УДК 004.043 https://doi.org/10.31713/vt1202221

Шпортько О. В., к.т.н., доцент (ПВНЗ «Міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука», м. Рівне, ITShportko@ukr.net)**, Шпортько Л. В., ст. викладач** (Рівненський фаховий коледж економіки та бізнесу, м. Рівне, LChportko@gmail.com), **Бомба А. Я., д.т.н., професор** (Національний університет водного господарства та природокористування, м. Рівне, abomba@ukr.net)

ФОРМУВАННЯ РОЗКЛАДУ МОДИФІКОВАНОГО АЛГОРИТМУ LZ77 У ФОРМАТІ DEFLATE З ОРІЄНТАЦІЄЮ ДОВЖИН ОДНАКОВИХ ПОСЛІДОВНОСТЕЙ НА ЦІЛІ ПІКСЕЛІ

Проаналізовано взаємовплив основних етапів стиснення зображень без втрат та формату словникової компресії Deflate. Обґрунтована доцільність перерозподілу довжин суміжних замін модифікованого алгоритму LZ77 з їх орієнтацією на цілі пікселі в процесі прогресуючого ієрархічного стиснення зображень без втрат. Встановлено, що під час першого проходу цього алгоритму з пошуком однакових послідовностей по найближчих опрацьованих раніше пікселях орієнтацію на цілі пікселі доцільно застосовувати для всіх графічних файлів, а в процесі другого проходу з пошуком однакових послідовностей в словнику – лише для дискретно-тонових зображень. Подано результати застосування різних розкладів модифікованого алгоритму LZ77 для стиснення зображень набору ACT. Показано, що застосування комбінованої орієнтації розкладу модифікованого алгоритму LZ77 на цілі пікселі дає змогу додатково зменшити коефіцієнти стиснення окремих зображень на 0.01–0.02 bpb, сповільнюючи при цьому кодування в середньому лише на 0.05 с.

***Ключові слова*: розклади словникового алгоритму LZ77; прогресуюче стиснення зображень; стиснення без втрат.**

Як відомо, обсяги даних, які міститься у файлах та передаються каналами зв'язку через локальні мережі та Інтернет [1, С. 28], систематично зростають. Їх стиснення дає змогу пропорційно підвищити швидкість обміну інформацією по мережі та зменшити обсяги використання дискового простору. Невід'ємною складовою

цих даних є мультимедійна інформація загалом та зображення, як їх частина, зокрема. Тому вдосконалення існуючих та розробка нових графічних форматів з метою покращення показників стиснення зображень є на сьогодні актуальним науковим завданням.

Усі графічні формати за принципом стиснення зображень поділяють на два основні класи: з втратами та без втрат. Стиснення зображень без втрат у графічних форматах найчастіше відбувається в три етапи: на першому яскравості компонентів пікселів перетворюються за допомогою предикторів [2; 3, С. 200; 4, С. 315–317], які не стискають зображення, але збільшують нерівномірність розподілу яскравостей і тому підвищують ефективність третього етапу; на другому – контекстно-залежне кодування зменшує надлишковості між однаковими фрагментами чи фрагментами з однаковою структурою [3, С. 81–110; 5, С. 75–119]; на третьому етапі контекстно-незалежне кодування усуває надлишковості між переважаючими значеннями яскравостей компонентів [3, С. 25–80; 5, С. 31–49]. Однакові фрагменти чи фрагменти з однаковою структурою в основному трапляються в дискретно-тонових зображеннях. Тому для таких малюнків істотно зменшує коефіцієнт стиснення (відношення розмірів стиснутого до нестиснутого файлів зображення, надалі – КС) другий етап кодування, а третій етап є єдиною універсальною фазою кодування для всіх зображень.

Основний принцип контекстно-незалежного кодування третього етапу сформулюємо так: **довжина коду довільного елемента з більшою ймовірністю не повинна перевищувати довжину коду будь-якого елемента з меншою ймовірністю**. Стосовно зображень, цей принцип базується на фундаментальному положенні теорії інформації, згідно з яким для мінімізації довжини коду послідовності кожне значення елемента *i* (яскравість окремої компоненти *brightness* (для окремої компоненти кожного пікселя зображень True Color $brightness = \overline{0,255}$) чи значення контекстно-залежного коду) з ймовірністю появи $p_i = n_i / N$ (де n_i – абсолютна частота елемента *i*, а $N = \sum_i n_i$ – загальна кількість елементів) доцільно кодувати

$$l_i = -\log_2 p_i = \log_2 \frac{N}{n_i} \qquad (1)$$

бітами [5, С. 17]. Тоді середня довжина коду елемента блоку після застосування будь-якого контекстно-незалежного алгоритму згідно з формулою of Shannon не може бути меншою *ентропії джерела* [3, С. 25–26]

$$H = -\sum_i p_i \times \log_2 p_i \ . \qquad (2)$$

Назвемо l_i *довжиною ентропійного коду* елемента *i.* Як відомо, ентропія джерела (2) зменшується зі збільшенням нерівномірності розподілу ймовірностей (частот) між елементами [2].

