Educational Research
Научно-методически статии

ДВУМЕРНИ МАСИВИ: АЛГОРИТМИ ЗА ТЪРСЕНЕ И ЕКСПЕРИМЕНТИ

Павел Азълов
Pennsylvania State University

Резюме. В статията се разглеждат алгоритми за търсене в числени 2D частично наредени масиви, т.е. масиви, чиито елементи са ненамаляващи по редове и колонки. Накратко са разгледани четири известни алгоритъма, единият от които е алгоритъмът за търсене *Saddleback*. Описан е и един нов алгоритъм, наречен *алгоритъм за търсене чрез концентрични подмасиви*. Основната идея на алгоритъма е конструирането на редица от подмасиви с намаляващи размери на редовете и колонките, като всеки следващ подмасив на редицата се съдържа в предшестващия го подмасив. По този начин, ако търсеното число се съдържа в първоначалния масив, то числото се съдържа и във всеки подмасив на редицата, последният от които съдържа само един елемент със стойност, равна на търсеното число. Всичките пет алгоритъма са реализирани в средата за програмиране Microsoft Visual C++. Извършени са експерименти, в които са измерени времената за изпълнението им за разнообразни стойности на броя на редовете и колонките в масивите. Част от резултатите от проведените експерименти са дадени в няколко таблици и са илюстрирани със съответни графики. Най-добри времена от функциите, реализиращи съответните алгоритми, са получени от алгоритъма за търсене с концентрични подмасиви.

Keywords: search in 2D arrays, 2D partially ordered arrays, Saddleback algorithm, empirical analysis

1. Въведение

1.1. Основна задача

В тази статия се разглеждат алгоритми за търсене в числени двумерни масиви, (2D масиви), с M реда и N колонки, M и N са цели положителни числа. Предполага се, че елементите на масивите са ненамаляващи числени редици по редове и колонки, т.е. ако a е такъв масив, тогава са в сила следните две свойства:

– *Наредба по редове*: $a[i][j] \le a[i][j+1]$, за всяко $i = 1, 2, \ldots, M$ и $1 \le j < N$.

– *Наредба по колонки*: $a[i][j] \le a[i+1][j]$, за всяко $j = 1, 2, \ldots, N$ и $1 \le i < M$.

Определение. 2D масив, за елементите на който са в сила горните две свойства, ще наричаме *частично нареден* масив. Частните случаи на 2D масиви с размерност 1xN и Mx1, т.е. едномерните наредени масиви, също ще наричаме час-

тично наредени масиви. За частично нареден масив приемаме и масива с размери 1x1, т.е. масив от един елемент.

Основна задача [*Търсене в частично нареден масив*] Нека a да е 2D частично нареден масив, а x число от базовия тип данни на масива. Да се построи алгоритъм, с който се търси елемент от масива със стойност x, т.е. алгоритъм, с който се търсят индекси p и q на масива, $1 \le p \le M$, $1 \le q \le N$, такива че $a[p][q] = x$ или да се констатира, че в масива не съществува елемент със стойност x.

В някакъв смисъл формулираната задача е обобщение на задачата за търсене в нареден едномерен масив, в който случай задачата е перфектно решена с метода „Разделяй и владей“, а съответният алгоритъм е познат като „Двоично търсене“.

1.2. Свойства на 2D частично наредените масиви

Частично наредените 2D масиви притежават редица свойства. По-долу са посочени само тези от тях, които са в основата на алгоритмите за търсене, разгледани в тази статия.

Нека a е произволен 2D частично нареден масив, а x е произволно число от базовия тип на масива. В сила са следните свойства:

P1. Ако x е елемент от масива, то $a[1][1] \le x \le a[M][N]$. Ще акцентираме две очевидни, но важни следствия, които многократно се използват по-нататък:
– $a[1][1]$ е най-малкото число в масива, а $a[M][N]$ – най-голямото;
– Ако $x < a[1][1]$ или $x > a[M][N]$, тогава x не е елемент на a.

P2. Ако $a[M-1][N-1] < x < a[M][N]$, тогава x може да бъде елемент на масива само ако е в последия ред и/или колонка.

