РІВНЕНСЬКИЙ ДЕРЖАВНИЙ ГУМАНІТАРНИЙ
УНІВЕРСИТЕТ

ВОЛИНСЬКИЙ МАТЕМАТИЧНИЙ ВІСНИК

СЕРІЯ
ПРИКЛАДНА МАТЕМАТИКА

Збірник наукових праць

Випуск 5 (14)

Рівне-2008

"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 № 4 від 28.11.2008 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

Хроніка

УДК 004.043

Шпортько О.В.

АЛГОРИТМИ ОПТИМІЗАЦІЇ РОЗКЛАДУ LZ77 ТА ВИБОРУ РОЗМІРІВ БЛОКІВ ДИНАМІЧНИХ КОДІВ ХАФМАНА ДЛЯ СТИСНЕННЯ ДАНИХ У ФОРМАТІ DEFLATE

Проаналізовано особливості кодування даних, обґрунтовано можливості та описано алгоритми оптимізації розкладу LZ77 і вибору розмірів блоків динамічних кодів Хафмана для формату DEFLATE. Наведено фрагменти програм для реалізації запропонованих алгоритмів та результати їх застосування для стиснення зображень у форматі PNG.

Інформація, що міститься у файлах та передається каналами зв'язку, як правило, має високий рівень надлишковості. Стиснення даних дозволяє пропорційно підвищити швидкість обміну інформацією по мережі та зменшити обсяги використання дискового простору. На сьогоднішній день існує декілька основних методів та безліч алгоритмів стиснення даних [3]. Переважна більшість архіваторів та споріднених програм використовують алгоритми, що комбінують ідеї основних методів. Найчастіше серед них зустрічаються словникові алгоритми стиснення даних. Існує два різновиди алгоритмів словникового методу [1, 2]. Перший з них бере початок від методу LZ77 і базується на заміні послідовності чергових елементів потоку посиланням на аналогічну закодовану послідовність елементів у вигляді пари чисел *<довжина; зміщення від закінчення закодованої частини потоку>*, якщо така послідовність зустрічалася раніше. Другий бере початок від методу LZ78 і базується на заміні послідовності чергових елементів потоку індексом аналогічної послідовності у словнику, що формується під час виконання алгоритму. Оскільки словникові алгоритми використовують опрацьований раніше текст, то вони належать до класу контекстно-залежних. На практиці, для отримання кращих показників стиснення даних результати дії словни-

кового алгоритму додатково стискають одним з контекстно-незалежних алгоритмів (на основі кодів Хафмана, арифметичного стиснення чи інших).

Саме такий підхід реалізовано у форматі стиснення даних DEFLATE, де результати дії словникового алгоритму LZ77 стискаються контекстно-незалежними кодами Хафмана (ідея використання цих кодів полягає у заміні елементів з більшою частотою послідовностями меншої кількості біт, ніж для елементів з меншою частотою).

Розглянемо детальніше особливості стиснення словникового методу LZ77 на основі найуживанішого *жадібного* розкладу вхідної послідовності [3]. Нехай послідовність елементів потоку $s_1 \dots s_{j-1}$ вже закодована і занесена в словник. Тоді на черговому кроці алгоритму для послідовності незакодованих елементів буфера $s_j s_{j+1} \dots$ шукається співпадаюча послідовність $s_i s_{i+1} \dots$ максимальної довжини *len* (саме тому такий розклад називається *жадібним*), що бере початок у закодованій частині (словнику) послідовності $(i<j)$. При виявленні співпадаючої послідовності, $s_j \dots s_{j+len-1}$ кодується парою чисел $<len; j-i>$, послідовність $s_j \dots s_{j+len-1}$ заноситься в словник та виконується перехід до кодування потоку, починаючи з елемента s_{j+len} (очевидно, що для досягнення стиснення *len* має перевищувати 2). Якщо ж співпадаюча послідовність в словнику відсутня, то елемент s_j заноситься в закодовані дані та словник без змін і виконується перехід до кодування потоку, починаючи з наступного елемента s_{j+1}. Кодування припиняється після опрацювання останнього елемента. Максимальні розміри словника та буфера встановлюються конкретним алгоритмом. Сукупність словника з закодованими символами та буфера з незакодованими ще називають *ковзаючим вікном* [3], оскільки вони весь час синхронно переміщуються по елементах потоку.

