STUDIA INFORMATICA 2010
Volume 31 Number 3(91)

Roman STAROSOLSKI
Politechnika Śląska, Instytut Informatyki

ZASTOSOWANIE METODY ZMNIEJSZONEJ CZĘSTOŚCI AKTUALIZACJI SŁOWNIKA W UNIWERSALNYM ALGORYTMIE KOMPRESJI LZW

Streszczenie. W niniejszej pracy przedstawiono badania nad zastosowaniem metody zmniejszonej częstości aktualizacji słownika w uniwersalnym algorytmie kompresji LZW. Wyniki badań wskazują, że w przeciętnym przypadku metoda ta pozwala na kilkudziesięcioprocentowe zwiększenie prędkości algorytmu LZW kosztem pogorszenia uzyskiwanych współczynników kompresji o kilka procent, a w przypadku niektórych plików zwiększeniu prędkości kompresji towarzyszy poprawa uzyskiwanych współczynników kompresji.

Słowa kluczowe: uniwersalny algorytm kompresji, algorytm LZW, algorytm adaptacyjny, metoda zmniejszonej częstości aktualizacji słownika, drzewo trie, $O(n)$

APPLICATION OF THE REDUCED DICTIONARY UPDATE FREQUENCY METHOD TO THE LZW UNIVERSAL DATA COMPRESSION ALGORITHM

Summary. This paper presents research on application of the reduced dictionary update frequency method to the LZW universal data compression algorithm. The results show, that in the average case the above-mentioned method allows to increase compression speed by dozens of percents at the expense of worsening the ratio by a few percent, however, in a case of some files the speed increase is accompanied by the improvement in the compression ratio.

Keywords: universal compression, LZW algorithm, adaptive algorithms, reduced dictionary update frequency method, trie structure, $O(n)$

1. Wprowadzenie

W niniejszej pracy opisano badania nad zastosowaniem metody zmniejszonej częstości aktualizacji słownika w uniwersalnym algorytmie kompresji LZW. Metoda zmniejszonej częstości aktualizacji wcześniej badana była zarówno dla pewnego algorytmu kompresji statystycznej [1], jak i dla algorytmu LZW [2], jednak wspomniane badania dotyczyły jedynie bezstratnej kompresji obrazów. Dla danych obrazowych metoda pozwoliła na zwiększenie prędkości algorytmu kompresji opartego na kodowaniu LZW o kilkadziesiąt procent, kosztem nieznacznego, z praktycznego punktu widzenia, pogorszenia uzyskiwanych przez algorytm współczynników kompresji (tj. o kilka procent). Należy tutaj nadmienić, że w przypadku kompresji obrazów LZW był tylko jednym z kilku elementów bardziej złożonego algorytmu kompresji.

Algorytmy uniwersalne (takim algorytmem jest LZW) przeznaczone są do kodowania dowolnych klas danych; choć pierwotnym obszarem ich zastosowania była kompresja tekstów, to stosowane są obecnie do danych wszelkiego typu, w tym nawet do danych multimedialnych. Algorytm LZW należy do klasy algorytmów słownikowych; ze względu na niewielką złożoność czasową algorytmy słownikowe stosowane są obecnie do kompresji danych przesyłanych sieciami komputerowymi, co pozwala na przesłanie większej ilości danych w jednostce czasu przez kanał transmisyjny o ograniczonej przepustowości, jak również do kompresji danych składowanych na nośnikach pamięci masowej. Celem opisywanych badań było sprawdzenie, czy metoda zmniejszonej częstości aktualizacji słownika zastosowana do algorytmu LZW sprawdzi się również w uniwersalnej kompresji danych.

Układ dalszej części niniejszej pracy jest następujący: w punkcie 2 szczegółowo omówiono badany algorytm, tj.: podstawowy algorytm LZW, użytą strukturę danych słownika, metodę zmniejszonej częstości aktualizacji słownika, sposób implementacji podstawowych operacji realizowanych w algorytmie oraz złożoności obliczeniowe algorytmu. Punkt 3 zawiera opis przeprowadzonych badań, a punkt 4 krótko podsumowuje ich wyniki.

2. Algorytm kompresji

2.1. Algorytm LZW

Algorytm LZW to zaproponowane w roku 1984 przez T.A. Welcha udoskonalenie zaprezentowanego w roku 1978 przez J. Ziva i A. Lempela algorytmu LZ78 [3]. Oba algorytmy podczas kompresji gromadzą informacje o kompresowanych danych w tzw. słowniku, stąd często określane są mianem słownikowych. Słownik to struktura zawierająca frazy (krótkie

ciągi) znaków, które wystąpiły w już przetworzonej części kompresowanego ciągu. W algorytmie LZ78 pobiera się kolejne znaki kodowanego ciągu budując z nich pewną frazę. Po pobraniu kolejnego znaku i dodaniu go na końcu już zbudowanej frazy sprawdza się, czy słownik tę frazę zawiera. Jeżeli frazy nie ma w słowniku, to wyprowadzany jest indeks najdłuższej dopasowanej frazy i ostatnio pobrany znak. Na początku kompresji słownik jest pusty, a każdorazowo po zakodowaniu frazy i znaku wstawia się do słownika frazę, dla której operacja szukania zakończyła się niepowodzeniem i rozpoczyna się budowanie nowej frazy od frazy pustej. Modyfikacja zaproponowana przez Welcha polega na wstępnym wypełnieniu słownika jednoznakowymi frazami z wszystkimi znakami alfabetu. Dzięki tej modyfikacji wyprowadzane mogą być tylko indeksy fraz, a współczynniki kompresji uzyskiwane przez algorytm LZW są lepsze niż w przypadku algorytmu LZ78. Algorytm LZW można w uproszczeniu przedstawić za pomocą pseudokodu:

Algorytm LZW

```
zainicjalizuj słownik
f := fraza pusta
while (nie pobrano wszystkich znaków komunikatu) do
    pobierz znak s
    if (f|s znajduje się w słowniku) then
        f := f|s
    else
        wyprowadź indeks frazy f
        wstaw frazę f|s do słownika
        f := s
    endif
endwhile
wyprowadź indeks frazy f
```

Działanie algorytmu kompresji i dekompresji LZW zilustrowane zostanie na poniższym przykładzie. Przykład ten ilustruje również pewien szczególny przypadek powodujący, iż algorytm dekompresora nie jest trywialnym odtworzeniem operacji wykonywanych przez kompresor. Dekompresor po otrzymaniu kodu danej frazy, na wyjście wyprowadza odpowiednią frazę ze słownika, ale w stosunku do kompresora nowe frazy do słownika wstawiane są z opóźnieniem. Nowa fraza może być wstawiona do słownika dopiero po otrzymaniu kodu następnej frazy, gdyż składa się ona z frazy aktualnej i pierwszego znaku następnej frazy.

