Zaawansowane algorytmy i struktury danych
Algorytm kompresji Lempel-Ziv

Dawid Bitner

9 czerwca 2021

1 Wstęp

Algorytmy z rodziny Lempel-Ziv, głównie takie jak *LZ77* czy *LZ78* są algorytmami kompresji bezstratnej. Kompresję bezstratną możemy zdefiniować jako kodowanie bezstratne w taki sposób, aby minimalizować długość komunikatu po zakodowaniu. Kodem jest zbiór słów kodowych przyporządkowanych ciągom symboli alfabetu źródła bądź też poszczególnym symbolom tego alfabetu. Natomiast słowa kodowe to ciągi symboli pewnego alfabetu, który będzie określany mianem alfabetu kodu. Algorytmy takie nie nadają się jednak do kompresji wszystkich typów informacji. Nie znajdą zastosowania m.in.: w kompresowaniu strumienia liczb losowych i pseudolosowych.

2 Opis historyczny (geneza)

W roku 1977 dwaj izarealscy naukowcy - informatycy: Abraham Lempel i Jacob Ziv opublikowali przełomowy algorytm kompresji bezstratnej, który swoją nazwę zawdzięcza pierwszym literom nazwiska twórców i dacie jego opracowania - czyli *LZ77*. Był to algorytm przełomowy, ponieważ jako pierwszy algorytm wykorzystywał słownik do kompresji danych. *LZ77* używa słownika dynamicznego - nazywanego również często oknem przesuwnym. Rok później, w 1978, ta sama para naukowców opublikowała kolejny algorytm - *LZ78*. Algorytm ten również używa słownika, jednak w przeciwieństwie do *LZ77*, algorytm ten analizuje dane wejsciowe i generuje statyczny słownik zamiast generować go dynamicznie. Zarówno algorytmy *LZ77*, jak i *LZ78* szybko zyskały na popularności - powstały ich modyfikacje jak np.: *LZW*.
Ciekawostką jest, że większość powszechnie stosowanych algorytmów wywodzi się z algorytmu *LZ77*. Nie wynika to z przewagi technicznej tego algorytmu, ale z tego że algorytmy *LZ78* zostały obciążone patentami po tym, jak pochodny algorytm *LZW* został opatentowany w 1984 roku i autor patentu zaczął pozywać dostawców oprogramowania, administratorów serwerów, a nawet użytkowników końcowych za używanie formatu GIF bez licencji.

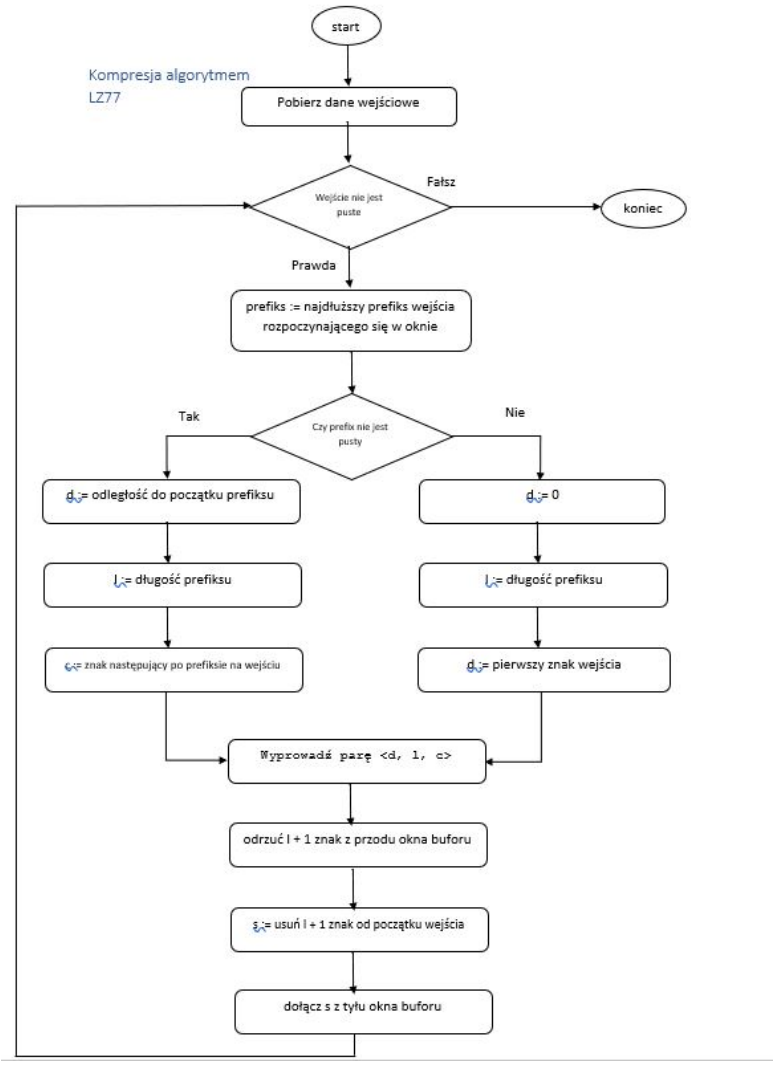

Rysunek 1: Twórcy algorytmu: Abraham Lempel oraz Jacob Ziv.