Кращі на декілька відсотків результати стиснення LZ77 стосовно *жадібного* розкладу при істотному збільшенні часу виконання отримуються за допомогою алгоритму майже оптимального розкладу [3]. Ідея цього алгоритму полягає в обчисленні мінімальної вартості (кількості біт) кодування потоку від початку до кожного чергового елемента. Згід-

но алгоритму

$$V_j = \min_{i<j}\left(V_i + xord_{i+1,j}\right), \qquad (1)$$

де V_k – мінімальна вартість кодування потоку від початку до *k*-го елемента включно, $xord_{i+1,j}$ – мінімальна вартість зберігання пари *<довжина; зміщення>* для кодування елементів з *i+1* по *j*-й. На перший погляд, цей алгоритм описується за допомогою чотирьох вкладених циклів по елементах потоку: перший – для елементів *j*, стосовно яких визначається мінімальна вартість кодування, другий – для елементів *i*, згідно яких визначається мінімальна вартість кожного елемента, третій – по співпадаючих послідовностях, четвертий – по елементах цих послідовностей. Але якщо врахувати, що значення $xord_{i+1,j}$ існують лише для тих елементів, між якими можливе кодування парою *<довжина; зміщення>*, то зовнішній цикл можна відкинути, аналізуючи лише **можливі** вартості кодувань від кожного *i*-го символа до наступних. Після визначення мінімальної вартості кодування всього потоку відбираються позиції та зміщення, що дозволяють досягнути цієї мінімальної вартості, і саме з них формується закодований потік.

При декодуванні окремі елементи копіюються у вихідний потік та в буфер без змін. Пари ж *<довжина; зміщення>* декодуються шляхом послідовного копіювання з кінця буфера за вказаним зміщення в початок буфера та вихідний потік необхідної кількості елементів. Природно, що алгоритм декодування має розрізняти окремі елементи та групи *<довжина; зміщення>*. У форматі DEFLATE з цією метою довжини та елементи кодуються одним числом в межах [0; 285]. Коди з діапазону 0-255 відповідають кодам окремих елементів, 256 позначає закінчення потоку, а діапазон 257-285 містить базові значення довжин. Після базових значень довжин вказується визначена форматом кількість біт, що разом з базовим значенням однозначно визначає довжину. Зміщення зберігається аналогічно довжині – у вигляді базового значення та додаткових біт. Базове значення довжини знаходиться в межах [0; 29]. В цьому форматі максимальне значення довжини закодованої послідовності

може досягати 285, а зміщення – 32768.

Для кодування елементів/довжин та зміщень у форматі DEFLATE використовуються дві незалежні таблиці Хафмана, оскільки ці величини мають різні частоти та різні діапазони можливих значень. Кожна пара таких таблиць діє в межах окремого блоку даних і містить статичні чи динамічні коди Хафмана [3, 4]. Статичні коди визначаються форматом і не дозволяють оптимально стиснути дані, оскільки не враховують їх особливостей. Динамічні коди Хафмана дозволяють оптимальніше стиснути дані, але сповільнюють процес кодування, оскільки вимагають попереднього додаткового проходу по даних для збору статистики використання кодів елементів. Крім цього, на початку кожного блоку стиснутих даних за допомогою динамічних кодів Хафмана міститься заголовок, що дозволяє декодеру згенерувати ці коди для кожного елемента. Заголовок такого блоку може займати від 90 до 1435 біт, тобто потребує додаткових витрат пам'яті для зберігання.

Актуальною на сьогодні є проблема оптимізації стиснення даних у форматі DEFLATE. У цій роботі йтиметься про шляхи її розв'язання за рахунок врахування всіх можливостей розкладу LZ77, організації взаємодії використаних методів стиснення між собою та можливості вибору розмірів блоків динамічних кодів Хафмана.

Проаналізуємо можливості оптимізації стиснення даних у форматі DEFLATE на прикладі потоку *молмусмолмолмолмус*. В закодованому вигляді згідно *жадібного* розкладу LZ77 він запишеться у вигляді *м о л м у с <4; 6> <3; 3> <5;12>*. Покрокові результати дії алгоритму перед корегуванням словника та буфера для цього потоку відображено в табл. 1.