Одним з вдалих прикладів поєднання контекстно-залежного та контекстно-незалежного етапів кодування, затвердженого на рівні стандарту є формат словникового стиснення Deflate. Цей формат був розроблений Філом Кацем в 1993 році для другої версії архіватора PKZIP [5, С. 94–95] та формалізований в RFC 1951 [6] у 1996 році. Формат Deflate для опрацювання вхідного потоку використовує контекстно-залежний словниковий алгоритмом LZ77 [7], а результати його роботи для забезпечення найменших КС стискає динамічними кодами Хафмана [8]. На сьогодні цей формат використовується також в багатьох популярних архіваторах (наприклад, GZIP) та графічних форматах (наприклад, PNG [4, С. 249–318]) і не потребує придбання ліцензій для використання в прикладному ПЗ.

Алгоритм LZ77 як складова частина формату Deflate базується на заміні під час кодування у вихідному потоці однакової послідовності для *буфера* посиланням на аналогічну вже закодовану послідовність *словника* у вигляді пари чисел *<довжина однакової послідовності, зміщення від кінця словника до попередньої однакової послідовності>.* Якщо для елементів з початку буфера віднайти однакову послідовність в словнику не вдається, то перший елемент (*літерал*) з початку буфера записується в кінець словника та у вихідний потік без змін і кодування продовжується аналогічно з наступного елемента буфера. У форматі Deflate довжини замін та окремі літерали алгоритму LZ77 кодуються разом числами в межах [0; 285]. При цьому числа з діапазону [0; 255] відповідають кодам окремих літералів, 256 позначає закінчення блоку, а числа з діапазону [257; 285] вказують на базові значення довжин. Після базових значень довжин міститься визначена форматом додаткова кількість бітів (до п'яти), що разом з базовим значенням однозначно визначає довжину заміни [5, С. 98]. Зміщення зберігається відразу

після відповідної довжини заміни аналогічно – у вигляді базового значення та додаткових бітів (до 13). Базове значення зміщення знаходиться в межах [0; 29] [5, С. 99].

Зрозуміло, що однакових послідовностей різної довжини для початку буфера у словнику може бути декілька, тому й альтернативних розкладів алгоритму LZ77 можливо сформувати чимало і серед цих розкладів доцільно обирати той, що забезпечує мінімальний КС. Але аналізувати ефективність всеможливих розкладів алгоритму LZ77 в процесі стиснення зображень недоцільно через обмеження на час кодування. Тому на практиці використовують один з поширених варіантів формування розкладу алгоритму LZ77. Охарактеризуємо найпопулярніші з таких розкладів, які ми опосередковано використали в цьому дослідженні:

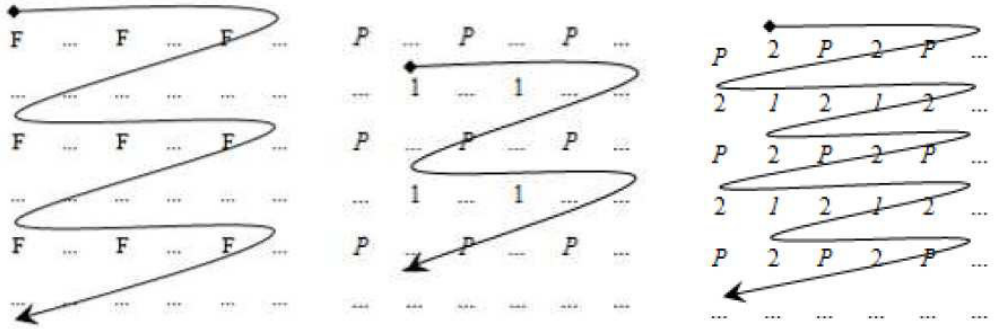
1. **«Жадібний» розклад** алгоритму LZ77 базується на використанні на кожному кроці заміни <*довжина*; *зміщення*> найбільшої довжини. Природно, що цей розклад забезпечує найшвидше стиснення, але й практично завжди – високі КС [5, С. 106], оскільки не аналізує альтернативні варіанти. Ми також використовуємо «жадібний» розклад для найшвидшого кодування зображень.

2. **«Лінивий» розклад** (lazy parsing) базується на відмові від використання віднайденої заміни <*довжина*; *зміщення*> з чергової позиції потоку, якщо з наступної позиції вдається віднайти довшу заміну. Але довжина контекстно-незалежного коду літерала з чергової позиції потоку може перевищувати довжину його коду в заміні, якщо цей літерал зустрічається рідко. Крім цього, «лінивий» розклад може ітеративно породжувати послідовності літералів, які могли б входити в окремі заміни [5, С. 104–105]. Тому в цій роботі нами використовувався лише підхід формування цього розкладу для реалізації орієнтації довжин замін на цілі пікселі.

