STUDIA INFORMATICA 2013
Volume 34 Number 1 (110)

Jacek WIDUCH
Politechnika Śląska, Instytut Informatyki

ANALIZA PORÓWNAWCZA ALGORYTMÓW WYSZUKIWANIA WZORCA W TEKŚCIE

Streszczenie. Jednym z problemów związanych z przetwarzaniem tekstów jest problem wyszukiwania wzorca w tekście, którego celem jest wyznaczenie wszystkich wystąpień w zadanym tekście innego tekstu, zwanego wzorcem. W niniejszej pracy dokonano analizy porównawczej istniejących algorytmów wyszukiwania wzorca w tekście, przy czym kryterium porównawczym jest czas wyszukiwania wzorca. Wyniki przeprowadzonych badań zamieszczono w pracy.

Słowa kluczowe: tekst, wzorzec, alfabet, wyszukiwanie wzorca, prefiks, sufiks, funkcja prefiksowa

A COMPARATIVE ANALYSIS OF STRING SEARCHING ALGORITHMS

Summary. One of the text processing problems is the pattern matching problem. The goal of the problem is to find all places where one text or string, called pattern, is found within the given text. In this paper, a comparative analysis of existing string matching algorithms is presented, and the comparison criterion is the time of searching the pattern in the text. The results of the tests are also presented.

Keywords: text, pattern, alphabet, string searching, prefix, suffix, prefix function

1. Wstęp

Tekst jest jedną z najczęściej stosowanych form zapisu oraz przekazu informacji. Jako przykład można podać zbiory publikacji, książek, różnego rodzaju instrukcje czy też dokumentacje. Także w informatyce tekst jest powszechnie stosowany, gdzie np. wiele dokumentacji do oprogramowania jest zapisanych w formie plików z tekstem. Inną dziedziną

jest biologia molekularna, w której struktury molekuł są zapisane w postaci ciągów nukleotydów lub aminokwasów, które z kolei są zapisane jako ciągi znaków. Dzięki rozwojowi sieci Internet znaczna część informacji jest dostępna poprzez serwisy WWW w postaci tekstowej. Wymienione zastosowania są jedynie przykładowymi, gdyż tekst jako forma przekazu i zapisu informacji znajduje zastosowanie także w innych dziedzinach.

Jednym z elementów przetwarzania tekstów jest wyszukiwanie wzorca, którego celem jest odnalezienie w zadanym tekście pewnego innego tekstu, nazywanego wzorcem. Problem może polegać tylko na znalezieniu dowolnego wystąpienia wzorca w tekście lub znalezieniu wszystkich wystąpień. Operacja wyszukiwania wzorca w tekście jest jedną z operacji zaimplementowanych w powszechnie stosowanym oprogramowaniu, jak np. przeglądarki internetowe, edytory tekstów, środowiska programistyczne. Także system operacyjny oferuje np. możliwość znalezienia plików, zawierających określony wzorzec.

W niniejszej pracy dokonano analizy istniejących algorytmów wyszukiwania wzorca w tekście, przy czym za kryterium porównawcze przyjęto czas jego wyszukiwania. Badania przeprowadzono dla algorytmów: Naive, Knutha–Morrisa–Pratta, Boyera–Moore'a, Horspoola, Quick Search, Smitha, Raity i Not So Naive. Został zbadany wpływ rozmiaru alfabetu, z którego znaków jest zbudowany tekst i wzorzec oraz wpływ długości tekstu i wzorca na czas wyszukiwania. Skupiono się na algorytmach, w których tekst i wzorzec są reprezentowane w postaci ciągu znaków, a samo wyszukiwanie odbywa się przez porównywanie symboli tekstu i wzorca. Wymieniony sposób wyszukiwania nie jest jedyny. Przykładowo, w algorytmie Rabina–Karpa w procesie wyszukiwania wzorca korzysta się z pojęcia stosowanego w teorii liczb, jakim jest przystawianie dwóch liczb modulo pewna trzecia liczba, a tekst zawierający k symboli jest interpretowany jako liczba k cyfrowa [7, 13]. Istnieją także algorytmy wyszukiwania wzorca, w których tekst jest przechowywany w zaawansowanych strukturach danych, jakimi są drzewa i tablice sufiksów [2, 9, 15].

Struktura niniejszej pracy przedstawia się następująco. W rozdziale 2 został omówiony problem wyszukiwania wzorca w tekście, zawarto w nim podstawowe definicje i stosowane oznaczenia. Rozdział 3 zawiera opis badanych algorytmów, a opis przeprowadzonych badań eksperymentalnych zawarto w rozdziale 4. Rozdział 5 jest podsumowaniem pracy.

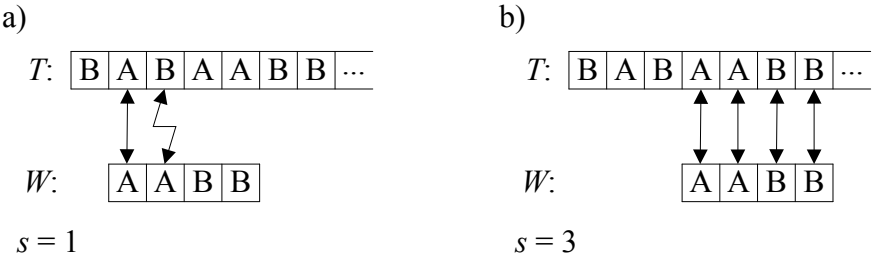
2. Problem wyszukiwania wzorca w tekście

W problemie wyszukiwania wzorca w tekście są dane dwie tablice symboli: tablica $T[1\ldots n]$ długości n nazywana tekstem i tablica $W[1\ \ldots\ m]$ długości m nazywana wzorcem. Obydwie tablice zawierają symbole należące do pewnego skończonego alfabetu Σ. Liczba symboli należących do alfabetu jest nazywana jego rozmiarem i oznaczana przez $|\Sigma|$. Tablice

te niekiedy też są nazywane słowami. Wzorzec W występuje w tekście T z przesunięciem s, tj. rozpoczyna się w tekście od pozycji $s + 1$, jeżeli zachodzi zależność (1).

$$0 \le s \le n - m \wedge \forall i \in \{1,...,m\} T[s+1] = W[i] \tag{1}$$

Jeżeli wzorzec występuje w tekście z przesunięciem s, to s jest nazywane poprawnym przesunięciem, w przeciwnym razie jest nazywane niepoprawnym przesunięciem. Problem wyszukiwania wzorca polega na znalezieniu wszystkich poprawnych przesunięć wzorca W w tekście T. Zwykle przyjmuje się, że $m \le n$, w przeciwnym przypadku nie istnieje poprawne przesunięcie. Przykład poprawnego przesunięcia został przedstawiony na rys. 1b, a jest nim przesunięcie $s = 3$, natomiast przykładem niepoprawnego przesunięcia jest $s = 1$ (rys. 1a).



Rys. 1. Przykład niepoprawnego przesunięcia i poprawnego przesunięcia wzorca W w tekście T
Fig. 1. An example of incorrect shift and correct shift of the pattern W in the text T

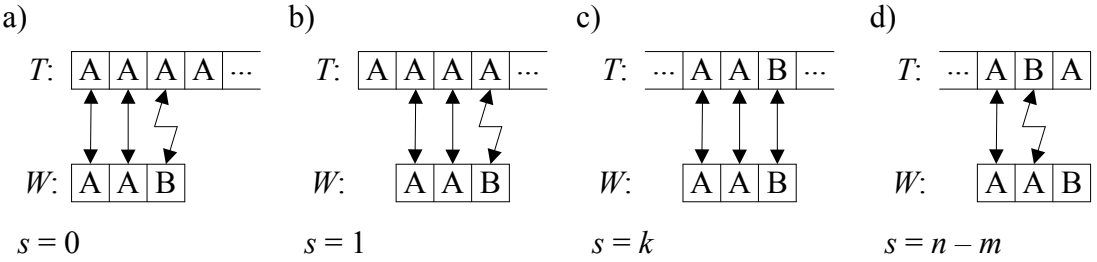
Dowolne słowo Z może być potraktowane jako złożenie dwóch innych słów X i Y o długościach równych $|X|$ i $|Y|$, co jest oznaczane przez $Z = XY$. Słowo Z zawiera symbole słowa X, za którymi występują symbole słowa Y i ma długość $|Z| = |X| + |Y|$. Słowo Y jest nazywane sufiksem słowa Z i jest oznaczane przez $Y \supset Z$, natomiast słowo X jest nazywane prefiksem słowa Z i jest oznaczane przez $X \subset Z$. Niech będzie dane słowo Z = abbcdd, to np. są prawdziwe zależności abb $\subset Z$ i dd $\supset Z$.

3. Algorytmy wyszukiwania wzorca w tekście

3.1. Algorytm Naive

Najprostszym algorytmem wyszukiwania wzorca w tekście jest algorytm naiwny (ang. *Naive*), należący do grupy algorytmów siłowych (ang. *brute force*) [4, 6, 7]. Wyszukiwanie wzorca odbywa się przez sprawdzanie kolejnych jego przesunięć, począwszy od przesunięcia $s = 0$ (rys. 2a). Dla danego przesunięcia s jest dokonywane porównywanie kolejnych symboli wzorca, tj. $W[1]$, $W[2]$, …, z odpowiednimi symbolami tekstu, tj. $T[s + 1]$, $T[s + 2]$, … . W momencie wykrycia niezgodności symboli wzorca i tekstu porównywanie jest przerywane i następuje sprawdzenie kolejnego przesunięcia, tj. $s = 1$ (rys. 2b). Operacja porównywania symboli kończy się w momencie znalezienia wzorca w tekście (rys. 2c). W tym przypadku

wzorzec występuje w tekście z przesunięciem $s = k$, tj. rozpoczyna się od pozycji $k + 1$. Podczas wyszukiwania zachodzi konieczność sprawdzenia $n - m + 1$ przesunięć (rys. 2d), stąd pesymistyczna złożoność czasowa algorytmu jest równa $O((n - m + 1)\, m)$.



