Wstęp do Programowania potok funkcyjny

Marcin Kubica

2018/2019

Outline

Problem wyszukiwania wzorca w tekście

Na tym wykładzie zajmiemy się następującym problemem:

Definition (Problem wyszukiwania wzorca w tekście)

Dane: dwa napisy, wzorzec `w` (długości m) i tekst `t` (długości n), $n \geq m$.
Zakładamy, że napis to lista symboli lub liczb całkowitych z ustalonego ograniczonego zakresu.
Szukamy wystąpień wzorca w tekście (oznaczenie: jest ich k).
Wynik: lista pozycji wszystkich wystąpień wzorca w tekście.

Naiwny algorytm wyszukiwania wzorca

Example

Algorytm naiwny dla każdej możliwej pozycji w tekście sprawdza czy występuje na niej wzorzec.

```
let rec prefix w t =
  match w with
    [] -> true |
    hw::tw ->
      match t with
        [] -> false |
        ht::tt -> (hw = ht) && prefix tw tt;;
```

Naiwny algorytm wyszukiwania wzorca

Example

```
let pattern_match w t =
  let rec iter t n acc =
    let a = if prefix w t then n::acc else acc
    in
      match t with
        [] -> rev a |
        _::tail -> iter tail (n+1) a
  in iter t 1 [];;
```

Złożoność czasowa: $T(n+m) = \Theta(n \cdot m) = \Theta(n^2)$.
Można lepiej.

Algorytm Rabina-Karpa

Idea algorytmu:

- Obliczamy *odcisk palca* szukanego wzorca.
- Dla kolejnych fragmentów tekstu długości takiej jak wzorzec sprawdzamy ich odciski palca.
- Inny odcisk oznacza brak wystąpienia.
- Jeśli odciski palca się zgadzają, to sprawdzamy, czy faktycznie występuje tam wzorzec.
- Ograniczenie: obliczenie odcisku palca dla kolejnego fragmentu tekstu w czasie stałym.

Algorytm Rabina-Karpa

Odcisk palca:

- Symbole utożsamiamy z liczbami całkowitymi z przedziału od 0 do $m-1$.
- Odcisk palca to:

$$
\begin{aligned}
f(w) = & \\
&= \left(w.(m-1) + w.(m-2)\cdot p + \cdots + w.(0)\cdot p^{m-1}\right) \mod q = \\
&= \left(w.(m-1) + p\cdot\left(w.(m-2) + p\cdot\left(\cdots + p\cdot w.(0)\right)\right)\right) \mod q
\end{aligned}
$$

$$
\begin{aligned}
f(t.(i..i+m-1)) = & \\
&= \left(t.(i+m-1) + t.(i+m-2)\cdot p + \cdots + t.(i)\cdot p^{m-1}\right) \mod q = \\
&= \left(t.(i+m-1) + p\cdot\left(t.(i+m-2) + p\cdot\left(\cdots + p\cdot t.(i)\right)\right)\right) \mod q
\end{aligned}
$$

dla odpowiednio dobranych stałych p i q.

Algorytm Rabina-Karpa

Odcisk palca, c.d.:

- p powinno być liczbą pierwszą, $p \geq m$.
- q powinno być dużą liczbą pierwszą, taką żeby $q \cdot p$ mieściło się jeszcze w zakresie arytmetyki liczb całkowitych, $q > n$.
- Kolejny odcisk palca można obliczyć na podstawie poprzedniego:

$$
\begin{aligned}
& f(t.(i+1..i+m)) = \\
& \quad = \quad (f(t.(i..i+m-1)) \cdot p + t.(i+m) - t.(i) \cdot p^m) \mod q = \\
& \quad = \quad ((f(t.(i..i+m-1)) \cdot p) \mod q \quad + \\
& \qquad\quad t.(i+m) - t.(i) \cdot (p^m \mod q)) \mod q
\end{aligned}
$$

Algorytm Rabina-Karpa

Obliczanie odcisku palca

```
let p = 257
and q = 4177969;;

let fingerprint str =
  let rec sumuj acc pow pos =
    if pos < 0 then acc
    else sumuj ((acc + pow * code str.(pos)) mod q)
               ((pow * p) mod q)
               (pos -1)
  in sumuj 0 1 (length str - 1);;

let shift fi pot x y =
  let res = (((fi * p) mod q + y - x * pot) mod q)
  in if res < 0 then (res + p * q) mod q
     else res;;
```

