

BIBLIOTEKA GŁÓWNA
I OSRODEK INFORMACJI NAUKOWO-TECHNICZNEJ
POLITECHNIKI WROCŁAWSKIEJ

Raport ser. PRE nr 55

ALGORYTM KOMPRESJI
WEKTOROW BINARNYCH
/rozprawa doktorska/

Andrzej Siemiński

Słowa kluczowe: kompresja danych, kodowanie,
wektory binarne, wyszukiwanie
informacji, inwersyjna orga-
nizacja zbioru dokumentów

Promotor:

Doc. dr Czesław Daniłowicz

Wrocław 1984

Mgr inż. Andrzej Siemiński

Biblioteka Główna i Ośrodek
Informacji Naukowo-Technicznej
Politechniki Wrocławskiej

Zakład Systemów Informacyjnych

Wybrzeże Wyspiańskiego 27, 50-370 Wrocław
Bud. A-1, pok. 203 c, tel. 20-37-99

Raport wpłynął do Redakcji
Biblioteki Gł.OINT PWr
4.10.1983

Spis treści

WSTĘP	5
1. PRZEGLĄD METOD KOMPRESJI STOSOWANYCH W INFORMATYCE	11
1.1. Metody semantyczne	12
1.2. Metody statystyczne	17
2. KOMPRESJA STRUKTUR BINARNYCH	32
2.1. Struktury binarne	32
2.2. Przegląd algorytmów kompresji struktur binarnych	37
2.3. Kodowanie predykcyjne	52
3. TEORETYCZNE PODSTAWY KOMPRESJI	56
3.1. Ilość informacji jako funkcja prawdo- podobieństwa wystąpienia zdarzenia	57
3.2. Bezpamięciowe źródło informacji	58
3.3. Ergodyczne źródło Markowa	59
3.4. Źródła złożone	61
3.5. Entropia źródła a kodowanie jego wiadomości	71
4. OPIS PROPONOWANEGO ALGORYTMU KOMPRESJI WEKTORÓW BINARNYCH	74
4.1. Opis działania algorytmu A	75
4.2. Analiza działania algorytmu A	82
4.3. Wykorzystanie algorytmu A do kompresji wektorów binarnych	92

5. OPIS EKSPERYMENTU	103
5.1. Generowanie i testowanie wektorów binarnych	103
5.2. Porównanie sprawności kodowania algorytmów kompresji wektorów binarnych	113
PODSUMOWANIE	122
LITERATURA	125
ZAŁĄCZNIKI	130

WSTĘP

Maszyny cyfrowe wykorzystywane są coraz częściej do wykonywania zadań polegających na stosunkowo prostym przetwarzaniu dużych zbiorów danych. Wymagane do realizacji takich zadań liczne transmisje między pamięcią operacyjną a pomocniczą w istotny sposób wpływają na globalny czas przetwarzania. Coraz większego znaczenia nabierają koszty przechowywania danych zwłaszcza, jeżeli zapamiętane są one na nośnikach o bezpośrednim dostępie. Obserwowany obecnie stały rozwój sieci komputerowych powoduje zwiększanie ilości danych transmitowanych łąkami telekomunikacyjnymi.

Wymienione wyżej przyczyny skłaniają do poszukiwania metod skracania długości zapisu danych, tj. ich kompresji. Dzięki kompresji możliwe jest bowiem zmniejszenie kosztów przetwarzania poprzez skrócenie czasu transmisji danych, zredukowanie objętości przechowywanych zbiorów oraz skrócenie czasu wykonywania programu przez ograniczenie liczby operacji wejścia-wyjścia. Stosowanie kompresji nie jest pozbawione wad. Powoduje ona zwiększenie stopnia skomplikowania oprogramowania, utrudnia kontrolę poprawności danych, zaś brak ogólnie przyjętych standardów [1] poważnie ogranicza wymianę danych w postaci skompresowanej.

Kompresja stosowana jest przy przetwarzaniu struktur binarnych. Strukturami binarnymi nazywane będą w pracy dane utworzone z elementów przyjmujących wartość 1 lub 0. Kompresja takich danych jest szczególnie pożądana, bowiem cechują się one dużą długością zapisu przy jednoczesnej znacznej jego redundancji. Struktury te są wykorzystywane powszechnie do reprezentowania cyfrowych zapisów obrazów i list adresów w systemach wyszukiwa-

nia informacji.

Dotychczas opracowano kilka algorytmów kompresji wektorów binarnych. Algorytmy te osiągają właściwe sobie maksymalne wartości współczynnika kompresji jedynie wówczas, gdy kodowanie jest poprzedzone ustaleniem statystycznych własności wektora. Współczynnik kompresji, K_m , jest podstawowym miernikiem jakości pracy algorytmu i w przypadku kompresji struktur binarnych wartość jego jest wyznaczana na podstawie równania:

$$K_m = \frac{D_{we}}{D_{wy}}$$

gdzie:

- D_{we} - wejściowa długość zapisu danych
- D_{wy} - długość zapisu danych po kompresji.

Ustalenie własności danych pozwala na określenie parametrów działania algorytmu, które muszą być zapamiętane łącznie ze skompresowaną postacią danych, zwiększając, w niektórych przypadkach dość istotnie, jej długość. Opracowane proste algorytmy kompresji (np. Bradleya, Golomba) wykorzystują niewielką liczbę parametrów oraz osiągają współczynniki kompresji o wartościach niewiele różniących się od maksymalnych możliwych do uzyskania, lecz ma to miejsce jedynie wtedy, gdy przetwarzane dane mają własności ciągów pochodzących z bezpamięciowego, binarnego źródła informacji. Dane spełniające dobrze te warunki, występują dość rzadko w praktyce. Algorytmy przeznaczone do kompresji wszystkich rodzajów wektorów binarnych spotykanych w wyżej wymienionych zastosowaniach są bardziej skomplikowane na ponadto wymagają ustalenia, a następnie przechowania znacznej liczby parametrów określających sposób ich działania (np. algorytm Huffmana). Wymienione powyżej cechy opracowanych dotychczas algorytmów kompresji utrudniają znacznie wykorzystanie tych

algorytmów w praktyce.

Wektory binarne stosowane są w systemach wyszukiwania informacji powstałych na Politechnice Wrocławskiej. Doświadczenia zebrane przy opracowaniu tych systemów pozwoliły na ustalenie pożądanych cech algorytmu kompresji wektorów binarnych:

- osiągnięcie skrócenia długości zapisu wektorów binarnych nie odbiegającego od wyników uzyskiwanych przez najlepsze z opublikowanych dotychczas algorytmów,
- wyeliminowanie (lub znaczne ograniczenie zakresu) ustalania własności statystycznych przetwarzanych wektorów,
- prostota kodowania i dekodowania,
- stosowanie niewielkiej liczby parametrów określających sposób działania algorytmu.

Celem przeprowadzonych badań było znalezienie algorytmu spełniającego wszystkie wymienione wyżej wymagania. Zakres przeprowadzonych prac obejmował:

- opracowanie algorytmu kompresji wektorów binarnych,
- teoretyczne oszacowanie sprawności działania zaproponowanego algorytmu,
- eksperymentalne zweryfikowanie sprawności algorytmu.

Analiza literatury dotyczącej kompresji danych wykazała, że w literaturze polskiej brak jest prac wnoszących istotny wkład do rozwiązywania problemów związanych z kompresją, zaś publikowane w literaturze zagranicznej zarówno prace przeglądowe, jak i opisy poszczególnych algorytmów dotyczą z zasady metod kompresji danych o określonych właściwościach. Z tego powodu autor niniejszej pracy uznał za celowe zamieszczenie w rozdziale 1 przeglądu metod kompresji najczęściej stosowanych w informatyce. Prezentowane metody zostały podzielone na semantyczne i statystyczne. W metodach semantycznych projektant na podstawie

swej wiedzy o sposobie wykorzystania danych, ich treści, strukturze oraz wzajemnych powiązaniach między danymi określa sposób przetwarzania prowadzący do ich kompresji. Algorytmy stosujące metody semantyczne cechuje krótki czas wykonania, stabilność i znaczne wartości współczynnika kompresji. Zaletom tym towarzyszą jednak poważne wady: wspomniane już ograniczenie stosowania do danych o znanej strukturze, duża złożoność oprogramowania (większość danych wymaga oddzielnych algorytmów kompresji) oraz silny związek z konkretnym zastosowaniem. Metody statystyczne nie nakładają ograniczeń na postać danych wejściowych. Warunkiem ich efektywnego wykorzystania jest wcześniejsze (przed kompresją) zebranie informacji o statystycznych własnościach danych. Najczęściej sprowadza się to do ustalenia częstości występowania ciągów znaków w danych testowych.

Rozdział 2 poświęcony jest algorytmom kompresji struktur binarnych. Większość opublikowanych ostatnio algorytmów dotyczy kompresji tego rodzaju danych. Szczególnie dokładnie przedstawione są w tym rozdziale algorytmy kompresji wektorów binarnych, a więc te, z którymi porównywany jest proponowany przez autora algorytm.

W rozdziale 3 przedstawiono sposoby obliczania lub aproksymowania maksymalnej wartości współczynnika kompresji danych pochodzących z binarnych źródeł: bezpamięciowych, Markowa i złożonych. Do wyznaczenia minimalnej możliwej do osiągnięcia długości zapisu konieczne jest określenie entropii źródła generującego dane. Obliczenie entropii źródeł bezpamięciowych lub źródeł Markowa stosowanych do generowania struktur binarnych nie napotyka na większe problemy. Ciągi pochodzące z takich źródeł są jednak dobrym przybliżeniem dla stosunkowo niewielkiej

klasy struktur binarnych występujących w rzeczywistości. Z tego powodu do porównywania sprawności algorytmów kompresji wykorzystywane są dysponujące większymi możliwościami generatywnymi tzw. źródła złożone. Źródła te szczególnie dobrze nadają się do generowania ciągów mających własności zbliżone do własności wektorów binarnych występujących w systemach wyszukiwania informacji. Autor nie napotkał w literaturze metody obliczania minimalnej długości zapisu danych pochodzących ze źródeł złożonych i dlatego też w pracy zaproponował i zweryfikował sposób jej szacowania.

W rozdziale 4 zamieszczono opis i analizę sposobu działania proponowanego algorytmu kompresji wektorów binarnych oraz oznaczonych przez A czterech jego wersji nazwanych A_1 , A_w , A_2 i A_n . W czasie kompresji algorytm A przekształca wejściowy wektor X na dwa wektory wyjściowe Y i Z. Przeprowadzona analiza wykazała, że w przypadku przetwarzania bezpamięciowego wektora X, ilość informacji zawartej w Y i Z jest równa ilości informacji przekazywanej przez X. Algorytmy A_1 i A_2 przeznaczone są do kompresji wektorów bezpamięciowych, przy czym jedynie dla A_1 będącego najprostszą wersją algorytmu A zalecane jest ustalenie przed kompresją częstości występowania jedynek w X. Algorytmy A_2 i A_n przetwarzają dane w dwóch etapach. W pierwszym z nich kodowany jest X, a w drugim - utworzony wektor Y. A_n powstał z A_2 poprzez dodanie do niego kodowania predykcyjnego Y. Zadaniem kodowania predykcyjnego jest, przy niezminionej długości wektora, zmniejszenie liczby jedynek w nim zapisanych. W rezultacie A_n sprawnie kompresuje wektory pochodzące nie tylko ze źródeł bezpamięciowych.

W rozdziale 5 zamieszczono opis przeprowadzonego eksperymentu. Jego celem było skonfrontowanie rzeczywistych i przewidywanych

wartości współczynnika kompresji osiąganych przez algorytm A oraz porównanie ich z wartościami uzyskanymi przez inne algorytmy.

Dla potrzeb eksperymentu napisano programy generujące i testujące wektory binarne oraz je kompresujące za pomocą wszystkich wersji algorytmu A. Spotykane w literaturze opisy algorytmów nie zawsze zawierają dane pozwalające na porównanie ich z innymi algorytmami. Dlatego też w ramach eksperymentu napisano programy umożliwiające ustalenie współczynników kompresji osiąganych przez te algorytmy.

1. PRZEGLĄD METOD KOMPRESJI STOSOWANYCH W INFORMATYCE

Dążność do kompresji towarzyszyła zawsze procesowi przekazywania informacji. Zaobserwować można tu na przykładzie języka naturalnego. Skłonność do skracania długości wypowiedzi spowodowała: upraszczanie podstawowych struktur gramatycznych, powszechne stosowanie skrótów, skracanie wyrazów po ich wprowadzeniu do powszechnego użycia (np. automobil-auto) itp. Mimo działań tego typu nadmiarowość (redundancja) przekazu informacji w języku naturalnym jest zaskakująco duża [2, 3, 4].

Intensywne badania nadmiarowości przekazu wiadomości wiążą się z rozwojem telekomunikacji. Koszt przekazu wiadomości jest proporcjonalny do czasu trwania transmisji, kompresja daje zatem łatwe do obliczenia korzyści ekonomiczne. Metody kompresji stosowane w telekomunikacji wykorzystują:

- aproksymowanie wejściowych sygnałów uprzednio wybranymi funkcjami,
- próbkowanie sygnałów, a następnie transmitowanie tylko tych wartości, które różniły się dostatecznie od ostatnio przekazanej,
- zmniejszanie redundancji osiągane poprzez eliminowanie transmisji tych sygnałów, które mogą być ustalone przez analizę przekazanych już sygnałów lub porównanie ze wzorcem,
- kodowanie sygnałów; stosowane kody pozwalają często na wykrywanie i poprawianie błędów powstałych podczas transmisji.

Telekomunikacja ma za zadanie transmisję analogowych zwykle sygnałów przy zachowaniu ustalonego stopnia wierności przekazu. W przeciwieństwie do tego informatyka zajmuje się przetwarzaniem głównie dyskretnych danych, przy czym nie jest dopuszczalne z reguły jakiegokolwiek ich zniekształcenie w wyniku ich przekazu na odległość (transmisja) lub w czasie (zapis). Dlatego też w infor-

matyce stosuje się zwykle różnorodne warianty kodowania.

Kompresja możliwa jest dzięki ^(?)redundancji danych. Rozróżnia się redundancję treści i redundancję formy zapisu danych.

Nadmiarowość treści wynika z faktu, że dla konkretnego zastosowania nie wszystkie informacje zawarte w danych mają znaczenie. Często też z niektórych danych można zrezygnować, gdyż informacje w nich zawarte są do uzyskania na podstawie analizy pozostałych danych. Podział informacji na istotną i nieistotną musi być przeprowadzony przez człowieka. ^{Nadmiarowość}Redundancja treści jest wielkością względną odnoszoną do sposobu wykorzystania danych.

Nadmiarowość formy wynika z nieoptymalnego (z punktu widzenia długości zapisu) sposobu zapisania informacji. Wykrycie i ograniczenie tego typu redundancji jest możliwe po przeprowadzeniu statystycznej analizy własności danych wejściowych.

Stosując jako kryterium rodzaj usuwanej ^{Nadmiarowość}redundancji, metody kompresji wykorzystywane w informatyce można podzielić na metody semantyczne i statystyczne.

1.1. Metody semantyczne

Jak wspomniano we wstępie w metodach tej grupy projektant w pełni określa sposób działania algorytmu kompresji. Przy jego tworzeniu uwzględnia on wiedzę o sposobie wykorzystania danych, ich treści, strukturze oraz wzajemnych powiązaniach między danymi. W krańcowym przypadku może on podjąć decyzję o pominięciu w trakcie kompresji pewnych danych. Ma to miejsce wówczas, gdy są one nieistotne z punktu widzenia danego zastosowania lub też można je uzyskać na podstawie analizy pozostałych danych. Podejmowane są próby wykorzystania do kompresji informacji zawartych w formalnym opisie struktury danych, np. w definicji rekordów [5]. Algorytmy tworzone na podstawie tych metod charakteryzuje krótki czas wykonania, stabilność i znaczne wartości współczynnika kompresji. Zaletom tym towarzyszą jednak i poważne wady: ograniczenie za-

kresu wykorzystania algorytmów do danych o znanej strukturze, powodowanie wzrostu złożoności oprogramowania (większość danych wymaga oddzielnych algorytmów kompresji) oraz silny związek z konkretnym zastosowaniem. Poniżej przedstawiono kilka typowych metod należących do omawianej grupy.

1.1.1. Ograniczenie zakresu stosowanych znaków

W większości maszyn cyfrowych reprezentacja znaku zajmuje 8 bitowy bajt. Pozwala to na posługiwanie się $2^8 = 256$ różnymi znakami. Zwykle nie jest wymagana aż tak wielka ich liczba i dlatego powszechnie stosowane kody (np. EBCDIC, ASCII) nie wykorzystują znacznej części kombinacji wartości bitów w bajcie. Liczbę różnych stosowanych znaków można ograniczyć do 64, jeżeli zrezygnuje się z małych (lub dużych) liter oraz z części znaków specjalnych. Jeśli dane zawierają wyłącznie litery i podstawowe znaki interpunkcyjne, to wystarczające jest kodowanie 32 znaków. Przyjęcie takich ograniczeń pozwala na posługiwanie się słowami kodowymi liczącymi 6 lub 5 bitów. Wartości współczynnika kompresji osiągane dzięki takiemu prostemu przekształceniu wynoszą odpowiednio 1.333 oraz 1.6. [6]

1.1.2. Kodowanie liczb

W najprostszym przypadku zapisane znakowo liczby kompresuje się zamieniając je na postać binarną. Długość zapisu binarnego określa się na podstawie zakresu wartości oraz wymaganej dokładności przekształcenia kodowanych liczb.

Maszyny cyfrowe wykorzystujące 8 bitowy bajt dysponują możliwością przekształcenia znakowego zapisu liczby w postać spakowaną, w której na każdą cyfrę przypadają jedynie 4 bity. Zaproponowano też [7] efektywne algorytmy kodujące ciągi dwucyfrowe za pomocą

7 bitów, ciągi 3 cyfrowe - 10 bitami.

W pewnych zastosowaniach dopuszczalna jest zmiana wartości liczby w wyniku jej zakodowania, o ile nie zostanie przekroczona pewna ustalona uprzednio wielkość graniczna. Algorytm logarytmicznej kompresji liczb binarnych [8] konwertuje w prosty sposób liczby o długości do 19 bitów na standardowy 8 bitowy kod. Pierwsze 4 bity kodu zawierają pomniejszony o 4 licznik znaczących bitów wejściowych, zaś pozostałe 4 bity to odpowiednio drugi, trzeci i czwarty, najbardziej znaczący bit liczby wejściowej. Tak więc liczba 0000010110010101011 zostanie zakodowana jako 10100110.

Interesująca wydaje się być propozycja wykorzystania do zapisu liczb rzeczywistych systemu liczb Hackenbush (SLH) będącego modyfikacją binarnego zapisu liczb. Dla odróżnienia liczb binarnych i liczb z SLH te ostatnie kodowane będą literami L oraz R.

W SLH część całkowita liczby jest zapisywana unarnie (tzn. składa się z samych L lub samych R), zaś część ułamkowa-binarnie.

Liczby kodowane są ciągami stałej długości, wystarczającej do zapewnienia wymaganej dokładności. Wartość liczby z SLH oblicza się stosując poniższe równania:

1. $|RX| = - |L\bar{X}|$
2. $|LLX| = 1 + |LX|$
3. $|LRX| = (X)$
4. $|\text{ciąg pusty}| = 0$

gdzie

$|X|$ - wartość liczbowa ciągu X

$\bar{X}$ - ciąg, który powstał z X poprzez wzajemne zastąpienie L i R

(X) - ułamek binarny powstały z X po zastąpieniu L przez 1, R przez 0 oraz po umieszczeniu kropki ułamkowej na skrajnej lewej pozycji np. $(LLRL) = 0,1101$

Przykład 1

$$|RRRLRLLR| = - |LLLRLRRL| = -(1 + LLRLRRL) = -(2 + LRLRRL) = \\ = -(2 + 0,1001) = -2,5625$$

Jak wykazała przeprowadzona przez E. R. Berlekampa analiza [9], SLH zapewnia w wielu zastosowaniach znaczne skrócenie długości zapisu liczb przy jednoczesnym zachowaniu wymaganej dokładności przekształcenia liczba-kod.

1.1.3. Kodowanie powtarzających się znaków

W niektórych zastosowaniach dopuszczalna jest kompresja tekstu wejściowego, osiągana dzięki zastąpieniu ciągów spacji pojedynczym odstępem oraz usuwaniu nie znaczących zer ze znakowych zapisów liczb. W sytuacji, gdy dane zawierają wiele ciągów powtarzających się znaków oraz gdy niedopuszczalne jest zniekształcenie danych w wyniku kompresji, często stosuje się kodowanie długości ciągu (run-length coding). Ciąg takich samych znaków zastępowany jest wtedy pojedynczym znakiem, po którym następuje licznik długości ciągu. Licznik ten o długości 1 bajta musi być utworzony w ten sposób, by nie mógł być on zinterpretowany jako znak. [10]. W słowach kodowych pochodzących z EBCDIC drugi najbardziej znaczący bit jest zawsze równy 1 i dzięki temu bajty będące licznikami binarnymi o wartościach nie przekraczających 63 mogą być stosowane w tym algorytmie (bit ten jest wtedy równy 0). Kodowanie długości ciągów jest stosowane często przy kompresji struktur binarnych. Różne jego warianty omówione są dokładniej w następnym rozdziale.

1.1.4. Kody wyliczeniowe

Wszystkie dopuszczalne wartości kompresowanej danej są porządkowane. W trakcie kompresji w miejsce aktualnej wartości danej

umieszcza się jej kolejny numer. Zapis po kompresji ma zatem długość $\lceil \log_2 n \rceil$, gdzie n jest liczbą różnych wartości przyjmowanych przez daną. Często kodowane są w ten sposób daty: rok (dla ustalonego stulecia) wymaga 7 bitów, miesiąc - 4 zaś dzień - 5. Dzięki temu data zakodowana jest na 16 bitach, tj. 2 bajtach, podczas gdy zwykle wymagane jest do tego celu posłużenie się 10 bajtami (zapis postaci 1983.04.10) [11].

Przykład 2

Data w postaci wejściowej	Data w postaci skompresowanej
1983.04.10	$1010011\ 0100\ 01010_2 = 42634_{10}$
1910.10.15	$0001010\ 1100\ 01111_2 = 5519_{10}$

1.1.5. Algorytmy wykorzystujące uporządkowanie danych

Jeżeli rekordy w zbiorze uporządkowane są według rosnącej (malejącej) wartości pewnego pola, to wówczas można w tym polu umieścić tylko te znaki, które pozwalają na odróżnienie go od sąsiednich pól. Podejście takie zastosowano w eksperymencie, którego celem było określenie źródła Markowa (patrz pkt. 3.3) generującego ciągi o własnościach zbliżonych do języka naturalnego [12]. Wiadomościami elementarnymi wytwarzanymi przez to źródło były słowa, zaś maksymalny rozpatrywany jego rząd wynosił 7. Uzyskanie, przy takich założeniach, wiarygodnych wyników spowodowało konieczność przetwarzania tekstu wejściowego liczącego 5 mln słów. Zbiór wszystkich ciągów siedmiosłowych zawartych w danych miał długość 210 mln znaków. Eksperyment wymagał sortowania tych ciągów, co było praktycznie niemożliwe bez ich uprzedniej kompresji. Zastosowano kod wyliczeniowy, w którym każdemu słowu przyporządkowano liczbę będącą jego numerem kolejnym na liście częstości występowania wszystkich słów. Zamiana 5 mln słów na ich słowa kodowe wymagała zapamiętania w pamięci

operacyjnej pełnej, uporządkowanej alfabetycznie listy wszystkich słów występujących w danych. Możliwe stało się to dopiero po skompresowaniu jej algorytmem zamieniającym listę słów na listę ciągów znaków rozdzielonych liczbami. Liczba ta podaje ilość końcowych znaków słowa, które należy usunąć, tak by po dodaniu ciągów znaków następującego po liczbie uzyskać kolejne słowo.

Przykład 3

lista wejściowa

answer

answering

answers

antennae

antennas

⋮

lista skompresowana:

0 answer 0 ing 3 s 5 tennae 1 s

50 najczęściej występujących słów (stanowiących około 34% częstości występowania wszystkich słów) zapamiętanych było w postaci nie zmienionej w oddzielnej liście. Średni czas wymagany na zakodowanie słowa wynosił 5 ms, mimo że wymagało ono w większości przypadków odtworzenia fragmentu listy.

1.2. Metody statystyczne

Metody statystyczne nie nakładają ograniczeń na postać danych wejściowych. Stosowanie każdej z tych metod musi być poprzedzone zebraniem danych statystycznych o częstościach pojawiania się pojedynczych znaków lub też ich ciągów w danych testowych. Na tej podstawie ustalane są parametry algorytmu realizującego kompresję.

Oszacowanie wielkości maksymalnej kompresji wymaga uprzedniego zdefiniowania hipotetycznego źródła informacji generującego

dane wejściowe a następnie obliczenia jego entropii (patrz rozdział 3). Metody statystyczne stosowane są często do danych, w których przeważa język naturalny. Przeprowadzono wiele eksperymentów mających na celu oszacowanie wielkości jego entropii. Język naturalny przybliżany był ciągiem wiadomości wytworzonym przez źródła bezpamięciowe lub Markowa. Źródła te generowały pojedyncze znaki zgodnie z prawdopodobieństwami ustalonymi na podstawie analizy dostatecznie długiego tekstu wejściowego. Uzyskane wyniki przedstawione są w tab. 1. Rzeczywista ilość informacji przekazywana człowiekowi przez literę jest znacznie mniejsza i wynosi poniżej 2 bitów. Tak wielkie zmniejszenie ilości informacji jest spowodowane wykorzystywaniem przez człowieka wiedzy o semantycznych, statystycznych i syntaktycznych własnościach języka naturalnego.

Tab. 1. Wartości entropii źródeł przybliżających
język naturalny

Rodzaj źródła	Entropia (w bitach)
bezpamięciowe	4.03
Markowa I rzędu	3.32
Markowa II rzędu	3.1

W przedstawionych poniżej metodach kompresja osiągana jest dzięki:

- dostosowaniu długości słowa kodowego do częstości występowania kodowanych ciągów znaków,
- wybraniu zbioru ciągów znaków o równych w przybliżeniu częstościach występowania i następnie przydzieleniu im słów kodowych o stałej długości,
- przekształceniu danych do postaci umożliwiającej zastosowanie

algorytmów kompresji ciągów binarnych,
d) liczbowemu kodowaniu tekstu.

1.2.1. Kody o słowach kodowych zmiennej długości

Algorytm przydzielania wiadomościom elementarnym pochodzącym ze źródła bezpamięciowego kodów związanych (tj. kodów blokowych, jednoznacznie dekodowalnych oraz mających minimalną średnią długość słowa kodowego) przedstawił Huffman [13]. Opisana poniżej wersja algorytmu tworzy kody binarne.

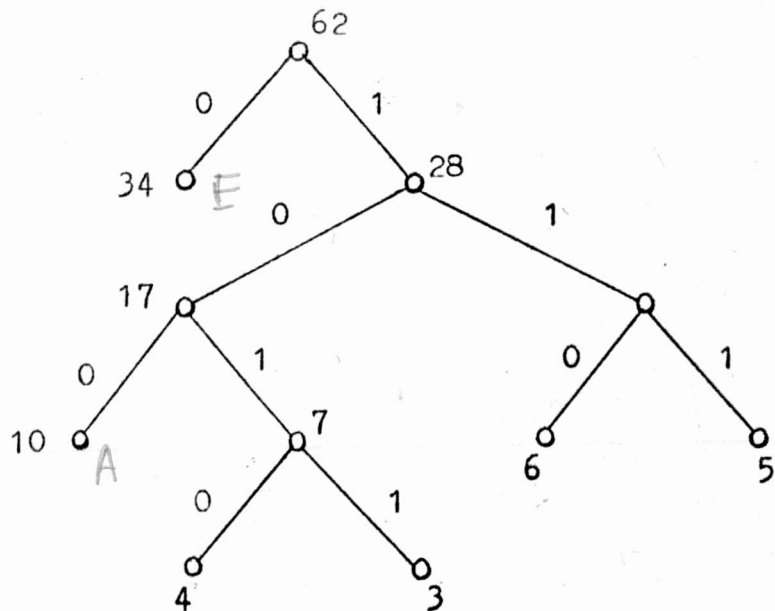
Niech n oznacza liczbę kodowanych wiadomości elementarnych. Algorytm wymaga wstępnego uporządkowania w kolejności malejącej częstości występowania wiadomości elementarnych. Praca algorytmu polega na tworzeniu drzewa binarnego [14]. Na początku każdej częstości przyporządkowuje się wzajemnie jednoznacznie wierzchołek. Przy budowaniu drzewa binarnego wykonywane są następujące czynności:

- sumowane są dwie najmniejsze częstości występowania
- dla nowo utworzonej częstości tworzony jest nowy wierzchołek, który połączony zostaje z wykorzystanymi w pkt a wierzchołkami krawędziami oznaczonymi w jednym przypadku przez 1 w drugim - przez 0,
- z listy częstości usuwa się dwie najmniejsze z nich oraz wprowadza się do niej częstość będącą ich sumą w taki sposób, by zachować jej uporządkowanie.

Proces ten powtarzany jest $n - 1$ razy, gdzie n jest liczbą wejściowych częstości.

Przykład 4

Znak	A	B	C	D	E	F
częstość występowania	10	3	6	4	34	5



Znak	A	B	C	D	E	F
słowo kodowe	100	1011	110	1010	0	111

Wartości entropii języka naturalnego podane w tab. 1 pozwalają na oszacowanie maksymalnego współczynnika kompresji, jaki można osiągnąć stosując kody Huffmana do kompresji typowego tekstu. Gdy założymy, że znak w wejściowym tekście reprezentowany jest przez 8 bitów, wówczas wartość współczynnika kompresji przy przydzielaniu słów kodowych pojedynczym literom nie może przekroczyć 1,98. Wartości współczynnika przy kodowaniu par i trójek znaków wynoszą odpowiednio 2.41 i 2.58. Osiągnięcie takiej kompresji okupione jest jednak koniecznością stosowania kilkuset bądź nawet kilku tysięcy słów kodowych zmiennej długości.

Podejmowane są próby wykorzystania kodów Huffmana w systemach minikomputerowych [15]. Ich wszechstronność pozwala na efektywne skracanie długości zapisu zbiorów mających zróżnicowane własności statystyczne, jak np.: programy źródłowe, programy wynikowe, zbiory danych.

Stosowanie kodów Huffmana wymaga znajomości częstości występowania kodowanych znaków. W sytuacjach, gdy częstości te są nieznane lub zmienne w czasie, proponowane jest stosowanie uniwersalnych schematów kodowania [16, 17]. Wykorzystywane tu algorytmy stosują bufor o ustalonym rozmiarze pamięci, w którym przechowywane są ostatnio przetwarzane dane. Aktualnie przetwarzane dane kodowane są poprzez wskazanie odpowiadających im fragmentów bufora. Alternatywne podejście zaprezentowane zostało przez Zivą i Lempela. Jak wykazała przeprowadzona przez nich analiza [18, 19] przedstawiony algorytm może uzyskać kompresję danych porównywalną z tą osiągniętą przez algorytm Huffmana przy pełnej znajomości wejściowych częstości. Wadą tych algorytmów jest znaczny stopień ich skomplikowania.

Maszyny cyfrowe przystosowane są do przetwarzania ciągów bitów o ustalonych długościach takich, jakimi są, np. bajty lub słowa maszynowe. Dlatego też kody Huffmana wymagają bardziej skomplikowanego oprogramowania i dłuższego czasu na kodowanie i dekodowanie niż kody mające stałą długość słowa kodowego. Ponadto nie skorygowany błąd w czasie jednego z tych procesów może spowodować błędne odtworzenie nie przetworzonego jeszcze fragmentu tekstu. Czynione są starania mające na celu ograniczenie tych wad, np. poprzez zwięzły zapis tablic służących do kodowania i dekodowania, umożliwiający jednocześnie przyspieszenie obu tych działań [20, 21]. Stosowane są również kody znacznie prostsze od kodów Huffmana, lecz mimo to pozwalające na uzyskanie dość znacznej kompresji danych tekstowych. Heaps [22] zaproponował algorytm który przydziela 7 najczęściej występującym znakom słowa kodowe o długości 3 bitów. Pozostałe znaki otrzymują 8 bitowe ciągi kodowe. Siedem najczęściej występujących znaków stanowi około 52 % wystąpień wszystkich znaków, zatem kodowanie to

pozwała na osiągnięcie współczynnika kompresji równego w przybliżeniu 1.48. W innej wersji algorytmu 13 najczęstszych znaków dostaje słowo kodowe długości 4 bitów, pozostałe znaki tak jak poprzednio - słowo o długości 8 bitów. Pozwala to na jeszcze większe skrócenie długości zapisu, współczynnik kompresji wynosi w przybliżeniu 1.66 (około 80% wystąpień wszystkich znaków przypada na 13 najczęstszych z nich).

1.2.2. Kody o słowach kodowych stałej długości (KSD)

W algorytmie Huffmana długość słów kodowych dostosowywana jest do różnych częstości występowania pojedynczych znaków bądź też ich par, trójek itp. Trudności w operowaniu ciągami bitów zmiennej długości skłaniają do posłużenia się kodem o słowach mających stałą długość. Prosty kod tego typu został przedstawiony przez P. Golenkę i A. Siemińskiego [23]. W zaproponowanym algorytmie kompresja jest uzyskana poprzez wzajemną zamianę kodów najczęściej występujących bigramów (par znaków) oraz najrzadszych pojedynczych znaków. W przeprowadzonym eksperymencie zamiana taka następowała w 28 przypadkach (wejściowa baza danych zapisana była za pomocą 6 bitowych znaków). Pozwoliło to temu bardzo szybkiemu i prostemu algorytmowi na osiągnięcie współczynnika kompresji wynoszącego w przybliżeniu 1.25. KSD posiada minimalną redundancję wówczas, gdy wszystkie kodowane za pomocą jednego słowa kodowego ciągi znaków występują z takimi samymi częstościami. Działania na ciągach bitów zmiennej długości zastąpione są przez znacznie łatwiejsze do oprogramowania działania na ciągach bajtów zmiennej długości.

Stosowanie KSD wymaga uprzedniego znalezienia zbioru o ustalonej z góry liczności, zawierającego ciągi znaków o zbliżonych częstościach występowania oraz ustalenia sposobu kodowania danych wejściowych.

Jeżeli algorytm stosowany jest do kompresji danych tekstowych wówczas zbiór ten może zawierać fragmenty tekstu (dowolne ciągi znaków) lub też fragmenty słów (ciągi składające się wyłącznie z liter). Fragmenty tekstu pozwalają na osiągnięcie wyższej kompresji, tym niemniej często stosowane są też fragmenty słów.

Dane w skompresowanej postaci mogą bowiem wtedy służyć do automatycznego indeksowania lub wyszukiwania informacji.

Liczba różnych ciągów wynosi najczęściej 256 - słowo kodowe ma wówczas długość 8 bitów i mieści się całkowitą liczbą razy w słowie maszynowym, co znacznie ułatwia oprogramowanie algorytmu kompresji. W skład zbioru wchodzi zawsze wszystkie pojedyncze znaki występujące w danych, dzięki czemu możliwe jest zawsze zakodowanie dowolnego ciągu wejściowego.

