ФІЗИКА І ХІМІЯ ТВЕРДОГО ТІЛА
Т. 3, № 1 (2002) С. 361-364

PHYSICS AND CHEMISTRY OF SOLID STATE
V. 3, № 1 (2002) P.361-364

УДК 004.627

В.І. Голота

Порівняння алгоритмів внутрішнього сортування символьних строк

Прикарпатський університет імені Василя Стефаника,
76025, м.Івано-Франківськ, вул.Шевченка,79 ,
т.59-61-09, E-mail: kre@pu.if.ua

Запропоновано реалізацію швидкодіючого алгоритму внутрішнього сортування символьних строк із мінімальною пам'яттю. Для порівнюваних алгоритмів сортування експериментально визначено чутливість на символьних строках із різним порядком елементів і швидкодію на символьних строках довжиною до 100 Мбайт.

Ключові слова: стискання даних, алгоритми сортування, символьні строки.

Стаття поступила до редакції 18.04.2002; прийнята до друку 03.06.2002

I. Вступ

В блоко-орієнтованих програмах безвтратного стискання даних використовуються алгоритми лексикографічного сортування символьних строк [1, 2]. Величина блоку може опційно змінюватися в межах 256 – 1024 Кб, залежно від необхідної швидкодії чи ефективності стискання даних. В алгоритмах лексикографічного сортування використовуються швидкі алгоритми внутрішнього сортування символьних строк. Нові персональні комп'ютери Pentium-4 з операційною системою Windows XP дозволяють збільшувати розмір оперативної пам'яті понад 256 Мб. Це дає змогу програмам внутрішнього сортування використовувати значно більшу оперативну пам'ять. В зв'язку з цим виникає необхідність експериментального порівняння алгоритмів внутрішнього сортування символьних строк в діапазоні розмірів співрозмірних з доступною оперативною пам'яттю.

II. Набори даних для порівняння.

Для алгоритмів внутрішнього сортування теоретично визначені оцінки ефективності для мінімального, очікуваного і максимального числа порівнянь [3]. Ці оцінки отримані при певних припущеннях щодо структури даних: випадковий характер розподілу елементів, однакова ймовірність значень елементів, відсутність повторюваності і чередувань. Для символьних строк великих розмірів ці припущення не справджуються, тому можливі відхилення від теоретичних оцінок. Такі відхилення, тобто чутливість алгоритмів, оцінювались на символьних строках однакової довжини із різним порядком елементів. Швидкодія алгоритмів оцінювалась на символьних строках різної довжини від 2.3 до 100 Мб із елементами повторюваності. Базову строку взято із файлу *c:\winnt\system32\shell32.dll* розміром 2.36 Мб. Строки більших довжин в 2, 3, 4, 5, 10, 15, …, 45 разів взяті із файлів отриманих шляхом об'єднувального копіювання вхідного файлу *in* у вихідний файл *out* з допомогою команди MS DOS *copy /b in+in+ … +in out*.

III. Результати порівняння алгоритмів сортування

Алгоритми внутрішнього сортування використовують лінійні або нелінійні структури даних. Алгоритми, що використовують лінійні структури даних мають час роботи порядку $O(N^2)$, тому вони використовуються при невеликих значеннях N. До таких алгоритмів відноситься лінійний вибір, лінійний вибір з обміном, лінійний вибір з підрахунком, просіювання, лінійна вставка [3]. Алгоритми, що використовують нелінійні структури даних, мають час роботи порядку $O(N \cdot log_2 N)$ [4]. З цих алгоритмів для порівняння було вибрано наступні мінімальні по пам'яті [5, 6, 7]:

– розширене сортування просіюванням (Шел - *shellsort*);

– швидке сортування (два інтервали, Хоар – *quicksort2*);

– швидке сортування (три інтервали, модифікація Седгевіка – *quicksort3*);

– пірамідальне сортування (*heapsort*);

– порозрядне сортування (*quickradix*).

