Algorytmy dopasowywania wzroców do tekstu

dr inż. Aleksander Smywiński-Pohl

Plan

- algorytm naiwny
- automat skończony
- algorytm Knutha-Morrisa-Pratta

Oznaczenia

Oznaczenia

- $T[1..n]$ - tekst o długości n, tablica znaków, łańcuch znaków

Oznaczenia

- $T[1..n]$ - tekst o długości n, tablica znaków, łańcuch znaków
- $P[1..m]$ - wzorzec o długości m, $m \leq n$

Oznaczenia

- $T[1..n]$ - tekst o długości n, tablica znaków, łańcuch znaków
- $P[1..m]$ - wzorzec o długości m, $m \leq n$
- Σ - alfabet, zbiór znaków tekstu

Oznaczenia

- $T[1..n]$ - tekst o długości n, tablica znaków, łańcuch znaków
- $P[1..m]$ - wzorzec o długości m, $m \leq n$
- Σ - alfabet, zbiór znaków tekstu
- Σ^* - zbiór wszystkich łańcuchów skończonej długości nad alfabetem Σ

Oznaczenia

- $T[1..n]$ - tekst o długości n, tablica znaków, łańcuch znaków
- $P[1..m]$ - wzorzec o długości m, $m \leq n$
- Σ - alfabet, zbiór znaków tekstu
- Σ^* - zbiór wszystkich łańcuchów skończonej długości nad alfabetem Σ
- ϵ - łańcuch pusty

Oznaczenia

- $T[1..n]$ - tekst o długości n, tablica znaków, łańcuch znaków
- $P[1..m]$ - wzorzec o długości m, $m \leq n$
- Σ - alfabet, zbiór znaków tekstu
- Σ^* - zbiór wszystkich łańcuchów skończonej długości nad alfabetem Σ
- ϵ - łańcuch pusty
- $|x|$ - długość łańcucha x

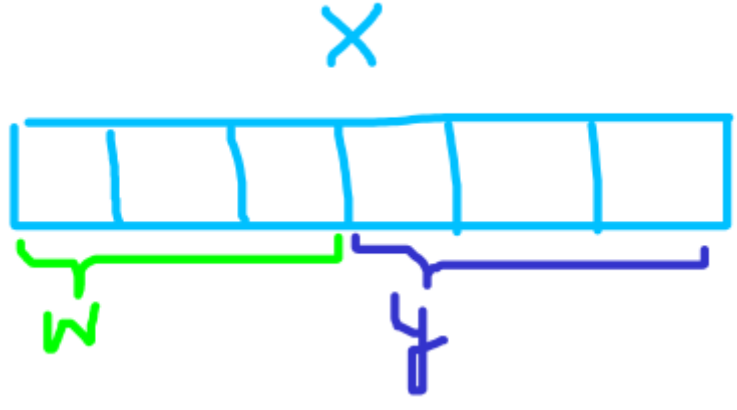
Oznaczenia

- $T[1..n]$ - tekst o długości n, tablica znaków, łańcuch znaków
- $P[1..m]$ - wzorzec o długości m, $m \leq n$
- Σ - alfabet, zbiór znaków tekstu
- Σ^* - zbiór wszystkich łańcuchów skończonej długości nad alfabetem Σ
- ϵ - łańcuch pusty
- $|x|$ - długość łańcucha x
- xy - konkatenacja łańcuchów x i y

Definicje

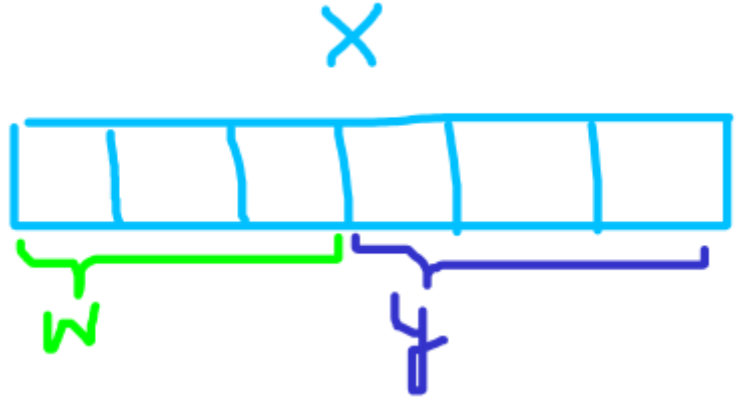
Definicje

- $w \sqsubset x$ - w jest **prefiksem** łańcucha $x \equiv \exists y \in \Sigma^* : x = wy$
  - $|w| \leq |x|$

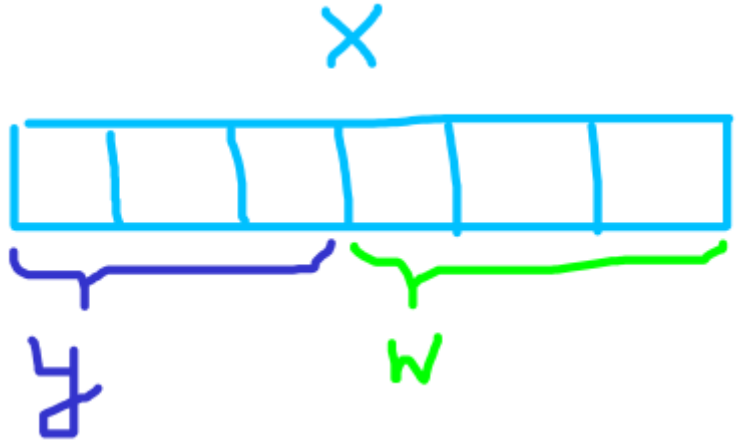


Definicje

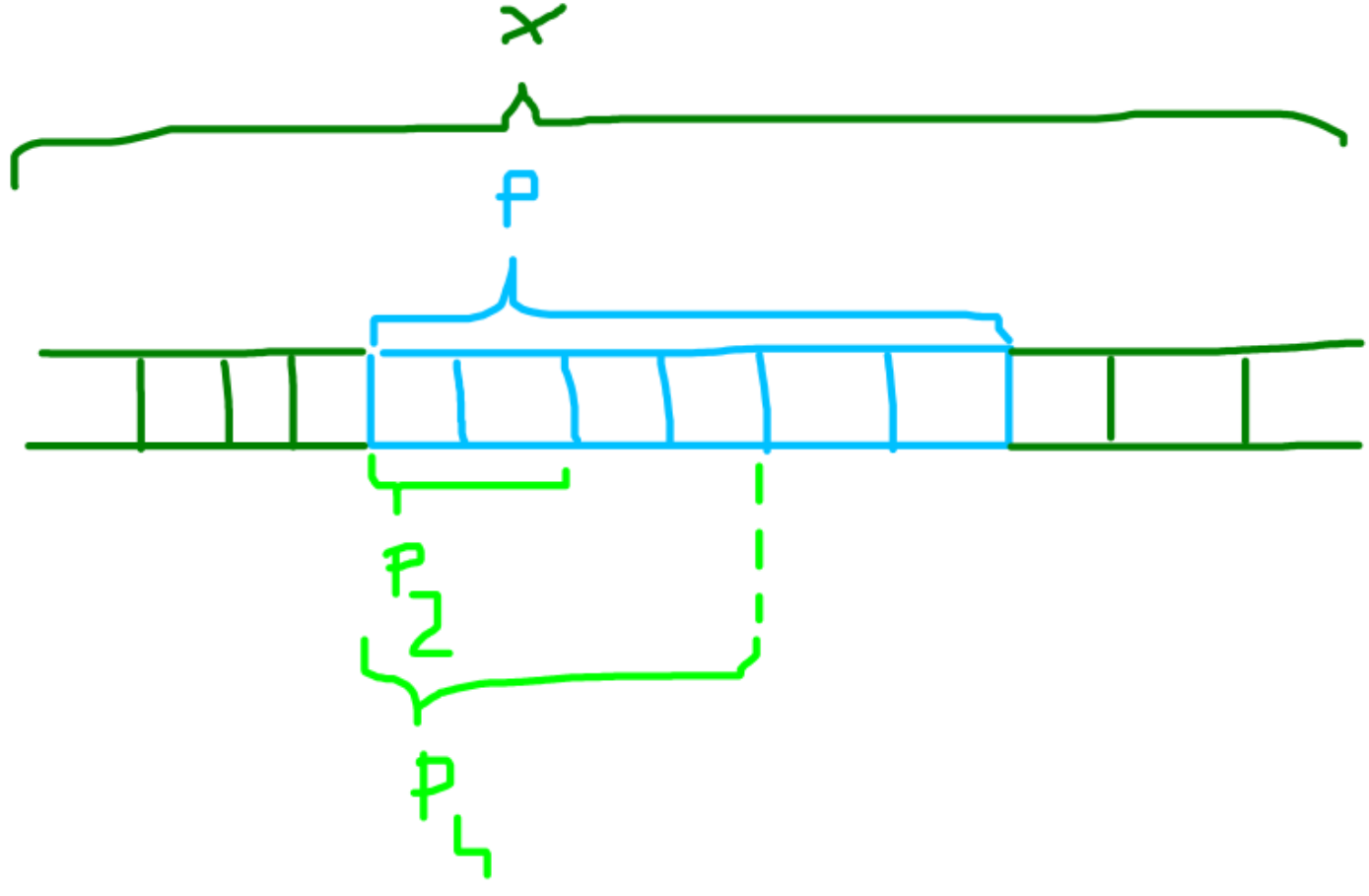
- $w \sqsubset x$ - w jest **prefiksem** łańcucha $x \equiv \exists y \in \Sigma^* : x = wy$
  - $|w| \leq |x|$



