Рівненський державний гуманітарний університет

ВОЛИНСЬКИЙ МАТЕМАТИЧНИЙ ВІСНИК

СЕРІЯ ПРИКЛАДНА МАТЕМАТИКА

Збірник наукових праць

Випуск 7 (16)

Рівне-2010

"Волинський математичний вісник. Серія прикладна математика" публікує результати досліджень з математичного моделювання і обчислювальних методів та суміжної проблематики в галузі математики, інформатики, механіки. Розрахований на наукових працівників, викладачів ВНЗ, аспірантів та студентів старших курсів.

"Волынский математический вестник. Серия прикладная математика".
The "Volyn Mathematical Bulletin. Applied Mathematics Series".

Видається у Рівненському державному гуманітарному університеті при сприянні Інституту кібернетики ім. В. М. Глушкова НАН України, Інституту прикладних проблем механіки і математики ім. Я. С. Підстригача НАН України, навчальних закладів та наукових товариств Волинського регіону. Друкується за ухвалою Вченої ради РДГУ (протокол № 3 від 29 жовтня 2010 р.).

Адреса редакції: 33028, Україна, м. Рівне, вул. Остафова, 31,
Рівненський державний гуманітарний університет,
кафедра інформатики та прикладної математики, редакція ВМВ.
Тел.: +380362260444. E-mail: vmvspm@gmail.com.

ISBN 966-7281-02-8

Зміст

УДК 004.043

Бомба А. Я., Шпортько О. В., Шпортько Л. В.

ВИКОРИСТАННЯ СТАТИЧНОГО АРИФМЕТИЧНОГО КОДУВАННЯ У РАСТРОВОМУ ГРАФІЧНОМУ ФОРМАТІ PNG

Проаналізовано особливості контекстно-незалежного кодування у форматі компресії зображень без втрат PNG. Запропоновано спосіб та алгоритми використання арифметичного кодування замість кодування Хафмана у форматі словникового стиснення Deflate, що використовується форматом PNG. Наведено фрагменти програм мовою C для реалізації розглянутого способу арифметичного кодування. Як свідчать експерименти, реалізація запропонованих алгоритмів дозволяє, наприклад, прискорити декодування зображень набору ACT у форматі PNG на 10 – 20 %.

Вступ. На сьогодні для зберігання зображень без втрат Web-дизайнери, розробники сайтів та ПЗ найчастіше використовують формат PNG, оскільки він забезпечує прийнятні показники стиснення та високу швидкість декодування. Проблема підвищення ефективності стиснення зображень у форматі PNG є актуальною сьогодні і буде актуальною в найближчому майбутньому, оскільки навіть наближення до її розв'язання дозволяє зменшити розміри відповідних файлів, що, в свою чергу, дає змогу прискорити їх завантаження з мережі та підвищити ефективність використання дискового простору. Стиснення зображень у цьому форматі [4] досягається за рахунок послідовного застосування предикторів [2], контекстно-залежного словникового алгоритму LZ77 [8] та контекстно-незалежного кодування Хафмана [3]. При цьому предиктори зменшують кореляцію між сусідніми пікселами зображення, що мають, як правило, близькі яскравості кольорів; алгоритм LZ77 усуває надлишковості між однаковими фрагментами, а разом з предикторами – і між однаковими коливаннями кольорів зображення; кодування ж Хафмана зменшує надлишковості між переважаючими яскравостями пікселів. Оптимізація розкладу LZ77 розглядалася в [6], альтернативні варіанти

цього розкладу – в [7], способи вибору предикторів рядків – в [1]. У цій же статті пропонується спосіб підвищення ефективності контекстно-незалежного кодування у форматі PNG.

Особливості стиснення зображень у форматі PNG. Постановка задачі. У PNG-файлах, що використовуються на сьогодні, стиснуті дані зберігаються у відокремлених блоках згідно формату словникового стиснення Deflate [3, 4]. Відповідно до цього формату, у стиснутих блоках містяться результати групової компресії кодами Хафмана розкладу LZ77 відхилень яскравостей компонент пікселів від прогнозованих предикторами значень.

