УДК 004.627

В. Голота, І. Когут, Т. Голота
Прикарпатський університет

ПРО МОДИФІКАЦІЮ СТАНДАРТНОГО АЛГОРИТМУ ЛЕКСИКОГРАФІЧНОГО СОРТУВАННЯ ДАНИХ

Запропоновано модифікацію стандартного алгоритму лексикографічного сортування даних, що базується на сортуванні базових рядків і визначенні без сортування порядку похідних рядків. Модифікований алгоритм, порівняно із стандартним, дає змогу значно скоротити час стискання файлів розмірами понад 0.3 Mb.

Сьогодні при стисканні даних одним із поширених є a Block Sorting Lossles Data Compression Algorithm. В цьому алгоритмі використовується Burrows-Wheeler Transformation (BWT) [1] на основі стандартного лексикографічного сортування даних [2]. Автори пропонують модифіковану версію цього алгоритму. Для оцінки пропонованого алгоритму використовувалась тестова програма, яка містила simple run-length encoder, standard BWT, Move to Front encoder, an order-0 adaptive arithmetic encoder [3]. Під час виконання тестової програми час стискання файлів невеликих розмірів до 0.3 Mb мав прийнятні значення (десятки секунд на PC Pentium-200). Якщо розміри файлів зростали до 1 Mb, час стискання ставав неприйнятним (десятки хвилин). Аналіз статистики звернень до функцій показав, що 90 % з них – це виклики функції сортування даних qsort().

Для зменшення часу стискання файлів розмірами понад 0.3 Mb потрібно зменшити кількість викликів функції qsort(). Цього можна досягти за рахунок модифікації стандартного лексикографічного сортування, в якому використовується матриця символів. Така матриця символів для масиву вказівників p[n] і рядка s[n] = "aparar", n=6 і до і після лексикографічного сортування наведена на рисунку.

									d=0	d=1	d=2	d=3		
0	p[0]=0	a	p	a	r	a	r	p[0]=0	a	p	a	r	a	r
1	p[1]=1	p	a	r	a	r	*a*	p[1]=4	a	r	*a*	*p*	*a*	*r*
2	p[2]=2	a	r	a	r	*a*	*p*	p[2]=2	a	r	a	r	*a*	*r*
3	p[3]=3	r	a	r	*a*	*p*	*a*	p[3]=1	p	a	r	a	r	*a*
4	p[4]=4	a	r	*a*	*p*	*a*	*r*	p[4]=5	r	*a*	*p*	*a*	*r*	*a*
5	p[5]=5	r	*a*	*p*	*a*	*r*	*a*	p[5]=3	r	a	r	*a*	*p*	*a*

а *б*

Матриця символів: а) до лексикографічного сортування;
б) після лексикографічного сортування: p[] – масив вказівників на рядок символів s[],
d – глибина лексикографічного сортування

Насправді матриця символів не створюється, а моделюється за допомогою оператора s [p[i] + d], i = 0, ... ,n-1; d=0, ... ,n-1, в якому індекс p[i] + d при досягненні значення n встановлюється в 0. Так забезпечується циклічний рух індексу p[i] + d. Сортуються також не рядки символів, а вектор вказівників на рядки.

За стандартним алгоритмом лексикографічне сортування проводиться з використанням функції qsort* (lo, hi, d, ...), в якій задається інтервал і глибина сортування. У нашому випадку інтервали сортування lo-hi при поступовому збільшенні глибини d матимуть значення, вказані у табл.1.

Таблиця 1

Інтервали при стандартному алгоритмі лексикографічного сортування

d=0	d=1	d=2	d=3
0 – 5	0 – 2	1 – 2	1 – 2
–	4 – 5	4 – 5	–

Загальна кількість викликів функції qsort* (lo, hi, d, ...) становить 6. Необхідно відзначити, що функція qsort* (lo, hi, d, ...) викликає стандартну функцію qsort(), що сортує рядки, довжина яких n = hi - lo + 1. При цьому qsort() виконує 1.386n lg n порівнянь символів [4], на що і витрачається основний час сортування.

Якщо подивитись матрицю символів до лексикографічного сортування, то можна побачити, що її рядки є циклічними зсувами на один розряд вліво від початкового рядка. Крім того, початковий рядок може містити трисимвольні підрядки з однаковими початковим і кінцевим символами. В нашому випадку такими підрядками будуть "apa", "ara", "rar". При циклічних зсувах вліво такі підрядки опиняються на початку рядків. Назвемо такі рядки базовими. Наступний зсув вліво формує похідні рядки від базових. Характерною ознакою таких базових і похідних рядків є переставлені символи в перших двох розрядах s[0], s[1]: ap – pa, ar – ra. Важливою властивістю таких рядків є можливість їх ідентифікації тільки за першими двома символами. Оскільки базових і похідних рядків, з однаковими парами символів в перших двох розрядах, може бути досить багато (у нашому випадку ar–ra, ar–ra), то в цьому випадку необхідна додаткова інформація для однозначної ідентифікації таких рядків. Оскільки нас цікавить зв'язок з базовими відсортованими рядками і залежними невідсортованими рядками, то такою інформацією може бути таблиця частотності відсортованих пар символів. Якщо всі можливі пари символів s[0], s[1] перетворити у ціле типу Int16 (s[0]<<8) + s[1], то розмір такої таблиці буде Tab[65536]. В нашому випадку така таблиця буде тільки частково заповненою.