Rys. 2. Wyszukiwanie wzorca W w tekście T z użyciem algorytmu naiwnego
Fig. 2. Searching of the pattern W in the text T using the naive algorithm

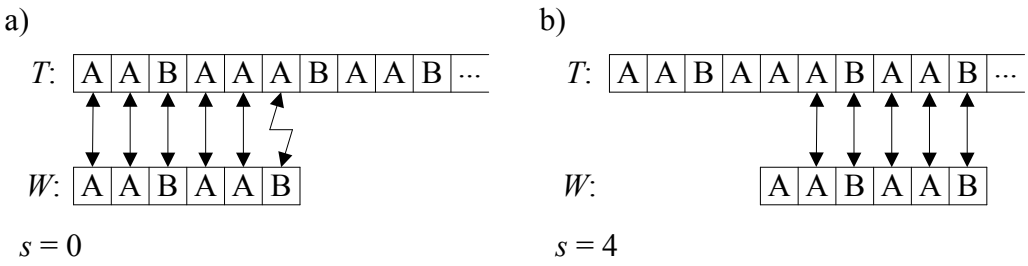
Algorytm wyszukiwania naiwnego został przedstawiony w postaci procedury *Naive*, która wyznacza zbiór S, zawierający wszystkie poprawne przesunięcia wzorca W w tekście T.

```
Wejście: T, W – tekst i wzorzec
         n, m – długość tekstu i wzorca
Wyjście: S – zbiór poprawnych przesunięć wzorca W w tekście T
 1:  procedure Naive(T, W, n, m, s)
 2:  begin
 3:    S := ∅;
 4:    for s := 0 to n - m do
 5:      i := 1;
 6:      while i <= m cand T[s + i] = W[i] do
 7:        i := i + 1;
 8:      end while
 9:      if i = m + 1 then          {znaleziono wzorzec}
10:        S := S ∪ {s};
11:      end if
12:    end for
16:  end
```

3.2. Algorytm Knutha–Morrisa–Pratta

W algorytmie naiwnym podczas analizy danego przesunięcia s nie korzysta się z informacji uzyskanej podczas analizy poprzedniego przesunięcia. Zatem po wykryciu niezgodności dla $s = 0$ (rys. 2a) analiza przesunięcia $s = 1$ jest rozpoczynana od pierwszego symbolu wzorca (rys. 2b). Z informacji uzyskanej podczas analizy przesunięcia $s = 0$ można zauważyć, że nie jest konieczne porównywanie symboli $W[1]$ i $T[2]$, gdyż są one zgodne, tak więc porównanie to jest nadmiarowe. Porównywanie symboli dla $s = 1$ należy więc rozpocząć od drugiego symbolu wzorca.

Kolejną nadmiarowością w algorytmie naiwnym jest analiza wszystkich przesunięć. Z analizy przesunięcia $s = 0$ (rys. 3a) wynika, że przesunięcia $s = 1$, 2 i 3 są niepoprawne, stąd nie jest konieczna ich analiza. Kolejnym przesunięciem, jakie należy poddać analizie, jest $s = 4$, przy czym analizę należy rozpocząć od drugiego symbolu wzorca (rys. 3b).



Rys. 3. Wyszukiwanie wzorca *W* w tekście *T* z użyciem algorytmu KMP
Fig. 3. Searching of the pattern *W* in the text *T* using the KMP algorithm

Opisana idea wyszukiwania wzorca została użyta w algorytmie Knutha–Morrissa–Pratta (KMP) opublikowanym w 1977 roku, opisanym procedurą *KnuthMorrisPratt* [4, 6, 7, 8, 14]. W algorytmie tym po wykryciu niezgodności symboli tekstu i wzorca jest ustalane kolejne przesunięcie, jakie zostanie poddane analizie oraz od którego symbolu wzorca będzie kontynuowane porównywanie (wiersze 6–8). Dla symboli tekstu porównywanie jest kontynuowane począwszy od symbolu, dla którego wykryto niezgodność. W przypadku porównania zgodnego, tj. zgodności symbolu tekstu i wzorca, symbol tekstu jest poddawany porównaniu co najwyżej jeden raz. W przypadku wykrycia niezgodności następuje przejście do analizy kolejnego przesunięcia. Wszystkich przesunięć jest $n - m + 1$, co oznacza, że liczba wszystkich porównań symboli wzorca i tekstu jest równa co najwyżej $2n - m + 1$. Czasowa złożoność algorytmu wyszukiwania jest więc równa $O(n)$[1].

```
Wejście: T, W – tekst i wzorzec
         n, m – długość tekstu i wzorca
Wyjście: S – zbiór poprawnych przesunięć wzorca W w tekście T
 1:  procedure KnuthMorrisPratt(T, W, n, m, s)
 2:  begin
 3:    ComputeNext(W, m, next);
 4:    S := ∅; j := 1;
 5:    for i := 1 to n do
 6:      while j > 0 cand T[i] ≠ W[i + j] do    {niezgodność symboli}
 7:        j := next[j];
 8:      end while
 9:      if T[i] = W[j + 1] then
10:        j := j + 1;
11:      end if
12:      if j = m then          {znaleziono wzorzec}
13:        S := S ∪ {i - m};    {wzorzec występuje z przesunięciem i - m}
14:        j := next[j];
15:      end if
16:    end for
17:  end
```

Podczas wyszukiwania wzorca indeks, od którego jest rozpoczynane porównywanie symboli wzorca dla analizowanego przesunięcia *s*, jest odczytywany z tablicy *next* (wiersze 7 i 14). Dlatego przed rozpoczęciem wyszukiwania wzorca w tekście konieczne jest jej wyznaczenie, co odbywa się w procedurze *ComputeNext* na podstawie analizy wzorca.

[1] W złożoności tej nie jest uwzględnione wyznaczanie tablicy *next*.

Wartości tablicy *next* są wyznaczane na podstawie zależności (1), w której wartość $p[j]$ jest określona funkcją prefiksową wzorca opisaną zależnością (2). Wartość $p[j]$ jest więc równa maksymalnej długości prefiksu $W[1 \ldots j-1]$ będącego sufiksem $W[1 \ldots j]$.

$$next[j] = \begin{cases} p[j] & \text{w przypadku kiedy } W[j] \neq W[p[j]] \\ next[p[j]] & \text{w przypadku kiedy } W[j] = W[p[j]] \end{cases} \tag{1}$$

$$p[j] = \{k : k < j \text{ i } W[1 \ldots k] \supset W[1 \ldots j]\} \tag{2}$$

Zmienna j w procedurze przyjmuje wartości nieujemne i jest zwiększana o 1 co najwyżej $m - 1$ razy. Operacja w wierszu 6 jest wykonywana więc co najwyżej $m - 1$ razy, a zatem złożoność czasowa procedury jest równa $O(m)$. Wyszukiwanie wzorca w tekście z użyciem algorytmu KMP odbywa się więc w czasie $O(n + m)$.

```
Wejście: W, m – wzorzec i jego długość
Wyjście: next – tablica indeksów
 1:  procedure ComputeNext(W, m, next)
 2:  begin
 3:    i := 1; j := 0; next[1] := 0;
 4:    repeat
 5:      while j > 0 cand W[i] ≠ W[j] do
 6:        j := next[j];
 7:      end while
 8:      i := i + 1; j := j + 1;
 9:      if W[i] = W[j] then
10:        next[i] := next[j];
11:      else
12:        next[i] := j;
13:      end if
14:    until i > m;
15:  end
```

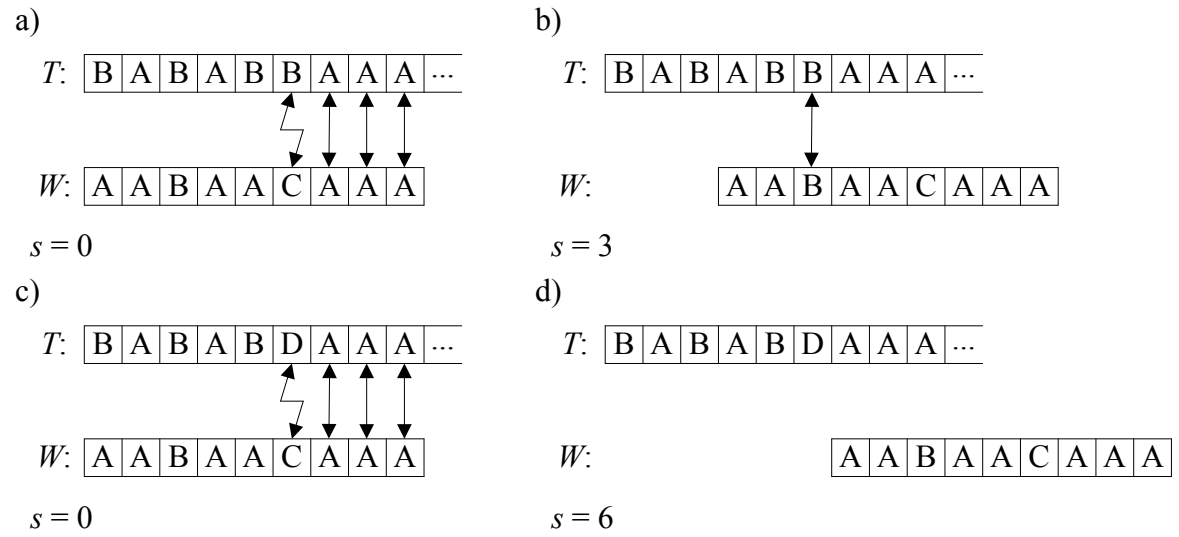
3.3. Algorytm Boyera–Moore'a

W algorytmie Boyera–Moore'a (BM) opublikowanym w 1977 roku, przedstawionym w postaci procedury *BoyerMoore*, wzorzec jest analizowany od ostatniego symbolu, a nie od pierwszego jak w algorytmach Naive i KMP [4, 5, 6, 7, 8]. Podobnie jak w algorytmie KMP jest dokonywane pominięcie niektórych przesunięć s, ponieważ bez konieczności ich analizy można stwierdzić, że są one nieprawidłowe. Wyszukiwanie wzorca odbywa się w kolejnych iteracjach, gdzie w przypadku pesymistycznym analizowanych jest $n - m + 1$ przesunięć (wiersze 5–16). Po wykryciu niezgodności symboli tekstu i wzorca (wiersz 10) następuje wybór kolejnego analizowanego przesunięcia s (wiersz 11), a o jego wyborze decyduje większa spośród dwóch wartości *delta*1 i *delta*2. W przypadku znalezienia wzorca w tekście kolejne analizowane przesunięcie jest ustalane na podstawie wartości *delta*2 (wiersz 14).