Водночас на сьогодні обхід яскравостей пікселів зображень у популярних графічних форматах, які виконують стиснення без втрат, як правило здійснюється послідовно по рядках зверху вниз, а у кожному рядку – поспіль зліва направо. Тому вивести стиснуте зображення у цих форматах можливо лише після завершення декодування, а декомпресія їх файлів з мільйонами пікселів при такому способі обходу може тривати декілька секунд незалежно від розміру області чи роздільної здатності пристрою виводу. Для прискорення ж виводу великих зображень у форматах компресії з

втратами вже застосовують прогресуюче (поступальне) ієрархічне опрацювання пікселів [3, С. 176] (наприклад вейвлети [9]). Під час такого способу опрацювання пікселі зображення обходять пошарово, збільшуючи щоразу роздільну здатність (прогресуюча складова), і в процесі послідовної обробки яскравостей пікселів чергового шару використовують дані попередніх шарів (ієрархічна складова). Зупинити таке декодування в процесі прогресуючого ієрархічного обходу можливо вже після декомпресії шару з кількістю пікселів, не меншою від області виводу по кожній з осей, не очікуючи відтворення всього зображення.

Для реалізації прогресуючого ієрархічного стиснення зображень без втрат в [10] нами було запропоновано відповідні предиктори та схему обходу, за якою на першому шарі пікселі зображення опрацьовуються послідовно, починаючи з першого у верхньому лівому куті, по рядках зверху вниз, а у кожному рядку – підряд зліва направо з кроком $h_1 = 2^k$, де *k* визначається з умови $k = \left\lceil \log_2 \left(\frac{\max\left(\min\left(height; width\right); 16\right) - 1}{15} \right) \right\rceil$, *height* – кількість рядків, *width* – кількість стовпців пікселів зображення (рис. 1, а). На наступних шарах ($l = \overline{2, k+1}$) проміжні пікселі зображення обробляються в два проходи: на першому послідовно опрацьовуються ті з них, які містяться на перетині діагоналей квадратів з вершинами у суміжних пікселях попередніх шарів з кроком $h_l = 2^{k+2-l}$ як по рядках, так і по стовпцях (див. рис. 1, б), а на другому необроблені пікселі послідовно обходяться між суміжними пікселями попередніх шарів і пікселями першого проходу з тим самим кроком по стовпцях і з вдвічі зменшеним – по рядках (див. рис. 1, в). На рис. 1 символом *F* позначені пікселі першого шару, символом *P* – пікселі попередніх шарів, цифрою *1* – пікселі першого проходу чергового шару, цифрою *2* – пікселі другого проходу чергового шару.



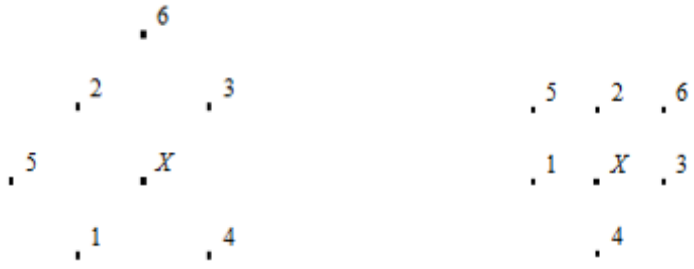
а) б) в)

Рис. 1. Черговість обходу пікселів в процесі прогресуючого ієрархічного опрацювання: а) пікселів першого шару; б) пікселів першого проходу чергового шару; в) пікселів другого проходу чергового шару

Для підвищення ефективності застосування алгоритму LZ77 в процесі прогресуючого ієрархічного стиснення зображень без втрат на всіх шарах, починаючи з другого, в [11] ми обґрунтували доцільність виконання пошуку однакових послідовностей для елементів буфера не лише у словнику, **а й починаючи з найближчих раніше опрацьованих пікселів з кроком чергового шару**. Оскільки серед раніше опрацьованих пікселів черговий піксель *X* має найвищий рівень кореляції саме з найближчими серед них, то найменші 6 зміщень (з кодами від 1 до 6) ми закріпили за найближчими опрацьованими раніше пікселями (рис. 2), а для забезпечення однозначності декодування коди зміщень в словнику збільшили на 6.

З метою додаткового зменшення КС зображень у цій статті ми наведемо власний варіант формування розкладу алгоритму LZ77 у форматі Deflate в процесі прогресуючого ієрархічного стиснення RGB-зображень з орієнтацією довжин замін на цілі пікселі та дослідимо ефективність його застосування.