Описуючи словникові алгоритми, фіксовану кількість попередніх закодованих неподільних елементів (літералів) вхідного потоку називають словником, а наступних незакодованих – буфером. Алгоритм LZ77 базується на заміні у вихідному потоці послідовності чергових літералів буфера посиланням на аналогічну послідовність літералів словника у вигляді пари чисел <*довжина; зміщення від закінчення словника*>. Очевидно, що для забезпечення стиснення довжина заміни має бути не менше 3. У випадку відсутності аналогічної послідовності літералів у словнику, перший літерал буфера переноситься у вихідний потік без змін. Після цього закодовані літерали (чи літерал) переносяться з початку буфера в кінець словника і кодування продовжується аналогічно аж до закінчення літералів вхідного потоку. Наприклад, потік кодів біт пікселів *36 38 35 35 36 38 35 36 36 38 35 38 36 38 35 35 38* в закодованому вигляді записується як *36 38 35 35 <3; 4> 36 <3; 4> 38 <4; 12> 38*.

Під час декодування кодів алгоритму LZ77 окремі літерали копіюються у вихідний потік без змін. Пари ж <*довжина; зміщення*> декодуються шляхом послідовного копіювання з кінця вихідного потоку за вказаним зміщенням в кінець вихідного потоку необхідної кількості літералів. Природно, що алгоритм декодування має розрізняти окремі літерали та групи <*довжина; зміщення*>. Згідно з алгоритмом LZH, у форматі DEFLATE з цією метою довжини замін та окремі літерали кодуються разом. Після базових значень довжин міститься визначена фо-

рматом кількість біт, що разом з базовим значенням однозначно визначає довжину заміни. Зміщення зберігається після відповідної довжини аналогічно – у вигляді базового значення та додаткових біт.

Ідея використання контекстно-незалежних методів кодування, які застосовуються на останньому етапі більшості компресорів, полягає у заміні чисел з більшою частотою (тут і надалі – абсолютною) кодами меншої кількості біт, ніж для чисел з меншою частотою. Згідно теореми Шеннона, елемент s_i з ймовірністю появи $p(s_i)$ найвигідніше кодувати $-\log p(s_i)$ бітами (тут і надалі логарифм береться за основою 2). Тоді середня довжина коду елемента після застосування контекстно-незалежного алгоритму має наближатися до *ентропії джерела* [3]:

$$H = -\sum_i p(s_i) \times \log\, p(s_i). \qquad (1)$$

Ентропія джерела зменшується при збільшенні нерівномірності розподілу ймовірностей між елементами. Сьогодні найчастіше використовуються два альтернативних контекстно-незалежних методи: кодування Хафмана, що застосовуються і в форматі Deflate, та арифметичне кодування.

Кодування Хафмана ставить у відповідність кожному елементу залежно від його ймовірності (частоти появи) фіксований префіксний код [3, 5]. Виконують це кодування найчастіше в такій послідовності:

1) підраховують ймовірності (частоти) окремих елементів;
2) впорядковують елементи за спаданням ймовірностей;
3) ітеративно поєднують до отримання одного елемента два елементи з найменшими ймовірностями (найчастіше – останні в утвореному списку). При цьому дописують першому з поєднуваних елементів код *0*, другому – *1*, сумують ймовірності цих елементів для обчислення ймовірності утвореного елемента та вставляють утворений елемент у відсортований список ймовірностей;
4) утворюють коди Хафмана, записуючи сформовані коди у зворотному порядку – від вершини до кожного елемента.

Середня довжина коду Хафмана співпадає з ентропією джерела лише тоді, коли для всіх елементів s_i довжини їх оптимальних кодів

$-\log p(s_i)$ цілі [3]. У даному дослідженні визначено причини та виведено формули для розрахунку відхилень цих величин внаслідок окремих поєднань елементів.