Przykład. Algorytmem LZW kodujemy komunikat ***barbararabarbarbar***. Wstępnie słownik wypełniamy frazami (w nawiasach podano indeksy fraz): ***a*** (0), ***b*** (1) i ***r*** (2). W pierwszej kolumnie tabeli 1 widnieje pozostała do zakodowania część komunikatu, poszczególne jej fragmenty rozdzielone znakami „|” to odpowiednio: najdłuższy przedrostek znaleziony w słowniku, pierwszy znak, po pobraniu którego nie powiodła się operacja wyszukania w słowniku, oraz pozostała jeszcze niepobierana część komunikatu.

Tabela 1

Kodowanie LZW

koder			dekoder		
znaki pozostałe do zakodowania	**kod**	**fraza wstawiana do słownika**	**kod**	**wyprowadzana fraza**	**fraza wstawiana do słownika**
b*|*a*|*rbararabarbarbar	1	***ba*** (3)	1	***b***	–
a*|*r*|*bararabarbarbar	0	***ar*** (4)	0	***a***	***ba*** (3)
r*|*b*|*ararabarbarbar	2	***rb*** (5)	2	***r***	***ar*** (4)
ba*|*r*|*arabarbarbar	3	***bar*** (6)	3	***ba***	***rb*** (5)
r*|*a*|*rabarbarbar	2	***ra*** (7)	2	***r***	***bar*** (6)
ar*|*a*|*barbarbar	4	***ara*** (8)	4	***ar***	***ra*** (7)
a*|*b*|*arbarbar	0	***ab*** (9)	0	***a***	***ara*** (8)
bar*|*b*|*arbar	6	***barb*** (10)	6	***bar***	***ab*** (9)
barb*|*a*|*r	10	***barba*** (11)	10	***barb***	***barb*** (10)
***a*|*r*|**	4	–	4	***ar***	***barba*** (11)

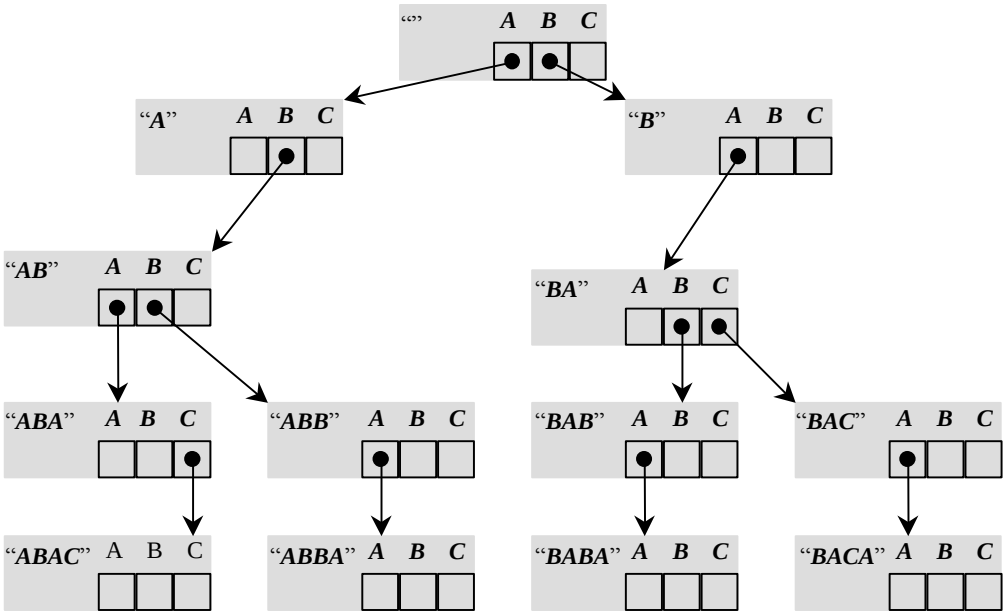
Wspomniany przypadek szczególny występuje tu w momencie (komórki podkreślone w tabeli 1), gdy dekoder otrzymuje indeks frazy 10, której jeszcze nie ma w słowniku. Taka sytuacja ma miejsce jedynie w przypadku, gdy dana fraza jest konkatenacją ostatnio zakodowanej frazy i pierwszego znaku ostatnio zakodowanej frazy. W przypadku gdy dekoder otrzymuje indeks frazy, która nie znajduje się w słowniku, należy do słownika wstawić frazę utworzoną przez konkatenację ostatnio wyprowadzonej frazy i pierwszego znaku tej frazy oraz wyprowadzić tak utworzoną frazę na wyjście.

2.2. Struktura słownika

Zazwyczaj strukturę danych słownika konstruuje się z zastosowaniem funkcji mieszających znacznie przyspieszających proces wyszukiwania frazy. W tym przypadku należy wykonać jedną operację wyszukiwania w słowniku na każdy znak czytanego pliku. W opisywanym algorytmie do przechowywania fraz zastosowany został słownik w postaci uproszczonego drzewa trie. W literaturze, oprócz pochodzącego z języka angielskiego terminu „drzewo trie” [4] spotyka się również określenie „drzewo wektorowe” [5]. Struktura ta pozwala na zmodyfikowanie wyjściowego algorytmu LZW, przez co uzyskuje się zmniejszenie liczby wyszukiwań w słowniku z jednego dla każdego znaku kompresowanego ciągu do jednego na wyprowadzony indeks frazy. Średnio wyszukiwań będzie tyle razy mniej, ile wynosi średnia długość frazy.

Frazy umieszczane w słowniku przez algorytm LZW mają jedną, istotną ze względu na strukturę danych słownika, cechę: jeżeli w słowniku jest zawarta dana fraza, to muszą się tam również znajdować wszystkie jej prefiksy. Cecha ta pozwala na uproszczenie struktury drzewa trie. Wystarczy stosować tylko jeden rodzaj węzła i nie trzeba uzupełniać alfabetu

o znak „koniec frazy”. W konsekwencji każda z fraz przechowywanych w słowniku, niezależnie od swojej długości, wymaga takiej samej (stałej) liczby bajtów.



