Algorytmy i Struktury Danych, 14. ćwiczenia

2018-01-19

Spis treści

1 Algorytm Aho-Corasick

Dane: tekst T oraz zbiór wzorców $W = \{W_1, \ldots, W_k\}$.
Cel: Zaznaczenie w tekście wszystkich wystąpień wzorców z W w czasie $O(|T| + \sum_{w \in W} |w|)$.

Pre-processing:

- Przygotowujemy drzewo TRIE zawierające wzorce $W_1, \ldots, W_k$,
- Dla każdego węzła drzewa $L(v)$ oznacza napis powstały z konkatenacji etykiet na ścieżce z korzenia do v,
- Dla każdego węzła musimy obliczyć $f(v)$ (failure function), $f(v) = x$ jeśli x ($x \neq v$) jest **najniższym** węzłem drzewa TRIE, który: odpowiada sufiksowi $L(v)$ (tzn. $L(x)$ musi być sufiksem $L(v)$).

  Wyjątkowo dla korzenia $f(root) = root$. Dla pozostałych wierzchołków, wartości $f(v)$ obliczamy dla kolejnych węzłów drzewa (idąc poziomami od korzenia w dół) korzytając z następującego algorytmu:

$p = parent(v)$
$c =$ znak na etykiecie krawędzi (p, v), $x = f(p)$
while $x \neq root$ **and** *nie istnieje krawędź na literę* c *z* x **do**
  $x = f(x)$
if $x \neq parent$ **and** *istnieje krawędź na literę* c *z* x **then**
  $x = child(x, c)$ (przejście w dół na literę c)
$f(v) = x$

- Dla każdego węzła oblicz $out(v)$ (zbiór wzorców które są rozpoznawane po osiągnięciu v) (początkowo $out(v) = \{i\}$ jeśli W_i kończy się w v, potem idąc od korzenia uzupełniamy $out(v) := out(v) + out(f(v))$)

Algorytm:

- algorytm utrzymuje bieżący węzeł w drzewie TRIE (aktualnie najdłuży prefiks któregoś z wzorca, który odpowiada tekstowi), początkowa wartość to korzeń
- przeglądamy tekst znak po znaku,
- jeśli kolejny znak odpowiada krawędzi idącej w dół w drzewie, to po niej schodzimy,
- jeśli brak takiej krawędzi to przechodzimy w drzewie do kolejnego wierzchołka idąć po funkcji f.

Algorithm 1: Aho-Corasick(T, W)

```
TRIE := przygotuj drzewo TRIE dla W i wyznacz funkcje f(v) i out(v)
v = root
foreach i = 1, ..., |T| do
    c = T[i]
    while v ≠ root and nie istnieje krawędź na literę c z v do
        v = f(v)
    if istnieje krawędź na literę c z v then
        v = child(v, c) (przejście w dół na literę c)
    if out(v) ≠ ∅ then
        zgłoś wystąpienia wzroców z out(v)
```

2 Algorytm Bakera

Wyszukiwanie dwuwymiarowych wzorców.

Dany jest tekst $T[1..n, 1..n]$ oraz wzorzec $W[1..m, 1..m]$ należy wyznaczyć pary (i, j) t.że $T[i..(i + m - 1), j..(j + m - 1)] = W$.

- przygotuj zbiór wzorców $\{W_i = W[i, 1..m] : 1 \leq i \leq m\}$ (kolejne kolumny wzorca),
- za pomocą algorytmu Aho-Corasick znajdź wystąpienia wzorców W_i w poszczególnych kolumnach tekstu, wynikiem niech będzie tablica $A[1..n, 1..n]$ t.że $A[i, j] = k$ jeśli $A[i, j..(j + m)] = W_k$ lub $A[i, j] = -1$ wpp.
- za pomocą algorytmu KMP w poszczególnych wierszach tabli A odszukaj wystąpienia ciągu $1, 2, \ldots, m$ (jeśli kolumny W_i się powtarzają, to ciąg będzie trochę inny)

3 Obliczanie tablicy LCP

Tablica LCP zawiera długość najdłuższego wspólnego prefiksu pomiędzy kolejnymi sufiksami z tablicy sufiksowej.

Function LCP(T, SA)

```
n = |T|
l = 0
for i = 1, ..., n do
    k = SA^{-1}[i]
    j = SA[k - 1]
    while T[i + l] = T[j + l] do
        l = l + 1
    LCP[k] = l
    if l > 0 then l = l - 1
return LCP
```

4 Zastosowanie tablicy sufiksowej

- liczba różnych podsłów — suma wartości LCP
- najdłuższe wspólne podsłowo słów X i Y $T' = X\$Y\#$ i szukamy pary kolejnych sufiksów w $SA_{T'}$ o maksymalnej wartości LCP z których jeden rozpoczyna się w X a drugi w Y,
- wyszukiwanie wielu wzorców — liczymy tablicę sufiksową dla napisu $T' = T\$_1 w_1 \$_2 w_2 \ldots \$_k w_k$