Algorytmy ustalające zbiór ciągów znaków można podzielić na dwie grupy. Algorytmy pierwszej z nich (zstępujące) zmniejszają w trakcie pracy, algorytmy zaś z drugiej grupy (wstępujące) zwiększają początkowy zestaw ciągów.

1.2.2. Algorytm zstępujący

Algorytm opisany przez E.J. Schnegrafa [24] rozpoczyna pracę od ustalenia częstości występowania w danych testowych wszystkich

ciągów znaków o długościach mniejszych od maksymalnej, a także ustalenia liczby różnych słów, w których ciągi te występują. Następnie każdemu z ciągów o długości ≥ 2 przypisuje się grupę ciągów. Tworzą ją wszystkie ciągi w nim zawarte, np. dla ciągu ABC grupa składa się z A, B, C, AB, BC. Algorytm stara się wybrać ciągi o jak największych długościach, występujące ze zbliżonymi częstościami oraz zminimalizować liczbę wyrazów nie zawierających ani jednego wybranego ciągu. W tym celu przeglądana jest lista ciągów, zaczynając od tych najdłuższych i występujących w jak największej liczbie różnych wyrazów. Ciąg jest włączany do zbioru, jeżeli częstość jego występowania oraz częstości występowania każdego z ciągów jego grupy przekraczają wielkość graniczną. Po wybraniu ciągu częstość jego występowania jest odejmowana od częstości każdego z ciągów jego grupy.

Przykład 5

Ciągi wraz ze swymi grupami	Wejściowe częstości występowania	Zmodyfikowane częstości występowania (po włączeniu ABC)
ABC	45	0
AB	47	2
BC	69	24
A	95	50
B	80	35
C	105	60
DBC	30	30
DB	50	50
BC	69	24

Ciągi wraz ze swymi grupami	Wejściowe częstości występowania	Zmodyfikowane częstości występowania (po włączeniu ABC)
D	67	67
B	80	35
C	105	60

Niech N oznacza maksymalną dopuszczalną długość ciągu (zwykle wynosi ona około 10), zaś p - graniczną częstość występowania ciągu. Wybieranie ciągów realizowane jest za pomocą 12 poniższych kroków.

Krok 1: $A := N$

Krok 2: Wybierz wszystkie ciągi o długości równej A i częstości występowania przekraczającej p . Nadaj b wartość równą liczbie ciągów spełniających te warunki.

Krok 3: Uporządkuj wybrane ciągi wg rosnącej częstości występowania, a następnie (przy równych częstościach) wg rosnącej liczby wyrazów zawierających ciąg.

Krok 4: $j := 0$

Krok 5: $j := j + 1$

Krok 6: Znajdź grupę ciągów przypisaną j -temu ciągowi.

Krok 7: Jeżeli jakikolwiek z ciągów grupy ma częstość mniejszą od granicznej, przejdź do kroku 9.

Krok 8: Włącz j -ty ciąg do słownika, odejmij wartość częstości jego występowania od częstości występowania ciągów jego grupy.

Krok 9: Jeżeli $j \neq b$, przejdź do wykonania kroku 5.

Krok 10: $A := A - 1$

Krok 11: Jeżeli $A \geq 2$, przejdź do wykonania kroku 2.

Krok 12: Umieść w zbiorze wybranych ciągów wszystkie pojedyncze znaki.

Zmiana wartości parametru p pozwala na regulowanie liczności uzyskanego zbioru.

1.2.2. Algorytm wstępujący

Zaproponowany przez Lyncha [25] algorytm wstępujący, rozpoczynając od zbioru zawierającego wszystkie pojedyncze znaki, dodaje do niego w kolejnych krokach wybrane ciągi znaków. Parametrem algorytmu jest graniczna częstość występowania ciągu oznaczana tak jak i w poprzednim algorytmie przez p . Nowo wprowadzone ciągi (powstałe przez konkatenację dwu ciągów umieszczonych już w zbiorze) są stosowane przy kodowaniu, jeżeli występują częściej niż p razy. Włączenie do zbioru nowych ciągów powoduje zmianę częstości stosowania zapisanych tam poprzednio ciągów, dlatego też algorytm w każdym kroku wykonuje próbną kompresję danych testowych. Możliwe jest dzięki temu wyeliminowanie stosowanych zbyt rzadko ciągów.

W każdym kroku algorytm wstępujący wykonuje następujące czynności:

- a) zliczenie częstości występowania w danych testowych wszystkich par ciągów ze zbioru,
- b) łączenie par o częstości przekraczającej p ; powstaje wtedy ciąg dodawany następnie do zbioru,
- c) kodowanie danych testowych za pomocą algorytmu AK2,
- d) zliczenie częstości stosowania w trakcie kodowania ciągów; ciągi wykorzystywane rzadziej niż p razy są usuwane ze zbioru.

Proces ten jest powtarzany tak długo, jak jest to konieczne do uzyskania wymaganej liczności zbioru. Przeprowadzone eksperymenty wykazały, że zbiór ciągów zapewnia większą kompresję,

jeżeli w kolejnych krokach wartość t jest obniżana w ten sposób, by za każdym razem dodawano do zbioru 10-20 nowych ciągów.

Kodowanie z wykorzystaniem ciągów zmiennej długości nie jest bynajmniej prostym zadaniem. Wykorzystując ustalony zbiór ciągów, dane można zakodować na wiele różnych sposobów. Każdy z nich musi zapewnić poprawne dekodowanie. Maksymalną długość skompresowanych danych uzyska się, gdy pojedyncze słowo kodowe przypisze się każdemu znakowi. Zadaniem przedstawionych trzech algorytmów jest zmniejszenie tej długości. Dwa pierwsze z nich przetwarzają dane wejściowe w blokach liczących N znaków. Algorytm AK1 rozpatruje wszystkie możliwe sposoby zakodowania bloku danych i wybiera ten, który zapewnia maksymalną kompresję. Algorytm AK2 stosuje zawsze słowo kodowe najdłuższego ciągu ze zbioru, przy czym ciąg ten musi być zawarty w niezakodowanym jeszcze fragmencie bloku. Algorytm AK3 nie wymaga podziału danych na bloki. Wykorzystuje on przy kompresji zawsze najdłuższy ciąg zbioru, jaki pokrywa się z nieskompresowanymi jeszcze początkowymi znakami tekstu.

Przykład 6

Dane wejściowe: KOMPRESJA TEKSTU

Długość bloku $N = 17$ znaków

Zbiór wybranych ciągów: PRESJA OMPR SJA T TU EKS RE ES

K O M P R E S J A T E U

Algorytm	Podział danych na ciągi	Długość skompenso- wanych da- nych
AK1	K OMP RE SJA T EKS TU	6
AK2	K O M PRESJA T EKS TU	7
AK3	K OMPR E S J A T EKS TU	9

Porównanie sprawności kodowania za pomocą tych trzech algorytmów umożliwia tab. 2. Jak widać różnice w uzyskiwanych współczynnikach kompresji nie są zbyt wielkie i w takiej sytuacji za jedyne kryterium może służyć czas kodowania. Pod tym względem zdecydowanie najlepszy jest algorytm AK3.

Tab. 2. Porównanie efektywności kodowania
algorytmów AK1, AK2, AK3 [26]

Algorytm	Względny czas kompresji	Współczynnik kompresji
AK1	1	1.62
AK2	0.456	1.59
AK3	0.3	1.53

1.2.3. Zastosowanie algorytmów kodowania ciągów binarnych do kompresji danych tekstowych

Lynch przedstawił [27] próbę wykorzystania algorytmów kompresji ciągów binarnych do skrócenia długości zapisu tekstu. Algorytmy te osiągają kompresję tylko wtedy, gdy dane wejściowe cechuje znaczne zróżnicowanie częstości występowania zer i jedynek. Dane tekstowe nie spełniają tego warunku i dlatego też konieczne jest ich wstępne przekształcenie.

Przed przystąpieniem do kompresji należy uporządkować znaki wg malejącej częstości występowania, następnie przekodować dane nie zmieniając ich długości (słowo kodowe ma taką samą długość jak wejściowa reprezentacja znaku) w ten sposób, by zmniejszyć liczbę jedynek w zapisie danych - słowo kodowe i -tego znaku na liście częstości zawiera nie więcej jedynek niż słowo kodowe jakiegokolwiek innego znaku o numerze kolejnym większym od i . Gdy za-

łożymy, że w danych wejściowych znak reprezentowany jest przez 6 bitów, to wówczas najczęstszemu znakowi przypiszemy 6 zer. Słowa kodowe znaków o numerach kolejnych na liście od 2 do 7 będą się składać z 5 zer i jednej jedynki itp. Przekształcenie takie nie zawsze zapewnia wystarczającą liczbę zer i dlatego Lynch zaproponował kodowanie w analogiczny sposób par znaków. Prawdopodobieństwa pojawienia się bitu równego 0 w kilku bibliograficznych bazach danych przed i po ich przekodowaniu są zamieszczone w tab. 3.

Tab. 3. Częstości występowania zer w wejściowych
i przekształconych bibliograficznych bazach
danych

Rodzaj danych	Baza		
	MARC	INSPEC	CHEMICAL TITLES
wejściowe	0.664	0.606	0.685
po przekodowaniu pojedynczych znaków	0.7512	0.762	0.833
po przekodowaniu par znaków	0.822	0.833	0.87

Kompresję przetworzonego tekstu uzyskiwano poprzez kodowanie długości ciągu następujących po sobie zer. Stosowano słowa kodowe stałej długości równej 3 lub w innej wersji algorytmu 4 bity. Do kodowania wykorzystano dwa algorytmy: konwencjonalnego kodowania długości runu^{*}) oraz algorytm Bradleya. Oba algorytmy są opisane w następnym rozdziale. Osiągnięte współczynniki kompresji są podane w tab. 4.

^{*}) W pracy, mianem runu określa się nieprzerwany ciąg zer zakończony jedynką lub nieprzerwany ciąg jedynek zakończony zerem.

Tab. 4. Wartości współczynnika kompresji bibliograficznych
baz danych kompresowanych algorytmem kodowania
długości ciągu

Sposób przetwarzania danych	Baza		
	MARC	INSPEC	CHEMICAL TITLES
przekodowanie par znaków	1,3869	1,3679	1,5723
przekodowanie poje- dynczych znaków	1,1627	1,189	1,385

1.2.4. Liczbowe kodowanie tekstu

Znaki w opisanym przez Hanna [28] algorytmie liczbowego kodowania tekstu otrzymują słowa kodowe zmiennej długości, przy czym długości te nie muszą być liczbami całkowitej długości. Algorytm wymaga uporządkowania w kolejności malejącej częstości występowania wszystkich znaków. Każdemu z nich przypisany jest ciąg $\left\lfloor \frac{i-1}{B} \right\rfloor + 1$ liczb całkowitych, pierwsze $\left\lfloor \frac{i-1}{B} \right\rfloor$ z nich to zera natomiast ostatnia ma wartość $i \bmod B + 1$, gdzie i jest numerem kolejnym znaku w liście częstości, a B jest parametrem algorytmu. W ten sposób lista została podzielona na grupy liczące po B znaków. W ramach każdej grupy znaki otrzymują kod takiej samej długości równej w przybliżeniu $\left(1 + \left\lfloor \frac{i-1}{B} \right\rfloor\right) \log_2(B + 1)$.

Na początku kompresji znaki są zastępowane przez odpowiadające im ciągi liczb całkowitych. Uzyskany ciąg można interpretować jako liczbę z $B + 1$ -narnego systemu liczenia. Kodowanie sprowadza się do konwersji kolejnych N cyfrowych fragmentów tej liczby na system binarny.

Przykład 7

Dla $B = 2$ i $N = 4$ oraz danych wejściowych: ABECB

znak	A	B	C	D	E	F
i	1	2	3	4	5	6
ciąg liczb	2	1	0 2	0 1	0 0 2	0 0 1

Po zastąpieniu znaków odpowiednimi ciągami liczb otrzymujemy:

21002021

Kodowanie:

$$2100/3 = 63/10 = 111111/2$$

$$2021/3 = 61/10 = 111101/2$$

Dane po kompresji: 111111111101

Wartość parametru B winna być ustalona w ten sposób, by częstości występowania znaków z różnych grup możliwie znacznie się różniły. Ze względu na prostotę oprogramowania algorytmu, wygodnie jest wybrać taką wartość N, by wszystkie wymagane działania arytmetyczne można było wykonać pojedynczą instrukcją języka wewnętrznego maszyny cyfrowej.

Brak ogólnie przyjętych danych testowych powoduje, że publikowane w artykułach wartości współczynnika kompresji nie są zawsze w pełni porównywalne. W większości przypadków algorytmy statystyczne stosowane do kompresji danych tekstowych pozwalają na co najwyżej dwukrotne skrócenie wejściowej długości zapisu. Najwyższą kompresję pozwalają osiągnąć trudne do stosowania kody Huffmana.

2. KOMPRESJA STRUKTUR BINARNYCH

Informacja w maszynach cyfrowych przedstawiana jest za pomocą elementów przyjmujących jeden z kilku dopuszczalnych stanów. Najłatwiejsze w realizacji i jednocześnie najbardziej niezawodne okazały się elementy o dwu wyróżnialnych stanach. Każdą informację zapamiętaną w maszynie cyfrowej można zatem interpretować jako liczbę z dwójkowego systemu liczenia.

Komputer może przetwarzać dane o ile zdefiniowany jest sposób reprezentowania ich przez takie liczby.

2.1. Struktury binarne

Standardowe sposoby zapisu określone są jedynie dla najczęściej stosowanych danych takich jak, np. liczby stałe - i zmiennoprzecinkowe czy też litery. W wypadku danych niestandardowych (zapis położenia figur na szachownicy, mapy meteorologicznej itp.) są one określane przez twórcę oprogramowania. Okazuje się, że w pewnych przypadkach przetwarzanie upraszcza się, jeżeli danym elementarnym przypisuje się jednocyfrowe liczby, tzn. słowo kodowe 0 lub 1. Zapisane w ten sposób dane nazywane są strukturą binarną.

Jak już podano we wstępie, struktury binarne wykorzystywane są najczęściej do przechowywania cyfrowych zapisów rysunków oraz listów adresów w Systemach Wyszukiwania Informacji (SWI).

Obecnie coraz częściej stosuje się przechowywanie i przesyłanie łączami telekomunikacyjnymi danych graficznych. Znajduje to zastosowanie przy organizowaniu telekonferencji, konwencjonalnym facsimile, elektronicznej poczcie, w bazach danych zawierających dane graficzne oraz w systemach powszechnej informacji.

Szczególnie mocno trend ten zaznacza się w Japonii. Stosowany tam alfabet składający się z wielu tysięcy znaków utrudnia posługiwanie się maszynami do pisania i przekazywanie danych tekstowych w postaci ciągu znaków. Dokumenty, w większości pisane ręcznie, muszą być w związku z tym transmitowane na podobnych zasadach jak rysunki techniczne. Komputery stosuje się nie tylko do prostego przekazywania, przechowywania i odtwarzania danych graficznych, lecz także do ich przetwarzania. Na przykład satelitarne obrazy powierzchni Ziemi przekształcane są często w ten sposób, że niewiele różniące się barwy w obrazie wejściowym reprezentowane są po przetworzeniu przez zdecydowanie różniące się kolory. Pozwala to człowiekowi na łatwą ocenę stanu upraw rolnych, stopnia zanieczyszczenia środowiska naturalnego itp. Opracowane już zostały programy umożliwiające nawet poprawianie niewłaściwie naświetlonych lub nieostrych zdjęć fotograficznych [29].

W celu uzyskania cyfrowego zapisu obrazu, w pierwszej fazie jest on rozkładany na elementy obrazu (EO). Przy zastosowaniu standardowej rozdzielczości na 1 mm^2 przypadają 64 EO. W następnej fazie koduje się analogowy sygnał świetlny pochodzący z każdego elementu. Powszechnie stosowane są słowa kodowe o długości 1 bitu, w którym EO uznane za jasne otrzymują ciąg kodowy 0, zaś czarne - 1. W specjalistycznych zastosowaniach (np. przetwarzanie zdjęć radiologicznych) konieczne jest wprowadzenie kilkudziesięciu pośrednich stopni szarości bądź też uwzględnianie kolorów. W dalszej części pracy rozpatrywane będą jedynie obrazy czarno-białe. Zapisywane są one w postaci dwuwymiarowej macierzy o elementach przyjmujących wartość równą 0 lub 1.

Listy inwersyjne są jednym z najczęściej stosowanych sposobów organizowania dokumentów w systemach wyszukiwania informacji. Dokumenty w systemie opisywane są za pomocą zbioru wartości wyróż-

nionych cech takich jak: autor, rok wydania, treść (zestaw słów kluczowych ujmujących zasadniczy temat dokumentu) itp. Metoda list inwersyjnych organizacji zbioru dokumentów polega na tworzeniu listy systemowych adresów opisów dokumentów posiadających wspólną wartość cechy. Adresem tym jest zwykle numer kolejny umieszczenia opisu w systemie. Każda wartość cechy posiada oddzielną listę. W pytaniu zadanym systemowi wyodrębniane są cechy i ich poszukiwane wartości oraz ich wzajemne powiązania. Adresy opisów dokumentów zgodnych z pytaniem uzyskuje się po wykonaniu operacji logicznych na odpowiednich listach.

Podstawową zaletą takiej organizacji zbioru dokumentów jest niewielki czas wymagany na wyszukiwanie informacji. Z tego powodu różne warianty list inwersyjnych są stosowane powszechnie w retrospektywnych systemach wyszukiwania informacji (systemach zawierających opisy dokumentów jakie ukazały się w określonej dziedzinie na przestrzeni kilku lub nawet kilkunastu lat) oraz w SWI pracujących w trybie bezpośrednim.

Listy adresów można zapisać w postaci ciągu liczb, lecz rozwiązanie takie posiada niedogodności: listy posiadają wówczas zmienną długość a operacje logiczne muszą być specjalnie oprogramowywane. Obie te przyczyny niekorzystnie wpływają na skomplikowanie i czas wykonania algorytmów przetwarzających listy. W alternatywnym rozwiązaniu każdemu dokumentowi przypisuje się bit o tym samym numerze kolejnym w każdej z list. Bit ten w takiej reprezentacji listy jest równy 1, gdy wartość cechy występuje o opisie tego dokumentu, i 0 - w przypadku przeciwnym. Dzięki temu wszystkie listy mają stałą długość równą liczbie dokumentów; ułatwia to też znacznie ich aktualizację. Maszyny cyfrowe mogą wykonywać operacje logiczne na ciągach bitów za pomocą pojedynczej instrukcji języka wewnętrznego. Pozwala to na uproszczenie

oprogramowania, skrócenie średniego czasu oczekiwania na odpowiedź, a także, jak wykazały badania [30], na zmniejszenie jego wariancji, co ma szczególne znaczenie w systemach działających w trybie bezpośrednim.

W obu powyższych wypadkach dane mają postać struktury binarnej o znacznych rozmiarach - SWI zawierają opisy dziesiątków tysięcy dokumentów, a cyfrowy zapis obrazu tworzyć może wiele milionów bitów. Dane te charakteryzuje duża redundancja zapisu, liczba zer znacznie przekracza liczbę jedynek, a ponadto są one pogrupowane. Wykorzystywany w systemie wyszukiwania informacji zbiór cech i wyróżnionych wartości ma poważny wpływ na efektywność działania całego systemu. Uwzględnianie wartości cech wspólnych dla znacznej części dokumentów lub przeciwnie dla bardzo niewielkiej ich liczby powoduje istotny wzrost liczby pamiętanych list inwersyjnych, któremu nie towarzyszy wzrost trafności i zupełności udzielanych odpowiedzi. W rezultacie, w działających systemach prawdopodobieństwo wystąpienia jedynki w strukturach reprezentujących listy inwersyjne nie przekracza zwykle 0.1. Ponadto, jak wykazały badania [31], pewne fragmenty tych struktur zawierają znacznie więcej jedynek niż pozostałe. Wynika to ze sposobu pracy bibliotek i ośrodków informacyjnych. Opracowują one nadchodzące materiały w partiach zawierających dokumenty o zbliżonej tematyce. Powoduje to nadawanie opisom mającym wiele wspólnych cech niewiele różniących się adresów. Przeciętny rysunek zawiera znacznie więcej punktów jasnych niż ciemnych. Znajduje to odzwierciedlenie w liczności zer i jedynek w jego cyfrowym zapisie. Zapis obrazu, w którym jedynki są rozmieszczone w sposób zupełnie przypadkowy niesie co prawda maksymalną ilość informacji (spośród wszystkich zapisów o takiej samej liczbie jedynek), lecz jest ona nieprzydatna dla odbiorcy. Większość istotnej semantycznie informacji przekazywana jest przez linie, którym

w cyfrowym zapisie odpowiadają ciągi sąsiadujących bitów o wartości 1.

Niezależnie od wyżej wymienionych podobieństw występują też różnice w własnościach struktur binarnych reprezentujących cyfrowe zapisy obrazów i listy inwersyjne. Zapis listy jest jednowymiarowy i dlatego struktury takie nazywa się wektorami binarnymi. We fragmentach wektora z gęściej zapisanymi jedynekami występuje stosunkowo wiele zer, po dekodowaniu konieczne jest otrzymanie wejściowej postaci danych. Zapis obrazu jest strukturą dwuwymiarową. Daje się zauważyć silna korelacja w występowaniu jedynek między kolejnymi wierszami tablicy. Ponadto są one zwykle pogrupowane w nierozdzielane zerami ciągi. Często też dopuszczalne jest uzyskanie po dekodowaniu uproszczonej, lecz mimo to czytelnej wersji rysunku. Proces tworzenia zapisu cyfrowego prowadzi do degradacji jakości obrazu i tym samym wierność przekazu nie ma tu tak istotnego znaczenia. Wektory binarne są transmitowane między pamięcią operacyjną a pomocniczą, zaś zapisy obrazów łączami telekomunikacyjnymi, w których prawdopodobieństwo wystąpienia nieskorygowanego błędu transmisji jest bez. porównania większe. Z tego powodu stosowane są często kody pozwalające na ograniczenie wpływu błędów przekazu na jakość obrazu.

Dalsza część rozdziału zawiera opisy przykładowych algorytmów kompresji struktur binarnych. Globalna ocena algorytmu winna uwzględniać:

- a) Rodzaj struktur, do kodowania których jest przeznaczony algorytm. Część algorytmów dostosowana jest do kompresji struktur pochodzących ze źródeł bezpamięciowych, inne starają się wykorzystywać grupowanie jedynek do zwiększenia współczynnika kompresji. Osiągane jest to poprzez: dostosowywanie parametrów działania algorytmu do własności przetwarzanych danych, zapa-

miętywanie w skompresowanym zapisie fragmentów wejściowych danych z zapisanymi jedynekami (lub też ich kodów) bądź też poprzez kodowanie predykcyjne.

b) Wartości współczynników kompresji.

Większość z przedstawionych w pracy wartości współczynnika kompresji pochodzi z artykułów opisujących algorytmy, pozostałe wartości zostały uzyskane w trakcie przeprowadzonego eksperymentu (patrz rozdział 5).

c) Stabilność kompresji

W niektórych artykułach zamieszczano dane o wpływie niedopasowania parametrów algorytmu do własności danych na wielkość współczynnika kompresji. Zależność tę starano się także ustalić w eksperymencie przeprowadzonym przez autora niniejszej pracy.

d) Czas pracy algorytmu

Na czas pracy algorytmów mają wpływ: złożoność algorytmów realizujących kompresję i dekompresję, wymagany stopień szczegółowości badania danych wejściowych i sposób ustalania parametrów działania algorytmu, a także liczba przebiegów koniecznych do przetworzenia danych.

2.2. Przegląd algorytmów kompresji struktur binarnych

Sześć pierwszych opisywanych algorytmów nie wykorzystuje przy kodowaniu zależności występujących między wierszami cyfrowego zapisu obrazu i dlatego też (pominąwszy kody Huffmana zaproponowane przez CCITT^{*)}) stosowanie ich do kompresji obrazów powinno być poprzedzone wstępnym jego przekształceniem za pomocą kodowania predykcyjnego.

Bezwarunkowe prawdopodobieństwa wystąpienia jedynek i zer w kodowanych strukturach binarnych oznaczane są odpowiednio przez

P_1 i P_0 .

^{*)} Międzynarodowy Komitet Doradczy dla Telegrafii i Telefonii.

2.2.1. Kod Huffmana

Kod Huffmana można zastosować do kompresji dowolnej struktury binarnej. Słowa kodowe przydzielane są n bitowym fragmentom struktury lub też długościom runów.

Kompresja struktur bezpamięciowych

Zwiększanie n w algorytmach kodujących fragmenty struktury powoduje zmniejszanie się redundancji kodu Huffmana, a tym samym wzrost kompresji. Towarzyszy temu jednocześnie wykładniczy wzrost liczby stosowanych słów kodowych. Prawdopodobieństwo wystąpienia jednej z postaci fragmentu zawierającej dokładnie k zer wynosi $P_0^k P_1^{n-k}$. Obliczone w ten sposób częstości pojawiania się każdej z 2^n możliwych postaci fragmentu są wykorzystywane do tworzenia słów kodowych algorytmem przedstawionym w rozdziale 1. Złożoność algorytmu uniemożliwia podanie dokładnej wartości l - średniej długości słowa kodowego. Natomiast wiadomo, że mieści się ona w granicach:

$$H(P_0) \leq 1 \leq H(P_0) + \frac{1}{n},$$

gdzie

$$H(P_0) = -P_0 \log_2 P_0 - P_1 \log_2 P_1$$

tym samym współczynnik kompresji spełnia poniższe nierówności:

$$\frac{n}{H(P_0)} \gg K_m \gg \frac{\frac{n^2}{nH(P_0)} + 1}{1}$$

Jak wykazały badania, dla $0.9 < P_0 < 0.98$ redundancja kodów Huffmana mniejsza jest od 0.19, jeżeli $n > 9$. Stosowane dla takich n kody zawierają kilkaset ciągów kodowych, tablice konieczne do kodowania i dekodowania osiągają znaczne rozmiary, a czas poświęcony na tworzenie słów kodowych wywiera wpływ na efektywność całego procesu. Wartości współczynnika kompresji i jego stabilność osiągana przy przetwarzaniu wektorów pochodzących ze źródeł

bezpamięciowych i złożonych są zamieszczone w tab. 5. Kod ten charakteryzuje się znacznym spadkiem sprawności towarzyszącym wzrostowi P_0 . Zaletą tego kodu jest natomiast dość duża stabilność współczynnika kompresji.

Kodowanie długości runów jest bardziej efektywne. Pozwala ono na osiągnięcie takiej samej sprawności kodu przy wykorzystaniu o wiele mniejszej liczby stosowanych słów kodowych. Niech d oznacza maksymalną długość runu zer kompresowanego jednym słowem kodowym. W kodzie, dodatkowy ciąg kodowy przewidziany jest dla nieprzerwanego ciągu zer o długości d bitów. Przy przetwarzaniu runu o długości $i > d$ koduje się najpierw pierwszych d jego zer, a pozostałe bity traktuje się jak run o długości równej $i-d$.

Przy przetwarzaniu struktur bezpamięciowych prawdopodobieństwo pojawienia się runu o długości i wynosi: $P_0^{i-1}P_1$, liczba zaś słów kodowych użytych przy jego kompresji: $\left\lceil \frac{i}{d} \right\rceil$.

W przeprowadzonym eksperymencie zbadano stabilność oraz wartości współczynników kompresji wektorów bezpamięciowych oraz pochodzących ze źródeł złożonych. Wyniki zamieszczone są w tab. 6.

Kompresja struktur niejednorodnych

W 1980 r. CCITT zaproponowała wykorzystanie kodów Huffmana do transmisji cyfrowych zapisów obrazów. Wymaganą szybkość działania uzyskuje się dzięki zastosowaniu opracowanych ostatnio algorytmów [32, 33]. Transmisja odbywa się wiersz po wierszu, po zakodowaniu każdego z nich zapisuje się sekwencję synchronizującą, składającą się z ciągu wartości bitów nie występujących w żadnym pojedynczym słowie kodowym ani też w dowolnej ich konkatenacji. Dzięki temu przekłamanie w trakcie transmisji powoduje błędne utworzenie tylko jednego wiersza. Kodowane są długości runów zarówno zer jak i jedynek. Runy te występują zawsze na przemian. Przypisywane im słowa kodowe pochodzą z oddzielnych tablic.

Tab. 5. Algorytm Huffmana (przydzielanie słów kodowych fragmentom wektora o długości 8 bitów) - uzyskane wartości k_m

wektory bezpamięciowe						wektory niejednorodne			
x_1	0,0998	0,0753	0,0498	0,0253	0,0102	0,0874	0,0652	0,0501	0,0454
k_m	2,139	2,599	3,396	4,822	6,378	2,508	3,143	3,824	4,115
k'_m	1,998	2,533	3,396	4,748	6,264	2,449	3,143	3,789	4,066

Uwaga: wartości k'_m uzyskano kodując wektory za pomocą słów kodowych ustalonych na podstawie analizy wektorów oznaczonych *

Tab. 6. Algorytm Huffmana (przydzielanie słów kodowych - długościom runów, $d = 100$) - uzyskane wartości k_m

wektory bezpamięciowe						wektory niejednorodne			
x_1	0,0998	0,0753	0,0498	0,0253	0,0102	0,0874	0,0652	0,0501	0,0454
k_m	2,132	2,603	3,529	5,945	12,666	2,471	3,259	4,444	5,172
k'_m	1,984	2,511	3,529	5,332	7,915	2,404	3,259	4,202	4,635

Ustalono je przy pomocy algorytmu Huffmana po przeprowadzeniu analizy długości runów w zapisach 8 obrazów uznanych przez CCITT za standardowe. Tablice kodowe wraz z obszernym uzasadnieniem oraz analizą przyjętych rozwiązań znaleźć można w artykule R. Huntera i H. Robinsona [34].

2.2.2. Konwencjonalne kodowanie długości runów (KKD)

Kodowanie długości runów zer odbywa się na takich samych zasadach jak w wypadku kodów Huffmana; stosowane jednak w KKD słowa kodowe mają stałą długość. Oznaczane przez d liczba różnych słów kodowych przekracza o 1 maksymalną długość runu kodowanego jednym słowem i dlatego wykorzystywane wartości spełniają zatem warunek:

$$d = 2^i - 1, \quad i \in \mathbb{N}$$

Przykład 8

Dla $d=3$, kod ma postać następującą:

run	długość runu	słowo kodowe
1	1	00
01	2	01
001	3	10
000	3	11

wektor wejściowy: 001000011000101001

wektor skompresowany: 1011010011000110

Niech $P(l)$ oznacza prawdopodobieństwo tego, że run zawierać będzie l zer.

$$P(L) = \begin{cases} P_0^L P_1, & 0 \leq L \leq d-1 \\ P_0^L, & L = d \end{cases}$$

Słowo kodowe nadawane jest średnio

$$\lambda = dP(d) + \sum_{i=0}^{d-1} (i+1)P(i) \text{ bitom wejściowym.}$$

Każdy z ciągów kodowych ma długość $A = \log_2(d + 1)$, zatem K_m wynosi:

$$K_m = \frac{\lambda}{A} = \frac{1 - P_0^d}{(1 - P_0) \log_2(d + 1)}$$

Rys. 1 przedstawia zależność między osiągniętym współczynnikiem kompresji i P_0 , dla $1 < d \leq 127$ [35]. Na tym rysunku każda z krzywych odpowiada jednemu kodowi, można więc na jego podstawie określić również stabilność współczynnika kompresji. Jak widać jest ona niewielka. Kody otrzymane dla $d=63$ lub $d=127$, sprawnie kompresujące ciągi w których $P_0 > 0.96$, cechuje redundancja przekraczająca 0.376 dla $P_0 = 0.9$.

Dużą zaletą algorytmu jest jego prostota zapewniająca znaczną szybkość przetwarzania. Algorytm ten jest dostosowany do kompresji wektorów pochodzących ze źródeł bezpamięciowych o niewielkim P_1 .

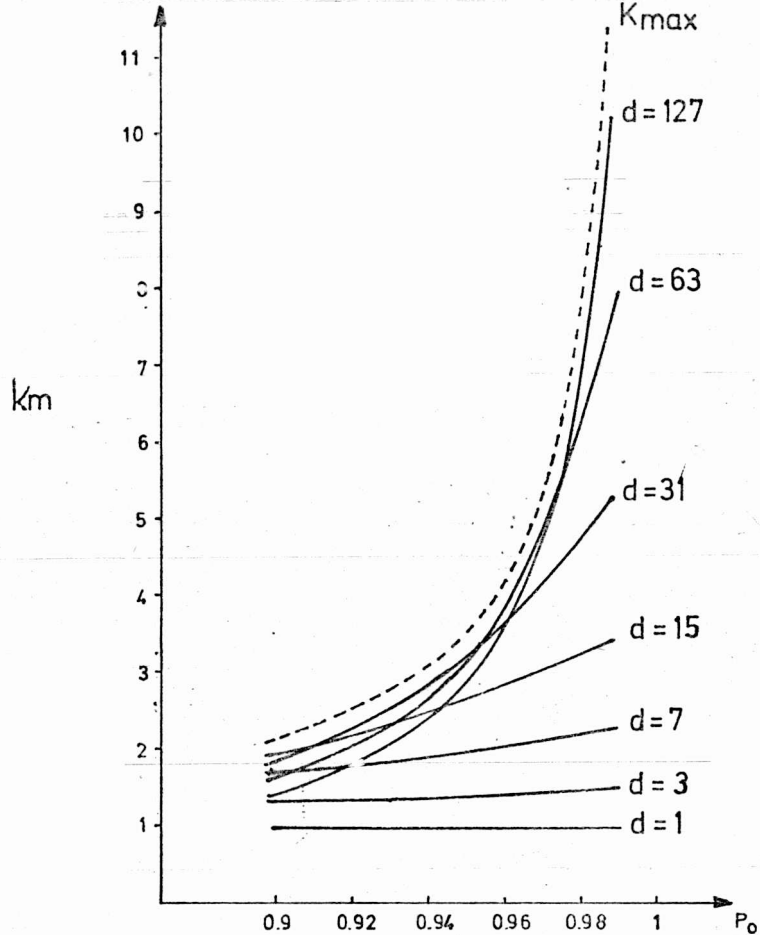
2.2.3. Kody Golomba

Kody Golomba stosuje się do kompresji wszystkich rodzajów wektorów binarnych. Także i w tym wypadku kompresja osiągana jest dzięki kodowaniu długości runów zer. Słowa kodowe mają zmienną długość, lecz ich prosta struktura umożliwia szybkie przetwarzanie oraz posługiwanie się tablicami o niewielkich rozmiarach.