Wartość *delta*1, nazywana heurystyką niezgodności, jest tablicą o rozmiarze równym $|\Sigma|$. W przypadku wykrycia niezgodności symboli $T[s + i]$ oraz $W[i]$, na podstawie wartości $delta1[T[s + i]]$ jest określana wartość, o jaką należy zwiększyć przesunięcie s tak, aby

wystąpiła zgodność symbolu $T[s + i]$ z symbolem wzorca, przy założeniu że wzorzec zawiera symbol $T[s + i]$.

```
Wejście: T, W – tekst i wzorzec
         n, m – długość tekstu i wzorca
         Σ - alfabet, z którego symboli składa się tekst i wzorzec
Wyjście: S – zbiór poprawnych przesunięć wzorca W w tekście T
 1:  procedure BoyerMoore(T, W, n, m, Σ, s)
 2:  begin
 3:    ComputeDelta1(W, m, Σ, delta1); ComputeDelta2(W, m, delta2);
 4:    S := ∅; s := 0;
 5:    while s <= n – m do
 6:      i := m;
 7:      while i > 0 cand W[i] = T[s + i] do {zgodność symbolu tekstu i wzorca}
 8:        i := i - 1;
 9:      end while
10:      if i > 0 then       {nie znaleziono wzorca}
11:        s := s + max(delta1[T[s + i]] + i - m, delta2[i]);
12:      else                {znaleziono wzorzec}
13:        S := S ∪ {s};
14:        s := s + delta2[1];
15:      end if
16:    end while
19:  end
```



Rys. 4. Przykład działania heurystyki niezgodności
Fig. 4. An example of working the bad character heuristics

Po wykryciu niezgodności symboli $T[6]$ = B i $W[6]$ = C (rys. 4a), zgodnie z heurystyką niezgodności kolejnym analizowanym przesunięciem jest $s = 3$, dla którego jest zapewniona zgodność symbolu $T[6]$ z symbolem wzorca (rys. 4b). Przesunięcia $s = 1$ i 2 zostają pominięte z powodu niezgodności symbolu $T[6]$ z symbolami wzorca. Drugim przypadkiem jest wykrycie niezgodności symboli $T[s + i]$ i $W[i]$, przy czym symbol $T[s + i]$ nie występuje we wzorcu. Ponieważ symbol $T[6]$ = D nie występuje we wzorcu (rys. 4c), należy przesunąć wzorzec tak, aby pominąć niezgodny symbol $T[6]$. Kolejnym analizowanym przesunięciem jest więc $s = 6$. W najlepszym przypadku niezgodność może wystąpić dla symboli $T[s + m]$

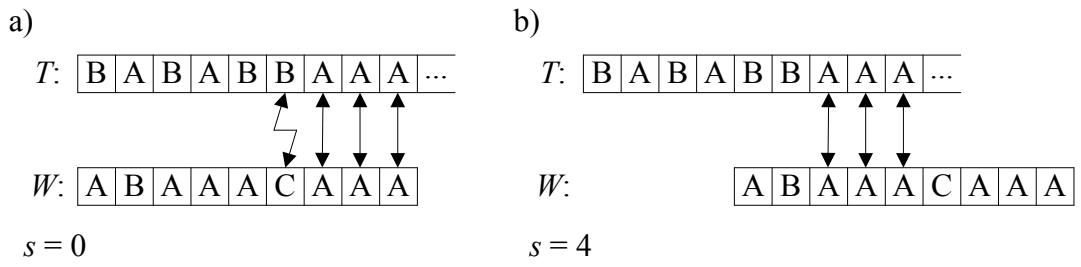
i $W[m]$, gdzie symbol $T[s + m]$ nie należy do wzorca, dzięki czemu przesunięcie s można zwiększyć o m. Daje to przewagę porównywania rozpoczynanego od ostatniego symbolu wzorca nad porównywaniem symboli począwszy od pierwszego symbolu.

Heurystyka niezgodności *delta*1 jest wyznaczana w procedurze *ComputeDelta*1 na podstawie równania (3), a w celu jej wyznaczenia dla każdego symbolu $c \in W$ należy określić indeks i jego ostatniego wystąpienia we wzorcu, przy czym nie jest uwzględniane wystąpienie na pozycji m (wiersze 6–8). Dla symboli, które nie należą do wzorca, wartość *delta*1 jest równa m (wiersz 4). W celu wyznaczenia heurystyki niezgodności konieczne jest przeanalizowanie wszystkich symboli alfabetu (wiersze 3–5) i wzorca (wiersze 6–8), co odbywa się w czasie $O(|\Sigma| + m)$.

$$delta1[c] = \begin{cases} \min\{m - i : 1 \le i < m \wedge W[i] = c\} & \text{kiedy } c \in W \\ m & \text{kiedy } c \notin W \end{cases} \tag{3}$$

```
Wejście: W – wzorzec
         m – długość wzorca
         Σ - alfabet, z którego symboli składa się tekst i wzorzec
Wyjście: delta1 – tablica wartości heurystyki niezgodności
 1:  procedure ComputeDelta1(W, m, Σ, delta1)
 2:  begin
 3:    for each c ∈ Σ do
 4:      delta1[c] := m;
 5:    end for
 6:    for i := 1 to m - 1 do
 7:      delta1[W[i]] := m - i;
 8:    end for
 9:  end
```

Drugą użytą heurystyką jest heurystyka dobrego sufiksu wyznaczana przez procedurę *ComputeDelta*2, a jej wartości są zawarte w tablicy *delta*2 o rozmiarze m. Dla przesunięcia s, *delta*2[j] określa minimalną wartość, o jaką należy zwiększyć przesunięcie s, aby zachodziła zgodność symboli $T[s + j + 1]$, …, $T[s + m]$ z symbolami wzorca. Przykładowo, po wykryciu niezgodności symboli $T[6]$ = B i $W[6]$ = C (rys. 5a), na podstawie heurystyki dobrego sufiksu kolejnym analizowanym przesunięciem jest s = 4, dla którego zachodzi zgodność symboli $T[7]$, …, $T[9]$ z symbolami wzorca (rys. 5b).



Rys. 5. Przykład działania heurystyki dobrego sufiksu
Fig. 5. An example of working the good suffix heuristics

Heurystyka dobrego sufiksu jest wyznaczana w procedurze *ComputeDelta*2 na podstawie funkcji prefiksowej określonej równaniem (2). Korzystając z procedury *ComputePrefix*, są wyznaczane w czasie $O(m)$ i zapamiętywane w tablicach p i p', wartości funkcji prefiksowej dla wzorca W i W', będącego odwróconym wzorcem W, tj. $W'[i] = W[m - i + 1]$ dla $i = 1, \ldots, m$ (wiersz 3). Heurystyka dobrego sufiksu *delta*2 jest wyznaczana w czasie $O(m)$, zgodnie z równaniem (4).

$$delta2[j] = min(\{m - p[m]\} \cup \{k - p'[k]: 1 \le k \le m \text{ i } j = m - p'[k]\}) \tag{4}$$

```
Wejście: W – wzorzec
         m – długość wzorca
Wyjście: delta2 – tablica wartości heurystyki dobrego sufiksu
 1:  procedure ComputeDelta2(W, m, delta2)
 2:  begin
 3:    ComputePrefix(W, m, p); ComputePrefix(W', m, p');
 4:    for i := 1 to m do
 5:      delta2[i] := m - p[m];
 6:    end for
 7:    for k := 1 to m do
 8:      if delta2[m - p'[k]] > k - p'[k] then
 9:        delta2[m - p'[k]] > k - p'[k];
10:      end if
11:    end for
12:  end

Wejście: W – wzorzec
         m – długość wzorca
Wyjście: p – tablica zawierająca wartości funkcji prefiksowej
 1:  procedure ComputePrefix(W, m, p)
 2:  begin
 3:    p[1] := 0; j := 0;
 4:    for i := 2 to m do
 5:      while j > 0 cand W[j + 1] ≠ W[i] do
 6:        j := p[j];
 7:      end while
 8:      if W[j + 1] = W[i] then
 9:        j := j + 1;
10:      end if
11:      p[i] := j;
12:    end for
13:  end
```

Czas działania algorytmu BM zależy od czasu wyznaczania heurystyk niezgodności i dobrego sufiksu, co odbywa się odpowiednio w czasie $O(|\Sigma| + m)$ i $O(m)$, oraz czasu wyszukiwania samego wzorca. Wyszukiwanie wzorca w wierszach 5–16 procedury *BoyerMoore* odbywa się przez analizę każdego poprawnego przesunięcia s w pesymistycznym czasie równym $O((n - m + 1)\, m)$. Zatem złożoność algorytmu BM w pesymistycznym przypadku jest równa $O((n - m + 1)\, m + |\Sigma|)$.

3.4. Algorytm Horspoola

Opublikowany w 1980 roku algorytm Horspoola, nazywany też algorytmem Boyera-Moore'a–Horspoola, jest uproszczeniem algorytmu BM [4, 6, 12]. W algorytmie tym po

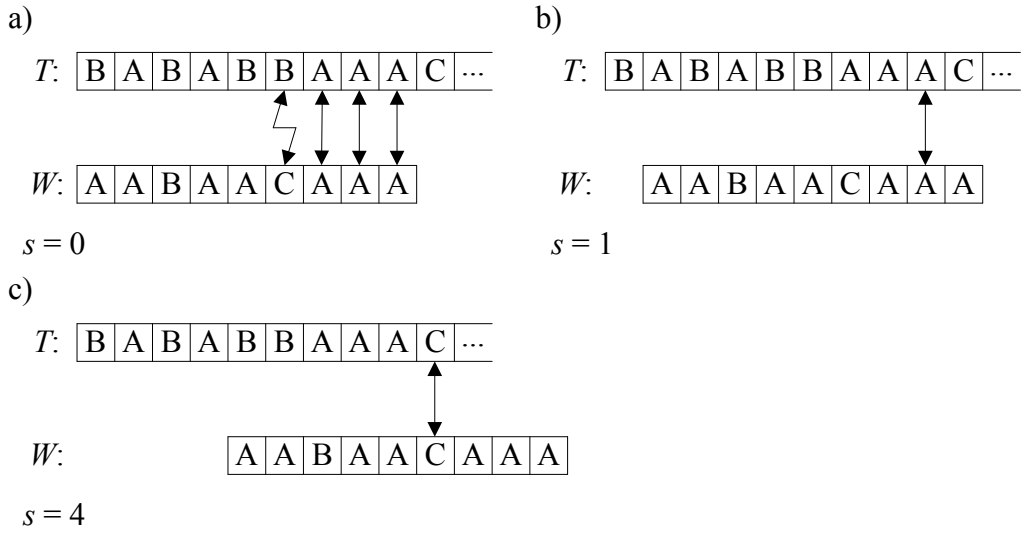
wykryciu niezgodności symboli tekstu i wzorca o wyborze kolejnego analizowanego przesunięcia *s* (wiersz 11 procedury *BoyerMoore*) decyduje wyłącznie heurystyka niezgodności *delta*1 (heurystyka dobrego sufiksu *delta*2 nie jest wyznaczana). Ponadto, analizowane przesunięcie jest ustalane na podstawie symbolu tekstu porównywanego z ostatnim symbolem wzorca, tj. *T*[*s* + *m*]. Zatem operacja określenia przesunięcia *s* ma postać (w ten sposób jest ustalane przesuniecie w przypadku wykrycia niezgodności symboli tekstu i wzorca oraz w wierszu 14 w przypadku znalezienia wzorca):

```
11:         s := s + delta1[T[s + m]];
```

Jedyną różnicą podczas wyszukiwania wzorca jest sposób wyboru przesunięcia *s* poddawanego analizie oraz brak wyznaczania heurystyki *delta*2 w wierszu 3 procedury *BoyerMoore*, stąd treść algorytmu Horspoola nie zostanie przedstawiona w postaci oddzielnej procedury. Złożoność czasowa algorytmu w przypadku pesymistycznym jest równa $O((n - m + 1)\, m + |\Sigma|)$.