Спочатку врахуємо, що алгоритмом LZ77 не забороняється виходити за межі словника в область буфера. Головне – щоб співпадаюча послідовність розпочиналася в області словника. Фактично це дозволяє ефективно кодувати групи повторів. Необхідно лише, щоб перша регулярна частина знаходилася в буфері. Справді, якщо $s_j=s_i$, $s_{j+1}=s_{i+1}$, … , $s_{k-1}=s_j$, $s_k=s_j$, $s_{k+1}=s_{j+1}$ … , то $s_k=s_j=s_i$, $s_{k+1}=s_{j+1}=s_{i+1}$ … . Ця особливість алго-

ритму дозволяє швидше та ефективніше кодувати вхідний потік. Наприклад, три останніх кроки з табл. 1 можна оптимізувати як показано у табл. 2. Декодування потоку при цьому не зазнає ніяких змін, оскільки частина буфера, що використовується, буде синхронно відновлюватися з словника.

Розглянемо у табл. 3 ще один варіант трьох останніх кроків з табл. 1. Застосування динамічних кодів Хафмана до довжин, описаних в табл. 3, дасть кращі результати, ніж до даних, описаних в табл. 1, оскільки в цьому варіанті зустрічаються лише два різних значення довжини (3 та 6), а в початковому – три (3, 4 та 5).

Таблиця 1. Покрокові результати стиснення послідовності "молмусмолмолмолмус" згідно алгоритму LZ77

№ кроку	Ковзаюче вікно		Співпадаюча послідовність	Закодовані дані	
	словник	буфер		<довжина; зміщення>	елемент
1.	-	молмусмолмолмолмус	-	-	м
2.	м	олмусмолмолмолмус	-	-	о
3.	мо	лмусмолмолмолмус	-	-	л
4.	мол	мусмолмолмолмус	-	-	м
5.	молм	усмолмолмолмус	-	-	у
6.	молму	смолмолмолмус	-	-	с
7.	молмус	молмолмолмус	молм	<4; 6>	-
8.	молмусмолм	олмолмус	олм	<3; 3>	-
9.	молмусмолмолм	олмус	олмус	<5;12>	-

Таблиця 2. Вдосконалення результатів стиснення послідовності "молмусмолмолмолмус" за рахунок виходу за межі словника

№ кроку	Ковзаюче вікно		Співпадаюча послідовність	Закодовані дані	
	словник	буфер		<довжина; зміщення>	елемент
7.	молмус	молмолмолмус	молм	<4; 6>	-
8.	молмусмолм	олмолмус	олмолм	<6; 3>	-
9.	молмусмолмолмолм	ус	ус	<2;12>	-

Таблиця 3. Вдосконалення результатів стиснення послідовності "молмусмолмолмолмус" за рахунок дублювання значень довжин

№ кроку	Ковзаюче вікно		Співпадаюча послідовність	Закодовані дані	
	словник	буфер		<довжина; зміщення>	елемент
7.	молмус	молмолмолмус	мол	<3; 6>	-
8.	молмусмол	молмолмус	молмол	<6; 3>	-
9.	молмусмолмолмол	мус	мус	<3;12>	-

Наведений приклад демонструє, що для досягнення оптимальніших результатів кодування згідно формату DEFLATE недостатньо лише *жадібного* розкладу LZ77.

Описаний алгоритм майже оптимального розкладу LZ77 згідно (1) можна без змін застосувати для покращення показників стиснення у форматі DEFLATE з статичними кодами Хафмана. Віднайти універсальний оптимальний алгоритм кодування для формату DEFLATE з динамічними кодами Хафмана неможливо, оскільки для оптимального розкладу LZ77 необхідно знати вартості кодування елементів, довжин та зміщень, тобто їх коди Хафмана, а для визначення оптимальних кодів Хафмана необхідно мати елементи розкладу LZ77.