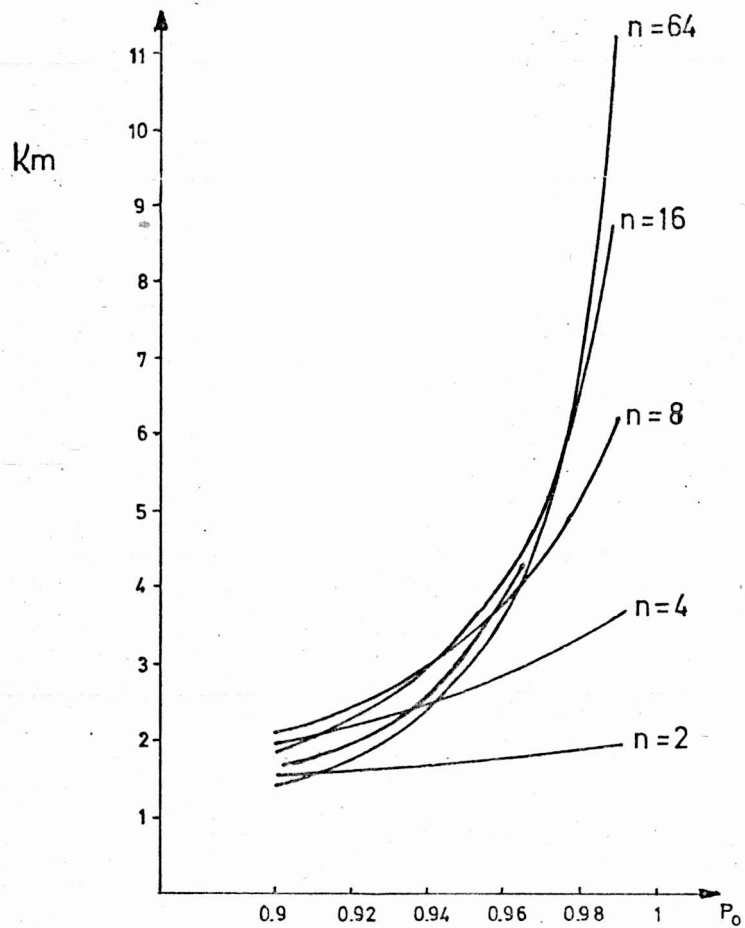
Kodowanie struktur bezpamięciowych

Jedynym parametrem algorytmu Golomba jest liczba naturalna n wybrana w ten sposób, że $P_0 \approx 0,5^n$. Działanie algorytmu zostanie przedstawione dla przypadku, gdy n jest potęgą 2. Niespełnienie tego warunku powoduje konieczność wprowadzenia modyfikacji [36].

Kod runu o długości l jest konkatencją:



Rys. 1. Algorytm KKD – osiągnięte wartości współczynnika kompresji wektorów bezpamięciowych.



Rys. 2. Algorytm Golomba – osiągnięte wartości współczynnika kompresji wektorów bezpamięciowych.

- a) kodu wstępnego składającego się z $(l-1) \text{div } n$ jedynek
- b) separatora - bitu o wartości 0
- c) kodu końcówki - binarnego licznika o wartości $(l-1) \bmod n$ i długości $\log_2(n)$ bitów.

Przykład 9

Dla $n = 2$, kod ma następującą postać:

długość runu	kod wstępny	separator	kod kończący
1	-	0	0
2	-	0	1
3	1	0	0
4	1	0	1
5	11	0	0
6	11	0	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$

wektor wejściowy: 0010110000010001

wektor skompresowany: 100 01 00 1101 101.

Średnia długość λ runu wynosi:

$$\lambda = \sum_{i=1}^{\infty} i P_0^{i-1} P_1 = \frac{1}{P_1},$$

natomiast średnia długość kodu jest równa:

$$L = \sum_{i=1}^{\infty} (\log_2 n + i) 2^{-i} = \log_2 n + 2,$$

tym samym

$$K_m = \frac{\log_2 n + 2}{P_1}$$

Na rys. 2 przedstawiono zależność między P_0 a współczyn-
nikiem kompresji dla $n = 2, 4, 8, 16, 64$ [35]. Podobnie jak to miało
miejsce w wypadku KKD, rysunek ten pozwala na jednoczesne osza-
cowanie stabilności kompresji. Kod Golomba wykazują wyższe niż
KKD wartości współczynników kompresji i niego lepszą jej stabilność

Kodowanie wektorów niejednorodnych

Zmodyfikowana wersja kodów Golomba przeznaczona do kompresji wektorów niejednorodnych pozwala na zmianę wartości n w trakcie kodowania runu. Dzięki temu różnicuje się liczbę zer, którym przypisuje się jedynekę w kodzie wstępnym a także długość kodu końcówki. Pozwala to na lepsze, niż w standardowej wersji algorytmu, dostosowanie sposobu kodowania danych do własności wektora. Przedstawione poniżej uogólnione kody Golomba (multi Golomb code) zostały zaproponowane przez L.R. Bahla i H. Kobayashi [35].

Przetwarzanie danych rozpoczyna się od ustalenia rzeczywistego rozkładu $P(l)$ w danych testowych, po czym określa się wartości funkcji $S(l)$ zdefiniowanej w sposób następujący:

$$S(l) = 1 - \sum_{i=0}^{l-1} P(i)$$

$S(l)$ jest zatem prawdopodobieństwem tego, że run zer osiągnie długość równą co najmniej l . Następnie wybiera się ciąg liczb naturalnych

$$m_1 = 2^{N_1}, m_2 = 2^{N_2}, \dots, m_k = 2^{N_k}, \text{ gdzie } N_1, N_2, \dots, N_k \in \mathbb{N}$$

takich, że:

$$S(m_1) \approx 0,5^1$$

$$S(m_1 + m_2) \approx 0,5^2$$

$$\vdots$$

$$S(m_1 + m_2 + \dots + m_k) \approx 0,5^k$$

Runy o długości i spełniające nierówności:

$$m_1 + m_2 + \dots + m_{k-1} \leq i < m_1 + m_2 + \dots + m_k$$

otrzymują słowa kodowe składające się z $k - 1$ jedynek, separatora (0) oraz N_k bitowego licznika o wartości $i - (m_1 + m_2 + \dots + m_{k-1})$

Autorzy modyfikacji zastosowali standardowe i uogólnione kody Golomba do kompresji cyfrowego zapisu 3 dokumentów. Sprawności kodów wynosiły odpowiednio 0,67 i 0,87. Przy obliczaniu entropii źródła generującego dane uwzględniono rzeczywisty rozkład długości runów zer.

Stabilność uogólnionych kodów Golomba oraz osiągnięte przez nie w czasie eksperymentu współczynniki kompresji są przedstawione w tab. 7.

Uproszczoną wersję uogólnionych kodów Golomba zaproponował J. Tenhola [37]. W wersji tej wszystkie liczby N_j spełniają warunek $N_j = N_{j-1} + 1$.

2.2.4. Algorytm iteracyjny

Algorytm iteracyjny jest stosowany do kompresji wektorów binarnych w systemach wyszukiwania informacji SKAT oraz SIR opracowanych na Politechnice Wrocławskiej. Jest on przeznaczony do kompresji wektorów niejednorodnych.

Wektor jest dzielony na k bitowe fragmenty. Fragment nazywany jest niezerowym wtedy gdy choć jeden z jego bitów jest równy 1. W trakcie kodowania tworzone są dwa ciągi: wynikowy i pośredni. Do ciągu wynikowego zapisywane są wszystkie niezerowe fragmenty. O w ciągu pośrednim reprezentuje fragment złożony z samych zer, zaś 1-fragmenty niezerowe. Po przetworzeniu w ten sposób całego wejściowego wektora, w następnym kroku jest kodowany wg tych samych zasad wektor pośredni. Proces ten kontynuowany jest tak długo, jak osiągane jest skrócenie całkowitej długości zapisu.

Przykład 10

Dla $k = 2$

wektor wejściowy: 0000000100100001

Tab. 7. Zmodyfikowany algorytm Golomba ($n = 300$) - uzyskane wartości k_m

	wektory bezpamięciowe					wektory niejednorodne			
x_1	0,0998	0,0753	0,0498	0,0253	0,0102	0,0874	0,0652	0,0501	0,0454
k_m	2,099	2,549	3,470	5,827	11,898	2,371	3,084	4,107	4,826
k_m'	1,923	2,468	3,470	5,028	4,440	2,349	3,084	3,795	3,959

Tab. 8. Algorytm iteracyjny - uzyskane wartości k_m

	wektory bezpamięciowe				wektory niejednorodne			
	x_1				x_1			
m	0,250	0,100	0,0503	0,0102	0,0872	0,0650	0,0499	0,0453
2	1,068	1,636	2,482	7,846	1,944	2,495	3,378	3,857
3	1,100	1,703	2,625	8,3195	2,113	2,714	3,758	4,317
4	1,074	1,695	2,543	8,048	2,051	2,722	3,729	4,337
5	1,038	1,655	2,420	7,519	1,975	2,597	3,614	4,208
6	1,009	1,585	2,298	7,097	1,963	2,529	3,529	4,108

ciąg pośredni: 00 01 01 0 1

ciąg wynikowy: 011001

całkowita długość zapisu: $8 + 6 = 14$

ciąg pośredni: 0 1 1 1

ciąg wynikowy: 01 10 01 01 01 01

całkowita długość zapisu: $6 + 6 + 4 = 16$

Po wykonaniu drugiego kroku nastąpiło zwiększenie całkowitej długości zapisu, zatem wektor skompresowany ma postać: 00010101 011001. Pierwszych 8 bitów tworzy ciąg pośredni, zaś ostatnich 6-wynikowy.

Jak wykazał przeprowadzony eksperyment, algorytm ten cechuje przy przetwarzaniu wektorów pochodzących ze źródeł złożonych, dość znaczna wielkość oraz stabilność współczynnika kompresji (patrz tab. 8). Wybór długości fragmentu (dla $2 \leq k \leq 6$) nie ma znacznego wpływu na wyżej wymienione współczynniki. Algorytm nie wymaga wstępnej analizy własności wektora, jego prostota (pomimo iteracyjnego sposobu działania) pozwala na dość szybkie przetwarzanie, szczególnie jeśli fragmentami będą bajty lub słowa maszynowe. Przy kodowaniu ciągów bezpamięciowych znaczna liczba zer zapisywanych w ciągu wynikowym obniża współczynnik kompresji.

2.2.5. Algorytmy Wedekinda-Hardera

Algorytmy Wedekinda-Hardera [38] przeznaczone są do kompresji wektorów niejednorodnych. Podobnie jak to miało miejsce w wypadku algorytmu iteracyjnego, skompresowany zapis zawiera fragmenty wektora wejściowego, w których wystąpiły bity równe 1. Każde ze słów kodowych ma długość k bitów.

W pierwszej wersji algorytmu (nazwanej WH1) skrajny lewy bit słowa kodowego określa sposób jego interpretacji. Gdy jest on równy 0, to pozostałych $k-1$ bitów słowa traktowanych jest jako binarny licznik długości runu zer bądź też jako licznik długości

części takiego runu (jeśli długość jego przekracza maksymalną wartość licznika). W przeciwnym razie bity te są fragmentem wektora wejściowego zawierającym co najmniej jedną jedynkę. Przy kodowaniu kilkoma ciągami tej samej długości runu możliwe jest pominięcie redundantnego wtedy pierwszego bitu słowa kodowego, co zwiększa współczynnik kompresji, lecz równocześnie wprowadza drugą dopuszczalną długość słowa kodowego. Rezultatem tego jest skomplikowanie oprogramowania algorytmu i wydłużenie czasu jego wykonania. Pracujący w ten sposób algorytm oznaczany jest symbolem WH1M.

Druga wersja algorytmu Wedekinda-Hardera (WH2) wykorzystuje dwa pierwsze bity słowa kodowego do określenia jednego z 4 poniższych sposobów interpretacji następnych bitów:

- a) kolejne $k-2$ bity to niezerowy fragment wejściowego wektora
- b) kolejne $k-2$ bity to licznik długości runu jedynek
- c) kolejne $k-2$ bity to licznik długości runu zer
- d) kolejne $2k-2$ bity to licznik długości runu zer

Współczynniki kompresji algorytmów Wedekinda-Hardera uzyskane przy kodowaniu wektorów pochodzących ze źródeł bezpamięciowych i złożonych przy optymalnym doborze parametrów, są zamieszczone w tab.19,20. Dane te pochodzą z artykułu Nevalainena. Jak widać WH1M osiąga wysokie współczynniki kompresji przy przetwarzaniu niejednorodnych wektorów.

2.2.6. Kod Bradleya

Kod Bradleya [39] jest uogólnieniem algorytmu KKD. Niech d oznacza maksymalną długość runu kodowanego jednym k bitowym słowem kodowym. Słowa kodowe zostały podzielone na dwie grupy. Te, które należą do pierwszej z nich są przypisane runom o długościach mieszczących się w przedziale $\langle 1, d \rangle$. Ciągi kodowe drugiej grupy odpowiadają

nieprzerwanym ciągom zer o długościach: $d, 2d, \dots, (2^k - d)d$. [39].

Przykład 11

Dla $k = 3, d = 5$, kod ma następującą postać:

	długość runu lub ciągu zer	słowo kodowe
I grupa	1	000
	2	001
	3	010
	4	011
	5	100

II grupa	5	101
	10	110
	15	111

wektor wejściowy: 011000000100000000000001

wektor skompresowany: 001000101001110010

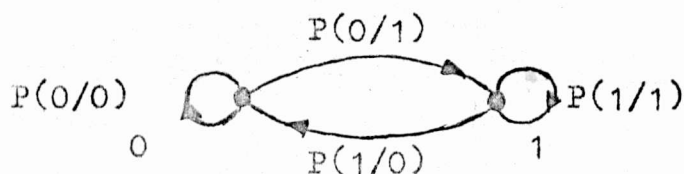
Jak widać KKD można traktować jako szczególny przypadek kodów Brandleya, dla których $d = 2^k - 1$. Kod Bradleya umożliwia osiągnięcie wysokich wartości współczynnika kompresji wektorów bezpamięciowych (patrz tab. 19).

Przedstawione poniżej trzy przykładowe algorytmy znajdują zastosowanie przy kompresji cyfrowych zapisów obrazów.

2.2.7. Kodowanie bloków

Zero-jedynkowa tablica cyfrowego zapisu obrazu zostaje w tym algorytmie [40] podzielona prostokąty (bloki) o wymiarach $m \times n$ bitów. Analiza typowych danych wskazuje, że dla niewielkich wartości m oraz n , bloki składające się z samych zer występują zdecydowanie najczęściej. Algorytm kompresuje dane, przydzielając każdemu takiemu blokowi jednobitowy kod 0. Słowo kodowe bloku niezerowego utworzone jest z $m \times n$ bitów składających się na ten blok poprze-

dzonych przedrostkiem równym 1. Przy szacowaniu wielkości współczynnika kompresji zakłada się, że wejściowa struktura binarna została wygenerowana przez źródło Markowa pierwszego rzędu o poniższym wykresie stanów (patrz rozdział 3):



Niech $P(0; m, n)$ oznacza prawdopodobieństwo wystąpienia w danych bloku składającego się z samych zer.

$$P(0; m, n) \approx (P(0)P(0/0)^{m-1})^n$$

gdzie $P(0)$ jest bezwarunkowym prawdopodobieństwem pojawiania się 0 w cyfrowym zapisie obrazu. Współczynnik kompresji wynosi:

$$K_m = \frac{mn}{mn(1 - P(0; m, n)) + 1}$$

W jednej z wersji tego algorytmu stosuje się kodowanie "psychowizualne", w którym bloki zawierające mniej jedynek niż wynosi pewna zadana z góry wartość są traktowane tak jak zerowe. Powoduje to wzrost $P(0/0)$, a tym samym wzrost współczynnika kompresji bez istotnego pogorszenia jakości odtworzonego obrazu.

2.2.8. Wyodrębnianie znaków

Opisany przez W.K. Pratt [41] algorytm dzieli tablicę z cyfrowym zapisem obrazu na bloki o wymiarach od dwu - do czterokrotnie przekraczających spodziewane rozmiary umieszczonych na obrazie znaków. Następnie w ramach bloku wyszukiwane są nieprzerwane ciągi jedynek reprezentujące linie. Każda z linii opisywana jest zbiorem wartości cech. Jeśli spełniają one zadane warunki, to linia traktowana jest tak jak znak. Opisy znaków stosowanych

przy kompresji są umieszczone w bibliotece. Opis nowo zidentyfikowanego znaku jest porównywany z opisami z biblioteki i jeśli nie jest on dostatecznie podobny do któregoś z nich, to jest on w niej umieszczany. W trakcie uaktualniania zawartości biblioteki stosuje się usuwanie z niej opisów niedostatecznie często wykorzystywanych. Słowo kodowe znaku zawiera identyfikator jego położenia w ramach dokumentu oraz kolejny numer jego opisu w bibliotece.

Jedynki w zapisie dokumentu nieprzetwarzane jako znaki są kodowane jednym ze znanych algorytmów kodowania długości runów. Wadą algorytmu jest jego duże skomplikowanie i wynikający stąd znaczny czas kodowania, w zamian za to osiąga on większe niż inne algorytmy współczynniki kompresji.

2.3. Kodowanie predycyjne

Kodowanie predycyjne nie zmienia długości zapisu obrazu, jego zadaniem jest zmniejszenie liczby jedynek występujących w strukturze binarnej [42]. Jak już to wspomniano poprzednio, w cyfrowych zapisach obrazów daje się zauważyć silna korelacja wartości sąsiadujących bitów. Dzięki temu, znając wartości elementów tablicy otaczających interesujący nas bit, można ze znacznym prawdopodobieństwem przewidzieć jego wartość.

Niech EO_{ij} , EX_{ij} i OU_{ij} oznaczają odpowiednio rzeczywistą, przewidywaną i wynikową (kompresowaną) wartość bitu położonego w i -tym wierszu i j -tej kolumnie tablicy. Zbiór k bitów wykorzystywany przy odgadywaniu wartości bitu jest nazwany pamięcią algorytmu. Tworzyć ją mogą jedynie bity uprzednio już przetworzone. Przykładowe 2, 4 i 7 elementowe pamięci algorytmu są przedstawione na rys. 3. Zwiększanie rozmiarów pamięci algorytmu umożliwia trafniejsze przewidywanie, lecz jednocześnie wydłuża czas kodowania predycyjnego.

Stosowane są dwa sposoby przewidywania wartości bitów. W pierwszym z nich analizuje się dane wejściowe ustalając, która z wartości odgadawanego bitu częściej towarzyszy każdej z 2^k możliwych postaci pamięci algorytmu. Wartości te tworzą wzorzec stosowany następnie przy kodowaniu i dekodowaniu predykcyjnym. Musi być on transmitowany (przechowywany) łącznie ze skompresowanymi danymi.

Drugi ze sposobów przewidywania wartości bitów pozwala na pominięcie wstępnej analizy obrazu. Niech $x_1, x_2, \dots, x_N$ oznacza zbiór wartości bitów w pamięci algorytmu, np. dla przypadku przedstawionego na rys. 3b są nimi:

$$x_1 = EO_{i-1,j-1} = 1, \quad x_2 = EO_{i-1,j} = 1,$$

$$x_3 = EO_{i-1,j+1} = 0, \quad x_4 = EO_{i,j-1} = 1$$

$N + 1$ wymiarowy wektor X oraz funkcja $F(X)$ są zdefiniowane w sposób następujący:

$$X = (1, x_1, x_2, \dots, x_N), \quad F(X) = \sum_{i=1}^N W_i X_i + W_0$$

gdzie $W = (W_0, W_1, W_2, \dots, W_N)$, $W_i \in R$

Wartości bitu przewiduje się zgodnie z poniższą regułą decyzyjną:

$$EX_{ij} = \begin{cases} 0 & ; \quad F(X) < 0 \\ 1 & ; \quad F(X) \geq 0 \end{cases}$$

Niezgodność przewidywanej i rzeczywistej wartości bitu powoduje zmianę wartości wektora W na W'

$$W' = \begin{cases} W + \alpha X, & \text{jeśli } EO_{ij} = 1 \text{ i } EX_{ij} = 0 \\ W - \alpha X, & \text{jeśli } EO_{ij} = 0 \text{ i } EX_{ij} = 1 \end{cases}$$

Wielkość parametru α wpływa na szybkość dostosowywania się algorytmu do własności przetwarzanych danych. Opracowano wiele reguł ustalania jego wielkości [43, 44], w najprostszym przypadku jest

ona dodatnią stałą rzeczywistą.

W trakcie kodowania predykcyjnego porównuje się przewidywaną i rzeczywistą wartość bitu. Prawidłowe odgadnięcie jego wartości powoduje zastąpienie go przez 0, błędne zaś przez 1. Podczas dekodowania predykcyjnego napotkanie bitu o wartości 0 nadaje mu wartość zgodną z przewidywaną zaś równego zeru przeciwnej.

Działania te przebiegają zatem zgodnie z poniższymi równaniami:

Kodowanie: $OU_{ij} = EX_{ij} \oplus EO_{ij}$

dekodowanie: $EO_{ij} = EX_{ij} \oplus OU_{ij}$

gdzie $\oplus$ oznacza dodawanie modulo 2.

Przykład 12

Dla $k = 2$, przy przewidywaniu wartości EO_{ij} uwzględniane są wartości $EO_{ij,j-1}$ oraz $EO_{i-1,j}$, stosowany jest pierwszy sposób przewidywania.

Pamięć		Wzorzec
$EO_{i,j-1}$	$EO_{i-1,j}$	
0	0	0
0	1	0
1	0	0
1	1	1

Wejściowy fragment struktury binarnej:

0 1 0 1 1 1 0 0 1 1 0

0 1 0 1 1 1 0 1 1 1 1

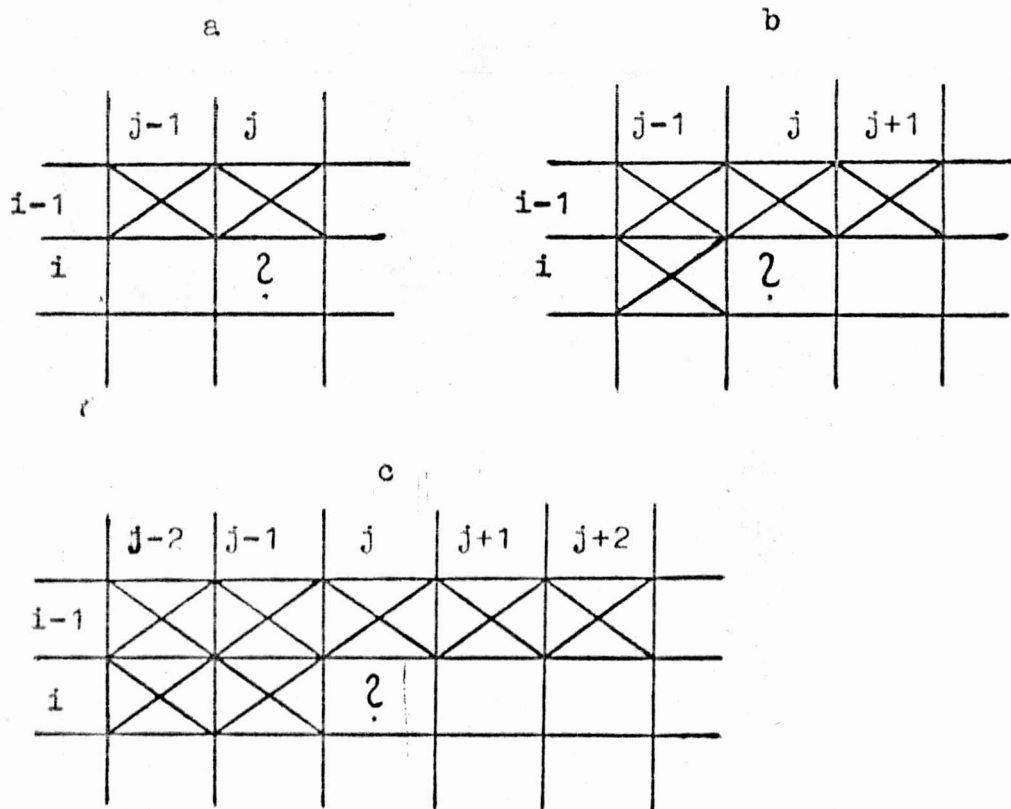
Struktura binarna po kodowaniu predykcyjnym:

0 1 0 1 1 1 0 0 1 1 0

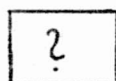
0 1 0 1 0 0 0 1 0 0 0

Kodowano jedynie podkreślone bity.

Po przetworzeniu zapisu obrazu w ten sposób, jest on kodowany jednym z algorytmów kompresji wektorów binarnych.



Rys.3. Przykładowe 2,4 i 7 elementowe pamięci algorytmu kodowania predykcyjnego



odgadywany EO



EO należący do pamięci algorytmu

3. TEORETYCZNE PODSTAWY KOMPRESJI

Algorytmy kompresji struktur binarnych zmniejszają redundancję sposobu zapisu danych. Przy badaniu sprawności algorytmów stosowane są zarówno dane rzeczywiste, jak i tworzone sztucznie. Stosowanie danych rzeczywistych związane jest z dwoma niedogodnościami: koniecznością znalezienia reprezentatywnego zbioru danych rzeczywistych oraz napisania programu realizującego kompresję dla każdego z badanych algorytmów. Znaczenie tych niedogodności zmniejsza istnienie ogólnie przyjętych danych testowych. Próba wprowadzenia takich danych została podjęta w 1980 r. przez CCITT. Organizacja ta opublikowała 8, uznanych za standardowe dokumentów, których stosowanie zaleca się przy porównywaniu sprawności algorytmów kompresji cyfrowych zapisów obrazów. Dane tego typu pozwalają na uszeregowanie algorytmów, natomiast nie na określenie tego, na ile uzyskiwane wartości współczynników kompresji zbliżają się do maksymalnych możliwych do uzyskania. Dla wyznaczenia tych ostatnich konieczne jest wykorzystanie danych wytwarzanych sztucznie oraz posłużenie się metodami stosowanymi w teorii informacji. Zadanie to mieści się w ramach klasycznej ilościowej teorii informacji i dlatego w pracy nie będą rozpatrywane wartościowe lub jakościowe [45] aspekty informacji zawartej w przetwarzanych danych. Sformalizowanie procesu wytwarzania danych ułatwia porównanie sprawności algorytmów opracowanych w różnych ośrodkach oraz pozwala na wyznaczenie maksymalnych wartości współczynnika kompresji i teoretyczne oszacowanie sprawności algorytmów. Do generowania struktur binarnych stosowane są źródła bezpamięciowe, Markowa i złożone. Dane mające własności struktur bezpamięciowych występują dość rzadko w praktyce. Powszechne stosowanie źródeł bezpamięciowych spowodowane jest znacznie łatwiejszą, niż w wypadku pozostałych

rodzajów źródeł, teoretyczną analizą sposobu działania algorytmu. Zgodnie z propozycją przedstawioną przez Nevalainena [31], do generowania ciągów mających cechy wektorów binarnych spotykanych w systemach wyszukiwania informacji wykorzystano źródła bezpamięciowe i złożone.

3.1. Ilość informacji jako funkcja prawdopodobieństwa wystąpienia zdarzenia

Ilość informacji przekazana w wyniku zaobserwowania zdarzenia jest wielkością obiektywną. Jeżeli zdarzenie A ma większe prawdopodobieństwo wystąpienia niż B, to zgodnie z intuicją zaistnienie B winno przekazać większą ilość informacji niż zaistnienie A. Miernikiem ilości informacji związanej z zaobserwowaniem zdarzenia powinna być zatem funkcja odwrotnie proporcjonalna do prawdopodobieństwa jego wystąpienia. Zaobserwowanie łącznego wystąpienia zdarzeń niezależnych powinno dostarczyć ilości informacji równej sumie ilości informacji wynikłej ze stwierdzenia pojedynczego wystąpienia tych zdarzeń. Powyższe wymagania spełnia funkcja:

$$I(A) = \log \frac{1}{P(A)}$$

gdzie $I(A)$ - ilość informacji dostarczona poprzez zaobserwowanie wystąpienia zdarzenia A

$P(A)$ - prawdopodobieństwo wystąpienia zdarzenia A.

Wybór podstawy logarytmu określa rodzaj stosowanej jednostki ilości informacji, a wzajemne przeliczenia między nimi umożliwia zależność:

$$\log_a X = \frac{1}{\log_b a} \log_b X$$

Najczęściej stosowany jest logarytm o podstawie 2, w dalszej części pracy oznaczany przez lb. Jednostka ilości informacji naz-

wana jest wtedy bitem. Stosując logarytm naturalny mierzymy ilość informacji w natych zaś przy podstawie równej 10-w hartleyach.

Pomiar ilości informacji zawartej w ciągu zaobserwowanych zdarzeń (wiadomości) wymaga precyzyjnego sformułowania zasad ich wytwarzania. Matematycznym modelem tego procesu jest źródło informacji. Jest to obiekt, który z wiadomości elementarnych tworzących skończony zbiór $S = \{s_1, s_2, \dots, s_q\}$ wytwarza zgodnie z ustalonym rozkładem prawdopodobieństw ciągły wiadomości elementarnych. W dalszej części rozdziału przedstawione zostaną 3 rodzaje takich źródeł: źródła bezpamięciowe, ergodyczne źródła Markowa oraz źródła złożone. Opisane też zostaną sposoby obliczania lub szacowania średniej ilości informacji przypadającej na wytworzoną przez nie wiadomość elementarną. Wybierając odpowiedni rodzaj źródła oraz parametry jego działania, można uzyskać ciąg wiadomości zbliżony własnościami statystycznymi do przetwarzanych danych.

3.2. Bezpamięciowe źródło informacji

Bezpamięciowe źródło informacji to źródło S wytwarzające wiadomości elementarne $s_1, s_2, \dots, s_q$ ze stałymi prawdopodobieństwami równymi odpowiednio $P_b(s_1), P_b(s_2), \dots, P_b(s_q)$. Średnia ilość informacji przypadająca na wiadomość elementarną wygenerowaną przez to źródło wynosi:

$$H(S) = - \sum_{i=1}^q P_b(s_i) \log P_b(s_i).$$

Ponieważ

$$\lim_{a \rightarrow 0} a \log a = 0,$$

zatem na mocy definicji przyjmuje się:

$$0 \log 0 = 0$$

$H(S)$ nazywa się entropią źródła. Można ją interpretować jako średnią miarę niepewności obserwatora, co do kolejnej wiadomości

elementarnej generowanej przez źródło. Entropia przyjmuje minimum równe 0 dla źródła wytwarzającego stale taką samą wiadomość elementarną. Maksymalną entropię wynoszącą $\log q$ osiągają źródła, w których prawdopodobieństwa wytworzenia wszystkich wiadomości elementarnych są takie same [46].

Duże znaczenie teoretyczne i praktyczne ma najprostsze źródło bezpamięciowe o dwuelementowym zbiorze wiadomości elementarnych zwane binarnym. Wykres przedstawiający entropię źródła binarnego $S = \{0,1\}$ jako funkcję $P_b(0)$ jest zamieszczony na rys. 4.

3.3. Ergodyczne źródło Markowa

Źródła Markowa są uogólnieniem źródeł bezpamięciowych. W danej chwili źródło Markowa $S = \{s_1, s_2, \dots, s_q\}$ m-tego rzędu znajduje się w jednym z q^m możliwych stanów wyznaczonym przez m ostatnio wytworzonych wiadomości elementarnych. Z każdym stanem związany jest rozkład bezwarunkowych prawdopodobieństw generowania wiadomości elementarnych. Opis takiego źródła zawiera wykaz generowanych wiadomości elementarnych oraz q^{m+1} prawdopodobieństw warunkowych o postaci:

$$P(s_i | s_{j1} s_{j2} \dots s_{jm})$$

gdzie $s_i, s_{j1}, s_{j2}, \dots, s_{jm} \in S$

$P(s_i | s_{j1}, s_{j2}, \dots, s_{jm})$ jest prawdopodobieństwem wytworzenia s_i pod warunkiem, że m poprzednio wygenerowanymi wiadomościami elementarnymi były $s_{j1}, s_{j2}, \dots, s_{jm}$.

Jest to zapis mało czytelny i dlatego często stosuje się tzw. wykresy stanów źródła. Wykres taki to zorientowany graf ważony. Każdemu ze stanów odpowiada inny węzeł, a krawędzie są oznaczone odpowiednimi prawdopodobieństwami warunkowymi.

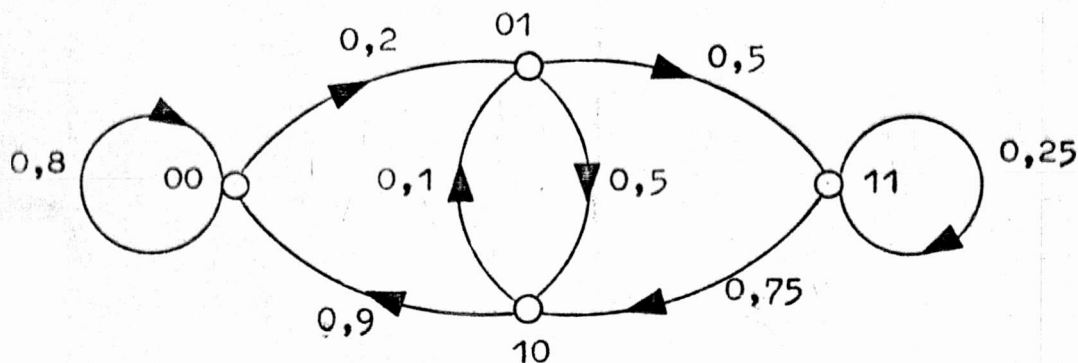
Przykład 13

$S = \{0,1\}$ jest źródłem Markowa 2-go rzędu.

Prawdopodobieństwa warunkowe przyjmują wartości:

$$P(0|00) = 0,8; \quad P(0|01) = 0,5; \quad P(0|10) = 0,9; \quad P(0|11) = 0,75$$

Wykres stanów powyższego źródła ma następującą postać:



Zdefiniowanie **ergodycznych źródeł** Markowa musi być poprzedzone wprowadzeniem kilku innych definicji [47].

Źródło nazywamy **nieprzewielnym**, gdy każdy stan można osiągnąć z dowolnego stanu.

Niech $P_j(k)$ oznacza prawdopodobieństwo tego, że źródło po wytworzeniu k wiadomości elementarnych znajdzie się ponownie w stanie j , zaś $f_j(k)$, że będzie to pierwszy powrót tego typu. Wówczas $f_j = \sum_{k=1}^{\infty} f_j(k)$ określa prawdopodobieństwo tego, że wychodząc ze stanu j , źródło kiedykolwiek do niego powróci. Stan j nazywamy **powracającym**, jeśli $f_j = 1$. Średni cykl powrotu μ_j definiuje się jako sumę:

$$\mu_j = \sum_{k=1}^{\infty} k f_j(k)$$

Jeżeli $\mu_j = \infty$, to stan j nazywany jest **zerowym**.

Stan j ma okres t , jeżeli t jest najmniejszą liczbą naturalną taką, że $P_j(d) = 0$, dla d niepodzielnych przez t . Stany powracające, które nie są ani zerowe, ani okresowe nazywamy **ergodycznymi**, a nieprzewielne źródło składające się wyłącznie z takich stanów źródłem **ergodycznym**. Źródła ergodyczne - ujmując to nieformalnie - to źródła, które niezależnie od charakteru początkowo wytwarzanych wiadomości generują z prawdopodobieństwem równym 1 pewien typowy

ciąg wiadomości. Mianem rozkładu stacjonarnego określa się bezwarunkowe prawdopodobieństwa wystąpienia wiadomości elementarnych w typowym ciągu wiadomości. W dalszej części pracy rozpatrywane są tylko ergodyczne źródła Markowa.