У форматі Deflate літерали/базові значення довжин та базові значення зміщень, що утворюються після виконання алгоритму LZ77, кодуються різними кодами Хафмана. Це пов'язано як з різними діапазонами елементів, так і з різними статистичними властивостями даних розподілів. Крім цього, коди Хафмана для літералів/довжин та зміщень визначаються для кожного блоку стиснутих даних окремо, залежно від частот їх використання, і називаються у цьому випадку *динамічними*. Для однозначного декодування кожного блоку на його початку міститься заголовок, що дозволяє для довільного елемента згенерувати його префіксний код Хафмана. Наприклад, в заголовку блоку динамічних кодів Хафмана міститься перелік довжин префіксних кодів елементів з діапазону [0; 15]. На перший погляд, здається, що стиснення даних в єдиний блок покращує коефіцієнт стиснення, адже тоді буде використано лише один заголовок замість багатьох. З іншого боку, розглянемо, наприклад, послідовність кодів *12112122213434433344*, що містить чотири різні коди *1, 2, 3, 4* з однаковою частотою *5*. Якщо згенерувати коди Хафмана для всієї послідовності, як єдиного блоку, то кожному коду буде поставлено у відповідність код Хафмана довжиною 2 біти і вся послідовність буде закодована в 40 біт. Якщо ж розбити цю послідовність на два рівні блоки, то в кожному з них зустрінуться лише по два різні коди, яким буде поставлено у відповідність код Хафмана довжиною 1 біт і дані всієї послідовності закодуються в 20 біт. Отже, малі блоки динамічних кодів Хафмана призводять до зайвих витрат пам'яті для зберігання заголовків, а великі – не враховують змін характеристик даних. Тому стиснуті дані перед кодуванням Хафмана потрібно розбивати на блоки, які мають різні статистичні характеристики.

Арифметичне стиснення (АС), на відміну від кодування Хафмана, ставить у відповідність кожному черговому елементу інтервал з довжиною, пропорційною його частоті [5]. При цьому з початкового інтервалу (найчастіше [0; 1)) обирають і надалі розглядають підінтер-

вал, що відповідає першому елементу потоку. Цей підінтервал знову розбивають пропорційно частотам окремих елементів і з нього обирають підінтервал, що відповідає другому елементу потоку. Вибір підінтервалів триває аналогічно аж до закінчення елементів потоку. Таким чином, кінцева довжина обраного підінтервала рівна добутку ймовірностей всіх елементів, а його початок залежить від порядку слідування цих елементів в потоці [3]. Результатом АС є будь-яке число з останнього отриманого підінтервала. Зважаючи на скінченність розрядності чисел, для запису результуючого числа щоразу відслідковують перші значущі цифри меж інтервалів і у випадку співпадання записують їх у вихідний потік та виключають з подальшого розгляду. Декодування наступного елемента АС виконують шляхом визначення відповідного чергового підінтервала, якому належить зчитане число. Враховуючи те, що термін дії основних патентів на АС вже минув і таке стиснення точніше кодує окремі елементи. Основною метою цього дослідження є аналіз ефективності використання АС замість кодування Хафмана у форматі PNG.

Дослідження ефективності кодів Хафмана. Визначено спочатку причини відхилень між ентропією джерела (1) та середньою довжиною коду Хафмана. Розглянемо, наприклад, коди Хафмана (див. табл. 1, в кутових дужках вказуються довжини замін) елементів розподілу літералів/довжин з отриманих вище результатів дії алгоритму LZ77, формування яких виконано згідно описаного вище алгоритму. Етапи поєднання елементів цього розподілу наведено на рис. 1 (поєднувані елементи виведені на сірому фоні).

Табл. 1. Характеристики елементів розподілу літералів/довжин

36 38 35 35 <3> 36 <3> 38 <4> 38

Позначення	Значення	Частота	$p(s_i)$	Код Хафмана	$-\log p(s_i)$
s_1	35	2	0.2	10	2.322
s_2	36	2	0.2	11	2.322
s_3	38	3	0.3	00	1.737
s_4	<3>	2	0.2	010	2.322
s_5	<4>	1	0.1	011	3.322

Етап 1

Елемент	Ймовірність	Код
s_3	0.3	
s_1	0.2	
s_2	0.2	
s_4	0.2	0
s_5	0.1	1

Етап 2

Елемент	Ймовірність	Код
s_3	0.3	
$s_{4,5}$	0.3	
s_1	0.2	0
s_2	0.2	1

Етап 3

Елемент	Ймовірність	Код
$s_{1,2}$	0.4	
s_3	0.3	0
$s_{4,5}$	0.3	1

Етап 4

Елемент	Ймовірність	Код
$s_{3,4,5}$	0.6	0
$s_{1,2}$	0.4	1

Рис. 1. Етапи поєднання елементів розподілу літералів/довжин
36 38 35 35 <3> 36 <3> 38 <4> 38

