Wyszukiwanie wzorca w skompresowanym tekście

Paweł Gawrychowski

10 października 2013

Wyszukiwanie wzorca w tekście

Czy dany wzorzec p występuje w danym tekście t? Jeśli tak, gdzie jest jego pierwsze wystąpienie?

Znajdź `kjfdkasl` w

```
rokjfdkjncbvdkojsdlkjsldskjxlkalkjfslakjlkxxcv
epofikflskdjflskjvnlmnapodierpereporipojdpdaja
kjrtrgdkjfdkaslkdjoretieodflkgjsnlgkjdslgkjldf
riudkxdjwoisdoiwlkmssoiwoiosdkjwoixkcjksjdkjws
wjnswoislkcxlkqpodskjzlapoqlksdxcmdfepowepofde
zirpotdpoitgiouyoewpoiewlkjdklnkjfdkaslldkjgrp
oieorisdlkweoidssdlkweoidscxmnosdwoioweiwoiwoi
eopripowedkljskljwekljsdldkjsxmcnweioiewdlskjd
rotirlekdlsdfdwmcslkcsdpowkdwpodkwpoekwpoporer
eporjmkjfdkaslpwiowjsklncxmncsldkwpoeiwpoikwed
ojreoijdkmndkjnfekreopreojkslkdjsapowi2poqwiqp
```

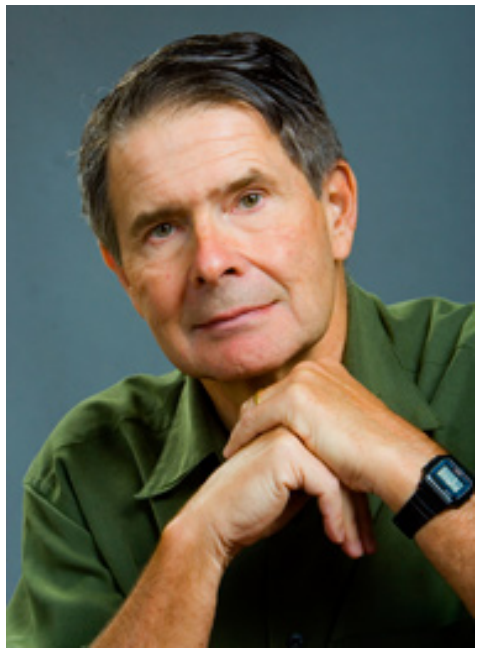
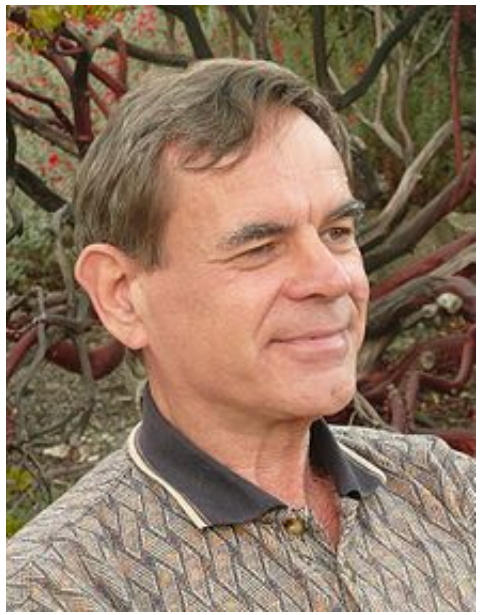
Znajdź `kjfdkasl` w

```
rokjfdkjncbvdkojsdlkjsldskjxlkalkjfslakjlkxxcv
epofikflskdjflskjvnlmnapodierpereporipojdpdaja
kjrtrgdkjfdkaslkdjoretieodflkgjsnlgkjdslgkjldf
riudkxdjwoisdoiwlkmssoiwoiosdkjwoixkcjksjdkjws
wjnswoislkcxlkqpodskjzlapoqlksdxcmdfepowepofde
zirpotdpoitgiouyoewpoiewlkjdklnkjfdkasllldkjgrp
oieorisdlkweoidssdlkweoidscxmnosdwoioweiwoiwoi
eopripowedkljskljwekljsdldkjsxmcnweioiewdlskjd
rotirlekdlsdfdwmcslkcsdpowkdwpodkwpoekwpoporer
eporjmkjfdkaslpwiowjsklncxmncsldkwpoeiwpoikwed
ojreoijdkmndkjnfekreopreojkslkdjsapowi2poqwiqp
```

Znajdź `kjfdkasl` w

```
rokjfdkjncbvdkojsdlkjsldskjxlkalkjfslakjlkxxcv
epofikflskdjflskjvnlmnapodierpereporipojdpdaja
kjrtrgdkjfdkaslkdjoretieodflkgjsnlgkjdslgkjldf
riudkxdjwoisdoiwlkmssoiwoiosdkjwoixkcjksjdkjws
wjnswoislkcxlkqpodskjzlapoqlksdxcmdfepowepofde
zirpotdpoitgiouyoewpoiewlkjdklnkjfdkaslldkjgrp
oieorisdlkweoidssdlkweoidscxmnosdwoioweiwoiwoi
eopripowedkljskljwekljsdldkjsxmcnweioiewdlskjd
rotirlekdlsdfdwmcslkcsdpowkdwpodkwpoekwpoporer
eporjmkjfdkaslpwiowjsklncxmncsldkwpoeiwpoikwed
ojreoijdkmndkjnfekreopreojkslkdjsapowi2poqwiqp
```

Wiadomo, że pierwsze wystąpienie wzorca w tekście można znaleźć w czasie proporcjonalnym do $|p| + |t|$.
Jest wiele różnych algorytmów osiągających taką złożoność, najbardziej znany został podany przez Knutha, Morrisa i Pratta (często nazywa się go **algorytmem KMP**).

W sytuacji, gdy tekst jest **bardzo** długi, konieczne jest przechowywanie go w skompresowanej postaci.

Pytanie

Co jeśli potrzebujemy wyszukać wystąpienie wzorca w tekście, który jest składowany w skompresowanej postaci? Oczywiście zawsze można go po prostu zdekompresować i użyć dowolnego algorytmu wyszukiwania wzorca, ale czy nie wydaje się to marnotrawstwem?

Jaki jest najbardziej naturalny problem związany z przetwarzaniem tekstu?

Wyszukiwanie wzorca w tekście

Czy dany wzorzec p występuje w danym tekście t? Jeśli tak, gdzie jest jego pierwsze wystąpienie?

Jaki jest najbardziej naturalny problem związany z przetwarzaniem tekstu?

Wyszukiwanie wzorca w skompresowanym tekście

Czy dany wzorzec p występuje w danym skompresowanym tekście t? Jeśli tak, gdzie jest jego pierwsze wystąpienie?

A skoro zadaliśmy już takie pytanie, nie sposób nie zauważyć jego naturalnego uogólnienia:

Wyszukiwanie skompresowanego wzorca w skompresowanym tekście

Czy dany skompresowany wzorzec p występuje w danym skompresowanym tekście t? Jeśli tak, gdzie jest jego pierwsze wystąpienie?

Oczywiście wszystko zależy od użytej metody kompresji.

1. kodowanie Huffmana,
2. metody oparte na algorytmie Lempela-Ziva,
3. transformata Burrowsa-Wheelera.

Oczywiście wszystko zależy od użytej metody kompresji.

1. kodowanie Huffmana,
2. metody oparte na algorytmie Lempela-Ziva,
3. transformata Burrowsa-Wheelera.

Metody kompresji oparte na algorytmie Lempela-Ziva

Tekst $t[1..N]$ jest dzielony na rozłączne bloki $b_1 b_2 \ldots b_n$. Każdy kolejny blok jest definiowany na podstawie bloków występujących na lewo od niego.

To co dokładnie rozumiemy przez "definiowany" zależy od konkretnego wariantu. Najczęściej używane są następujące dwa:

LZ77, LZ nowy blok b_i jest podsłowem już przetworzonego prefiksu z doklejonym dokładnie jednym znakiem, `zip`, `gzip`, `PNG`

LZ78, LZW nowy blok b_i powstaje przez doklejenie jednego znaku do wcześniejszego bloku. `compress`, `GIF`, `TIFF`, `PDF`

Przykład kompresji LZW:

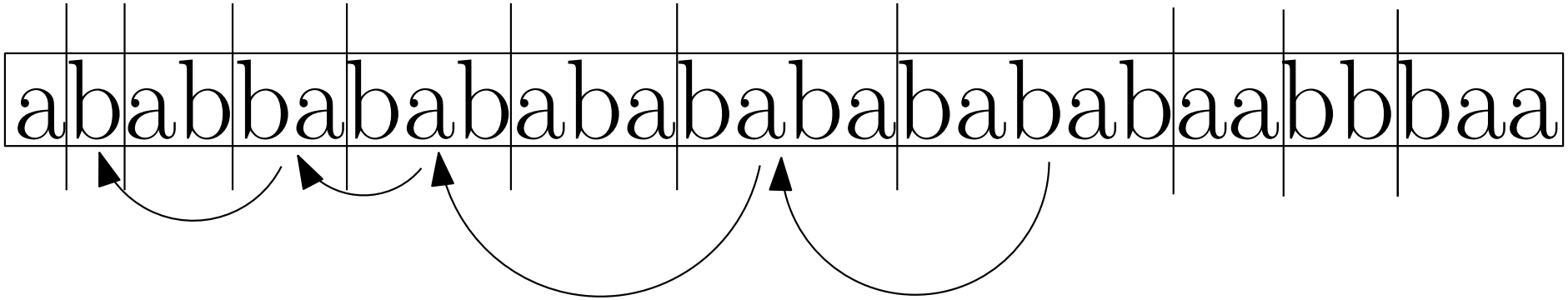
$$\boxed{\text{ababbababababababababaabbbaa}}$$

Co prawda $n \in \Omega(\sqrt{N})$, ale możliwa jest bardzo szybka kompresja/dekompresja.