Średnia ilość informacji dostarczana przez źródło będące w stanie $s_{j1}, s_{j2}, \dots, s_{jm}$ wynosi:

$$H(S|s_{j1}, s_{j2}, \dots, s_{jm}) = - \sum_{i=1}^q P(s_i | s_{j1}, s_{j2}, \dots, s_{jm})$$

$$\log P(s_i | s_{j1}, s_{j2}, \dots, s_{jm})$$

Po uśrednieniu tej wielkości na q^m możliwych stanów otrzymujemy entropię źródła Markowa m -tego rzędu:

$$H(S) = \sum_{q^m} P(s_{j1}, s_{j2}, \dots, s_{jm}) H(S|s_{j1}, s_{j2}, \dots, s_{jm})$$

gdzie $P(s_{j1}, s_{j2}, \dots, s_{jm})$ - prawdopodobieństwo znalezienia się źródła w stanie $s_{j1}, s_{j2}, \dots, s_{jm}$

Źródło bezpamięciowe o prawdopodobieństwach wytwarzania wiadomości elementarnych określonych rozkładem stacjonarnym źródła Markowa nazywamy źródłem do niego stowarzyszonym. Entropia źródła Markowa jest nie większa od entropii stowarzyszonego do niego źródła. Prawdopodobieństwa warunkowe ograniczają zmienność generowanych wiadomości i tym samym pozwalają na lepsze ich przewidywanie. Entropia źródła z przykładu 13 wynosi 0,775 natomiast entropia odpowiadającego mu źródła stowarzyszonego jest równa 0,834.

3.4. Źródła złożone

Źródło złożone $S_m = \{s_1, s_2, \dots, s_r\}$ definiuje się przez podanie prawdopodobieństw wytwarzania wiadomości elementarnych w każdym z m jego bezpamięciowych źródeł składowych oraz macierzy $T_{m \times m}$

prawdopodobieństw przejścia między dowolnymi źródłami składowymi. Wytworzenie wiadomości elementarnej przez S_m można przedstawić jako działanie dwuetapowe. W pierwszym etapie znajdując źródło, które wytworzyło ostatnią wiadomość elementarną dokonuje się wyboru źródła składowego zgodnie z rozkładem prawdopodobieństwa z macierzy T , w drugim - generowana jest wiadomość elementarna zgodnie z rozkładem prawdopodobieństwa właściwym wynikowemu źródłu składowemu. Obserwowanie powstałego już ciągu wiadomości na ogół nie pozwala zatem na podanie źródła składowego generującego następną wiadomość elementarną.

Źródła złożone stosowane są do porównywania efektywności algorytmów kompresji ciągów binarnych. Podanie maksymalnego, możliwego do osiągnięcia współczynnika kompresji wymaga znajomości entropii źródła generującego wektor. W celu oszacowania wielkości entropii źródła złożonego, w dalszej części rozdziału rozważane są dwa źródła pomocnicze S_{gn} i S_{dn} . Źródło S_{gn} jest zdefiniowane w ten sposób, że jego entropia jest co najmniej równa entropii odpowiadającego mu źródła złożonego. W wypadku źródeł złożonych o specyficznych własnościach otrzymane wzory na entropię są zgodne ze wzorami podającymi entropię źródeł bezpamięciowych i Markowa. Źródło S_{dn} wytwarza taką samą wiadomość jak odpowiadające mu źródło S_m , lecz jednocześnie dostarcza dodatkową informację, niedostępną zwykle w przypadku źródła złożonego. Entropia S_{dn} jest zatem nie większa od entropii analizowanego źródła S_m .

W dalszej części rozdziału rozważane jest najprostsze źródło złożone $S_2 = \{0,1\}$ o źródłach składowych nazwanych A i B. Macierz $T_{2 \times 2}$ dla takiego źródła przyjmuje postać

$$T = \begin{bmatrix} t_{AA} & t_{AB} \\ t_{BA} & t_{BB} \end{bmatrix}$$

gdzie t_{XY} jest prawdopodobieństwem przejścia ze źródła składowego X do Y ; $X, Y \in \{A, B\}$.

Niech P_X oznacza prawdopodobieństwo generowania przez S_2 wiadomości elementarnej pochodzącej ze źródła składowego X , P_{Xv} - prawdopodobieństwo wytworzenia wiadomości elementarnej v przez źródło składowe X . Zgodnie z opisem działania źródła złożonego:

$$P_A = P_A t_{AA} + P_B t_{BA}$$

$$P_A + P_B = 1$$

zatem

$$P_A = \frac{t_{BA}}{t_{AB} + t_{BA}}; \quad P_B = \frac{t_{AB}}{t_{AB} + t_{BA}}$$

Niech c_m oznacza ciąg wiadomości elementarnych o długości równej m . $P(c_m)$ oznacza przybliżone (nieuwzględniające wiadomości wytworzonych przed c_m) prawdopodobieństwo wygenerowania c_m przez S_2 , $P(c_m|X)$ zaś oznacza przybliżone prawdopodobieństwo wytworzenia c_m przez S_2 pod warunkiem, że ostatnia wiadomość elementarna c_m powstała w źródle składowym X . Może ona pochodzić ze źródła A lub B , zatem:

$$P(c_m) = P(c_m|A) + P(c_m|B) \quad (1)$$

Rekurencyjny sposób obliczania $P(c_m)$ określają równania:

$$\begin{aligned} P(c_m \cdot v|A) &= P(c_m|A) t_{AA} P_{Av} + P(c_m|B) t_{BA} P_{Av} \\ P(c_m \cdot v|B) &= P(c_m|A) t_{AB} P_{Bv} + P(c_m|B) t_{BB} P_{Bv} \end{aligned} \quad (2)$$

gdzie: $\cdot$ oznacza konkatenację ciągów wiadomości $v \in \{0, 1\}$

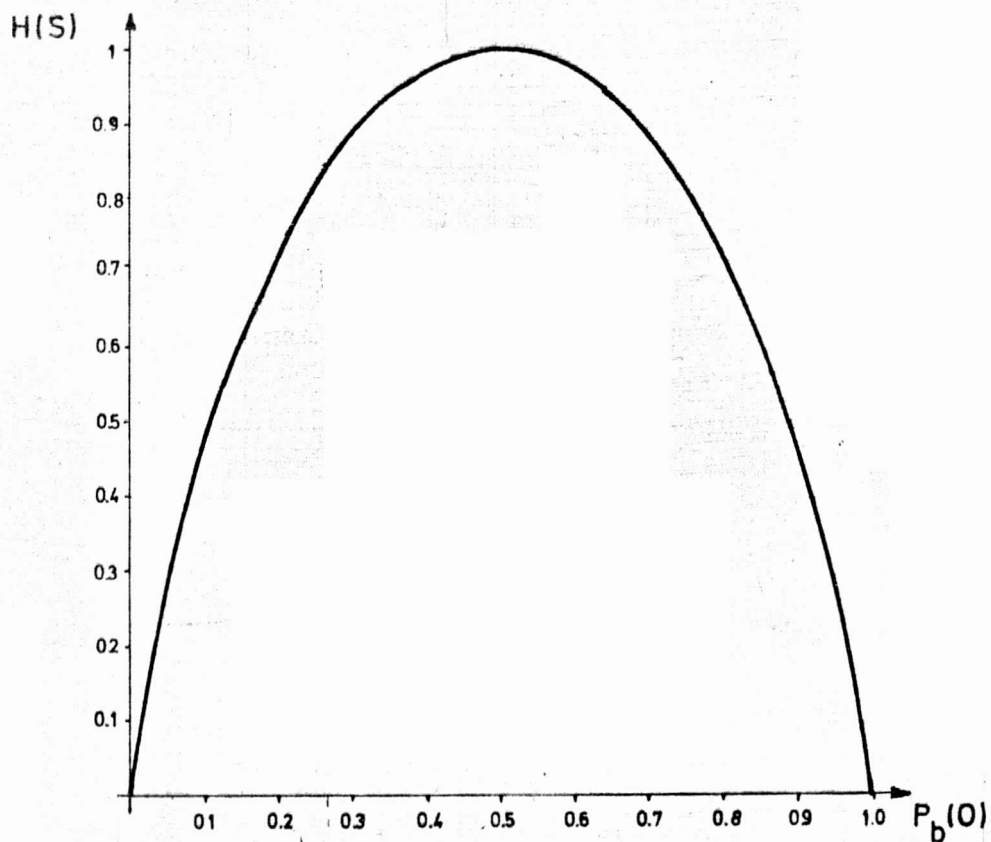
Dla ciągu składającego się z pojedynczej wiadomości elementarnej definiuje się:

$$P(v|A) = P_A P_{Av} \quad \text{oraz} \quad P(v|B) = P_B P_{Bv} \quad (3)$$

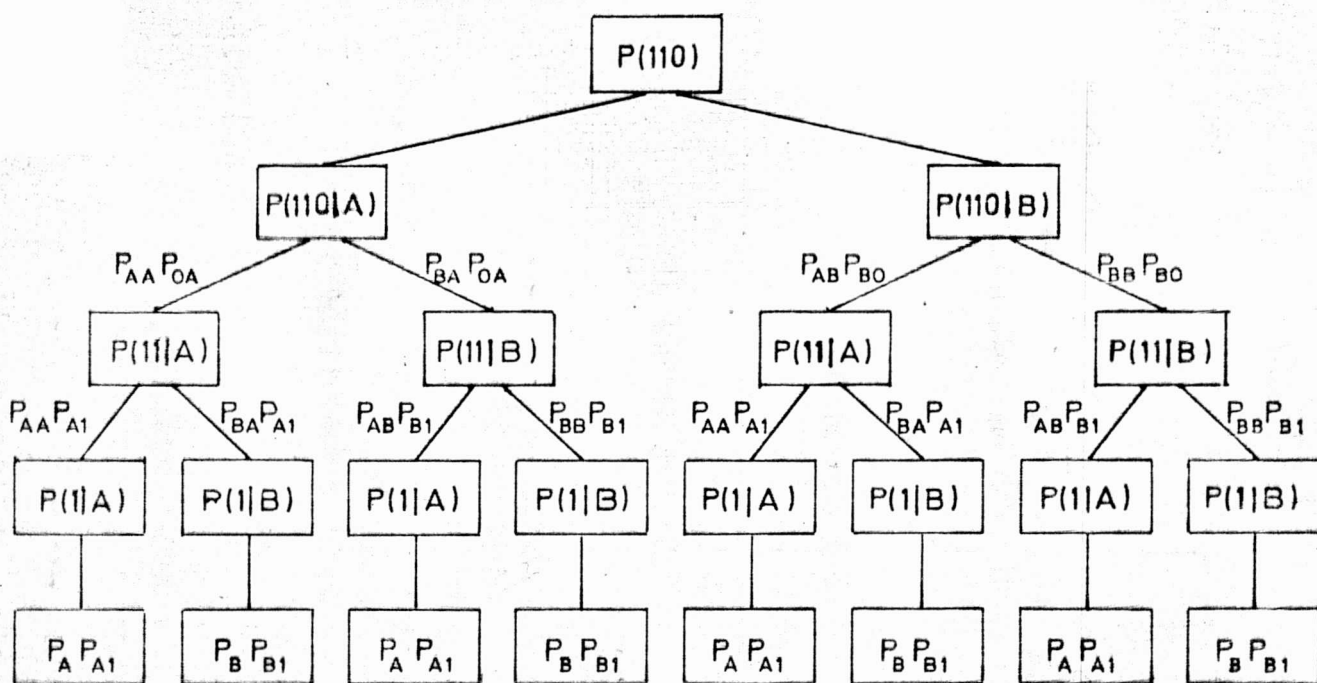
Graficzną interpretację wzorów (2) przedstawiono na rys. 5.

Przykład 14

$$P(01) = P(01|A) + P(01|B) = P(0|A) t_{AA} P_{A1} + P(0|B) t_{BA} P_{A1} +$$

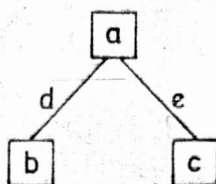


Rys. 4- Entropia bezpamięciowego, binarnego źródła $S=\{0,1\}$ jako funkcja $P_b(0)$.



Rys. 5. Graficzna interpretacja wzorów 1-3.

Uwaga: Zapis



oznacza $a = bd + ec$

$$P(O|B)t_{BB}^P P_{B1} + P(O|A)t_{AB}^P P_{B1} = P_A^P A O^t A A^P A_1 + P_B^P B O^t B A^P A_1 + \\ + P_B^P B O^t B B^P B_1 + P_A^P A O^t A B^P B_1$$

Wykorzystując wzory (1-3), można wykazać, że

$$P(c_m \cdot 1) + P(c_m \cdot 0) = P(c_m)$$

oraz

$$\sum_{c_m} P(c_m) = 1$$

Bez pamięciowe źródło informacji $S_{gm} = \{s_1, s_2, \dots, s_{2^m}\}$ o prawdopodobieństwach wytworzenia wiadomości elementarnych spełniających warunek:

$$P_b(s_{c_m}) = P(c_m)$$

nazwane jest górnym przybliżeniem m -tego rzędu źródła S_2 .

Analiza ciągu wiadomości źródła S_2 może dostarczyć więcej informacji o jego własnościach niż jest to zawarte w ciągu prawdopodobieństw $P(c_m)$. Wartość $H(S_{gm})/m$ ogranicza zatem z góry średnią niepewność obserwatora co do następnej wiadomości elementarnej generowanej przez źródło złożone.

Niech S_{zb} oznacza źródło złożone, dla którego $P_{A1} = P_{B1}$. Dla takiego źródła:

$$P(c_m \cdot v) = P(c_m \cdot v|A) + P(c_m \cdot v|B) = P(c_m|A)t_{AA}^P P_{Av} + \\ + P(c_m|B)t_{BA}^P P_{Av} + P(c_m|B)t_{BB}^P P_{Av} + P(c_m|A)t_{AB}^P P_{Av} = \\ = P(c_m|A)P_{Av} + P(c_m|B)P_{Av} = P(c_m)P_{Av}$$

oraz

$$P(v) = P_{Av}$$

Stosując m -krotnie powyższe równania dla $c_m = v_1 v_2 \dots v_m$, otrzymujemy:

$$P(v_1 v_2 \dots v_m) = P(v_1 v_2 \dots v_{m-1})P(v_m) = P(v_1)P(v_2) \dots P(v_m)$$

Zatem statystyczne własności ciągu wiadomości wygenerowanego przez

S_{zb} są takie same, jak ciągu pochodzącego z binarnego źródła bezpamięciowego, w którym $P_b(1) = P_{A1}$.

Wartości entropii górnego przybliżenia źródła S_{zb} spełniają warunek:

$$H(S_{g_i})/i = H_B$$

gdzie

$$i \in N$$

$$H_B = -P(0)\log P(0) - P(1)\log P(1)$$

Dowód

$$\begin{aligned} H(S_{gm+1}) &= - \sum_{c_{m+1}} P(c_{m+1}) \log P(c_{m+1}) = - \sum_{c_m} P(c_m) P(0) \log (P(c_m) P(0)) - \\ &- \sum_{c_m} P(c_m) P(1) \log (P(c_m) P(1)) = -P(0) \log P(0) \sum_{c_m} P(c_m) - \\ &- P(1) \log P(1) \sum_{c_m} P(c_m) = -P(0) \log P(0) - P(1) \log P(1) = H_B + H(S_{gm}) \end{aligned}$$

Stosując m krotnie powyższe równanie, otrzymujemy:

$$\frac{H(S_{gm})}{m} = \frac{H(S_{gm-1}) + H_B}{m} = \frac{m H_B}{m} = H_B \quad \text{c.b.d.o.}$$

Niech S_{zm} oznacza źródło złożone, dla którego $P_{A1} = 0$ i $P_{B1} = 1$. Dla takiego źródła:

$$\begin{aligned} P(c_m, 1) &= P(c_m, 1|A) + P(c_m, 1|B) = P(c_m, 1|A)t_{AB} + \\ &+ P(c_m, 1|B)t_{BB} = P(c_m, 1|B)t_{BB} \end{aligned}$$

oraz

$$P(c_m, 1) = P(c_m, 1|A) + P(c_m, 1|B) = P(c_m, 1|B),$$

zatem

$$P(c_m \cdot 11) = P(c_m \cdot 1) t_{BB}$$

W analogiczny sposób można wykazać prawdziwość podanych niżej równości:

$$P(c_m \cdot 00) = P(c_m \cdot 0) t_{AA}; \quad P(c_m \cdot 01) = P(c_m \cdot 0) t_{AB}$$

$$P(c_m \cdot 10) = P(c_m \cdot 1) t_{BA}$$

Stosując 6-krotnie uzyskane zależności, otrzymujemy w przypadku źródła S_{zm} :

$$P(0010110) = P(001011) t_{BA} = \dots = P(0) t_{AA} t_{AB} t_{BA} t_{AB} t_{BB} t_{BA}$$

Dla źródła S_{zm} :

$$P(0) = P_{AO} P_A + P_{OB} P_B = \frac{t_{BA}}{t_{BA} + t_{AB}};$$

$$P(1) = \frac{t_{AB}}{t_{BA} + t_{AB}}$$

oraz

$$\sum_{c_1} P(c_1 \cdot 0) = P(00) + P(10) = \frac{t_{AA} t_{BA} + t_{BA} t_{AB}}{t_{BA} + t_{AB}} = P(0)$$

Analogicznie:

$$\sum_{c_1} P(c_1 \cdot 1) = P(1)$$

Zatem dla $m = 1$ zachodzi $P(c_m \cdot 0) = P(0)$ oraz $P(c_m \cdot 1) = P(1)$.

Założmy, że równości te zachodzą dla dowolnego $k \in \mathbb{N}$. Wówczas:

$$\begin{aligned} \sum_{c_{k+1}} P(c_{k+1} \cdot 0) &= \sum_{c_k} P(c_k \cdot 00) + \sum_{c_k} P(c_k \cdot 10) = \\ &= t_{AA} \sum_{c_k} P(c_k \cdot 0) + t_{BA} \sum_{c_k} P(c_k \cdot 1) = \\ &= t_{AA} P(0) + t_{BA} P(1) = P(0) \end{aligned}$$

oraz analogicznie:

$$\sum_{c_{k+1}} P(c_{k+1} \cdot 1) = P(1)$$

Tym samym zgodnie z zasadą indukcji matematycznej dla dowolnej liczby naturalnej m zachodzi:

$$\sum_{c_m} P(c_m \cdot 0) = P(0) \quad \text{oraz} \quad \sum_{c_m} P(c_m \cdot 1) = P(1)$$

Wartości entropii górnego przybliżenia źródła S_{zm} spełniają warunek:

$$\lim_{m \rightarrow \infty} \frac{H(S_{gm})}{m} = H_m$$

gdzie $m \in \mathbb{N}$

$$H_m = -P(0)(t_{AA} \text{ lb } t_{AA} - t_{AB} \text{ lb } t_{AB}) - P(1)(t_{BA} \text{ lb } t_{BA} + \\ + t_{BB} \text{ lb } t_{BB})$$

Dowód

$$\begin{aligned} H(S_{gm+2}) &= - \sum_{c_{m+2}} P(c_{m+2}) \text{ lb } P(c_{m+2}) = - \sum_{c_m} P(c_m \cdot 00) \text{ lb } P(c_m \cdot 00) - \\ &- \sum_{c_m} P(c_m \cdot 01) \text{ lb } P(c_m \cdot 01) - \sum_{c_m} P(c_m \cdot 10) \text{ lb } P(c_m \cdot 10) - \\ &- \sum_{c_m} P(c_m \cdot 11) \text{ lb } P(c_m \cdot 11) = - \sum_{c_m} P(c_m \cdot 0) t_{AA} (\text{lb } P(c_m \cdot 0) + \\ &+ \text{lb } t_{AA}) - \sum_{c_m} P(c_m \cdot 0) t_{AB} (\text{lb } P(c_m \cdot 0) + \text{lb } t_{AB}) - \\ &- \sum_{c_m} P(c_m \cdot 1) t_{BA} (\text{lb } P(c_m \cdot 1) + \text{lb } t_{BA}) - \\ &- \sum_{c_m} P(c_m \cdot 1) t_{BB} (\text{lb } P(c_m \cdot 1) + \text{lb } t_{BB}) = \\ &- (t_{AA} \text{ lb } t_{AA} + t_{AB} \text{ lb } t_{AB}) \sum_{c_m} P(c_m \cdot 0) - \end{aligned}$$

$$\begin{aligned}
 & - (t_{BA} \lg t_{BA} + t_{BB} \lg t_{BB}) \sum_{c_m} P(c_m \cdot 1) - \\
 & - \sum_{c_m} P(c_m \cdot 0) \lg P(c_m \cdot 0) - \sum_{c_m} P(c_m \cdot 1) \lg P(c_m \cdot 1) = \\
 & = H_m + H(S_{gm+1})
 \end{aligned}$$

Ponieważ

$$H(S_{g1}) = H_B,$$

zatem

$$\lim_{m \rightarrow \infty} \frac{H(S_{gm})}{m} = \lim_{m \rightarrow \infty} \frac{(m-1)H_m + H_B}{m} = H_m \quad \text{c.b.d.o.}$$

Jak widać S_{zm} zachowuje się tak jak binarne źródło Markowa 1-go rzędu o poniższych prawdopodobieństwach warunkowych wytwarzania wiadomości elementarnych:

$$P(0|0) = t_{AA}; \quad P(0|1) = t_{BA}; \quad P(1|0) = t_{AB}; \quad P(1|1) = t_{BB},$$

przy założeniu, że znamy zawsze jedynie m ostatnich wygenerowanych przez nie wiadomości elementarnych.

Rozważmy źródło informacji $S_2(n)$ generujące wiadomości elementarne zgodnie z zasadami podanymi dla źródła złożonego S_2 , lecz dodatkowo wskazujące źródło składowe, które wytworzyło $n + 1$ -szą, licząc wstecz, wiadomość elementarną. Niech $a(c_n)$ oznacza prawdopodobieństwo wytworzenia przez $S_2(n)$ ciągu c_n pod warunkiem, że bit poprzedzający bezpośrednio ciąg c_n powstał w źródle składowym A. Zgodnie z opisem $S_2(n)$:

$$a(c_m) = a(c_m|A) + a(c_m|B)$$

$$a(c_m \cdot v|A) = a(c_m|A)t_{AA}^P{}_{Av} + a(c_m|B)t_{BA}^P{}_{Av}$$

$$a(c_m \cdot v|B) = a(c_m|A)t_{AB}^P{}_{Bv} + a(c_m|B)t_{BB}^P{}_{Bv}$$

dla $m = 1, 2, \dots, n$

oraz

$$a(v) = t_{AA} P_{Av} + t_{AB} P_{Bv}$$

gdzie $a(c_m|X)$ - prawdopodobieństwo wytworzenia ciągu c_m przez $S_2(c)$ pod warunkiem, że ostatni element c_m powstał w źródle składowym X oraz, że element poprzedzający bezpośrednio ciąg c_n wygenerowało źródło A .

Jeżeli wiadomość elementarna poprzedzająca c_n powstała w B , wówczas odpowiednie prawdopodobieństwa oznaczane są przez $b(c_m)$, $b(c_m|A)$, $b(c_m|B)$.

Źródło informacji $S_{dn} = \{s_1, s_2, \dots, s_{2^n}\}$ nazwiemy dolnym przybliżeniem n -tego rzędu źródła S_2 , jeżeli prawdopodobieństwo wytworzenia wiadomości elementarnej s_{c_n} wynosi $a(c_n)$, gdy wiadomość elementarna poprzedzająca bezpośrednio ciąg c_n powstała w A oraz $b(c_n)$ - w przypadku przeciwnym.

Entropia tak określonego źródła S_{dn} jest równa:

$$H(S_{dn}) = -P_A \sum_{c_n} a(c_n) \lg a(c_n) - P_B \sum_{c_n} b(c_n) \lg b(c_n)$$

Wartość $\frac{H(S_{dn})}{n}$ można interpretować jako średnią miarę niepewności obserwatora co do następnej wiadomości elementarnej generowanej przez S_{gm} .

Przy określaniu źródła S_{gm} zwykle nie wykorzystujemy wszystkich zależności statystycznych występujących w ciągu wiadomości S_2 w przypadku źródła S_{dm} wykorzystujemy natomiast informację, która nie jest zwykle dostępna na podstawie analizy ciągu wiadomości S_2 . Entropia źródła złożonego S_2 jest zatem ograniczona z góry ciągiem wartości $\frac{H(S_{gm})}{m}$, z dołu zaś ciągiem $\frac{H(S_{dm})}{m}$. Program obliczający $H(S_{gm})/m$ i $H(S_{dm})/m$ wg wyprowadzonych zależności zamieszczony jest w zał. 1. Wartości przyjmowane przez te ciągi dla $m = 1, 2, \dots, 7$ dla źródeł złożonych wykorzystywanych w pracy do porównywania sprawności algorytmów kompresji ciągów

binarnych są zamieszczone na rys. 6.

3.5. Entropia źródła a kodowanie jego wiadomości

Kodem nazywamy odwzorowanie zbioru wszystkich możliwych ciągów wiadomości elementarnych źródła $S = \{s_1, s_2, \dots, s_q\}$ na pewien zbiór ciągów utworzonych z elementów zbioru $X = \{X_1, X_2, \dots, X_r\}$ będącego zbiorem sygnałów elementarnych. Pojedynczy sygnał pochodzący z dwuelementowego zbioru X nazywany jest binitem. Najczęściej stosowane są kody blokowe, tj. takie, w których każdą z wiadomości elementarnych S odwzorowuje się na ustalony ciąg sygnałów elementarnych X nazwany ciągiem (słowem) kodowym. Niech X_i oznacza ciąg kodowy przypisany wiadomości s_i .

Kod blokowy jest nieosobliwy, jeżeli wszystkie ciągi kodowe X_i różnią się nawzajem.

Taki kod blokowy, który ciągom wiadomości elementarnych $s_{i1}, s_{i2}, \dots, s_{in}$ przyporządkowuje ciągi słów kodowych $X_{i1}, X_{i2}, \dots, X_{in}$ nazywamy n -krotnym jego rozszerzeniem.

Często wymaga się, by stosowane kody umożliwiały poprawne (wierne) odtworzenie zakodowanych wiadomości. Kody spełniające ten warunek nazywamy jednoznacznie dekodowalnymi.

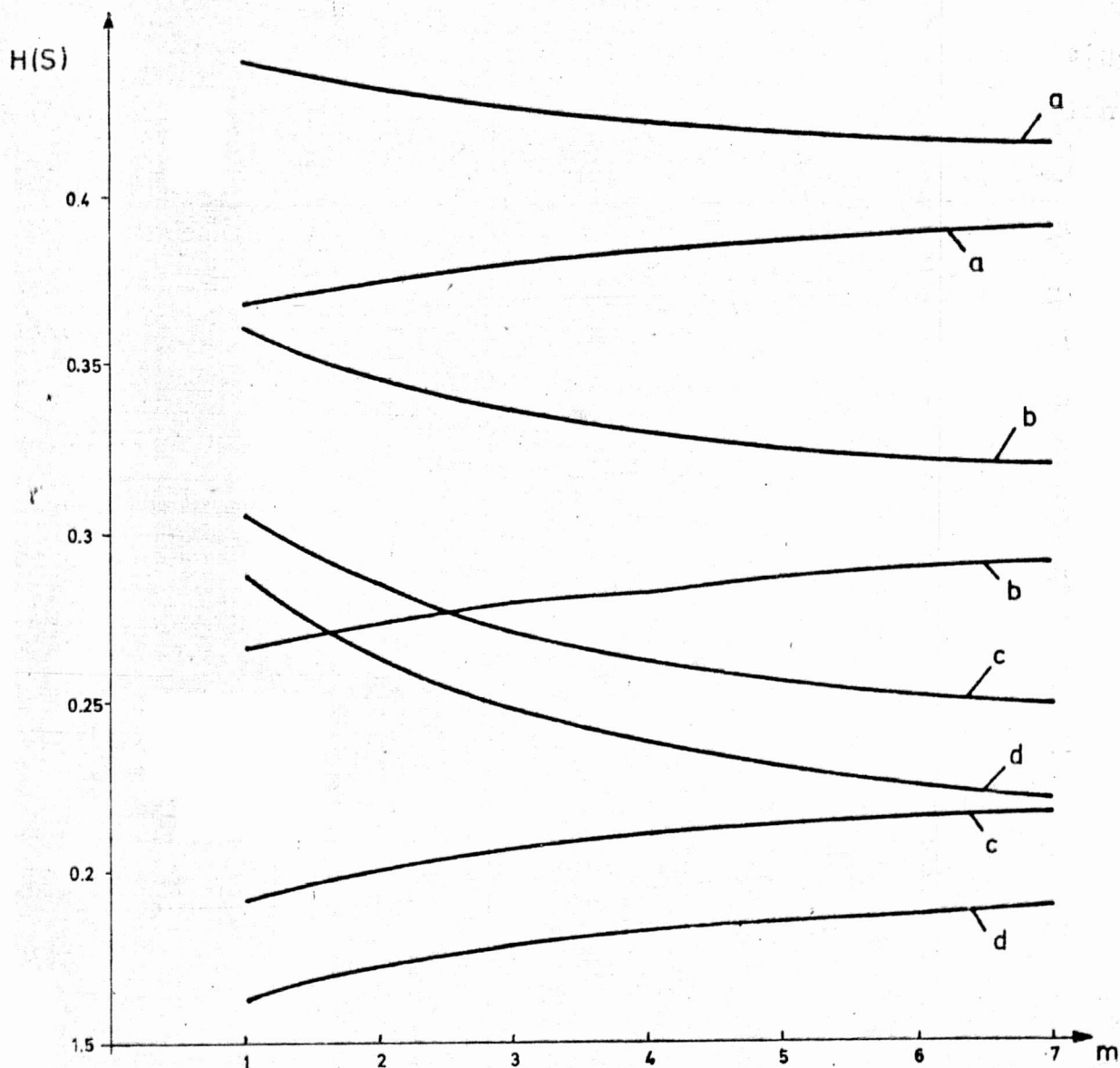
Kod blokowy jest jednoznacznie dekodowalny, jeżeli dla dowolnego skończonego n jego n -krotne rozszerzenie jest kodem nieosobliwym.

Średnią długość ciągu kodowego definiuje równanie:

$$L = \sum_{i=1}^q p_i l_i$$

gdzie: p_i - prawdopodobieństwo wytworzenia wiadomości s_i przez źródło S

l_i - liczba sygnałów elementarnych w X_i



Rys. 6. Szacowanie wartości entropii źródeł złożonych – wartości

$$\frac{H(Sd_m)}{m} \text{ i } \frac{H(Sg_m)}{m}$$

$$a - t_{AA} = 0.9; \quad t_{BB} = 0.99; \quad P_{A0} = 0.5; \quad P_{B0} = 0.95$$

$$b - t_{AA} = 0.9; \quad t_{BB} = 0.99; \quad P_{A0} = 0.5; \quad P_{B0} = 0.975$$

$$c - t_{AA} = 0.9; \quad t_{BB} = 0.99; \quad P_{A0} = 0.5; \quad P_{B0} = 0.99$$

$$d - t_{AA} = 0.9; \quad t_{BB} = 0.99; \quad P_{A0} = 0.5; \quad P_{B0} = 0.995$$

Jeżeli kod jest jednoznacznie dekodowalny, wówczas:

$$H_r(S) \leq L$$

gdzie $H_r(S)$ - to entropia źródła S mierzona w jednostkach r -narnych.

Sprawność i redundancję kodu definiują równania:

$$S_k = \frac{H_r(S)}{L} \quad \text{oraz} \quad R = 1 - S_k$$

Redundancję kodu można zmniejszyć, zastępując kodowanie pojedynczych wiadomości elementarnych kodowaniem za pomocą jednego ciągu kodowego ciągów wiadomości elementarnych. Pozwala to na zwiększenie współczynnika kompresji, lecz jednocześnie powoduje znaczny wzrost rozmiarów tablic stosowanych do kodowania i dekodowania. Wady tej nie posiadają kody arytmetyczne należące do grupy kodów nieblokowych [48]. Pozwalają one na osiągnięcie średniej długości ciągu kodowego o wartości nieznacznie się różniącej od entropii kodowanego źródła już przy kodowaniu pojedynczych wiadomości elementarnych. W kodach tych stosuje się słowa kodowe o długościach będących liczbami rzeczywistymi. Łączenie ciągów kodowych realizuje się za pomocą operacji arytmetycznych, dekodowanie zaś umożliwia relacja mniejszości wartości ciągów kodowych. Mimo swych zalet kody arytmetyczne nie znajdują jeszcze w praktyce szerszego zastosowania głównie z powodu złożoności realizującego je oprogramowania.

4. OPIS PROPONOWANEGO ALGORYTMU KOMPRESJI WEKTORÓW BINARNYCH

Niniejszy rozdział zawiera opis i analizę sposobu działania algorytmu A - proponowanego algorytmu kompresji. Algorytm ten winien się charakteryzować dwoma cechami: osiąganiem wartości współczynnika kompresji wektorów binarnych stosowanych w SWI, zbliżonych do wartości uzyskiwanych przez inne algorytmy oraz dużą ich stabilnością pozwalającą na otrzymanie takiego skrócenia długości zapisu bez potrzeby ustalania statystycznych własności przetwarzanych danych. Obu tych cech, korzystnych z punktu widzenia praktycznego wykorzystania algorytmu, nie posiadają dotychczas opracowane algorytmy.

Przeprowadzona w tym rozdziale analiza pozwoliła na oszacowanie wartości współczynnika kompresji osiąganego przez ten algorytm przy kodowaniu wektorów binarnych pochodzących ze źródeł bezpamięciowych i złożonych. Uzyskane w ten sposób wyniki porównano następnie z wartościami współczynnika kompresji wektorów testowych wygenerowanych przez wyżej wymienione źródła. W przeprowadzonym eksperymencie do kodowania tych wektorów wykorzystano zarówno różne wersje algorytmu A, jak i inne algorytmy znane z literatury. Szczegółowy opis eksperymentu wraz z analizą uzyskanych wyników jest zamieszczony w rozdziale 5.

Przy opisie i analizie działania algorytmu A wykorzystano następujące oznaczenia:

- X - wektor wejściowy
- X_i - i -ta grupa wektora X
- Y - wektor wyjściowy
- Y_i - i -ty bit wektora Y

Z - wektor pomocniczy

Z_i - i -ta grupa wektora Z

$\overline{G}$ - długość wektora G

$\tilde{G}$ - długość kodu wektora G

g_1 - częstość występowania 1 w G

g_0 - częstość występowania 0 w G

$h(G)$ - przybliżona ilość informacji przekazywanej przez G

$$h(G) = \overline{G}(-g_0 \lg g_0 - g_1 \lg g_1)$$

$t(G)$ - rzeczywista ilość informacji przekazywanej przez G

Uwaga: G oznacza dowolny z wektorów X , Y , Z .