Таблиця 2

Таблиця частотності пар перших двох символів рядка

Int16	s[0], s[1]	Частотність
24944	a, p	1
24946	a, r	2
28769	p, a	1
29281	r, a	2

Така таблиця створюється в процесі порозрядного сортування цілих чисел типу Int16 на глибину d=0, наприклад, стандартною програмою radix(). При порозрядному символьному сортуванні така таблиця створюється тільки для символів, що не забезпечує ідентифікації рядків.

Після створення таблиці частотності пар перших двох символів рядків проводиться сортування базових рядків на необхідну глибину. Після закінчення сортування кожного базового рядка визначається відсортована послідовність рядків без сортування на основі таблиці частотності пар символів. Відсортовані базові і похідні рядки виключаються із розгляду при формуванні наступних інтервалів сортування, що зменшує кількість викликів функцій сортування. У такому випадку сортування матиме послідовність, відображену у табл.3.

Таблиця 3

Інтервали при модифікованому алгоритмі лексикографічного сортування

d=0-1	d=2	d=3
0 – 5	1 – 2	1 – 2
–	–	–

Загальна кількість викликів функцій radix() і qsort* (lo, hi, d, ...) становить 3.

Можлива така реалізація сортування базових рядків і визначення відсортованого порядку похідних рядків з використанням таблиці частотності пар символів на мові програмування С:

```
for (  base = 0; base < 256; base++  ) {    /* for 1,  цикл по першому символу*/
 for (  j = 0; j < 256; j++  ) {      /*  for 2, цикл по другому символу */
  if ( base != j ) { /* if 1, перші два символи повинні бути різні */
 if ( sorted[ ( base << 8 ) + j ] == 0 )  { /* if 2, перевірка на невідсортованість */
     lo = tab_frequency [ ( base << 8 ) + j  ];
     hi = tab_frequency [ ( base << 8 ) + j +1 ] - 1;
if ( hi > lo) qsort (  s, p, lo, hi, d  ); / * сортування базового рядка */
} /* end if 2   */
   sorted[( base << 8 ) + j ] == 1;      /* відмітка відсортованості рядка */
}   /* end if 1  */
}    /* end for 2 */
/* копія таблиці частотності */
for ( j=0; j<256; j++ ) tab[j] = tab_frequency[ (j<<8) + base ];
    /* for 3, розрахунок відсортованого порядку похідних рядків */
for ( j = tab_frequency [( base << 8 ) ]; j < tab_frequency [( base << 8 ) + 1]; j++ ) {
        k = p[j] - 1;
        if( k < 0 ) k+=n;   /* контроль індексу к на діапазон 0 - n */
        c = s[k];              /* символ для таблиці частотності        */
        p[ tab[c]++ ] = k;  /* відсортований порядок похідних рядків */
} /* end for 3 */
} /* end for1 */
```

Час стискання файлів різних розмірів програмою із використанням стандартного і модифікованого алгоритмів лексикографічного сортування наведено у табл. 4.

Таблиця 4

Час стискання файлів різних розмірів програмою із використанням стандартного і модифікованого алгоритмів

Розмір файла, Mb	Тст, с	Тмод, с	Тст /Тмод
0.042	1.88	0.83	2.26
0.094	4.13	1.48	2.79
0.469	25.46	4.30	5.92
0.911	55.65	8.46	6.57
2.201	182.30	21.11	8.63

Як видно із табл. 4, ефективність модифікованого алгоритму зростає зі збільшенням розмірів файлів.

Висновки. Запропоновано модифікацію стандартного алгоритму лексикографічного сортування даних, що базується на сортуванні базових рядків і визначенні без сортування порядку похідних рядків. Модифікований алгоритм, порівняно із стандартним, дає змогу значно скоротити час стискання файлів розмірами понад 0.3 Mb.

1. Burrows M. and Wheeler D.J. A block - sorting Lossless Data Compression Algorithm // SRC Research Report 124, Digital System Research Center, Palo Alto, CA, May 1994. 2. Fenwick P. Block Sorting Text Compression - Final Report, The University of Auckland, Department of Computer Science, Technical Report 130, March 1996. 3. Numerical recipes in C. The Art of Scientific Computing. William H. Press ... [et. al]. Cambridge University Press, 1995. 4. Hoare, C.A.R. Quicksort // Computer Jornal, 1962. 5. P.10–15.