Для виявлення можливостей зменшення КС модифікованого алгоритму LZ77 в процесі прогресуючого ієрархічного стиснення зображень без втрат шляхом оптимізації його розкладу проаналізуємо розподіл частот коротких замін різної довжини, які були використані нами для стиснення зображень набору Archive Comparison Test (http://www.compression.ca/act/act-files.html, надалі ACT) в процесі «жадібного» розкладу цього алгоритму (табл. 1). Цей набір містить 3 синтезованих (№№ 1 (з шумами), 2, 7), та



а) б)

Рис. 2. Коди зміщень до яскравостей суміжних опрацьованих раніше пікселів на шарах, починаючи з другого: а) для першого проходу; б) для другого проходу

5 фотореалістичних (решта) зображень. В табл. 1 наведені дані для довжин замін алгоритму LZ77 до 10 елементів включно (такі заміни назвемо **короткими)**, оскільки у форматі Deflate довші заміни кодуються однаковими груповими кодами з суміжними значеннями [5, С. 98] і їх перерозподіл істотно не впливає на КС.

Таблиця 1

Кількості коротких замін різної довжини, використаних для стиснення зображень набору ACT в процесі «жадібного» розкладу алгоритму LZ77

Довжина заміни	№ файла								В середньому по набору
	1	2	3	4	5	6	7	8	
3	17443	54673	44	57155	7660	91270	19929	33689	35233
4	3562	3864	11	9430	1763	8798	5058	5886	4797
5	772	2777	6	3463	765	3612	2275	1873	1943
6	497	23509	2	10512	711	5401	7917	3208	6470
7	112	1861	1	1470	164	247	1465	684	751
8	28	1416	0	726	61	80	687	285	410
9	471	18739	0	4183	48	179	5861	599	3760
10	239	1438	0	432	13	17	927	147	402
Разом:	**23124**	**108277**	**64**	**87371**	**11185**	**109604**	**44119**	**46371**	**53764**

Як слідує з табл. 1, короткі заміни найчастіше зустрічаються в дискретно-тонових зображеннях та фотореалістичних знімках з багатьма середніми чи малими об'єктами. Також бачимо, що **серед суміжних довжин коротких замін алгоритму LZ77 найчастіше зустрічаються довжини, кратні цілим пікселям** (3, 6 чи 9 елементів). І це не дивно, адже в зображеннях найчастіше повторюються

кольори цілих пікселів, а не кольори пікселів і суміжних компонентів у відповідності до послідовності їх обходу. Тому після застосування контекстно-незалежного алгоритму, враховуючи (1), ентропійні коди довжин замін, кратних цілим пікселям, будуть в середньому суттєво коротші від ентропійних кодів суміжних довжин. Наприклад, для зображення frymire.bmp (№ 2 в наборі АСТ) середня довжина замін з трьох елементів буде меншою від довжини замін з чотирьох елементів на log(54673)-log(3864)≈3.82 біта. Тому, якщо в цьому зображенні код першого чи останнього елемента заміни з чотирьох елементів коротший 3.82 біта, то для зменшення КС навіть доцільно закодувати такий елемент окремо, а заміну скоротити до трьох елементів. Але такі ситуації трапляються в зображенні порівняно рідко і тому цей підхід зменшує розміри стиснутих зображень лише на десятки байтів.

Суттєвіше зменшити КС зображень можливо, якщо для обрізаних елементів короткої заміни алгоритму LZ77 (заміни з довжинами 4 чи 5 обрізаються до 3 елементів, з довжинами 7 чи 8 – до 6 елементів, заміни довжиною 10 – до 9 елементів) вдається віднайти суміжну заміну. Якщо ж суміжну заміну алгоритму LZ77 для обрізаних елементів віднайти не вдалося, то ці елементи доцільно почергово повертати до попередньої заміни, щоразу повторюючи пошук суміжної заміни для залишених обрізаних елементів при їх наявності. Наприклад, на рис. 3 схематично відмічено заміну довжиною 5 елементів (верхня частина), яку обрізано до пікселя (нижня частина), а її два обрізаних елементи включено до наступної заміни.

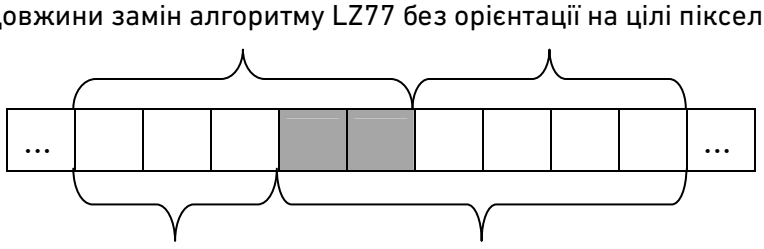


Рис. 3. Зміни довжин двох суміжних замін алгоритму LZ77 внаслідок скорочення першої заміни до довжини пікселя (обрізані елементи виділені на сірому фоні)

Тут дві суміжних заміни до орієнтації на цілі пікселі мали довжини 5 і 4 елементи, а після орієнтації – 3 і 6 елементів відповідно. Змінені довжини в середньому мають більші частоти і тому, враховуючи (1), закодуються меншою кількістю бітів контекстно-незалежним алгоритмом.

