Wyszukiwanie wzorca w tekście

Dane:

- $T[1 \ldots n]$ – tekst długości n
- $P[1 \ldots m]$ – wzorzec długości m ($m < n$).

P występuje w T z przesunięciem s, jeśli $T[s+1 \ldots s+m] = P[1 \ldots m]$. (Takie s jest *poprawnym* przesunięciem).

Oznaczenia:

Σ^* – zbiór wszystkich słów nad alfabetem Σ (łącznie ze słowem pustym ϵ).

$|x|$ – długość słowa x

xy – *konkatenacja* (sklejenie) słów x i y

$w \sqsubset x$ – w jest *prefiksem* x (tzn. $\exists_{y \in \Sigma^*}\ x = wy$)

$w \sqsupset x$ – w jest *sufiksem* x (tzn. $\exists_{y \in \Sigma^*}\ x = yw$)

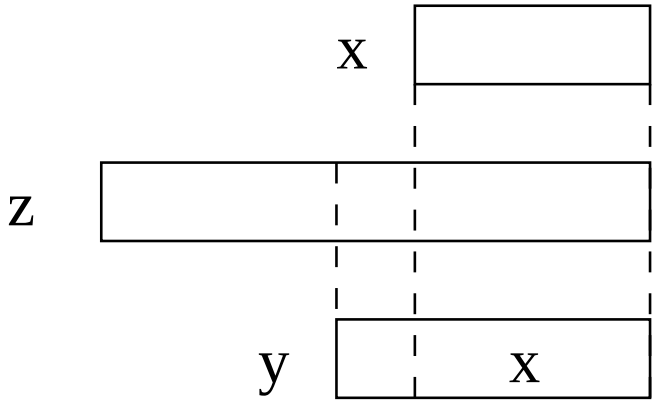
Lemat o nierozłącznych sufiksach

Lemat 1. (o nierozłącznych sufiksach) Niech $x, y, z \in \Sigma^*$, takie że $x \sqsupset z$ i $y \sqsupset z$. Wtedy:

- jeśli $|x| \leq |y|$, to $x \sqsupset y$
- jeśli $|y| \leq |x|$, to $y \sqsupset x$
- jeśli $|y| = |x|$, to $y = x$

D-d.

Przypadek $|x| \leq |y|$:



Pozostałe przypadki – podobnie. □

Algorytm naiwny

Koszt sprawdzania warunku: $P[1\ldots m] = T[s+1\ldots s+m]$ jest $\Theta(m)$.

```
Naive-String-Matcher(T, P)
1 n ←length[T]
2 m ←length[P]
3 for s ← 0 to n − m do
4    if P[1...m] = T[s + 1...s + m] then
5       drukuj "występuje z przesunięciem s"
```

Czas: $\Theta((n-m+1)\cdot m)$

($n-m+1$ iteracji `for`, $\Theta(m)$ – koszt sprawdzenia w `if`)

Jeśli m i $n-m+1$ są $\Omega(n)$, to czas: $\Theta(n^2)$.

Algorytm Karpa – Rabina

Niech $d = |\Sigma|$. Σ – cyfry w systemie o podstawie d. Wzorzec i badany fragment tekstu – m-cyfrowe liczby w systemie o podstawie d.

Przykład. $\Sigma = \{0, 1, \ldots, 9\}$. Liczbę p odpowiadającą wzorcowi $P[1 \ldots m]$ można wyznaczyć w czasie $O(m)$ wg reguły Hornera:

$$p = P[m] + 10(P[m-1] + 10(P[m-2] + \ldots + 10(P[2] + 10P[1]) \ldots))$$

Podobnie wyznaczamy t_0 – liczbę reprezentowaną przez $T[1 \ldots m]$. Dla $s \geq 0$ wyznaczamy t_{s+1} reprezentowaną przez $T[s+2 \ldots s+m+1]$ wyznaczamy z t_s:

$$t_{s+1} = 10(t_s - 10^{m-1} T[s+1]) + T[s+m+1]$$

(*Usuwamy cyfrę* $T[s+1]$, *przesuwamy pozostałe w lewo i dodajemy* $T[s+m+1]$.)