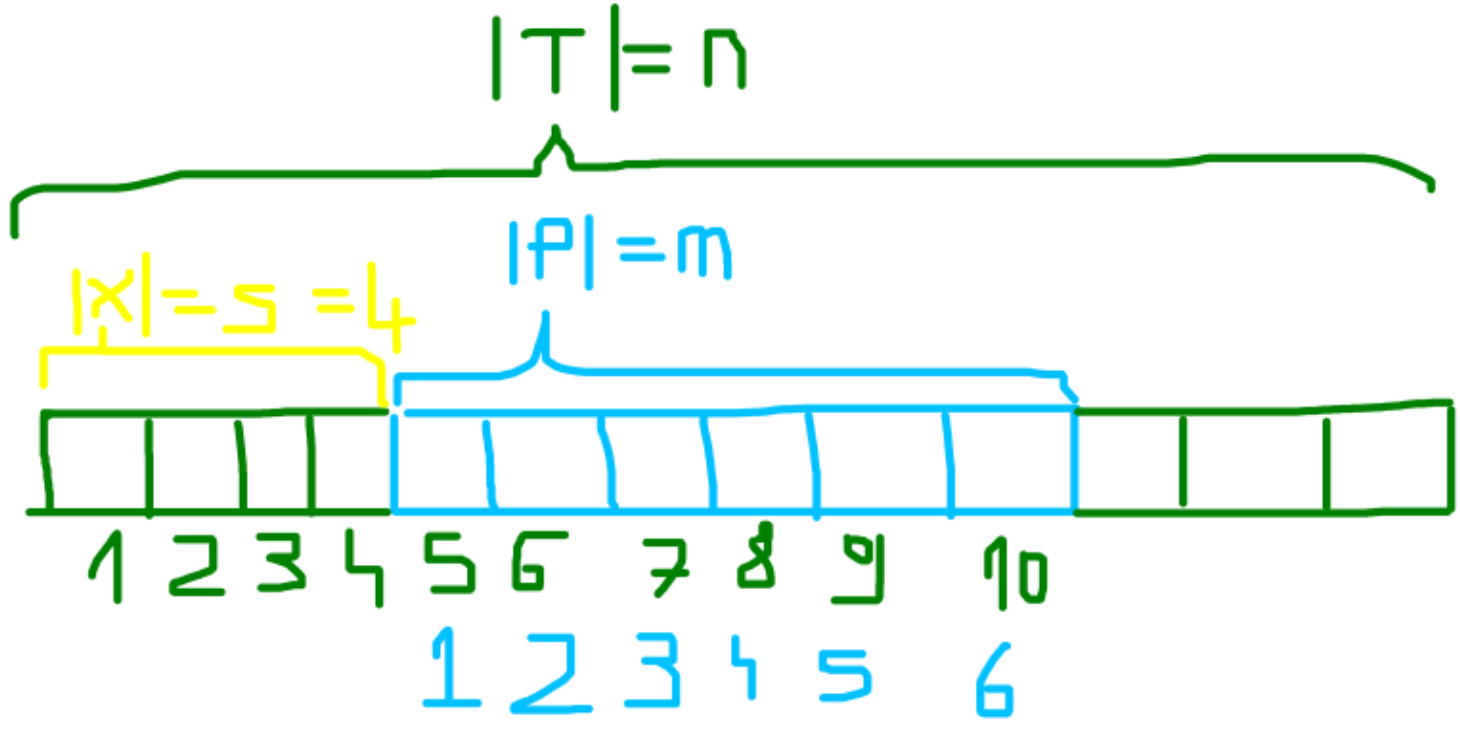
- $w \sqsupset x$ - w jest **sufiksem** łańcucha $x \equiv \exists y \in \Sigma^* : x = yw$
  - $|w| \leq |x|$


x
y
w

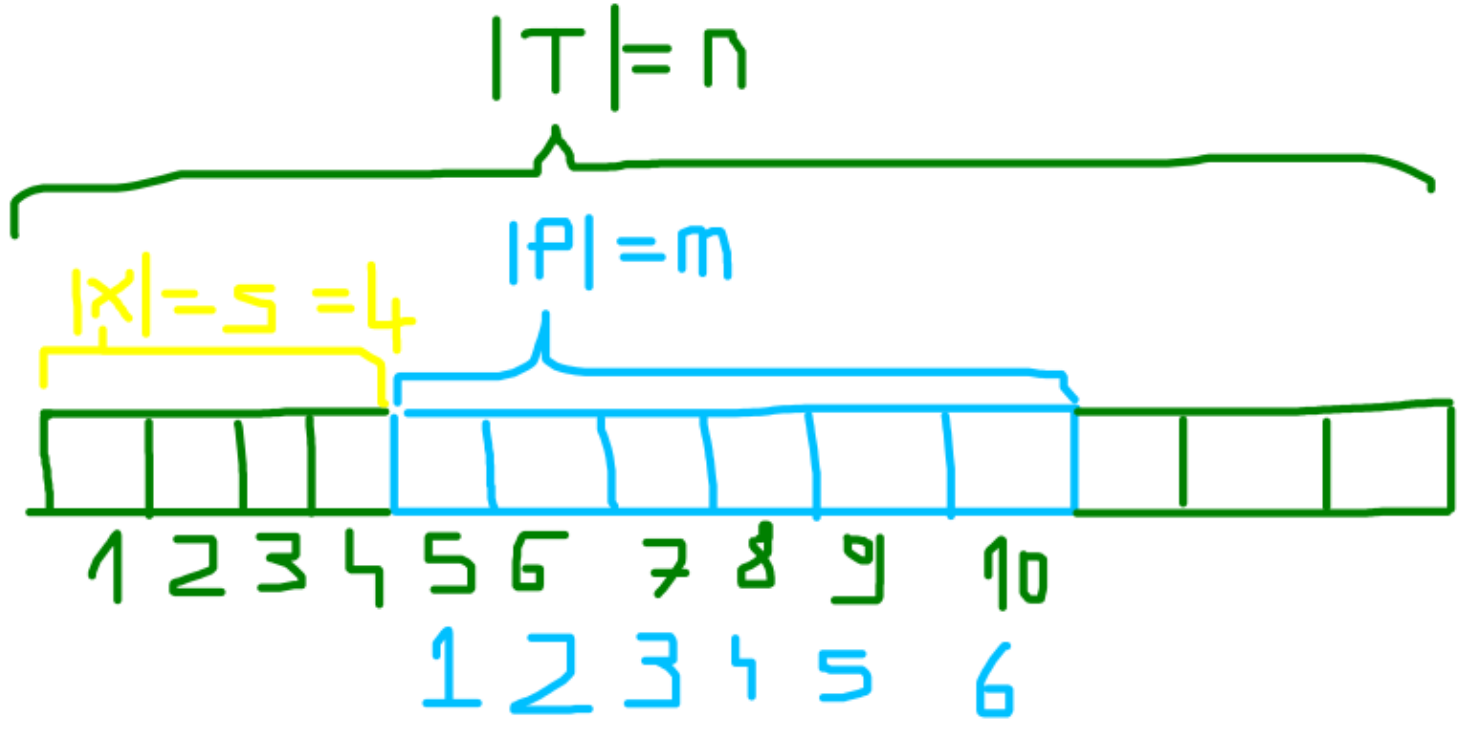
- P_k - prefiks długości k wzorca/tekstu P



- **wzorzec P o długości m występuje w tekście T o długości n z przesunięciem s:**
  - $0 \leq s \leq n - m \wedge T[s+1..s+m] = P[1..m]$

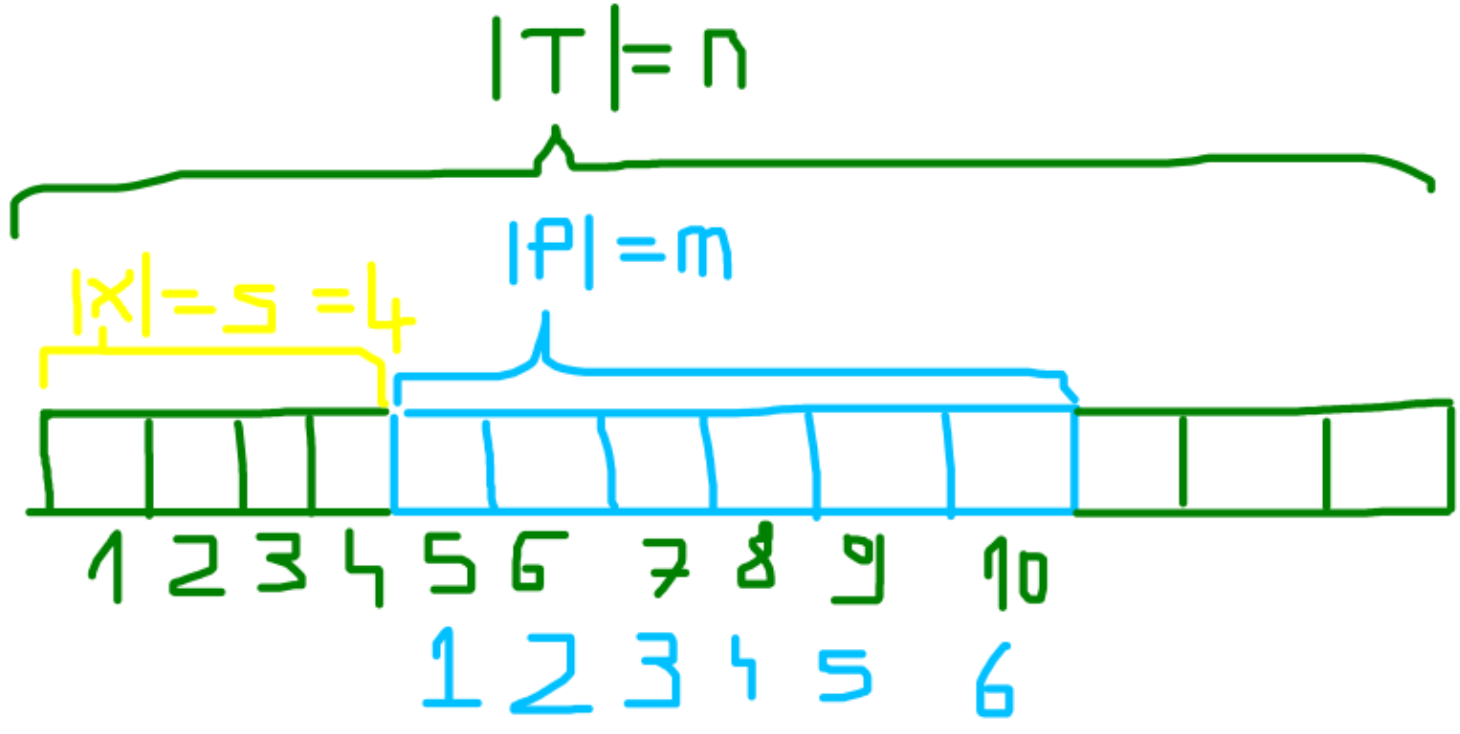


- **wzorzec P o długości m występuje w tekście T o długości n z przesunięciem s:**
  - $0 \leq s \leq n - m \wedge T[s+1..s+m] = P[1..m]$



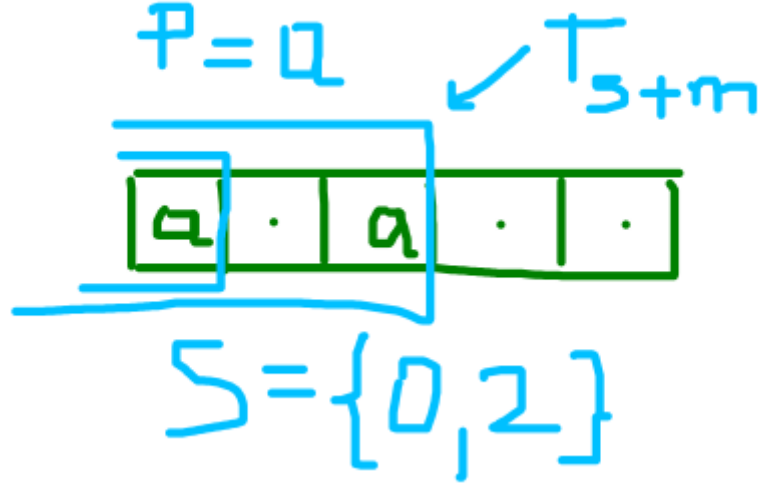
- **poprawne przesunięcie** s - wzorzec P występuje z przesunięciem s

- **wzorzec P o długości m występuje w tekście T o długości n z przesunięciem s:**
  - $0 \leq s \leq n - m \wedge T[s+1..s+m] = P[1..m]$



- **poprawne przesunięcie s** - wzorzec P występuje z przesunięciem s
- **niepoprawne przesunięcie s** - wzorczec P nie występuje z przesunięciem s

- **dopasowanie wzroca P do tekstu T**
  - znalezienie wszystkich poprawnych przesunięć s wzroca P w tekście T
  - znalezienie wszystkich przesunięć s, dla których $P \sqsupset T_{s+m}$