Algorytm Rabina-Karpa

Potęgowanie mod q

```
let exp_mod b k =
  let rec iter k a =
    if k = 0 then a else iter (k - 1) ((a * b) mod q)
  in iter k 1;;
```

Sprawdzenie wystąpienia

```
let occurrence w t i =
  let rec check j =
    if j < 0 then true
    else if w.(j) = t.(i+j) then
      check (j-1)
    else false
  in check (Array.length w - 1);;
```

Algorytm Rabina-Karpa

Główna procedura

```
let pattern_match w t =
  let n = Array.length t
  and m = Array.length w
  in let f = fingerprint w m
     and pot = exp_mod p m
     in let rec iter acc i fi =
          let a = if (fi = f) && (occurrence w t i)
                  then i::acc else acc
          in if i = n-m then List.rev a
             else iter a (i+1) (shift fi pot (code t.(i))
                                             (code t.(i + m)))
        in iter [] 0 (fingerprint t m);;
```

Algorytm Rabina-Karpa

- Oczekiwana liczba dopasowań odcisków palca: $k + \frac{n-k}{q}$.
- Oczekiwana złożoność czasowa:
  $T(n+m) = \Theta(n + m \cdot (k + \frac{n-m-k+1}{q})) = \Theta(n + m \cdot k)$.
- Pesymistyczna złożoność czasowa: $T(n+m) = \Theta(n \cdot m)$.
- Złożoność pamięciowa: $\Theta(k)$.
- Można lepiej ...

Algorytm Morrisa-Pratta

- Pozwala wyznaczyć wszystkie wystąpienia wzorca w tekście w czasie liniowym.
  Ma jednak więcej zastosowań.
- Dla zadanego tekstu $t = [|t_1; t_2; \ldots; t_n|]$ obliczamy wartości tzw. *funkcji prefiksowej* P.
  $P(i)$ to największe takie j, że:
  - $0 \leq j < i$,
  - $[|t_1; \ldots; t_j|] = [|t_{i-j+1}; \ldots; t_i|]$.

  $P(i)$ to długość najdłuższego właściwego prefiksu $[|t_1; t_2; \ldots; t_i|]$, który jest równocześnie sufiksem tego napisu.

Algorytm Morrisa-Pratta

Example

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
t_i	b	a	r	b	a	r	a	r	a	b	a	r	b	a	r	b	a	r
$P(i)$	0	0	0	1	2	3	0	0	0	1	2	3	4	5	6	4	5	6

Własności funkcji prefiksowej

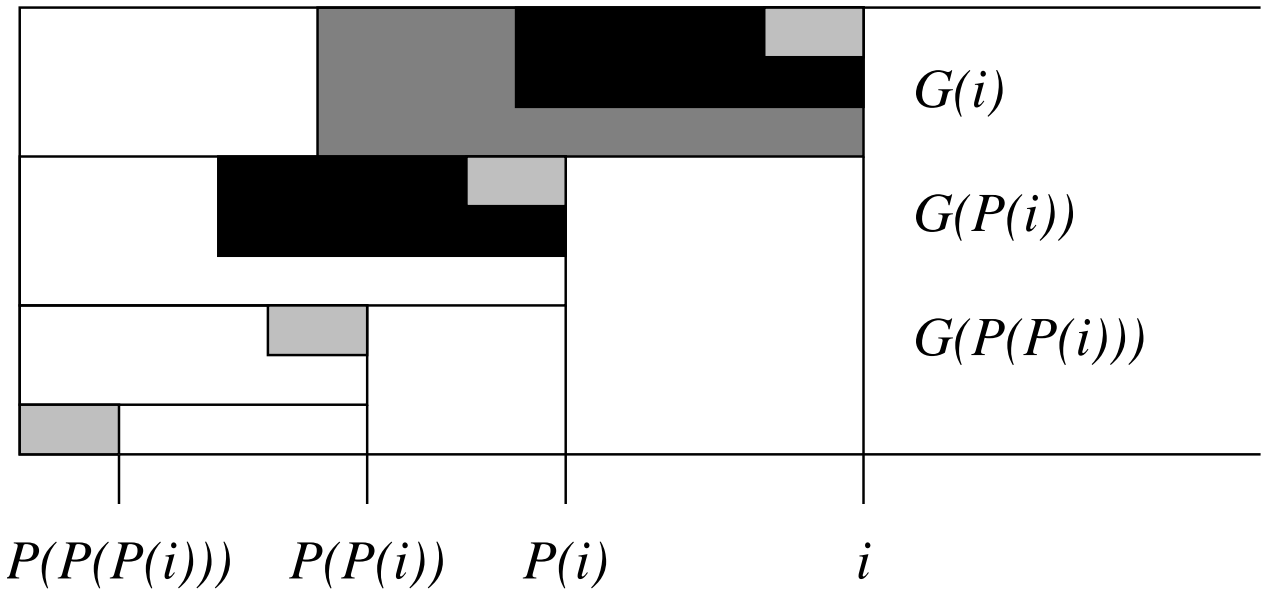
- Oznaczmy przez $G(i)$ zbiór długości wszystkich właściwych prefiksów $[|t_1; t_2; \ldots; t_i|]$, które są równocześnie sufiksami tego napisu:

$$G(i) = \{0 \leq j < i : [|t_1; \ldots; t_j|] = [|t_{i-j+1}; \ldots; t_i|]\}$$

  Przyjmujemy $G(0) = \emptyset, \quad P(0) = 0.$
- $P(i) = \max G(i)$,
- $P(1) = 0, \quad P(i) \geq 0, \quad P(i) < i$ (dla $i > 0$),
- $i > P(i) > P(P(i)) > \cdots > P^k(i) = 0$, dla pewnego $k \in \mathbb{N}$,
- $G(i+1) = \{0\} \cup \{j+1 : j \in G(i) \ \wedge \ t_{i+1} = t_{j+1}\}$,

Własności funkcji prefiksowej

- $G(i) = \{P(i)\} \cup G(P(i))$
- $P(i)$ to największy element w $G(i)$.
- Mniejsze elementy $G(i)$ to długości prefiksów $[|t_1; t_2; \ldots; t_{P(i)}|]$, które są jego sufiksami.



Własności funkcji prefiksowej

- $G(i) = \{P^j(i) : 1 \leq j \leq k\}$
- $G(i+1) = \{0\} \cup \{P^j(i) + 1 : 1 \leq j \leq k \ \wedge \ t_{i+1} = t_{P^j(i)+1}\}$
- $P(i+1) = \max(\{0\} \cup \{P^j(i)+1 : 1 \leq j \leq k \wedge t_{i+1} = t_{P^j(i)+1}\})$
- Dla $i > 0$ mamy:

$$P(i+1) = \begin{cases} P^j(i) + 1 & ; \text{ gdzie } j \text{ jest najmniejsze takie, że} \\ & \quad 1 \leq j \leq k \ \wedge \ t_{i+1} = t_{P^j(i)+1} \\ 0 & ; \text{ wpp.} \end{cases}$$

Algorytm obliczania funkcji prefiksowej

Algorytm obliczania funkcji prefiksowej

```
let pref t =
  let p = make (length(t) + 1) 0
  and pj = ref 0
  in begin
    for i = 2 to length t do
      while (!pj > 0) && (t.(!pj) <> t.(i - 1)) do
        pj := p.(!pj)
      done;
      if t.(!pj) = t.(i - 1) then pj := !pj + 1;
      p.(i) <- !pj
    done;
    p
  end;;
```

Zastosowanie tablicy prefiksowej do wyszukiwania wzorców

- Obliczamy tablicę prefiksową dla szukanego wzorca.
- Unikamy wielokrotnego przeglądania tych samych fragmentów tekstu.
- Po dopasowaniu i pierwszych znaków wzorca mamy również dopasowane $P(i)$ znaków wzorca.
- Uzyskujemy złożność czasową $\Theta(n+m)$ i pamięciową $\Theta(m+k)$.

Zastosowanie tablicy prefiksowej do wyszukiwania wzorców

Algorytm wyszukiwania wzorców

```
let find x y =
  let i = ref 0
  and j = ref 0
  and w = ref []
  and p = pref x
  in
    while !i <= length y - length x do
      j := p.(!j);
      while (!j < length x) && (x.(!j) = y.(!i + !j)) do
        j := !j + 1
      done;
      if !j = length x  then w := !i::!w;
      i := !i + if !j > 0 then !j - p.(!j) else 1
    done;
    rev !w;;
```