3 Algorytmy

3.1 LZ77

W kodowaniu *LZ77* przechowuje się bufor (tzw. okno) będące połączeniem bufora słownikowego który zawiera określoną liczbę ostatnio kodowanych symboli i bufora kodowania z pewną liczbą symboli, które dopiero mają zostać w kolejnym kroku poddane kodowaniu. Rozpoczynając proces kodowania wypełniamy bufor słownikowy powtórzeniami pierwszego symbolu zawartości ciągu kodowanego, a bufor kodowania przedrostkiem kodowanego komunikatu, o długości równej długości bufora kodowania. Następnie cyklicznie wyszukujemy najdłuższy przedrostek f bufora kodowania w oknie. Poszukiwania rozpoczynamy od końca bufora słownikowego po lewej stronie i przesuwamy się w prawo. Znalezione w ten sposób dopasowanie f może mieć długość większą od długości bufora słownikowego, nie większą jednak od długości okna. Po znalezieniu f wyprowadzamy trójkę $< d, l, c >$, gdzie d oznacza pozycję f w buforze słownikowym, l to długość f, a c to symbol, dla którego f jest przedrostkiem w oknie, czyli pierwszy symbol znajdujący się w buforze kodowania za f. Po wyprowadzeniu trójki $< d, l, c >$ zawartość okna przesuwamy o l pozycji w lewo, wprowadzając jednocześnie do bufora kodowania kolejne, nie pobrane jeszcze symbole komunikatu.

3.1.1 Schemat blokowy



3.1.2 Pseudokod

```
WHILE wejście nie jest puste DO
    prefiks := najdłuższy prefiks wejścia rozpoczynającego się w oknie

    IF prefiks nie jest pusty
        d := odległość do początku prefiksu
        l := długość prefiksu
        c := znak następujący po prefiksie na wejściu
    ELSE
        d := 0
        l := 0
        c := pierwszy znak wejścia
    END IF

    wyprowadź na wyjście <d, l, c>

    odrzuć l + 1 znak z przodu okna buforu
    s := usuń l + 1 znak od początku wejścia
    dołącz s z tyłu okna buforu
REPEAT
```

3.1.3 Schemat poglądowy działania na przykładzie - Kodowanie

Poniżej zaprezentowano sposoby kompresji i dekompresji za pomocą algorytmów Lempel-Ziv (LZ77, *LZ78*), oraz dodatkowo sposób kompresji za pomocą algorytmu Lempel-Ziv-Welch (LZW). W każdym z przykładów posłużyłem się ciągiem znaków: *aaabbbbbbaaaabbbb*.
Dla przykładu ustalamy słownik (lewa strona) i bufor (prawa strona) na 8.

```
01234567 | 01234567
aaaaaaaa | aaabbbbb -> a                  (A)
aaaaaaaa | aaabbbbb -> <0, 3, b>   <- 4  (B)
aaaaaaab | bbbbbaaa -> <7, 5, a>   <- 6  (C)
abbbbbba | aaabbbb  -> <7, 3, b>   <- 4  (D)
bbbaaaab | bbb      -> <0, 2, b>   <- 3  (E)
```

Wynik działania: a <0, 3, b> <7, 5, a> <7, 3, b> <0, 2, b>

(A) W pierwszej kolejności wypełniamy cały słownik pierwszym znakiem z ciągu do zaszyfrowania. W typ przypadku jest to litera „a".

(B) W buforze odnajdujemy pierwszy ciąg stałych znaków. Jest to „aaa". Odpowiada on ciągowi w słowniku, który zaczyna się od pozycji nr 0, którą wstawiamy na pierwszym miejscu w nawiasie, następnie sprawdzamy w którym miejscu w buforze występuje najbliższy znak różniący się od „a" – jest to „b" na pozycji nr 3. Liczbę 3 wpisujemy na drugim miejscu do nawiasu. Na ostatnim miejscu w nawiasie umieszczamy znak „b". Przesuwamy w lewo o 4 (3 + 1) pozycje zestaw naszych znaków, dopisujemy do buforu 4 kolejne znaki z ciągu, który ma zostać poddany kompresji.

(C) Znak „b" występuje na pozycji 7 w słowniku. Ciąg znaków w buforze zawiera się w pozycjach od 0 do 4. Kolejny różny znak występuje na pozycji 5. Zapisujemy dane do nawiasów. Ciąg przesuwamy jak w powyższym punkcie o 5 (4 + 1) znaków.

(D) Znak „a" występuje na pozycji 7 w słowniku – z przeskokiem do pozycji 0. Ciąg znaków w buforze zawiera się w pozycjach od 0 do 2. Kolejny różny znak występuje na pozycji 3. Zapisujemy dane do nawiasów. Ciąg przesuwamy jak w powyższym punkcie o 4 (3 + 1) znaków. Bufor nie zostaje w pełni wypełniony, ponieważ w startowym ciągu znaków nie pozostała wystarczająca liczba znaków.

(E) Znak „b" występuje na pozycjach od 0 do 2 w słowniku. Ciąg znaków w buforze zawiera się w pozycjach od 0 do 2. Kolejny różny znak występuje na pozycji 3. Zapisujemy dane do nawiasów. Ciąg przesuwamy jak w powyższym punkcie o 4 (3 + 1) znaków. Bufor nie zostaje w pełni wypełniony, ponieważ w startowym ciągu znaków nie pozostała wystarczająca liczba znaków.

3.1.4 Schemat poglądowy działania na przykładzie - deKodowanie

DeKodowanie algorytmów *LZ77* i *LZ78* została umieszczona w sprawozdaniu jako dodatek. Dekompresujemy ten sam przykład:

a <0, 3, b> <7, 5, a> <7, 3, b> <0, 2, b>

```
01234567 | 01234567
aaaaaaaa |              a                  (A)
aaaaaaaa | aaab         <0, 3, b>   <- 4   (B)
aaaaaaab | bbbbba       <7, 5, a>   <- 6   (C)
abbbbbba | aaab         <7, 3, b>   <- 4   (D)
bbbaaaab | bbb          <0, 2, b>   <- 3   (E)
```

Wynik odczytujemy od góry: aaab bbbbba aaab bbb

(A) W pierwszej kolejności wypełniamy cały słownik pierwszym znakiem z ciągu do zaszyfrowania. W typ przypadku jest to litera „a”. Odczytując „a”.