З іншого боку, послідовна обрізка суміжних замін може збільшити кількість замін меншої довжини, які за результатами аналізу замін Deflate-блоку виявляться неефективними і будуть відкинуті. Наприклад, замість трьох замін по чотири елементи може бути сформовано чотири заміни по три елементи. Тому суміжну заміну, яка покриває обрізані елементи, будемо вважати **прийнятною** тоді, коли крім обрізаних елементів вона покриває ще й принаймні два наступні елементи.

Описаний підхід формування розкладу модифікованого алгоритму LZ77 з орієнтацією довжин замін на цілі пікселі по суті аналогічний механізму формування вже згаданого «лінивого» розкладу, але перший з цих розкладів в найгіршому випадку шукає додаткові заміни <*довжина*; *зміщення*>, починаючи лише з обрізаних елементів, а другий – з кожного елемента заміни.

Запропонована нами в [11] модифікація алгоритму LZ77 реалізується в два проходи: на першому виконується пошук замін по найближчих опрацьованих раніше пікселях (див. рис. 2), а на другому – віднаходження замін по словнику в основному для тих компонентів пікселів, які не увійшли в заміни першого проходу. Відповідно, орієнтацію довжин замін на цілі пікселі слід виконувати в кожному з цих проходів окремо.

Позначимо через *trimBestLength* кількість обрізаних елементів попередньої заміни, а через *bestLength* – довжину кращої віднайденої суміжної заміни з поточної позиції потоку, яка завжди перевищує 2 згідно обмежень формату Deflate. Початкове значення *trimBestLength* – це остача від ділення довжини попередньої заміни на 3, яка не перевищує 2. Тому, враховуючи ці обмеження, суміжна заміна буде неприйнятною, коли *bestLength*-*trimBestLength*=1 (тобто *bestLength*=3 і *trimBestLength*=2). З врахуванням цих позначень та обмежень фрагмент програми мовою C++ для пошуку замін по найближчих опрацьованих раніше компонентах під час першого проходу модифікованого алгоритму LZ77 з орієнтацією довжин замін на цілі пікселі може бути таким:

```
// шукаємо найдовше співпадіння по суміжних пікселях
LongestAdjacentMatch(bestLength, bestOffset);
// якщо знайдено заміну і вона перекриває крім обрізаних елементів
// попередньої заміни не менше двох суміжних елементів (є прийнятною)
if (bestLength>0 && (bestLength-trimBestLength!=1))
 {if (LZLazyParsing && bestLength<=maxLenRoundPixelReplaceLiteral &&
     bestLength%3!=0) // якщо виконується орієнтування на пікселі і заміна
     // коротка і є що обрізати – обрізаємо знайдену заміну до пікселя
  {trimBestLength=bestLength%3; // кількість обрізаних елементів
   bestLength-=trimBestLength; // зменшуємо довжину заміни
   prevBestOffset=bestOffset; } // запам'ятовуємо зміщення заміни
  else trimBestLength=0; // інакше заміна не обрізається
  long adres=j*row_width+3*i+k; // адреса компоненти чергового пікселя
  for (i1=0; i1<bestLength; i1++) // зберігаємо зміщення віднайденої заміни
   {offsetAdjacentPixel[adres]=bestOffset;
    if (adres%3==2) adres+=3*step-2; // перехід до наступного пікселя
    else adres++; }} // перехід до наступної компоненти
else // заміна не знайдена
 // якщо є обрізані елементи попередньої заміни
 if (LZLazyParsing && trimBestLength>0)
  {offsetAdjacentPixel[j*row_width+3*i+k]=prevBestOffset; // то розширюємо її
   trimBestLength--; } // зменшуємо кількість обрізаних елементів
```