Algorytm Karpa – Rabina

Kłopot: p i t_s – DUŻE liczby
Rozwiązanie: Obliczenia – modulo q, gdzie q liczba (zwykle pierwsza), taka że $d \cdot q$ mieści się w słowie komputera.
Niech $h = d^{m-1} \bmod q$.
h odpowiada liczbie: $(100\ldots0)_d$ ($m-1$ *zer*) modulo q.
Wtedy:

$$t_{s+1} = (d(t_s - T[s+1] \cdot h) + T[s+m+1]) \bmod q$$

Jeśli $t_s \equiv p \pmod q$, to $T[s+1\ldots s+m] \neq P[1\ldots m]$ (nie musimy sprawdzać!).

```
Rabin-Karp-Matcher(T,P,d,q)
 1  n ←length[T]
 2  m ←length[P]
...
```

Rabin-Karp-Matcher

```
 3  h ← d^{m−1} mod q   ▷ czas: O(m)
 4  p ← 0
 5  t_0 ← 0
 6  for i ← 1 to m do
 7     p ← (d · p + P[i]) mod q
 8     t_0 ← (d · t_0 + T[i]) mod q
 9  for s ← 0 to n − m do
10     if p = t_s then
11        if P[1 . . . m] = T[s + 1 . . . s + m] then
12           drukuj "występuje z przesunięciem s"
13     if s < n − m then
14        t_{s+1} ← (d(t_s − T[s + 1] · h) + T[s + m + 1]) mod q
```

Czas: $O(m) + (\text{liczba błędnych strzałów}) \cdot O(m) +$
$(\text{liczba wystąpień } P \text{ w } T) \cdot O(m) + O(n - m + 1)$

Algorytm Karpa – Rabina

Ponieważ co q-ta liczba jest $\equiv p(\mathrm{mod} q)$, więc można oczekiwać (przy losowym wyborze q), że liczba błędnych strzałów jest $O(n/q)$.
Niech v – liczba wystąpień P w T.
Oczekiwany czas: $O(n) + O(m(v + n/q))$.
Jeśli v jest $O(1)$ a $q > m$, to oczekiwany czas : $O(n + m)$.

Automaty skończone

Automat skończony $M = (Q, q_0, A, \Sigma, \delta)$

- Q – skończony zbiór *stanów*
- $q_0 \in Q$ – stan *początkowy*
- $A \subseteq Q$ – zbiór stanów *akceptujących*
- $\delta : Q \times \Sigma \to Q$ – *funkcja przejść*

Rozszerzona funkcja przejść: $\phi : \Sigma^* \to Q$ taka, że

$$\begin{aligned} \phi(\epsilon) &= q_0 \\ \phi(wa) &= \delta(\phi(w), a), \quad \text{dla } w \in \Sigma^* \text{ i } a \in \Sigma \end{aligned}$$

($\phi(w)$ – *stan automatu (startującego z* q_0*) po wczytaniu słowa* w.)
M *akceptuje* w, jeśli $\phi(w) \in A$.

Automat wyszukiwania wzorca P

Niech $P_k = P[1 \ldots k]$ (prefiks P długości k), $T_i = T[1 \ldots i]$ (prefiks T długości i).
funkcja *sufiksowa* dla P – funkcja $\sigma : \Sigma^* \to \{0, \ldots, m\}$ taka że:

$$\sigma(x) = \max\{k : P_k \sqsupset x\}$$

(*Długość najdłuższego prefiksu P, który jest sufiksem x*)
Automat dla P:

- $Q = \{0, \ldots, m\}$, $q_0 = 0$, $A = \{m\}$
- $\delta(q, a) = \sigma(P_q a)$

Finite-Automaton-Matcher

```
Finite-Automaton-Matcher(T, δ, m)
1 n ← length[T]
2 q ← 0
3 for i ← 1 to n do
4     q ← δ(q, T[i])
5     if q = m then
6         s ← i − m
7         drukuj "występuje z przesunięciem s"
```

Czas: $O(n)$.

analiza poprawności

Lemat 2.(Nierówność dla funkcji sufiksowej) Dla $x \in \Sigma^*$ i $a \in \Sigma$ zachodzi $\sigma(xa) \leq \sigma(x) + 1$.