4.1. Opis działania algorytmu A

4.1.1. Kodowanie

Algorytm A przetwarza wejściowy wektor X pobierając z niego kolejne ciągi zwane grupami. Oznaczana przez m liczba elementów tworzących grupę jest jedynym parametrem działania algorytmu. Jeżeli zachodzi potrzeba, to ostatnia grupa powstaje po uzupełnieniu X brakującymi zerami. W trakcie kompresji powstają dwa wektory: wyjściowy Y i pomocniczy Z . Wektor Y tworzą wszystkie jedynki z X oraz część jego zer, natomiast do Z dopisywane są wszystkie niezerowe (nie składające się z samych zer) grupy z X po ich pełnym rozpisaniu. Czynności wykonywane przy rozpisaniu grupy a na grupę b obejmują:

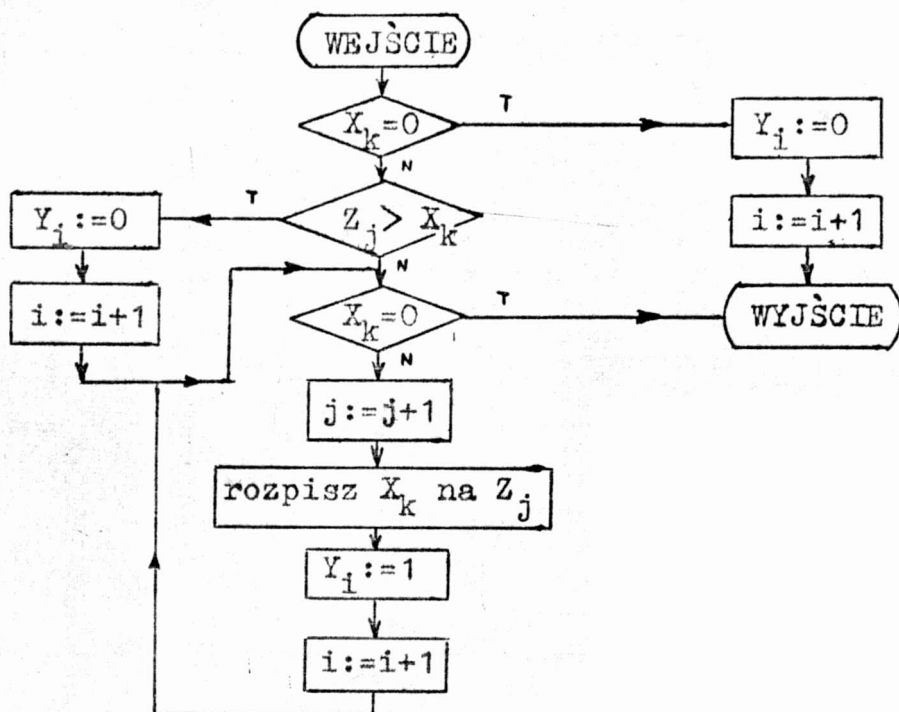
- wyzerowanie b ,
- nadanie wartości 1 elementowi b o numerze $l(a)$,
- wyzerowanie elementu o numerze $l(a)$ w a

gdzie $l(a)$ jest funkcją określoną na niezerowych grupach, której wartością jest numer kolejny (liczony od lewej strony, począwszy od zera) skrajnie lewego elementu w grupie a mającego wartość 1,

np. $1(0100101) = 1$, $1(0001) = 3$.

Jeżeli wyżej wymienione czynności będą powtarzane tak długo, aż ciąg a będzie się składał z samych zer, to powiemy, że grupa a została w pełni rozpisana; np. przy pełnym rozpisaniu grupy 010110 powstaną ciągi: 010000, 000100 oraz 000010.

Niech x oraz z oznaczają odpowiednio aktualnie przetwarzaną grupę X i ostatnio zapisaną grupę Z . Pobrana podczas kodowania zerowa grupa z X powoduje jedynie dopisanie 0 do wektora Y . W wypadku pobrania grupy niezerowej porównuje się jej wartość z ostatnio zapisaną grupą Z . Jeżeli $z > x$, to do Y dopisywane jest 0. Oprócz tego, każda niezerowa grupa X powoduje dopisanie wszystkich jej jedynek do Y , a także dopisanie do Z wszystkich grup powstałych z pełnego rozpisania x . Sposób postępowania w trakcie kodowania grupy z X jest zamieszczony na rys. 7.



Rys. 7. Algorytm A - kodowanie grupy z wektora X

Przed rozpoczęciem kompresji wektor Y jest pusty ($\bar{Y} = 0$), zaś Z zawiera grupę składającą się z samych zer, z wyjątkiem

elementu na $m-1$ pozycji. Algorytm A przetwarza X , pobierając z niego kolejne grupy i kodując je w wyżej wymieniony sposób. Po zakończeniu kompresji wektor Z tworzy $(\overline{X} x_1 + 1)$ m elementów. Pierwsza jego grupa ma zawsze postać $0 \dots 01$. Grupa ta nie jest wymagana przy dekodowaniu i dlatego też nie jest zapamiętana w skompresowanej postaci X .

Program w Pascalu realizujący kompresję jest zamieszczony w zał. 2

Przykład 15

Dla $m = 3$, $X = 000\ 110\ 000\ 000\ 000\ 000\ 000\ 001\ 000\ 000$

powstałe w wyniku kompresji wektory Y i Z tworzone są w następujący sposób (podkreślono przetwarzane fragmenty X i utworzone fragmenty Y i Z):

$X = \underline{000}\ 110\ 000\ 000\ 000\ 000\ 000\ 001\ 000\ 000$

$Y = \underline{0}$

$Z = 001$

001 jest początkową zawartością Z

$X = 000\ \underline{110}\ 000\ 000\ 000\ 000\ 000\ 001\ 000\ 000$

$y = \underline{011}$

$Z = 001\ \underline{100\ 010}$

001 $<$ 100 zatem do y dopisano jedynie 11

$X = 000\ 110\ \underline{000\ 000\ 000\ 000\ 000}\ 001\ 000\ 000$

$Y = 011\ \underline{00000}$

kodowanie 5 grup zerowych X powoduje jedynie dopisanie do Y 00000

$Z = 001\ 100\ 010$

$X = 000\ 110\ 000\ 000\ 000\ 000\ 000\ \underline{001}\ 000\ 000$

$Y = 01100000\ \underline{01}$

$Z = 001100\ 010\ \underline{001}$

010 $>$ 001 zatem dopisanie 1 do Y poprzedzone zostało umieszczeniem tam 0.

$X = 000\ 110\ 000\ 000\ 000\ 000\ 000\ 001\ \underline{000\ 000}$

$Y = 0110000001\underline{00}$

$Z = 001\ 100\ 010\ 001$

4.1.2. Dekodowanie

Przy dekodowaniu wykorzystywany jest m bitowy wskaźnik R. Służy on do odtwarzania niezerowych grup z wektora X. Podczas tego procesu wektory Y i Z są przetwarzane sekwencyjnie, począwszy od strony lewej do prawej. Niech i oraz j oznaczają indeksy odpowiednio pierwszego nie przetworzonego elementu Y i pierwszej nie przetworzonej grupy Z. Dekodowanie grupy X rozpoczyna się od badania wartości Y_i . Jeżeli $Y_i = 0$, to oznacza, że poszukiwana grupa X składała się z samych zer. W przeciwnym przypadku nadaje się R wartość Z_j , zwiększa się wartości i oraz j o 1, a następnie porównywana jest wartość Z_{j-1} oraz Z_j . Jeżeli $Z_{j-1} > Z_j$ oraz $Y_i = 1$, to sygnalizuje, że te dwie grupy powstały z rozpisania tej samej grupy X. Należy zatem dodać Z_j do R, zwiększyć wartości obu indeksów i powtórzyć badanie. Czynności te powtarza się tak długo (maksymalnie m razy) dopóki nie zostanie spełniony któryś z wyżej wymienionych warunków. Zachodzenie nierówności $Z_{j-1} \leq Z_j$ wskazuje na pochodzenie Z_{j-1} i Z_j z różnych grup X. Jeżeli $Y_i = 0$, to oznacza, że Z_{j-1} i Z_j powstały z różnych grup X, mimo że spełnienie nierówności $Z_{j-1} > Z_j$ pozwala na ich zsumowanie. Sposób postępowania przy dekodowaniu grupy X jest przedstawiony na rys. 8.

Przed rozpoczęciem dekodowania wektor X jest pusty, indeksy i oraz j wskazują na początki odpowiednich wektorów. Do wektora Z dopisana jest grupa składająca się z samych zer z wyjątkiem skrajnie lewej pozycji. Pozwala to na odtworzenie ostatniej grupy X, o ile była ona niezerowa. Dekodowanie kończy się po przetworze-

niu ostatniego bitu Y.

Program w Pascalu realizujący dekodowanie jest zamieszczony w zał. 2.

Przykład 16

Dla $m = 2$, $Z = 10100111$, $Y = 01000110000$

dekodowanie wektora X przebiega w następujący sposób:

$Z = 10100110$

$Y = \underline{0}1000110000$

$X = \underline{00}$

grupa została dopisana do Z przed rozpoczęciem dekodowania

$Z = \underline{10}100110$

$Y = 0\underline{1}000110000$

$X = 00\underline{10}$

grupy 10 i 10 z wektora Z nie mogły powstać z rozpisania tej samej grupy X, do tego wektora dopisano zatem jedynie 10

$Z = 10100110$

$Y = 01\underline{000}110000$

$X = 001\underline{0000000}$

dopisanie 3 grup zerowych do X

$Z = 10\underline{100}110$

$Y = 01000\underline{1}10000$

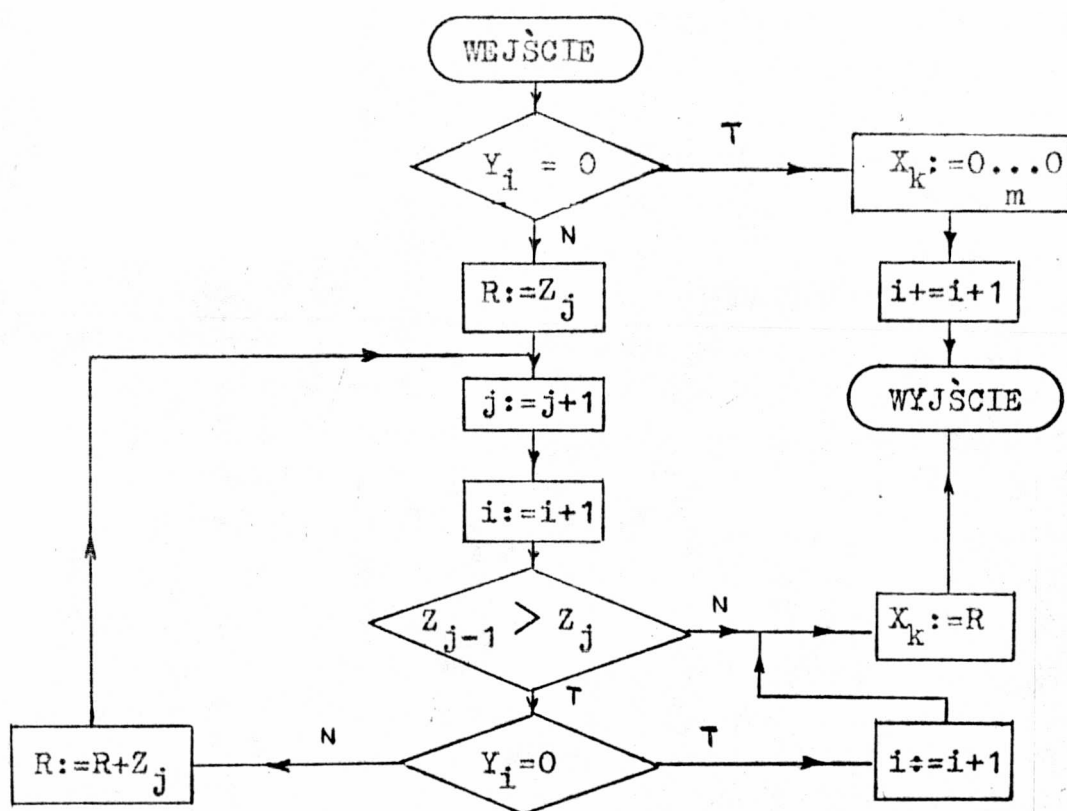
$X = 0010000000\underline{11}$

$10 > 01$ oraz $Y_i = 1$, zatem odtworzona w tym kroku grupa X powstała po zsumowaniu dwu kolejnych grup Z

$Z = 10100110$

$Y = 010001\underline{10000}$

$X = 0010000000\underline{1100000000}$



Rys. 8. Algorytm A - dekodowanie grupy z wektora X

4.1.3. Przekształcanie wektora Z

Wektor Z utworzony przez algorytm kompresji A cechuje znaczna redundancja zapisu. Kolejne jego m bitowe fragmenty zawierają dokładnie po jednej 1. W celu osiągnięcia współczynników kompresji o wartościach zbliżonych do maksymalnych konieczne jest zatem kodowanie wektora Z. Algorytm A stosuje liczbowe kodowanie kolejnych m bitowych podciągów Z. Podczas tego procesu wartości funkcji 1 dla następujących po sobie grup Z są traktowane jak cyfry z m-rnego systemu pozycyjnego. Liczby składające się z k kolejnych takich cyfr są następnie zapisywane w systemie binarnym. Oznaczana przez "a" wartość liczby reprezentującej grupy $z_i, z_{i+1}, \dots, z_{i+k-1}$ jest obliczana za pomocą poniższego równania:

$$a = \sum_{j=0}^{k-1} m^j \cdot 1(z_{i+j})$$

Sprawność kodowania Z jest uzależniona od wartości k. W proponowanym algorytmie ustala się wartość k tak, by maksymalizować funkcję $F(k,m)$ określoną następująco:

$$F(k,m) = \frac{1bm^k}{\lceil 1bm^k \rceil}$$

a jednocześnie tak, by możliwe było wykonywanie operacji arytmetycznych wymaganych przy kodowaniu i dekodowaniu za pomocą pojedynczych instrukcji języka wewnętrznego maszyny cyfrowej. Zachodzi to wówczas, gdy oznaczana przez b maksymalna wartość $\lceil 1bm^k \rceil$ jest mniejsza niż standardowa długość zapisu liczby całkowitej w maszynie cyfrowej. Wartości funkcji $F(k,m)$ dla $3 \leq m \leq 14$ przy $b = 23$ oraz $b = 31$ są podane poniżej. Z zamieszczonych tam danych wynika, że ten dość znaczny wzrost wartości b tylko w nielicznych przypadkach spowodował zwiększenie sprawności kodowania Z.

Przykład 17

Dla $m = 3, k = 5,$

$$F(5,3) = \frac{1b \cdot 243}{8} = 0.9906,$$

wektora Z mającego postać:

100 010 001 100 100 100 010 001 001 100 001 001 100 100 010

$$a_1 = 0 \cdot 3^0 + 1 \cdot 3^1 + 2 \cdot 3^2 + 0 \cdot 3^3 + 0 \cdot 3^4 = 21$$

$$a_2 = 75, \quad a_3 = 89$$

Zakodowany wektor Z:

$$00010 \ 101 \ 01001011 \ 01011001 \Big|_2 = 1395545 \Big|_{10}$$

Wartości funkcji $F(k,m)$

$m \backslash b$	23	31
3	0,9906	0,9979
4	1,0	1,0
5	0,9951	0,9951
6	0,9942	0,9942
7	0,9908	0,9962
8	1,0	1,0
9	0,9906	0,9906
10	0,9966	0,9966
11	0,9884	0,9884
12	0,9958	0,9958
13	0,9868	0,9963
14	0,9932	0,9932

4.2. Analiza działania algorytmu A

W ramach przeprowadzonej analizy badano wartości osiąganego przez algorytm A współczynnika kompresji. Współczynnik ten jest najważniejszym czynnikiem mającym wpływ na ocenę algorytmu. Zwiększenie jego wartości pozwala na podniesienie liczby dokumentów przechowywanych w SWI. Czas wymagany przez algorytm na kodowanie i dekodowanie jest (szczególnie w przypadku stosowania wersji A_n algorytmu) dłuższy niż dla prostych algorytmów, takich jak np. KKD. Czasy te mają jednak znacznie mniejszy wpływ na ocenę algorytmu. Szybkość reakcji systemu na pytanie użytkownika zależy bowiem w głównej mierze od liczby wykonywanych transmisji między pamięcią operacyjną a pomocniczą.

4.2.1. Ciągi pochodzące ze źródeł bezpamięciowych

Jedynym parametrem decydującym o własnościach wektora pochodzącego z bezpamięciowego źródła informacji jest prawdopodobieństwo wygenerowania przez nie 1. Dla dostatecznie długiego X wartość tę można przybliżyć przez x_1 . Rzeczywista ilość informacji dostarczanej przez taki wektor X wynosi:

$$t(X) = \bar{X} h(X),$$

a maksymalny możliwy do osiągnięcia współczynnik kompresji:

$$k_{\max} = \frac{1}{h(X)}.$$

Długość wektora Z po zakodowaniu X równa się:

$$\bar{Z} = m x_1 \bar{X}$$

Do wyznaczenia długości Y konieczne jest dokładne przeanalizowanie sposobu pracy algorytmu. W Y umieszczane wszystkie jedynki z X oraz część zapisanych tam zer. Wprowadzona poniżej funkcja e umożliwia oszacowanie liczby takich zer. Określona na dwu grupach a i b funkcja e , przyjmująca wartości będące liczbami naturalnymi z przedziału $\langle 1, m \rangle$, jest zdefiniowana następująco:

$$e(a, b) = \begin{cases} l(b) - l(a), & l(b) > l(a) \\ l(b) - l(a) + m, & l(b) \leq l(a) \end{cases}$$

Funkcja ta wyznacza podobieństwo grup, np.:

$$e(0010, 0001) = 1, e(0010, 0100) = 3, e(0010, 0010) = 4$$

Niech r_i i r_{i+1} oznaczają dwie sąsiednie grupy w Z . Podobieństwo ich wynosi k , jeżeli dopisanie grupy r_{i+1} nastąpiło w wyniku kodowania jednego z poniższych ciągów (w nawiasach podano prawdopodobieństwo pojawienia się odpowiedniego runu):

$$\begin{array}{ll} \begin{array}{c} 0 \dots 01 \\ K-1 \end{array} & (x_0^{K-1} x_1) \\ \begin{array}{c} 0 \dots 0 \\ K-1 \end{array} \quad \begin{array}{c} 0 \dots 01 \\ m \end{array} & (x_0^{K-1} x_0^m x_1) \\ \begin{array}{c} 0 \dots 0 \\ K-1 \end{array} \quad \begin{array}{c} 0 \dots 0 \dots 0 \dots 0 \\ m \quad m \end{array} & (x_0^{K-1} x_0 \dots x_0^m x_1) \end{array}$$

Łączne prawdopodobieństwo wystąpienia takiego zdarzenia podczas kodowania wynosi:

$$P_r(e(r_i, r_{i+1}) = K) = \sum_{i=0}^{\infty} x_0^{im} x_0^{K-1} x_1 = x_0^{K-1} x_1 \sum_{x=0}^{\infty} x_0^{im} =$$

$$= \frac{x_0^{K-1} x_1}{1 - x_0^m} \quad (4)$$

Jak widać podobieństwo kolejnych zawartości R tym trudniej jest przewidzieć, im bardziej x_0 zbliżony jest do 1.

Algorytm A nie przepisuje do Y żadnego zera w runów o długości nieprzekraczającej m. Z runów o długości c przepisanych do Y zostanie $\lfloor (c - 1)/m \rfloor$ zer oraz jedna jedynka. Jeżeli podobieństwo dwu kolejnych zawartości R wynosi d, to oznacza, że zakodowano run o długości równej $l = d - 1 + m$ i, gdzie $i = 0, 1, \dots$ zaś $d - 1$ jest liczbą tzw. bezwzględnie pomijanych zer runu.

W dostatecznie długim ciągu X liczbę takich zer, oznaczaną przez O_u , można aproksymować równaniem:

$$O_u = \bar{X} x_1 \sum_{i=1}^m (i - 1) P_r(e(r_i, r_{i-1}) = i) = \bar{X} x_1 \sum_{i=1}^m (i-1) \frac{x_0^{K-1} x_1}{1 - x_0^m} =$$

$$= \bar{X} \frac{x_1^2}{1 - x_0^m} \sum_{i=1}^{m-1} i x_0^i$$

Ponieważ

$$\sum_{i=1}^{m-1} i x_0^i = \frac{x_0 - x_0^m (m x_1 + x_0)}{(1 - x_0)^2}$$

zatem

$$O_u = \frac{\bar{X} x_1^2}{1 - x_0^m} \frac{x_0 - x_0^m (m x_1 + x_0)}{(1 - x_0)^2} = \bar{X} \frac{x_0 - x_0^m (m x_1 + x_0)}{1 - x_0^m}$$

Oznaczana przez O_p liczba 0 przepisywanych do Y jest równa

$$O_p = \frac{\bar{X} x_0 - O_u}{m} = \bar{X} \frac{x_0 (1 - x_0^m) - x_0 + x_0 (m - m x_0 + x_0)}{m (1 - x_0^m)} =$$

$$= \bar{X} \frac{x_0^m x_1}{1 - x_0^m}$$

Długość ciągu Y wynosi zatem

$$\begin{aligned}\bar{Y} &= \bar{X} x_1 + 0_p = \frac{\bar{X}}{1 - x_0^m} (x_1(1 - x_0^m) + x_0^m x_1) = \\ &= \frac{\bar{X} x_1}{1 - x_0^m}\end{aligned}$$

natomiast częstość występowania 0 w Y równa się

$$y_0 = \frac{0_p}{\bar{Y}} = \frac{\bar{X} x_1 x_0^m}{1 - x_0^m} \cdot \frac{1 - x_0^m}{\bar{X} x_1} = x_0^m$$

Na podstawie powyższych zależności można obliczyć ilość informacji zawartej w Z i Y. Oba te ciągi pozwalają na jednoznaczne odtworzenie X, ilość ta nie może być więc mniejsza niż $\bar{X} h(x)$.

Maksymalna ilość informacji jaka może być zawarta w Y wynosi

$$\begin{aligned}h(Y) &= \bar{X} \frac{x_1}{1 - x_0^m} (-x_0^m \text{lb } x_0^m - (1 - x_0^m) \text{lb } (1 - x_0^m)) = \\ &= \bar{X} \frac{x_1}{1 - x_0^m} (x_0^m \text{lb } \frac{1 - x_0^m}{x_0^m} - \text{lb } (1 - x_0^m))\end{aligned}$$

Równanie (4) pozwala na obliczenie $p(Z)$ - szacunkowej ilości informacji przekazywanej przez Z:

$$\begin{aligned}p(Z) &= \bar{X} x_1 \sum_{j=1}^m -P_r(e(r_i, r_{i+1}) = j) \text{lb } (P_r(e(r_i, r_{i+1}) = j)) = \\ &= \bar{X} x_1 \sum_{j=1}^m \frac{x_1 x_0^{i-1}}{1 - x_0^m} \text{lb } \frac{1 - x_0^m}{x_1 x_0^{j-1}} = \bar{X} \frac{x_1^2}{1 - x_0^m} (\text{lb}(1 - x_0^m) \sum_{j=1}^m x_0^j - \\ &- \text{lb } x_1 \sum_{j=0}^{m-1} x_0^j - \text{lb } x_0 \sum_{j=1}^{m-1} j x_0^j) = \\ &= \bar{X} \frac{x_1^2}{1 - x_0^m} (\text{lb } \frac{(1 - x_0^m)}{x_1} \sum_{i=0}^{m-1} x_0^i - \text{lb } x_0 \sum_{i=1}^{m-1} i x_0^i) = \\ &= \bar{X} \frac{x_1^2}{1 - x_0^m} \left(\frac{1 - x_0^m}{x_1} \text{lb } \frac{(1 - x_0^m)}{x_1} - \frac{x_0 - x_0^m(mx_1 + x_0)}{x_1^2} \text{lb } x_0 \right) =\end{aligned}$$

$$= \bar{X} (x_1 \text{ lb } \frac{(1 - x_0^m)}{x_1} - \frac{x_0 - x_0^m (m x_1 + x_0)}{1 - x_0^m} \text{ lb } x_0)$$

Obecnie można już przystąpić do obliczenia szacunkowej ilości informacji zawartej w obu ciągach Z i Y.

$$\begin{aligned} p(Z) + h(Y) &= \frac{\bar{X}}{1 - x_0^m} \left[(1 - x_0^m) x_1 \text{ lb } \frac{(1 - x_0^m)}{x_1} - x_0 \text{ lb } x_0 + \right. \\ &+ x_0^m m x_1 \text{ lb } x_0 + x_0^{m+1} \text{ lb } x_0 + x_1 x_0^m \text{ lb } \frac{(1 - x_0^m)}{x_0^m} - x_1 \text{ lb } (1 - x_0^m) \left. \right] = \\ &= \frac{\bar{X}}{1 - x_0^m} \left[x_1 \text{ lb } (1 - x_0^m) - x_0^m x_1 \text{ lb } (1 - x_0^m) - x_1 \text{ lb } x_1 + \right. \\ &+ x_0^m x_1 \text{ lb } x_1 - x_0 \text{ lb } x_0 + x_0^m x_1 \text{ lb } x_0^m + x_0^{m+1} \text{ lb } x_0 + x_1 x_0^m \cdot \\ &\left. \text{lb} (1 - x_0^m) - x_1 x_0^m \text{ lb } x_0^m - x_1 \text{ lb } (1 - x_0^m) \right] = \\ &= \frac{\bar{X}}{1 - x_0^m} (-x_1 \text{ lb } x_1 + x_0^m x_1 \text{ lb } x_1 - x_0 \text{ lb } x_0 + x_0^m x_0 \text{ lb } x_0) = \\ &= \bar{X} (-x_1 \text{ lb } x_1 - x_0 \text{ lb } x_0) = t(x) \end{aligned}$$

Powyższa równość oznacza, że algorytm nie wprowadza w trakcie kompresji dodatkowej informacji do Z lub Y oraz, że Y może być traktowany jak ciąg pochodzący ze źródła bezpamięciowego ($t(Y) = h(Y)$). Nie oznacza to jednak, że algorytm A jest w stanie osiągnąć maksymalną dopuszczalną kompresję przy kodowaniu dowolnego bezpamięciowego X. Spowodowane jest to koniecznością kodowania Z oraz redundancją zapisu wektora Y.

Sprawność kodowania wektora Z jest obniżana przez dwa czynniki. Pierwszym z nich jest stosowana w algorytmie zamiana liczb z systemu m-rnego na binarny drugim - nieuwzględnianie przy kodowaniu tendencji do grupowania się podobnych do siebie grup Z. Jak wykazała przeprowadzona analiza (patrz 4.1.3), znaczenie pierwszego z tych czynników jest stosunkowo niewielkie. Wpływ drugiego z tych czynników na obniżenie sprawności kodowania Z

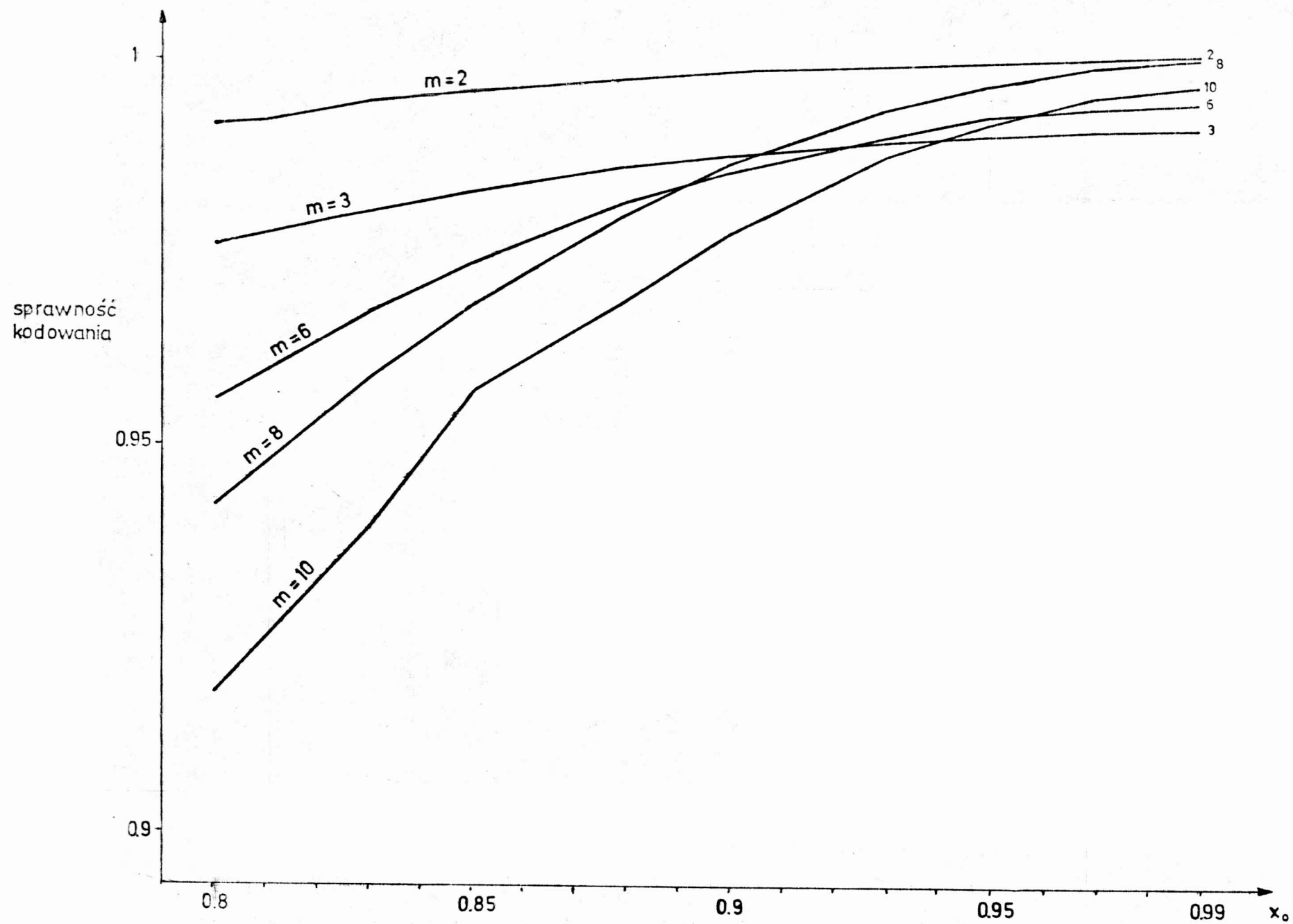
zwiększa się wraz ze wzrostem x_1 i m (patrz tab. 9). Zakres wpływu każdego z nich odczytać można z rys. 9 i 10. Przedstawiona na nich jest sprawność kodowania Z dla kilku wybranych wartości m przy $b = 23$ (rys. 9) i $b = 31$ (rys. 10). Krzywe odpowiadające wartościom m równym całkowitym potęgom 2 mają taki sam przebieg na obu wykresach. Sprawność kodowania jest obniżana w tym przypadku jedynie przez drugi z wymienionych wyżej czynników. Jak widać dla $x_0 \approx 0,8$, sprawność ta rośnie wraz z maleniem m - decydujący wpływ mają występujące zależności między kolejnymi grupami Z , natomiast dla $x_0 \approx 1$, uporządkowanie sprawności kodowania Z jest zgodne z kolejnością wyznaczoną przez odpowiednie wartości funkcji $F(k,m)$. Zmiana wartości b z 23 na 31 w największym stopniu poprawiła sprawność kodowania dla $m = 3$.

Tab. 9. Zróżnicowanie wartości prawdopodobieństw

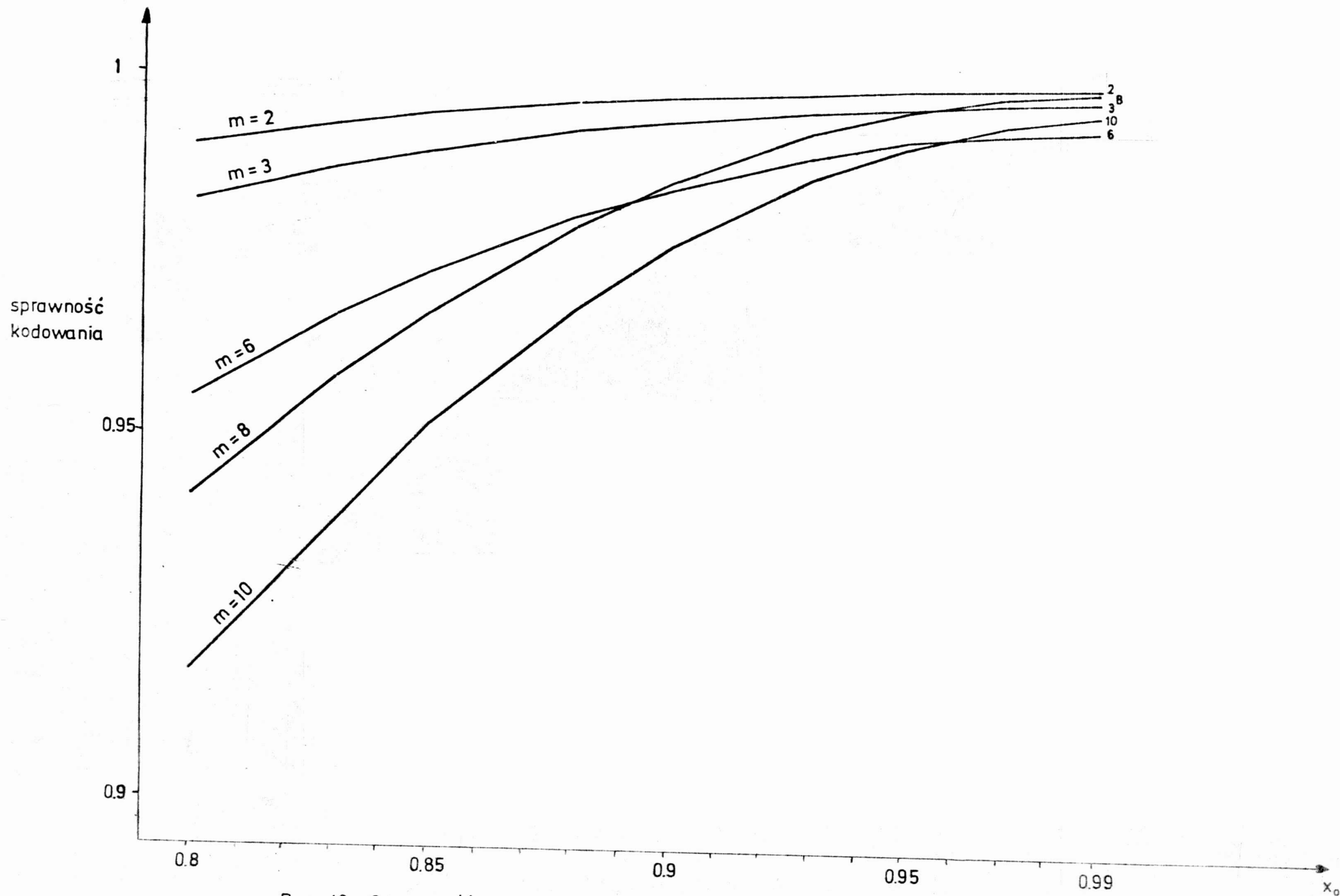
$P_r(e(r_i, r_{i+1}) = k)$ dla wybranych m oraz X_0

$$(H(P_r) = - \sum_{k=1}^m P_r(e(r_i, r_{i+1}) = k) \lg P_r(e(r_i, r_{i+1}) = k)$$

m	X_0	$P_r(e(r_i, r_{i+1}) = 1)$	$P_r(e(r_i, r_{i+1}) = m)$	$\lceil \lg m \rceil$	$H(P_r)$
2	0,8	0,5555	0,4445	1	0,99107
	0,9	0,52632	0,47368	1	0,998
	0,99	0,5025	0,4975	1	0,9998
3	0,8	0,33875	0,17441	2	1,956
	0,9	0,29078	0,21198	2	1,99
	0,99	0,2538	0,2462	2	1,999
4	0,8	0,2403	0,0504	3	2,826
	0,9	0,1756	0,084	3	2,9587
	0,99	0,1294	0,1206	3	2,999



Rys. 9. Sprawność kodowania x dla wybranych wartości m ($b = 23$)



Rys. 10. Sprawność kodowania wektora x dla wybranych wartości m ($b = 31$)

4.2.2. Wektory niejednorodne

Jak już to zaznaczono w rozdziale 2, jedynki w spotykanych w praktyce wektorach binarnych przejawiają często silną tendencję do grupowania się. Wektory posiadające takie własności generowane są przez źródła złożone opisane w rozdziale 3. Źródło takie określa się poprzez podanie X_{OA} i X_{OB} - prawdopodobieństw wytworzenia 0 przez oba jego bezpamięciowe źródła składowe oznaczane przez A i B oraz przez podanie macierzy T

$$T = \begin{bmatrix} t_{AA} & t_{AB} \\ t_{BA} & t_{BB} \end{bmatrix}$$

określające warunkowe prawdopodobieństwa zmiany źródła składowego generującego dane.