Złożoność obliczeniowa algorytmów dopasowywania wzorców

Algorytm	Czas preprocessingu	Czas dopasowywania	
naiwny	0	$O((n-m+1)m)$	
automat skończony	$O(m	\Sigma	)^*$ $\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$	

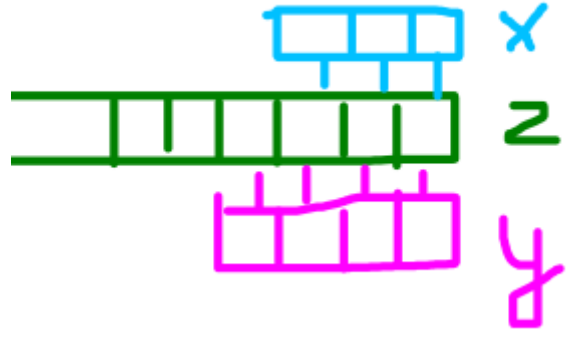
* algorytm prezentowany na wykładzie ma złożoność $O(m^3|\Sigma|)$

Lemat 1.1 (o zawieraniu sufiksów)

x, y, z - łańcuchy znaków

$x \sqsupset z \wedge y \sqsupset z \rightarrow$

- $|x| \leq |y| \rightarrow x \sqsupset y$
- $|x| \geq |y| \rightarrow y \sqsupset x$
- $|x| = |y| \rightarrow x = y$



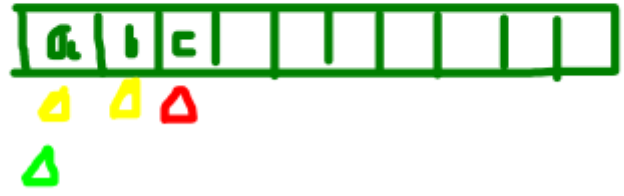
Algorytm naiwny

Algorytm naiwny

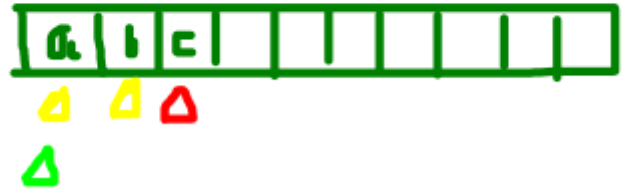
In [ ]:

```
def naive_string_matching(text, pattern):
    for s in range(0, len(text) - len(pattern) + 1):
        if(pattern == text[s:s+len(pattern)]):
            print(f"Przesunięcie {s} jest poprawne")
```

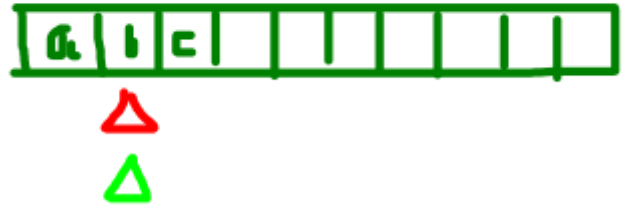
P = a b d

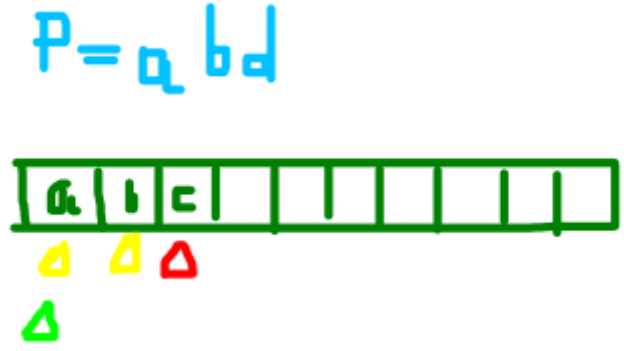


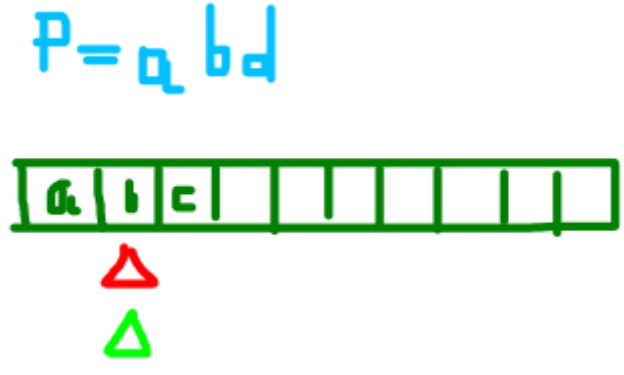
P = a b d



P = a b d

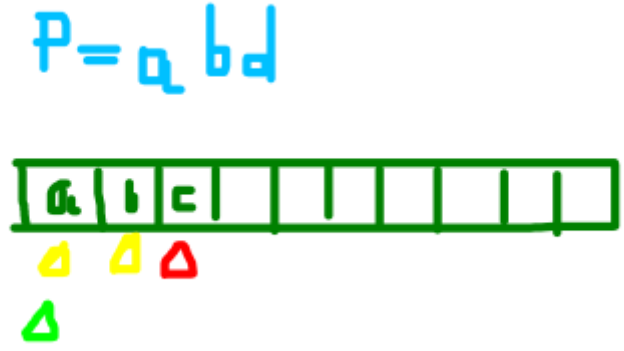


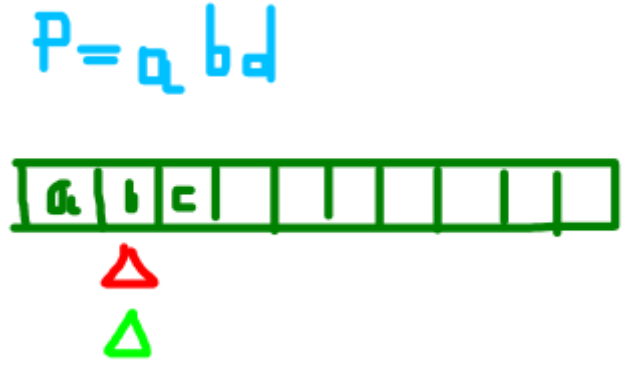




In [ ]:

```
naive_string_matching("abdabaaaabd", "aa")
```





In [ ]:

```
naive_string_matching("abdabaaaabd", "aa")
```

Złożoność czasowa dopasowania $O((n-m+1)m)$

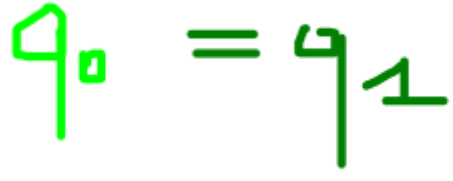
Automat skończony - definicja

Automat skończony M to krotka $(Q, q_0, A, \Sigma, \delta)$:

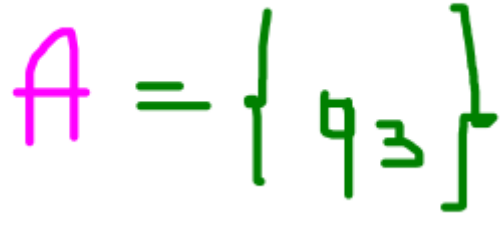
- Q - zbiór stanów

$$Q = \{q_1, q_2, q_3\}$$

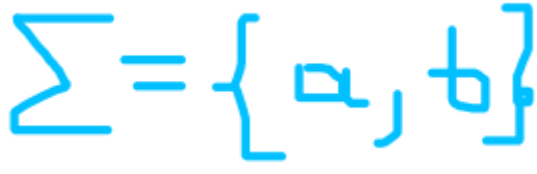
- $q_0 \in Q$ - stan początkowy



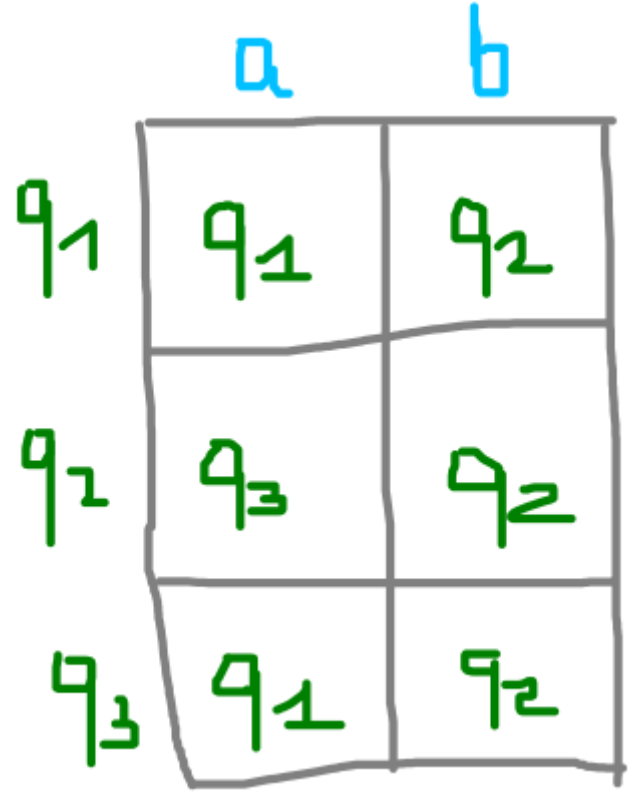
- $A \subset Q$ - zbiór stanów akceptujących