D-d. Niech $r = \sigma(xa)$.

Jeśli $r = 0$, to $r \leq \sigma(x) + 1$, bo $\sigma(x) \geq 0$.

Niech $r > 0$. Wtedy $P_r \sqsupset xa$. Stąd $P_{r-1} \sqsupset x$. Zatem $r - 1 \leq \sigma(x)$ (z definicji σ). $\square$

analiza poprawności

Lemat 2.(Nierówność dla funkcji sufiksowej) Dla $x \in \Sigma^*$ i $a \in \Sigma$ zachodzi $\sigma(xa) \leq \sigma(x) + 1$.

D-d. Niech $r = \sigma(xa)$.

Jeśli $r = 0$, to $r \leq \sigma(x) + 1$, bo $\sigma(x) \geq 0$.

Niech $r > 0$. Wtedy $P_r \sqsupset xa$. Stąd $P_{r-1} \sqsupset x$. Zatem $r - 1 \leq \sigma(x)$ (z definicji σ). $\square$

Lemat 3.(Rekurencja dla funkcji sufiksowej) Dla $x \in \Sigma^*$ i $a \in \Sigma$, jeśli $q = \sigma(x)$, to $\sigma(xa) = \sigma(P_q a)$.

D-d. Z definicji σ: $P_q \sqsupset x$. Stąd $P_q a \sqsupset xa$. Niech $r = \sigma(xa)$. Z Lematu 2: $r \leq q + 1$ (czyli: $|P_r| \leq |P_q a|$). Z Lematu 1: $P_r \sqsupset P_q a$, (bo $P_q a \sqsupset xa$, $P_r \sqsupset xa$ i $|P_r| \leq |P_q a|$). Stąd: $r \leq \sigma(P_q a)$ (czyli: $\sigma(xa) \leq \sigma(P_q a)$).

Ponieważ $P_q a \sqsupset xa$, zachodzi również $\sigma(P_q a) \leq \sigma(xa)$.

Zatem $\sigma(P_q a) = \sigma(xa)$ $\square$

analiza poprawności

Tw 4. Jeśli ϕ jest rozszerzoną funkcją przejść automatu wyszukiwania wzorca P, a $T[1 \ldots n]$ – tekstem wejściowym automatu, to $\phi(T_i) = \sigma(T_i)$ dla $i = 0, \ldots, n$.

D-d. Indukcja po i. Dla $i = 0$: $T_0 = \epsilon$ i $\phi(\epsilon) = \sigma(\epsilon) = 0$. Załóżmy, że $\phi(T_i) = \sigma(T_i)$ (założenie indukcyjne). Niech $q = \phi(T_i)$ oraz $a = T[i+1]$.

$$
\begin{aligned}
\phi(T_{i+1}) &= \phi(T_i a) && \text{(z def. } T_{i+1} \text{ i } a\text{)} \\
&= \delta(\phi(T_i), a) && \text{(z def. } \phi\text{)} \\
&= \delta(q, a) && \text{(z def. } q\text{)} \\
&= \sigma(P_q a) && \text{(z def. } \delta\text{)} \\
&= \sigma(T_i a) && \text{(z Lematu 3 i zał. ind.)} \\
&= \sigma(T_{i+1}) && \text{(z def. } T_{i+1}\text{)}
\end{aligned}
$$

□

wyznaczanie δ

```
Compute-Transition-Function(P, Σ)
1 m ← length[P]
2 for q ← 0 to m do
3   for a ∈ Σ do
4     k ← min{m + 1, q + 2}
5     repeat k ← k − 1
6       until P_k ⊐ P_q a
7     δ(q, a) ← k
8 return δ
```

Poprawność: bezpośrednio z definicji σ i δ.

`for` z wiersza 2 – $O(m)$ razy

`for` z wiersza 3 – $O(|\Sigma|)$ razy

`repeat` – $O(m)$ razy

Sprawdzenie warunku w `until` – czas $O(m)$.