Ймовірність, наприклад, елемента *38* у даному розподілі рівна 0.3, тому оптимальна довжина його коду складає -log(0.3)=1.737. Ми ж для запису коду Хафмана цього елемента витрачаємо 2 біти, тому й відхиляємося від ентропії лише по цьому елементу на 0.789 біта. Виконуючи поєднання двох елементів з найменшими різними ймовірностями, алгоритм генерує для них коди Хафмана однакової довжини. Цим самим, елементу з більшою ймовірністю (а, отже, і частотою) присвоюється більша довжина коду, а елементу з меншою ймовірністю – менша довжина коду від оптимальної. Тому використання зайвих біт для кодування одного елемента призводить до застосування меншої кількості біт для кодування інших елементів стосовно їх оптимальних довжин, хоча й не ліквідовує повністю відхилення середньої довжини коду Хафмана від ентропії. Доведемо це математично.

Твердження. Чим більше відрізняються між собою ймовірності елементів, що поєднуються під час виконання алгоритму Хафмана (тобто чим більше $-\log p(s_i)$ відхиляються від цілих чисел), тим більшою стає різниця між середньою довжиною коду Хафмана і ентропією джерела.

Доведення. Нехай внаслідок виконання алгоритму Хафмана виконується поєднання двох елементів з частотами N та M. Не зменшуючи загальності, вважатимемо $N \geq M$. Алгоритм Хафмана присвоює цим елементам коди довжиною 1, тобто загальна довжина коду для запису цих елементів складе N+M біт. Довжина ентропійного коду цих же елементів, згідно (1), дорівнює $-N\log\left(\frac{N}{N+M}\right)-M\log\left(\frac{M}{N+M}\right)$. Обчислимо відхилення довжини коду Хафмана від довжини ентропійного коду:

$$\begin{aligned}\Delta(N,M) &= N+M+N\log\left(\frac{N}{N+M}\right)+M\log\left(\frac{M}{N+M}\right)= \\ &= N\log\left(\frac{2N}{N+M}\right)+M\log\left(\frac{2M}{N+M}\right).\end{aligned} \tag{2}$$

Визначимо залежність цього відхилення від різниці частот N та M. Нехай S=(N+M)/2, K=(N-M)/2. Підставляючи ці змінні у (2), отримаємо

$$\Delta(S,K)=(S+K)\log(S+K)+(S-K)\log(S-K)-2S\log(S). \tag{3}$$

$$\frac{\partial\Delta(S,K)}{\partial K}=\log\left(\frac{S+K}{S-K}\right)=\log\left(\frac{N}{M}\right)\geq 0. \tag{4}$$

Частинна похідна відхилення по різниці між частотами, згідно (4), дорівнює нулю лише коли частоти елементів співпадають. У всіх інших випадках вона перевищує нуль. Тобто різниця між довжиною коду Хафмана і довжиною ентропійного коду не збільшується лише у випадку поєднань елементів з однаковими частотами (це також слідує з (2) та (3)). Отже, відхилення довжини коду Хафмана від довжини ентропійного коду збільшується при збільшенні різниці між поєднуваними частотами. Збільшення ж відхилень між загальними довжинами кодів свідчить про збільшення відхилень між середніми довжинами цих кодів. Твердження доведено.

Реалізація арифметичного кодування у форматі PNG. Розглянемо тепер результати арифметичного кодування перших трьох елементів цього ж розподілу (рис. 2).

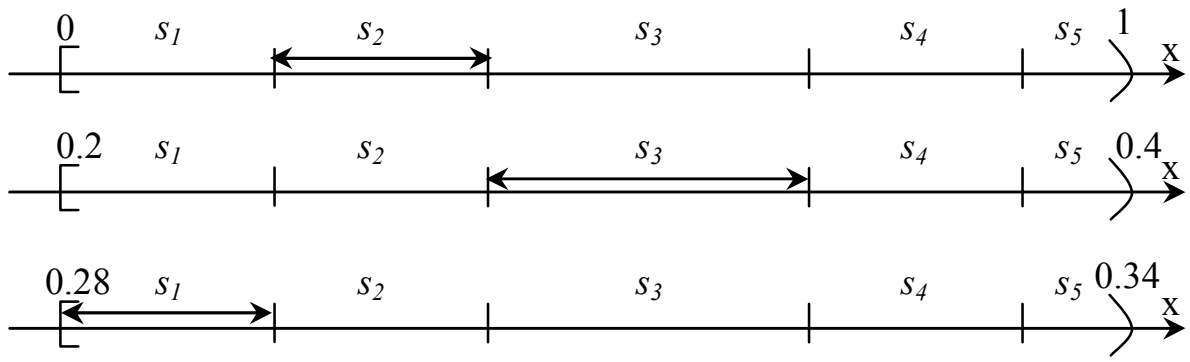

Рис. 2. Арифметичне кодування трьох перших елементів розподілу літералів/довжин 36 38 35 35 <3> 36 <3> 38 <4> 38