Rys. 1. Uproszczone drzewo trie
Fig. 1. Simplified trie structure

W zastosowanym wariancie drzewa trie węzeł zawiera tablicę wskaźników do potomków o rozmiarze równym wielkości alfabetu (po jednej pozycji dla każdego znaku alfabetu). Fraza (klucz) znajduje się w drzewie, gdy, począwszy od korzenia, można w drzewie wyznaczyć ścieżkę taką, że *n*-ta gałąź tworząca ścieżkę wychodzi z pozycji tablicy wskaźników znajdującej się pod indeksem odpowiadającym *n*-temu znakowi klucza. Na przykład drzewo trie z rys. 1, skonstruowane dla alfabetu {***A***, ***B***, ***C***}, zawiera frazy: “”, “***A***”, “***AB***”, “***ABA***”, “***ABAC***”, “***ABB***”, “***ABBA***”, “***B***”, “***BA***”, “***BAB***”, “***BABA***”, “***BAC***” i “***BACA***”.

Przez pojemność słownika rozumieć będziemy maksymalną liczbę fraz, które jednocześnie mogą znajdować się w słowniku. Pojemność słownika powinna być potęgą dwójki, gdyż indeksy fraz są kodowane z użyciem kodu binarnego stałej długości. Słowo kodowe dla frazy ma długość $\lceil \log_2(q) \rceil$ bitów, gdzie q jest liczbą fraz aktualnie przechowywaną w słowniku. Jeżeli ograniczymy pojemność do 2^{16}, a węzły drzewa trie umieścimy w tablicy, to wskaźniki będzie można zastąpić 16-bitowymi indeksami (które będą jednocześnie indeksami fraz) i uniknąć wielokrotnych alokacji pamięci.

Złożoność pamięciowa tak zdefiniowanej struktury słownika wynosi $2 \cdot \|S\| \cdot v$, gdzie $\|S\|$ jest rozmiarem alfabetu, którego znaki tworzą frazy przechowywane w słowniku, a v to pojemność słownika. Złożoność pamięciowa słownika dla alfabetu o 256 znakach wynosi zatem $512 \cdot v$. Ze względu na zastosowanie opisanej w następnym punkcie metody adaptacji, rzeczywisty rozmiar struktury danych słownika jest nieznacznie większy ($521 \cdot v$). Rozmiar

słownika wynosi więc od około 0.25 MB dla słownika o pojemności 512 fraz do około 32 MB dla słownika zdolnego pomieścić 65536 fraz.

Słownik po inicjalizacji zawiera frazę pustą. Fraza ta umieszczona jest w węźle o indeksie 0 i jest korzeniem drzewa trie. Ponieważ żadna inna fraza nie jest jej prefiksem, to wartość 0 w tablicy potomków węzła można traktować nie jako krawędź prowadzącą do węzła 0, ale jako brak krawędzi. Słownik po inicjalizacji zawiera również 256 jednoznakowych fraz (o indeksach 1 – 256).

Dysponując takim słownikiem, można zmodyfikować wyszukiwanie w słowniku. Przyjmijmy następujące oznaczenia: niech *f* oznacza frazę, *s* znak, s_i znak o wartości *i*, a $f \mid s$ konkatenację frazy f i znaku *s*. Zamiast sprawdzania, czy $f \mid s$ znajduje się w słowniku, sprawdzamy, czy fraza *f*, która, jak wiemy, znajduje się w słowniku, jest przedrostkiem frazy $f \mid s$. Fraza *f* znajduje się już w słowniku, ponieważ albo została znaleziona podczas poprzedniego wyszukiwania, albo jeżeli nie była znaleziona w poprzednim wyszukiwaniu, to ma długość jednego znaku i z definicji jest obecna w słowniku dla algorytmu LZW od momentu inicjalizacji słownika.

Sprawdzenie, czy fraza $f \mid s_i$ jest zawarta w słowniku, realizujemy badając, czy na pozycji *i* tablicy wskaźników w węźle odpowiadającym frazie *f* jest niepuste wskazanie. W przypadku znalezienia niepustego wskazania zapamiętujemy je, jako jednoznacznie wyznaczające frazę $f \mid s_i$. Dysponując takim słownikiem, dla pojedynczego znaku pliku wejściowego wykonujemy pojedyncze sprawdzenie elementu tablicy w węźle drzewa, podczas gdy dla zwykłego słownika dla pojedynczego znaku pliku wejściowego musimy wykonać wyszukiwanie całej dotychczas zbudowanej frazy.

2.3. Metoda zmniejszonej częstości aktualizacji słownika

Słownik o określonej pojemności, w miarę postępowania kompresji, wypełnia się nowymi frazami aż zostanie całkowicie wypełniony. Mówimy, że do momentu wypełnienia słownika algorytm LZW jest adaptacyjny. W momencie gdy słownik jest pełny, można zaprzestać wstawiania nowych fraz, tj. „zamrozić słownik”. W tym przypadku dla reszty pliku algorytm będzie stały – rozwiązanie takie będzie dobre tak długo, jak długo charakterystyka kompresowanych danych będzie pozostawała niezmienna. W praktyce, nawet dla pozornie jednorodnych zbiorów danych, takich jak na przykład tekst dużej powieści, algorytmy stałe uzyskują współczynniki kompresji istotnie gorsze od algorytmów adaptacyjnych. Można także powtórnie zainicjalizować słownik i budować go od nowa dla nowych danych. W algorytmie LZC [6], stosowanym przez kompresor compress systemu Unix, łączy się oba rozwiązania. Po wypełnieniu słownika zaprzestaje się wstawiania nowych fraz i monitoruje się chwilowy współczynnik kompresji. W razie pogorszenia się jego wartości wykonuje się powtórną ini-

cjalizację słownika (operacja ta powoduje chwilowe znaczne pogorszenie współczynnika kompresji).