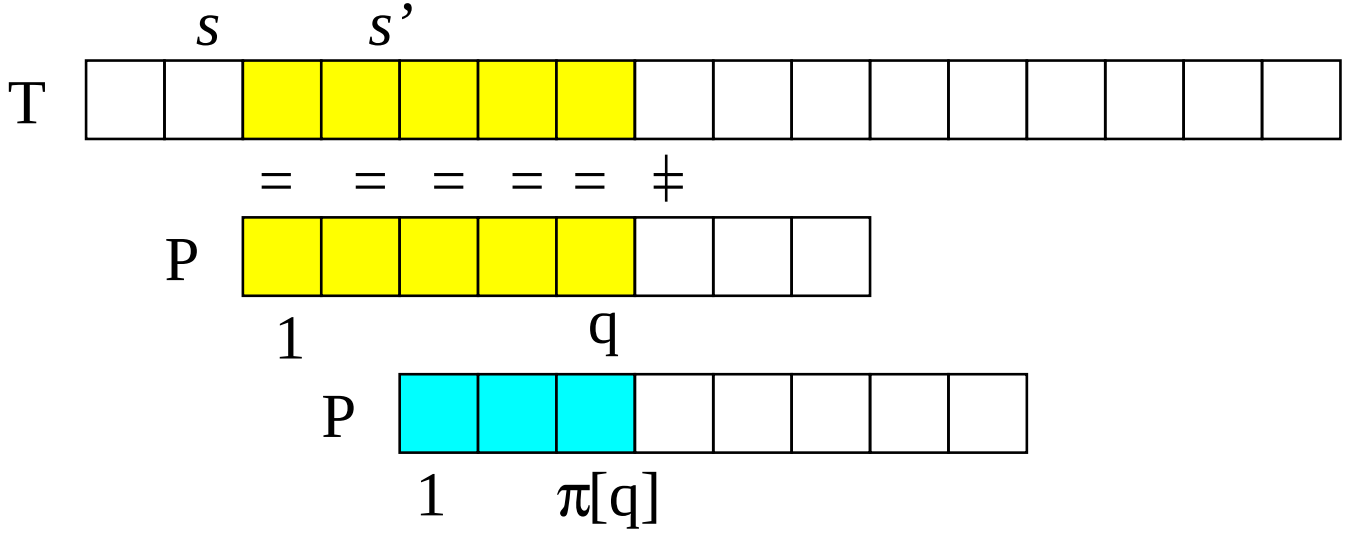
Łączny czas: $O(m^3|\Sigma|)$.

Algorytm Knutha-Morrisa-Pratta

Funkcja prefiksowa dla P – funkcja $\pi : \{1 \ldots m\} \to \{0 \ldots m-1\}$

$$\pi[q] = \max\{k : k < q \wedge P_k \sqsupset P_q\}$$

Jeśli $T[s+1 \ldots s+q] = P[1 \ldots q]$ i $T[s+q+1] \neq P[q+1]$, to następne przesunięcie, które warto sprawdzać: $s' = s + q - \pi[q]$.



KMP-Matcher

```
KMP-Matcher(T, P)
 1  n ← length[T]
 2  m ← length[P]
 3  π ← Compute-Prefix-Function(P)
 4  q ← 0
 5  for i ← 1 to n do
 6     while q > 0 and P[q + 1] ≠ T[i] do
 7        q ← π[q]  ▷  P[1 . . . q] = T[i − q . . . i − 1]
 8     if P[q + 1] = T[i] then
 9        q ← q + 1
10     if q = m then
11        drukuj "występuje z przesunięciem i − m"
12        q ← π[q]
```