Przykład kompresji LZW:

Co prawda $n \in \Omega(\sqrt{N})$, ale możliwa jest bardzo szybka kompresja/dekompresja.

Przykład kompresji LZW:



Co prawda $n \in \Omega(\sqrt{N})$, ale możliwa jest bardzo szybka kompresja/dekompresja.

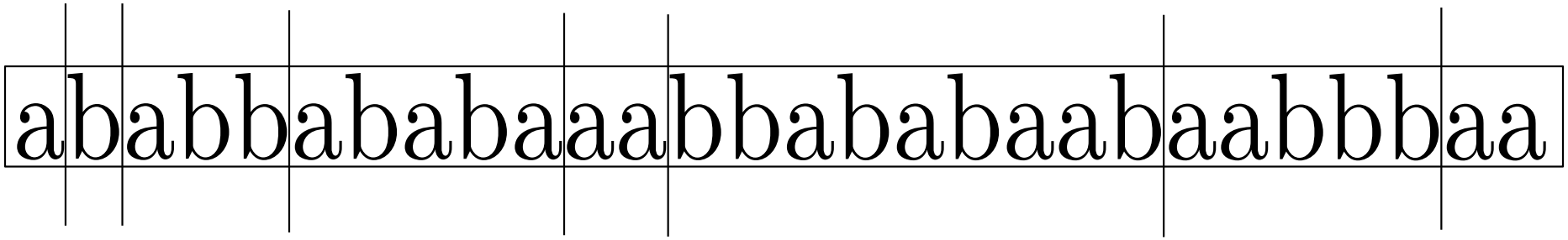
Przykład kompresji LZ:

$$\boxed{\text{ababbababaaaabbbabababaabaabbbbaa}}$$

Łatwo wyobrazić sobie przykład, w którym $n = \mathcal{O}(\log N)$. Co prawda taka sytuacja raczej nie wystąpi w praktyce, ale mamy z nią do czynienia w przypadku słów Fibonacciego, które często są wykorzystywane jako trudny przykład dla różnego rodzaju algorytmów tekstowych.

Możliwy jest też wariant, w którym nowo definiowany blok odwołuje się do samego siebie (self-referential LZ).

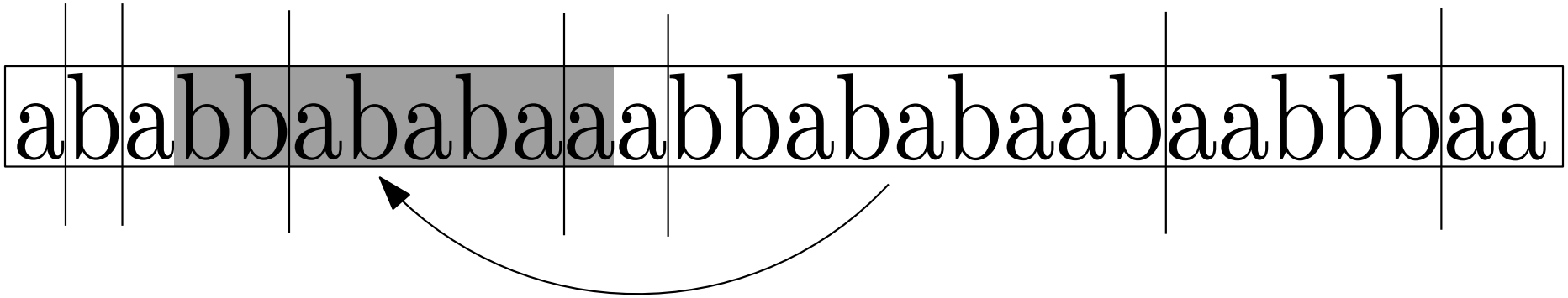
Przykład kompresji LZ:



Łatwo wyobrazić sobie przykład, w którym $n = \mathcal{O}(\log N)$. Co prawda taka sytuacja raczej nie wystąpi w praktyce, ale mamy z nią do czynienia w przypadku słów Fibonacciego, które często są wykorzystywane jako trudny przykład dla różnego rodzaju algorytmów tekstowych.

Możliwy jest też wariant, w którym nowo definiowany blok odwołuje się do samego siebie (self-referential LZ).

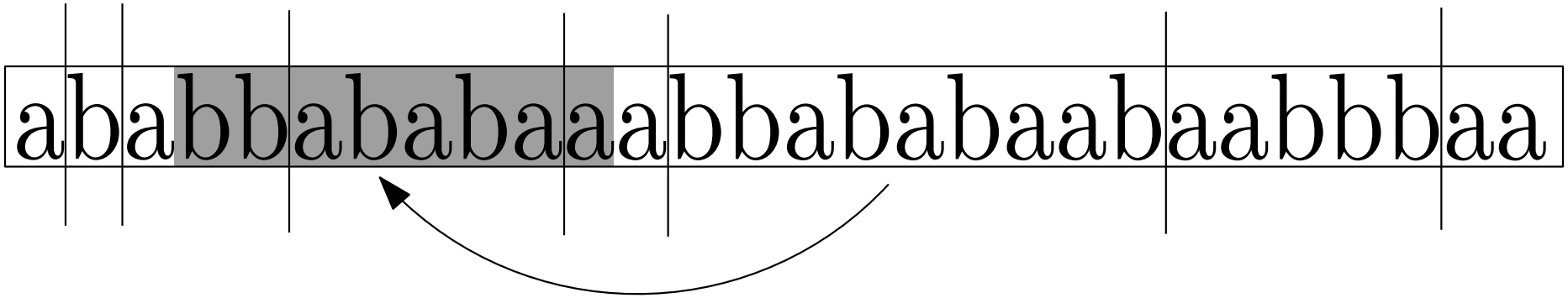
Przykład kompresji LZ:



Łatwo wyobrazić sobie przykład, w którym $n = \mathcal{O}(\log N)$. Co prawda taka sytuacja raczej nie wystąpi w praktyce, ale mamy z nią do czynienia w przypadku słów Fibonacciego, które często są wykorzystywane jako trudny przykład dla różnego rodzaju algorytmów tekstowych.

Możliwy jest też wariant, w którym nowo definiowany blok odwołuje się do samego siebie (self-referential LZ).

Przykład kompresji LZ:



Łatwo wyobrazić sobie przykład, w którym $n = \mathcal{O}(\log N)$. Co prawda taka sytuacja raczej nie wystąpi w praktyce, ale mamy z nią do czynienia w przypadku słów Fibonacciego, które często są wykorzystywane jako trudny przykład dla różnego rodzaju algorytmów tekstowych.

Możliwy jest też wariant, w którym nowo definiowany blok odwołuje się do samego siebie (self-referential LZ).

Kolejne bloki są opisane przez pary (w LZW) lub trójki (w LZ):

$$\ldots\mathrm{ababbababaaaabbbabababaabaabbbbaaa}\ldots$$

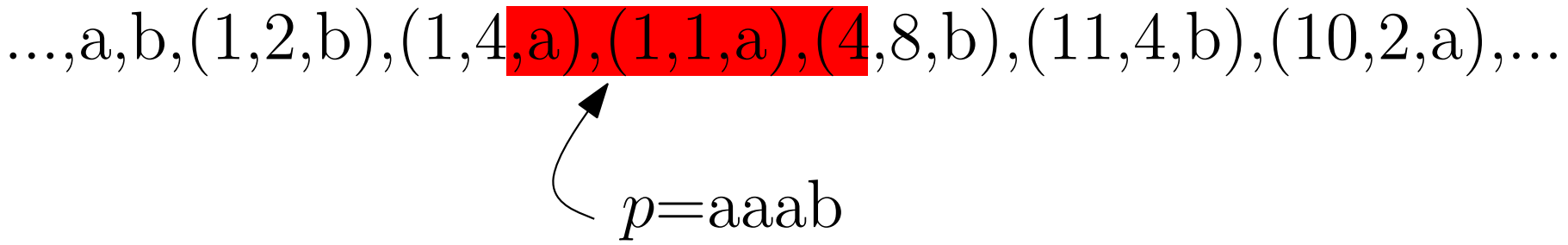
$$\ldots,\mathrm{a},\mathrm{b},(1,2,\mathrm{b}),(1,4,\mathrm{a}),(1,1,\mathrm{a}),(4,8,\mathrm{b}),(11,4,\mathrm{b}),(10,2,\mathrm{a}),\ldots$$

Kolejne bloki są opisane przez pary (w LZW) lub trójki (w LZ):

$$\ldots,\mathrm{a},\mathrm{b},(1,2,\mathrm{b}),(1,4,\mathrm{a}),(1,1,\mathrm{a}),(4,8,\mathrm{b}),(11,4,\mathrm{b}),(10,2,\mathrm{a}),\ldots$$

$$p{=}\mathrm{aaab}$$

Kolejne bloki są opisane przez pary (w LZW) lub trójki (w LZ):