(B) <0, 3, b> wypełniamy bufor w następujący sposób: od pozycji 0 do pozycji 2 (3 – 1) wypełniamy poprzednim znakiem: „a”. Na pozycji 3 wpisujemy znak b.

(C) Wracając do poprzedniego kroku. Przesuwamy się, podobnie jak przy kompresji o 4 (3 + 1) znaków. <7, 5, a>. W słowniku na pozycji 7 występuje „b”. W buforze wypisujemy „b” do pozycji 4. Na pozycji 5 wpisujemy „a”.

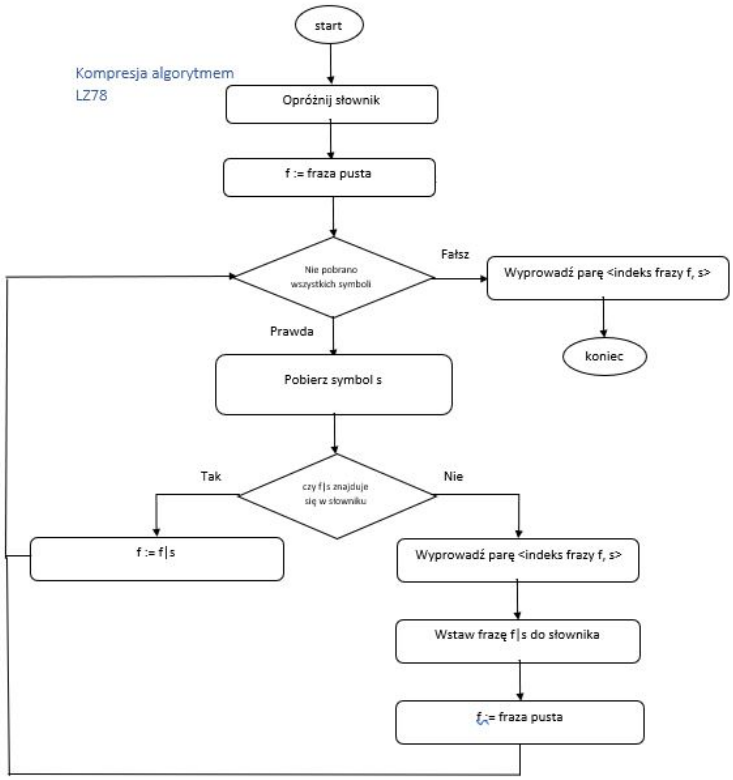
(D) Wracając do poprzedniego kroku. Przesuwamy się, podobnie jak przy kompresji o 6 (5 + 1) znaków. <7, 3, b>. W słowniku na pozycji 7 występuje „a”. W buforze wypisujemy „a” do pozycji 2. Na pozycji 3 wpisujemy „b”.

(E) Wracając do poprzedniego kroku. Przesuwamy się, podobnie jak przy kompresji o 4 (3 + 1) znaków. <0, 2, b>. W słowniku na pozycji 0 występuje „b”. W buforze wypisujemy „b” do pozycji 1. Na pozycji 2 wpisujemy „b”. Jako że wpisujemy ten sam znak to następuje zakończenie algorytmu.

3.2 LZ78

W algorytmie *LZ78* pobiera się następne symbole kodowanego ciągu budując z nich pewną frazę. Po pobraniu kolejnego symbolu i dodaniu go na końcu już zbudowane frazy sprawdza się, czy słownik tę frazę zawiera. Jeżeli frazy nie ma w słowniku, to wyprowadzany jest indeks najdłuższej dopasowanej frazy i ostatnio pobrany symbol. Po zakodowaniu frazy i symbolu wstawia się do słownika frazę, dla której operacja szukania zakończyła się niepowodzeniem i rozpoczyna się budowanie nowej frazy od frazy pustej.

3.2.1 Schemat blokowy



3.2.2 Pseudokod

```
opróżnij słownik
f := fraza pusta
WHILE nie pobrano wszystkich symboli komunikatu DO
    pobierz symbol s
    IF (f|s znajduje się w słowniku)
        f := f|s
    ELSE
        wyprowadź na wyjście <indeks frazy f, s>
        wstaw frazę f|s do słownika
        f := fraza pusta
    END IF
END WHILE
wyprowadź na wyjście <indeks frazy f, s>
```

3.2.3 Schemat poglądowy działania na przykładzie - Kodowanie

Dla przypomnienia, posługujemy się następującym ciągiem. *aaabbbbbbaaaabbbb*.

Index	Kodowany ciąg	Wynik
1	a	(0, a)
2	aa	(1, a)
3	b	(0, b)
4	bb	(3, b)
5	bbb	(4, b)
6	aaa	(2, a)
7	bbbb	(5, b)

Wynik: (0, a) (1, a) (0, b) (3, b) (4, b) (2, a) (5, b).