Compute-Prefix-Function

```
Compute-Prefix-Function(P)
 1 m ←length[P]
 2 π[1] ← 0
 3 k ← 0
 4 for q ← 2 to m do
 5   while k > 0 and P[k + 1] ≠ P[q] do
 6     k ← π[k] ▷ P[1 . . . k] = P[q − k . . . q − 1]
 7   if P[k + 1] = P[q] then
 8     k ← k + 1
 9   π[q] ← k
10 return π
```

analiza czasu działania

Czas `Compute-Prefix-Function` – Potencjał: k.
Początkowo $k = 0$ (po wierszu 3) oraz zawsze (po każdym wierszu 6 i 8) $k \geq 0$ bo $\pi[k] \geq 0$ dla każdego k (to wynika z poprawności `Compute-Prefix-Function` co pokażemy później).
Każde wykonanie wiersza 6 – zmniejszenie k, bo $\pi[k] < k$ (to też wynika z poprawności).
Koszt każdego wykonania pętli `while` – zamortyzowany zmniejszeniem potencjału w wierszu 6.
W wierszu 8 – zwiększamy potencjał o ≤ 1.
Pętla `for` – $m - 1$ iteracji.
Stąd koszt faktyczny $\leq$ koszt zamortyzowany $= O(m)$.
Koszt `KMP-Matcher`: $O(m + n)$
(`Compute-Prefix-Function`+podobna analiza z potencjałem: q).

Lemat 5

Niech $\pi^*[q] = \{\pi[q], \pi^{(2)}[q] \ldots, \pi^{(t)}[q]\}$, gdzie $\pi^{(0)}[q] = q$, $\pi^{(i+1)}[q] = \pi[\pi^{(i)}[q]]$, oraz t – najmniejsze, takie że $\pi^{(t)}[q] = 0$.

Lemat 5 (o iteracji funkcji prefiksowej) Dla $q = 1, \ldots, m$ mamy $\pi^*[q] = \{k : k < q \wedge P_k \sqsupset P_q\}$.

D-d. *Fakt:* $i \in \pi^*[q]$ implikuje $i < q \wedge P_i \sqsupset P_q$.

Uzasadnienie Faktu: Jeśli $i \in \pi^*[q]$, to $\exists_{0<u\leq t}\ i = \pi^{(u)}[q]$.

Indukcja po u. Dla $u = 1$: $i = \pi[q]$. Stąd wynika teza, bo $\pi[q] < q \wedge P_{\pi[q]} \sqsupset P_q$.

Krok indukcyjny: Korzystając z $P_{\pi[j]} \sqsupset P_j$ i przechodniości $<$ oraz $\sqsupset$ otrzymujemy tezę dla wszytkich u.

Z *Faktu* wynika $\pi^*[q] \subseteq \{k : k < q \wedge P_k \sqsupset P_q\}$ (*).

...

c.d. dowodu Lematu 5

...

Udowodnimy nie wprost, że $\{k : k < q \wedge P_k \supset P_q\} \subseteq \pi^*[q]$.
Załóżmy, że $A = \{k : k < q \wedge P_k \supset P_q\} \setminus \pi^*[q] \neq \emptyset$. Niech $j = \max(A)$. Ponieważ $\pi[q] = \max\{k : k < q \wedge P_k \supset P_q\} \wedge \pi[q] \in \pi^*[q]$, mamy $j < \pi[q]$. Niech $j' = \min\{l \in \pi^*[q] : l > j\}$. Mamy $P_j \supset P_q$, bo $j \in \{k : P_k \supset P_q\}$, oraz $P_{j'} \supset P_q$, bo $j' \in \pi^*[q]$ i (*). Zatem z Lematu 1 $P_j \supset P_{j'}$. Co więcej $j = \max\{k : k < j' \wedge P_k \supset P_{j'}\}$, czyli $j = \pi[j']$, a ponieważ $j' \in \pi^*[q]$, więc $j = \pi[j'] \in \pi^*[q]$. Sprzeczność. $\square$

Definicja E_{q-1}

Dla $q = 2, \ldots, m$, zdefiniujmy $E_{q-1} \subseteq \pi^*[q-1]$ jako

$$
\begin{aligned}
E_{q-1} &= \{k \geq 0 : k \in \pi^*[q-1] \wedge P[k+1] = P[q]\} \\
&\quad \text{z Lematu 5:} \\
&= \{k \geq 0 : k < q-1 \wedge P_k \sqsupset P_{q-1} \wedge P[k+1] = P[q]\} \\
&\quad \text{z definicji } \sqsupset\text{:} \\
&= \{k \geq 0 : k < q-1 \wedge P_{k+1} \sqsupset P_q\}
\end{aligned}
$$