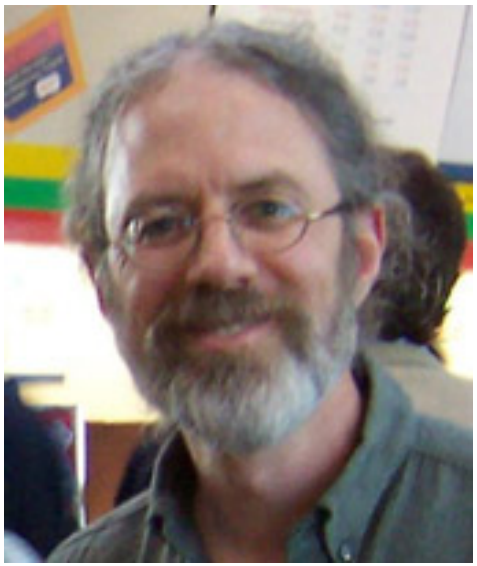
Oznaczenia

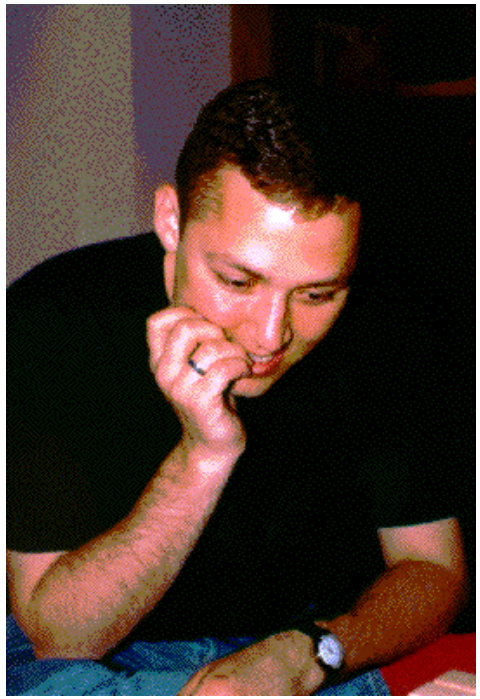
$t[1..N]$ tekst, który po skompresowaniu składa się z n bloków
$p[1..M]$ wzorzec, który po skompresowaniu składa się z m bloków

Wyszukiwanie wzorca w skompresowanym tekście

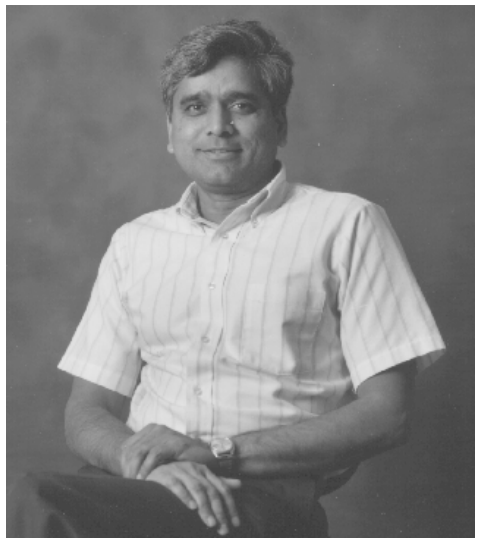
Wejście: $p[1..M]$ i ciąg n bloków definiujących tekst $t[1..N]$
Wyjście: czy p występuje w t?

Pierwszy nietrywialny algorytm wyszukiwania wzorca w LZW-skompresowanym tekście został podany w roku 1994 przez Amira, Bensona i Faracha. A tak naprawdę podali oni dwa algorytmy działające w czasie $\mathcal{O}(n \log M + M)$ i $\mathcal{O}(n + M^2)$.





Rok później ten drugi algorytm został poprawiony przez Kosaraju, który podał rozwiązanie działające w czasie $\mathcal{O}(n + M^{1+\epsilon})$.



Nasuwa się dość oczywiste pytanie: czy istnieje dla tego problemu algorytm działający w czasie $\mathcal{O}(n+M)$?

Tak!

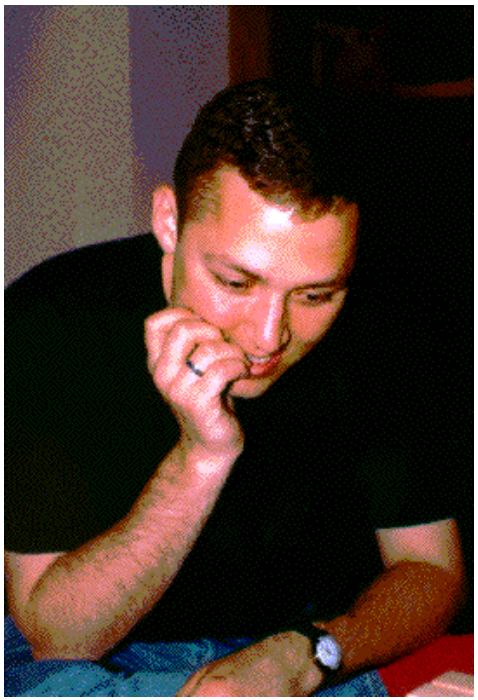
Problem wyszukiwania wzorca w LZW-skompresowanym tekście może być rozwiązany w optymalnym czasie $\mathcal{O}(n+M)$.

Nasuwa się dość oczywiste pytanie: czy istnieje dla tego problemu algorytm działający w czasie $\mathcal{O}(n+M)$?

Tak!

Problem wyszukiwania wzorca w LZW-skompresowanym tekście może być rozwiązany w optymalnym czasie $\mathcal{O}(n+M)$.

W przypadku (bardziej ogólnej) kompresji LZ nie jest oczywiste, że jesteśmy w stanie skonstruować algorytm o czasie działania zależnym od n, a nie N.
Okazuje się, że jest to jednak możliwe, co pokazali w roku 1995 Farach i Thorup. Ich algorytm wyszukiwania wzorca w LZ-skompresowanym tekście działa w (oczekiwanym) czasie $\mathcal{O}(n \log^2 \frac{N}{n} + M)$.



W roku 2010 Iacono i Özkan udowodnili, że jedną z części tego algorytmu może zostać przyśpieszony do $\mathcal{O}(n \log \frac{N}{n})$, jednak nie umieli osiągnąć lepszej złożoności dla całego problemu.
Czy możliwe jest osiągnięcie lepszego czasu działania?

Tak!

Problem wyszukiwania wzorca w LZ-skompresowanym tekście może być rozwiązany w czasie $\mathcal{O}(n \log \frac{N}{n} + m)$.

W roku 2010 Iacono i Özkan udowodnili, że jedną z części tego algorytmu może zostać przyśpieszony do $\mathcal{O}(n \log \frac{N}{n})$, jednak nie umieli osiągnąć lepszej złożoności dla całego problemu.
Czy możliwe jest osiągnięcie lepszego czasu działania?

Tak!

Problem wyszukiwania wzorca w LZ-skompresowanym tekście może być rozwiązany w czasie $\mathcal{O}(n \log \frac{N}{n} + m)$.

W obydwu algorytmach (wyszukiwania wzorca w LZW- i LZ-skompresowanym tekście) kluczowe jest szybkie operowanie na długich blokach tekstu.

W szczególności, konieczne jest skonstruowanie procedury, która mając dane dwa podsłowa wzorca znajdzie wystąpienie w ich konkatenacji.

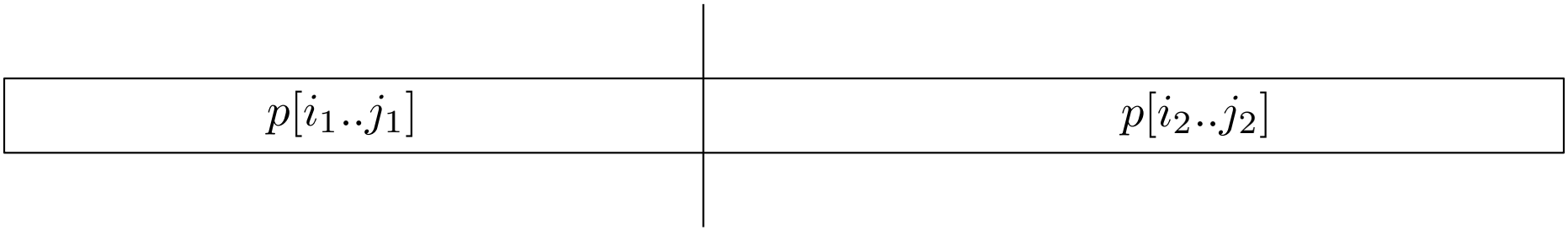
Lemat 4.1

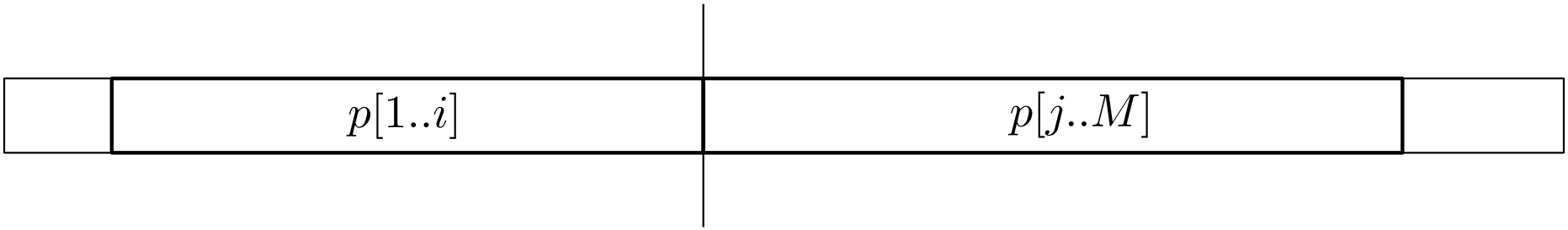
Mając dane dwa podsłowa wzorca można sprawdzić czy wzorzec występuje w ich konkatenacji w stałym czasie.