W przyjętym rozwiązaniu struktura danych słownika jest bardzo duża (słownik wymaga ponad 500 bajtów na każdą frazę). Inicjalizacja tak dużej struktury jest czasochłonna. Wykonywana kilkakrotnie podczas kompresji spowolniłaby średnią prędkość przetwarzania i przyczyniła się do gwałtownych spadków prędkości chwilowej. Aby po wypełnieniu słownika kompresor nadal mógł się adaptować do charakterystyki przetwarzanych danych, zastosowano następujące rozwiązanie: po wstawieniu frazy do słownika, jeżeli słownik jest pełny, to jest z niego usuwana jedna fraza, która nie jest prefiksem żadnej innej. Od momentu pierwszego wypełnienia słownika kompresor może wstawiać nową frazę do słownika za każdym razem, gdy wyprowadza znalezioną frazę, lub mniej często. Po wstawieniu frazy do słownika losowana jest liczba zaniechań kolejnych operacji aktualizacji słownika:

Zmodyfikowane kodowanie LZW po wypełnieniu słownika

```
zainicjalizuj słownik
f := fraza pusta
czekaj := 0
while not eof(plik wejściowy) do
    pobierz znak s
    if f|s znajduje się w słowniku then
      f := f|s
    else
      wyprowadź kod frazy f
      if czekaj = 0 then
        wstaw frazę f|s do słownika
        usuń ze słownika jedną frazę
        czekaj := random(zakres)
      else
        dec(czekaj)
      endif
      f := s
    endif
endwhile
```

Metoda ta pozwala, przez dobranie zakresu losowanych wartości, na zwiększenie prędkości kompresji kosztem wolniejszej adaptacji do charakterystyki kompresowanych danych. W przypadku algorytmu kompresji statystycznej [1] wykazano, że ww. metoda pozwala również na zrównoleglenie algorytmu z natury sekwencyjnego [7].

Poczynając od momentu wypełnienia słownika przez częstość aktualizacji słownika rozumieć będziemy stosunek liczby fraz, po zakodowaniu których następuje aktualizacja słownika do liczby zakodowanych fraz. Pełna częstość aktualizacji słownika (100%) oznacza, że wstawianie frazy do słownika następuje każdorazowo po zakodowaniu frazy. Jeżeli losowana liczba zaniechań kolejnych operacji aktualizacji słownika znajduje się w przedziale $[0, j]$ i ma rozkład równomierny, to wyrażona w procentach częstość aktualizacji słownika φ wynosi

$$\varphi = \frac{2}{j+2} \cdot 100\%$$

Zastosowana metoda adaptacji kompresora wymaga szybkiego wyszukiwania i usuwania fraz niebędących prefiksami innych fraz, czyli będących liśćmi drzewa trie. Ze względu na wymagania metody adaptacji strukturę danych słownika uzupełniono o tablicę liści, a w węzłach drzewa trie umieszczono dodatkowe pola zawierające: licznik potomków, wskaźnik na rodzica, wskaźnik do tablicy liści i ostatni znak frazy przechowywanej w węźle.

Frazy umieszczane w słowniku podczas inicjalizacji (fraza pusta i 256 jednoznakowych fraz) muszą być traktowane jako frazy nieusuwalne, nawet jeśli są liśćmi drzewa trie. Fraza pusta jest prefiksem wszystkich fraz w słowniku. Fraza ta nie zostanie usunięta, jeżeli w słowniku znajdują się inne frazy, a ponieważ usuwanie fraz następuje dopiero po wypełnieniu słownika, to nie ma niebezpieczeństwa usunięcia frazy pustej. Inaczej rzecz się ma z jednoznakowymi frazami znajdującymi się w słowniku LZW po inicjalizacji. Ponieważ po wypełnieniu słownika jednoznakowe frazy mogą nadal być liśćmi drzewa trie, należy uniemożliwić usunięcie tych fraz. Można by to osiągnąć przez modyfikację algorytmu polegającą na przykład na sprawdzaniu, czy fraza, którą chcemy umieścić w tablicy liści, nie jest frazą jednoznakową. Struktura danych słownika pozwala na zabezpieczenie fraz przed usunięciem bez modyfikacji algorytmu. Występujący w węźle licznik potomków służy do sprawdzania, czy fraza jest liściem drzewa trie. Jeżeli liczniki potomków w węzłach odpowiadających jednoznakowym frazom otrzymają podczas inicjalizacji wartość 1 (zamiast 0), to jednoznakowe frazy nigdy nie zostaną umieszczone w tablicy liści. Podczas inicjalizacji słownika niewykorzystane pozycje w tablicy węzłów są wypełniane w następujący sposób: elementy tablicy wskaźników na potomków węzła i licznik potomków otrzymują wartość 0, wartości w pozostałych polach węzła są nieistotne, nie musi być im nadana żadna konkretna wartość.

2.4. Operacje na strukturze słownika

Na strukturę danych słownika składają się zatem tablica węzłów, tablica liści i kilka zmiennych sterujących. Składniki te wraz z krótkim komentarzem zostały przedstawione w poniższym pseudokodzie:

Struktura danych słownika

```
poj_słow  – pojemność słownika
index     – typ: 16-bitowa liczba całkowita bez znaku, do przechowywania indeksów fraz,
              oraz licznika potomków węzła
byte      – typ: 8-bitowa liczba całkowita bez znaku, do przechowywania znaków

słow : array[0..poj_słow-1] of          – tablica węzłów drzewa trie
         record
           kraw : array[0..255] of index;     – tablica krawędzi wychodzących
           l_kraw : index;                    – licznik potomków
```

```
            liść : index;                    – pozycja w tablicy liści
                                               nieistotna, gdy l_kraw ≠ 0
            rodzic : index;                  – wskaźnik na rodzica
            znak : byte;                     – ostatni znak frazy w tym węźle
          end;
słow_pełny : boolean;   – znacznik przyjmujący podczas inicjalizacji słownika wartość false
                          przyjmuje wartość true po wypełnieniu słownika
wolny_wiersz : index;   – dla słow_pełny=false pierwsza wolna pozycja w tablicy węzłów
                          dla słow_pełny=true jedyna wolna pozycja w tablicy węzłów
liście : array[0..poj_słow-1] of index;   – tablica liści
liść_ostatni : index;     – indeks ostatniego elementu w tablicy liści (liczba liści – 1)
liść_usuń : index;        – indeks (w tablicy liści) węzła do usunięcia
```

Wyszukiwanie frazy $f \mid s$ realizowane jest teraz przez odwołanie pośrednie do tablicy. Przypisanie $f := s$ realizowane jest jako `f := s + 1`, gdyż jednoznakowe frazy zawierające znaki o wartościach 0 – 255 umieszczone są w tablicy przechowującej słownik pod indeksami odpowiednio 1 – 256:

Kodowanie po wypełnieniu słownika

```
zainicjalizuj słownik
f := 0                                – f := fraza pusta
czekaj := 0
while not eof(plik wejściowy) do
    pobierz znak s
    if słow[f].kraw[s] <> 0 then      – jeżeli f | s znajduje się w słowniku
      f := słow[f].kraw[s]            – f := f | s
    else
      wyprowadź kod frazy f
      if czekaj = 0 then
        wstaw_frazę(f, s)
        usuń_frazę()
        czekaj := random(zakres)
      else
        dec(czekaj)
      endif
      f := s + 1                      – f := s
    endif
endwhile
```

Procedury `wstaw_frazę` i `usuń_frazę` ilustrują podstawowe operacje wykonywane w algorytmie LZW na słowniku. Procedura `wstaw_frazę` (pseudokod poniżej) wstawia frazę $f \mid s$ do słownika. Nowa fraza wstawiana jest do tablicy przechowującej słownik na pozycję o indeksie `wolny_wiersz`. Wartość zmiennej `wolny_wiersz` nie jest w procedurze `wstaw_frazę` aktualizowana. Wyznaczenie nowej wartości dla tej zmiennej jest istotne tylko dla niewypełnionego słownika.