Wniosek 7

Wn. 7. Niech π – f-cja prefiksowa dla P. Wtedy, dla $q = 2, \ldots, m$ zachodzi:

$$\pi[q] = \begin{cases} 0, & \text{jeśli } E_{q-1} = \emptyset \\ 1 + \max(E_{q-1}), & \text{jeśli } E_{q-1} \neq \emptyset \end{cases}$$

D-d. Jeśli $E_{q-1} = \emptyset$ to nie ma $0 \leq k < q-1$, dla którego $P_{k+1} \sqsupset P_q$. Stąd $\pi[q] = 0$ (tylko $\epsilon = P_0 \sqsupset P_q$).
Niech $E_{q-1} \neq \emptyset$. Z definicji $\pi[q]$

$$\begin{aligned} \pi[q] &= \max\{k+1 \geq 0 : k+1 < q \wedge P_{k+1} \sqsupset P_q\} \\ &= \max\{k+1 \geq 0 : k < q-1 \wedge P_{k+1} \sqsupset P_q\} \\ &= 1 + \max(E_{q-1}) \text{ (bo } E_{q-1} \neq \emptyset). \square \end{aligned}$$

poprawność Compute-Prefix-Function

Przypisanie w wierszu 2 – poprawne, bo $\pi[1] = 0$.
Pokażemy, że wartości $\pi[q]$ dla $q > 1$ też są poprawnie wyznaczone.
Na początku każdej iteracji `for` zachodzi: $k = \pi[q-1]$ (*wymuszone w wierszach* 2, 3 *oraz w wierszu* 9) i wyznaczone są poprawne wartości $\pi[j]$, dla $j = 1, \ldots q-1$ (*założenie indukcyjne*).
W pętli `while` przeglądamy wartości $k \in \pi^*[q-1]$ do chwili znalezienia takiej, że $P[k+1] = P[q]$ (czyli $k \in E_{q-1}$). Jeśli takie k istnieje, to w wierszu 7 k jest największą taką wartością (*startujemy od* $k = \pi[q-1] = \max(\pi^*[q-1])$ *i każde iterowanie* $k \leftarrow \pi[k]$ *zmniejsza* k) i w wierszu 8 k zwiększa się o 1.
Jeśli takie k nie istnieje to w wierszu 7 i 9 $k = 0 = \pi[q]$.
Z Wniosku 7: w obu przypadkach $\pi[q]$ – poprawne.

poprawność KMP-Matcher

Wynika z poprawności `Finite-Automaton-Matcher`: Pokażemy, że wiersze 6-9 `KMP-Matcher` symulują wiersz 4 `Finite-Automaton-Matcher` t.j.: $q \leftarrow \delta(q, T[i])$.

Fakt: $\delta(q, T[i]) - 1 \in \pi^*[q] \cup \{q\}$ albo $\delta(q, T[i]) = 0$.

D-d Faktu: Niech $k = \delta(q, T[i])$. Wtedy $P_k \sqsupset P_q T[i]$ (*z def.* δ *i* σ). Zatem, albo $k = 0$, albo $k \geq 1$ i $P_{k-1} \sqsupset P_q$ (*przez usunięcie ostatniego znaku*), czyli $k - 1 \in \{k : P_k \sqsupset P_q\} = \pi^*[q] \cup \{q\}$. $\square$

Niech q' – wartość q na początku `while`. Pętla `while` w wierszach 6 i 7 przegląda $\pi^*[q'] \cup \{q'\}$ w porządku malejącym aż $P[q+1] = T[i]$ lub $q = 0$. W pierwszym przypadku q zwiększone o 1 a w przeciwnym pozostaje $q = 0$. Z *Faktu*: nic nie pominęliśmy, więc po wierszu 9 $q = \delta(q', T[i])$.

Uwaga: Wiersz 12 – aby uniknąć odwołania do $P[m+1]$. Wiersz 12 nie szkodzi, bo $\delta(m, a) = \delta(\pi[m], a)$ (równoważnie $\sigma(Pa) = \sigma(P_{\pi[m]}a)$) – ćw.