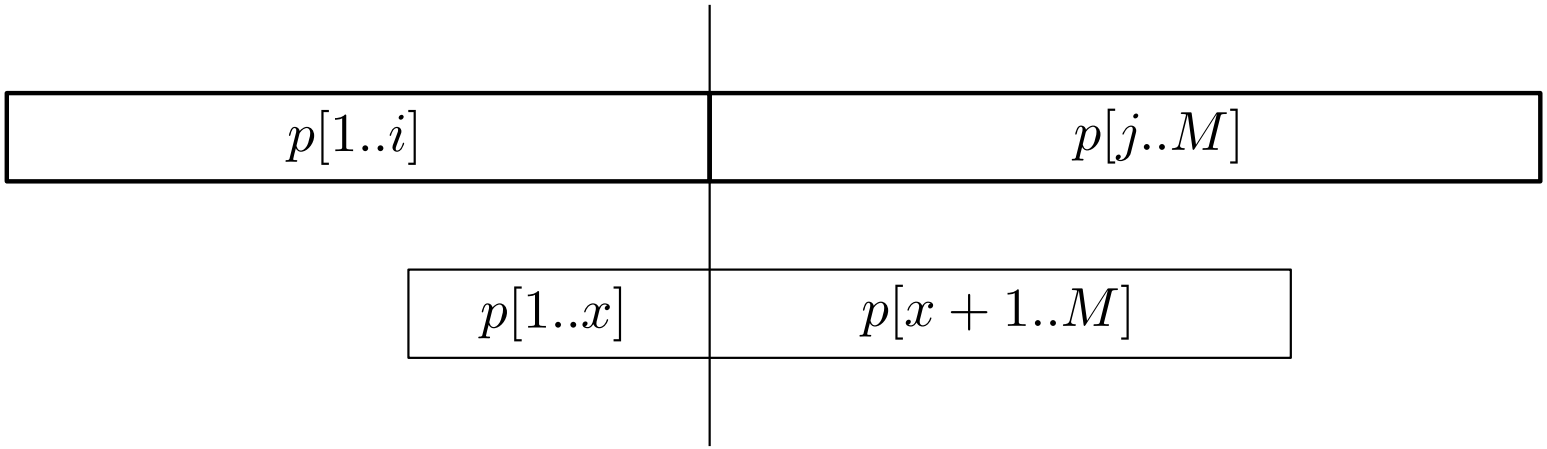
W obydwu algorytmach (wyszukiwania wzorca w LZW- i LZ-skompresowanym tekście) kluczowe jest szybkie operowanie na długich blokach tekstu.
W szczególności, konieczne jest skonstruowanie procedury, która mając dane dwa podsłowa wzorca znajdzie wystąpienie w ich konkatenacji.

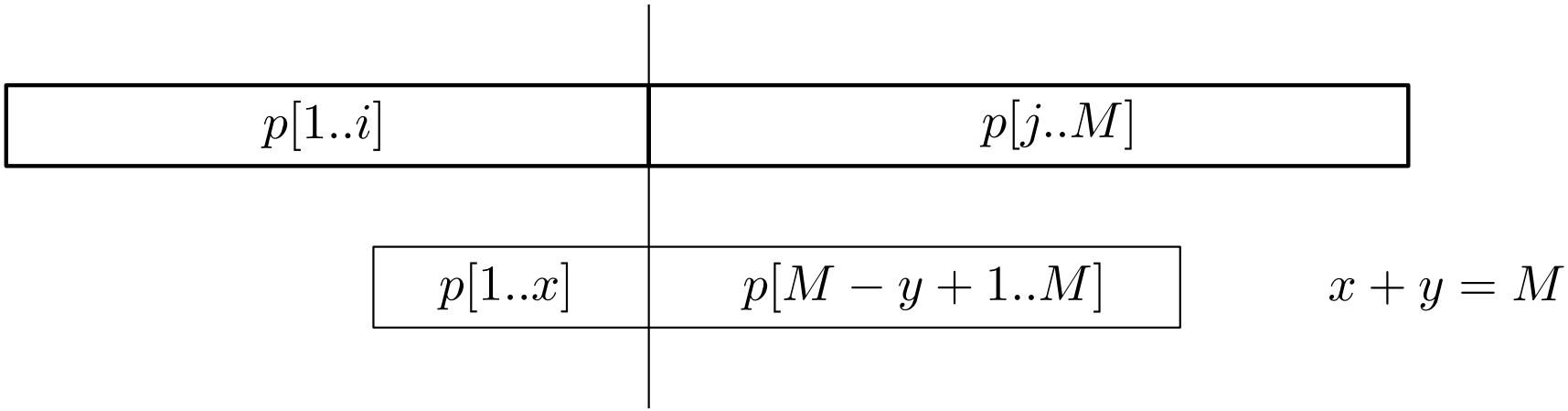
Lemat 4.1

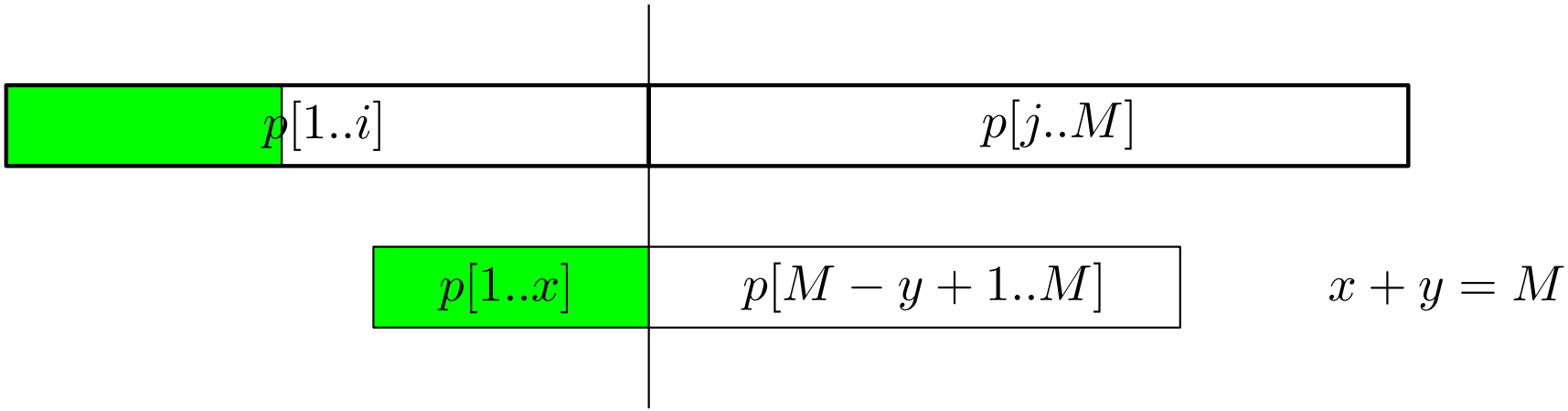
Mając dane dwa podsłowa wzorca można sprawdzić czy wzorzec występuje w ich konkatenacji w stałym czasie.