Jeżeli przyjmie się założenie, że zmiany w rozkładzie $P_r(e(r_i, r_{i-1}) = k)$ występujące przy kodowaniu elementów X pochodzących z różnych źródeł składowych nie wpływają w istotny sposób na przebieg kompresji, to częstość występowania 0 w tym fragmentach Y, które pochodzą ze źródła składowego A wynosi:

$$Y_{OA} = X_{OA}^m \quad (5)$$

oraz analogicznie $Y_{OB} = X_{OB}^m$.

Średnia długość ciągu elementów pochodzących ze źródła A w wektorze X równa jest

$$X_A = \sum_{i=1}^{\infty} i t_{AA}^{i-1} t_{AB} = \frac{t_{AB}}{t_{AA}} \sum_{i=1}^{\infty} i t_{AA}^i = \frac{1}{t_{AB}}.$$

Długość takiego ciągu w wektorze Y jest mniejsza i wynosi:

$$Y_A = \frac{X_{1A}}{(1 - X_{OA}^m) t_{AB}}$$

oraz analogicznie:

$$Y_B = \frac{X_{1B}}{(1 - X_{OA}^m) t_{AB}}.$$

Własności wektora Y są zatem zbliżone do własności wektora wytworzonego przez źródło złożone o macierzy przejść

$$T = \begin{bmatrix} 1 - \frac{1}{Y_A} & \frac{1}{Y_A} \\ \frac{1}{Y_B} & 1 - \frac{1}{Y_B} \end{bmatrix}$$

i prawdopodobieństwach wygenerowania 0 podanych przez równanie (5).

Wektor Y powstaje z X po usunięciu z niego części zer. W Y jedynki przejawiają zatem silniejszą niż w X tendencję do grupowania się, co może pozwolić na zmniejszenie ich liczby przez kodowanie predykcyjne. Przy przetwarzaniu wektorów binarnych wystarczające jest zastosowanie uproszczonej wersji takiego kodowania. Jedynym jego parametrem jest, oznaczana przez k, liczba wiadomości elementarnych uwzględnionych w trakcie badania własności wektora. Jeżeli $P(c.0) \gg P(c.1)$, gdzie c jest ciągiem k ostatnio analizowanych wiadomości elementarnych, to przewidywaną wartością kolejnej wiadomości elementarnej jest 0, w wypadku przeciwnym jest nią 1.

Przykład 18

Źródło złożone z $P_{AO} = 0,5$ $P_{BO} = 0,95$

$$T = \begin{bmatrix} 0,9 & 0,1 \\ 0,01 & 0,99 \end{bmatrix}$$

Po skompensowaniu wektora X utworzonego przez źródło z przy $m = 3$ otrzymujemy:

$$Y_{OA} = 0,125; \quad Y_{OB} = 0,857$$

$$T = \begin{bmatrix} 0,825 & 0,175 \\ 0,0285 & 0,9715 \end{bmatrix}$$

Dla $k = 2$

$$P(001) = 0,104147; \quad P(000) = 0,516956;$$

$$\begin{aligned} P(011) &= 0,029482; & P(010) &= 0,104147; \\ P(101) &= 0,039960; & P(100) &= 0,093669; \\ P(111) &= 0,071678; & P(110) &= 0,039960. \end{aligned}$$

Przybliżone prawdopodobieństwa pojawienia się 0 w Y przed i po kodowaniu predykyjnym wynoszą odpowiednio 0,7545 oraz $P(000) + P(010) + P(100) + P(111) = 0,786$.

4.3. Wykorzystanie algorytmu A do kompresji wektorów binarnych

4.3.1. Algorytm A_1 - kompresja jednokrotna

Długości wektora Y i kodu wektora Z po zakodowaniu algorytmem A_1 bezpamięciowego wektora S wynoszą odpowiednio:

$$\bar{Y} = \bar{X} \frac{x_1}{1 - x_0^m}$$

oraz

$$\tilde{Z} = w(m) \bar{X} x_1 \text{ lb } m$$

Kod Z tworzy $\bar{X} x_1$ cyfr z m-rnego systemu pozycyjnego; wprowadzenie w (m) umożliwia uwzględnienie wzrostu długości zapisu X wywołane jego kodowaniem. Przeprowadzone obliczenia wykazały, że dla $b = 31$ i $2 \leq m \leq 28$, $w(m) < 1,0232$.

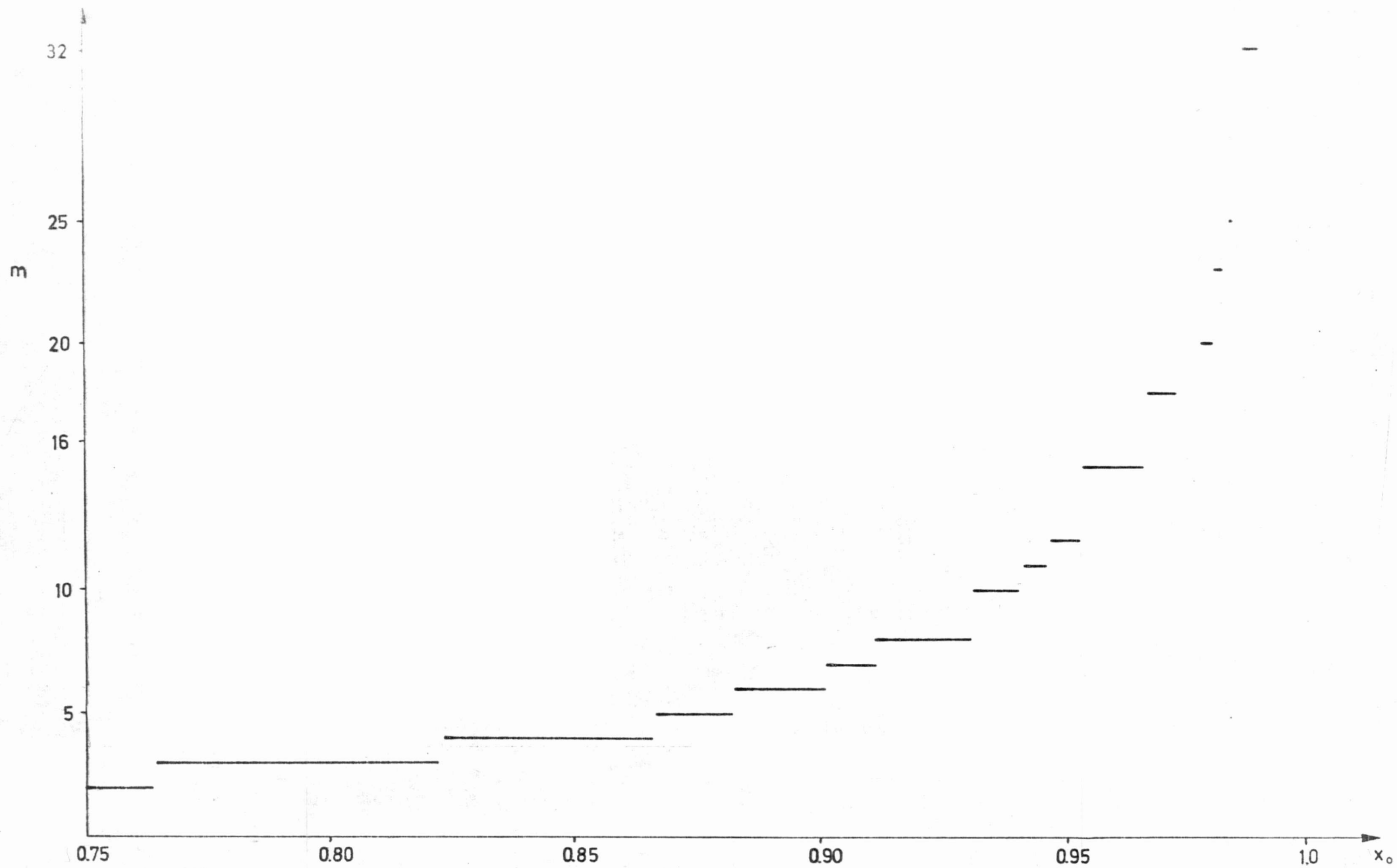
Stosując algorytm A_1 zakładamy, że jedynym kryterium oceny pracy algorytmu jest wartość współczynnika kompresji. Optymalną wartością m jest zatem liczba całkowita minimalizująca Y i Z. Optymalne wartości m dla $0,75 < s_0 < 0,995$ są przedstawione na rys. 11. Przy ich wyznaczaniu można posłużyć się przybliżoną zależnością:

$$m_{\text{opt}} = \left\lfloor \frac{\ln 0,5}{\ln x_0} + 0,6 \right\rfloor$$

Osiągany współczynnik kompresji wynosi:

$$k_m = \frac{1}{w(m)x_1 \text{ lb } m + \frac{x_1}{1 - x_0^m}} = \frac{1 - x_0^m}{x_1 [w(m)y_1 \text{ lb } m + 1]}$$

Przeprowadzony eksperyment wykazał dość dużą zgodność prze-



Rys. 11. Algorytm A_1 - optymalne wartości m .

widywanych, osiąganych i maksymalnych współczynników kompresji (patrz tab. 10). Stabilność K_m algorytmu A_1 przedstawia tab. 11. Umieszczone w czwartej kolumnie cyfry t spełniają równanie:

$$t = \lceil 100 R_m(X) \rceil$$

gdzie $R_m(X)$ jest redundancją kodowania algorytmem A_1 bezpamięciowego wektora X ; przy grupie utworzonej z m elementów.

W drugiej kolumnie zapisano redundancję kodowania X dla optymalnej wartości m .

Tab. 10. Porównanie przewidywanych, osiągniętych i maksymalnych wartości współczynników kompresji uzyskiwanych przez A_1

x_0	m	Współczynnik kompresji		
		osiągnięty	przewidywany	maksymalny
0,2516	2	1,214	1,214	1,229
0,1031	6	2,070	2,070	2,089
0,0509	12	3,411	3,421	3,446
0,0102	64	12,184	12,184	12,227

4.3.2. Algorytm A_w - kompresja wielokrotna

Jak wykazała przeprowadzona uprzednio analiza przy m równym 2 algorytm A przekształca bezpamięciowy wektor X na dwa bezpamięciowe wektory: wektor Y oraz kod wektora Z , a ponadto łączna ilość informacji zawartej w Y i Z jest równa ilości zawartej w X . Nie oznacza to bynajmniej, że kodowanie zawsze prowadzi do kompresji danych. Zakres wartości x_0 , dla których osiągane jest skrócenie długości zapisu wektora wejściowego (dla $m = 2$) w wyniku kodowania proponowanym algorytmem wyznacza nierówność:

Tab. 11. Algorytm A_1 - stabilność wartości współczynnika kompresji

X_0	m_{opt}	R_{min}	5	10	20	30	40	50	60	70	80	90
0.995	111	0.007							9	998877687766655554444344334333233322		
0.990	64	0.004						98787655454434333222222133322211133222211111221122212222233322				
0.985	45	0.005						987765444543232221211121122121221224433332244444333443444455445555566666				
0.980	32	0.004						876544332222322132112121223223323343343666555555777776666777777888999999				9
0.975	28	0.006						87644322222111213221332232333444554556666688888887		99999		
0.970	21	0.006						86544222111221112243324444455556667777788888				
0.965	17	0.005						976432312111122223355446656667677888999	9			
0.960	16	0.005						97543223122122333334466658777889899				
0.955	16	0.006						965332113122223444556688779999	9			
0.950	13	0.007						75322111322333455566779999				
0.945	12	0.008						9533221123234445677899				
0.940	10	0.008						74221212243455567889				
0.935	10	0.007						95322122235456677899				
0.930	8	0.008						74212122346567788				
0.925	8	0.006						6321223345667889				
0.920	8	0.005						532122334577899				
0.915	8	0.006						84211224456889				
0.910	7	0.008						7321123456799				
0.905	7	0.008						63212345678				
0.900	6	0.009						52112346789				
0.895	6	0.009						4212245779				
0.890	6	0.009						9321235678				
0.885	5	0.010						8312235689				
0.880	5	0.009						721234679				
0.875	5	0.009						62123478				
0.870	5	0.009						62124579				
0.865	4	0.010						51234689				
0.860	4	0.008						4123569				
0.855	4	0.007						4123579				
0.850	4	0.006						312468				
0.845	4	0.006						312468				
0.840	4	0.007						213579				
0.835	4	0.008						921358				
0.830	4	0.009						921368				
0.825	4	0.010						822469				
0.820	3	0.010						722479				
0.815	3	0.009						71257				
0.810	3	0.008						61258				
0.805	3	0.008						51269				
0.800	3	0.008						51369				
0.795	3	0.008						4137				
0.790	3	0.008						4137				
0.785	3	0.009						4148				
0.780	3	0.010						3248				
0.775	3	0.012						3259				
0.770	3	0.013						3259				
0.765	3	0.015						225				
0.760	2	0.016						226				
0.755	2	0.014						226				

$$\frac{\bar{x} x_1}{1 - x_0^2} + \bar{x} x_1 < \bar{x}$$

zatem w kompresowalnych wektorach wartość x_0 musi się mieścić w granicach:

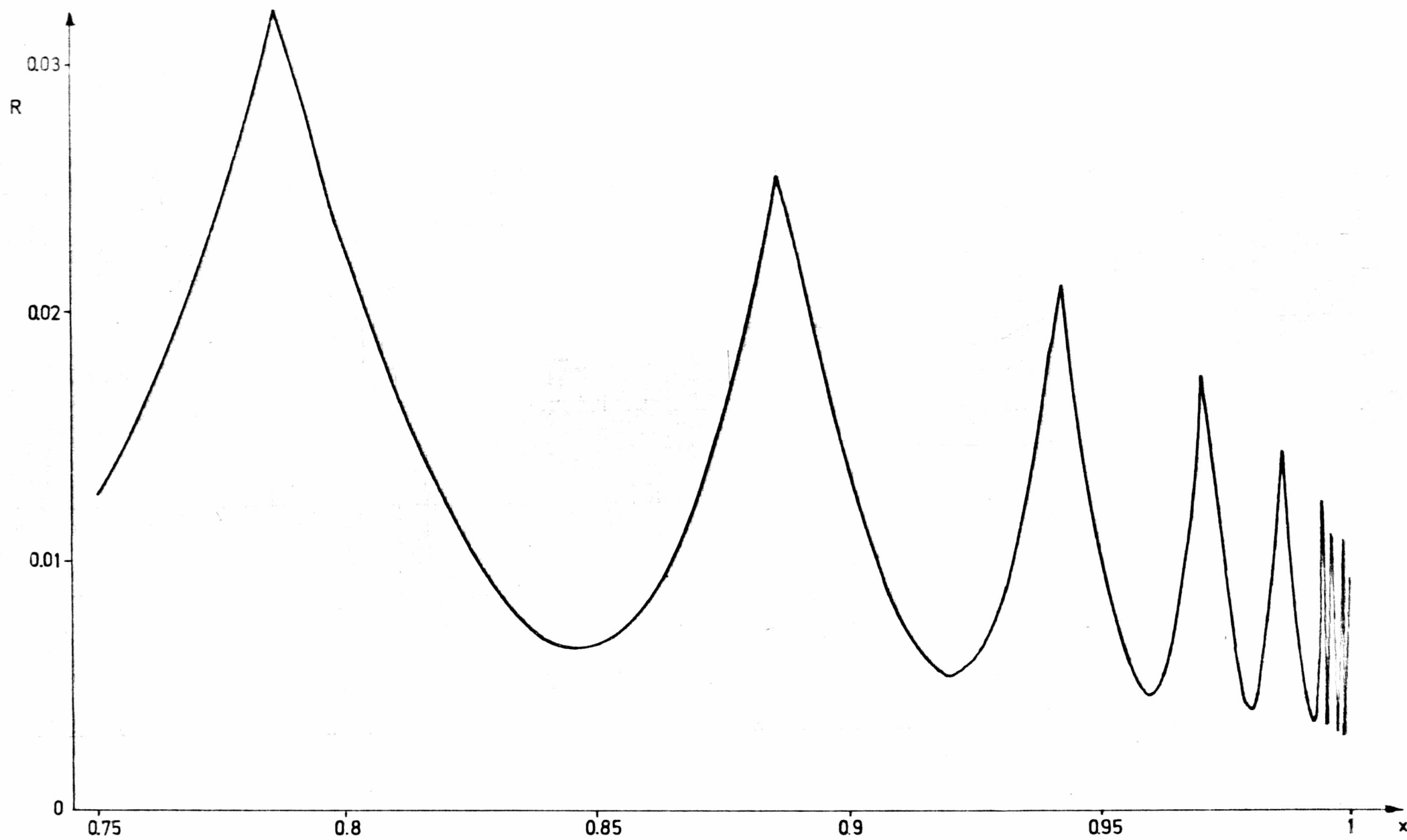
$$1 > x_0 > \frac{\sqrt{5} - 1}{2} \approx 0,618 \quad (6)$$

A_w jest iteracyjnym algorytmem kompresji. Długość grupy w A_w wynosi zawsze 2, ułatwia to znacznie kodowanie Z. Kolejno tworzone wektory Z nie są po ich zakodowaniu dalej przetwarzane, natomiast wektory Y są kompresowane na takich samych zasadach jak X w algorytmie A. Proces ten jest kontynuowany tak długo, aż częstość występowania 0 w utworzonym wektorze Y nie wykróczy poza zakres definiowany nierównością (6). Na rys. 12 przedstawiono wykres zależności redundancji kodowania algorytmem A_w od wartości x_0 . Lokalne maksima redundancji przypadają dla tych wartości x_0 , dla których wynikowe y_0 znacznie różni się od 0,5.

Główną zaletą algorytmu A_w jest jego bezparametryczność. W takiej samej postaci jest on stosowany do kompresji wszystkich ciągów bezpamięciowych. Do jego wad natomiast zaliczyć należy konieczność wielokrotnego przetwarzania wektorów (co wydłuża czas kodowania i dekodowania) oraz niewielką stabilność współczynnika kompresji.

4.3.3. Algorytm A_2 - kompresja dwukrotna

Przedstawione powyżej algorytmy A_1 i A_w posiadają wzajemnie uzupełniające się zalety. Algorytm A_1 charakteryzuje większa szybkość działania niż A_w , a ponadto osiąga on współczynniki kompresji o wartościach zbliżonych do maksymalnych. A_w jest algorytmem bezparametrycznym. Algorytm A_2 traktować można jak połączenie A_1 i A_w . Kompresja przebiega w dwu etapach. W pierwszym z nich w trakcie kodowania X algorytmem A oblicza się jednocześ-



Rys. 12. Algorytm A_w – redundacja kodu.

nie y_0 , co pozwala w drugim etapie na kompresję wektora Y przy wykorzystaniu optymalnej wartości m .

Niech m_1 i m_2 oznaczają długości grup odpowiednio w pierwszym i drugim etapie kompresji. Zgodnie z uprzednio wykazanymi zależnościami po pierwszym etapie otrzymujemy:

$$y_0 = x_0^{m_1}; \quad \bar{Y} = \frac{\bar{X} x_1}{1 - x_0^{m_1}}; \quad \tilde{Z} \approx x_1 \bar{X} \text{ lb } m_1; \quad \tilde{X} = \bar{Y} + \tilde{Z}$$

W drugim etapie przetwarzany jest wektor Y , długości utworzonych wtedy ciągów Y' i Z' wynoszą:

$$\bar{Y}' = \bar{Y} \frac{1 - x_0^{m_1}}{1 - x_0^{m_1 m_2}} = \frac{\bar{X} x_1}{1 - x_0^m} \frac{1 - x_0^{m_1}}{1 - x_0^m} = \frac{\bar{X} x_1}{1 - x_0^m}$$

$$\bar{Z}' \approx \bar{Y} y_1 \text{ lb } m_2 = \frac{\bar{X} x_1}{1 - x_0^{m_1}} (1 - x_0^{m_1}) \text{ lb } m_2 = \bar{X} x_1 \text{ lb } m_2$$

Łączna długość X po jego skompresowaniu wynosi zatem:

$$\begin{aligned} \tilde{X} &\approx \tilde{Z} + \bar{Z}' + \bar{Y}' = \bar{X} x_1 \text{ lb } m_1 + \bar{X} x_1 \text{ lb } m_2 + \frac{\bar{X} x_1}{1 - x_0^m} = \\ &= \bar{X} x_1 \text{ lb } m + \frac{\bar{X} x_1}{1 - x_0^m} \end{aligned}$$

Jak widać $\tilde{X}$ jest prawie dokładnie równe długości wektora X po zakodowaniu go algorytmem A_1 przy $m = m_1 m_2$. Różnice w tych długościach wynikają jedynie z odmiennych sprawności kodowania X dla różnych m .

Dla $m_1 = 2$ możliwe jest efektywne kompresowanie wektorów X najbardziej zróżnicowanych wartościach x_0 . Jak wykazał eksperyment, dla $m_1 = 2, 3, 4, 5$ algorytm A_2 cechuje znaczna stabilność współczynnika kompresji (patrz tab. 12). Maksymalne (bądź też zbliżone do maksymalnych) wartości współczynnika kompresji osiągane były dla $m_1 = 4$. Za wyborem takiej wartości m_1 przemawia także i to, że ułatwia ona w znacznym stopniu kodowanie wektora Z .

4.3.4. Algorytm A_n

W rozdziale 2 przedstawiono algorytmy przeznaczone do kompresji wektorów niejednorodnych. Uzyskują one współczynniki kompresji przekraczające $1/h(x)$. Osiągnięcie takiej kompresji możliwe jest dzięki, dostosowywaniu parametrów działania algorytmu do specyficznych własności wektora, umieszczaniu w skompresowanej postaci wektora X jego fragmentów zawierających jedynek lub też dzięki kodowaniu predykcyjnemu. Algorytm A w wersji przeznaczonej do kodowania wektorów niejednorodnych (oznaczanej przez A_n) wykorzystuje to ostatnie podejście.

A_n jest zmodyfikowaną wersją A_2 . W algorytmie A_n kompresja Y jest poprzedzona kodowaniem predykcyjnym tego wektora. Zadaniem kodowania predykcyjnego jest zmniejszenie liczby jedynek w Y oraz doprowadzenie do bardziej równomiernego ich rozłożenia w tym wektorze. Oba te czynniki powodują, że przekształcony Y jest kompresowany algorytmem A w większym stopniu, niż jest to możliwe do osiągnięcia przy jego pierwotnej postaci. Niezbędne do kodowania predykcyjnego ustalenie własności wektora Y ma miejsce podczas jego tworzenia, zaś samo kodowanie predykcyjne w czasie kodowania Y .

W praktyce, przy kodowaniu wektorów zamiast obliczenia $P(c_k.0)$ i $P(c_k.1)$ (patrz 4.2.2) stosowana jest w algorytmie A_n analiza własności statystycznych wektora Y . Analiza ta sprowadza się do ustalenia, jaka wartość elementu częściej występuje po każdej k elementowej sekwencji. W trakcie kodowania predykcyjnego po pojawieniu się określonej sekwencji przewiduje się, że kolejny element ma tę wartość, która w trakcie analizy występowała częściej po tej sekwencji. Przed rozpoczęciem zarówno analizy jak i kodowania predykcyjnego, zakłada się, że sekwencja ta przejmie z góry ustaloną postać.

Kodowanie predycyjne można znacznie przyspieszyć poprzez wykorzystanie tablicy P. Umożliwia ona jednoczesne przekodowanie m kolejnych elementów y. Takie przetwarzanie wektora y jest zgodne ze sposobem kodowania tego wektora przez algorytm A w drugim etapie kompresji. Element $P(i,j)$, gdzie i oraz j są odpowiednio ostatnio przekodowaną i aktualnie przetwarzaną grupą Y, określa postać grupy po kodowaniu predycyjnym.

Przykład 19.

Dla $k = 2$, $Y = 001\ 111\ 110\ 010\ 000\ 000\ 100\ 011\ 111\ 110$, początkowa sekwencja składa się z samych zer

		sekwencja bitów			
		00	01	10	11
liczba wystąpień po sekwencji	0	8	2	3	2
	1	4	2	0	9
Przewidywana wartość bitu		0	1	0	1

Przy $m = 3$, tablica P przyjmuje postać:

grupa aktualnie przekodowywana

grupa ostatnio przekodowana		000	001	010	011	100	101	110	111
	000	000	001	011	010	110	111	101	100
	001	100	101	111	110	010	011	001	000
	010	000	001	011	010	110	111	101	100
	011	100	101	111	110	010	011	001	000
	100	000	001	011	010	110	111	101	100
	101	100	101	111	110	010	011	001	000
	110	000	001	011	010	110	111	101	100
	111	100	111	111	110	010	011	001	000

$$y_0 = 0,5 \quad y'_0 = 22/30 \approx 0,733$$

$$Y' = 001000001011000000110010000001$$

Skompresowana postać wektora X musi zawierać 2^k przewidywanych wartości elementów. Wzrost m w pierwszym etapie kompresji skraca Y oraz silniej grupuje jedyńki w tym sektorze, co pozwala na ich skuteczniejsze usuwanie przy kodowaniu predykcyjnym. Towarzyszy temu jednakże wzrost długości kodu X . Oba te, częściowe kompensujące się czynniki, mają wpływ na wynikową wartość współczynnika kompresji. W rezultacie algorytm A_n cechuje dość znaczna stabilność jego wartości. Algorytm A_n stosowany do kompresji wektorów bezpamięciowych skraca długość ich zapisu w stopniu nie mniejszym niż A_2 .

Zestawienie rzeczywistych i przewidywanych wartości współczynnika kompresji jest zamieszczone w tab. 13.

W trakcie przeprowadzonego eksperymentu $k = 5$. Maksymalne lub też niewiele od nich odbiegające wartości współczynnika kompresji (podobnie jak to miało miejsce w wypadku algorytmu A_2) osiągane były przez A_n przy $m_1 = 4$. Wykorzystanie przy kodowaniu X grup o długości 4 bitów w pierwszym etapie kompresji pozwala zatem na osiągnięcie znacznych jej wartości przy kodowaniu wektorów występujących w SWI, a tym samym uwalnia od potrzeby ustalania przed kompresją własności X .

Tab. 12. Algorytm A_2 - stabilność wartości współczynnika kompresji

x_0	k_m max	k_m osiągnięty	m_1	m_2
0,09855	2,153	2,138	2	3
		2,138	3	2
		2,126	4	2
		2,080	5	2
0,04927	3,530	3,502	2	7
		3,506	3	4
		3,494	4	4
		3,500	5	3
0,0105	12,227	12,184	2	32
		12,129	3	19
		12,184	4	16
		12,118	5	13

Tab. 13. Porównanie przewidywanych i osiągniętych wartości współczynników kompresji algorytmu A_n

x_0 przewidy- wany	x_0 rzeczy- wisty	k_m przewidy- wany	k_m rzeczy- wisty
0,0901	0,09018	2,370	2,369
0,0682	0,0668	3,058	3,089
0,0545	0,05244	3,828	3,967
0,050	0,04878	4,247	4,309

5. OPIS EKSPERYMENTU

W ramach przeprowadzonego eksperymentu zbadano współczynniki kompresji osiągane przez opisane w poprzednim rozdziale wersje A_1 , A_2 i A_n algorytmu A, kody Huffmana przydzielane długościom runów lub fragmentom wektora o stałej długości, uogólnione kody Golomba a także przez algorytm iteracyjny. Celem eksperymentu było porównanie rzeczywistych i przewidywanych współczynników kompresji uzyskanych przez proponowany algorytm, a także uzupełnienie tych danych dotyczących innych algorytmów, które nie były podane w dostępnej literaturze.

Wykorzystany w eksperymencie program został napisany w Fortranie.

Program ten jest zamieszczony w zał. 3. Uzyskane rezultaty pozwalają jedynie na oszacowanie współczynników kompresji.

Większość rozpatrywanych algorytmów wykorzystuje do kompresji algorytmy o podobnym stopniu złożoności i dlatego też różnice w czasach kodowania realizowanego przez różne algorytmy tylko w nieznacznym stopniu wpływają na globalny czas przetwarzania danych. Czynnikiem mającym największy wpływ na szybkość działania SWI jest liczbaostępów do pamięci pomocniczej koniecznym do obsługi pytania użytkownika..

5.1. Generowanie i testowanie wektorów binarnych

Przy generowaniu wektorów binarnych wykorzystano standardowy podprogram FPMCRV. Jest to multiplikatywny generator liczb pseudolosowych o rozkładzie równomiernym w przedziale (0,1).

Generowanie liczb przez podprogram musi być poprzedzone określeniem wartości startowej, od której rozpoczyna obliczanie kolej-

nych liczb pseudolosowych. Stosowane w eksperymencie wartości startowe wynosiły $a = 0,12345678901$ oraz $b = 0,21547965$. Uzyskane z generatora ciągi liczb poddano następnie testom. Testy stosowane do weryfikowania generatorów liczb pseudolosowych można podzielić na dwie grupy: testy zgodności rozkładu i testy niezależności [49]. Zadaniem testów pierwszej grupy jest zweryfikowanie hipotezy o pochodzeniu uzyskanych liczb z populacji o określonym rozkładzie. W ramach przeprowadzonego eksperymentu badano: wartości oczekiwanej średniej arytmetycznej liczb oraz ich kwadratów, równomierność rozkładu liczb oraz ilość liczb przekraczających zadaną wielkość graniczną. Testy drugiej grupy mają za zadanie ustalić, czy wygenerowane liczby są niezależne względem siebie. Przeprowadzony test niezależności rozkładu uwzględnił sposób wykorzystania generatora. Badał on długości runów zer w utworzonym wektorze.

W trakcie tworzenia wektora binarnego w celu ustalenia wartości każdego z jego elementów generowano liczbę pseudolosową a następnie porównywano ją z parametrem p . Jeżeli liczba była większa od p , wówczas nadawano mu wartość 0, w przeciwnym wypadku 1.

W eksperymencie wykorzystano następujące źródła bezpamięciowe:

$B0_v(0,25)$, $B1_v(0,1)$ $B2_v(0,05)$ $B3_v(0,01)$.

Liczby w nawiasach podają odpowiednie prawdopodobieństwa wygenerowania jedynki, natomiast $v \in \{a, b\}$ wskazuje na startową wartość parametru podprogramu FPMCRV. Wykorzystywane do generowania wektorów binarnych źródła złożone posiadały macierz T o postaci:

$$T = \begin{bmatrix} 0,9 & 0,1 \\ 0,01 & 0,99 \end{bmatrix}$$

Wartość P_{A1} wynosiła 0,5. Źródła te różniły się wartościami P_{B1} wynoszącymi odpowiednio:

$Z0_v = 0,05$ $Z1_v = 0,025$ $Z2_v = 0,01$ $Z3_v = 0,005$

5.1.1. Badanie średniej arytmetycznej liczb i ich wariancji

Niech $\bar{X}_N$ oznacza zmienną losową o wartości równej średniej arytmetycznej N kolejnych liczb pseudolosowych, zaś $E(\bar{X}_N)$ - jej wartość oczekiwana. Testowany algorytm ma generować liczby o rozkładzie równomiernym w przedziale $(0,1)$, należy zatem zweryfikować hipotezę $E(\bar{X}_N) = 0,5$ w stosunku do hipotezy alternatywnej $E(\bar{X}_N) \neq 0,5$. W tym celu wyznacza się wartości c_1 i c_2 takie, że:

$$P(\bar{X}_N \leq c_1) = P(\bar{X}_N > c_2) = \frac{\alpha}{2}$$

gdzie α jest tzw. poziomem istotności testu. Zwykle stosuje się $\alpha = 0,1, 0,05, 0,01$. Liczby c_1 i c_2 wyznaczają przedział wartości $\bar{X}_N$, w którym powinna się znaleźć z prawdopodobieństwem $1 - \alpha$ średnia arytmetyczna liczb. Liczby te można obliczyć, korzystając z zależności [49] :

$$P(\bar{X}_N \leq x) \approx \Phi \left[\left(x - \frac{1}{2} \right) \sqrt{12N} \right] - \frac{1}{20N} \Phi^{(3)} \left[\left(x - \frac{1}{2} \right) \sqrt{12N} \right]$$

gdzie Φ i $\Phi^{(3)}$ są odpowiednio dystrybucją i trzecią pochodną gęstości rozkładu normalnego $N(0,1)$.

Wartości c_1 i c_2 dla $\alpha = 0,1; 0,05; 0,01$ oraz $N = 10000, 20000, \dots, 50000$ są zamieszczone w tab. 14. Ostatnia kolumna tej tablicy zawiera zaobserwowane $\bar{X}_N$ dla obu wartości startowych algorytmu.