Для порівняння алгоритмів використовувався ПК Pentium-3 (533 МГц) із операційною системою Windows 2000 Server. Алгоритми реалізовувалися на мові програмування С з використанням компілятора MS VC++ 6.0. Чутливість алгоритмів до різного рорядку елементів визначалась на строках довжиною 58.9 Мб із наступними структурами даних:

– випадковість (строки створені з допомогою генератора випадкових чисел (функція *rand()*);

– повторюваність (строки створені шляхом об'єднувального копіювання);

– впорядкованість (попередньо відсортовані строки по зростанню і по спаданню);

– чергування (строки в яких парні елементи відсортовані по зростанню, а непарні по спаданню);

– відсортованість по половинах (строки в яких відсортовані елементи в інтервалах *0 – N/2-1* і *N/2 – N-1*);

Перевірка на повторюваність визначає чутливість до дублювання, порядку і робочої довжини. Перевірка на чергованість і відсортованість по половинах визначає чутливість до робочої довжини і порядку. Це два із багатьох методів, якими можна задати невипадковість даних. В табл. 1 для порівнюваних алгоритмів приведено час сортування (в *сек*) символьних строк із різним порядком елементів. Приведені результати дозволяють проаналізувати чутливість алгоритмів до порядку елементів.

Як видно із табл. 1 всі алгоритми, за вийнятком *quickradix*, чутливі до зміни порядку елементів символьної строки.

В табл. 2, 3 приведений час сортування символьних строк з повторюваностями в діапазоні довжин 2,3 – 106,1 Мб. Швидкість сортування для алгоритмів (*V*, *Мб/с*) визначалась на основі даних табл. 2, 3 і наведена на рис. 1.

Як видно із рис. 1 всі алгоритми мають стабільну швидкість сортування у діапазоні довжин строк до 80 Мб. При більших довжинах строк швидкість сортування падає внаслідок вичерпання доступної оперативної пам'яті і використання дискової віртуальної пам'яті. Найбільшу і прийнятну швидкість сортування символьних строк довжиною до 80 Мб мають алгоритми *quickradix*, *quicksort3* і *quicksort2*. Крім цього алгоритм *quickradix* має швидкість сортування приблизно в 12 разів більшу від *quicksort3*.

Таблиця 1

Час сортування (с) символьних строк із різним порядком елементів

Алгоритм	Порядок елементів символьних сторок					Відносна макс. чутливість*, %
	випад-ковий	відсорто-ваний по зростанню	відсорто-ваний по спаданню	чергу-вання	відсор-товані половини	
quickradix	1,30	1,28	1,31	1,30	1,27	2,3
quicksort3	21,87	11,80	17,33	17,93	16,07	26,5
quicksort2	23,09	15,96	18,43	19,05	18,89	30,8
Heapsort	67,94	31,59	40,66	42,24	33,06	53,5
Shellsort	71,55	19,62	51,03	42,21	21,45	72,6

* Примітка. Відносна максимальна чутливість визначена відносно випадкового порядку елементів.

Таблиця 2

Час сортування (с) символьних строк розміром 2,3 - 11,8 Мб

Алгоритм	Розмір символьних строк, Мб				
	2,3	4,7	7,1	9,4	11,8
quickradix	0,05	0,10	0,15	0,20	0,26
quicksort3	0,57	1,2	1,83	2,49	3,04
quicksort2	0,66	1,38	2,13	2,92	3,70
heapsort	1,65	3,63	5,61	7,60	9,56
shellsort	1,74	4,07	6,59	9,27	11,65

Таблиця 3

Час сортування (с) символьних строк розміром 23,6 – 106,1 Мб

Алгоритм	Розмір символьних строк, Мб							
	23,6	35,4	47,2	58,9	70,8	82,5	94,3	106,1
quickradix	0,51	0,77	1,03	1,28	1,54	1,78	2,04	15,03
quicksort3	6,76	9,39	13,48	15,53	19,17	21,99	27,23	44,41
quicksort2	7,94	12,31	16,72	20,81	25,4	30,29	34,99	74,85
heapsort	19,69	29,65	41,02	50,31	60,73	71,64	83,13	103,79
shellsort	25,26	38,88	55,44	68,19	85,71	101,81	116,71	427,77

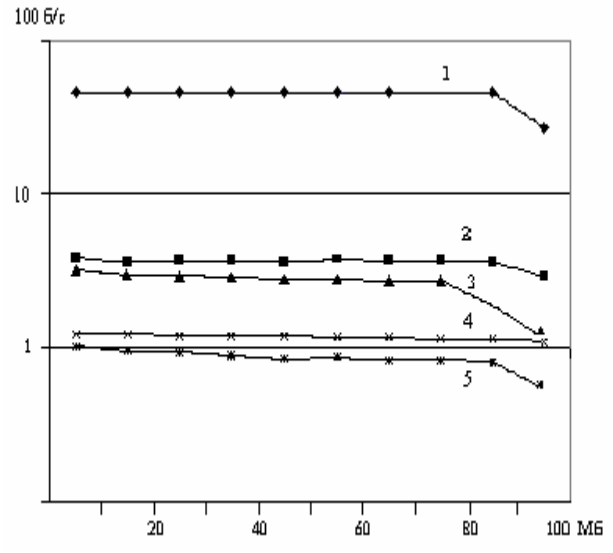

Рис. 1. Швидкість сортування символьних строк довжиною до 106.1 Мб:
1 – quickradix; 2 – quicksort3; 3 – quicksort2; 4 – heapsort; 5 – shellsort.