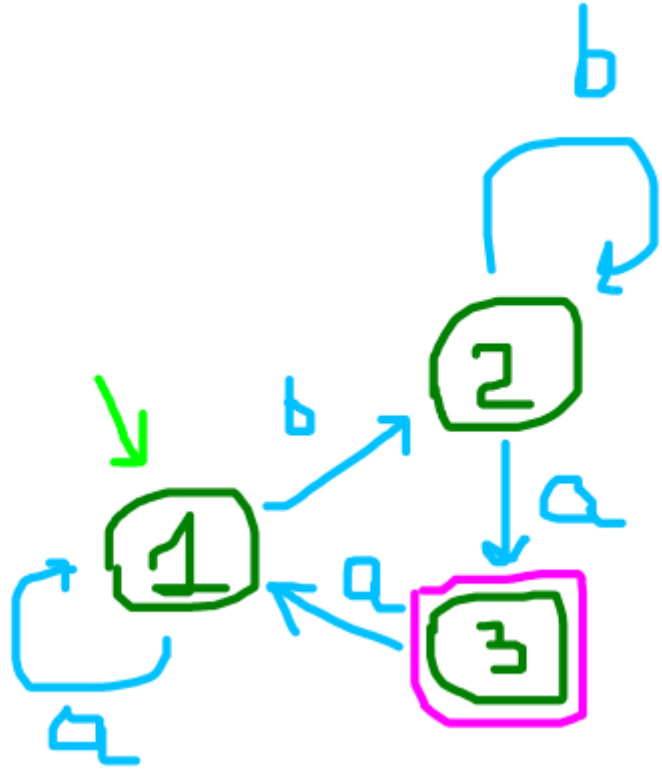


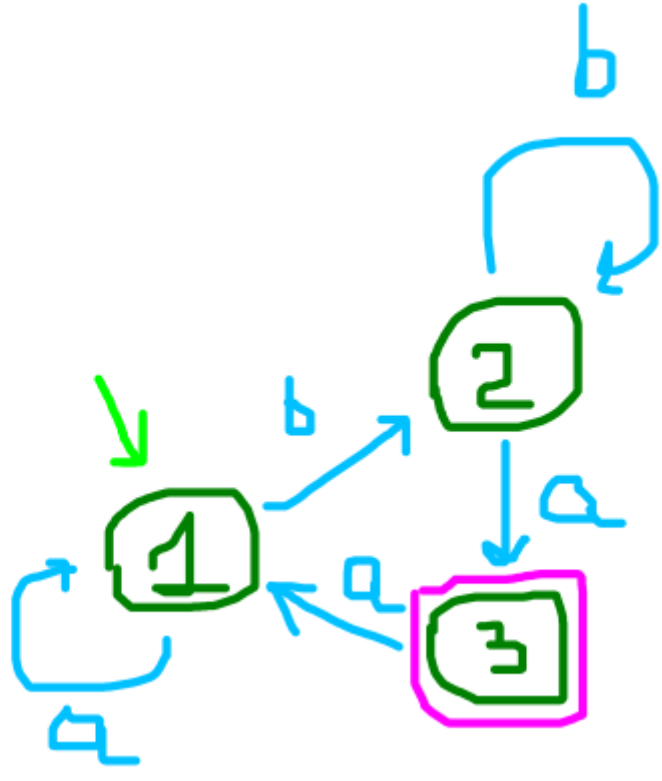
- Σ - alfabet, skończony zbiór znaków



- $\delta : Q \times \Sigma \to Q$ - funkcja przejścia

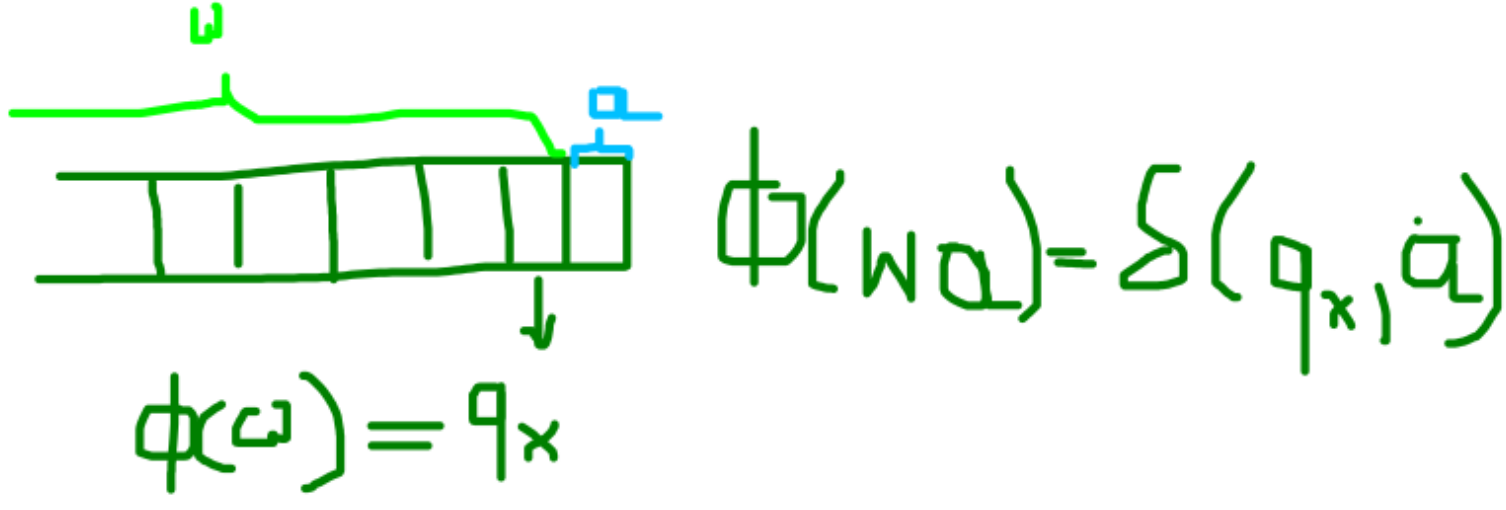



b
2
b
a
1
a
3
a



- automat **akceptuje** łańcuch $\equiv$ stan po przeczytaniu łańcucha $q \in A$
- w przeciwnym razie automat **odrzuca** łańcuch

- automat indukuje funkcję $\phi(w)$ zwaną **funkcją stanu końcowego**
- $\phi(w) : \Sigma^* \to Q$
    - $\phi(\epsilon) = q_0$
    - $\phi(wa) = \delta(\phi(w), a)$ dla $w \in \Sigma^*, a \in \Sigma$

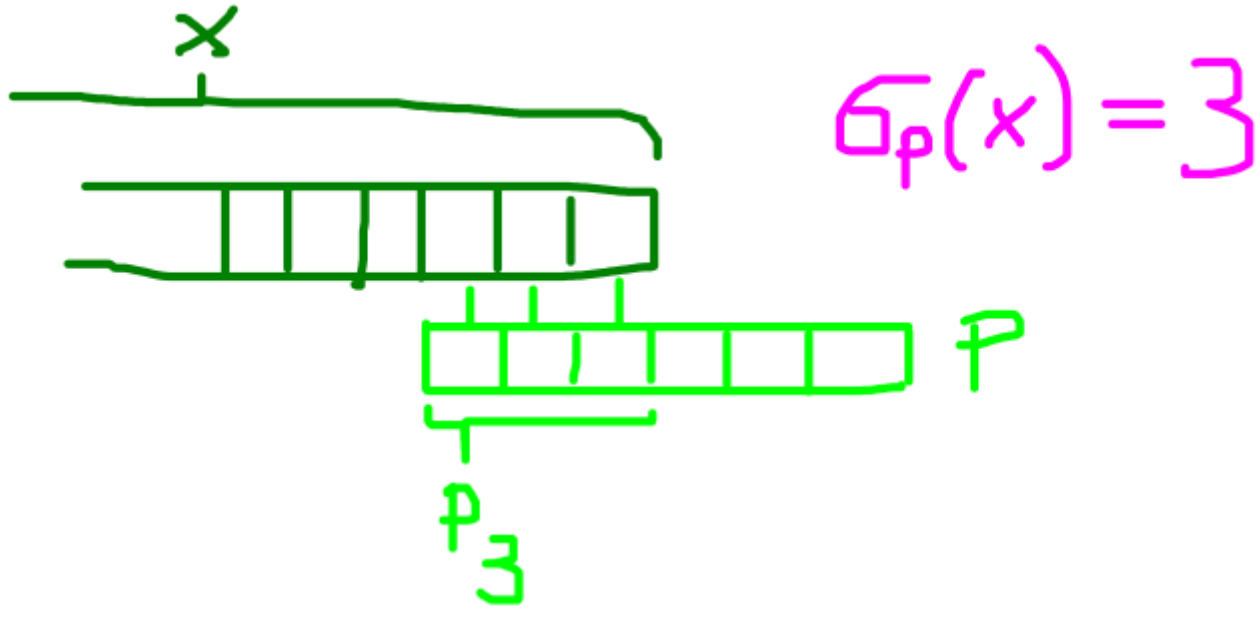


Funkcja sufiksowa σ_P odpowiadająca wzorcowi P

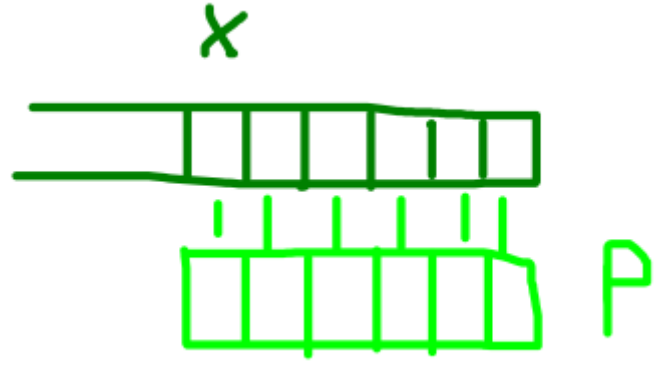
$|P| = m$

$\sigma_P : \Sigma^* \to \{0, 1, \dots m\}$

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

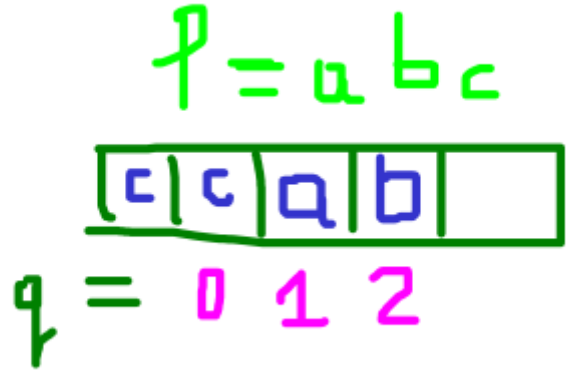

x
$\sigma_p(x) = 3$
p
p_3

- $P_0 = \epsilon$ jest sufiksem każdego łańcucha, co zapewni o poprawność definicji funkcji
- $|P| = m \rightarrow (\sigma_P(x) = m \leftrightarrow P \sqsupset x)$



Automat akceptujący teksty pasujące do wzorca $P[1..m]$

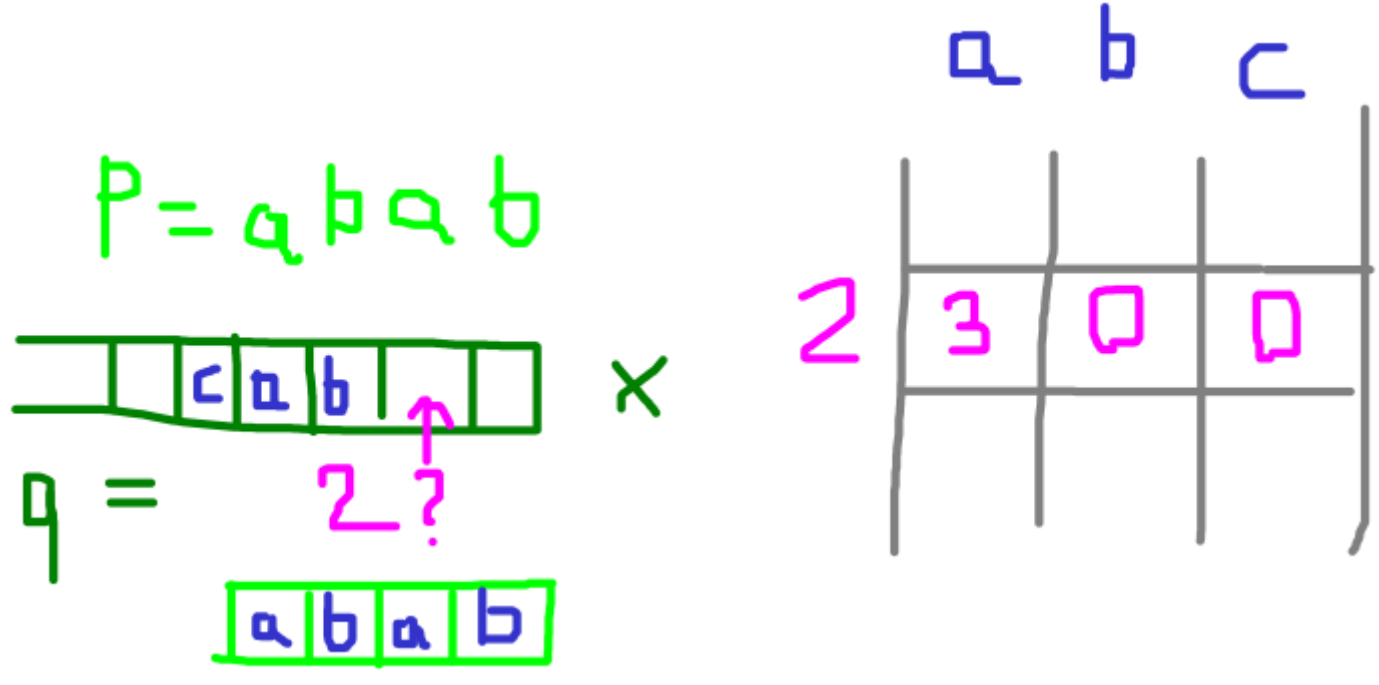
- $Q = \{0, 1, \ldots, m\}$
- $q_0 = 0$
- $A = \{m\}$ - jedyny stan akceptujący
- $\boldsymbol{\delta(q, a) = \sigma_P(P_q a)}$


P = a b c
c c a b
q = 0 1 2

Jak skonstruować funkcję δ? Kluczowa obserwacja:

$\boldsymbol{\sigma_P(T_i a) = \sigma_P(P_q a)}$, gdzie $q = \sigma_P(T_i)$

Innymi słowy - można to zrobić analizująca sam wzorzec P.