W analogiczny sposób porównuje się przewidywaną i zaobserwowaną wariancję liczb. Weryfikuje się wówczas hipotezę $E(\bar{X}_N^2) = \frac{1}{3}$ przy hipotezie alternatywnej $E(\bar{X}_N^2) \neq \frac{1}{3}$. Przy wyznaczaniu c_1 i c_2 korzysta się z zależności:

$$P(\bar{X}_N^2 \leq x) \approx \Phi \left[\left(x - \frac{1}{3} \right) \sqrt{\frac{45N}{4}} \right] - \frac{0,1065}{N} \Phi^{(2)} \left[\left(x - \frac{1}{3} \right) \sqrt{\frac{45N}{4}} \right]$$

Przewidywane i zaobserwowane wartości $\bar{X}_N^2$ są zamieszczone w tab. 15

Tab. 14 Badanie średniej arytmetycznej
liczb losowych

N	α			wartość początkowa FPMCRV	
	0,1	0,05	0,01	a	b
10 000	0,50476	0,5057	0,5074	0,49977	0,498977
	0,49524	0,4943	0,4926		
20 000	0,50335	0,504	0,5052	0,497827	0,499663
	0,49665	0,496	0,4948		
30 000	0,5027	0,5033	0,5043	0,499275	0,498363
	0,4973	0,4967	0,4957		
40 000	0,5024	0,5028	0,5037	0,499949	0,499295
	0,4976	0,4972	0,4963		
50 000	0,5021	0,5025	0,5033	0,500205	0,500781
	0,4979	0,4975	0,4967		

Tabela 15, Badanie średniej arytmetycznej
kwadratów liczb losowych

N	α			wartość początkowa FPMCRV	
	0,1	0,05	0,01	a	b
10 000	0,3383	0,33918	0,34097	0,332415	0,331577
	0,3284	0,3275	0,32570		
20 000	0,33681	0,33749	0,33875	0,331351	0,333104
	0,32985	0,32918	0,32792		
30 000	0,33617	0,33671	0,33774	0,332565	0,331942
	0,33049	0,329959	0,328927		
40 000	0,335793	0,336255	0,337149	0,333528	0,332862
	0,330874	0,330411	0,329517		
50 000	0,33553	0,335947	0,336760	0,333945	0,334238
	0,331133	0,330720	0,329907		

5.1.2. Badanie równomierności rozkładu

Niech D będzie ciągiem k liczb takich, że $d_1 < d_2 < \dots < d_k$ oraz $d_1 > 0$ i $d_k < 1$, zaś n_i oznacza liczbę wygenerowanych liczb pseudolosowych takich, że mieszczą się one w przedziale (c_{i-1}, c_i) . Dla dostatecznie dużych N statystyka:

$$\chi^2_{k-1} = \sum_{i=1}^k \frac{(n_i - Np_i)^2}{Np_i} = \frac{1}{N} \sum_{i=1}^k \frac{n_i^2}{p_i} - N \quad (7)$$

gdzie $p_i = d_i - d_{i-1}$ ma w przybliżeniu rozkład chi-kwadrat o $k-1$ stopniach swobody.

Zwykle test ten stosowany jest już wtedy, gdy $n_i > 5$ dla każdego. Statystykę wyznaczoną równaniem (7) zastosowano do zbadania równomierności rozkładu liczb w przedziale $(0,1)$ (przy $k = 101$). Wartości graniczne przyjmowane przez nią dla poziomów istotności $\alpha = 0,1; 0,05; 0,01$ oraz wartości uzyskane w trakcie badań liczb są podane w tab. 16.

5.1.3. Badanie liczby jedynek w wektorze

Niech S_N oznacza zmienną losową, której wartością jest liczba jedynek w badanym wektorze binarnym x . Przy poprawnym działaniu algorytmu generującego liczby losowe zmienna ta ma rozkład dwumienowy. Prawdopodobieństwo zaobserwowania w wektorze dokładnie k jedynek wynosi wtedy:

$$P(S_N = k) = \frac{N!}{k!(N-k)!} (1-p)^{N-k} p^k \quad (8)$$

Wzór (8) pozwala na wyznaczenie takich c_1 i c_2 , że

$$P(c_1 \leq S_N \leq c_2) = \frac{N!}{c_1!(N-c_1)!} (1-p)^{N-c_1} p^{c_1} + \dots + \frac{N!}{c_2!(N-c_2)!} (1-p)^{N-c_2} p^{c_2} = 1 - \alpha \quad (9)$$

Tab. 16. Badanie równomierności rozkładu (test
chi-kwadrat o 100 stopniach swobody)

wartości graniczne:

	α		
	0,1	0,05	0,01
G_1	82,36	77,93	70,06
G_2	118,50	124,34	135,81

wyniki badania:

N	wartości początkowe ppgm FPMCRV	
	a	b
10 000	87,314	112,181
20 000	122,725	97,727
30 000	114,894	91,980
40 000	117,851	92,783
50 000	116,24	103,127

Prawdopodobieństwo uzyskania N elementowego wektora, w którym liczba jedynek byłaby mniejsza niż c_1 lub większa niż c_2 wynosi zatem

Wyznaczanie c_1 i c_2 bezpośrednio ze wzoru (9) wiąże się z dużymi trudnościami obliczeniowymi i dlatego też w przypadku, gdy $Np < 10$ stosuje się zwykle przybliżenie rozkładu dwumianowego rozkładem normalnym. Wykorzystuje się do tego celu poniższą zależność [50] :

$$P(c_1 \leq S_N \leq c_2) \approx P\left(\frac{c_1 - 0,5 - Np}{Np(1-p)} \leq Z \leq \frac{c_2 + 0,5 - Np}{Np(1-p)}\right),$$

gdzie Z jest zmienną losową o rozkładzie $N(0,1)$.

Jak wykazał Raff [51], błąd przybliżenia nie przekracza $0,14/\sqrt{Np(1-p)}$. Wzór 8 umożliwia, analogicznie jak miało to miejsce przy badaniu średniej arytmetycznej liczb, obliczenie granicznych prawdopodobieństw jedynki w wektorach X_1 i X_1 takich, że dla poprawnie działającego generatora zachodzi:

$$P(X'_1 \leq X_1 \leq X''_1) = 1 - \alpha$$

Przewidywane i stwierdzone w trakcie eksperymentu wartości X_1 są zamieszczone w tab. 17.

5.1.4. Badanie długości runów

Do badania zaobserwowanych w utworzonych wektorach długości runów zer wykorzystano test chi-kwadrat. Przewidywane prawdopodobieństwo pojawienia się runu o długości 1 bitów wynosi:

$$P(\text{run} = 1) = p(1-p)^{1-1}$$

Statystyka przyjmuje zatem postać:

$$\chi^2_{k-1} = \sum_{i=1}^k \frac{(n_i - Nd_i)^2}{Nd_i} \quad (10)$$

gdzie

Tab. 17. Badanie częstości występowania jedynek w wektorze binarnym

N		x_0														
		0,1					0,05					0,01				
					war.pocz. FPMCRV					war.pocz. FPMCRV					war.pocz. FPMCRV	
		0,1	0,05	0,01	a	b	0,1	0,05	0,01	a	b	0,1	0,05	0,01	a	b
10 000	x'	0,09512	0,09435	0,09231	0,0980	0,0980	0,04647	0,04591	0,04443	0,0498	0,0482	0,00841	0,00816	0,00748	0,0087	0,0099
	x''	0,10488	0,10565	0,10769			0,05333	0,05409	0,05557			0,01153	0,01184	0,01252		
20 000	x'	0,09654	0,09593	0,09455	0,0998	0,0997	0,04743	0,04710	0,04605	0,0498	0,0504	0,00887	0,00869	0,00821	0,0102	0,0099
	x''	0,10346	0,10401	0,10545			0,05251	0,05290	0,05395			0,01113	0,01131	0,01179		
30 000	x'	0,09717	0,09673	0,09555	0,0987	0,1021	0,04795	0,04763	0,04677	0,0492	0,0515	0,00907	0,00893	0,00853	0,0104	0,0104
	x''	0,10283	0,10327	0,10445			0,05205	0,05237	0,05323			0,01093	0,01107	0,01147		
40 000	x'	0,09755	0,09716	0,09614	0,0985	0,1008	0,04822	0,04774	0,04720	0,0493	0,0514	0,00919	0,00907	0,00873	0,0101	0,0101
	x''	0,10245	0,10284	0,10386			0,05178	0,05206	0,05280			0,01081	0,01093	0,01128		
50 000	x'	0,09780	0,09746	0,09655	0,1006	0,1001	0,04841	0,04816	0,04750	0,0505	0,0511	0,00923	0,00916	0,00886	0,0105	0,0100
	x''	0,10219	0,10254	0,10345			0,05159	0,05184	0,05250			0,01072	0,01084	0,01114		

c.d. Tab. 17

		x_0				
		0,25				
N					war.pocz. FPMCRV	
		0,1	0,05	0,01	a	b
10 000	x'	0,24293	0,24182	0,23888	0,2483	0,2504
	x''	0,25707	0,25818	0,26121		
20 000	x'	0,24499	0,24421	0,24213	0,2545	0,2514
	x''	0,25501	0,25573	0,25787		
30 000	x'	0,24591	0,24527	0,24357	0,2516	0,2510
	x''	0,25409	0,25473	0,25643		
40 000	x'	0,24645	0,24590	0,24443	0,2522	0,2512
	x''	0,25355	0,25410	0,25557		
50 000	x'	0,24683	0,24633	0,24501	0,2520	0,2489
	x''	0,25317	0,25367	0,25499		

$$d_i = \begin{cases} p(1-p)^{i-1}, & i = 1, \dots, k-1 \\ 1 - \sum_{i=1}^{k-1} p(1-p)^{i-1}, & i = k \end{cases}$$

n_i - liczba zaobserwowanych runów o długości i bitów.

W stosowanym teście $k = 31$. Wartości graniczne przyjmowane przez statystykę określona równaniem (10). dla poziomów istotności $\alpha = 0,1, 0,05, 0,01$ oraz wartości uzyskane w trakcie badań są zamieszczone w tab. 18.

5.2. Porównanie sprawności kodowania algorytmów kompresji wektorów binarnych

Maksymalne wartości współczynników kompresji wektorów bezpanięciowych i pochodzących ze źródeł złożonych osiągnięte przez omawiane w pracy algorytmy zamieszczone są w tab. 19 i 20.

Wektory wykorzystane w trakcie eksperymentu miały długości wynoszące od 30000 do 50000 bitów. Każdy z wektorów wygenerowano w dwu wersjach, przy których tworzeniu wykorzystano wartości startowe podprogramu FPMCRV wynoszące odpowiednio 0,12345678901 oraz 0,21547965.

W tabelach tych podano zaobserwowane częstości występowania jedynek w przetwarzanych wektorach. Różnią się one od teoretycznie zakładanego prawdopodobieństwa pojawiania się jedynki. Dla wektorów bezpanięciowych o długościach rzędu kilkudziesięcia tysięcy bitów wystąpienie większej zgodności przewidywanej i rzeczywistej liczby jedynek jest jednak mało prawdopodobne, patrz tab. 17. Wszystkie wektory niejednorodnie wykorzystane w badaniach miały długość 50000 - maksymalną długość przyjętą dla eksperymentu.

Tab. 18. Badanie długości runów zer w
wektorze binarnym (test chi-kwadrat
o 30 stopniach swobody)

wartości graniczne:

	α		
	0,1	0,05	0,01
C_1	20,599	18,49	14,95
C_2	40,26	43,50	50,89

wyniki badania:

N	$x_0 = 0,1$		$x_0 = 0,05$	
	wartość początkowa FPMCRV		wartość początkowa FPMCRV	
	a	b	a	b
10 000	34,106	34,347	32,979	29,706
20 000	26,899	34,861	30,652	33,093
30 000	28,385	26,143	34,166	30,010
40 000	26,986	20,710	32,587	34,277
50 000	24,216	26,815	28,191	36,288

Tab. 19 Porównanie wartości współczynników kompresji wektorów bespanięciowych

	Rodzaj źródła											
	#	BO _a	BO _b	#	B1 _a	B1 _b	#	B2 _a	B2 _b	#	B3 _a	B3 _b
K _{max}	1,23	1,23	1,24	2,13	2,15	2,13	3,49	3,53	3,44	12,38	12,24	12,37
Przewidywane x ₁	0,25			0,1			0,05			0,01		
Rzeczywiste x ₁		0,252	0,249		0,985	0,1001		0,0493	0,051		0,0101	0,010
A ₁		1,21	1,22		2,13	2,11		3,50	3,41		12,18	12,34
A ₂ (m ₁ = 4)		1,15	1,16		2,13	2,10		3,49	3,41		12,18	12,34
Alg. Huffmana (max.dl.un.d=100)		1,21	1,22		2,14	2,12		3,53	3,43		12,36	12,49
Alg. Huffmana (dl. fgm. n=8)		1,23	1,24		2,14	2,12		3,34	3,33		6,37	6,38
Zmodyfikowany alg. Golomba (m _{max} =300)		1,16	1,18		2,12	2,09		3,49	3,38		11,95	11,88
Alg. Golomba	1,07			1,88			3,36			12,25		
Alg. iteracyjny		1,09	1,10		1,71	1,70		2,66	2,62		8,36	8,39
KKD	1,15			1,98			2,90			11,24		

Tab. 20. Porównanie wartości współczynników kompresji wektorów niejednorodnych

	Rodzaj źródła											
	≠	NO _a	NO _b	≠	N1 _a	N1 _b	≠	N2 _a	N2 _b	≠	N3 _a	N3 _b
K _{max}		2,409 - 2,582			3,149 - 3,449			4,037 - 4,604			4,532 - 5,299	
Przewidywane x ₁		0,091			0,068			0,055			0,05	
Rzeczywiste x ₁		0,091	0,090		0,067	0,067		0,054	0,052		0,049	0,047
Alg. A _n (m ₁ =4)		2,36	2,37		3,07	3,09		3,87	3,97		4,31	4,50
Alg. Huffmana (max.dł.runu d=100)		2,41	2,41		3,19	3,22		4,23	4,32		4,90	5,07
Alg. Huffmana (dł. fgm n=8)		2,42	2,42		3,05	3,06		3,64	3,68		3,92	3,98
Zmodyfikowany alg. Golomba (m _{max} = 300)		2,31	2,32		3,03	3,07		4,09	4,02		4,70	4,97
Alg. iteracyjny		2,08	2,08		2,74	2,79		3,69	3,77		4,31	4,49
WH1	2,20			2,70			3,42			4,36		
WH2	1,78			2,31			3,04			3,81		
KKD	1,99			2,23			2,25			2,86		

Uwaga: Dane z kolumn oznaczonych ≠ pochodzą z artykułu O. Nevelainen [31].

c.d. Tab. 19.

	Rodzaj źródła											
	⌘	B0 _a	B0 _b	⌘	B1 _a	B1 _b	⌘	B2 _a	B2 _b	⌘	B3 _a	B3 _b
Alg. Bradleya	1,15			2,01			3,20			11,59		
WH1M	0,94			1,69			2,69			9,80		
WH2	0,88			1,47			2,63			9,52		

Uwaga: Dane z kolumn oznaczonych ⌘ pochodzą z artykułu Nevelainena [31] .

Przedstawione wyżej dane pozwalają na wyodrębnienie grup algorytmów charakteryzujących się podobnymi własnościami:

1. Algorytmy Bradleya, KKD, Golomba i A_1 dostosowane są do kompresji wektorów bezpamięciowych. Optymalny dobór parametrów możliwy jest po ustaleniu własności wektora. Umożliwia to algorytmom osiągnięcie współczynników kompresji o wartościach zbliżonych do maksymalnych, możliwych teoretycznie do uzyskania. Największe skrócenie długości zapisu w tej grupie osiąga A_1 . Algorytmy te wykorzystują niewielką liczbę parametrów (zwykle jest to pojedyncza liczba) których zapamiętanie nie powoduje praktycznie zmiany współczynnika kompresji. Dużą zaletą algorytmów tej grupy jest ich prostota, natomiast podstawową wadą są znacznie gorsze niż w przypadku innych algorytmów współczynniki kompresji wektorów niejednorodnych.

2. Algorytmy WH1M, WH2 oraz algorytm iteracyjny są dostosowane do kompresji wektorów niejednorodnych. Algorytmy te zapamiętują w przetworzonym wektorze fragmenty wektora wejściowego zawierające jedyńki, co powoduje znaczną redundancję kodowania bezpamięciowych.

W wypadku algorytmu iteracyjnego nie jest stosowana wstępna analiza własności przetwarzanych danych. W trakcie przeprowadzonego eksperymentu algorytm ten osiągał maksymalne współczynniki kompresji przy blokach o długości 3 lub 4 bitów. Zwiększenie długości bloku powoduje przyspieszenie kodowania i dekodowania. Towarzyszy temu jednak dość znaczny spadek współczynnika kompresji (patrz tab. 8).

3. Algorytmy wykorzystujące kody Huffmana i zmodyfikowane kody Golomba osiągają wysokie współczynniki kompresji przy przetwarzaniu wszystkich rodzajów badanych wektorów, przy czym redundancja kodowania wektorów niejednorodnych jest znacznie

większa niż wektorów bezpamięciowych. Algorytmy te stosują słowa kodowe zmiennej długości. Osiągnięcie podanych współczynników kompresji wymaga uprzedniego ustalenia własności wektora i określenia wartości parametrów. W wypadku zmodyfikowanego algorytmu Golomba parametrami tymi są wartości $m_1, m_2, \dots, m_k$. Dla algorytmu Huffmana natomiast są nimi tablice przyporządkowujące słowa kodowe fragmentom wektora o stałej długości lub też długościom runów. Spośród tych dwu sposobów wykorzystania algorytmu Huffmana znacznie bardziej efektywny jest ten, który przydziela słowa kodowe długościom runów. Kody Huffmana przy kodowaniu fragmentów wektora zapewniają wysokie współczynniki kompresji jedynie wtedy, gdy liczba stosowanych słów kodowych wynosi co najmniej kilkaset. Algorytm ten cechuje znaczny wzrost redundancji towarzyszący wzrostowi x_0 . Przydzielanie kodów długościom runów pozwala na uzyskanie porównywalnych, a dla wektorów o $x_1 < 0,05$ - wyższych współczynników kompresji przy wykorzystaniu znacznie mniejszej liczby słów kodowych. Przechowywanie lub przekazywanie wektora wraz z optymalną dla niego listą słów kodowych powoduje zmniejszenie rzeczywiście osiąganego współczynnika kompresji. Alternatywne rozwiązanie polega na wykorzystaniu kilku standardowych kodów oraz przetwarzaniu wektora za pomocą tego kodu, który jest najlepiej dostosowany do jego własności. Niedokładne zwykle dostosowanie struktury kodu do własności wektora powoduje także i w tym wypadku spadek współczynnika kompresji. Algorytmy tej grupy stosujące słowa kodowe zmiennej długości są z reguły trudniejsze do oprogramowania i wolniejsze w działaniu od innych omawianych tu algorytmów. Spośród algorytmów tej grupy najwyżej ocenić trzeba zmodyfikowany algorytm Golomba. Zapewnia on osiągnięcie wysokich współczynników kom-

presji. Jednocześnie prosta struktura słów kodowych pozwala na bez porównania szybsze kodowanie i dekodowanie, niż ma to miejsce w przypadku kodów Huffmana.

4. Algorytm A_n , podobnie jak i algorytmy należące do grupy 4, przeznaczony jest do kompresji dowolnych wektorów binarnych stosowanych w systemach wyszukiwania informacji. Uzyskiwane przez ten algorytm wartości współczynnika kompresji są, jak wykazał to eksperyment, w większości przypadków mniejsze jedynie od tych osiąganych przez kody Huffmana. Algorytm A_n posiada ponadto korzystne cechy nie występujące w omawianych poprzednio algorytmach:

(a) Wyeliminowanie wstępnej analizy własności wektora.

W trakcie kompresji ustalane są jedynie własności wektora Y , tworzonego podczas kodowania, a ponadto posiadającego długość nie przekraczającą zwykle $1/3$ długości wektora wejściowego.

b) Posługiwanie się jedynie dwoma parametrami działania:

- wielkością grupy w drugiej fazie kompresji
- ciągiem binarnym umożliwiającym dekodowanie predykcyjne.

Dokumenty w systemach wyszukiwania informacji są zwykle zapisane nie w jednym ogólnym zbiorze dokumentów, lecz w podzbiorach zawierających dokumenty o zbliżonej tematyce i przeznaczonych do zapisania na tym samym pakiecie dyskowym. Podzbiory te liczą często nie więcej niż kilka tysięcy dokumentów, co powoduje że algorytmy wykorzystujące znaczną liczbę parametrów osiągają w praktyce współczynniki kompresji o wartościach mniejszych od tych ustalonych w trakcie eksperymentu. W przeprowadzonym eksperymencie nie uwzględniono bowiem wydłużenia zapisu skompresowanego wektora spowodowanego przez konieczność zapisania parametrów

- c) Pobieranie podczas kodowania wektora jego m elementowych pociągów, co pozwala na znaczne przyspieszenie kompresji i dekodowania.

PODSUMOWANIE

Stosowanie kompresji umożliwia zwiększenie efektywności działania systemów informatycznych, w większości przypadków pozwala ona na znaczne zredukowanie długości zapisu danych. Dzięki temu zmniejszają się koszty ich przechowywania i przekazywania. W literaturze polskiej znaleźć można niewiele prac poświęconych kompresji danych, zaś większość ze znanych z literatury obcojęzycznej algorytmów zamieszczona jest jedynie w artykułach opisujących te algorytmy. Z tego powodu autor pracy uznał za konieczne przedstawienie w dwu pierwszych rozdziałach przeglądu metod kompresji najczęściej wykorzystywanych w informatyce. Przegląd ten może być użyteczny dla projektantów systemów informatycznych przy wyborze algorytmu kompresji najlepiej dostosowanego do ich potrzeb.

Kompresja danych jest szczególnie pożądana przy przetwarzaniu struktur binarnych. Dane tego typu cechuje bowiem duża długość i redundancja zapisu. Struktury te są wykorzystywane najczęściej do reprezentowania cyfrowych zapisów obrazów i list adresów dokumentów w systemach wyszukiwania informacji. Wyznaczenie maksymalnych wartości współczynnika kompresji wymaga obliczenia entropii źródła generującego dane. Do wytwarzania wektorów binarnych o własnościach wektorów występujących w systemach wyszukiwania informacji stosowane są źródła bezpamięciowe i złożone. Entropię źródeł bezpamięciowych oblicza się w prosty sposób. Autor pracy nie znalazł w literaturze sposobu obliczania entropii źródeł złożonych i dlatego też w rozdziale 3 zaproponował i zweryfikował algorytm obliczania przedziału, w jakim mieści się jej wartość.

Doświadczenia jakie zebrano na Politechnice Wrocławskiej przy projektowaniu i eksploatacji systemów wyszukiwania informacji pozwoliły na wyznaczenie cech, jakimi powinien charakteryzować się algorytm kompresji wektorów binarnych, tzw. jednowymiarowych struktur binarnych stosowanych w tych systemach.

Są nimi: znaczne skracanie długości zapisu, ograniczenie lub wyeliminowanie badania własności wektora poprzedzającego kompresję, stosowanie niewielkiej liczby parametrów oraz prostota kodowania i dekodowania. Zaproponowany przez autora algorytm spełniać miał wszystkie powyższe wymagania. W rozdziale 4 zamieszczono opis proponowanego algorytmu oraz czterech jego wersji: A_1 , A_2 , A_w i A_n . Pierwsze trzy przeznaczone są do kompresji wektorów bezpamięciowych. Jedynie A_1 , będący najprostszą wersją algorytmu, wymaga poprzedzenia kompresji ustaleniem częstości występowania jedynek w kodowanym wektorze. Czwarta wersja algorytmu, A_n , kompresuje wektor w dwu przebiegach. Zastosowanie kodowania predykcyjnego wektora powstałego po pierwszym etapie kompresji umożliwia efektywne skracanie długości zapisu wektorów występujących w systemach wyszukiwania informacji. W rozdziale tym zamieszczono także analizę działania algorytmu. Uzyskane rezultaty porównano następnie z wynikami przeprowadzonego eksperymentu. Stwierdzono dużą zgodność przewidywanych i uzyskanych wartości współczynników kompresji.

Rozdział 5 zawiera opis przeprowadzonego eksperymentu. W jego ramach wygenerowano i przetestowano wektory binarne pochodzące ze źródeł bezpamięciowych i złożonych. Następnie zakodowano je za pomocą proponowanych wersji algorytmu, a także za pomocą, uznanych za najbardziej sprawne, kodów Huffmana i zmodyfikowanych kodów Golomba. Uzyskaną kompresję porównano również z maksymalnym, możliwym teoretycznie do osiągnięcia skróceniem

długości zapisu oraz ze współczynnikami kompresji osiąganymi przez inne znane z literatury algorytmy. Dla większości z badanych wektorów największe i niewiele odbiegające od siebie wartości współczynników kompresji uzyskiwane były przez algorytm A_n oraz kody Huffmana i Golomba. Do zalet algorytmu A_n , nie występujących jednocześnie i w przypadku pozostałych algorytmów kompresji zaliczyć należy:

- efektywne kompresowanie wszystkich typów wektorów binarnych występujących w systemach wyszukiwania informacji,
- pominięcie fazy analizy własności wektora,
- niewielką liczbę parametrów, jaką należy zapamiętać łącznie ze skompresowaną postacią wektora,
- niewielki stopień skomplikowania algorytmu.

Wadą algorytmu A_n jest dwuprzebiegowe przetwarzanie wektora. Znaczenie tej wady jest jednak zmniejszone przez to, że przetwarzany w drugiej fazie wektor ma długość z reguły znacznie mniejszą niż $1/3$ długości wejściowego wektora.

Algorytm A_n spełnia zatem stawiane mu wymagania. Pozwala on (dzięki swojej prostocie) na stosowanie go w systemach wyszukiwania informacji.

Wykorzystanie opisanego w pracy algorytmu do kompresji cyfrowych zapisów obrazów musi być poprzedzone, podobnie jak ma to miejsce w wypadku większości algorytmów kompresji wektorów binarnych, kodowaniem predykcyjnym danych. Znaczne wartości współczynnika kompresji umożliwiają zastosowanie algorytmu A do kompresji baz danych. Wymagane do tego jest opracowanie algorytmu wstępnego przetworzenia danych mającego na celu zróżnicowanie częstości występowania w nich zer i jedynek.

LITERATURA

- [1] Held G.: Data compression standards. A necessity now. W: Interface 82. Proceedings. New York: Mc Graw-Hill 1982 s. 129-136.
- [2] Brillouin L.: Nauka a teoria informacji. Warszawa: PWN 1969
- [3] Maximer V.: Some remarks on entrophy prediction of natural language texts. Information Storage and Retrieval 1971 vol. 7 s. 293-295.
- [4] Cover T.M., King R. C.: A convergent gambling estimate of the entrophy of English. IEEE Transactions on Information Theory 1978 vol. 24 s. 413-421.
- [5] Lyons N.R.: Format based data compression. Data Base 1983 vol. 15 nr 2 s. 15-18.
- [6] Reghbati H.K.: An overview of data compression techniques. Computer 1981 vol. 14 s. 71-75.
- [7] Chen T.Ch., Ho I.T.: Storage efficient representation of decimal data. Communications of the ACM, 1975 vol. 18 s. 49-52
- [8] Shaeffer D.H.: Logarithmic compression of binary numbers. Proceedings of the IRE 1961 vol. 49 s. 1219.
- [9] Berlekamp E.R.: The hackenbush number system for compression of numerical data. Information and Control 1974 vol. 26 s. 134-140.
- [10] Martin J.: Data compaction. W.: Computer. Data - Base Organization. Englewood Cliffs: Prentice - Hall 1977 s. 578-587.
- [11] Alsberg P.A.: Space and time savings through data base compression and dynamic reconstruing. Proceedings of the IEEE. 1975 vol. 63 s. 1114-1122.

- [12] Damerau F.J.: Markov models and linguistic theory. The Hague: Mouton 1971.
- [13] Huffman D.A.: A method for the construction of minimum redundancy codes. Proceedings of the IRE 1952 vol. 40 s. 1098-1101.
- [14] Severance D.G.: A practitioners guide to data base compression. Information Systems 1983 vol. 8 nr 1 s. 51-62
- [15] Pechura M.: File archival techniques using data compression. Communications of the ACM 1982 vol. 25 nr 9 s. 605-611.
- [16] Ziv J.: Coding of sources with unknown statistics. Part 1. Probability of encoding error. IEEE Transactions on Information Theory 1972 vol. 18 s. 384-394.
- [17] Dawiszon L.D.: Universal noiseless coding. IEEE Transactions on Information Theory 1973 vol. 19 s. 783-795.
- [18] Ziv J., Lempel A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory 1977 vol. 23 s. 337-343.
- [19] Ziv J., Lempel A.: Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory 1978 vol. 24 s. 530-536.
- [20] Siemiński A.: Kompresja baz danych za pomocą kodów zmiennej długości. Informatyka 1979 R. 14, nr 12 s. 36-39.
- [21] Connel J.B.: A Huffman-Shannon-Fano code. Proceedings of the IEEE 1973 vol. 61 s. 1046-1047.
- [22] Heaps H.S.: Storage analysis of a compression coding for document data base. Infor 1972 vol. 10 s. 47-61

- [23] Golenko P., Siemiński A.: Experiments in bibliographic file compression. W: Software for data bases in teleprocessing systems. Warszawa: PWN 1981 s. 143-151.
- [24] Schuegraf E.J.: Selection of equifrequent word fragments for information retrieval. Information Storage and Retrieval. 1973 vol. 9 s. 697-711.
- [25] Cooper D., Lynch M.F.: Compression of Wiswesser line notation using variety generation. Journal of Chemical Information and Computer Sciences 1979 vol. 19 s. 165-169.
- [26] Schuegraf E.J.: A comparison of algorithms for data base compression by use of common strigns as language elements. Information Storage and Retrieval 1974 vol. 10 s. 309-319
- [27] Lynch M.F.: Compression of bibliographic files using an adaptation of run-length coding. Information Storage and Retrieval 1973 vol. 9 s. 207-214.
- [28] Hahn B.: A new technique for compression and storage of data. Communications of the ACM 1974 vol. 17 s. 434-436.
- [29] Cannon T.M., Hunt B.R.: Image processing by computer. Scientific American 1981 vol. 245 s. 214-225.
- [30] King D.R.: The binary vector as the basis of an inverted index file. Journal of Library Automation 1974 vol. 71 s. 307-314.
- [31] Nevelainen O., Jacobson M., Berg R.: Compression of clustered inverted files. W: Mathematical foundations of computer sciences. Berlin: Springer 1978 s. 393-402.
Lecture Notes in Computer Science vol. 64.
- [32] Patent. USA, nr 4 044 347. Variable-length to fixed-length conversion of minimum redundancy codes. Twórca: Van Voorhis D.C. Opubl. 1977.

- [33] Patent USA, nr 3 883 847. Uniform decoding of minimum redundancy codes. Twórcza: Frank A.J. Opubl. 1975.
- [34] Hunter R., Robinson H.: Internationale digital facsimile standards. Proceedings of the IEEE 1980 vol. 68 s. 854-868.
- [35] Bahl L.R., Kobayashi H.: Image data compression by predictive coding. 2. Encoding algorithms. IBM Journal of Research and Development 1974 vol. 19 s. 172-179.
- [36] Golomb S.W.: Run-length encoding. IEEE Transactions on Information Theory 1966 vol. 12 s. 399-409.
- [37] Teuhola J.: A compression method for clustered bit vectors. Information Processing Letters 1978 vol. 7 s. 308-311
- [38] Wedekind H., Härder T.: Datenbanksysteme II. Mannheim: B.I.-Wissenschaftsverlag 1976
- [39] Bradley S.D.: Optimizing a scheme for run length encodings. Proceedings of the IEEE 1969 vol. 57 s. 108-109.
- [40] Kunt M., Johnsen O.: Block coding of graphics: a tutorial review. Proceedings of the IEEE 1980 vol. 68 s. 770-786.
- [41] Pratt W.K., Capitant P.J., Chen W.H., Hamilton E.R., Wallis R.H.: Combined symbol matching facsimile data compression. Proceedings of the IEEE 1980 vol. 68 s. 786-796
- [42] Nilsson N.J.: Learning machines. New York: McGraw-Hill 1965.
- [43] Bahl L.R., Kobayashi H.: Image data compression by predictive coding. 1. Prediction algorithms. IBM Journal of Research and Development 1974 vol. 19 s. 164-171.
- [44] Fu K.S.: Learning control systems - review and outlook. IEEE Transactions on Automatic Control 1970 vol. 15 s. 210.

- [45] Lubański M.: Filozoficzne zagadnienia teorii informacji. Warszawa: Akademia Teologii Katolickiej 1975.
- [46] Abramson N.: Teoria informacji i kodowania. Warszawa: PWN 1969.
- [47] Feller W.: Wstęp do rachunku prawdopodobieństwa. Warszawa: PWN 1977.
- [48] Rissanen J., Langdon G.G.: Arithmetic coding. IBM Journal of Research and Development 1979 vol. 23 s. 149-162.
- [49] Zieliński R.: Generatory liczb losowych. Programowanie i testowanie na maszynach cyfrowych. Warszawa: WNT 1979.
- [50] Waller R.A.: Statistics. An introduction to numerical reasoning. San Francisco: Holden-Day 1979.
- [51] Raff M.S.: On approximating the point binomial. The Journal of the American Statistical Association 1956 vol. 51 s. 293-303.

Z A Ł A C Z N I K I

Załącznik 1. Program w języku PASCAL obliczający zakres wartości entropii złożonych

```

PROGRAM ZLOZONE (INPUT,OUTPUT);
TYPE T01 = 0..1;
    WKT = ARRAY [1..10] OF T01;
    RZR = (A,B);
    TYP = 1..3;
VAR T:ARRAY[RZR,RZR] OF REAL;
    ZWM:WKT; PZR:ARRAY[RZR] OF REAL;
    PRB : ARRAY [RZR,T01] OF REAL;
    J,IZG,ITIM,DR,I,K,ZMR,MAXK : INTEGER;
    RF,R,ED,EG,RA,RB,RC,RD,RK : REAL;
    ZC1,ZC2:ALFA;
    TNB : PACKED ARRAY [0..1024] OF CHAR;

PROCEDURE USTAWWKT (I:INTEGER);
VAR J,IA : INTEGER;
BEGIN
    IA:=I; FOR J:=1 TO K DO BEGIN
        ZWMC[J]:=IA MOD 2; IA:=IA DIV 2
    END
END;

FUNCTION LR (R:REAL):REAL;
BEGIN
    LR:=-R*LN(R)
END;

FUNCTION P (WM:INTEGER; ZR:RZR; V:TYP):REAL;
BEGIN
    IF WM=1 THEN CASE V OF
        1:P:=PZR[ZR]*PRB[ZR,ZWMC[1]];
        2:P:=TCA,ZR]*PRB[ZR,ZWMC[1]];
        3:P:=TCB,ZR]*PRB[ZR,ZWMC[1]]
    END
    ELSE
        P:=P(WM-1,A,V)* TCA,ZR]*PRB[ZR,ZWMC[WM]]+
            P(WM-1,B,V)*TCB,ZR]*PRB[ZR,ZWMC[WM]]
    END;

PROCEDURE USTALSTART;
BEGIN
    READLN(TCA,B],TCB,A],PRB[A,0],PRB[B,0],IZG,DR);
    TCA,A]:=1-TCA,B]; TCB,B]:=1-TCB,A];
    PRB[A,1]:=1-PRB[A,0]; PRB[B,1]:=1-PRB[B,0];
    R:=TCA,B]+TCB,A]; MILL(ITIM);
    PZRA:=TCB,A]/R; PZREB:=TCA,B]/R;
    WRITELN(' ',TCA,B]:8:5,TCB,A]:8:5,
        PRB[B,0]:8:5,PRB[A,0]:8:5,
        PZRA]:8:5,PZREB]:8:5);
END;

```

```

      (* S T A R T   P R O G R A M U   *)
BEGIN
  USTALSTART;

  FOR K:=1 TO IZG DO BEGIN
    EG:=0; ED:=0; RK:=0; RF:=0;
    MAXK:=1; FOR I:=1 TO K DO
      MAXK:=MAXK*2;
    FOR I:=0 TO MAXK-1 DO BEGIN TNBCIJ:='A';
      USTAWWKT(I);
      RA:=P(K,A,1); RB:=P(K,B,1);
      IF RB>RA THEN BEGIN RF:=RF+RB; TNBCIJ:='B' END
        ELSE RF:=RF+RA;
      RK:=RK+RB+RA; EG:=EG+LR(RA+RB);
      RC:=P(K,A,2)+P(K,B,2); RD:=P(K,A,3)+P(K,B,3);
      ED:=ED+LR(RC)*PZRCAJ+LR(RD)*PZRB;
      IF DR=1 THEN BEGIN
        FOR J:=1 TO I DO WRITE(ZWMCJJ);
        WRITELN(' ',I,RA:8:5,RB:8:5,RK:8:5);
        WRITELN(' ',RC:8:5,RD:8:5,EG:8:5,ED:8:5)
          END;
        END;
      RK:=K*LN(2); EG:=EG/RK; ED:=ED/RK;
      DATE(ZC1); TIME(ZC2); MILL(I);
      WRITELN(ZC1,' ',ZC2,I-ITIM); ITIM:=I;
      WRITELN(' OGRANICZENIE GORNE I DOLNE ',EG:8:5,ED:8:5);
      IF DR=1 THEN BEGIN FOR I:=0 TO MAXK-1 DO
        WRITE(TNBCIJ);
        WRITELN END;
      WRITELN(K,RF:15:7); WRITELN; WRITELN; WRITELN;
      END;
  END;
END.