1. Wybieramy pierwszy znak ciągu – „a”. W wyniku wpisujemy (0, a) – ponieważ słowo nie było wcześniej indeksowane – dlatego 0, drugą wartością wpisaną jest „a” – dodany znak do słowa – w tym przypadku słowo bazowe było puste („”).
   ~~a~~aabbbbbbbaaaabbbb – skreślamy dodany znak do słownika.
2. Kolejnym słowem z nieskreślonego ciągu, który nie występuje w słowniku jest „aa”. Aby osiągnąć takie słowo musimy pobrać zawartość słowa o indeksie 1 i rozszerzyć je o „a”. Stąd zapis (1, a).
   ~~aaa~~bbbbbbbaaaabbbb - wykreślamy ciąg.
3. Kolejnym znakiem jest „b” który w tworzonym słowniku nie występuje. W wyniku wpisujemy (0, b) – ponieważ słowo nie było wcześniej indeksowane – dlatego 0, drugą wartością wpisaną jest „b” – dodany znak do słowa – w tym przypadku słowo bazowe było puste („”).
   ~~aaab~~bbbbbbaaaabbbb – skreślamy dodany znak do słownika.

4. Kolejnym słowem z nieskreślonego ciągu, który nie występuje w słowniku jest „bb”. Aby osiągnąć takie słowo musimy pobrać zawartość słowa o indeksie 3 i rozszerzyć je o „b”. Stąd zapis (3, b).
   ~~aaabbb~~bbbaaaabbbb - wykreślamy ciąg.
5. Kolejnym słowem z nieskreślonego ciągu, który nie występuje w słowniku jest „bbb”. Aby osiągnąć takie słowo musimy pobrać zawartość słowa o indeksie 4 i rozszerzyć je o „b”. Stąd zapis (4, b).
   ~~aaabbbbbb~~aaaabbbb - wykreślamy ciąg.
6. Kolejnym słowem z nieskreślonego ciągu, który nie występuje w słowniku jest „aaa”. Aby osiągnąć takie słowo musimy pobrać zawartość słowa o indeksie 2 i rozszerzyć je o „a”. Stąd zapis (2, a).
   ~~aaabbbbbbaaaa~~bbbb - wykreślamy ciąg.
7. Kolejnym słowem z nieskreślonego ciągu, który nie występuje w słowniku jest „bbbb”. Aby osiągnąć takie słowo musimy pobrać zawartość słowa o indeksie 5 i rozszerzyć je o „b”. Stąd zapis (5, b).
   ~~aaabbbbbbaaaabbbb~~ - wykreślamy ciąg.

3.2.4 Schemat poglądowy działania na przykładzie - deKodowanie

Przykład: (0, a) (1, a) (0, b) (3, b) (4, b) (2, a) (5, b).

Zakodowany ciąg	Odkodowany ciąg	Index
(0, a)	a	1
(1, a)	aa	2
(0, b)	b	3
(3, b)	bb	4
(4, b)	bbb	5
(2, a)	aaa	6
(5, b)	bbbb	7

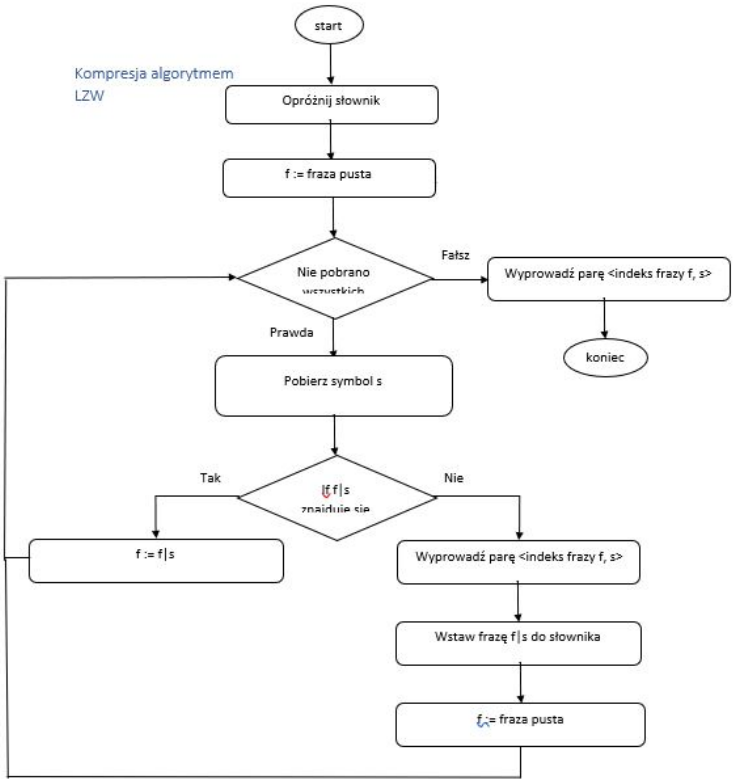
Wynik: a aa b bb bbb aaa bbbb

1. Odczytujemy pierwszy ciąg: (0, a) – indeks zerowy oznacza, że poprzedni ciąg był pusty, rozszerzamy go o „a". Słowo otrzymuje indeks 1. W wyniku otrzymujemy „a".
2. (1, a) – rozszerzamy słowo o indeksie 1 o znak „a". Słowo otrzymuje indeks 2. W wyniku otrzymujemy „aa".
3. (0, b) – oznacza pierwsze wystąpienie ciągu „b". indeks zerowy oznacza, że poprzedni ciąg był pusty, rozszerzamy go o „b". Słowo otrzymuje indeks 3. W wyniku otrzymujemy „b".
4. (3, b) – rozszerzamy słowo o indeksie 3 o znak „b". Słowo otrzymuje indeks 4. W wyniku otrzymujemy „bb".
5. (4, b) – rozszerzamy słowo o indeksie 4 o znak „b". Słowo otrzymuje indeks 5. W wyniku otrzymujemy „bbb".
6. (2, a) – rozszerzamy słowo o indeksie 2 o znak „a". Słowo otrzymuje indeks 6. W wyniku otrzymujemy „aaa".
7. (5, b) – rozszerzamy słowo o indeksie 5 o znak „b". Słowo otrzymuje indeks 7. W wyniku otrzymujemy „bbbb".