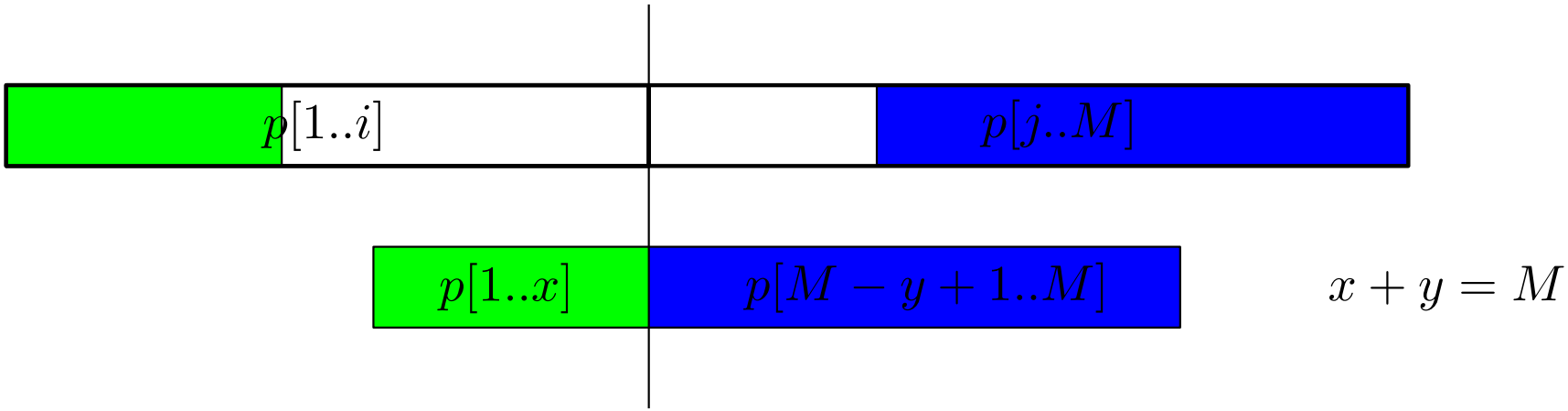

$p[i_1..j_1]$
$p[i_2..j_2]$

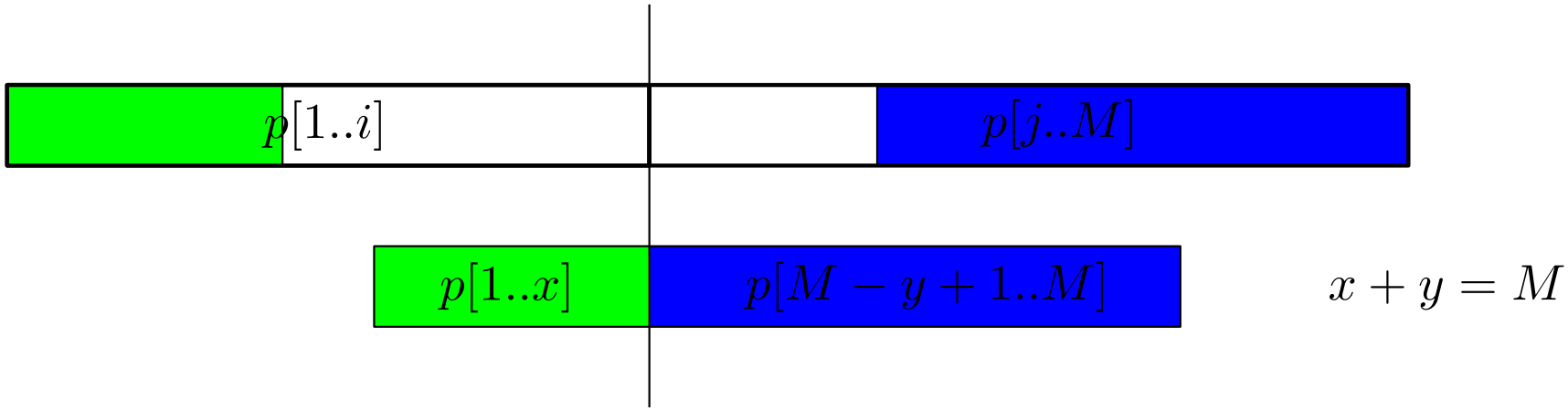

$p[1..i]$
$p[j..M]$


$p[1..i]$
$p[j..M]$
$p[1..x]$
$p[x+1..M]$


$p[1..i]$
$p[j..M]$
$p[1..x]$
$p[M-y+1..M]$
$x+y=M$


$p[1..i]$
$p[j..M]$
$p[1..x]$
$p[M-y+1..M]$
$x+y=M$


$p[1..i]$
$p[j..M]$
$p[1..x]$
$p[M-y+1..M]$
$x + y = M$



b jest borderem słowa *w* iff $w[1..b] = w[|w| - b + 1..|w|]$

Szukamy takich $x \in \text{border}(p[1..i])$ i $y \in \text{border}(p[j..M])$, że $x + y = M$.

border(w) może mieć bardzo dużo (liniowo) różnych elementów, które na pierwszy rzut oka wydają się nie mieć wiele wspólnego. Jeśli jednak popatrzymy się na:

$$\text{border}(w) \cap \{\left\lceil \frac{|w|}{2} \right\rceil, \ldots, |w| - 1, |w|\}$$

można zauważyć, że ta część tworzy ciąg arytmetyczny. A dokładniej, jeżeli d jest okresem w, są one postaci $|w| - \alpha d$. Nazywamy je **długimi borderami**.

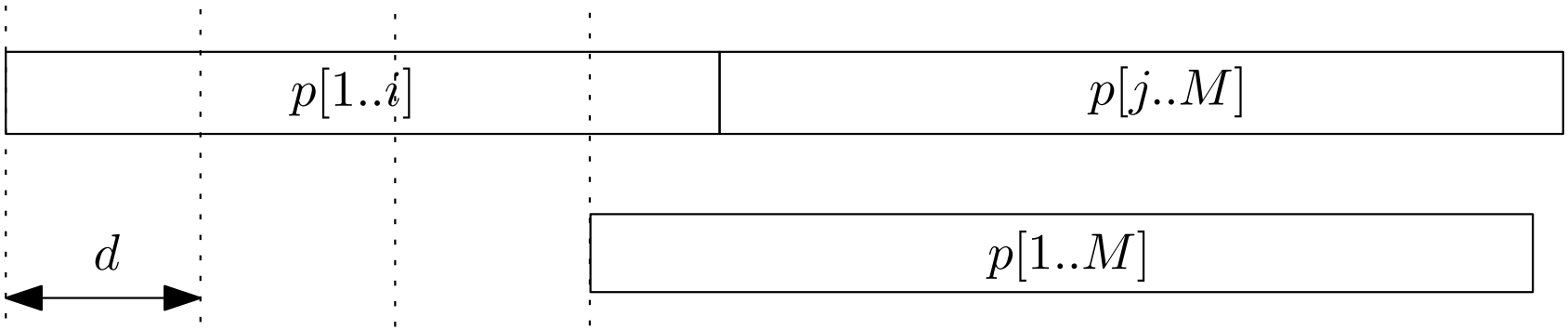
Teraz można zauważyć, że x lub y musi być długim borderem. Ze względu na symetrię wystarczy poradzić sobie z tym pierwszym przypadkiem.

border(w) może mieć bardzo dużo (liniowo) różnych elementów, które na pierwszy rzut oka wydają się nie mieć wiele wspólnego. Jeśli jednak popatrzymy się na:

$$\text{border}(w) \cap \{\left\lceil \frac{|w|}{2} \right\rceil, \ldots, |w| - 1, |w|\}$$

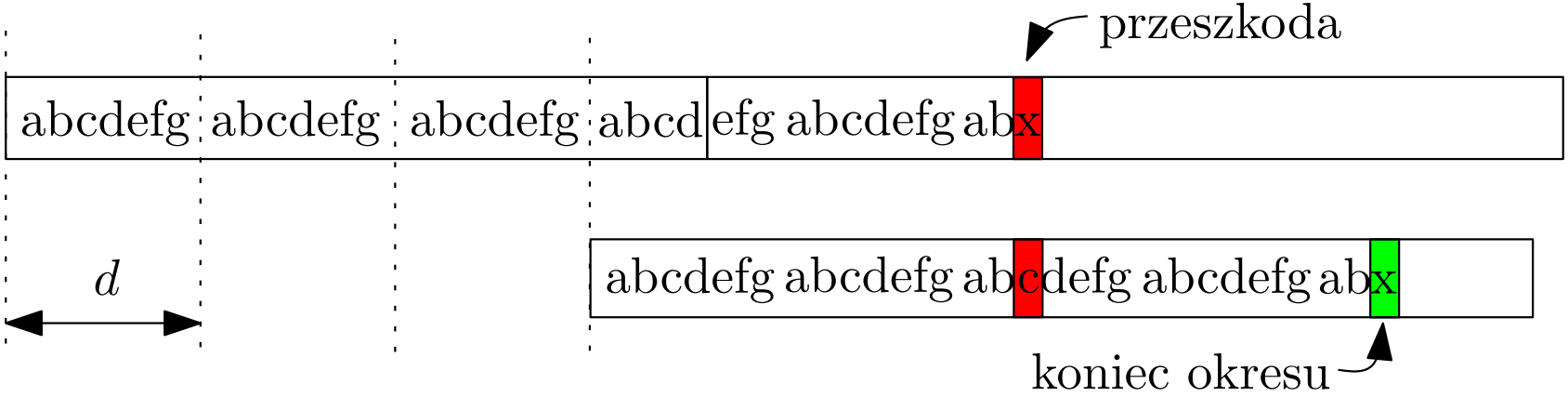
można zauważyć, że ta część tworzy ciąg arytmetyczny. A dokładniej, jeżeli d jest okresem w, są one postaci $|w| - \alpha d$. Nazywamy je **długimi borderami**.
Teraz można zauważyć, że x lub y musi być długim borderem. Ze względu na symetrię wystarczy poradzić sobie z tym pierwszym przypadkiem.

Wybieramy przesunięcie postaci αd, które zaczyna się w $p[1..i]$, ale jest tak daleko na prawo jak to tylko możliwe. Wyznaczamy pierwszą niezgodność tego przesunięcia.