```

Załącznik 2. Program w języku PASCAL kodujący i dekodujący
wektory binarne za pomocą algorytmu A

```
PROGRAM KOMPRESJA (INPUT, OUTPUT);
CONST NN=1000;
TYPE ZJ=-1..1; WKT=ARRAY[0..NN] OF ZJ;
VAR X, Y, Z : WKT;
    E, IZ, B, M, I, J, K : INTEGER;
    R : ARRAY [0..100] OF ZJ;

FUNCTION WG (VAR W:WKT; J:INTEGER):INTEGER;
VAR E, II : INTEGER;
BEGIN
    II := 0; FOR E:=J TO J+M-1 DO
        II:=II*2+W[E]; WG:=II;
    END;

PROCEDURE PISZW (VAR W:WKT; DL:INTEGER; ZN:CHAR);
VAR I:INTEGER;
BEGIN
    WRITE (ZN:3,' '); FOR I:=0 TO DL-1 DO
        WRITE (W[I]:1); WRITELN;
    END;

PROCEDURE KODUJGRUPEX;
VAR IX,E:INTEGER;
BEGIN
    IX:=WG(X,K);
    IF IX=0 THEN BEGIN
        YC[I]:=0; I:=I+1 END
    ELSE BEGIN
        IF IZ>IX THEN BEGIN YC[I]:=0; I:=I+1 END;
        REPEAT FOR E:=0 TO M-1 DO R[E]:=0; E:=K-1;
        REPEAT E:=E+1 UNTIL X[E]=1; (* USTALENIE JED *)
        X[E]:=0; R[E MOD M J]:=1; IZ:=0;
        FOR E:=0 TO M-1 DO BEGIN ZC[J]:=R[E];
            J:=J+1; IZ:=2*IZ+R[E] END;
        YC[I]:=1; I:=I+1; IX:=WG(X,K)
        UNTIL IX=0
    END
END;

PROCEDURE DEKODUJGRX;
VAR E : INTEGER;
BEGIN
    IF YC[I]=0 THEN BEGIN
        FOR K:=K TO K+M-1 DO XCK:=0;
            I:=I+1 END
        ELSE BEGIN I:=I+1;
            FOR E:=0 TO M-1 DO R[E]:=ZC[J+E]; (* ZAP R *)
            WHILE (WG(Z,J)>WG(Z,J+M)) AND (YC[I]=1) DO BEGIN
                FOR E :=0 TO M-1 DO R[E]:=R[E]+ZC[J+E+M]; I:=I+1;
                J:=J+M END;
        END;
```

```
IF (WG(Z,J)>WG(Z,J+M)) THEN I:=I+1;
E:=0; FOR K:=K TO K+M-1 DO BEGIN
XCKJ:=RCEJ; E:=E+1; END;
J:=J+M
END
END;

BEGIN
(* WCZYTANIE ROZMIARU GRUPY I WEKTORA X *)
READ(M); K:=0;
REPEAT READ(XCKJ); K:=K+1 UNTIL XCK-1<0;
K:=K-1; READLN; B:=K MOD M;
(* UZUPELNIENIE ZERAMI OSTATNIEJ GRUPY X *)
IF B<>0 THEN FOR B:=1 TO M-B DO BEGIN
XCKJ:=0; K:=K+1 END;
WRITELN(' M I DK ', M,K);
B:=K; PISZW(X,K,'X');
J:=0; I:=0; IZ:=1; K:=0; (* UST. WARUNKOW POCZ *)
WHILE K<>B DO BEGIN KODUJGRUPEX; K:=K+M END;
PISZW(Y,I,'Y'); PISZW(Z,J,'Z');
FOR E:=0 TO B DO XCEJ:=0;
FOR E:=J TO J+M-1 DO ZCJJ:=1; (* DOPISANIE JED *)
J:=0; K:=0; B:=I; I:=0;
REPEAT DEKODUJGRX UNTIL I=B;
WRITELN(' ODTWORZONY WEKTOR X ');
PISZW(X,K,'X');
END.
```

Załącznik 3. Program w języku FORTRAN generujący wektory binarne oraz kompresujący je za pomocą algorytmu A, kodów Huffmana, uogólnionych kodów Golomba i algorytmu iteracyjnego

LIST

```

      PROGRAM (PRG1)
      COMPRESS INTEGER AND LOGICAL
      INPUT 1 = CRO
      OUTPUT 2 = LPO
      TRACE 1
      END

MASTER STEROWANIE
CALL GLOWNY
END
SUBROUTINE GLOWNY
LOGICAL DRUK,DLOG
INTEGER OUT,DLX,DLY,NOWY,STARY
INTEGER TX(1000)
INTEGER TDH(30)
REAL NAG(20), START(10)
COMMON /DH2/ TDH
COMMON /DHASH/ ST,RMN,WYN
COMMON /P1/R1/SM/ISM/DRL/DLOG
COMMON /TABKODX/TX
DATA NAG(1) /8HGENERATO/
DATA NAG(2) /8HUSTAW /
DATA NAG(3) /8HPOJEDYNC/
DATA NAG(4) /24HPODWOJNYGLOBALNYMULTIGOL/
DATA NAG(7) /24HOPT-POJEOPT-PODWKONIEC /
DATA NAG(10) / 8HZMN GENE/
DATA NAG(11) /8HALG SKAT/
DATA NAG(12) /8HHUFFMAN /
DATA NAG(13) /8HKDL-HUFF/
DATA NAG(14) /16HST-MULTIST-HUFFM/
DATA NAG(16) /8HALG. A44/
RMN=268.4353301
WYN=0.389056098698347
ST=WYN
ISM=31
READ (1,1015) (START(IQ),IQ=1,10)
1015 FORMAT (10A8)
101 READ (1, 1001) R1, R4, IN, IA
1001 FORMAT (2A8, 2I0)
      CALL DATE(R3)
      CALL TIME(R2)
      WRITE (2, 1000) R1, R4, R2, R3
1000 FORMAT (/30X,2A8,3A20/)
      WRITE (2,1011)
1011 FORMAT (/5X,10H*****)
      WRITE (2,1015) (START(IQ),IQ=1,10)
      DRUK=.FALSE.
      IF(IA.EQ.1)DRUK=.TRUE.
      DLOG=DRUK
      DO 102 I=1,20

```

```

      CALL COMP8 (R1,NAG(I),K)
      IF (K.EQ.1) IA=I
102   CONTINUE
      GO TO (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16), IA
C     *** GENERATOR ***
1     CALL GENER(IN,STARY)
      IF (DRUK) CALL WYDRUK(IN,1)
      GOTO 101
C     *** USTAWIANIE TABLICY ***
2     READ (1,1002) IP
      IPOPA=IN
      WRITE (2, 1019) IP
1019  FORMAT (I20)
      IF (IP .GT. 9) GOTO 25
      IK=IP
      CALL LICZROZ (IP,IN)
      IP=2**IP
      GOTO 101
25    CALL MULTUS (IN,IP)
      IK=0
      GOTO 101
C     *** POJEDYNCZY ***
3     READ (1,1002) IP, IK, IS
      WRITE (2,1006) IP,IK,IS
1006  FORMAT (/30X,11HALGORYTM A1,3I5/)
32    DO 35 I=IP,IK,IS
      J=1
      CALL A1 (IN,OUT,I,DLX,DLY,J,STARY)
      IF (.NOT.DRUK) GO TO 33
      CALL DRUKX(DLX)
      CALL WYDRUK(DLY,J)
33    CALL DRK (IN,OUT)
35    CONTINUE
      GO TO 101
C     *** PODWOJNY ZM ***
4     READ (1,1002) IP,IK,IS
      WRITE (2,1013) IP,IK,IS
1013  FORMAT (/30X,11HALGORYTM A2,3I5)
      DO 45 I=IP,IK,IS
      J=1
      CALL A1 (IN,OUT,I,DLX,DLY,J,STARY)
      IA=DLX
      IB=DLY
      CALL MFIX (IP,R1)
      CALL A1 (IB,OUT,IP,DLX,DLY,J,STARY)
      OUT=OUT+IA
      CALL DRK (IN,OUT)
45    CONTINUE
      GOTO 101
C     *** ZM. DL. SLOWA ***
5     READ (1,1002) ISM
      GO TO 101
C     *** MULTIGOLOMBE ***
C     *** MAX. DL RUNU, DL DO BAD. ***

```

```
6      READ (1,1002) I
      WRITE (2,1012) I
1012   FORMAT (//30X,11HMULTIGOLOMB,2I7)
      CALL MULT (I)
      IF (IN .NE. IPOPA) CALL MULTUS (IN,I)
      CALL MULTWYN (IP,OUT,I)
      CALL DRK (IP,OUT)
      GO TO 101

C          *** KONIEC ***
9      WRITE (2,1005)
1005   FORMAT (/5X,16HKONIEC PRZEBIEGU)
      PAUSE
      GOTO 101

C          *** OPTYMALIZACJA 1 **
7      CALL MFIX (IP,R1)
      IS=1
      IK=IP
      GOTO 32

C          ** OPTYMALIZACJA 2 ***
C          *** ZM M1, DLBAD ***
8      READ (1,1002) IP,IK,IS,M
1002   FORMAT (5X,16I0)
      WRITE (2,1007) IP,IK,IS,M
1007   FORMAT (//30X,25HALGORYTM A2 - OPT. 2 KOD.,4I5)
      DO 85 I=IP,IK,IS
      J=1
          CALL A1 (IN,OUT,I,DLX,DLY,J,STARY)
          IA=DLX
          CALL REKOD (J,STARY,DLY,M)
          CALL MFIX (IP,R1)
          IF (DRUK) CALL WYDRUK (DLY,J)
          IB=DLY
          IF (IP .EQ. 1 ) IP=2
          CALL A1 (IB,OUT,IP,DLX,DLY,J,STARY)
          OUT=OUT+IA
          CALL DRK (IN,OUT)
85          CONTINUE
77      GOTO 101
```



```

      ***      ZMIANA  GENERATORA      ***
10   CONTINUE
150  READ(1,1003) ST,RMN
1003  FORMAT(5X,2F0.0)
      WRITE(2,1004) ST,RMN

1004  FORMAT(/,10X,27HNOWE PARAMETRY GENERATORA :,2F15.12)
      WYN=ST
      IF (RMN .GE. 0 ) GOTO 101
      IA=-RMN
      READ (1,1002) (TDH(IP),IP=1,IA)
      GOTO 101
C      ***      SKAT  IP,IK,IKROK      ***
11   READ (1,1002) IP,IK,IS
      WRITE (2,1008) IP,IK,IS
1008  FORMAT (//23X,13HALGORYTM SKAT,3I5)
      DO 1100 I=IP,IK,IS
          CALL SKAT (IN,OUT,I,STARY)
          WRITE (2,1009) I
1009  FORMAT (35X,11HDL. GRUPY =,I2)
          CALL DRK (IN,OUT)
1100  CONTINUE
      GOTO 101
C      ***      HUFFMAN      ***
C      ** ZMIANY, DL DO BAD. ***
12   READ (1,1002) I
      WRITE (2,1010) I
1010  FORMAT (//30X,13HKODY HUFFMANA,4I6)
      I=2**I
      GOTO 1301
C      ***      KDL-HUFFMAN      ***
C      *** MAX DL RUNU, DL BAD ***
13   READ (1,1002) I
      WRITE (2,1014) I
1014  FORMAT(//30X,22HHUFFMAN DLA RUN-LENGTH,2I7)
1301  CALL HUFFMAN (I)
      CALL WYNHUFF(I,OUT)
      CALL DRK(IN,OUT)
      GOTO 101
C      ***      STABILNOSC DLA MULTIGOLOMB **
14   CALL MULT (IP)
1401  CALL MULTWYN (I,OUT,IP)
      CALL DRK (I,OUT)
      READ (1,1002) M
      IF (M.EQ.0) GOTO 101
      CALL GENER (IN,STARY)
      CALL MULTUS (IN,IP)
      GOTO 1401
C      ***      STABILNOSC DLA HUFFMANA ***
15   CALL HUFFMAN (IP)
1501  CALL WYNHUFF (IP,OUT)
      CALL DRK (IN,OUT)
      READ (1, 1002) M

```

```

      IF (M.EQ.0) GOTO 101
      CALL GENER (IN, STARY)
      IF (IK.EQ.0) GOTO 1502
      CALL LICZROZ (IK, IN)
      GOTO 1501
1502  CALL MULTUS (IN, IP)
      GOTO 1501
C      ***  ALGORYTM  A44  ***
16    READ (1,1002) I, IP, M
      WRITE (2,1020) I, IP, M
1020  FORMAT (//,30X,12HALGORYTM A44,3I5)
      J=1
      CALL A1 (IN, OUT, I, DLX, DLY, J, STARY)
      IDWM=2**M
      READ (1,1002) (TX(JX), JX=1, IDWM)
      WRITE (2,1021) (TX(JX), JX=1, IDWM)
1021  FORMAT (/,10X,15HPAMIEC ALGORYTU,32I2)
      IA=DLX
      CALL REKODA (J, STARY, DLY, M)
      IB=DLY
      CALL A1 (IB, OUT, IP, DLX, DLY, J, STARY)
      OUT=OUT+IA
      CALL DRK (IN, OUT)
      GOTO 101
      END

```

```

      SUBROUTINE REKOD (ST, NW, DY, M)
      INTEGER X(1024), ST, NW, DY, G
      COMMON /TABKODX/X/BIT/G/P1/R1
      CALL SRT (ST, NW)
      CALL USTAW (ST, DY, M)
      IW=0
      IC=0
      MAX=2**M
      DO 100 I=1, DY
          CALL BBIT
          IB=G
          IA=X(IW+1)
          G=0
          IF (IA .NE. IB) G=1
          IC=IC+G
          IW=IW*2+IB
          IW=MOD(IW, MAX)
          CALL UBIT
100    CONTINUE
      R1=FLOAT(IC)/DY
      WRITE (2,1000) R1
1000  FORMAT (20X,16HREKODOWANIE P1 =,F7.5)
      ST=NW
      NW=NW+1+DY/20
      RETURN
      END

```

SUBROUTINE REKODA(ST, NW, DY, M)

```

INTEGER X(1024),ST,NW,DY,G
COMMON /TABKODX/X/BIT/G/P1/R1
CALL SRT (ST,NW)
IW=0
IC=0
MAX=2**M
DO 100 I=1,DY
    CALL BBIT
    IB=G
    IA=X(IW+1)
    G=0
    IF (IA .NE. IB) G=1
    IC=IC+G
    IW=IW*2+IB
    IW=MOD(IW,MAX)
    CALL UBIT
100    CONTINUE
R1=FLOAT(IC)/DY
WRITE (2,1000) R1
1000 FORMAT (20X,16HREKODOWANIE P1 =,F7.5)
ST=NW
NW=NW+1+DY/20
RETURN
END

SUBROUTINE A1 (IN,OUT,M,DX,DY,ST,NW)
INTEGER OUT,DX,DY,ST,NW,G
COMMON /BIT/G/P1/R1/SM/ISM
RHA(PR)=-PR*ALOG2(PR)-(1-PR)*ALOG2(1-PR)
CALL SRT (ST,NW)
    DY=0
    CALL USPX (M,ISM,DX)

I1=0
MR=1
MB=1
R=0
DO 200 I=1,IN
    CALL BBIT
    R=R+G
    IF (G .NE. 1) GOTO 120
    CALL ZAPX (MB,DX)
    CALL UBIT
    MR=MB
    DY=DY+1
    I1=I1+1
    GOTO 150
120    IF (MB .NE. MR) GOTO 150
    CALL UBIT
    DY=DY+1
150    MB=MB+1
    IF (MB .GT. M ) MB=1
200    CONTINUE

R=R/IN
IF (R .EQ.0 .OR. R .EQ. 1) R1=0.99
R1=R

```

```

R=1/RHA(R)
WRITE (2,1000) IN,M ,R ,R1
1000 FORMAT (//,10X,29HKOMPRESJA WEKTORA O DŁUGOSCI ,
*15,9H ( M =,12,8H, KMAX =,F7.4,2H ),
*6X,6HP1WE =,F7.4)
R1=FLOAT(I1)/DY
IF (R .EQ.0 .OR. R .EQ. 1) R1=0.99
R=1/RHA(R1)
OUT=DX+DY
RA=FLOAT(IN)/OUT
ST=NW
NW=NW+1+DY/20
WRITE(2,1001) RA,R,DX,DY,R1
1001 FORMAT (15X,2HK=,F7.4,3H (,F7.4,2H ),7X,
*7HDL. X =,15,7X,7HDL. Y =,15,
*6X,6HP1WY =,F7.4)
RETURN
END
SUBROUTINE WYDRUK (IDW,IG)
REAL DR(13),KOD(3)
COMMON /BIT/IBIT
DATA KOD(1)/9H0 1/,KOD(3)/8H
WRITE (2,1000) IDW
1000 FORMAT (///20X,24HWYDRUK WEKTORA BINARNEGO, 2H ,
*12H0 DŁUGOSCI ,15//)
CALL SRT(IG,1)
CALL COPY8(DR(13),KOD(3))
IZW=0
ILC=0+1
R=IDW/100.0
IF (R .NE. IDW/100) R=R+1
ID=INT(R)*100
DO 200 I=1,ID
    IZN=MOD(IZW,8)+1
    ISL=IZW/8+1
    CALL BBIT
    IF (ILC.GT.IDW) IBIT=2
    CALL COPY(1,DR(ISL),IZN,KOD(IBIT+1),1)
    IZW=IZW+1
    IF (IZW.LT.100)GO TO 200
    WRITE (2,1001)(DR(J),J=1,13)
1001 FORMAT (10X,13A8)
    IZW=0
200 ILC=ILC+1
RETURN
END
SUBROUTINE GENER (IDW,IX)
REAL TM(8)
INTEGER G
COMMON /DHASH/ST,RMN,WYN/BIT/G
COMMON /P1/A
WRITE (2,1000)IDW
1000 FORMAT(//20X,20HGENEROWANIE WEKTORA ,
*21HBINARNEGO O DŁUGOSCI-,16,

```

```

*6H BITOW/27X,27HRODZAJ STOSOWANEGO ZRODLA :)
  CALL SRT(1,1)
  WYN=ST
  I1=0
  READ (1,1001) I
1001  FORMAT(5X,I0)
      GOTO (100,200,300),I
100   READ (1,1002) A
1002  FORMAT(5X,8F0.0)
      WRITE(2,1003) A
1003  FORMAT(32X,19HBEZPAMIECIOWE, P1 =,F8.5)
      DO 150 I=1,IDW
          CALL HASH
          G=0
          IF(WYN.LE.A) G=1
          CALL UBIT
150   I1=I1+G
      GO TO 500
200   READ (1,1001) IP
      IP=2*IP
      READ(1,1002) (TM(I),I=1,IP)
      IPOF=0
      DO 250 I=1,IDW
          CALL HASH
          G=0
          IF(WYN.LE.TM(IPOF+1)) G=1
          CALL UBIT
          I1=I1+G
          IPOF=2*IPOF+G
          IF (IPOF.GE.IP) IPOF=IPOF-IP
250   CONTINUE
      WRITE(2,1004) (TM(I),I=1,IP)
1004  FORMAT(32X,9HMARKOWA -,8F10.7)
      GO TO 500
300   READ(1,1002) (TM(I),I=1,4)
      WRITE(2,1005) (TM(I),I=1,4)
1005  FORMAT(32X,13HZLOZONE TAB ,F7.4,4X,
*5HTBA =,F7.4,4X,5HPA1 =,F7.4,4X,
*5HPB1 =,F7.4)
      IPOF=1
      DO 350 I=1,IDW
          CALL HASH
          IF(WYN.LE.TM(IPOF)) IPOF=IPOF+1
          IF (IPOF.GT.2) IPOF=1
          G=0
          CALL HASH
          IF(WYN.LE.TM(2+IPOF)) G=1
          CALL UBIT
350   I1=I1+G
500   A=I1/FLGAT(IDW)
      WRITE(2,1006) A
1006  FORMAT(/25X,4HP1 =,F7.5)
      IX=IDW/20+2
      RETURN

```

```

END
      SUBROUTINE DRK(IWE,IWY)
      R=IWE/FLOAT(IWY)
      WRITE(2,1000) IWE,IWY
1000  FORMAT(/20X,19HDLUGOSC WEKTORA S =,I5,5X,
      *41HSUMARYCZNA DLUGOSC WEKTORA PO KOMPRESJI =,
      *I5/)
      WRITE(2,1001) R
      WRITE(2,1001) R
      WRITE(2,1001) R
      WRITE(2,1001) R
1001  FORMAT(1H+,30X,16HWSP. KOMPRESJI =,F10.5)
      RETURN
      END

      SUBROUTINE CZYTX(IW)
      LOGICAL JEST
      INTEGER WGR,DLSM,WDB,WSK,D,Z,WG,DLX
      INTEGER X(1024)
      COMMON /TABKODX/X
      COMMON /PARKODX/JEST,WMAX,WDB,WSK,ILG,ILB,WG,IGB
      IF(.NOT.JEST) RETURN
      IF(IGB.LT.ILG) GO TO 100
      IGB=0
      WSK=WSK+1
      WDB=IFIX(WMAX)/WG
100  IW=X(WSK/WDB)
      X(WSK)=MOD(X(WSK),WDB)
      WDB=WDB/WG
      IGB=IGB+1
      RETURN
      END

      SUBROUTINE DRUKX
      LOGICAL JEST
      INTEGER WGR,DLSM,WDB,WSK,D,Z,WG,DLX
      INTEGER X(1024)
      COMMON /TABKODX/X

      COMMON /PARKODX/JEST,WMAX,WDB,WSK,ILG,ILB,WG,IGB
      WRITE(2,1000) ILG,ILB,(X(J),J=1,WSK)
1000  FORMAT(/20X,29HWYDRUK ZAKODOWANEGO WEKTORA X
      *,5H      (,I2,1H,,I2,2H )/100(10I10))
      RETURN
      END

      SUBROUTINE USTAW (SK,IDL,M)
      INTEGER X(1024),G,SK
      LOGICAL DLOG
      COMMON /TABKODX/X,BIT/G,DRL/DLOG
      CALL LICZSTYK (SK,IDL,M)
      POWY=0
      MAX=2**(M+1)
      WRITE (2,1000) (X(I),I=1,MAX)
      J=1
1000  FORMAT (32(16I5/))

```

```

DO 100 I=1,MAX,2
    IW=X(I)
    G=0
    IF (IW .GE. X(I+1)) GOTO 50
    G=1
    IW=X(I+1)
50    X(J)=G
    POWY=POWY+FLOAT(IW)/IDL
100    J=J+1
    WRITE (2,1001)POWY
1001    FORMAT (10X,4HP0 =,F7.5)
    WRITE (2,1000) (X(I),I=1,MAX/2)
    RETURN
END

SUBROUTINE LICZSTYK (SK,IDL,M)
    INTEGER X(1024),G,SK
    LOGICAL DLOG
    COMMON /TABKODX/X/BIT/G/DRL/DLOG
    CALL SRT (SK,1)
    MAX=2*(M+1)
DO 100 I=1,MAX
    100    X(I)=0
    IW=0
    DO 200 I=1,IDL
        CALL BBIT
        IW=IW*2+G
        IW=MOD(IW,MAX)
    200    X(IW+1)=X(IW+1)+1
    RETURN
END

SUBROUTINE WYNHUFF (I,OUT)
    INTEGER LICZ (1024),ROB (1024), WSK (1024)
    INTEGER SP (4000)
    COMMON /TABKODX/LICZ/WKT/SP,ROB,WSK
    LOGICAL DRUK
    INTEGER OUT,H
    COMMON /P1/R1/DRL/DRUK
    RC=0
    RD=0
    DO 600 H=1,I
        IF (LICZ(H) .EQ. 0) GOTO 15
        IF (DRUK) WRITE (2,10) LICZ(H),ROB(H)
10    FORMAT (15,3H: !,I3)
15    CONTINUE
        RC=RC+LICZ(H)
    600    RD=RD+LICZ(H)*ROB(H)
    OUT=RD
    RD=RD/RC
    WRITE (2,11) RD,OUT
11    FORMAT (10X,14HRS. DL. KODU =,F8.5,I10)
    RETURN
END

SUBROUTINE MFIX (IF,R1)
    INTEGER TM(49)

```

```

DATA TM/2*4,8*5,6*6,4*7,8*8,4*10,2*12,3*13,
*4*16,2*19,21,24,24,32,32,38/
RO=1-R1
IF (RO .GT. 0.76) GOTO 3
IP=2
RETURN
1 IF (RO .GT. 0.815) GOTO 4
IP=3
RETURN
4 IF (RO .GT. 0.865) GOTO 5
IP=4
RETURN
5 IF (RO .NE. 1.0) GOTO 51
IP=100
RETURN
51 IF (RO .LT. 0.98) GOTO 52
IP=ALOG2(0.5)/ALOG2(RO)+1
RETURN
52 I=(RO-0.86)*400
IP=TM(I)
RETURN
END
SUBROUTINE SRT (SKAD,GDZIE)
INTEGER ROB(10000)
INTEGER H,D,F,C,J,G
INTEGER SKAD,GDZIE
COMMON /PROCBIT/H,D,F,C,J,L,R/BIT/G
COMMON /WKT/ROB
J=SKAD
C=GDZIE
D=1
F=0
R=ROB(J)
L=0
H=1048576
RETURN
END
SUBROUTINE UBIT
INTEGER ROB(10000)
INTEGER H,D,F,C,J,G
COMMON /PROCBIT/H,D,F,C,J,L,R/BIT/G
COMMON /WKT/ROB
F=F+G*D
D=D*2
ROB(C)=F
IF (D.NE.H) RETURN
D=1
F=0
C=C+1
RETURN
END
SUBROUTINE BRIT
INTEGER ROB(10000)
INTEGER H,D,F,C,J,G

```



```

COMMON /PROCBIT/H,D,F,C,J,L,R/BIT/G
COMMON /WKT/ROB
R=R/2
G=1
K=R
IF (K .EQ. R) G=0
L=L+1
R=K
IF (L.LT.20) RETURN
J=J+1
L=0
R=ROB(J)
RETURN
END

      SUBROUTINE ZAPX(IW,IDLX)
LOGICAL JEST
INTEGER WGR,DLSM,WDB,WSK,D,Z,WG,DLX
INTEGER X(1024)
COMMON /PARKODX/JEST,WMAX,WDB,WSK,ILG,ILB,WG,IGB
COMMON /TABKODX/X
IF (JEST) GO TO 200
IF (IGB.LT.ILG) GO TO 100
IGB=0
IDLX=IDLX+ILB
WDB=WDB/WG
100  IGB=IGB+1
      RETURN
200  IF (IGB.LT.ILG) GOTO 300
      WSK=WSK+1
      X(WSK)=0
      IGB=0
      IDLX=IDLX+ILB
      WDB=IFIX(WMAX)/WG
300  X(WSK)=X(WSK)+WDB*(IW-1)
      IGB=IGB+1
      WDB=WDB/WG
      RETURN
      END

SUBROUTINE HASH
INTEGER TDH (30)
COMMON /DH2/TDH
COMMON /DHASH/ST,RMN,WYN
IF (RMN .LE. 0) GOTO 100
WYN=WYN*RMN
WYN=WYN-AINT(WYN)
RETURN
100  IF (RMN .EQ. 0) GOTO 200
      IK=-RMN
      CALL FPACRV (IK,TDH(1))
      WYN = TDH(IK)/FLOAT(8388607)
      RETURN
200  WYN = FPMCRV(WYN)
      RETURN
      END

```

```

SUBROUTINE SKAT (IN,OUT,M,NW)
  INTEGER OUT,DWE,DX,DY,DWY,ST
  ST=1
  OUT=0
  DWE=IN
100  CALL ROBSKAT (DWE,DWY,DX,DY,M,ST,NW)
  IF (DWY .GE. DWE) GOTO 200
  DWE=DY
  OUT=OUT+DX
  GOTO 100
200  OUT=OUT+DWE
  RETURN
  END
SUBROUTINE ROBSKAT (IN,OUT,DX,DY,M,ST,NW)
  INTEGER OUT,DX,DY,ST,G
  COMMON /BIT/G
  CALL SRT (ST,NW)
  I=IN/M
  IF (I .NE. IN/FLOAT(M)) I=I+1
  DX=0
  DY=0
  DO 100 J=1,I
    ID=0
    DO 50 IC=1,M
      CALL BBIT
50      ID=ID+G
      G=0
      IF (ID .EQ. 0) GOTO 70
      DX=DX+M
      G=1
70      DY=DY+1
      CALL UBIT
100  CONTINUE
  OUT=DX+DY
  ST=NW
  NW=NW+1+DY/20
  RETURN
  END
SUBROUTINE HUFFMAN (I)
  INTEGER D,H,C,E,F,A,G
  INTEGER LICZ (1024),ROB (1024), WSK (1024)
  INTEGER SP (4000) , MEM(1024)
  COMMON /TARKODX/LICZ/WKT/SP-ROB,WSK,MEM
  DO 100 H=1,I
100    LICZ(H)=LICZ(H)*I+H-1
  DO 200 H=1,I
    D=0
    C=LICZ(H)
    DO 250 A=1,I
250      IF (C .LE. LICZ(A)) D=D+1
    ROB(D)=(C+1-H)/I
200    MEM(H)=D
    C=1
    H=I

```

```

400   E=ROB(H)+ROB(H-1)
      F=H-2
430   IF (E .LT. ROB(F)) GOTO 450
      ROB(F+1)=ROB(F)
      ROB(F)=E
      F=F-1
      IF (F .GE. 1) GOTO 430
450   WSK(C)=F+1
      C=C+1
      H=H-1
      IF (H .GE. 3) GOTO 400
      H=2
      ROB(1)=1
      ROB(2)=1
      A=C-1
500   G=WSK(A)
      E=ROB(G)+1
      DO 510 F=G,H-1
510   ROB(F)=ROB(F+1)
      ROB(H)=E
      H=H+1
      ROB(H)=E
      A=A-1
      IF (A .GE. 1) GOTO 500
      DO 600 H=1,I
600   WSK(H)=ROB(H)
      DO 700 H=1,I
      D=MEM(H)
      LICZ(H)=(LICZ(H)+1-H)/I
700   ROB(H)=WSK(D)

      RETURN
      END
      SUBROUTINE MULT (J)
      LOGICAL DR
      INTEGER SP(4000), DLRUN(1000), DLK(1001), SI(1001)
      REAL S(1001)
      COMMON /TABKODX/SI
      COMMON /WKT/SP,DLRUN,DLK,S
      COMMON /DRL/DR
      SUMA = 0
      DO 10 I=1,J
10    SUMA = SUMA+SI(I)
      R = 0
      DO 15 I=1,J
      RC=1-R/SUMA
      R=R+SI(I)
      RB = SI(I)/SUMA
15    S(I)=RC
      GRN=J
      I=0
      I2=1
300   IDL=1
      RI2=0.5**I2
310   J=I+2**IDL

```

```

      IF (J .GE. GRN) GOTO 350
      IF (S(J) .LT. RI2) GOTO 315
      R1=S(J)-RI2
      IDL=IDL+1
      GOTO 310
315    R2=RI2-S(J)
      IF (R1 .LT. R2) IDL=IDL-1
      DLK(I2)=IDL
      I=I+2**IDL
      DLRUN(I2)=I
      I2=I2+1
      GOTO 300
350    DLK(I2)=IDL
      DLRUN(I2)=J
      DLRUN(I2+1)=10000
      IF (DR) WRITE(2,104)
104    FORMAT(15HDL. KODU I RUNU)
      DO 380 I=1,I2
380    DLK(I)=DLK(I)+I
      IF (DR) WRITE (2,103) (DLK(I),DLRUN(I),I=1,I2)
103    FORMAT (10(2I4,3H *))
      RETURN
      END
      SUBROUTINE MULTUS (DLW,MXR)
      INTEGER X(1000),G,DLW,MXR
      LOGICAL DRUK
      COMMON /TARKODX/X/BIT/G/DRL/DRUK
      DO 100 I=1,MXR
100    X(I)=0
      CALL SRT (1,1)
      J=0
      DO 200 I=1,DLW
      J=J+1
      CALL BBIT
      IF (G .EQ. 0) GOTO 200
120    IF (J .LE. MXR) GOTO 150
      X(MXR)=X(MXR)+1
      J=J-MXR
      GOTO 120
150    X(J)=X(J)+1
      J=0
200    CONTINUE
      IF (J .EQ. 0) GOTO 500
250    IF (J .LE. MXR) GOTO 300
      X(MXR)=X(MXR)+1
      J=J-MXR
      GOTO 250
300    X(J)=X(J)+1
500    CONTINUE
      IF (DRUK) WRITE (2,1000) (X(I),I=1,MXR)
1000  FORMAT (/9H DL RUNOW/10(10I10/))
      RETURN
      END
      SUBROUTINE MULTWYN (IN,OUT,J)

```

```

INTEGER IN,OUT,J
LOGICAL DR
INTEGER SP(4000),DLRUN(1000),DLK(1001),SI(1001)
REAL S(1001)
COMMON /TABKODX/SI
COMMON /WKT/SP,DLRUN,DLK,S
IN=0
OUT=0
DO 100 I=1,J
100   IN=IN+SI(I)*I
   ISC=0
   IA=1
   IW=1
150   DO 200 I=IA,DLRUN(IW)
       IF (I .GT. J) GOTO 200
       ISC=ISC+SI(I)
200   CONTINUE
       OUT=OUT+ISC*DLK(IW)
       ISC=0
300   IA=DLRUN(IW)+1
       IF (IA .GE. 999) RETURN
       IW=IW+1
GOTO 150
END
INTEGER X(1000),G,DLW,MXR
LOGICAL DRUK
COMMON /TABKODX/X/BIT/G/DRL/DRUK
CALL SRT(1,1)
DO 100 I=1,2**MXR
100   X(I)=0
   IW=0
   J=0
   DO 200 I=1,DLW
       J=J+1
       CALL BBIT
       IW=IW*2+G
       IF (J .NE. MXR) GOTO 150
       X(IW+1)=X(IW+1)+1
       IW=0
       J=0
150   CONTINUE
200   CONTINUE
RETURN
END
LIBRARY
READ FROM (MT,-.FSGE)
FINISH

```