W węźle o indeksie `wolny_wiersz` procedura `wstaw_fraze()` nadaje wartości następującym polom: `rodzic`, `znak` i `liść`. Pozostałe pola powinny zawierać wartość 0 nadaną podczas inicjalizacji bądź pozostawioną po usunięciu węzła.

Procedura wstawiania frazy *f | s* do słownika

```
procedure wstaw_frazę(f : index, s : byte)
begin
  inc(słow[f].l_kraw)                  – wstaw nową krawędź do węzła f
  słow[f].kraw[s] := wolny_wiersz
  słow[wolny_wiersz].rodzic := f       – wstaw liść
  słow[wolny_wiersz].znak := s
  if słow[f].l_kraw = 1 then           – jeżeli rodzic był liściem
    liście[słow[f].liść] := wolny_wiersz    – to nowy liść zastępuje
    słow[wolny_wiersz].liść = słow[f].liść – rodzica w tablicy liści
  else                                 – jeżeli rodzic nie był liściem
    inc(liść_ostatni)                        – to wstaw liść do tablicy liści
    liście[liść_ostatni]=wolny_wiersz
    słow[wolny_wiersz].liść=liść_ostatni
  endif
end
```

Usunięcie pojedynczej frazy z pełnego słownika realizowane jest przez procedurę `usuń_frazę` (pseudokod poniżej). Tablica liści przeglądana jest z użyciem zmiennej `liść_usuń`. Jeżeli usunięty zostanie ostatni liść w tablicy (o indeksie `liść_ostatni`), to do następnego wywołania procedury `usuń_frazę()` wartość zmiennej `liść_usuń` będzie większa o 1 od indeksu ostatniego liścia `liść_ostatni`. Nie stanowi to problemu, gdyż w takiej sytuacji pierwszą czynnością wykonywaną po ponownym wywołaniu procedury będzie dekrementacja zmiennej `liść_usuń`. Po zakończeniu działania procedury `usuń_frazę()` zmienna `wolny_wiersz` zawiera indeks jedynego wolnego miejsca w tablicy przechowującej słownik. Po wykonaniu tej procedury słownik już nie jest pełny.

W momencie pierwszego wypełnienia słownika znacznikowi `słow_pelny` nadawana jest wartość `true`. Ponieważ od tego momentu procedura `usuń_frazę()` jest w algorytmie LZW wywoływana zawsze bezpośrednio po procedurze `wstaw_frazę()`, to nie ma konieczności dodatkowego nadawania wartości zmiennym stanu struktury danych słownika. W momencie wywoływania procedury `wstaw_frazę()` wartość `true` znacznika `słow_pelny` oznacza, że w słowniku jest miejsce na jedną i tylko jedną frazę. Po usunięciu węzła nie ma potrzeby wyzerowania elementów tablicy wskaźników na potomków węzła ani licznika potomków. Pola te zawierają wartość 0, gdyż usuwany węzeł był liściem.

Procedura usuwania pojedynczej frazy z wypełnionego słownika

```
procedure usuń_frazę()
var
  rodzic : indeks;   – rodzic usuwanego liścia
begin
  if liść_usuń > 0 then   – wyznacz liść do usunięcia
    dec(liść_usuń)
  else
    liść_usuń := liść_ostatni
  endif
```

```
  wolny_wiersz := liście[liść_usuń]     – indeks zwalnianej pozycji w słowniku
  rodzic := słow[wolny_wiersz].rodzic   – znajdź rodzica usuwanego liścia

  słow[rodzic].kraw[słow[wolny_wiersz].znak] := 0   – usuwamy krawędź
  dec(słow[rodzic].l_kraw)                            – z węzła rodzica

  if(słow[rodzic].l_kraw) <> 0 then – jeżeli rodzic nie jest liściem, to usuwamy liść wsta-
                                    – wiając w jego miejsce ostatni liść z tablicy liści
    liście[liść_usuń] := liście[liść_ostatni]
    słow[liście[liść_ostatni]].liść := liść_usuń
    dec(liść_ostatni)
  else                          – rodzic jest liściem, zastępuje byłego potomka w tablicy liści
    liście[liść_usuń] := rodzic
    słow[rodzic].liść := liść_usuń
  endif
end
```

2.5. Złożoności obliczeniowe algorytmu

Na złożoność pamięciową algorytmu ma wpływ pojemność słownika, nie zależy ona natomiast od rozmiaru kompresowanego pliku. W zrealizowanej implementacji, dla 8-bitowego alfabetu i słowników o pojemności do 2^{16} fraz, każda fraza w słowniku wymaga 521 bajtów, zatem złożoność pamięciową M kompresora można opisać wzorem

$$M = 521 \cdot v \text{ bajtów}$$

gdzie v oznacza pojemność słownika.

Algorytm charakteryzuje się liniową złożonością czasową. W trakcie kompresji przechowuje się dotychczas zbudowaną frazę f. Kolejny znak s ciągu wymaga pojedynczego sprawdzenia, czy w słowniku jest fraza $f \mid s$. Jeżeli fraza $f \mid s$ znajduje się w słowniku, to znak s dołączamy na koniec frazy f. Jeżeli frazy $f \mid s$ nie ma, to wyprowadzany jest indeks frazy f i rozpoczyna się budowanie nowej frazy f od jednoznakowej frazy zawierającej tylko znak s. W algorytmie LZW po wyprowadzeniu indeksu frazy wstawia się frazę $f \mid s$ do słownika. Jeżeli po wstawieniu nowej frazy słownik jest pełny, to usuwana jest z niego jedna fraza niebędąca prefiksem żadnej innej (o długości przynajmniej 2 znaków). Po wypełnieniu słownika kompresor może wstawiać nową frazę do słownika za każdym razem, gdy wyprowadza znalezioną frazę lub z mniejszą częstością. Po wstawieniu frazy do słownika losowana jest liczba zaniechań kolejnych operacji aktualizacji słownika. Złożoność czasową T kompresora słownikowego można opisać wzorem

$$T(n) = n\,(c_F + c_K / l_f + \varphi\, c_U / l_f) = O(n)$$

gdzie c_F to przypadający na pojedynczy znak ciągu, koszt wyszukania w słowniku frazy oraz modyfikacji frazy, c_K to koszt zakodowania frazy, l_f to średnia długość wyprowadzanej frazy, φ to częstość aktualizacji słownika, a c_U to koszt pojedynczej aktualizacji słownika.