Тому опишемо алгоритм оптимізації розкладу LZ77 для стиснення даних у форматі DEFLATE, що моделює динамічні таблиці Хафмана по ходу опрацювання вхідного потоку. При цьому врахуємо, що менші зміщення мають, як правило, менші вартості кодувань, оскільки вимагають меншої кількості додаткових біт. Визначаючи вартості кодування переходів, будемо спиратися на довжини кодів Хафмана, що розраховуються для попереднього блоку даних. Оскільки в наступному блоці даних можуть зустрітися коди, що були відсутні у попередньому блоці, то при ініціалізації таблиць статистик блоку присвоїмо кожному символу мінімальну частоту, тобто 1. Будемо розраховувати при опрацюванні вхідного потоку всі обчислені оптимальні значення, оскільки кожна позиція потоку може увійти в оптимальний розклад. Розміри блоків

підберемо так, щоб вони з одного блоку, відображали особливості даних, а з іншого – враховували динамічну зміну їх характеристик. Для першого блоку будемо перераховувати коди Хафмана через кожні 100 символів, оскільки попередні блоки для нього відсутні.

Модифікуємо алгоритм оптимального розкладу [3] з урахуванням сказаного вище та алгоритмів з [4]:

```
// length_table – об'єкт для опрацювання кодів Хафмана елементів/довжин
// distance_table – об'єкт для опрацювання кодів Хафмана для зміщень

// функція для підрахунку кількості біт коду Хафмана для елемента/довжини
unsigned char countBitLenLZ77(unsigned int len)
 {unsigned int code, extra, value, huffmancode;
  unsigned char huffmansize;
  if (len < 256) // якщо кодується окремий елемент
   {length_table->encode(len, huffmancode, huffmansize);
    return huffmansize;
   }
  else
   {unsigned int length = len - 256 ;
    lengthToCode(length, code, extra, value); // розкладаємо довжину на базове
                                              // значення та додаткові біти
    length_table->encode(code, huffmancode, huffmansize);
    return huffmansize + extra;
   }
 }
// функція для підрахунку  кількості біт коду Хафмана для зміщення
unsigned char countBitOffsetLZ77(unsigned int offset)
 {unsigned int code, extra, value, huffmancode;
  unsigned char huffmansize;
  distanceToCode(offset, code, extra, value); // розкладаємо зміщення на базове
                                              // значення та додаткові біти
  distance_table->encode(code, huffmancode, huffmansize);
  return huffmansize + extra;
 }
void initHuffmanCode() // встановлення початкових значень частот кодів
```

```
 {unsigned int i;
  for (i = 0; i <= 285; i++)
   length_table->frequencies[i] = 1;
  for (i = 0; i <= 29; i++)
   distance_table->frequencies[i] = 1;
 }
void reBuildHuffmanCode()          // перерахунок кодів Хафмана
{length_table->buildTable();       // перерахунок кодів елементів/довжин
 distance_table->buildTable();     // перерахунок кодів зміщень
}
void incrementHuffmanCode(unsigned int len, unsigned int offset)
 {//збільшення частот відповідних кодів Хафмана
  unsigned int code, extra, value;
  if (len < 256) // якщо кодується окремий елемент
   length_table->incrementFrequency(len);
  else
   {lengthToCode(len - 256, code, extra, value); // повертає базовий код довжини
    length_table->incrementFrequency(code);
    distanceToCode (offset, code, extra, value); // повертає базовий код зміщення
    distance_table->incrementFrequency(code);
   }
 }
struct
 {unsigned long countBit;     // вартість самого "дешевого" шляху від початку
                              // послідовності до позиції t серед проаналізованих
  unsigned int bestLen;       // елемент/довжина останнього кодування, що
                              // забезпечує "найдешевший" шлях до позиції t
  unsigned int bestOffset;    // зміщення останнього кодування, що
                              // забезпечує "найдешевший" шлях до позиції t
 }path[countElement];         // масив оцінок вартостей для кожного елемента
// реалізація основного алгоритму
for (t = 0; t < countElement; t++)
 path[t].price = ULONG_MAX;  // встановлюємо недосяжні вартості кодувань
                             // для забезпечення можливості покращень
unsigned long minCountBitPrevElement = 0; // початкова мінімальна вартість
                             // кодування до попереднього символа
```

```
unsigned long remaintElementBlock = sizeBlock; // кількість елементів, що
                                    // залишилися до кінця чергового блоку
bool firstBlock = true;             // прапорець першого блоку
initHuffmanCode();
// основний цикл розкладу
for (t = 0; t < countElement; t++)
 {if ((t % 100 == 0) && firstBlock)
   reBuildHuffmanCode(); // перераховуємо коди Хафмана для першого блоку
  if (remaintElementBlock == 0)    // при заповненні блоку
   {reBuildHuffmanCode();          // розраховуємо нові коди Хафмана
    initHuffmanCode();             // переходимо до кодування наступного блоку
    remaintElementBlock = sizeBlock;// по замовчуванню розмір блоку рівний 8192
    firstBlock = false;            // новий блок вже не перший
   }
  else
   remaintElementBlock--;
  literal = s[t];                  // елемент чергової позиції
  hashvalue = hashValue(t);        // значення хеш-функції з чергової позиції
  //аналізуємо можливість кодування одного елемента стосовно попередніх
  if (path[t].countBit > minCountBitPrevElement +
     countBitLenLZ77(literal))
   {// знайдено оптимальніший від попереднього варіант кодування
    path[t].countBit = minCountBitPrevElement + countBitLenLZ77(literal);
    path[t].bestLen = literal;
    path[t].bestOffset = 0;
   }
  if (hash_table[hashvalue].next != NULL) // в словнику є послідовності з
                          // аналогічним початком згідно хеш-функції
   {obroblenoLen = 2;   // довжина співпадання має перевищувати 2
   // перебираємо співпадаючі послідовності від закінчення словника до
   // початку, оскільки менші зміщення вимагають менше біт для кодування
    for (current = hash_table [hashvalue].next; current != NULL;
         current = current->next) // детально описано в [3, 4, 5]
     {//визначаємо довжину співпадання активної та чергової послідовності
      for (len = 0; (s[t + len] == s[current->index + len]) &&
                  (len < maxLength); len++);
```

```
    if (len > obroblenoLen) // знайдено довшу співпадаючу послідовність
     {offset = t - current->index;
      currentCountBitOffsetLZ77 = countBitOffsetLZ77(offset);
      for (i = obroblenoLen + 1; i <= len; i++)
       {currentCountBitLenLZ77 = countBitLenLZ77(256 + i);
        if (path[t + i - 1].countBit >minCountBitPrevElement+
            currentCountBitOffsetLZ77 + currentCountBitLenLZ77)
         {path[t + i - 1].countBit = minCountBitPrevElement+
          currentCountBitOffsetLZ77 + currentCountBitLenLZ77;
          path[t + i - 1].bestLen = literal + 256;
          path[t + i - 1].bestOffset = offset;
         }
       }
      obroblenoLen = len;
      if (obroblenoLen == maxLength)
       break;
     }
    }
  }
 // готуємося переходити до наступного елемента
 MoveHashEntry(t, hashvalue); // модифікуємо хеш-таблицю
 minCountBitPrevSimbol = path[t].countBit;
}
// записуємо оптимальний шлях від кінця до початку
countOptimum = 0; // номер оптимального кроку
t = countElement - 1;
while (t >= 0)
 {optimum[countOptimum].bestLen = path[t].bestLen;
  optimum[countOptimum].bestOffset = path[t].bestOffset;
  if (path[t].bestLen < 256)
   t--
  else
   t-= (path[t].bestLen - 256);
  countOptinun++;
 }
```