А фрагмент програми для пошуку замін по словнику з врахуванням замін першого проходу під час другого проходу формування розкладу модифікованого алгоритму LZ77 з орієнтацією довжин замін на цілі пікселі схематично може мати такий вигляд:

```
// шукаємо найдовше співпадіння по словнику та суміжних пікселях
LongestMatch(bestLength, bestOffset);
// якщо заміна відсутня або є неприйнятною суміжною заміною
if (bestLength==0 || (bestLength-trimPrevBestLength==1))
 {... // переносимо поточну компоненту з буфера в словник
  if (trimPrevBestLength==0) // обрізаних елементів з попередньої заміни немає
   {// записуємо у стиснуті дані поточний літерал
    UBYTE1 literal=imageDataPredict[currentPosImage];
    block_buffer[block_buffer_count]=literal;
    length_table->IncrementFrequency(literal); }
  else // повертаємо перший з обрізаних елементів в попередню заміну
   {// в розподілі літералів/довжин збільшуємо довжину попередньої заміни
    UBYTE2 lenPrev=block_buffer [block_buffer_count-2]-256;
    // зменшуємо частоту базового значення попередньої довжини заміни
    LengthToCode(lenPrev, code, extra, value);
    length_table->freq[code]--;
    lenPrev++; // інкрементуємо довжину попередньої заміни
    // зберігаємо збільшену довжину в стиснутих даних та частотах
    block_buffer[block_buffer_count-2]=lenPrev+256;
    LengthToCode(lenPrev, code, extra, value);
```

```
   length_table->IncrementFrequency(code);
   block_buffer_count--; // довжина стиснутих даних не збільшилася
   trimPrevBestLength--; } // зменшуємо кількість обрізаних елементів
  currentPosImage++; } // переходимо до наступної позиції
else // заміну знайдено
 {//якщо виконується орієнтування на пікселі і заміна коротка і є що
  //обрізати та куди записати – обрізаємо знайдену заміну до пікселя
  if (LZLazyParsing && bestLength<=maxLenRoundPixelReplaceLiteral &&
     bestLength%3!=0 && block_buffer_count<block_buffer_size-4)
   {trimPrevBestLength=bestLength%3; // кількість обрізаних елементів
    bestLength-=trimPrevBestLength; } // зменшуємо довжину заміни
  else
   trimPrevBestLength=0; // поточна заміна не обрізається
  // збільшуємо частоти базових значень довжини та зміщення заміни
  LengthToCode(bestLength, code, extra, value);
  length_table->IncrementFrequency(code);
  DistanceToCode(bestOffset, codeD, extra, value);
  distance_table->IncrementFrequency (codeD) ;
  // зберігаємо довжину та зміщення знайденої заміни в стиснутих даних.
  block_buffer [block_buffer_count] = 256 + bestLength;
  ++block_buffer_count ;
  block_buffer [block_buffer_count] = bestOffset;
  // модифікуємо хеш-ланцюги зі всіх літералів заміни
  for (unsigned int ii=0; ii<bestLength ; ii++)
   {... // переносимо поточну компоненту з буфера в словник
    currentPosImage++; }}
```

Проаналізуємо результати застосування різних варіантів розкладу модифікованого алгоритму LZ77 з орієнтацією довжин замін на цілі пікселі (табл. 2) на прикладі прогресуючого ієрархічного стиснення зображень набору ACT. Час декодування файлів, отриманих внаслідок застосування цих варіантів, практично не змінюється, оскільки розміри отриманих стиснутих зображень відрізняються несуттєво.

Таблиця 2

Коефіцієнти стиснення зображень набору ACT після застосування різних варіантів розкладу модифікованого алгоритму LZ77 з орієнтацією довжин замін на цілі пікселі, bpb

Варіант розкладу модифікованого алгоритму LZ77	№ файла								Середній КС
	1	2	3	4	5	6	7	8	
Без орієнтації на цілі пікселі	1.34	0.58	4.65	3.82	4.15	5.16	0.62	4.33	3.08
З орієнтацією на цілі пікселі під час першого проходу	1.34	0.57	4.65	3.81	4.15	5.16	0.62	4.32	3.08
З орієнтацією на цілі пікселі під час другого проходу	1.34	0.58	4.65	3.82	4.15	5.16	0.61	4.33	3.08
З орієнтацією на цілі пікселі під час двох проходів	1.34	0.57	4.65	3.81	4.15	5.16	0.60	4.33	3.08
З комбінованою орієнтацією на цілі пікселі	1.34	0.57	4.65	3.81	4.15	5.16	0.60	4.32	3.08
З комбінованою орієнтацією на цілі пікселі та розрахунком прогнозованих довжин кодів	1.34	0.56	4.65	3.81	4.15	5.15	0.60	4.32	3.07

Бачимо, що орієнтація довжин замін на цілі пікселі під час першого проходу модифікованого алгоритму LZ77 з пошуком замін по найближчих опрацьованих раніше компонентах дає змогу зменшити КС трьох зображень набору з восьми на 0.01 bpb, причому зменшення спостерігається як для дискретно-тонових, так і для фотореалістичних зображень. Така ж орієнтація в процесі другого проходу цього алгоритму з пошуком замін по словнику аналогічно зменшує КС лише для дискретно-тонового зображення № 7 і гірше проявляє себе на фотореалістичних зображеннях. І це не дивно, адже заміни по словнику мають більші зміщення від замін по найближчих опрацьованих пікселях і тому кодуються в середньому більшою кількістю бітів та можуть бути в подальшому відкинуті, як неефективні. Крім цього, орієнтація довжин замін на цілі пікселі під час першого проходу сповільнює кодування в середньому на 0.04 с, а під час другого – на 0.07 с.