3.5. Algorytm Quick Search

Algorytm Quick Serach opisany procedurą *QuickSearch*, będący modyfikacją algorytmu Horspoola, został opublikowany w 1990 roku [1, 6, 18]. W algorytmie tym wartość przesunięcia *s* jest ustalana także na podstawie heurystyki niezgodności, przy czym sposób jej użycia oraz wyznaczania są inne niż w przypadku algorytmu Horspoola. Heurystyka ta zostanie więc oznaczona przez *delta*3.



Rys. 6. Wyszukiwanie wzorca *W* w tekście *T* z użyciem algorytmów Horspoola i QuickSearch
Fig. 6. Searching of the pattern *W* in the text *T* using the Horspool and the QuickSearch algorithms

W algorytmie Horspoola po wykryciu niezgodności symboli *T*[6] = B i *W*[6] = C (rys. 6a) kolejnym analizowanym przesunięciem jest *s* = 1 (rys. 6b). Jak można zauważyć, po

wykryciu niezgodności dla przesunięcia s, symbol $T[s + m + 1]$ jest porównywany z symbolem wzorca, a w tym przypadku jest to symbol $T[10]$ = C. Zatem w algorytmie Quick Search wartość kolejnego analizowanego przesunięcia jest ustalana na podstawie symbolu $T[s + m + 1]$, w omawianym przykładzie będzie nim symbol $T[10]$ = C, a kolejnym analizowanym przesunięciem jest s = 4 (rys. 6c). Zatem jedyną różnicą w porównaniu z algorytmem Horspoola jest wyznaczenie wartości kolejnego analizowanego przesunięcia s, co odbywa się za pomocą operacji:

```
11:         s := s + delta3[T[s + m + 1]];
```

Heurystyka niezgodności *delta*3 jest określona równaniem (5) i jest wyznaczana za pomocą procedury *ComputeDelta*3. W odróżnieniu od heurystyki *delta*1, przy jej wyznaczaniu jest uwzględniany ostatni symbol wzorca (wiersze 6–8), a dla symboli nienależących do wzorca przyjmuje ona wartość $m + 1$ (wiersz 4).

$$delta3[c] = \begin{cases} \min\{m - i + 1 : 1 \le i \le m \wedge W[i] = c\} & \text{kiedy } c \in W \\ m + 1 & \text{kiedy } c \notin W \end{cases} \tag{5}$$

```
Wejście: W - wzorzec
         m - długość wzorca
         Σ - alfabet, z którego symboli składa się tekst i wzorzec
Wyjście: delta3 - tablica wartości heurystyki niezgodności
 1:  procedure ComputeDelta3(W, m, Σ, delta1)
 2:  begin
 3:    for each c ∈ Σ do
 4:      delta3[c] := m + 1;
 5:    end for
 6:    for i := 1 to m do
 7:      delta3[W[i]] := m - i + 1;
 8:    end for
 9:  end
```

Modyfikacja sposobu wyznaczania heurystyki *delta*3 nie wpływa na złożoność czasową, tak więc jest ona taka sama jak w przypadku wyznaczania heurystyki *delta*1, tj. $O(|\Sigma| + m)$. Sam algorytm wyszukiwania różni się jedynie wierszem 11, tak więc jego złożoność czasowa w przypadku pesymistycznym jest równa $O((n - m + 1)\, m + |\Sigma|)$.

3.6. Algorytm Smitha

Opublikowany w 1991 roku algorytm Smitha jest połączeniem algorytmów Horspoola i Quick Search [6, 17]. Analizowane przesunięcie s jest ustalane na podstawie większej z dwóch heurystyk niezgodności *delta*1 i *delta*3:

```
11:         s := s + max(delta1[T[s + m]], delta3[T[s + m + 1]]);
```

Wyznaczanie heurystyk niezgodności odbywa się za pomocą procedur *ComputeDelta*1 i *ComputeDelta*3 w czasie $O(|\Sigma| + m)$. Wyszukiwanie wzorca odbywa się w identyczny

sposób jak w algorytmach Horspoola i Quicksearch, a zatem pesymistyczna złożoność czasowa algorytmu Smitha jest równa $O((n - m + 1)\, m + |\Sigma|)$.

3.7. Algorytm Raity

Algorytm Raity [6, 16], opublikowany w 1992 roku i przedstawiony w postaci procedury *Raita*, jest modyfikacją algorytmu Horspoola. Przed rozpoczęciem porównywania symboli tekstu i wzorca (wiersze 8–10), w porównaniu z algorytmem Horspoola, jest dokonywane porównanie ostatniego, pierwszego oraz środkowego symbolu wzorca z symbolami tekstu (wiersz 6). Wyłącznie w przypadku zgodności wymienionych trzech symboli następuje przejście do porównywania pozostałych symboli wzorca z symbolami tekstu. Z racji wykonania porównania w wierszu 6, podczas porównywania symboli wzorca z symbolami tekstu (wiersze 8–10), nie ma już konieczności porównywania pierwszego i ostatniego symbolu wzorca z symbolami tekstu. Środkowy symbol wzorca jest porównywany ponownie, gdyż wymagałoby to sprawdzenia dodatkowych warunków. Pesymistyczna złożoność czasowa algorytmu Raity jest równa $O((n - m + 1)\, m + |\Sigma|)$.

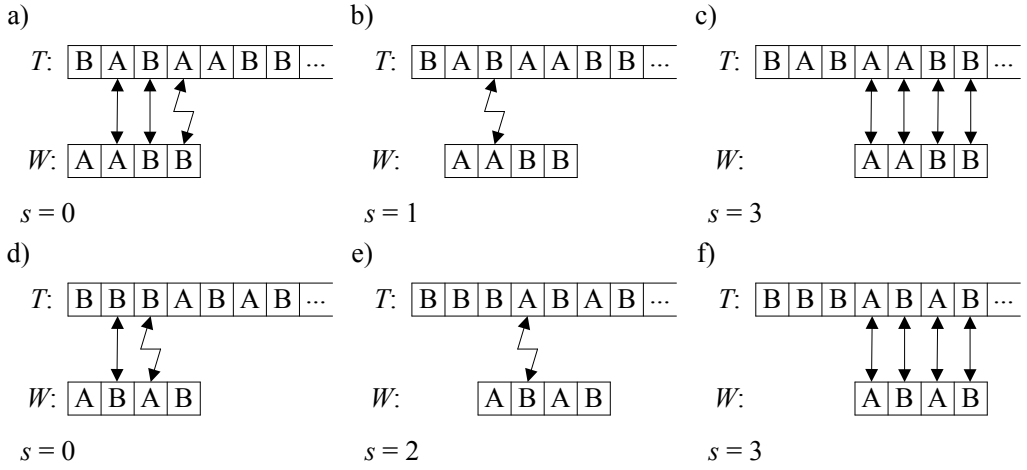
```
Wejście: T, W – tekst i wzorzec
         n, m – długość tekstu i wzorca
         Σ - alfabet, z którego symboli składa się tekst i wzorzec
Wyjście: S – zbiór poprawnych przesunięć wzorca W w tekście T
 1:  procedure Raita(T, W, n, m, s)
 2:  begin
 3:    ComputeDelta1(W, m, Σ, delta1);
 4:    S := ∅; s := 0;
 5:    while s <= n - m do
 6:      if W[m] = T[s + m] cand W[1] = T[s + 1] cand
             W[m / 2] = T[s + 1 + m / 2] then
 7:        i := m - 1;
 8:        while i > 1 cand W[i] = T[s + i] do    {zgodność symboli}
 9:          i := i - 1;
10:        end while
11:        if i = 1 then       {znaleziono wzorzec}
12:          S := S ∪ {s};
13:        end if
14:      end if
15:      s := s + delta1[T[s + m]];
16:    end while
17:  end
```

3.8. Algorytm Not So Naive

Przedstawiony w postaci procedury *NotSoNaive* algorytm, który został opublikowany w 1992 roku, jest modyfikacją algorytmu Naive [6, 10, 11]. W odróżnieniu od algorytmu Naive przesunięcie jest zwiększane o jedną lub dwie pozycje. Ponadto, porównywanie symboli tekstu i wzorca jest rozpoczynane od symbolu $W[2]$, a symbol $W[1]$ jest porównywany jako ostatni wyłącznie w przypadku wykrycia zgodności symboli $W[2]$, …, $W[m]$.

```
Wejście: T, W – tekst i wzorzec
         n, m – długość tekstu i wzorca
Wyjście: S – zbiór poprawnych przesunięć wzorca W w tekście T
 1:  procedure NotSoNaive(T, W, n, m, s)
 2:  begin
 3:    if W[1] = W[2] then
 4:      s1 := 2; s2 := 1;
 5:    else
 6:      s1 := 1; s2 := 2;
 7:    end if
 8:    S := ∅; s := 0;
 9:    while s <= n - m do
10:      if W[2] ≠ T[s + 2] then
11:        s := s + s1;
12:      else
13:        i := 2;
14:        while i <= m cand T[s + i] ≠ W[i] do
15:          i := i + 1;
16:        end while
17:        if i = m + 1 and T[s + 1] = W[1] then        {znaleziono wzorzec}
18:          S := S ∪ {s};
19:        end if
20:        s := s + s2;
21:      end if
22:    end while
23:  end
```

W celu prześledzenia działania algorytmu i sposobu wyboru przesunięcia poddawanego analizie należy rozpatrzyć dwa przypadki. W pierwszym z nich symbole $W[1]$ i $W[2]$ są identyczne, a w drugim przypadku są one różne. Dodatkowo w każdym z przypadków należy rozpatrzyć sytuację, kiedy zachodzi zgodność pierwszego porównywanego symbolu wzorca, tj. $W[2]$, z symbolem tekstu oraz kiedy tej zgodności nie ma.



Rys. 7. Wyszukiwanie wzorca W w tekście T z użyciem algorytmu Not So Naive
Fig. 7. Searching of the pattern W in the text T using the Not So Naive algorithm

Jako pierwszy zostanie rozpatrzony przypadek, kiedy $W[1] = W[2]$. Po wykryciu niezgodności symboli $W[4]$ i $T[4]$ (rys. 7a), przy wyborze kolejnego analizowanego przesunięcia korzysta się z faktu zgodności symboli $W[2]$ i $T[2]$. Z tego powodu kolejnym analizowanym przesunięciem jest $s = 1$ (rys. 7b). Dla $s = 1$ nie zachodzi zgodność symboli $W[2]$ i $T[3]$.

Ponieważ $W[1] = W[2]$, więc przesunięcie s można zwiększyć o 2 i przejść do analizy przesunięcia $s = 3$, tak aby pominąć porównanie symbolu $T[3]$ z $W[1]$ (rys. 7c).