Розглянемо тепер питання вибору розмірів блоків динамічних кодів Хафмана, адже кожен елемент розкладу LZ77 може кодуватися різною кількістю біт в різних блоках залежно від частоти його викорис-

тання. На перший погляд, здається, що стиснення даних в єдиний блок покращує степінь 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 послідовність на два рівні блоки, то в кожному з них зустрінуться лише по два різних коди, кожному з яких 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 розміри блоків динамічних кодів Хафмана пропонується визначати за наступним алгоритмом:

1. Розбиваємо весь вхідний потік кодів LZ77 на M блоків мінімальної довжини (наприклад, по 512 кодів). Для кожного блоку j підраховуємо частоти n_{ji} використання коду i та на їх основі генеруємо коди Хафмана з довжинами (кількістю біт) d_{ji}. Тоді кількість біт для зберігання кожного блоку складає $price_j = c_j + \sum_i n_{ji} d_i$, де c_j – кількість біт для зберігання заголовка блоку j;
2. Підраховуємо кількості біт для зберігання можливих поєднань M' сусідніх блоків. При цьому аналізувати ще раз вхідний потік непотрібно, оскільки $n'_{ji} = n_{ji} + n_{j+1,i}$ і на основі цих частот можна згенерувати коди Хафмана можливих поєднань сусідніх блоків з довжинами d'_{ji}. Кількість біт для зберігання можливих поєднань сусідніх блоків складає $price'_j = c^i_j + \sum_i n'_{ji} d'_i$;