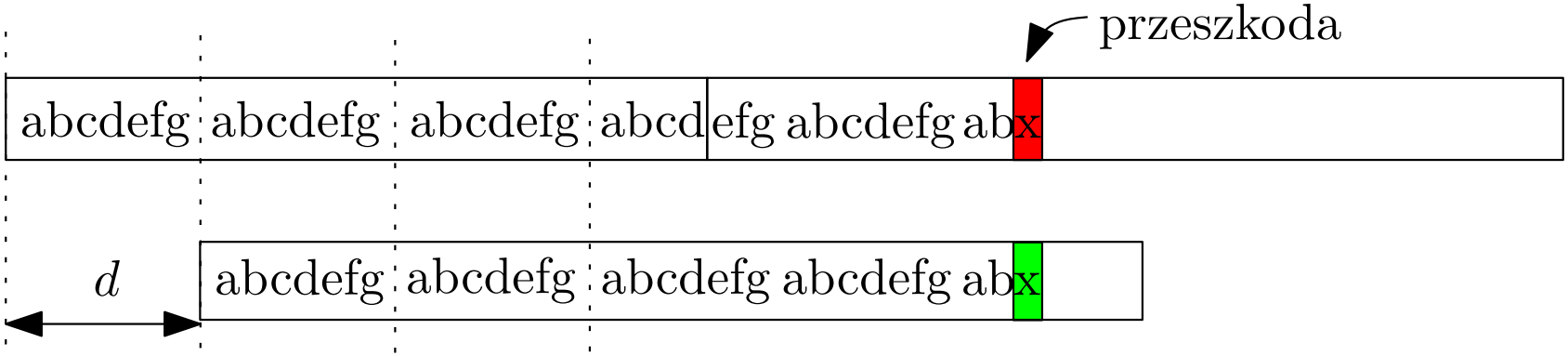
Okazuje się, że na podstawie pozycji przeszkody można wyznaczyć prosty arytmetyczny warunek, który pozwala nam na wyeliminowanie wszystkich przesunięć poza jednym. Sprawdzamy to jedyne potencjalne przesunięcie za pomocą kilku pytań o równość fragmentów wzorca.

Wybieramy przesunięcie postaci αd, które zaczyna się w $p[1..i]$, ale jest tak daleko na prawo jak to tylko możliwe. Wyznaczamy pierwszą niezgodność tego przesunięcia.



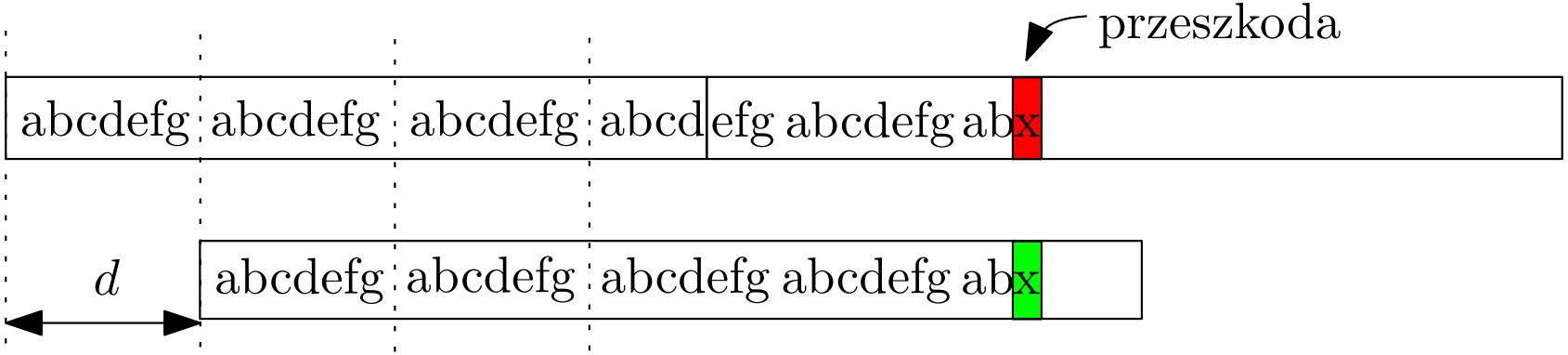
Okazuje się, że na podstawie pozycji przeszkody można wyznaczyć prosty arytmetyczny warunek, który pozwala nam na wyeliminowanie wszystkich przesunięć poza jednym. Sprawdzamy to jedyne potencjalne przesunięcie za pomocą kilku pytań o równość fragmentów wzorca.

Wybieramy przesunięcie postaci αd, które zaczyna się w $p[1..i]$, ale jest tak daleko na prawo jak to tylko możliwe. Wyznaczamy pierwszą niezgodność tego przesunięcia.



Okazuje się, że na podstawie pozycji przeszkody można wyznaczyć prosty arytmetyczny warunek, który pozwala nam na wyeliminowanie wszystkich przesunięć poza jednym. Sprawdzamy to jedyne potencjalne przesunięcie za pomocą kilku pytań o równość fragmentów wzorca.

Wybieramy przesunięcie postaci αd, które zaczyna się w $p[1..i]$, ale jest tak daleko na prawo jak to tylko możliwe. Wyznaczamy pierwszą niezgodność tego przesunięcia.



Okazuje się, że na podstawie pozycji przeszkody można wyznaczyć prosty arytmetyczny warunek, który pozwala nam na wyeliminowanie wszystkich przesunięć poza jednym. Sprawdzamy to jedyne potencjalne przesunięcie za pomocą kilku pytań o równość fragmentów wzorca.

Wyszukiwanie skompresowanego wzorca w skompresowanym tekście

Wejście: ciąg m bloków definiujących wzorzec $p[1..M]$ i ciąg n bloków definiujących tekst $t[1..N]$
Wyjście: czy p występuje w t?

W tym przypadku problem wydaje się istotnie trudniejszy: wszystkie liniowe algorytmy wyszukiwania wzorca bazują na mniej lub bardziej skomplikowanym przetworzeniu wszystkich prefiksów/sufiksów/... wzorca. Jeśli wzorzec jest skompresowany, nie wiadomo czy mamy na to czas.

Okazuje się jednak, że tak zdefiniowany problem wyszukiwania LZW-skompresowanego wzorca w LZW-skompresowanym tekście może być rozwiązany w czasie $\mathcal{O}((n+m)\log(n+m))$, co udowodnili w roku 1999 Gąsieniec i Rytter.

Nasuwa się oczywiste pytanie: czy można osiągnąć złożoność $\mathcal{O}(n+m)$?

Tak!

Problem wyszukiwania LZW-skompresowanego wzorca w LZW-skompresowanym tekście może być rozwiązany w optymalnym czasie $\mathcal{O}(n+m)$.

Pierwszym krokiem w rozwiązaniu jest wybranie "trudnego" fragmentu wzorca o (nieskompresowanej) długości $n+m$ i wygenerowanie wszystkich jego wystąpień w tekście. Następnie trzeba tylko sprawdzić, które z tych wystąpień rozszerzają się do wystąpień całego wzorca.

Nasuwa się oczywiste pytanie: czy można osiągnąć złożoność $\mathcal{O}(n+m)$?

Tak!

Problem wyszukiwania LZW-skompresowanego wzorca w LZW-skompresowanym tekście może być rozwiązany w optymalnym czasie $\mathcal{O}(n+m)$.

Pierwszym krokiem w rozwiązaniu jest wybranie "trudnego" fragmentu wzorca o (nieskompresowanej) długości $n+m$ i wygenerowanie wszystkich jego wystąpień w tekście. Następnie trzeba tylko sprawdzić, które z tych wystąpień rozszerzają się do wystąpień całego wzorca.

Nasuwa się oczywiste pytanie: czy można osiągnąć złożoność $\mathcal{O}(n+m)$?

Tak!

Problem wyszukiwania LZW-skompresowanego wzorca w LZW-skompresowanym tekście może być rozwiązany w optymalnym czasie $\mathcal{O}(n+m)$.

Pierwszym krokiem w rozwiązaniu jest wybranie "trudnego" fragmentu wzorca o (nieskompresowanej) długości $n+m$ i wygenerowanie wszystkich jego wystąpień w tekście. Następnie trzeba tylko sprawdzić, które z tych wystąpień rozszerzają się do wystąpień całego wzorca.

Kolejnym naturalnym uogólnieniem wyszukiwania wzorca w skompresowanym tekście jest szukanie więcej niż jednego wzorca.

Wyszukiwanie wielu wzorców w skompresowanym tekście

Wejście: wzorce $p_1, p_2, \ldots, p_k$ o sumarycznej długości M i ciąg n bloków definiujących tekst $t[1..N]$
Wyjście: czy któryś z p_i występuje w t?

Oczywiście można szukać każdego p_i osobno, ale jeśli będzie ich kilkaset tysięcy, wydaje się to nierozsądne.
Okazuje się, że dla wyszukiwania wielu wzorców w LZW-skompresowanym tekście można osiągnąć złożoność $\mathcal{O}(n + M^2)$, co pokazali w roku 1998 Kida, Takeda, Shinohara, Miyazaki i Arikawa.




Czy możliwa jest lepsza złożoność?

Tak!

Problem wyszukiwania wielu wzorców w LZW-skompresowanym tekście może być rozwiązany w czasie $\mathcal{O}(n + M^{1+\epsilon})$ lub $\mathcal{O}(n \log M + M)$.

Kluczowe dla rozwiązania jest zredukowanie oryginalnego problemu do zbudowania struktury danych umożliwiającej szybkie odpowiadanie na pytania dotyczące punktów/odcinków/prostokątów na płaszczyźnie.

Czy możliwa jest lepsza złożoność?

Tak!

Problem wyszukiwania wielu wzorców w LZW-skompresowanym tekście może być rozwiązany w czasie $\mathcal{O}(n + M^{1+\epsilon})$ lub $\mathcal{O}(n \log M + M)$.