Орієнтація довжин замін на цілі пікселі під час двох проходів зменшує КС трьох зображень з восьми на 0.01–0.02 bpb, аналогічно першому проходу, але погіршує відносно нього КС для фотореалістичного зображення № 8 та додатково сповільнює кодування в середньому на 0.01 с. Тому на практиці ми рекомендуємо використовувати **комбіновану орієнтацію довжин замін на цілі пікселі в процесі формування розкладу модифікованого алгоритму LZ77**:

1. Застосувати таку орієнтацію під час першого проходу з пошуком замін по найближчих опрацьованих раніше пікселях;

2. Визначити частку пікселів зображення, включених в заміни першого проходу.

3. Якщо частка пікселів, включених в заміни першого проходу, менше 80 % (наш критерій визначення того, що зображення не дискретно-тонове), то відмовитися від орієнтації на цілі пікселі в процесі другого проходу алгоритму LZ77.

Така комбінована орієнтація довжин замін алгоритму LZ77 на цілі пікселі (п'ятий рядок табл. 2) зменшує КС 50% зображень набору ACT на 0.01–0.02 bpb і сповільнює кодування в середньому лише на 0.05 с. Вплив цієї модифікації розкладу алгоритму LZ77 на решту зображень набору обмежується сотнями байтів і тому не впливає на показники табл. 2. Додатковий розрахунок прогнозованих довжин кодів розподілу літералів/довжин формату Deflate в процесі комбінованої орієнтації довжин замін дає змогу зменшити середні КС зображень набору ACT до 3.07 bpb (останній рядок табл. 2), хоча й сповільнює кодування у середньому ще на 0.17 с.

З наведених результатів дослідження приходимо до таких висновків:

1. Підвищити ефективність застосування класичного алгоритму LZ77 з пошуком по словнику в процесі прогресуючого ієрархічного стиснення зображень без втрат можливо за рахунок додаткового попереднього проходу пікселів зображення з пошуком однакових послідовностей, починаючи з найближчих опрацьованих раніше пікселів. Такий додатковий прохід дає змогу зменшити КС за рахунок однакових послідовностей, невключених в словник, та використання менших зміщень для найближчих опрацьованих пікселів.
2. Словникові алгоритми суттєво зменшують КС насамперед штучних дискретно-тонових зображень, оскільки такі зображення містять

багато однакових суміжних послідовностей яскравостей компонентів пікселів.

3. Додатково зменшити КС модифікованого алгоритму LZ77 для окремих зображень максимум на 0.02 bpb можливо шляхом орієнтації його розкладу на цілі пікселі. При цьому під час першого проходу даного алгоритму таку орієнтацію доцільно застосовувати для всіх, а під час другого – лише для дискретно-тонових зображень. Це дає змогу не збільшувати КС для всіх зображень і сповільнює кодування в середньому лише на 0.05 с.

Надалі, з метою додаткового зменшення розмірів файлів стиснутих зображень і прискорення декодування, ми плануємо вдосконалити алгоритм зменшення розміру сформованих Deflate-блоків [10] та пристосувати його до прогресуючого ієрархічного стиснення зображень без втрат.

1. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms, Third Edition. Kiyv : Dialektika, 2020. Vol. 1. 648 p. **2.** Shportko A., Postolatii V. Development of Predictors to Increase the Efficiency of Progressive Hierarchic Context-Independent Compression of Images Without Losses. *Proceedings of the 5th International Conference on Computational Linguistics and Intelligent Systems (COLINS 2021)* (Kharkiv, 22–23 apr. 2021). Volume I. ceur-ws.org, P. 1026–1038. **3.** Сэломон Д. Сжатие данных, изображений и звука. Москва : Техносфера, 2006. 368 с. **4.** Миано Дж. Форматы и алгоритмы сжатия изображений в действии. Москва : Триумф, 2003. 336 с. **5.** Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. Москва : ДИАЛОГ-МИФИ, 2003. 384 с. **6.** Deutsch P. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951. Alladin enterprises, 1996. 15 p. **7.** Ziv J., Lempel A. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*. May 1977. Vol. 23(3). P. 337–343. **8.** Huffman D. A Method for the Construction of Minimum Redundancy Codes. *Proceedings of the IRE*. Sept. 1952. Vol. 40(9). P. 1098–1101. **9.** Уэлстид С. Фракталы и вейвлеты для сжатия изображений в действии. Москва : Триумф, 2003. 320 с. **10.** Шпортько О. В. Оптимізація застосування модифікованого формату словникової компресії Deflate у процесі прогресуючого ієрархічного стиснення зображень без втрат. *Науковий вісник Чернівецького національного університету імені Юрія Федьковича. Сер. Комп'ютерні системи та компоненти*. 2013. Вип. 4. Т. 4. С. 40–52. **11.** Шпортько О. В., Шпортько Л. В., Бомба А. Я. Підвищення ефективності застосування словникових методів компресії для прогресуючого ієрархічного стиснення зображень без втрат. *Вісник*

Національного університету водного господарства та природокористування. Сер. Технічні науки. Рівне : НУВГП, 2021. Вип. 4 (96). С. 130–145.