Як бачимо, на кожному кроці алгоритму ширина інтервалу звужується пропорційно ймовірності чергового елемента і тому відхилення середньої довжини коду від ентропії є меншими, ніж при кодуванні Хафмана. Але для виконання такого арифметичного стиснення кодеру і, тим більше, декодеру мають бути відомі ймовірності (частоти) окремих елементів. Сьогодні широко використовуються дві стратегії формування інтервалів елементів: статична та адаптивна. У випадку статичної стратегії частоти окремих елементів передаються декодеру у стиснутих даних явно. При використанні адаптивної стратегії інтервали елементів формуються синхронно кодером та декодером в процесі опрацювання даних. Адаптивна стратегія забезпечує, як правило, кращі коефіцієнти стиснення, оскільки не потребує зберігання у стиснутих даних частот окремих елементів, але суттєво сповільнює роботу декомпресора, бо вимагає перерахунку ймовірностей в процесі декодування. Формати графічних файлів мають забезпечувати, насамперед, швидке декодування, тому для модифікації стандарту PNG ми використали статичну стратегію формування інтервалів елементів.

Для реалізації AC ми застосували range-кодер Е. Шелвіна [3], оскільки він виконує зчитування/запис байт а не біт даних і за рахунок цього значно прискорює виконання алгоритму кодування/декодування. Цей кодер для кожного елемента використовує інтервал з цілочисель-

ною довжиною, пропорційною його ймовірності. З метою зберігання довжини інтервалу для кожного елемента у заголовку блоку стиснутих даних замість довжини коду Хафмана нами записується кількість біт для зберігання відповідної довжини інтервалу в загальному інтервалі [0; 32767), а після заголовка – двійковий запис цієї довжини без першого біта (який завжди рівний одиниці). Мовою С підпрограми для генерування та запису довжин інтервалів за частотами елементів записуються, наприклад, так:

```
// функція для визначення кількості біт двійкового запису довжини інтервалу
int countBit(unsigned long diapazon)
 {if (diapazon==0) return 0;
  if (diapazon==1) return 1;
  if (diapazon<=3) return 2;
  if (diapazon<=7) return 3;
  ...
  if (diapazon<=16383) return 14;
  return 15; }
// масштабування сум частот до діапазону [0; 32767)
// для визначення довжин інтервалів елементів
// freq – масив частот та довжин інтервалів
// ehufsi – масив кількостей біт двійкових записів довжин інтервалів
const uint TOP=1<<24; // нижня межа допустимої ширини інтервалу
const uint lenRange=1<<15; // сума масштабованих частот
sumFreq=0;
for (i=0; i<countFreq; ++i)
 sumFreq+=freq[i];
if (sumFreq==0) return; // частоти відсутні – інтервали не визначаємо
mnog=(double)lenRange/sumFreq;
maxFreq=0, indexMaxFreq=0; sumFreq=0;
if (mnog<1) // частоти потрібно зменшити
 {for (i=0; i<countFreq; ++i)
   if (freq[i]>0)
    {freq[i]*=mnog;
     if (!freq[i])
```

```
     freq[i]=1;
    else
     if (freq[i]>maxFreq) // пошук максимальної частоти та її індекса
      {maxFreq=freq[i]; indexMaxFreq=i; }
    sumFreq+=freq[i]; }}
else // частоти потрібно збільшити
 {for (i=0; i<countFreq; ++i)
  if (freq[i]>0)
   {freq[i]*=mnog;
    if (freq[i]>maxFreq)
     {maxFreq=freq[i]; indexMaxFreq=i; }
    sumFreq+=freq[i]; }}
// враховуємо похибки округлень в максимальній частоті
freq[indexMaxFreq]+=lenRange-sumFreq
if (freq[indexMaxFreq]==lenRange)
 freq[indexMaxFreq]--; // для забезпечення можливості запису частоти
// підрахунок сум попередніх частот, кількостей біт двійкового запису
// масштабованих частот та генерація їх кодів без першого біта
sumFreq=0;
for (i=0 ; i<countFreq; ++i)
 if (freq[i]>0)
  {sumPprFreq[i]=sumFreq; // сума попередніх частот
   sumFreq+=freq[i];
   lenCode=countBit(freq[i]);
   ehufsi[i]=lenCode;
   for (j=0; j<lenCode-1; ++j)
    {if ((freq[i] & (1<<j)) != 0) // заносимо з двійкового запису лише одиниці
     bufFreq[pozBufFreq] |= (1 << pozBitFreq) ;
    ++pozBitFreq; // перехід до наступного біта
    if (pozBitFreq >= 8) // перехід до наступного байта
     {pozBufFreq++; bufFreq[pozBufFreq]=0; pozBitFreq=0; }}}
```