Kluczowe dla rozwiązania jest zredukowanie oryginalnego problemu do zbudowania struktury danych umożliwiającej szybkie odpowiadanie na pytania dotyczące punktów/odcinków/prostokątów na płaszczyźnie.

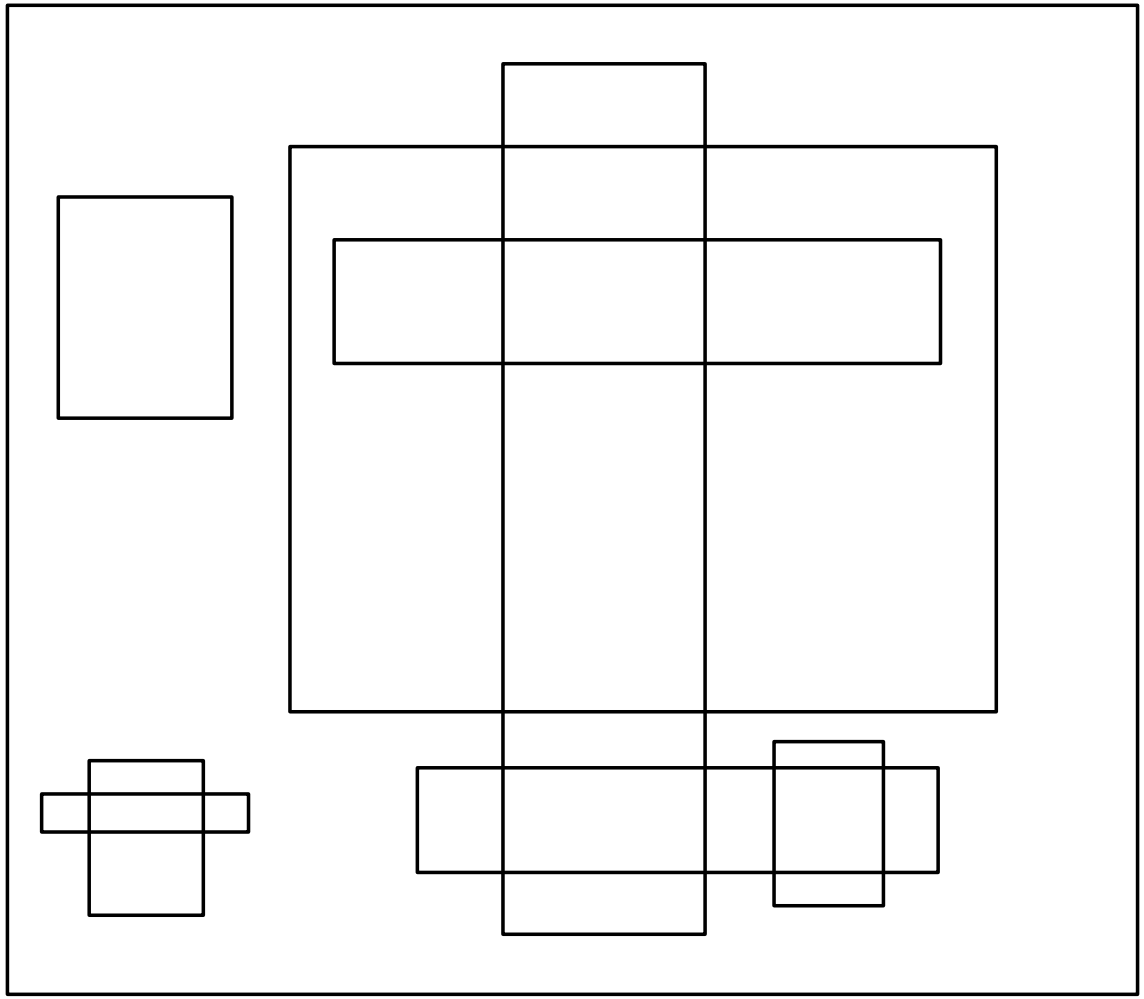
Czy możliwa jest lepsza złożoność?

Tak!

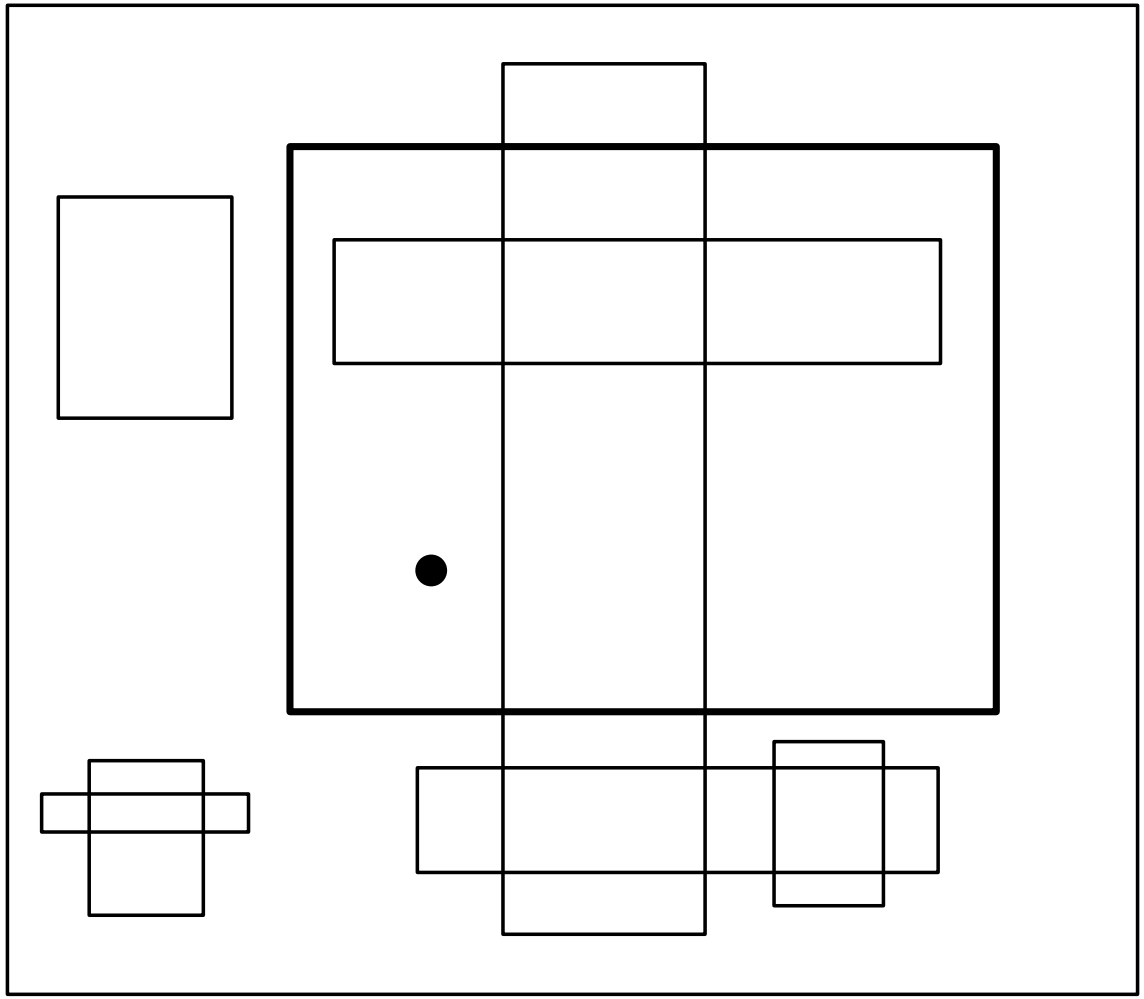
Problem wyszukiwania wielu wzorców w LZW-skompresowanym tekście może być rozwiązany w czasie $\mathcal{O}(n + M^{1+\epsilon})$ lub $\mathcal{O}(n \log M + M)$.

Kluczowe dla rozwiązania jest zredukowanie oryginalnego problemu do zbudowania struktury danych umożliwiającej szybkie odpowiadanie na pytania dotyczące punktów/odcinków/prostokątów na płaszczyźnie.

Na przykład, dla zbioru prostokątów o bokach równoległych do osi OX/OY chcemy zbudować strukturę danych, która po podaniu punktu szybko znajdzie prostokąt, do którego ten punkt należy.



Na przykład, dla zbioru prostokątów o bokach równoległych do osi OX/OY chcemy zbudować strukturę danych, która po podaniu punktu szybko znajdzie prostokąt, do którego ten punkt należy.



Ale czy wyszukiwanie wzorca naprawdę jest tym, co chcielibyśmy rozwiązać? Pewnie równie (a może nawet bardziej) interesujące są dla nas fragmenty tekstu, które wyglądają podobnie do wzorca.