3.3 LZW

Algorytm *LZW* jest udoskonaleniem algorytmu *LZ78* polegającym na wstępnym wypełnieniu słownika jednoznakowymi frazami ze wszystkimi symbolami występującymi w alfabecie. Wyprowadzane są tylko indeksy fraz. Współczynniki kompresji *LZW* są lepsze niż w przypadku używania algorytmu *LZ78*.

3.3.1 Schemat blokowy



3.3.2 Pseudokod

```
zainicjalizuj słownik
f := fraza pusta
WHILE nie pobrano wszystkich symboli komunikatu DO
    pobierz symbol s
    IF (f|s znajduje się w słowniku)
        f := f|s
    ELSE
        wyprowadź indeks frazy f
        wstaw frazę f|s do słownika
        f := s
    END IF
END WHILE
wyprowadź indeksy frazy f
```

3.3.3 Schemat poglądowy działania na przykładzie - Kodowanie

Dla przypomnienia, posługujemy się następującym ciągiem. *aaabbbbbbaaaabbbb*.
Algorytm jest bardzo podobny do algorytmu *LZ78* – z tym, że ograniczamy się do indeksów.

encode	decode	
	index	entry
-	1	a
-	2	b
1	3	aa
3	4	aab
2	5	bb
5	6	bbb
6	7	bbba
3	8	aaa
4	9	aabb
6	10	-

Wynik: 1 3 2 5 6 3 4 6

1. Alfabetycznie – indeksujemy pierwszy znak, czyli „a” – pod 1.
2. Alfabetycznie – indeksujemy drugi znak, czyli „b” – pod 2.
3. „a” w słowniku występuje, więc rozszerzamy o drugi znak: czyli „aa”. aaabbbbbbaaaabbbb.
4. Przesuwamy zaznaczenie w prawo i rozszerzamy o jeden znak: aaabbbbbbaaaabbbb.
5. Pozostawiamy ostatni znak i rozszerzamy zaznaczenie o jeden. Otrzymujemy: aaabbbbbbaaaabbbb.
6. Pozostawiamy ostatni znak i rozszerzamy zaznaczenie o jeden. Otrzymujemy: aaabbbbbbaaaabbbb.

7. Pozostawiamy ostatni znak i rozszerzamy zaznaczenie o jeden i dodatkowo rozszerzamy jeszcze raz o jeden, ponieważ bez rozszerzenia otrzymalibyśmy słowo „bbb" – które już istnieje, więc: aaabbbbbbbaaabbbb.
8. Pozostawiamy ostatni znak i rozszerzamy zaznaczenie o dwa, ponieważ rozszerzając o jeden otrzymamy „aa" – a takie słowo już występuje w naszym słowniku. Otrzymujemy: aaabbbbbbbaaaabbbb.
9. Pozostawiamy ostatni znak i rozszerzamy zaznaczenie o dwa, ponieważ rozszerzając o jeden otrzymamy „aab" – a takie słowo już występuje w naszym słowniku. Otrzymujemy: aaabbbbbbbaaaabbbb.
10. Pozostaje nam ciąg: aaabbbbbbbaaaabbbb – który występuje już pod indeksem 6 – opisujemy do wyniku.

4 Czas wykonywania algorytmów

Porównane zostały czasy wykonywania kompresji i dekompresji powyższych trzech algorytmów tj.: *LZ77*, *LZ78* i *LZW*. W tym celu wykorzystane zostało narzędzie timera z biblioteki *timeit*. Pierwszy test został przeprowadzony dla ciągu znaków wykorzystywanego w powyższych przykładach, czyli: *aaabbbbbbaaaabbbb*.
Drugi test został przeprowadzony dla ciągu:
wszczebrzeszyniechrzaszczbrzmiwtrzcinie.
Wyniki zostały przedstawione poniżej:

```
Zakodowany ciąg w LZ77: <0, 0, a> <0, 0, a> <0, 0, a> <0, 0, b> <0, 0, b> <0, 0, b> <3, 3, a> <0, 0, a> <0, 0, a> <3,
1, b> <0, 0, b> <0, 0, b> <3, 1, >
Czas kodowania LZ77:  0.0793999997767969
Odkodowany ciąg w LZ77:  ['a', 'a', 'a', 'b', 'b', 'b', 'b', 'b', 'b', 'a', 'a', 'a', 'a', 'b', 'b', 'b', 'b', '']
Czas dekodowania LZ77:  0.015600000097037992
- - - - - - - - - - - - - - - - - - -
Zakodowany ciąg w LZ78: <0, a> <1, a> <0, b> <3, b> <4, b> <2, a> <1, b> <5,  >
Czas kodowania LZ78:  0.025099999675148865
Odkodowany ciąg w LZ78: aaabbbbbbaaaabbbb
Czas dekodowania LZ78:  1.0990000000674627
- - - - - - - - - - - - - - - - - - -
Zakodowany ciąg w LZW:  [1, 3, 2, 5, 6, 3, 4, 6]
Czas kodowania LZW:  0.07049999976516119
Odkodowany ciąg w LZW: aaabbbbbbaaaabbbb
Czas dekodowania LZW:  0.33400000029359944
```