Після генерації інтервалів (масштабованих частот) елементів кодування чергового елемента зводиться до перенесення початку активного інтервалу і зменшення його ширини пропорційно відповідній частоті та масштабування ширини отриманого інтервалу до допустимих меж:

```
void ACEncoder::Encode (uint value)
 {low += sumPprFreq[value] * (range>>=15); // перенесення початку
  range*= freq[value]; // врахування частоти
  while(range<TOP) ShiftLow(), range<<=8;} // масштабування ширини інтервалу
```

Декодування ж чергового елемента полягає у визначенні індекса інтервалу, якому належить зчитаний код та у коригуванні параметрів цього інтервалу аналогічно кодуванню:

```
int ACDecoder::Decode ()
 {int index = code / (range>>=15); // черговий зчитаний код
  int index1=0; // визначення індекса інтервалу
  while (sumPprFreq[index1]<=index)
   index1++;
  index=index1-1;
  code -= sumPprFreq[index]*range; // перенесення початку
  range *= freq[index]; // врахування частоти
  // масштабування ширини інтервалу
  while( range<TOP ) code=(code<<8)|buffer[pozBuffer++], range<<=8;
  return index; }
```

Для прискорення декодування шляхом уникнення циклу пошуку початку інтервала чергового елемента, після зчитування заголовка блоку стиснутих даних доцільно створити байтовий масив, у якому для кожного значення загального інтервалу зберегти номер елемента, що йому відповідає:

```
sumFreq=0;
for (i=0; i<countFreq; i++)
 if (huffbits[i]) // елемент в розподілі наявний
  { // зчитування частоти з доповненням першим бітом
   currentFreq=(1<<(huffbits[i]-1)) | decoder.GetBits(huffbits[i]-1);
   freq[i]=currentFreq;
   sumPprFreq[i]=sumFreq;
   memset(inCode+sumFreq, i, currentFreq); // формування байтового масиву
   sumFreq+=currentFreq; }
```

Тоді процедура декодування перепишеться у вигляді:

```
int ACDecoder::Decode ()
 {int index = code / (range>>=15); // черговий зчитаний код
  if (index>=sumPprFreq[256]) // оскільки обробляємо байти
   index=inCode[index]+256;
  else
   index=inCode[index];
  code -= sumPprFreq[index]*range;
  range *= freq[index];
  while( range<TOP ) code=(code<<8)|buffer[pozBuffer++], range<<=8;
  return index; }
```

Застосування такого 32 Кб масиву дозволяє прискорити декодування в середньому на 46 %.

Результати експериментів. На завершення розглянемо результати застосування описаного способу арифметичного кодування для компресії восьми різнотипних 24-бітних зображень стандартного набору файлів ACT у форматі PNG (завантажити їх TIFF-версії можна, наприклад, з http://compression.ru/arctest/act/act-files.html, а віднайти опис – у [6]). Тестування проводилося за допомогою програми з CD до [4], у яку були внесені такі модифікації:

- забезпечена можливість виходу зі словника в буфер під час кодування повторів;
- реалізований вибір предиктора для рядка пікселів зображення на основі оцінки нерівномірності генерованих розподілів;
- розмір блоків даних збільшений до 64 Кб та відкинуті допоміжні текстові блоки.