3. Знаходимо величини виграшів від можливого поєднання сусідніх блоків $v_j = price_j + price_{j+1} - price'_j$ та обчислюємо серед них максимальний $v = \max_j v_j$;
4. Якщо $v>0$, то поєднуємо довжини та частоти сусідніх блоків, що дозволяє отримати цей виграш, перераховуємо кількості біт для зберігання можливих поєднань створеного блоку з сусідніми та повертаємося до п.3;
5. Поєднання блоків завершуємо, коли v стане недодатнім або залишиться лише один суцільний блок.

Насамкінець, наведемо результати дії описаних алгоритмів для стиснення файлів зображень у форматі PNG, що базується на форматі стиснення DEFLATE. За основу при програмуванні було взято текст компресора з [4]. При цьому над вихідними текстами було виконано наступні модифікації:

1. Забезпечено ведення неперервного словник для всього процесу кодування LZ77;
2. Реалізовано вихід за межі словника в область буфера;
3. Впроваджено розрахунок розмірів блоків динамічних кодів Хафмана та модифікації 1, 2;
4. Оптимізовано розклад LZ77 з використанням статичних кодів Хафмана та враховано модифікації 1, 2;
5. Оптимізовано розклад LZ77 з використанням прогнозованих динамічних кодів Хафмана та враховано модифікації 1, 2;
6. Реалізовано всі описані вище модифікації.

Тестування проводилося на стандартному наборі файлів ACT 24-бітних зображень (див. табл. 4) при аналізі до 256 попередніх співпадаючих послідовностей.

Результати тестування наведено в табл. 5-6 (коефіцієнт стиснення – це процент зменшення початкового розміру файла). Для порівняння в цих же таблицях наведено результати збереження у форматі PNG за допомогою програми MS Photo Editor 2000.

Таблиця 4. Характеристики зображень набору ACT

№ файла	Назва файла	Розміри зображення (пікселів)	Розміри файла (Кб)
1.	Clegg.bmp	814x880	2 101
2.	Frymire.bmp	1118x1105	3 622
3.	Lena.bmp	512x512	769
4.	Monarch.bmp	768x512	1 153
5.	Peppers.bmp	512x512	769
6.	Sail.bmp	768x512	1 153
7.	Serrano.bmp	629x794	1 464
8.	Tulips.bmp	768x512	1 153

Таблиця 5. Коефіцієнти стиснення файлів набору ACT

Програма \ Файл	1	2	3	4	5	6	7	8	Разом
Photo Editor	62,26	92,57	4,81	23,50	8,45	13,62	92,35	9,97	54,65
Базова програма	73,25	91,66	6,24	25,67	12,22	17,87	92,28	13,36	57,52
Модифікація 1	75,30	92,49	8,45	28,88	15,08	22,03	92,49	16,39	59,45
Модифікація 2	76,01	92,66	8,45	29,14	15,21	22,03	92,76	16,57	59,70
Модифікація 3	76,06	92,68	8,58	29,31	15,34	22,29	92,76	16,74	59,79
Модифікація 4	76,20	93,21	8,71	30,27	15,86	22,55	92,96	17,09	60,19
Модифікація 5	76,30	93,43	8,71	30,79	16,12	23,07	93,24	17,43	60,45
Модифікація 6	76,34	93,46	8,84	30,96	16,25	23,24	93,31	17,61	60,54

Таблиця 6. Час стиснення (в секундах) файлів набору ACT

Програма \ Файл	1	2	3	4	5	6	7	8	Разом
Photo Editor	3	4	2	3	2	3	2	3	22
Базова програма	17	36	7	11	7	12	14	11	115
Модифікація 1	16	28	7	11	8	11	11	11	103
Модифікація 2	16	26	8	11	7	11	11	11	101
Модифікація 3	92	35	88	91	81	93	15	120	615
Модифікація 4	147	238	87	98	78	102	110	114	974
Модифікація 5	278	364	95	109	84	111	173	124	1338
Модифікація 6	349	368	176	184	144	181	173	228	1803

