Algorytmy i Struktury Danych, 15. ćwiczenia

2020–01–22

Spis treści

1 Rozpoznawanie grafów dwudzielnych

Algorithm 1: Bipartite($v, color$)

$C[v] = color$
for $x \in adj(v)$ **do**
  if $C[x] = nil$ **then** Bipartite($x, 1 - color$)
  else if $C[x] = color$ **then** błąd graf nie jest dwudzielny

2 6 kolorowanie grafów planarnych

$$n - m + f = 2$$

(n — liczba wierzchołków, m — liczba krawędzi, f — liczba ścian)

$$m \leq 3v - 6$$

$$\frac{1}{2} \sum deg(v_i) \leq 3n - 6$$

$$\sum deg(v_i) \leq 6n - 12$$

czyli w grafie planarnym istnieje wierzchołek o stopniu co najwyżej 5.

Algorytm 6-kolorowania: tak długo jak graf nie jest pusty, znajdź wierzchołek v o stopniu ≤ 5. Usuń go z grafu, pokoloruj rekurencyjnie resztę, nadaj nowy kolor v (zawsze jest co najmniej jeden wolny).

3 Rozpoznawanie grafów zewnętrznie planarnych

Graf jest zewnętrznie planarny jeśli:

- jest planarny,
- można go tak narysować, żeby wszystkie jego wierzchołki leżały na jednej ścianie.

Opis algorytmu można znaleźć w:

- Manfred Wiegers: *Recognizing Outerplanar Graphs in Linear Time.* WG 1986: pp. 165-176.

Lemat 1. *Jeśli graf G jest zewnętrznie planarny to zawiera wierzchołek v, taki, że* $deg(v) \leq 2$.

Algorithm 2: Outerplanar(V, E)

```
if |V| ≤ 3 then
    zaznacz wszystkie krawędzie jako zewnętrzne,
    return true
else if ∃_v deg(v) = 1 then
    Niech w sąsiad wierzchołka v
    Zaznacz krawędź (v, w) jako zewnętrzną.
    return Outerplanar(V − {v}, E − {(v, w)})
else if ∃_v deg(v) = 2 then
    Niech w1, w2 sąsiedzi wierzchołka v
    if not Outerplanar(V − {v}, E − {(v, w1), (v, w2)} + {(w1, w2)})
     then
        return false
    if (w1, w2) krawędź wewnętrzna then
        return false
    Zaznacz krawędzie (v, w1), (v, w2) jako zewnętrzne
    Zaznacz krawędzie (w1, w2) jako wewnętrzną
    return true
else
    return false
```

4 3 kolorowanie grafów zewnętrznie planarnych

Algorytm 3-kolorowania: tak długo jak graf nie jest pusty, znajdź wierzchołek v o stopniu ≤ 2. Usuń go z grafu, pokoloruj rekurencyjnie resztę, nadaj nowy kolor v (zawsze jest co najmniej jeden wolny).

5 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
```

6 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 $T' = T\$_1 w_1 \$_2 w_2 \ldots \$_k w_k$

7 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 3: Aho-Corasick(T, W)

TRIE := przygotuj drzewo TRIE dla W i wyznacz funkcje $f(v)$ i $out(v)$
$v = root$
foreach $i = 1, \ldots, |T|$ **do**
 $c = T[i]$
 while $v \neq root$ **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 $out(v) \neq \emptyset$ **then**
  zgłoś wystąpienia wzroców z $out(v)$

8 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)