Результати тестування наведено в табл. 2-4, де показником компресії файлів обрано коефіцієнт стиснення (відношення розміру стиснутих даних до відповідних їм нестиснутих), виражений в bpp, тобто у кількості біт, що в середньому витрачаються для кодування одного пікселя зображення. Дані тестування програми без застосування АС наведено в другому та четвертому, а з застосуванням – у третьому та п'ятому стовпцях.

Табл. 2. Коефіцієнти стиснення (bpp) файлів зображень набору ACT у форматі PNG після застосування різних варіантів програми

Bmp-файл	Без предикторів		З предикторами	
	без AC	з AC	без AC	з AC
Clegg	5.75	5.64	5.67	5.67
Frymire	1.66	1.65	2.36	2.36
Lena	21.94	21.91	15.95	15.92
Monarch	16.96	16.94	13.20	13.18
Peppers	20.35	20.35	14.04	14.01
Sail	18.65	18.65	16.11	16.05
Serrano	1.72	1.72	2.43	2.43
Tulips	19.98	19.98	14.65	14.63
Середній	**13.38**	**13.36**	**10.55**	**10.53**
Сукупний	**9.62**	**9.60**	**8.02**	**8.01**

Табл. 3. Час кодування (с) файлів зображень набору ACT у форматі PNG різними варіантами програми на комп'ютері з частотою 300 МГц

Bmp-файл	Без предикторів		З предикторами	
	без AC	з AC	без AC	з AC
Clegg	7.52	7.20	13.07	12.79
Frymire	12.68	12.91	19.99	19.94
Lena	4.23	3.79	5.83	5.60
Monarch	5.88	5.38	10.27	10.11
Peppers	4.17	3.79	6.37	6.20
Sail	6.26	5.76	8.79	8.45
Serrano	5.11	5.11	8.13	8.08
Tulips	6.15	5.55	9.45	9.29
Разом	**52.01**	**49.49**	**81.90**	**80.46**

Табл. 4. Час декодування (с) файлів зображень набору ACT у форматі PNG різними варіантами програми на комп'ютері з частотою 300 МГц

Bmp-файл	Без предикторів		З предикторами	
	без AC	з AC	без AC	з AC
Clegg	3.46	2.80	3.62	3.24
Frymire	4.12	3.95	5.55	5.33
Lena	2.69	1.92	2.25	1.98

Прод. табл. 4.

Bmp-файл	Без предикторів		З предикторами	
	без АС	з АС	без АС	з АС
Monarch	3.29	2.47	3.08	2.69
Peppers	2.53	1.76	2.20	1.97
Sail	3.57	2.75	3.46	2.96
Serrano	1.65	1.53	2.25	2.14
Tulips	3.79	2.86	3.29	2.91
Разом	**25.10**	**20.04**	**25.70**	**23.22**

Як видно з таблиць, використання АС у програмі для зберігання зображень згідно стандарту PNG без застосування предикторів дозволило покращити коефіцієнт стиснення зображень набору ACT максимум на 0.43 %, а в середньому по набору – на 0.1 %. При цьому час кодування скоротився в середньому на 4.85 %, а декодування – на 20 %. Застосовуючи предиктори до тих самих зображень, коефіцієнт стиснення від використання АС покращився максимум на 0.26 %, а в середньому по набору – на 0.06 %; час кодування зменшився на 1.76 %, а декодування – на 10 %. Нижчі показники ефективності АС у випадку застосування предикторів пояснюються зменшенням кількості елементів, для яких виконується це кодування.

З наведених результатів дослідження та тестування можна зробити такі висновки:

1. Відхилення середньої довжини коду Хафмана від ентропії прямопропорційно залежить від двійкового логарифму частки більшої і меншої частот елементів, що поєднуються в процесі генерації цих кодів.
2. Альтернативою кодуванню Хафмана у форматах графічних файлів є байт-орієнтоване арифметичне кодування зі статичною стратегією формування інтервалів елементів. Для здійснення такого кодування, враховуючи структуру АС, у стиснутих даних необхідно забезпечити відокремлене зберігання арифметичних кодів кожного розподілу.