З наведених результатів тестування можна зробити такі висновки:

1. Розглянуті алгоритми підвищують, хоча й несуттєво, коефіцієнти стиснення всіх файлів набору ACT, не вимагають модифікацій декодера та програм перегляду зображень і за рахунок зменшення розмірів файлів лише прискорюють їх роботу. Саме тому вони можуть бути використані для збереження графічних файлів у форматах, що використовують метод стиснення DEFLATE;
2. Врахування можливостей ведення неперервного словника та виходу в область буфера в алгоритмі стиснення LZ77 дозволяє в середньому підвищити на 2,18% коефіцієнт стиснення та скоротити на 12% час стиснення;
3. Розрахунок розмірів блоків динамічних кодів Хафмана хоча й підвищує коефіцієнт стиснення на 0,09%, але сповільнює в середньому у 6 разів тривалість стиснення;
4. Оптимізація розкладу LZ77 з використанням статичних кодів Хафмана підвищує коефіцієнт стиснення в середньому на 0,49% при збільшенні витрат часу кодування у 9,7 разів. Використання прогнозованих динамічних кодів Хафмана дає кращі результати стосовно статичних на 0,26%, хоча й вимагає додатково на 50% більше часу;
5. Впровадження всіх запропонованих алгоритмів підвищує коефіцієнт стиснення на 3,02%, але вимагає у 18 разів більше часу на кодування;
6. Для оптимізації стиснення даних у форматах, що послідовно використовують декілька методів, при алгоритмізації кожного окремого методу слід враховувати особливості наступних методів.

1. *Ziv J. Lempel A.* A universal algorithm for sequential data compression // IEEE Transactions on Information Theory.- May 1977.- Vol. 23(3).- P. 337-343.
2. *Ziv J. Lempel A.* Compression of individual sequences via variable-rate coding // IEEE Transactions on Information Theory.- Sept. 1978.- Vol. 24(5).- P. 530-536.
3. Методы сжатия даннях. Устройство архиваторов, сжатие зображений и видео / *Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин*.- М.: ДИАЛОГ-МИФИ, 2003.- С. 75-118.

4. *Миано Дж.* Форматы и алгоритмы сжатия зображений в действии: Учеб. пособ.- М.: Триумф, 2003.- С. 249-318.
5. *Балашов К.Ю.* О повышении быстродействия LZW алгоритма сжатия информации.- http: //www.compression.ru.

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

Надійшла 11.04.2008

Шпортько А.В. АЛГОРИТМЫ ОПТИМИЗАЦИИ РАЗЛОЖЕНИЯ LZ77 И ВЫБОРА РАЗМЕРОВ БЛОКОВ ДИНАМИЧЕСКИХ КОДОВ ХАФФМАНА ДЛЯ СЖАТИЯ ДАННЫХ В ФОРМАТЕ DEFLATE // *Проанализированы особенности кодирования данных, обоснована возможность и описаны алгоритмы оптимизации разложения и выбора размеров блоков динамических кодов Хаффмана для формата DEFLATE. Приведены фрагменты программ для реализации предложенных алгоритмов и результаты их применения для сжатия изображений в формате PNG.*

Shportko A.V. ALGORITHMS OF THE OPTIMIZATION OF DECOMPOSITION OF LZ77 AND CHOICE OF SIZES OF BLOCKS OF RUN-TIME CODES OF HUFFMAN FOR THE COMPRESSION OF DATA IN FORMAT OF DEFLATE // *The features of the code of information are considered, the possibility and the algorithms of the optimization of time-table and the choice of sizes of blocks of dynamic codes of Huffman are described for the format of DEFLATE. The fragments of the programs for the realization of the offered algorithms and results of their application for the compression of images in the format of PNG are presented.*

Наукове видання

ВОЛИНСЬКИЙ МАТЕМАТИЧНИЙ ВІСНИК

серія прикладна математика

Збірник наукових праць

Випуск 5 (14)

Відповідальний за випуск
Бомба А.Я.

Підписано до друку . .2008 р.
Папір офсет. Формат 60/84 1/16.
Ум. друк. арк. 7,5. Тираж 100.

Редакційно-видавничий відділ
Рівненського державного гуманітарного університету
Україна, м. Рівне, 33028, вул. С. Бандери, 12