IV. Реалізація алгоритмів

Забезпечення високої швидкості сортування символьних строк довжиною до 80 Мб досягнуто за рахунок реалізації мінімального по пам'яті алгоритму *quickradix* з використанням тільки таблиці частотності символів. В стандартній реалізації алгоритму *quicksort3* збільшено розмір стеку до 1024 і введено контроль значень індексів, що поміщаються в стек.

Реалізації протестованих функцій *quickradix* і *quicksort3* на мові програмування С [8] мають наступний вигляд:

```
// quickradix – швидке порозрядне сортування
#include <mem.h>
/* lo – нижній індекс сортування, hi – верхній
індекс сортування,dest[N] – символьна строка
довжиною N, що сортується. lo<hi */
void quickradix ( int lo, int hi, unsigned char *dest )
{
  int i, j, k, ftab[256];
/* заповнення таблиці частотності символів */
 memset ( ftab, 0, sizeof (ftab) );
 for ( i=lo; i<=hi; i++ ) ftab[dest[i]]++;
 /* сортування */
 for ( k=lo, i=0; i<256; i++ )
 if ( ftab[i] > 0 ) for ( j=0; j<ftab[i]; j++ ) dest[k++] =
(unsigned char) i
}
/* quicksort3 – швидке 3-х інтервальне
сортування */
#include <stdio.h>
#include <stdlib.h>
#define swap(c1,c2) { unsigned char tmp = c1; c1 = c2;
c2 = tmp; }
static void vswap ( unsigned char* in, int p1, int p2, int
n ) {
  while ( n > 0 ) {
    swap(in[p1], in[p2]);
    p1++; p2++; n--;
  }
}
static int med3 ( unsigned char a, unsigned char b,
unsigned char c ) {
  unsigned char t;
  if (a > b) { t = a; a = b; b = t; };
  if (b > c) { t = b; b = c; c = t; };
  if (a > b)        b = a;
  return (int) b;
}
#define min(a,b) ((a) < (b)) ? (a) : (b)
typedef struct { int l; int h; } StackElem;
#define push(ll,hh){ stack[sp].l = ll;stack[sp].h =
hh;sp++; }
#define pop(ll,hh) { sp--;ll = stack[sp].l;hh =
stack[sp].h; }
#define STACK_SIZE 1024
/* lo – нижній індекс сортування, hi – верхній
індекс сортування, out[N] – символьна строка
довжиною N, що сортується. LO<HI */
void quicksort3 ( int LO, int HI, unsigned char* out )
{//_1
  int lo, Lo_i, Lo_eq, loSt, hi, Hi_i, Hi_eq, hiSt;
  int med, n, m, sp;
  StackElem stack[STACK_SIZE];
  sp = 0;
  loSt=LO; hiSt=HI;
  push ( loSt, hiSt ); // початковий запис у стек
/* цикл поки не пустий лічильник стека */
  while (sp > 0) {//_2
/* контроль розміру стека */
if(sp>=STACK_SIZE) {fprintf(stderr,"Stack overflaw
%d\n",sp);exit(1);}
    pop ( lo, hi ); /* читання стека */
    med = med3 ( out[lo],out[hi],out[(lo+hi)>>1] ); /*
визначення медіани */
/* : Lo_eq : Lo_i : Hi_i : Hi_eq : - індекси 4-х
інтервалів */
    Lo_eq = Lo_i = lo;
    Hi_i = Hi_eq = hi;
    while (true) {//_3 – цикл по перестановці
символів більших і менших від медіани
      while (true) {//_4 – цикл по пошуку символів
більших від медіани
        if (Lo_i > Hi_i) break;
        n = (int) out[Lo_i] - med;
/* формування першого інтервалу із символів, що
співпадають із медіаною */
  if (n == 0) { swap(out[Lo_i], out[Lo_eq]); Lo_i++;
Lo_eq++; continue; };
        if (n >  0) break;
        Lo_i++;
      }//4_
      while (true) {//_5 – цикл по пошуку символів
менших від медіани
        if (Lo_i > Hi_i) break;
        n = (int) out[Hi_i] - med;
```

```
/* формування другого інтервалу із символів, що
співпадають із медіаною */
 if (n == 0) { swap(out[Hi_i], out[Hi_eq]); Hi_i--;
Hi_eq--; continue; };
          if (n <  0) break;
          Hi_i--;
        }//_5
        if (Lo_i > Hi_i) break;
/* перестановка символів більших і менших від
медіани */
        swap(out[Lo_i], out[Hi_i]); Lo_i++; Hi_i--;
      }//3_
/* контроль індексів */
     if( Hi_i != Lo_i-1) {fprintf(stderr,"bad termination
in quickSort3");exit(1);}
     if (Hi_eq < Lo_eq)  continue;
/* злиття 2-х інтервалів в один із елементами, що
однакові з медіаною */
     n = min(Lo_eq-lo, Lo_i-Lo_eq);  vswap(out, lo,
Lo_i-n, n);
     m = min(hi-Hi_eq,  Hi_eq-Hi_i); vswap(out, Lo_i,
hi-m+1, m);
/* визначення нових індексів для невідсортованих
інтервалів */
     n = lo +  Lo_i  - Lo_eq - 1;
     m = hi - (Hi_eq - Hi_i) + 1;
/* контроль індексів і запис їх у стек */
     if(lo<n) push ( lo, n );
     if(m<hi) push ( m, hi );
   }//2_
}//1_
```