Znajdź `kjfdkasl` z ≤ 1 niezgodnością w

```
rokjfdkjncbvdkojsdlkjsldskjxlkalkjfslakjlkxxcv
epofikflskdjflskjvnlmnapodierpereporipojdpdaja
kjrtrgdkjfdkaslkdjoretieodflkgjsnlgkjdslgkjldf
riudkxdjwoisdoiwlkmssoiwoiosdkjwoixkjadkasljws
wjnswoislkcxlkqpodskjzlapoqlksdxcmdfepowepofde
zirpotdpoitgiouyoewpoiewlkjdklnkjfdkaslldkjgrp
oieorisdlkweoidssdlkweoidscxmnosdwoioweiwoiwoi
eopripowedkdljskljweklkjfdasljsxmcnweioiewdlsk
rotirlekdlsdfdwmcslkcsdpowkdwpodkwpoekwpoporer
eporjmkjfdkaslpwiowjsklncxmncsldkwpoeiwpoikwed
ojreoijdkmndkjnfekreopreojkslkdjsapowi2poqwiqp
```

Znajdź `kjfdkasl` z ≤ 1 niezgodnością w

```
rokjfdkjncbvdkojsdlkjsldskjxlkalkjfslakjlkxxcv
epofikflskdjflskjvnlmnapodierpereporipojdpdaja
kjrtrgdkjfdkaslkdjoretieodflkgjsnlgkjdslgkjldf
riudkxdjwoisdoiwlkmssoiwoiosdkjwoixkjadkasljws
wjnswoislkcxlkqpodskjzlapoqlksdxcmdfepowepofde
zirpotdpoitgiouyoewpoiewlkjdklnkjfdkasldkjgrp
oieorisdlkweoidssdlkweoidscxmnosdwoioweiwoiwoi
eopripowedkdljskljweklkjfdasljsxmcnweioiewdlsk
rotirlekdlsdfdwmcslkcsdpowkdwpodkwpoekwpoporer
eporjmkjfdkaslpwiowjsklncxmncsldkwpoeiwpoikwed
ojreoijdkmndkjnfekreopreojkslkdjsapowi2poqwiqp
```

Znajdź `kjfdkasl` z $\leq$ 1 błędem w

```
rokjfdkjncbvdkojsdlkjsldskjxlkalkjfslakjlkxxcv
epofikflskdjflskjvnlmnapodierpereporipojdpdaja
kjrtrgdkjfdkaslkdjoretieodflkgjsnlgkjdslgkjldf
riudkxdjwoisdoiwlkmssoiwoiosdkjwoixkjadkasljws
wjnswoislkcxlkqpodskjzlapoqlksdxcmdfepowepofde
zirpotdpoitgiouyoewpoiewlkjdklnkjfdkaslldkjgrp
oieorisdlkweoidssdlkweoidscxmnosdwoioweiwoiwoi
eopripowedkdljskljweklkjfdasljsxmcnweioiewdlsk
rotirlekdlsdfdwmcslkcsdpowkdwpodkwpoekwpoporer
eporjmkjfdkaslpwiowjsklncxmncsldkwpoeiwpoikwed
ojreoijdkmndkjnfekreopreojkslkdjsapowi2poqwiqp
```

Znajdź `kjfdkasl` z $\leq$ 1 błędem w

```
rokjfdkjncbvdkojsdlkjsldskjxlkalkjfslakjlkxxcv
epofikflskdjflskjvnlmnapodierpereporipojdpdaja
kjrtrgdkjfdkaslkdjoretieodflkgjsnlgkjdslgkjldf
riudkxdjwoisdoiwlkmssoiwoiosdkjwoixkjadkasljws
wjnswoislkcxlkqpodskjzlapoqlksdxcmdfepowepofde
zirpotdpoitgiouyoewpoiewlkjdklnkjfdkasllldkjgrp
oieorisdlkweoidssdlkweoidscxmnosdwoioweiwoiwoi
eopripowedkdljskljweklkjfdasljsxmcnweioiewdlsk
rotirlekdlsdfdwmcslkcsdpowkdwpodkwpoekwpoporer
eporjmkjfdkaslpwiowjsklncxmncsldkwpoeiwpoikwed
ojreoijdkmndkjnfekreopreojkslkdjsapowi2poqwiqp
```

Przybliżone wyszukiwanie wzorca w skompresowanym tekście

Wejście: $p[1..M]$, parametr k i ciąg n bloków definiujących tekst $t[1..N]$
Wyjście: czy p występuje w t z $\leq k$ błędami/niezgodnościami?

Przybliżone wyszukiwanie wzorca w skompresowanym tekście

Wejście: $p[1..M]$, parametr k i ciąg n bloków definiujących tekst $t[1..N]$
Wyjście: czy p występuje w t z $\leq k$ błędami/niezgodnościami?

Dla wyszukiwania z niezgodnościami w LZW-skompresowanym tekście:

- $\mathcal{O}(nMk)$ Kärkkäinen, Navarro, Ukkonen 2000,
- $\mathcal{O}(\frac{MN}{\log M} + Mn)$ Crochemore, Landau, Ziv-Ukelson 2002,
- $\mathcal{O}(nk^4 + nM)$ Bille, Fagerberg, Gørtz 2007.

Przybliżone wyszukiwanie wzorca w skompresowanym tekście

Wejście: $p[1..M]$, parametr k i ciąg n bloków definiujących tekst $t[1..N]$
Wyjście: czy p występuje w t z $\leq k$ błędami/niezgodnościami?

Dla wyszukiwania z niezgodnościami w LZW-skompresowanym tekście:

- $\mathcal{O}(nMk)$ Kärkkäinen, Navarro, Ukkonen 2000,
- $\mathcal{O}(\frac{MN}{\log M} + Mn)$ Crochemore, Landau, Ziv-Ukelson 2002,
- $\mathcal{O}(nk^4 + nM)$ Bille, Fagerberg, Gørtz 2007.

Przybliżone wyszukiwanie wzorca w skompresowanym tekście

Wejście: $p[1..M]$, parametr k i ciąg n bloków definiujących tekst $t[1..N]$
Wyjście: czy p występuje w t z $\leq k$ błędami/niezgodnościami?

Dla wyszukiwania z błędami w LZW-skompresowanym tekście:

- $\mathcal{O}(n+M)$ if $m \leq w$ Kida, Takeda, Shinohara, Arikawa 1999,
- $\mathcal{O}(nk^3 \log k + nM \log k)$ Bille, Fagerberg, Gørtz 2007 + Amir, Lewenstein, Porat 2000,
- $\mathcal{O}(nM\sqrt{k \log k})$ Bille, Fagerberg, Gørtz 2007 + Amir, Lewenstein, Porat 2000.

Wszystkie wyżej wymienione rozwiązania wymagają czasu $\Omega(nM)$ nawet przy małych k. Czy jest to konieczne?

Nie!

Problem wyszukiwania wzorca z k niezgodnościami w LZW-skompresowanym tekście może być rozwiązany w czasie $\mathcal{O}(n\sqrt{M}k^2)$.

Kluczowe w obydwu rozwiązaniach jest zaadaptowanie technik używanych dla nieskompresowanych tekstów, które należy zmodyfikować tak, aby wykorzystać powtarzalną strukturę LZW-skompresowanych tekstów. Przydatne okazuje się to, że umiemy znajdować wszystkie wystąpienia wielu wzorców w LZW-skompresowanym tekście.

Wszystkie wyżej wymienione rozwiązania wymagają czasu $\Omega(nM)$ nawet przy małych k. Czy jest to konieczne?

Nie!

Problem wyszukiwania wzorca z k niezgodnościami w LZW-skompresowanym tekście może być rozwiązany w czasie $\mathcal{O}(n\sqrt{M}k^2)$.

Kluczowe w obydwu rozwiązaniach jest zaadaptowanie technik używanych dla nieskompresowanych tekstów, które należy zmodyfikować tak, aby wykorzystać powtarzalną strukturę LZW-skompresowanych tekstów. Przydatne okazuje się to, że umiemy znajdować wszystkie wystąpienia wielu wzorców w LZW-skompresowanym tekście.

Wszystkie wyżej wymienione rozwiązania wymagają czasu $\Omega(nM)$ nawet przy małych k. Czy jest to konieczne?

Nie!

Problem wyszukiwania wzorca z k błędami w LZW-skompresowanym tekście może być rozwiązany w czasie $\mathcal{O}(n\sqrt{M}k^3)$.

Kluczowe w obydwu rozwiązaniach jest zaadaptowanie technik używanych dla nieskompresowanych tekstów, które należy zmodyfikować tak, aby wykorzystać powtarzalną strukturę LZW-skompresowanych tekstów. Przydatne okazuje się to, że umiemy znajdować wszystkie wystąpienia wielu wzorców w LZW-skompresowanym tekście.

Wszystkie wyżej wymienione rozwiązania wymagają czasu $\Omega(nM)$ nawet przy małych k. Czy jest to konieczne?

Nie!

Problem wyszukiwania wzorca z k błędami w LZW-skompresowanym tekście może być rozwiązany w czasie $\mathcal{O}(n\sqrt{M}k^3)$.

Kluczowe w obydwu rozwiązaniach jest zaadaptowanie technik używanych dla nieskompresowanych tekstów, które należy zmodyfikować tak, aby wykorzystać powtarzalną strukturę LZW-skompresowanych tekstów. Przydatne okazuje się to, że umiemy znajdować wszystkie wystąpienia wielu wzorców w LZW-skompresowanym tekście.

Dziękuję za uwagę!

LZW	$\mathcal{O}(n+M)$	SODA 2011
	$\mathcal{O}(n+m)$	w pełni skompresowane, STACS 2012
	$\mathcal{O}(n+M^{1+\epsilon})$	wiele wzorców, CPM 2012
	$\mathcal{O}(n\log M+M)$	
	$\mathcal{O}(n\sqrt{M}k^2)$	k niezgodności/błędów, ISAAC 2013
	$\mathcal{O}(n\sqrt{M}k^3)$	
LZ	$\mathcal{O}(n\log\frac{N}{n}+M)$	ESA 2011