Drugim przypadkiem jest sytuacja, w której $W[1] \neq W[2]$. Na rys. 7d przedstawiono sytuację, kiedy nie zachodzi zgodność symboli $W[3]$ i $T[3]$, przy czym zachodzi zgodność symboli $W[2]$ i $T[2]$. Ponieważ $W[1] \neq W[2]$, dlatego zwiększenie przesunięcia s o jedną pozycję spowodowałoby porównanie niezgodnych symboli $W[1]$ i $T[2]$. Zatem kolejnym analizowanym przesunięciem jest $s = 2$ (rys. 7e), dla którego nie zachodzi zgodność symboli $W[2]$ i $T[4]$. W tym przypadku, na podstawie zależności $W[1] \neq W[2]$, przesuniecie jest zwiększane o jedną pozycję i następuje przejście do analizy przesunięcia $s = 3$ (rys. 7f).

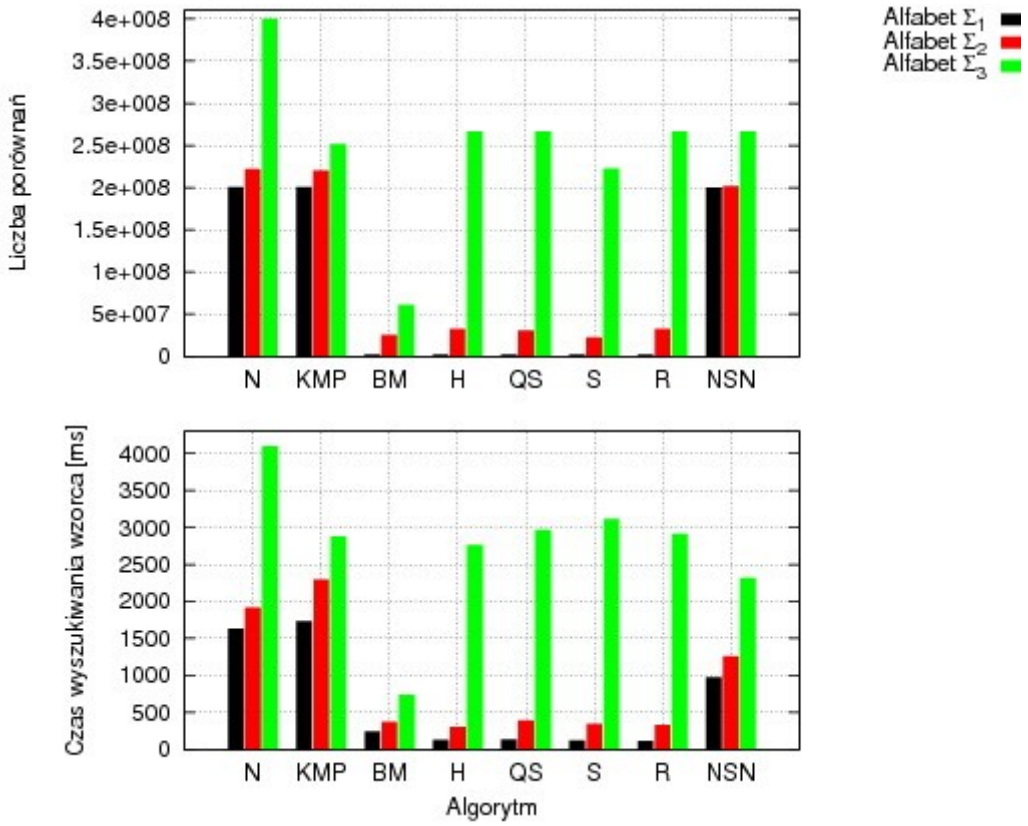
Przed rozpoczęciem wyszukiwania jest dokonywane sprawdzenie zgodności symboli $W[1]$ i $W[2]$ (wiersze 3–7 procedury *NotSoNaive*). Na tej podstawie są ustalane i zapamiętywane w $s1$ i $s2$ wartości, o jakie jest zwiększane przesunięcie s. Podczas analizy przesunięcia s (wiersze 10–21) jako pierwsza jest sprawdzana w wierszu 10 zgodność symboli $W[2]$ i $T[s+2]$. W przypadku wykrycia niezgodności symboli analiza przesunięcia s jest przerywana i na podstawie wartości $s1$ następuje wybór kolejnego przesunięcia (wiersz 11). W przeciwnym przypadku są porównywane pozostałe symbole wzorca z symbolami tekstu (wiersze 13–19). Po wykryciu niezgodności symboli lub po znalezieniu wzorca kolejne analizowane przesunięcie s jest ustalane na podstawie wartości $s2$ (wiersz 20).

Ustalenie wartości $s1$ i $s2$ odbywa się w czasie stałym, a podczas wyszukiwania wzorca maksymalnie zostanie przeanalizowanych $n - m + 1$ przesunięć (rys. 2d), dlatego pesymistyczna złożoność czasowa algorytmu jest równa $O((n - m + 1)\, m)$.

4. Wyniki badań eksperymentalnych

Badania eksperymentalne przeprowadzono z użyciem algorytmów opisanych w rozdziale 3, a ich celem była ich ocena ze względu na czas wyszukiwania wzorca w tekście oraz ocena wpływu rozmiaru alfabetu na czas wyszukiwania. W badaniach użyto pięciu różnych tekstów o długościach n równych odpowiednio: 10^7; $5 \cdot 10^7$; 10^8; $1{,}5 \cdot 10^8$ i $2 \cdot 10^8$ znaków, dla których dokonano wyszukiwania sześciu różnych wzorców o długościach m równych: 2, 5, 10, 20, 50 i 100 znaków. Użyte teksty i wzorce były złożone z symboli trzech różnych alfabetów różniących się rozmiarem:

- alfabetu Σ_1 zawierającego wszystkich 256 znaków z rozszerzonej tablicy kodu ASCII,
- alfabetu Σ_2 zawierającego 10 znaków będących cyframi, tj. $\Sigma_2 = \{\text{‘0’}, \ldots, \text{‘9’}\}$,
- alfabetu Σ_3 zawierającego 2 znaki będące cyframi, tj. $\Sigma_2 = \{\text{‘0’}, \text{‘1’}\}$.



Rys. 8. Wpływ rozmiaru alfabetu na czas wyszukiwania wzorca o długości $m = 100$ w tekście o długości $n = 2 \cdot 10^8$ i liczbę wykonywanych porównań

Fig. 8. An influence of the size of the alphabet on the time of searching the pattern of the length $m = 100$ in the text of the length $n = 2 \cdot 10^8$ and the number of comparisons

W opisie wyników zostały użyte następujące skróty nazw algorytmów:

- N – Naive,
- KMP – Knutha–Morrisa–Pratta,
- BM – Boyera–Moore'a,
- H – Horspoola,
- QS – Quick Search,
- S – Smitha,
- R – Raity,
- NSN – Not So Naive.

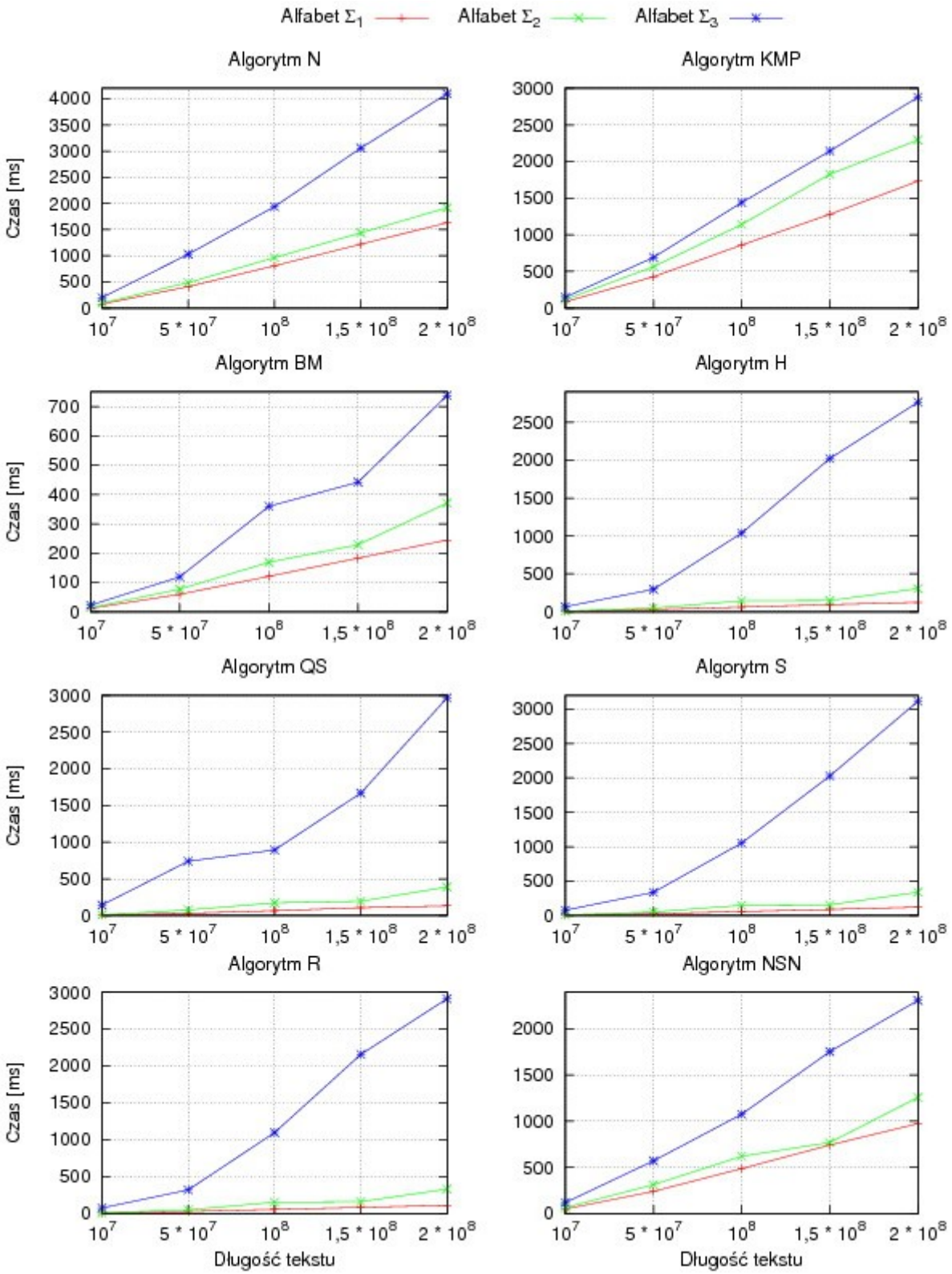
Na rys. 8 przedstawiono wykres zawierający liczbę wykonanych porównań oraz czas wyszukiwania wzorca o długości $m = 100$ w tekście o długości $n = 2 \cdot 10^8$ z użyciem poszczególnych algorytmów. Dla każdego algorytmu największy czas wyszukiwania wzorca został uzyskany dla alfabetu Σ_3, a najmniejszy czas dla alfabetu Σ_1. Prawdopodobieństwo zgodności porównywanego symbolu tekstu i wzorca jest równe $1 / |\Sigma|$, a więc maleje ono wraz ze wzrostem rozmiaru alfabetu. Im mniejsze prawdopodobieństwo zgodności symboli, tym mniejsza jest liczba wykonywanych porównań podczas sprawdzania danego przesunięcia

s, gdyż istnieje możliwość wczesnego wykrycia niepoprawnego przesunięcia. Z tego powodu liczba wykonywanych porównań maleje wraz ze wzrostem rozmiaru alfabetu, co jest widoczne na wykresie przedstawionym na rys. 8. Liczba wykonywanych porównań wpływa na czas wyszukiwania wzorca, stąd dla każdego algorytmu najmniejszy czas wyszukiwania wzorca został uzyskany dla alfabetu Σ_1, a największy dla alfabetu Σ_3.