Algorytm automatu skończonego

Algorytm automatu skończonego

In [15]:

```python
def fa_string_matching(text, delta):
    q = 0
    length = len(delta) - 1
    for i in range(0, len(text)):
        q = delta[q][text[i]]
        if(q == length):
            print(f"Przesunięcie {i + 1 - q } jest poprawne")
            # ponieważ przeczytaliśmy (s+q)-ty znak tekstu, ale w Pythonie ma on indeks o jeden mniejszy,
            # dlatego wartośc musimy zwiększyć o 1
```

Algorytm automatu skończonego

In [15]:

```python
def fa_string_matching(text, delta):
    q = 0
    length = len(delta) - 1
    for i in range(0, len(text)):
        q = delta[q][text[i]]
        if(q == length):
            print(f"Przesunięcie {i + 1 - q } jest poprawne")
            # ponieważ przeczytaliśmy (s+q)-ty znak tekstu, ale w Pythonie ma on indeks o jeden mniejszy,
            # dlatego wartośc musimy zwiększyć o 1
```

Złożoność czasowa dopasowania $\Theta(n)$

Funkcja przejścia

Funkcja przejścia

In [16]:

```
pattern = "aba"

delta = [
    {"a": 1, "b": 0}, # 0
    {"a": 1, "b": 2}, # 1
    {"a": 3, "b": 0}, # 2
    {"a": 1, "b": 2}, # 3
]
```

Funkcja przejścia

In [16]:

```
pattern = "aba"

delta = [
    {"a": 1, "b": 0}, # 0
    {"a": 1, "b": 2}, # 1
    {"a": 3, "b": 0}, # 2
    {"a": 1, "b": 2}, # 3
]
```

In [17]:

```
fa_string_matching("abaabaaaaba", delta)
```

```
Przesunięcie 0 jest poprawne
Przesunięcie 3 jest poprawne
Przesunięcie 8 jest poprawne
```

Poprawność działania automatu skończonego

- lemat o nierówności funkcji sufiksowej σ
- lemat o rekursywności funkcji sufiksowej σ
- twierdzenie o porawności automatu skończonego

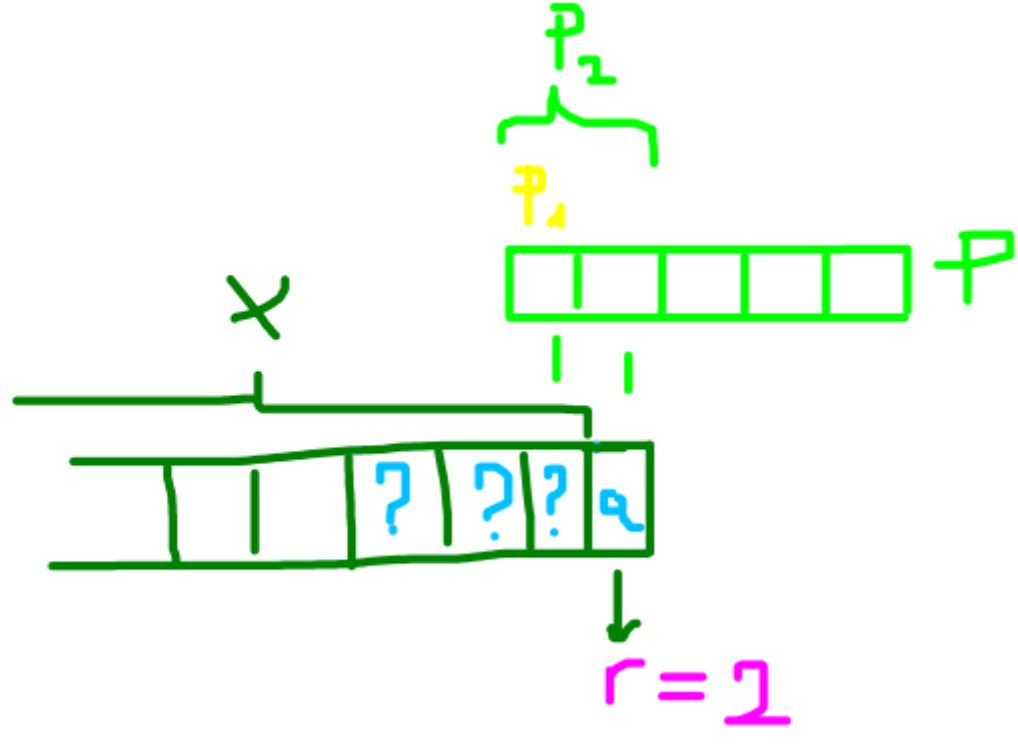
Lemat 1.2 (o nierówności funkcji sufiksowej)

Przyjmując:

- x - łańcuch znaków
- a - znak

$\forall \boldsymbol{x, a : \sigma_P(xa) \leq \sigma_P(x) + 1}$

1. Niech $r = \sigma_P(xa)$
2. Jeśli $r = 0$, wtedy warunek nierówności jest spełniony, ponieważ σ_P jest nieujemna.
3. Jeśli $r > 0$:
    A. $P_r \sqsupset xa$
    B. $P_{r-1} \sqsupset x$
    C. $\boldsymbol{r - 1 \leq \sigma_P(x)}$
    D. $\sigma_P(xa) = r \leq \sigma_P(x) + 1$

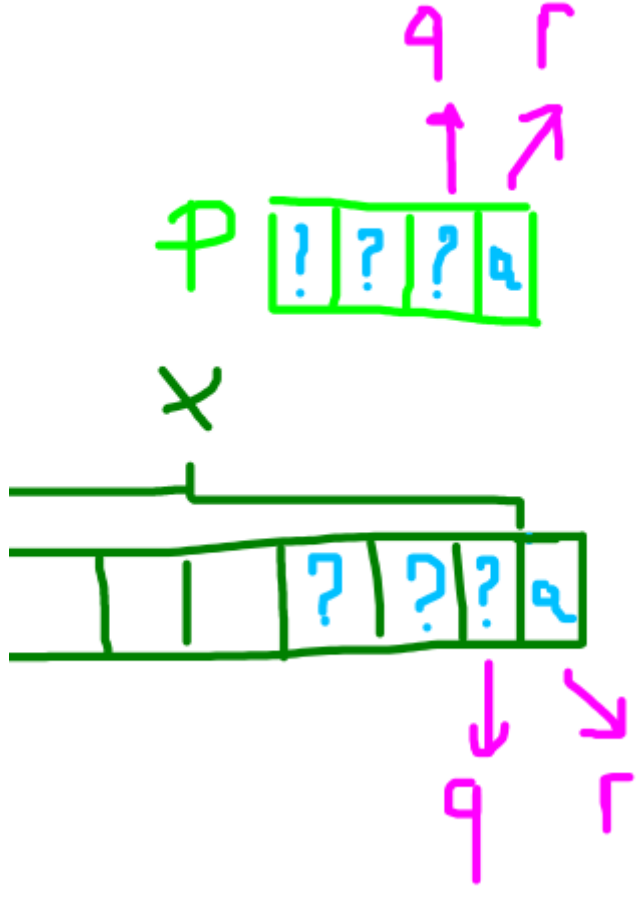


Lemat 1.3 (o rekursywności funkcji sufiksowej)

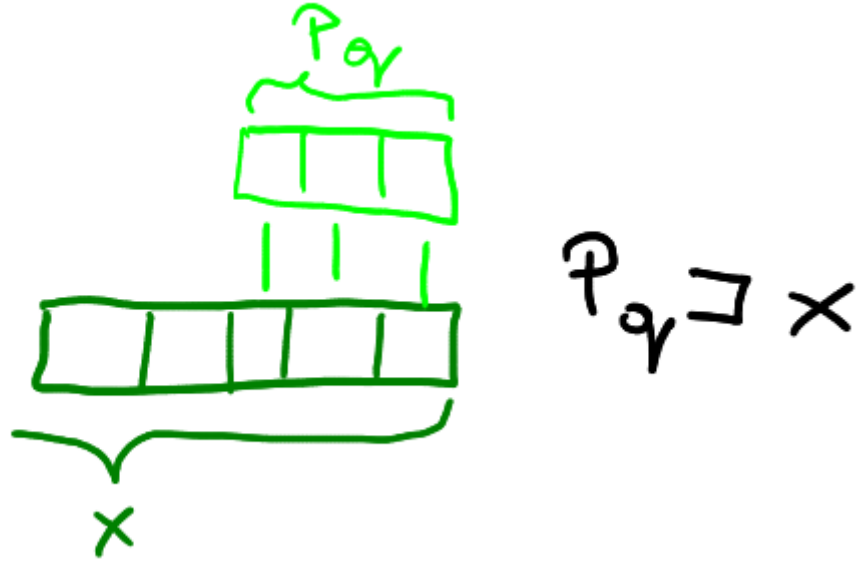
Przyjmując:

- x - łańcuch znaków
- a - znak

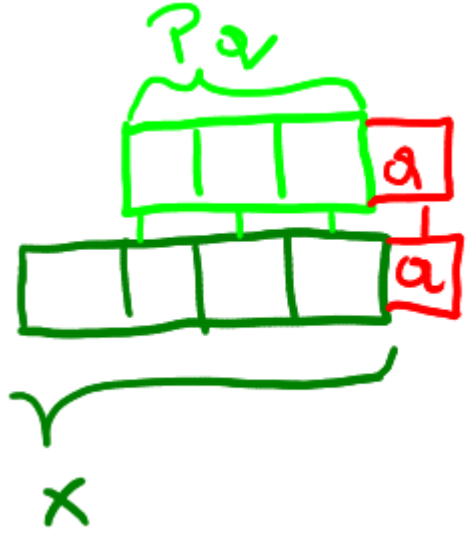
$$\forall \boldsymbol{x, a : q = \sigma_P(x) \rightarrow \sigma_P(xa) = \sigma_P(P_q a)}$$


q
r
p
?
?
?
q
x
?
?
?
q
q
r

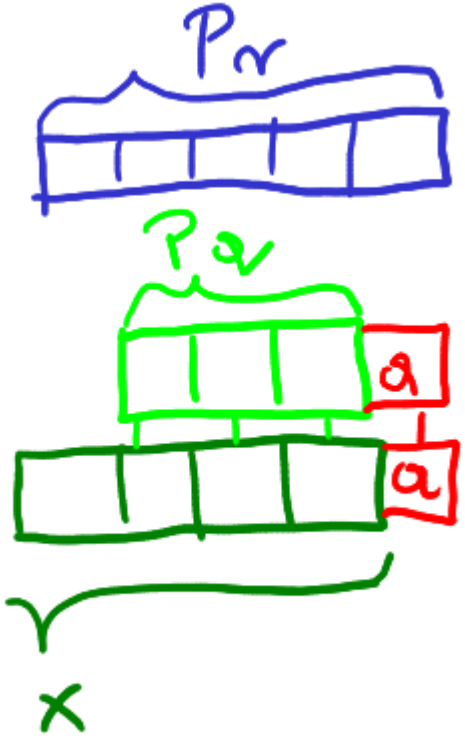
1. $P_q \sqsupset x$ z definicji σ



1. $P_q a \sqsupset xa$ poprzez dodanie znaku a na końcu prefiksu wzorca oraz na końcu tekstu



1. niech $r = \sigma_P(xa)$, wtedy:
   A. P_r
      $\sqsupset xa$

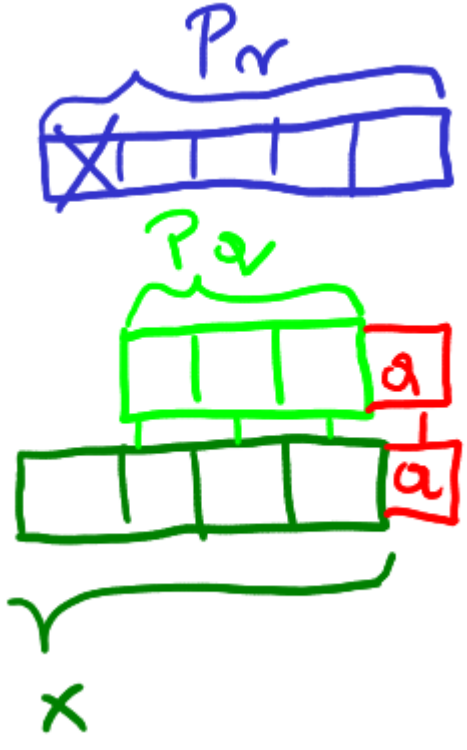


$P_q \; a \; \exists \; x \; a$

$r = \sigma_p(x a)$

1. niech $r = \sigma_P(xa)$, wtedy:

   A. $r \leq q + 1$ z lematu 1.2

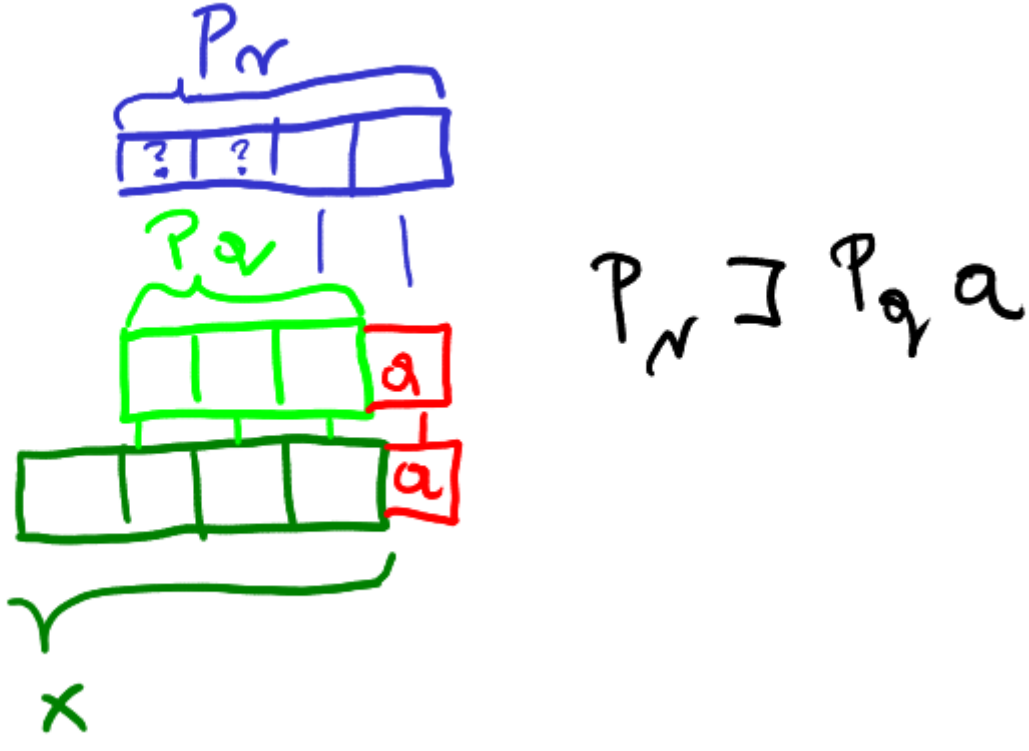


$r \leq q + 1$

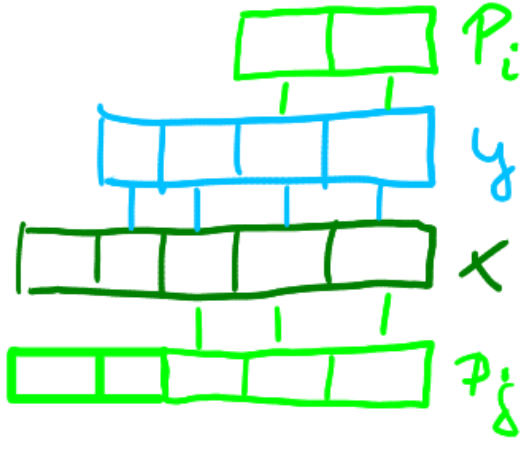
1. niech $r = \sigma_P(xa)$, wtedy:
    A. $|P_r|$
    $= r$
    $\leq q$
    $+ 1$
    $=$
    $|P_q a$
    $|$

1. niech $r = \sigma_P(xa)$, wtedy:

   A. $\boldsymbol{P_r \sqsupset P_q a}$ (z $P_q a \sqsupset xa$, $P_r \sqsupset xa$ oraz $|P_r| \leq |P_q a|$ i lematu 1.1)



1. niech $r = \sigma_P(xa)$, wtedy:
   A. σ_P
      (P_r)
      $\leq \sigma_P$
      $(P_q a)$



$$y \sqsupset x \rightarrow \underbrace{\sigma_P(y)}_{=i} \leq \underbrace{\sigma_P(x)}_{=j}$$

1. niech $r = \sigma_P(xa)$, wtedy:
    A. $r \leq \sigma_P (P_q a)$
    B. $\sigma_P(xa) \leq \sigma_P (P_q a)$
    C. $\sigma_P(P_q a) \leq \sigma_P (xa)$ ponieważ $P_q a \sqsupset xa$
    D. $\sigma_P(xa) = \sigma_P (P_q a)$

Twierdzenie 1.4 (o poprawności automatu skończonego)

Niech:

- ϕ_P - funkcja stanu końcowego dla wzroca P,
- $T[1..n]$ - tekst dopasowywany do wzorca P,

wtedy

$$\boldsymbol{\phi_P(T_i) = \sigma_P(T_i)}$$

Dowód indukcyjny.

dla $i = 0$, twierdzenie jest prawdziwe, ponieważ $T_0 = \epsilon$, więc $\phi_P(T_0) = 0 = \sigma_P(T_0)$,

Dowód indukcyjny.

dla $i = 0$, twierdzenie jest prawdziwe, ponieważ $T_0 = \epsilon$, więc $\phi_P(T_0) = 0 = \sigma_P(T_0)$,

Załóżmy, że $\phi_P(T_i) = \sigma_P(T_i)$ i sprawdźmy, czy $\phi_P(T_{i+1}) = \sigma_P(T_{i+1})$.

Dowód indukcyjny.

dla $i = 0$, twierdzenie jest prawdziwe, ponieważ $T_0 = \epsilon$, więc $\phi_P(T_0) = 0 = \sigma_P(T_0)$,

Załóżmy, że $\phi_P(T_i) = \sigma_P(T_i)$ i sprawdźmy, czy $\phi_P(T_{i+1}) = \sigma_P(T_{i+1})$.

Niech $q \equiv \phi_P(T_i)$ oraz $a \equiv T[i+1]$.

Dowód indukcyjny.

dla $i = 0$, twierdzenie jest prawdziwe, ponieważ $T_0 = \epsilon$, więc $\phi_P(T_0) = 0 = \sigma_P(T_0)$,

Załóżmy, że $\phi_P(T_i) = \sigma_P(T_i)$ i sprawdźmy, czy $\phi_P(T_{i+1}) = \sigma_P(T_{i+1})$.

Niech $q \equiv \phi_P(T_i)$ oraz $a \equiv T[i+1]$.

$$
\begin{array}{rcll}
\phi_P(T_{i+1}) & = & \phi_P(T_i a) & \text{z definicji } T_{i+1} \text{ oraz } a \\
& = & \delta(\phi_P(T_i), a) & \text{z definicji } \phi \\
& = & \delta(q, a) & \text{z defnicji } q \\
& = & \sigma_P(P_q a) & \text{z definicji } \delta \\
& = & \sigma_P(T_i a) & \text{z lematu 1.3 oraz indukcji} \\
& = & \sigma_P(T_{i+1}) & \text{z definicji } T_{i+1}
\end{array}
$$

Algorytm generujący funkcję przejścia

Algorytm generujący funkcję przejścia

In [21]:

```
def transition_table(pattern):
    result = []
    for q in range(0, len(pattern) + 1):
        result.append({})
        for a in ["a", "b"]:
            k = min(len(pattern) + 1, q + 2)
            while True:
                k = k - 1
                # x[:k] - prefiks o długości k
                # x[-k:] - sufiks o długości k
                if(k == 0 or pattern[:k] == (pattern[:q] + a)[-k:]):
                    break
            result[q][a] = k
    return result
```

Algorytm generujący funkcję przejścia

In [21]:

```python
def transition_table(pattern):
    result = []
    for q in range(0, len(pattern) + 1):
        result.append({})
        for a in ["a", "b"]:
            k = min(len(pattern) + 1, q + 2)
            while True:
                k = k - 1
                # x[:k] - prefiks o długości k
                # x[-k:] - sufiks o długości k
                if(k == 0 or pattern[:k] == (pattern[:q] + a)[-k:]):
                    break
            result[q][a] = k
    return result
```

In [22]:

```python
transition_table("aba")
```

Out[22]:

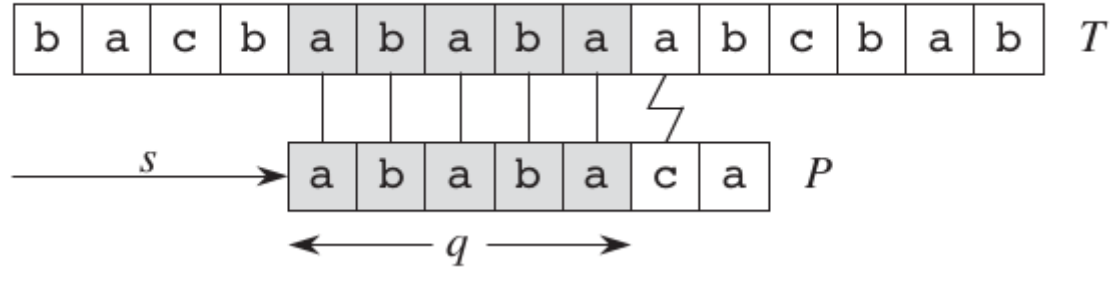
```
[{'a': 1, 'b': 0}, {'a': 1, 'b': 2}, {'a': 3, 'b': 0},
{'a': 1, 'b': 2}]
```

Złożoność czasowa algorytmu generującego funkcję przejścia

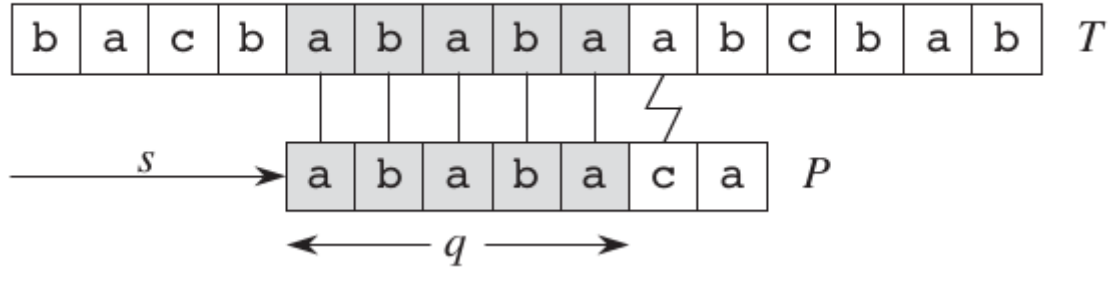
$O(m^3|\Sigma|)$

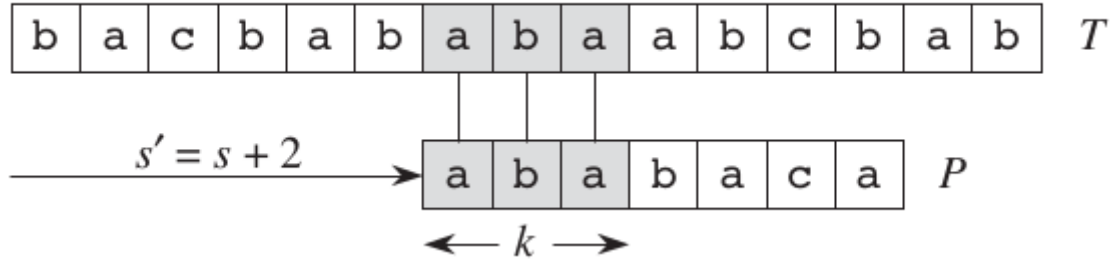
Algorytm Knutha-Morrisa-Pratta

Algorytm Knutha-Morrisa-Pratta

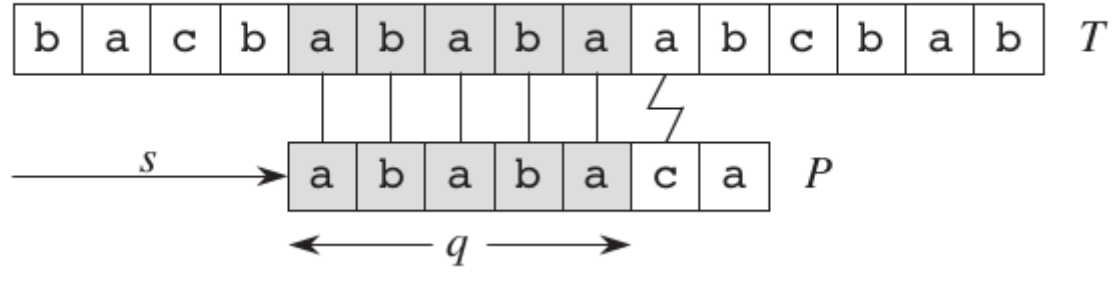


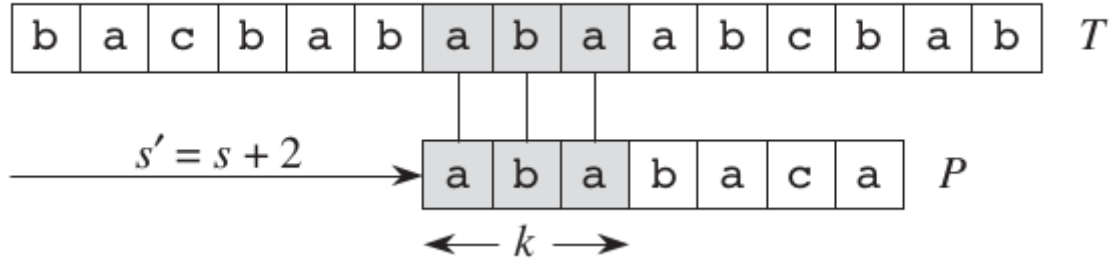
Algorytm Knutha-Morrisa-Pratta

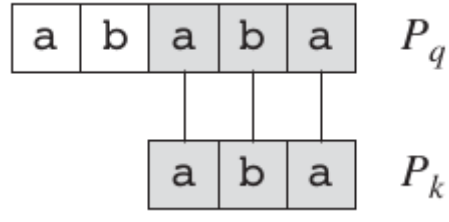



b a c b a b a b a a b c b a b
T
s′ = s + 2
a b a b a c a
P
k

Algorytm Knutha-Morrisa-Pratta




b a c b a b a b a a b c b a b
T
s′ = s + 2
a b a b a c a
P
k


a b a b a
P_q
a b a
P_k

Funkcja prefiksowa

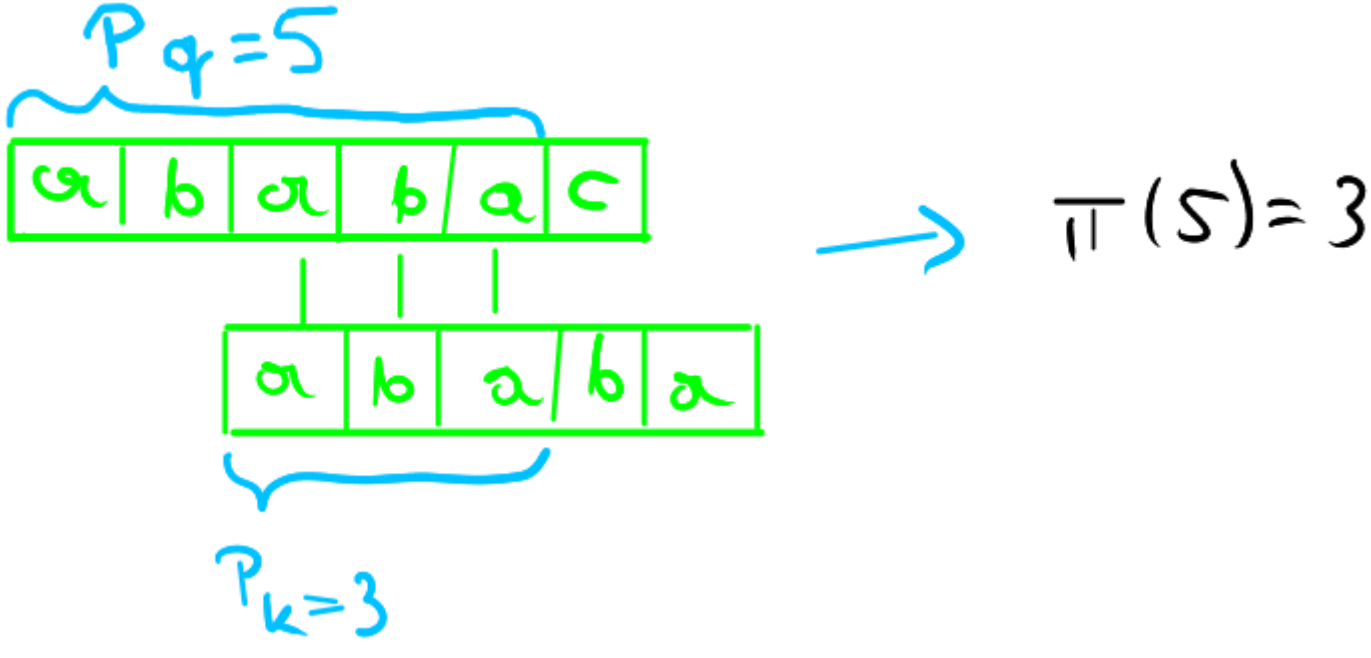
$$\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$$

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

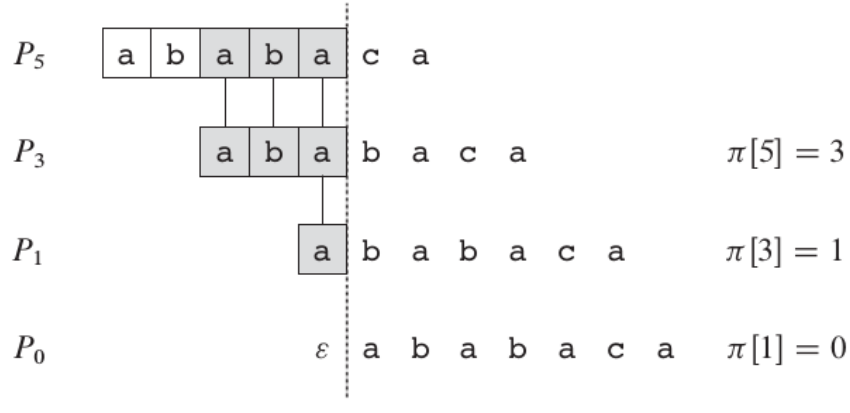
Funkcja prefiksowa

$$\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$$

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

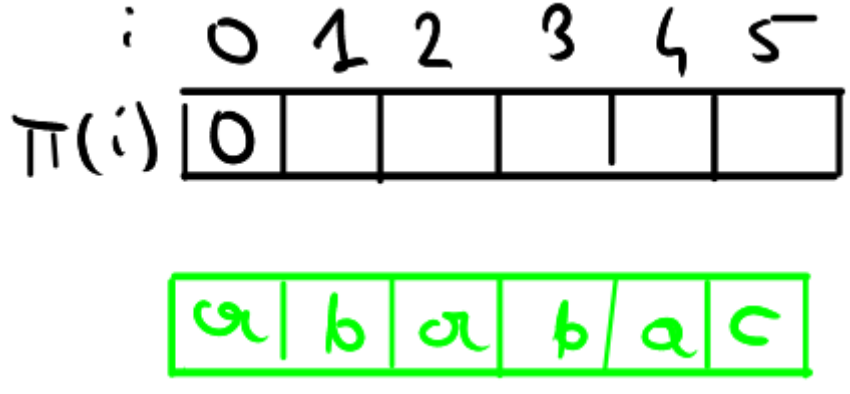


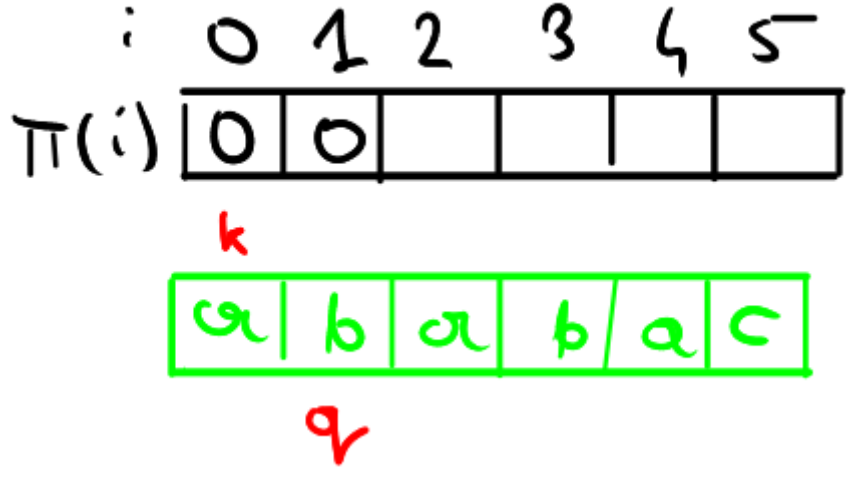
i	1	2	3	4	5	6	7
$P[i]$	a	b	a	b	a	c	a
$\pi[i]$	0	0	1	2	3	0	1