V. Висновки

В результаті тестування п’яти алгоритмів сортування на символьних строках розміром до 106.1 Мб визначено їх чутливість і швидкість сортування. Найменш чутливим до порядку елементів є алгоритм *quickradix*.

Швидкість сортування для всіх алгоритмів є практично постійною у діапазоні довжин символьних строк до 80 Мб, тобто при використанні тільки оперативної пам’яті комп’ютера. Найбільшу швидкість сортування символьних строк мають алгоритми *quickradix*, *quicksort3* і *quicksort2*. Крім цього алгоритм *quickradix* має швидкість сортування приблизно в 12 разів більшу від *quicksort3*. Таку високу швидкість сортування отримано за рахунок реалізації мінімального по пам’яті алгоритму порозрядного сортування, що використовує тільки таблицю частотності символів. Алгоритми *quickradix*, *quicksort3* і *quicksort2* можуть бути рекомендовані для лексикографічного сортування в блокоорієнтованих програмах стискання даних. Алгоритм *quickradix* необхідно використовувати для лексикографічного сортування, яке базується на сортуванні символів, а *quicksort3* і *quicksort2* – на сортуванні індексів або вказівників символів. Величина буфера лексикографічного сортування при використанні алгоритму *quickradix* може бути збільшена в 10 разів.

В.І. Голота *– завідувач лабораторіями кафедри радіофізики і електроніки.*

[1]. K. Sayood *Introduction to Data Compession*. San Diego, CA.: Academic Press, 636 c. (2000).
[2]. В. Голота, І. Когут Про модифікацію стандартного алгоритму лексикографічного сортування даних // *Вісн. НУ “Львівська політехніка*, **415**, с. 180–83 (2001).
[3]. Г. Лорин. *Сортировка и системи сортировки*. Наука, М. 384 с. (1983).
[4]. Д.Э. Кнут *Искуство программирования*, том. 3. Сортировка и поиск, 2-е изд.: Пер. с англ.: Уч. пос. Издательский дом “Вильямс”, М., 832 с. (2000).
[5]. C.A.R. Hoare, *Quicksort*. Computer Journal, **5**(1), p. 10–15 (1962).
[6]. R. Sedgewick. *Quick Sort With Equal Keys* // SIAM J. Comp., **6**(2), p.240–267 (1977).
[7]. H. William Press [et. al.]. *Numerical recipes in C*. The Art of Scientific Computing. Cambridge University Press (1995).
[8]. Бьярн Страуструп. *Язык программирования C++*. Пер. с англ., К. “ДиаСофт”, 264 с. (1993).

V.I. Holota

Matching of Algorithms for Interior Strings Sorting

Vasyl Stefanyk Precarpathian University 57, Shevchenka Str., Ivano-Frankivsk, 76000
(3422) 59-61-09, E-mail: kre@pu.if.ua

The realization of the quick algorithm of the internal sorting of character strings with minimal memory is proposed. The sensitivity and quickness of compared algorithms experimentaly are determined for character strings with different order of elements and string size to 100 Mb.