Dokonując porównania czasów wyszukiwania wzorców z użyciem poszczególnych algorytmów, dla alfabetu Σ_3 najmniejszy czas wyszukiwania został uzyskany dla algorytmu BM, a największy zgodnie z oczekiwaniami dla algorytmu N o największej złożoności czasowej. Wpływ długości wzorca i tekstu na czas wyszukiwania wzorca z użyciem poszczególnych algorytmów został przedstawiony na rys. 9 i 10. Na rys. 9 przedstawiono zależność czasu wyszukiwania w tekście wzorca o długości $m = 100$ od długości tekstu. Podczas wyszukiwania zostały użyte teksty i wzorce każdego z trzech alfabetów Σ_1, Σ_2 i Σ_3. Dla każdego algorytmu czas rośnie wraz z długością tekstu. Ponadto, podobnie jak w przypadku wyników przedstawionych na rys. 8, dla każdego algorytmu największy czas wyszukiwania został uzyskany dla alfabetu Σ_3, a najmniejszy czas dla alfabetu Σ_1.

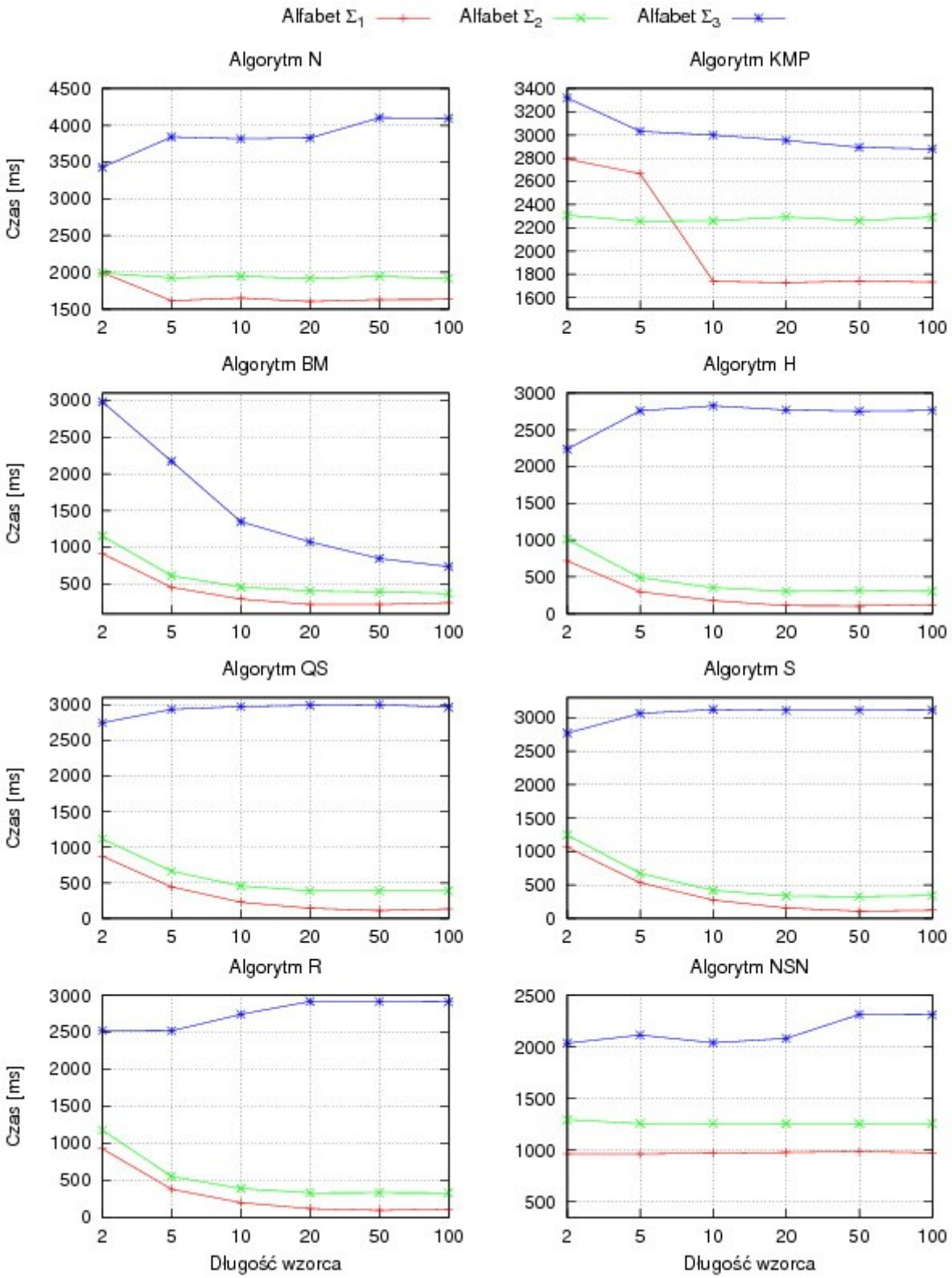
Porównanie czasów wyszukiwania z użyciem każdego z algorytmów zostało przedstawione także na rys. 10, zawierającym zależność czasu wyszukiwania od długości wzorca. Podczas wyszukiwania użyto wzorców oraz tekstów o długości $n = 2 \cdot 10^8$ zawierających symbole każdego z trzech alfabetów. Podobnie jak w poprzednim przypadku największy czas wyszukiwania został uzyskany dla alfabetu Σ_3, a najmniejszy dla alfabetu Σ_1. Wyjątkiem jest algorytm KMP, dla którego najmniejszy czas wyszukiwania wzorców o długościach $m = 2$ i 5 uzyskano dla alfabetu Σ_2. Algorytm BM jest jedynym algorytmem, dla którego czas wyszukiwania wzorca maleje wraz ze wzrostem jego długości dla każdego z trzech badanych alfabetów. W przypadku algorytmów H, QS, S i R czas wyszukiwania wzorca maleje wraz ze wzrostem długości wzorca wyłącznie dla alfabetów Σ_2 i Σ_3, podczas gdy dla alfabetu Σ_1 jest on najmniejszy dla wzorca o długości $m = 2$, a dla kolejnych wzorców nieznacznie on wzrasta. Ponadto, dla wzorca o długości $m = 100$ czas wyszukiwania dla alfabetu Σ_1 jest ok. 6 razy większy od czasu wyszukiwania dla pozostałych dwóch alfabetów, podczas gdy dla algorytmu BM jest on tylko ok. 2-krotnie większy.

Wyniki przedstawione na rys. 9 i 10 ilustrują wpływ rozmiaru alfabetu na czas wyszukiwania wzorca i zgodnie z nimi czas wyszukiwania, niezależnie od algorytmu, maleje wraz ze wzrostem rozmiaru alfabetu. Fakt ten wynika z prawdopodobieństwa zgodności porównywanego symbolu tekstu i wzorca, które jest odwrotnie proporcjonalne do rozmiaru alfabetu. Czas wyszukiwania rośnie wraz z liczbą wykonywanych porównań, a liczba wykonywanych porównań rośnie wraz ze wzrostem prawdopodobieństwa, a co za tym idzie maleje wraz ze wzrostem rozmiaru alfabetu.



Rys. 9. Wpływ długości tekstu na czas wyszukiwania wzorca o długości m = 100 w tekście, wykresy dla algorytmów

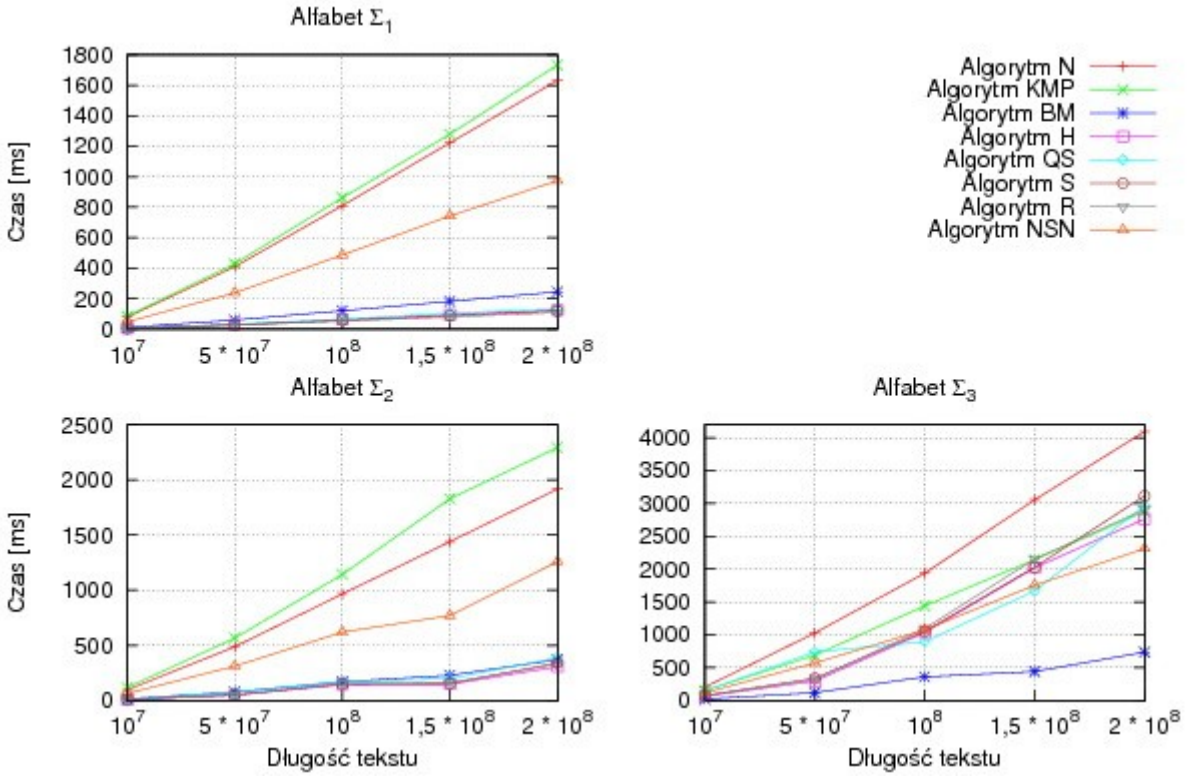
Fig. 9. An influence of the length of the text on the time of searching the pattern of the length m = 100 in the text, charts for algorithms