Rysunek 1: Twórcy algorytmu: Abraham Lempel oraz Jacob Ziv.

```
Zakodowany ciąg w LZ77: <0, 0, w> <0, 0, s> <0, 0, z> <0, 0, c> <0, 0, z> <0, 0, e> <0, 0, b> <0, 0, r> <4, 2, s> <3,
1, y> <0, 0, n> <0, 0, i> <0, 0, e> <0, 0, c> <0, 0, h> <0, 0, r> <0, 0, z> <0, 0, a> <0, 0, s> <3, 1, c> <0, 0, z>
<0, 0, b> <0, 0, r> <3, 1, m> <0, 0, i> <0, 0, w> <0, 0, t> <0, 0, r> <0, 0, z> <0, 0, c> <0, 0, i> <0, 0, n> <0, 0,
i> <0, 0, e>
Czas kodowania LZ77:  0.2681000000848144
Odkodowany ciąg w LZ77:  ['w', 's', 'z', 'c', 'z', 'e', 'b', 'r', 'z', 'e', 's', 'z', 'y', 'n', 'i', 'e', 'c', 'h', '
r', 'z', 'a', 's', 'z', 'c', 'z', 'b', 'r', 'z', 'm', 'i', 'w', 't', 'r', 'z', 'c', 'i', 'n', 'i', 'e']
Czas dekodowania LZ77:  0.05375999993238016
- - - - - - - - - - - - - - - - - - -
Zakodowany ciąg w LZ78: <0, w> <0, s> <0, z> <0, c> <3, e> <0, b> <0, r> <5, s> <3, y> <0, n> <0, i> <0, e> <4, h>
<7, z> <0, a> <2, z> <4, z> <6, r> <3, m> <11, w> <0, t> <14, c> <11, n> <11, e>
Czas kodowania LZ78:  0.08909999996831175
Odkodowany ciąg w LZ78: wszczebrzeszyniechrzaszczbrzmiwtrzcinie
Czas dekodowania LZ78:  1.7466000003878435
- - - - - - - - - - - - - - - - - - -
Zakodowany ciąg w LZW:  [1, 2, 3, 4, 3, 5, 6, 7, 19, 16, 8, 9, 10, 5, 4, 11, 22, 12, 16, 18, 21, 3, 13, 10, 1, 14, 2
2, 4, 10, 26, 5]
Czas kodowania LZW:  0.19659999998111743
Odkodowany ciąg w LZW: wszczebrzeszyniechrzaszczbrzmiwtrzcinie
Czas dekodowania LZW:  1.8853999999919324
```

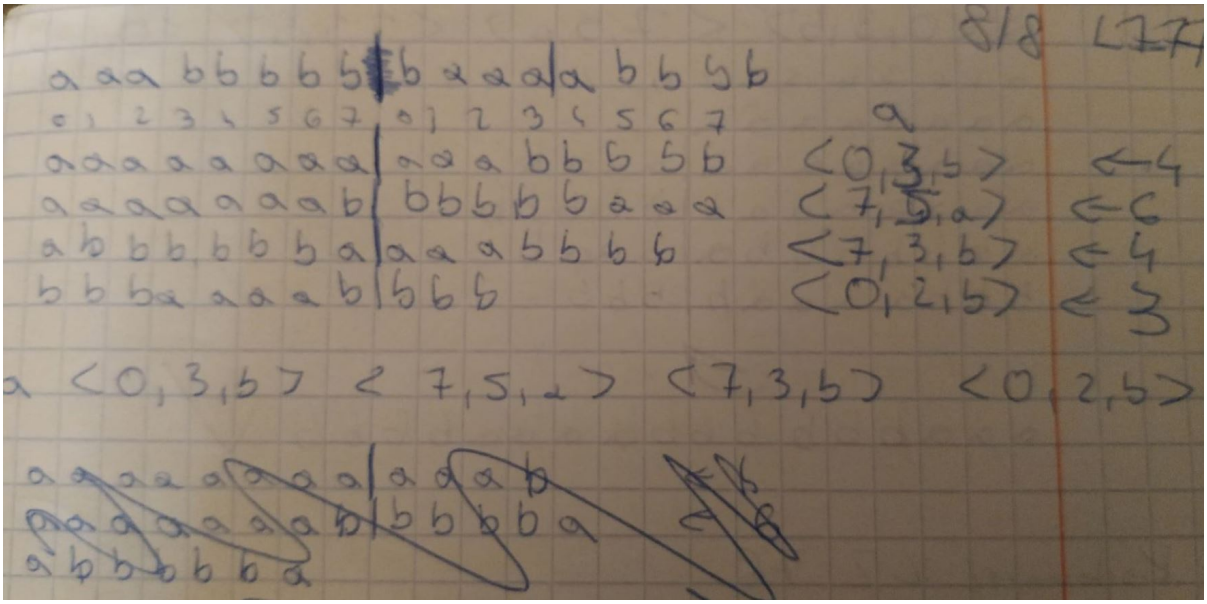
Rysunek 2: Twórcy algorytmu: Abraham Lempel oraz Jacob Ziv.