Algorytm Boyera-Moore'a

```
Boyer-Moore-Matcher(T, P, Σ)
 1  n ← length[T]
 2  m ← length[P]
 3  λ ← Last-Occurence-Function(P, m, Σ)
 4  γ ← Good-Suffix-Function(P, m)
 5  s ← 0
 6  while s ≤ n − m do
 7     j ← m
 8     while j > 0 and P[j] = T[s + j] do
 9        j ← j − 1
10     if j = 0 then
11        drukuj "występuje z przesunięciem s"
12        s ← s + γ[0]
13     else s ← s + max(γ[j], j − λ[T[s + j]])
```

Heurystyka niezgodności

Załóżmy, że $P[j] \neq T[s+j]$. Niech $k = \max(\{0\} \cup \{i : 1 \leq i \leq m \wedge T[s+j] = P[i]\})$ Wtedy możemy przesunąć wzorzec o $j - k$:

- $k = 0$: $T[s+j]$ nie występuje w P i można przesunąć P o j nie pomijając żadnego wystąpienia P w T.
- $k < j$: Ostatnie wystąpienie $T[s+j]$ w P jest $j - k$ pozycji na lewo od $T[s+j]$. Aby dopasować P do T trzeba go przesunąć o $\geq j - k$ pozycji.
- $k > j$: Proponowane przesunięcie $j - k$ jest < 0 i zostanie zignorowane.

Last-Occurence-Function

```
Last-Occurence-Function(P,m,Σ)
1 for each a ∈ Σ do
2    λ[a] = 0
3 for j ← 1 to m do
4    λ[P[j]] ← j
5 return λ
```

Czas: $O(|\Sigma| + m)$.

Heurystyka dobrego sufiksu

Dla tekstów Q i R definiujemy relację
$Q \sim R \equiv Q \sqsupset R \vee R \sqsupset Q$.
Zachodzą własnośći: $Q \sim R \equiv R \sim Q$ (*symetria* $\sim$) oraz
$Q \sqsupset R$ i $S \sqsupset R$ implikuje $Q \sim S$ (*Lemat 1*).
Heurystyka: Jeśli $P[j+1 \ldots m] = T[s+j+1 \ldots s+m]$ i
$P[j] \neq T[s+j]$, to można przesunąć P o:

$$\gamma[j] = m - \max\{k : 0 \leq k < m \wedge P[j+1 \ldots m] \sim P_k\}$$

(*najmniejsze dodatnie przesunięcie* $(m-k)$ *gwarantujące, że* $T[s+j+1 \ldots s+m] \sim P_k$.)
γ (*funkcja dobrego sufiksu*) – dobrze zdefiniowana, gdyż
$P[j+1 \ldots m] \sim P_0$.
Ponadto: Zawsze $\gamma[j] \geq 1$ (zawsze dodatnie przesunięcie w algorytmie).

funkcja dobrego sufiksu

Fakt: $\gamma[j] \leq m - \pi[m]$ (bo: $P_{\pi[m]} \sqsupset P$ oraz $P[j+1 \ldots m] \sqsupset P$, co implikuje $P_{\pi[m]} \sim P[j+1 \ldots m]$, czyli $\max\{k : 0 \leq k < m \wedge P[j+1 \ldots m] \sim P_k\} \geq \pi[m]$).

Wniosek:

$\gamma[j] = m - \max\{k : \pi[m] \leq k < m \wedge P[j+1 \ldots m] \sim P_k\}$.

Ponadto: $P_k \sqsupset P[j+1 \ldots m]$ i $k < m$ implikuje $P_k \sqsupset P_m$ i (z definicji π) $k \leq \pi[m]$.