W przypadku struktury danych słownika zbudowanej na podstawie uproszczonego drzewa trie umieszczonego w tablicy wyszukanie w słowniku frazy $f \mid s$, gdy znany jest indeks

frazy *f* realizowane jest za pomocą dwóch odwołań do tablicy. Modyfikacja frazy, tj. wydłużenie jej o *s*, jeżeli znaleziono $f \mid s$, bądź inicjalizacja w przeciwnym razie, to pojedyncza operacja przypisania. Zakodowanie frazy w zrealizowanej implementacji wymaga kilku operacji przypisania. Kodowanie wykonujemy dla każdej wyprowadzonej frazy. Oznaczony przez c_U koszt pojedynczej aktualizacji słownika (wstawienia jednej frazy i skasowania jednej frazy) wymaga kilkakrotnie więcej odwołań do tablicy i przypisań niż c_F. W zależności od wartości parametru φ aktualizację słownika wykonujemy dla każdej wyprowadzonej frazy lub rzadziej.

Należy zwrócić uwagę, że podczas dekompresji nie dokonuje się wyszukiwania w słowniku, co powoduje, iż algorytm LZW jest „asymetryczny". Dekompresja dla algorytmu LZW ma mniejszą złożoność czasową od kompresji. Znacznie mniejsza jest również złożoność pamięciowa. Ponieważ słownik nie musi umożliwiać szybkiego wyszukiwania fraz, nie potrzebujemy drzewa trie. Złożoność pamięciowa słownika dla dekompresora jest rzędu

$$O(v\,(c + \lceil \log_2(\|S\|) / 8 \rceil))$$

gdzie $\|S\|$ jest rozmiarem alfabetu, a *v* to pojemność słownika.

3. Badania

3.1. Implementacja algorytmu

Zrealizowana w języku C implementacja algorytmu zbudowana została z wykorzystaniem wspomnianej we wprowadzeniu implementacji algorytmu LZW użytej w algorytmie kompresji obrazów [2]. Implementacja umożliwia kompresję ze słownikiem o pojemności do 32768 fraz. Po wstawieniu frazy do słownika, jeżeli słownik jest pełny, to usuwana jest z niego jedna fraza niebędąca prefiksem żadnej innej oraz losowana jest liczba zaniechań kolejnych operacji aktualizacji słownika. W zrealizowanej implementacji aktualizacja słownika mogła następować po wyprowadzeniu 100%, 66.6%, 40.0%, 22.2%, 11.8%, 6.06%, 3.08%, 1.55%, lub 0.778% fraz.

3.2. Procedura badawcza

Badania przeprowadzono na komputerze wyposażonym w procesor AMD Athlon64 3200+ (2.0 GHz) i system operacyjny Windows XP. Implementację omawianego algorytmu skompilowano kompilatorem języków C/C++ środowiska MS Visual Studio 2005. Otrzymano aplikację jednowątkową. Pomiary wykonywano następująco: przeprowadzano jednokrotną kompresję, dla której czasu nie mierzono, a następnie kompresowano ten sam plik wielokrot-

nie (nie mniej niż 10 razy i nie krócej niż przez 1 sekundę), mierzono łączny czas tych przebiegów i dzielono go przez liczbę przebiegów. Czas kompresji mierzony był jako suma czasów zwracanych po zakończeniu procesu przez system operacyjny: czasu wykonywania kodu procesu kompresji i czasu wykonywania kodu procedur jądra systemu wywołanych przez proces (kernel time + user time).

Prędkość kompresji wyrażamy w megabajtach na sekundę [MB/s], gdzie $1\ \mathrm{MB} = 2^{20}$ bajtów. Współczynnik kompresji wyrażamy w bitach na znak [bpc]: $8\ e / n$, gdzie e oznacza rozmiar skompresowanego pliku, n – rozmiar nieskompresowanego pliku. Należy zwrócić uwagę, że współczynnik kompresji wyrażony w bitach na znak jest tym lepszy, im jego wartość mniejsza.

3.3. Wyniki badań

Do eksperymentów użyto zestawu 12 dużych i różnorodnych plików testowych Silesia Corpus. Zestaw opracowany został przez dr. Sebastiana Deorowicza i dostępny jest pod adresem http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia, gdzie znajduje się również dokładny opis poszczególnych plików zawartych w zestawie. W badaniach brano pod uwagę przede wszystkim uśrednione wyniki dla wszystkich plików z zestawu, dla wybranych konfiguracji kompresora zaprezentowane zostaną również wyniki dla poszczególnych plików.

Pomiary współczynnika i prędkości (tabele 2 i 3 oraz rys. 2) kompresji przeprowadzono dla wszystkich dostępnych częstości aktualizacji słownika i ze wszystkimi dostępnymi w zrealizowanej implementacji pojemnościami słownika, będącymi potęgami dwójki: 512, 1024, 2048, 4096, 8192, 16384 i 32768 fraz.

Zastosowanie metody zmniejszonej częstości aktualizacji słownika pozwoliło na znaczne zwiększenie prędkości kompresji. Dla danej pojemności słownika, w porównaniu do prędkości uzyskanej przy pełnej częstości aktualizacji, możliwe jest zwiększenie prędkości kompresji o od ponad 100% do ponad 200% przez zastosowanie minimalnej dostępnej częstości aktualizacji słownika (0.778%). Przyspieszenie odbywa się kosztem pogorszenia średniego współczynnika kompresji o od 8.7% do 20.4%. Z praktycznego punktu widzenia pogorszenie współczynnika o kilkanaście procent wydaje się zbyt duże, by mogło być uznane za nieistotne. Można zauważyć, że dla dużych częstości aktualizacji poprawa prędkości jest proporcjonalna do wzrostu średniego współczynnika kompresji; zmniejszanie częstości poniżej pewnego progu skutkuje natomiast w coraz mniejszej poprawie prędkości i coraz większym wzroście współczynnika, zatem lepszym wyborem będzie częstość aktualizacji większa od minimalnej. Jeżeli wybierzemy częstości aktualizacji, dla których pogorszenie współczynnika nie przekroczy 5% w porównaniu do wyników pełnej częstości aktualizacji, to uzyskamy zwiększenie prędkości o od 72% do 98%. Wydaje się, że w wielu praktycznych zastosowaniach

kompresji, np. do transmisji danych przesyłanych siecią komputerową, taki koszt uzyskania prawie dwukrotnego zwiększenie prędkości kompresji (tj. pogorszenie współczynnika o niecałe 5%) jest nieznaczny lub nawet może być uznany za nieistotny.