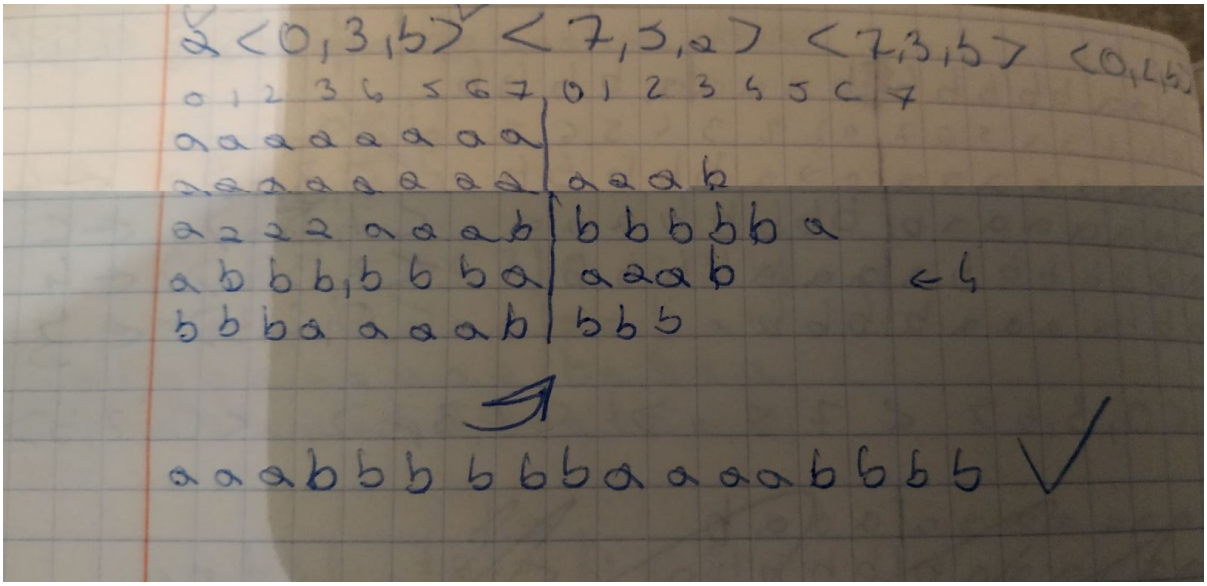
Algorytmy kompresji w kodowaniu *LZ77* jest dużo bardziej złożony niż algorytm dekompresji. W kodowaniu *LZ77* można wpływać bezpośrednio na prędkość kompresji, regulując parametry kodera (rozmiar słownika i bufora kodowania - w teście odpowiednio ustawione na 8 i 6). Stąd możliwe, że najlepsze wyniki w powyższym teście osiągnął właśnie algorytm *LZ77*. Wyniki czasowe działania algorytmów *LZ78* i *LZW* (który tak naprawdę jest wariantem *LZ78*) są bardzo zbliżone do siebie.

5 Dodatek: ręczne obliczenia

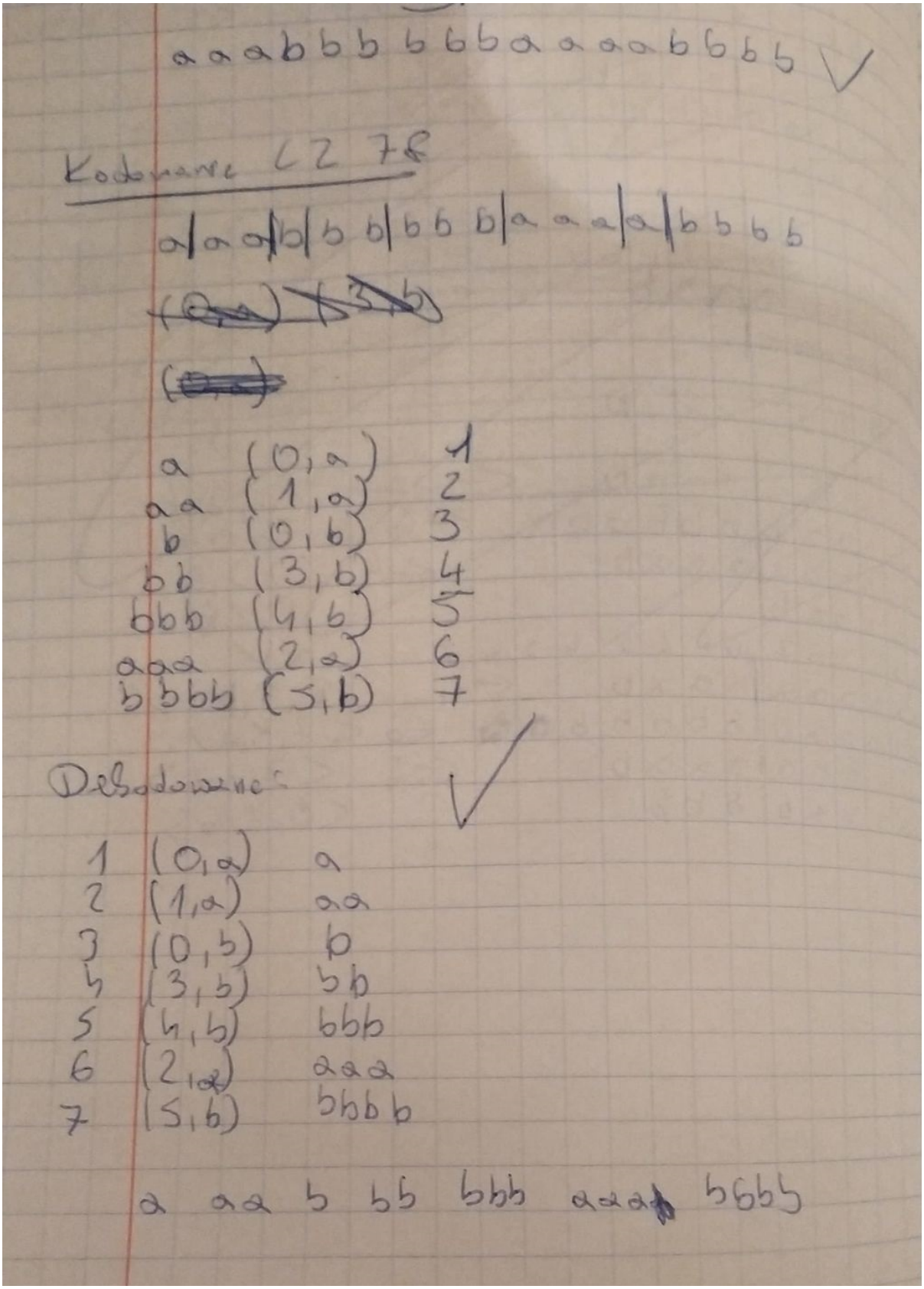
Zaprezentowane zostały tutaj ręczne obliczenia, które posłużyły do przedstawienia przykładów we wcześniejszych podpunktach.