3. Для реалізації статичної стратегії формування інтервалів елементів АС у форматі PNG у заголовку кожного блоку стиснутих даних доцільно замість довжин кодів Хафмана зберігати кількості біт для запису довжин інтервалів, а після заголовка – двійкові коди довжин без першого біта.
4. Прискорити декодування арифметичних кодів зі статичним формуванням інтервалів на понад 40 % дозволяє використання допоміжного масиву, в якому для кожного значення загального інтервалу зберігається номер елемента, що йому відповідає.
5. Застосування АС замість кодування Хафмана дозволяє покращити, хоча й незначно, всі показники ефективності стиснення зображень у форматі PNG і тому може бути впроваджене в наступні версії цього формату.

1. *Бомба А. Я.* Алгоритм оптимізації вибору фільтра для попереднього опрацювання зображень перед стисненням на основі методу "предиктор – коректор" / А. Я. Бомба, О. В. Шпортько // Вісник Національного університету "Львівська політехніка". – 2008. – № 621. – С. 46-54. – (Серія: Інформаційні системи та мережі).
2. *Бредихин Д. Ю.* Сжатие графики без потерь качества [Електронний ресурс] / Д. Ю. Бредихин. – 2004. – http: //www.compression.ru/ download/ articles/ i_lless/ bredikhin_2004_lossless_image_compression_ doc.rar
3. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео / Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин. – М. : ДИАЛОГ-МИФИ, 2003. – 384 с.
4. *Миано Дж*. Форматы и алгоритмы сжатия изображений в действии: учеб. пособ. / Дж. Миано. – М. : Триумф, 2003. – С. 249-318. – (Практика программирования).
5. *Сэломон Д.* Сжатие данных, изображений и звука. / Д. Сэломон. – М.: Техносфера, 2006. – 368 с. – (Мир программирования: цифровая обработка сигналов)
6. *Шпортько О. В*. Алгоритми оптимізації розкладу LZ77 та вибору розмірів блоків динамічних кодів Хафмана для стиснення даних у форматі Deflate / О. В. Шпортько // Волинський математичний вісник. Серія прикладна математика. Випуск 5 (14). – Рівне: РДГУ, 2008. – С. 297-311.
7. *Шпортько О. В.* Оптимізація використання статичних предикторів у процесі стиснення зображень без втрат / О. В. Шпортько // Відбір і обробка інформації. – 2008. – № 28 (104). – С. 82-89.

8. *Ziv J.* A universal algorithm for sequential data compression / J. Ziv, A. Lempel // IEEE Transactions on Information Theory. – May 1977. – Vol. 23 (3). – P. 337-343.

Рівненський державний гуманітарний університет, м. Рівне
E-mail: abomba@ukr.net

Рівненський державний гуманітарний університет, м. Рівне
E-mail: chportko@yandex.ru
chportko@ukr.net

ДВНЗ "Рівненський коледж економіки та бізнесу", м. Рівне
E-mail: lchportko@yandex.ru

Надійшла 24.04.2010

Бомба А. Я., Шпортько А. В., Шпортько Л. В. ИСПОЛЬЗОВАНИЕ СТАТИЧЕСКОГО АРИФМЕТИЧЕСКОГО КОДИРОВАНИЯ В РАСТРОВОМ ГРАФИЧЕСКОМ ФОРМАТЕ PNG // *Проанализированы особенности контекстно-независимого кодирования в формате компрессии изображений без потерь PNG. Предложен способ и алгоритмы использования арифметического кодирования вместо кодирования Хафмана в формате словарного сжатия Deflate, который используется форматом PNG. Приведены фрагменты программ на языке С для реализации рассмотренного способа арифметического кодирования. Как свидетельствуют эксперименты, реализация предложенных алгоритмов позволяет, например, ускорить декодирование изображений набора АСТ в формате PNG на 10 - 20 %.*

Bomba A. Ya., Shport'ko A. V., Shport'ko L. V. THE USE OF STATISTIC ARITHMETIC CODE IN RASTER GRAPHIC FORMAT OF PNG // *The features of context-independent code in the format of the compression of images without the losses of PNG have been analysed. A method and algorithms of the use of arithmetic code instead of the code of Huffman in the format of dictionary compression of Deflate, which is used in the format of PNG are offered. The fragments of the programs in the language of C for the realization of the considered method of arithmetic code are offered. As experiments show, realization of the offered algorithms allows, for example, to accelerate decoding of representing of images the set of the ACT in the format of PNG in 10 - 20 %.*