REFERENCES:

1. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms, Third Edition. Kiyv : Dialektika, 2020. Vol. 1. 648 p. **2.** Shportko A., Postolatii V. Development of Predictors to Increase the Efficiency of Progressive Hierarchic Context-Independent Compression of Images Without Losses. *Proceedings of the 5th International Conference on Computational Linguistics and Intelligent Systems (COLINS 2021)* (Kharkiv, 22–23 apr. 2021). Volume I. ceur-ws.org, P. 1026–1038. **3.** Selomon D. Szhatie dannykh, izobrazheniy i zvuka. Moskva : Tekhnosfera, 2006. 368 p. **4.** Miano Dj. Formatyi i algoritmyi sjatiya izobrajeniy v deystvii. Moskva : Triumf, 2003. 336 s. **5.** Vatolin D., Ratushnyak A., Smirnov M., YUkin V. Metodyi sjatiya dannyih. Ustroystvo arhivatorov, sjatie izobrajeniy i video. Moskva : DIALOG-MIFI, 2003. 384 s. **6.** Deutsch P. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951. Alladin enterprises, 1996. 15 p. **7.** Ziv J., Lempel A. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*. May 1977. Vol. 23(3). P. 337–343. **8.** Huffman D. A Method for the Construction of Minimum Redundancy Codes. *Proceedings of the IRE*. Sept. 1952. Vol. 40(9). P. 1098–1101. **9.** Uelstid S. Fraktalyi i veyvletyi dlya sjatiya izobrajeniy v deystvii. Moskva : Triumf, 2003. 320 s. **10.** Shportko O. V. Optymizatsiia zastosuvannia modyfikovanoho formatu slovnykovoi kompresii Deflate u protsesi prohresuiuchoho iierarkhichnoho stysnennia zobrazhen bez vtrat. *Naukovyi visnyk Chernivetskoho natsionalnoho universytetu imeni Yuriia Fedkovycha. Ser. Kompiuterni systemy ta komponenty*. 2013. Vyp. 4. T. 4. S. 40–52. **11.** Shportko O. V., Shportko L. V., Bomba A. Ya. Pidvyshchennia efektyvnosti zastosuvannia slovnykovykh metodiv kompresii dlia prohresuiuchoho iierarkhichnoho stysnennia zobrazhen bez vtrat. *Visnyk Natsionalnoho universytetu vodnoho hospodarstva ta pryrodokorystuvannia. Ser. Tekhnichni nauky*. Rivne : NUVHP, 2021. Vyp. 4 (96). S. 130–145.

Shportko A. V., Candidate of Engineering (Ph.D.), Associate Professor (Academician Stepan Demianchuk International University of Economics Humanities, Rivne), **Shportko L. V., Senior Lecturer** (Vocational College of Economics and Business, Rivne)**, Bomba A. Ya., Doctor of Engineering, Professor** (National University of Water and Environmental Engineering, Rivne)

FORMATION OF THE DECOMPOSITION OF THE MODIFIED LZ77 ALGORITHM IN DEFLATE FORMAT WITH ORIENTATION OF LENGTHS OF THE SAME SEQUENCES ON WHOLE PIXELS

The article analyzes the interaction of the main stages of lossless image compression and the format of Deflate dictionary compression. The advantages and disadvantages of «greedy» and «lazy» schedules of the LZ77 algorithm as part of the Deflate format are described. The expediency of redistribution of adjacent substitution lengths of the modified LZ77 algorithm with their orientation on whole pixels in the process of progressive hierarchical compression of lossless images is substantiated. It is established that during the first pass of this algorithm with search of identical sequences of the nearest previously processed pixels the orientation on whole pixels should be applied to all graphic files, and during the second pass with the search of identical sequences in the dictionary – only for discrete images. The results of the application of different schedules of the modified LZ77 algorithm for image compression of the ACT set are presented.

According to the results of the study, the following conclusions were made:

1. It is possible through additional pre-passage of image pixels in search of identical sequences, starting from the nearest previously processed pixels to increase the efficiency of the classical algorithm LZ77 with dictionary search in the process of progressive hierarchical compression of lossless images. This additional pass allows to reduce the compression ratios due to the same sequences which are not included in the dictionary, and the use of smaller offsets for the nearest processed pixels.

2. Vocabulary algorithms reduce the compression ratios of primarily artificial discrete-tone images significantly, as such images contain many of the same adjacent sequences of brightness of the

pixel components.

3. It is possible to further reduce the compression ratio of the modified LZ77 algorithm for individual images by a maximum of 0.02 bpb by combining its decomposition into whole pixels. This orientation allows not to increase the compression ratios for all images and slows down the encoding by an average of only 0.05 s.

***Keywords*: schedules LZ77 dictionary algorithm; progressive image compression; lossless compression.**