5.1 Kodowanie i dekodowanie w *LZ77*

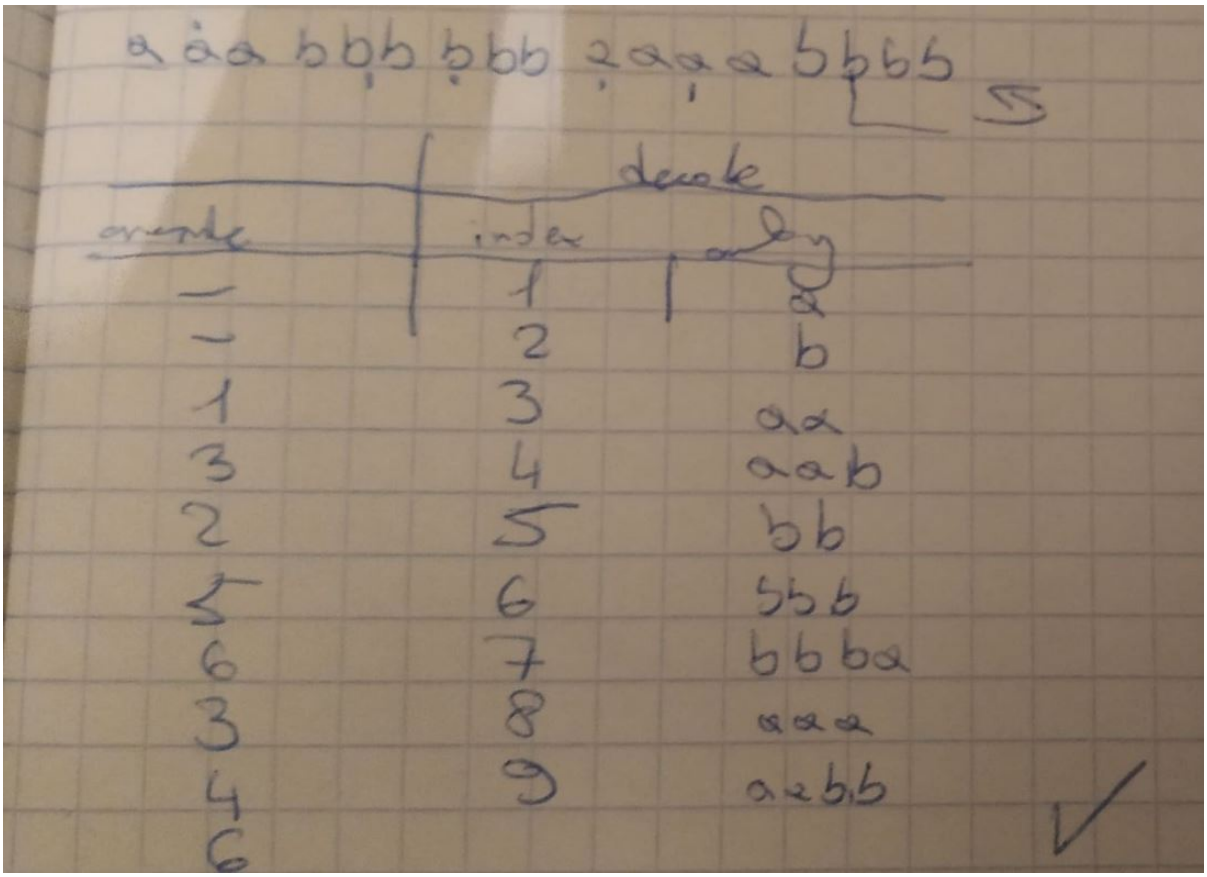




5.2 Kodowanie i dekodowanie w *LZ78*



5.3 Kodowanie w *LZW*



SCHEMATY BLOKOWE I POGLĄDOWE - WRAZ Z OPISEM SĄ DOSTĘPNE W WERSJI DO EDYCJI W ZAŁĄCZONYM DO SPRAWOZDANIA PLIKU .DOCX.

Literatura

[1] Drozdek A.: Wprowadzenie do kompresji danych. WNT, Warszawa 1999.

[2] Starosolski R.: Algorytmy bezstratnej kompresji danych. Studia Informatica, Vol. 24 Number 1(52), Politechnika Śląska 2003.

[3] Ziv J., Lempel A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, Vol. 32(3), May 1977, pp. 337-43.

[4] Ziv J., Lempel A.: Compression of individual sequences via variable rate coding. IEEE Transactions on Information Theory, Vol. 24(5), Sept. 1978, pp. 530-6.

[5] Wolfram S. A New Kind of Science. Champaign, IL: Wolfram Media, 2002. 1069.

[6] USPTO Patent no. 4814746.
http://www.theregister.co.uk/1999/09/01/unisys_demands_5k_licence_fee
(dostęp 09.06.2021)