Mamy zatem:

$$\begin{aligned} \gamma[j] &= m - \max(\{\pi[m]\} \cup \\ &\quad \{k : 0 \leq k < m \wedge P[j+1 \ldots m] \sqsupset P_k\}) \quad (**) \end{aligned}$$

(*t.j. nie warto sprawdzać* $\{k : 0 \leq k < m \wedge P_k \sqsupset P[j+1 \ldots m]\}$)

funkcja dobrego sufiksu

Niech P' – odwrócony P (*t.j.* $\forall_{1 \le i \le m}\ P'[i] = P[m-i+1]$).
Niech π' – f-cja prefiksowa dla P'.
Fakt: Jeśli $k = \max\{k : 0 \le k < m \wedge P[j+1 \ldots m] \sqsupset P_k\}$, to $\pi'[l] = m - j$, gdzie $l = (m-k) + (m-j)$
(równoważnie: $j = m - \pi'[l]$ i $k = m - l + \pi'[l]$ (!)).
Uwaga: $l \le m$ (bo $m - j \le k$, bo $P[j+1 \ldots m] \sqsupset P_k$) oraz $l \ge 1$ (bo $k < m$ i $j \le m$). Zatem $\pi'[l]$ – dobrze zdefiniowane.
D-d Faktu: $P[j+1 \ldots m]$ w P odpowiada P'_{m-j} w P'.
$P[j+1 \ldots m] \sqsupset P_k$ odpowiada
$P'_{m-j} = P'[m-k+1 \ldots m-k+(m-j)]$, czyli
$P'_{m-j} \sqsupset P'_{(m-k)+(m-j)} = P'_l$. Stąd $\pi'[l] \ge m - j$, bo $m - j < l$.
...

funkcja dobrego sufiksu

Załóżmy, że $p = \pi'[l] > m - j$. Wtedy z def. π': $P'_p \sqsupset P'_l$, czyli $P'[1 \ldots p] = P'[l - p + 1 \ldots l]$ co odpowiada:
$P[m - p + 1 \ldots m] = P[m - l + 1 \ldots m - l + p]$.
Podstawiając $l = 2m - k - j$ dostajemy:
$P[m - p + 1 \ldots m] = P[k - m + j + 1 \ldots k - m + j + p]$.
Stąd $P[m - p + 1 \ldots m] \sqsupset P_{k-m+j+p}$.
Stąd $P[j + 1 \ldots m] \sqsupset P_{k-m+j+p}$, bo
$m - p + 1 < m - (m - j) + 1 = j + 1$.
Mamy $k - m + j + p > k$, bo $p > m - j$, oraz
$k - m + j + p < k - m + j + l = k - m + j + (2m - k - j) = m$,
bo $p = \pi'[l] < l$. Niech $k' = k - m + j + p$. Zatem $k < k' < m$ i $P[j + 1 \ldots m] \sqsupset P_{k'}$ – sprzeczność z wyborem k. Stąd $\pi'[l] \leq m - j$. $\square$

funkcja dobrego sufiksu

Korzystając z (!) oraz implikacji (ćw.):
$(j = m - \pi'[l]) \;\Rightarrow\; \left(P[j+1 \ldots m] \sqsupset P_{m-l+\pi'[l]}\right)$ (!!)
mamy:

$$
\begin{aligned}
\gamma[j] &= m - \max(\{\pi[m]\} \cup \\
&\quad \{k : 0 \le k < m \wedge P[j+1 \ldots m] \sqsupset P_k\}) \text{ (**)} \\
&= m - \max(\{\pi[m]\} \cup \\
&\quad \{m - l + \pi'[l] : 1 \le l \le m \wedge j = m - \pi'[l]\}) \text{ z (!) i (!!)} \\
&= \min(\{m - \pi[m]\} \cup \\
&\quad \{l - \pi'[l] : 1 \le l \le m \wedge j = m - \pi'[l]\})
\end{aligned}
$$

Good-Suffix-Function

```
Good-Suffix-Function(P,m)
 1 π ←Compute-Prefix-Function(P)
 2 P′ ←odwrócone P
 3 π′ ←Compute-Prefix-Function(P′)
 4 for j ← 0 to m do
 5    γ[j] ← m − π[m]
 6 for l ← 1 to m do
 7    j ← m − π′[l]
 8    if γ[j] > l − π′[l] then
 9       γ[j] ← l − π′[l]
10 return γ
```

Czas: $O(m)$.

Czas algorytmu Boyera-Moore'a: $O((n-m+1)m+|\Sigma|)$, jednak w praktyce dużo przesunięć może być pominiętych.