Rys. 10. Wpływ długości wzorca na czas wyszukiwania wzorca w tekście o długości $n = 2 \cdot 10^8$ znaków, wykresy dla algorytmów

Fig. 10. An influence of the length of the pattern on the time of searching the pattern in the text of the length $n = 2 \cdot 10^8$, charts for algorithms

Wyniki zaprezentowane na rys. 9 i 10 przedstawiono także na rys. 11 i 12, zawierających dla poszczególnych alfabetów czasy obliczeń wyszukiwania wzorca z użyciem każdego algorytmu. Na rys. 11 przedstawiono wpływ długości tekstu na czas wyszukiwania wzorca o długości m = 100. Dla każdego alfabetu czas wyszukiwania wzorca z użyciem każdego algorytmu rośnie wraz ze wzrostem długości tekstu. Dla alfabetów Σ_1 i Σ_2 największe czasy zostały uzyskane dla algorytmów N, KMP i NSN, które są zdecydowanie większe od czasów wyszukiwania wzorca z użyciem pozostałych algorytmów. Zbliżone są natomiast czasy wyszukiwania z użyciem algorytmów BM, H, QS, S i R, przy czym nieznacznie większy czas dla alfabetu Σ_1 uzyskano w przypadku algorytmu BM. Dokonując porównania czasów wyszukiwania dla alfabetu Σ_3, można zauważyć zdecydowaną przewagę algorytmu BM, dla którego jest on ok. 3 razy mniejszy od czasu wyszukiwania z użyciem algorytmu NSN.

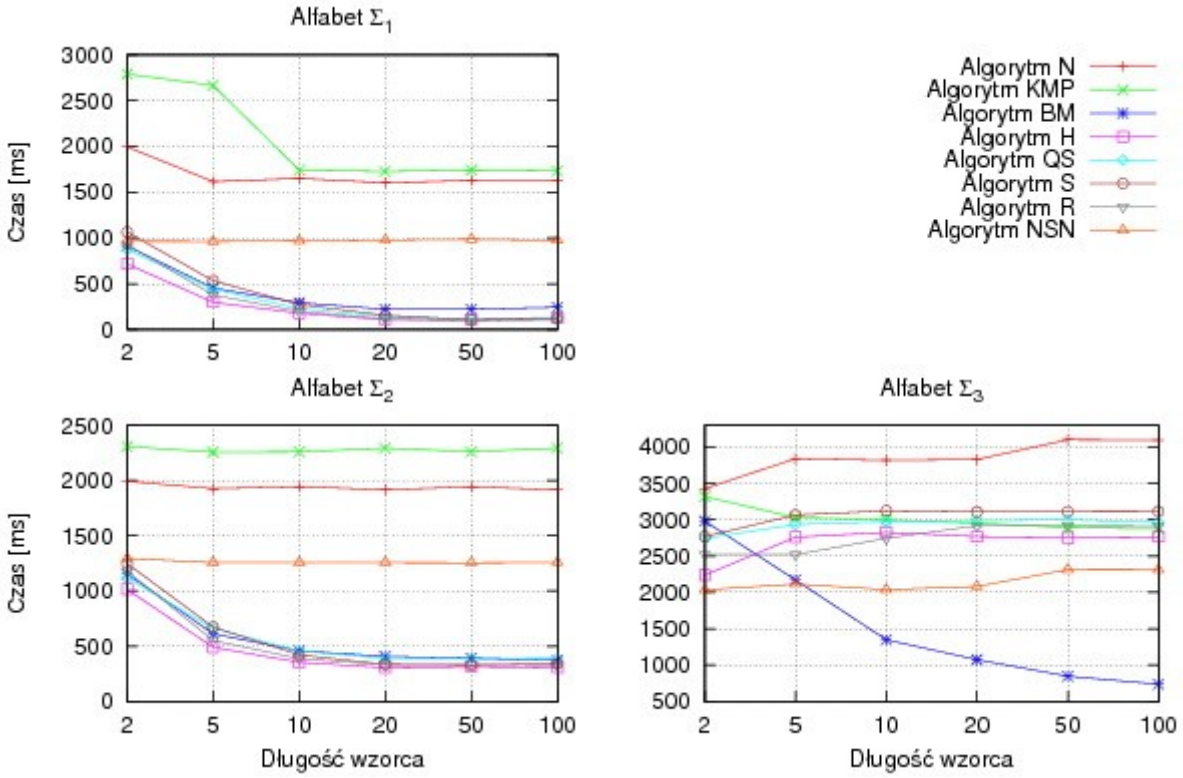


Rys. 11. Wpływ długości tekstu na czas wyszukiwania wzorca o długości m = 100 w tekście, wykresy dla alfabetów

Fig. 11. An influence of the length of the text on the time of searching the pattern of the length m = 100 in the text, charts for alphabets

Na rys. 12 dla każdego alfabetu przedstawiono wpływ długości wzorca na czas jego wyszukiwania z użyciem każdego z badanych algorytmów. Zależności między uzyskanymi wynikami są analogiczne do zależności między wynikami przedstawionymi na rys. 11. Dla alfabetów Σ_1 i Σ_2 najmniejsze czasy wyszukiwania wzorca zostały uzyskane dla algorytmów BM, H, QS, S i R, przy czym w przypadku algorytmu BM i alfabetu Σ_1 jest on nieznaczenie większy od czasów wyszukiwania wzorca z użyciem pozostałych wymienionych algorytmów.

Dla każdego z wymienionych algorytmów, w przypadku alfabetów Σ_1 i Σ_2, czas wyszukiwania wzorca maleje wraz ze wzrostem długości wzorca. W przypadku alfabetu Σ_3 algorytm BM jest jedynym algorytmem, dla którego czas wyszukiwania wzorca maleje wraz ze wzrostem jego długości, dla pozostałych algorytmów czas ten nieznacznie rośnie.



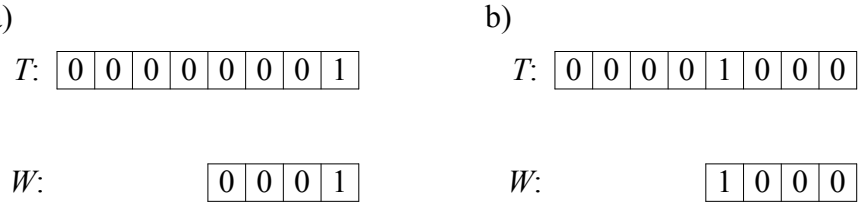
Rys. 12. Wpływ długości wzorca na czas wyszukiwania wzorca w tekście o długości $n = 2 \cdot 10^8$ znaków, wykresy dla alfabetów

Fig. 12. An influence of the length of the pattern on the time of searching the pattern in the text of the length $n = 2 \cdot 10^8$, charts for alphabets

Z porównania wyników przedstawionych na rys. 11 i 12, dla alfabetu Σ_3 o małym rozmiarze wynika zdecydowana przewaga algorytmu BM nad pozostałymi algorytmami, dla którego czas wyszukiwania wzorca o długości $m = 100$ jest ok. 3-krotnie mniejszy od czasu wyszukiwania z użyciem algorytmu NSN i ok. 6-krotnie mniejszy od najwolniejszego algorytmu N. W przypadku pozostałych dwóch alfabetów największe czasy zostały uzyskane dla algorytmów N, KMP, NSN i są one zdecydowanie większe od czasów wyszukiwania z użyciem pozostałych algorytmów.

Dodatkowo zbadano dwa szczególne przypadki dla tekstu i wzorca zawierającego symbole alfabetu Σ_3, gdzie długość wzorca wynosiła $m = 100$ symboli. W pierwszym przypadku tekst i wzorzec zawierają na wszystkich pozycjach symbol ‘0’, a na ostatniej pozycji symbol ‘1’ (rys. 13a). W drugim przypadku symbol ‘1’ występuje we wzorcu na pierwszej pozycji, a na pozostałych pozycjach symbol ‘0’. Tekst natomiast zawiera symbol

‘1’ wyłącznie na pozycji $n - m + 1$ (rys. 13a). W obydwu przypadkach wzorzec występuje na końcu tekstu, tj. z przesunięciem $s = n - m$.



Rys. 13. Przykład szczególnego przypadku wzorca W i tekstu T
Fig. 13. An example of particular the pattern W and the text T

Wyniki przeprowadzonych badań dla obydwu szczególnych przypadków zostały przedstawione w tabeli 1. Pierwszy przypadek, tj. kiedy symbol ‘1’ znajduje się na ostatniej pozycji w tekście i we wzorcu, został oznaczony jako $\Sigma_3(1)$. Drugi przypadek, tj. kiedy symbol ‘1’ we wzorcu znajduje się na pierwszej pozycji, a w tekście na pozycji $n - m + 1$, oznaczono jako $\Sigma_3(2)$. W obydwu przypadkach czas wyszukiwania wzorca rośnie wraz z długością tekstu, co jest spowodowane wzrostem liczby porównań. Dla każdego algorytmu, z wyjątkiem algorytmów Horspoola i Raity, dla przypadku $\Sigma_3(1)$ uzyskano większy czas wyszukiwania wzorca niż w przypadku $\Sigma_3(2)$.

Tabela 1

Zależność czasu wyszukiwania wzorca o długości $m = 100$ od długości tekstu n dla dwóch szczególnych przypadków tekstu i wzorca

Algorytm	Przypadek	Czas wyszukiwania wzorca [ms]				
		$n = 10^7$	$n = 5 \cdot 10^7$	$n = 10^8$	$n = 1{,}5 \cdot 10^8$	$n = 2 \cdot 10^8$
N	$\Sigma_3(1)$	3541	17780	35369	53006	70669
	$\Sigma_3(2)$	108	469	1091	1399	2130
KMP	$\Sigma_3(1)$	212	942	1798	2662	3973
	$\Sigma_3(2)$	126	673	1251	2024	2761
BM	$\Sigma_3(1)$	82	387	794	1169	1541
	$\Sigma_3(2)$	51	246	486	730	979
H	$\Sigma_3(1)$	71	345	686	1042	1378
	$\Sigma_3(2)$	144	702	1395	2095	2805
QS	$\Sigma_3(1)$	1791	8895	17813	26650	35556
	$\Sigma_3(2)$	133	633	1233	1926	2618
S	$\Sigma_3(1)$	1805	8955	17867	26842	35727
	$\Sigma_3(2)$	147	736	1448	2127	2935
R	$\Sigma_3(1)$	94	452	899	1362	1801
	$\Sigma_3(2)$	102	494	980	1460	1953
NSN	$\Sigma_3(1)$	3490	17545	35393	52559	70075
	$\Sigma_3(2)$	1744	8728	17443	26178	34937

W tabeli 2 zostały przedstawione pesymistyczne złożoności czasowe przebadanych algorytmów oraz czasy wyszukiwania wzorców o długości $m = 100$ w tekstach o długości

$n = 2 \cdot 10^8$. W tabeli przedstawiono wyniki dla tekstów i wzorców zawierających symbole alfabetów Σ_1 ,Σ_2, Σ_3 oraz dla dwóch szczególnych przypadków $\Sigma_3(1)$ i $\Sigma_3(2)$ tekstów i wzorca.