Tabela 2

Średni współczynnik kompresji [bpc]

częstość aktualizacji	**pojemność słownika**						
	512	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**
100%	4.894	4.508	4.248	4.018	3.795	3.572	3.466
66.6%	4.926	4.534	4.279	4.044	3.833	3.612	3.506
40.0%	4.986	4.574	4.303	4.085	3.886	3.669	3.560
22.2%	5.052	4.612	4.353	4.143	3.950	3.742	3.627
11.8%	5.111	4.674	4.413	4.208	4.025	3.828	3.708
6.06%	5.156	4.728	4.482	4.283	4.106	3.924	3.805
3.08%	5.211	4.791	4.559	4.365	4.194	4.035	3.921
1.55%	5.263	4.862	4.641	4.450	4.295	4.153	4.044
0.778%	5.322	4.940	4.726	4.553	4.414	4.280	4.175

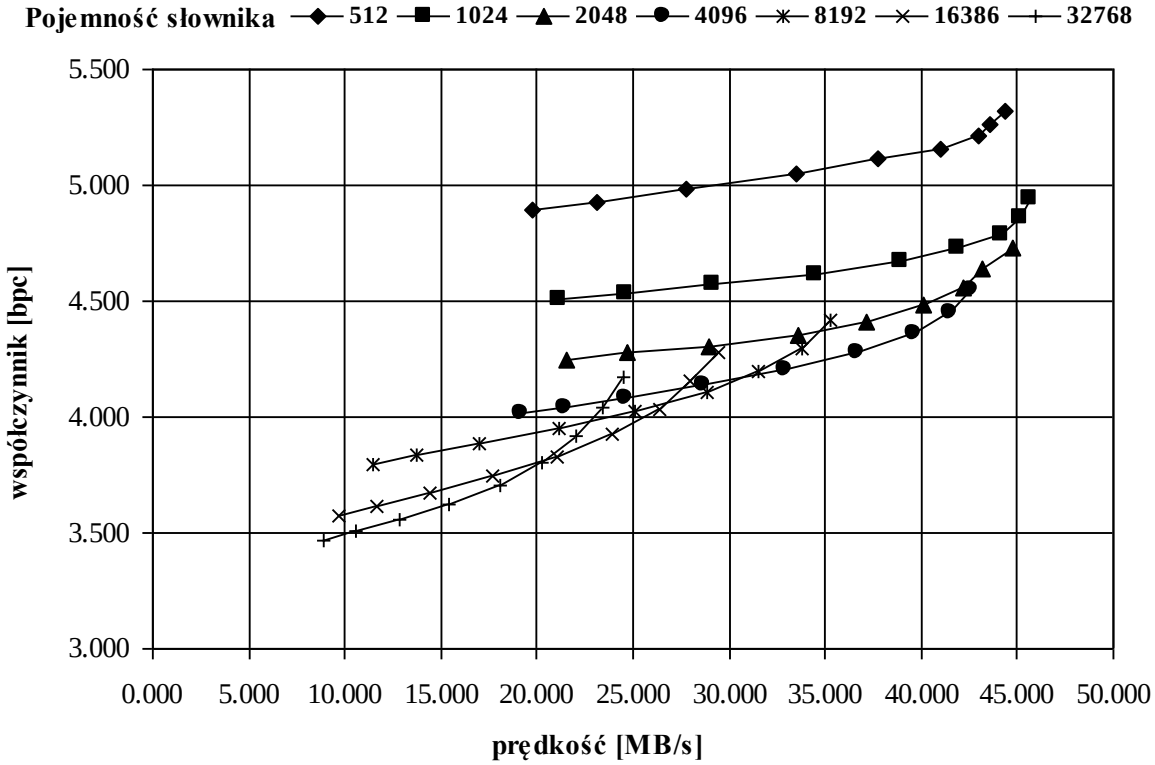
Tabela 3

Średnia prędkość kompresji [MB/s]

częstość aktualizacji	**pojemność słownika**						
	512	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**
100%	19.8	21.1	21.5	19.2	11.5	9.6	8.9
66.6%	23.1	24.6	24.7	21.5	13.7	11.6	10.5
40.0%	27.8	29.1	28.9	24.6	17.0	14.4	12.8
22.2%	33.5	34.5	33.6	28.7	21.2	17.7	15.4
11.8%	37.8	38.9	37.1	32.9	25.1	21.1	18.1
6.06%	41.0	41.9	40.1	36.6	28.9	23.9	20.3
3.08%	43.0	44.2	42.2	39.7	31.5	26.3	22.0
1.55%	43.6	45.1	43.2	41.5	33.8	28.0	23.4
0.778%	44.4	45.6	44.7	42.6	35.3	29.5	24.5

Porównując za sobą wyniki dla różnych pojemności słownika, można zauważyć, że pojemności słownika 512 i 1024 fraz pozwalają uzyskać podobne prędkości i gorsze współczynniki kompresji niż w przypadku słownika 2048 fraz. Prędkości kompresji dla słowników o pojemnościach ponad 4096 fraz są znacznie mniejsze od prędkości uzyskanej dla słownika 4096 fraz (prawdopodobnie przyczyną różnic jest architektura komputera, na którym wykonywano badania, np. rozmiar i sposób wykorzystania przez algorytm pamięci cache procesora). W przypadku dużych pojemności słownika (zwłaszcza dla pojemności 32768 fraz) zastosowanie zmniejszonej częstości aktualizacji powoduje znacznie większe pogorszenie współczynnika kompresji niż w przypadku mniejszych pojemności słownika – prawdopodobnie duży słownik aktualizowany z niewielką częstością nie jest w stanie wystarczająco szybko zaadaptować się do zmieniającej się charakterystyki kompresowanych danych. W efekcie, zastosowanie mniejszego słownika, np. 4096 fraz i częstości aktualizacji 40%, daje lepszy

współczynnik i lepszą prędkość niż zastosowanie ośmiokrotnie większego słownika aktualizowanego z częstością 0.778%.