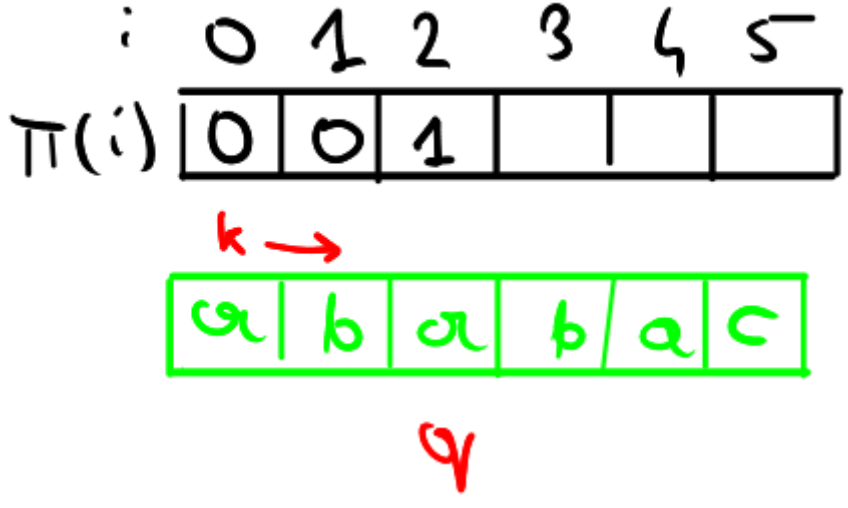


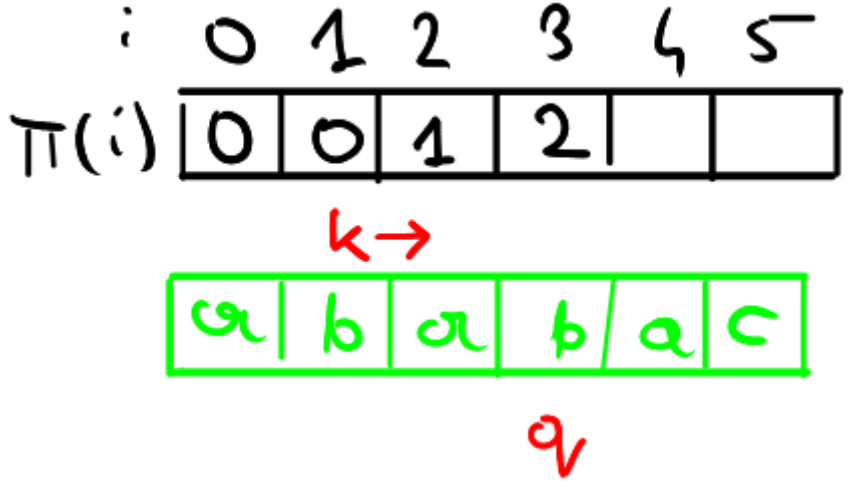
In [23]:

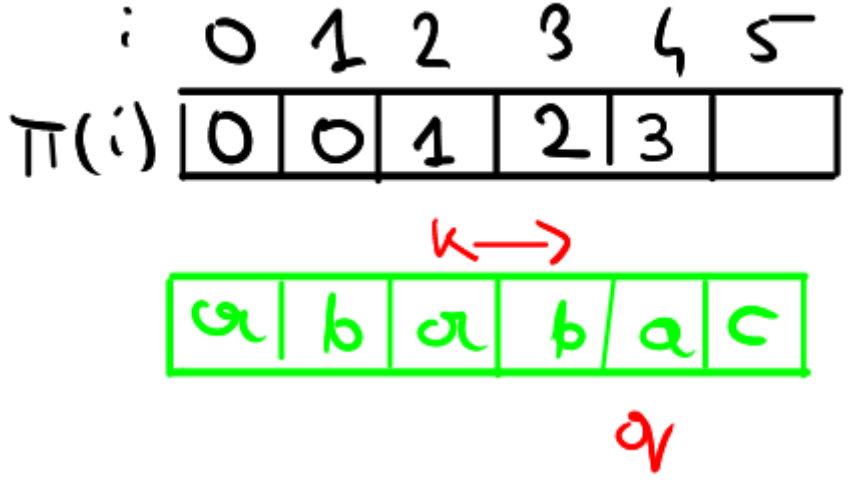
```
def prefix_function(pattern):
    pi = [0]
    k = 0
    for q in range(1, len(pattern)):
        while(k > 0 and pattern[k] != pattern[q]):
            k = pi[k-1]
        if(pattern[k] == pattern[q]):
            k = k + 1
        pi.append(k)
    return pi
```

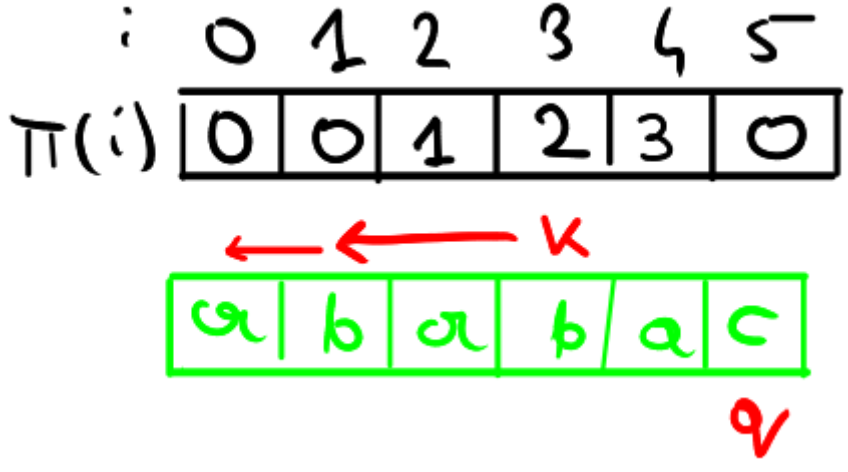

i
0 1 2 3 4 5
π(i)
0
a b a b a c

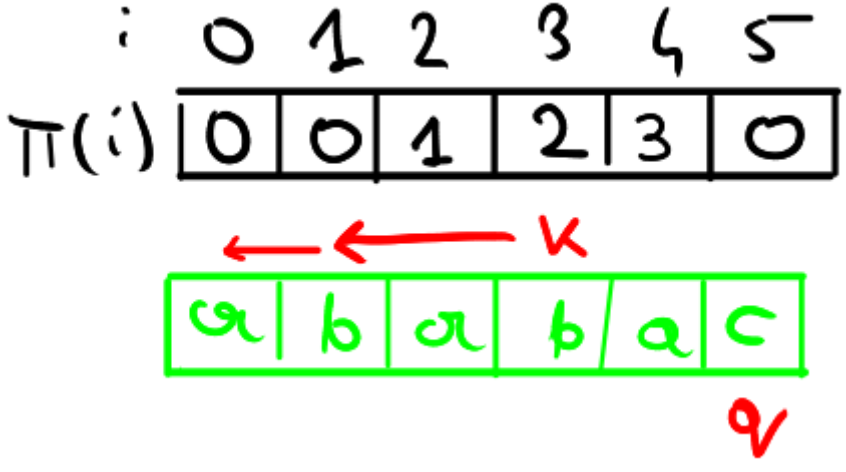

i
0 1 2 3 4 5
π(i)
0 0
k
a b a b a c
q


i
0 1 2 3 4 5
π(i)
0 0 1
k
a b a b a c
q

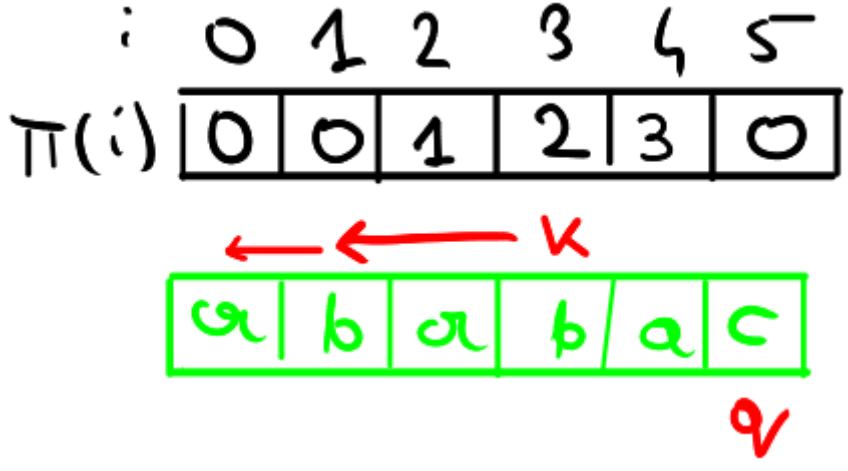

i
0 1 2 3 4 5
π(i)
0 0 1 2
k→
a b a b a c
q


i
0 1 2 3 4 5
π(i)
0 0 1 2 3
k
a b a b a c
q


i
0 1 2 3 4 5
π(i)
0 0 1 2 3 0
k
a b a b a c
q



Złożoność obliczeniowa tworzenia tablicy π: $\Theta(|P|)$



Złożoność obliczeniowa tworzenia tablicy π: $\Theta(|P|)$

In [24]:

```
def kmp_string_matching(text, pattern):
    pi = prefix_function(pattern)
    q = 0
    for i in range(0, len(text)):
        while(q > 0 and pattern[q] != text[i]):
            q = pi[q-1]
        if(pattern[q] == text[i]):
            q = q + 1
        if(q == len(pattern)):
            print(f"Przesunięcie {i + 1 - q} jest poprawne")
            q = pi[q-1]
```

In [25]:

```
prefix_function("abacab")
```

Out[25]:

```
[0, 0, 1, 0, 1, 2]
```

In [25]:

```
prefix_function("abacab")
```

Out[25]:

```
[0, 0, 1, 0, 1, 2]
```

In [26]:

```
kmp_string_matching("abaabaaaaba", "aba")
```

```
Przesunięcie 0 jest poprawne
Przesunięcie 3 jest poprawne
Przesunięcie 8 jest poprawne
```

In [ ]:

```

```