa, to np. *bbab* zaczynające się na pozycji 2 i *bbab* zaczynające się na pozycji 6.

Rozwiązanie: Budujemy drzewo sufiksowe S, oraz w każdym węźle u zapisujemy:

- liczbę liści w poddrzewie,
- głębokość (suma wag krawędzi na ścieżce $r \to u$),
- minimalny i maksymalny indeks liścia w poddrzewie.