Tabela 2

Zestawienie pesymistycznej złożoności czasowej oraz czasów wyszukiwania wzorca o długości $m = 100$ w tekstach o długości $n = 2 \cdot 10^8$ złożonych z symboli różnych alfabetów

Algorytm	Złożoność czasowa	Czas wyszukiwania wzorca [ms]				
		Σ_1	Σ_2	Σ_3	$\Sigma_3(1)$	$\Sigma_3(2)$
N	$O((n-m+1)\,m)$	1632	1918	4097	70669	2130
KMP	$O(n+m)$	1733	2294	2880	3973	2761
BM	$O((n-m+1)\,m+\lvert\Sigma\rvert)$	245	371	738	1541	979
H	$O((n-m+1)\,m+\lvert\Sigma\rvert)$	125	305	2765	1378	2805
QS	$O((n-m+1)\,m+\lvert\Sigma\rvert)$	134	391	2967	35556	2618
S	$O((n-m+1)\,m+\lvert\Sigma\rvert)$	122	341	3116	35727	2935
R	$O((n-m+1)\,m+\lvert\Sigma\rvert)$	110	330	2912	1801	1953
NSN	$O((n-m+1)\,m)$	975	1259	2313	70075	34937

5. Podsumowanie

W niniejszej pracy dokonano analizy porównawczej istniejących algorytmów wyszukiwania wzorca w tekście, przy czym kryterium porównawczym stanowił czas wyszukiwania. W pracy skupiono się wyłącznie na algorytmach, w których zarówno tekst, jak i szukany wzorzec są reprezentowane w postaci ciągów znaków, a samo wyszukiwanie odbywa się przez porównywanie symboli tekstu i wzorca. Badania przeprowadzono z użyciem ośmiu algorytmów: Naive, Knutha–Morrisa–Pratta, Boyera–Moore'a, Horspoola, Quick Search, Smitha, Raity i Not So Naive. Podczas badań skupiono się nad zbadaniem zależności czasu wyszukiwania wzorca od długości tekstu, długości wzorca oraz rozmiaru alfabetu, z którego symboli składa się tekst i wzorzec.

Podczas badań zostały użyte trzy różne alfabety różniące się rozmiarem. Użyto alfabetu Σ_1 zawierającego wszystkich 256 znaków z rozszerzonej tablicy kodu ASCII, alfabetu Σ_2 zawierającego 10 znaków będących cyframi oraz alfabetu Σ_3 zawierającego wyłącznie 2 znaki będące cyframi. Dla każdego z alfabetów użyto pięciu różnych tekstów o długościach równych odpowiednio: 10^7; $5 \cdot 10^7$; 10^8; $1{,}5 \cdot 10^8$ i $2 \cdot 10^8$ znaków, oraz sześciu różnych wzorców o długościach równych odpowiednio: 2, 5, 10, 20, 50 i 100 znaków. Dla każdego badanego algorytmu czas wyszukiwania wzorca maleje wraz ze wzrostem rozmiaru alfabetu. Własność ta wynika z prawdopodobieństwa zgodności porównywanego symbolu tekstu i wzorca, które jest odwrotnie proporcjonalne do rozmiaru alfabetu. Dodatkowo zostały zbadane dwa szczególne przypadki tekstów i wzorca zawierające symbole alfabetu złożonego

z dwóch symboli ‘0’ i ‘1’, w których wzorzec występuje na końcu tekstu. W obydwu przypadkach wzorzec i tekst zawierają wyłącznie jeden symbol ‘1’. W pierwszym przypadku symbol ‘1’ występuje w tekście i we wzorcu na ostatniej pozycji. W drugim przypadku we wzorcu symbol ‘1’ występuje na pozycji pierwszej, a w tekście na pozycji $n - m + 1$.

Dla każdego badanego algorytmu, niezależnie od użytego alfabetu, czas wyszukiwania wzorca rośnie wraz z długością tekstu. W przypadku alfabetów Σ_1 i Σ_2 największe czasy wyszukiwania wzorca zostały uzyskane dla algorytmów Naive, Knutha–Morrisa–Pratta i Not So Naive, natomiast dla pozostałych algorytmów czasy te są mniejsze i niewiele się różnią od siebie. W przypadku alfabetu Σ_3 najmniejszy czas wyszukiwania uzyskano dla algorytmu Boyera–Moore’a, który jest ok. 3-krotnie mniejszy od czasu wyszukiwania z użyciem kolejnego algorytmu, jakim jest Not So Naive. Zatem można wyciągnąć wniosek, iż najlepszym algorytmem dla alfabetów o małym rozmiarze jest algorytm Boyera–Moore’a. Algorytm ten byłby więc najlepszym algorytmem dla wyszukiwania wzorców w tekstach będących ciągami bitów. Algorytm ten, dla każdego z badanych alfabetów, jest także jedynym algorytmem, dla którego czas wyszukiwania maleje wraz z długością wzorca.

W niniejszej pracy dokonano analizy algorytmów, w których wyszukiwanie wzorca w tekście odbywa się przez porównywanie symboli tekstu i wzorca, gdzie tekst i wzorzec są reprezentowane w postaci ciągów znaków. W ramach dalszych prac można dokonać badań algorytmów, w których podczas wyszukiwania wzorca są używane tablice i drzewa sufiksów.

BIBLIOGRAFIA

1. Abu–Alhaj M. M., Halaiyqah M., Abu–Hashem M. A., Hnaif A. A., Abouabdalla O., Manasrah A. M.: An innovative platform to improve the performance of exact string matching algorithms. International Journal of Computer Science and Information Security, Vol. 7, No. 1, 2010, s. 280÷283.
2. Adjeroh D., Bell T., Mukherjee A.: The Burrows–Wheeler Transform: Data Compression,
3. Suffix Arrays, and Pattern Matching. Springer, 2008.
4. Atallah M. J.: Algorithms and Theory of Computation Handbook. CRC-Press, 1st edition, 1998.
5. Boyer R. S., Moore J. S.: A Fast String Searching Algorithm. Communications of the ACM, Vol. 20, No. 10, 1977, s. 762÷772.
6. Charras Ch., Lecroq T.: Handbook of Exact String Matching Algorithms. College Publications, 2004.

7. Cormen T. H., Leiserson Ch. E., Rivest R. L.: Wprowadzenie do algorytmów. WNT, Warszawa 2000.
8. Crochemore M., Rytter W.: Jewels of Stringology. World Scientific Pub Co Inc, 1st edition, 2002.
9. Gusfield D.: Algorithm on strings, trees, and sequences. Cambridge University Press, 1997.
10. Hancart C.: Une analyse en moyenne de l'algorithme de Morris et Pratt et de ses raffinements. Théorie des Automates et Applications, Actes des 2e Journées Franco–Belges, D. Krob ed., Rouen, France, 1992, s. 99÷110.
11. Hancart C.: Analyse exacte et en moyenne d'algorithmes de recherche d'un motif dans un texte, Thése de doctorat de l'Université de Paris, France, 1993.
12. Horspool R. N.: Practical fast searching in strings. Software – Practice and Experience, Vol. 10, No. 6, 1980, s. 501÷506.
13. Karp R. M., Rabin M. O.: Efficient randomized pattern–matching algorithms. Technical Report TR–31–81 , Aiken Computation Laboratory, Harvard University, 1981.
14. Knuth D., Morris Jr. J. H., Pratt V.: Fast pattern matching in strings. SIAM Journal on Computing, Vol. 6, No. 2, 1977, s. 323÷350.
15. Manber U., Myers E.W.: Suffix Arrays: A New Method for On-Line String Searches. SIAM Journal of Computing, Vol. 22, No. 5, 1993, s. 935÷948.
16. Raita T.: Tuning the Boyer–Moore–Horspool string searching algorithm. Software – Practice & Experience, Vol. 22, No. 10, 1992, s. 879÷884.
17. Smith s. D.: Experiments with a very fast substring search algorithm. Software – Practice and Experience, Vol. 21, No. 10, 1991, s. 1065÷1074.
18. Sunday D. M.: A very fast substring search algorithm. Communications of the ACM, Vol. 33, No. 8, s. 132÷142, 1990.

Wpłynęło do Redakcji 26 września 2012 r.

Abstract

In this paper, a comparative analysis of string searching algorithms is presented, and the comparison criterion is the time searching the pattern in the text. The study is focused on algorithms where the text and the pattern are represented by strings, and the searching is performed by comparing symbols of the text and the pattern. The study was carried out with eight algorithms: Naive, Knuth–Morris–Pratt, Boyer–Moore, Horspool, Quick Search, Smith,

Raita and Not So Naive. The dependence of searching time on the length of the text, the length of the pattern and the size of the alphabet were examined.

Three kinds of alphabets were used: the alphabet Σ_1 containing all 256 characters of the extended ASCII table, the alphabet Σ_2 containing 10 characters of digits and the alphabet Σ_3 containing only two characters. For each of the alphabets were used 5 different texts of the lengths n equals 10^7; $5 \cdot 10^7$; 10^8; $1{,}5 \cdot 10^8$ and $2 \cdot 10^8$ characters and 6 different patterns of the lengths m equals 2, 5, 10, 20, 50 and 100 characters. For each of the tested algorithms, the searching time decreases with increasing of the size of the alphabet (Fig. 8–10). It is due to the likelihood of equality compared symbols of the text and the pattern, which is inversely proportional to the size of the alphabet.

For each tested algorithm the searching time increases with the length of the text (Fig. 9). In the test, where texts and patterns containing characters of the alphabets Σ_1 and Σ_2 were used, the greatest searching time has been obtained for Naive, Knuth–Morris–Pratt and Not So Naive algorithms (Fig. 11). For the other algorithms, the searching times are smaller and they differ little from each other. In the test, where the alphabet Σ_3 was used, the smallest searching time has been obtained for Boyer–Moore algorithm. It can be concluded that the best algorithm for the text containing symbols of the small size alphabet is Boyer–Moore algorithm. It was the only one algorithm for which the searching time decreases with the length of the pattern (Fig. 12).

Adres

Jacek WIDUCH: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16,
44–100 Gliwice, Polska, jacek.widuch@polsl.pl