Rys. 2. Wyniki kompresji dla różnych pojemności i częstości aktualizacji słownika
Fig. 2. Compression speed and ratio for various dictionary sizes

Tabela 4

Współczynnik kompresji dla słownika 4096 fraz [bpc]

plik	rozmiar [MB]	częstość aktualizacji				
		100%	40.0%	11.8%	3.08%	0.778%
dickens	9.7	3.552	3.582	3.626	3.678	3.770
mozilla	48.8	3.984	4.232	4.583	4.956	5.272
mr	9.5	3.014	3.057	3.154	3.239	3.299
nci	32.0	1.122	1.132	1.142	1.159	1.184
ooffice	5.9	5.187	5.431	5.748	6.063	6.345
osdb	9.6	6.397	6.247	6.198	6.190	6.203
reymont	6.3	2.709	2.727	2.757	2.784	2.846
samba	20.6	3.045	3.201	3.451	3.762	4.130
sao	6.9	7.355	7.422	7.517	7.634	7.744
webster	39.5	3.056	3.111	3.183	3.252	3.336
xml	5.1	1.924	2.041	2.286	2.736	3.431
x-ray	8.1	6.875	6.836	6.854	6.924	7.077

W tabelach 4 i 5 przedstawiono wyniki uzyskane dla poszczególnych plików z zestawu danych testowych przez kompresor ze słownikiem 4096 fraz i kilkoma częstościami aktualizacji. Jak widać, wyniki kompresji oraz efekty zastosowania metody zmniejszonej częstości

aktualizacji słownika zależą od rodzaju danych, które kompresujemy. Występują znaczne różnice w prędkości kompresji różnych plików, różnice te nie wynikają jedynie z rozmiaru plików, jednak w każdym przypadku, stosując metodę zmniejszonej częstości aktualizacji, można uzyskać znaczne zwiększenie prędkości kompresji. Dla kilku plików (*mozilla, oofice, samba, xml*) zastosowanie metody zmniejszonej częstości aktualizacji słownika pogarsza współczynnik kompresji znacznie bardziej, niż w przypadku średniego współczynnika dla całego zestawu. W przypadku paru innych (*osdb, x-ray*) metoda zmniejszonej częstości aktualizacji poprawia współczynnik kompresji (jednocześnie znacznie poprawiając prędkość kompresji).

Tabela 5

Prędkość kompresji dla słownika 4096 fraz [MB/s]

plik	rozmiar [MB]	częstość aktualizacji				
		100%	40.0%	11.8%	3.08%	0.778%
dickens	9.7	16.8	22.2	31.1	39.4	43.2
mozilla	48.8	16.7	21.9	30.1	36.4	39.8
mr	9.5	24.5	32.0	41.7	49.4	52.9
nci	32.0	41.3	50.0	58.9	64.8	69.2
ooffice	5.9	12.4	16.3	22.9	30.1	31.7
osdb	9.6	11.2	14.7	21.7	29.0	26.5
reymont	6.3	20.0	25.6	33.7	40.5	44.2
samba	20.6	19.9	25.9	35.1	43.4	47.1
sao	6.9	10.7	14.5	22.1	28.0	30.7
webster	39.5	18.9	24.6	33.7	41.8	45.7
xml	5.1	25.1	31.3	39.6	43.2	45.9
x-ray	8.1	12.2	16.8	23.7	30.1	33.6

4. Podsumowanie

W niniejszej pracy opisano badania nad zastosowaniem metody zmniejszonej częstości aktualizacji słownika w uniwersalnym algorytmie kompresji LZW. Struktura słownika algorytmu zbudowana została na podstawie uproszczonego drzewa trie. Wyniki badań przeprowadzonych na obszernym zestawie danych testowych wskazują, że ww. metoda pozwala na kilkudziesięcioprocentowe zwiększenie prędkości algorytmu LZW kosztem pogorszenia o kilka procent uzyskiwanych przez algorytm współczynników kompresji. Efekty zastosowania metody są różne dla różnych plików; w przypadku niektórych plików uzyskano znaczne zwiększenie prędkości kompresji, któremu towarzyszyła jednoczesna poprawa uzyskiwanych współczynników kompresji.

Podziękowanie. *Praca niniejsza powstała w wyniku projektu badawczego nr BK-219-/RAu2/2009 zrealizowanego w Instytucie Informatyki Politechniki Śląskiej.*

BIBLIOGRAFIA

1. Starosolski R.: Simple Fast and Adaptive Lossless Image Compression Algorithm. Software–Practice and Experience, 2007, Vol. 37(1), s. 65÷91, DOI 10.1002/spe.746.
2. Starosolski R.: Fast and adaptive lossless grayscale image compression using the LZW algorithm. Archiwum Informatyki Teoretycznej i Stosowanej, 1999, Vol. 11, Nr 2, s. 171÷193.
3. Ziv J., Lempel A.: Compression of individual sequences via variable rate coding. IEEE Transactions on Information Theory, Sept. 1978, Vol. 24(5), s. 530÷536.
4. Drozdek A., Simon D. L.: Struktury danych w języku C. WNT, Warszawa 1996.
5. Reingold E. M., Nievergelt J., Deo N.: Algorytmy kombinatoryczne. PWN, Warszawa 1985.
6. Thomas S., Orost J.: Compress (version 4.0) program and documentation. 1985.
7. Starosolski R.: Parallelization of an adaptive compression algorithm using the reduced model update frequency method. Theoretical and Applied Informatics, 2007, Vol. 19(2), s. 103÷115.

Recenzent: Dr inż. Bożena Wieczorek

Wpłynęło do Redakcji 26 listopada 2009 r.

Abstract

This paper presents research on application of the reduced dictionary update frequency method to the LZW universal data compression algorithm. Previously, the effects of applying this method were analyzed for dictionary and statistical natural and medical lossless image compression algorithms. For image data, the method allowed to significantly increase the compression speed without worsening of the compression ratio significantly. In this paper, we describe in detail the key operations on the dictionary structure and report complexities of our variant of the LZW algorithm. The dictionary structure of the LZW algorithm was implemented using the modified trie, which is feasible for the reduced dictionary update frequency method, since it permits to quickly remove certain individual phrases from the dictionary. Practical experiments were performed using diverse set of files from the Silesia Corpus. The results obtained for various types of data show that in the average case the reduced dictionary update frequency method allows increasing of the compression speed by dozens of percents

at the expense of worsening the ratio by a few percents, however, in a case of some files the speed increase is accompanied by the improvement in the compression ratio.

Adres

Roman STAROSOLSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, roman.starosolski@polsl.pl.