P3. Всяка правоъгълна област от 2D частично нареден масив е също 2D частично нареден масив, който по-нататък ще бъде наричан *подмасив*.

P4. Елементите на всеки диагонален ред, успореден на „главния диагонал“, са в ненамаляваща последователност. Понятието главен диагонал, когато масивът не е квадратен ($M \ne N$), е интуитивно ясно.

Доказателствата следват непосредствено. Ще докажем само свойство P1.

Доказателство. Ако x е елемент от масива, тогава съществува поне една двойка индекси p и q, $1 \le p \le M$, $1 \le q \le N$, такива, че $x = a[p][q]$. Можем да запишем следната редица от неравенства:

$$a[1][1] \le a[p][1] \le a[p][q] \le a[p][N] \le a[M][N]$$

1.3. Литературна справка

Формулираната по-горе задача е математическа, която в по-общ вид е изследвана още през 70-те години (Agarwal&Sharir, 1988; Linial&Saks, 1985), но публи-

кации се срещат и сега (Bird, 2006; Bird, 2010)[1]. С приложенията си в различни области, като например в компютърната графика и изчислителната геометрия, тази задача става още по-атрактивна. Разработени са разнообразни алгоритми за решението ѝ, а програмни реализации и коментари към тях, срещани в Internet, се публикуват и днес.[2,3]

Особен интерес се проявява към алгоритъма, наречен *Saddleback*, известен отпреди 40 години. Ръкописен материал за него се съдържа в архива на E. Dijkstra.[1,4] В него той пише "*The origin of the algorithm is unnkown; its name has been invented by David Gries*". В свои публикации, книги и лекционни материали[5] D. Gries разглежда този алгоритъм и днес алгоритъмът е един от класическите примери, които се представят в университетски курсове по алгоритми и формални методи за доказване на коректността на програми. Вече не е лесно да се посочат разнообразните идеи на модификации на алгоритъма Saddleback. Той е намерил място и в колекцията от „перли по проектиране на функционални алгоритми" на Richard Bird (2006; 2009).

Някои основни и добре познати идеи за алгоритми за търсене в 2D частично наредени масиви са разгледани накратко в т. 2. Представен е и един нов алгоритъм, описан в т. 3. В т. 4 е описана процедура за генериране на 2D частично наредени масиви, които се използват в т. 5 за експериментите с разгледаните алгоритми.

2. Известни алгоритми за решаване на основната задача

При словесното описание на алгоритмите индексите на масивите започват от 1, а не от 0, както това е при езиците C/C++. Означенията на променливите и параметрите са унифицирани по следния начин:

- a – име на частично нареден 2D масив с размерност MxN, M – брой редове и N – брой колонки;
- x е число, което се търси в 2D масива;
- $[\alpha..\beta]$ – интервал от индекси в масива a, посочващи поредни номера на редове или колонки , $\alpha \leq \beta$. Например с $[r1..r2]$ е означен интервалът от редове на масива от $r1$ до $r2$ включително.
- r, $r1$, $r2$, p, $p1$, $p2$ – индекси от интервала $[1..M]$, посочващи пореден номер на ред от a;
- c, $c1$, $c2$, q, $q1$, $q2$ – индекси от интервала $[1..N]$, посочващи пореден номер на колонка от a.

Реализизацията на всички алгоритми е извършена в среда на програмиране Microsoft Visual C++ 2010.

2.1. Алгоритъм „Saddleback“

Търсенето на елемент от масива със стойност x в алгоритъма Saddleback започва с елемента $a[r][c]$, който се намира в последния ред ($r = M$), първа колонка ($c = 1$). Ако $a[r][c] = x$, елементът е намерен и търсенето завършва. Ако $a[r][c] < x$, търсеният елемент не може да бъде в тази колонка и търсенето продължава в следващата колонка ($c + 1$). В противен случай, т.е. ако $a[r][c] > x$, търсеният елемент не може да бъде в този ред и се тества елементът от същата колонка, но от предходния ред ($r - 1$). Кодът на функцията, реализираща този алгоритъм, е даден по-долу:

```
bool alg1(int a[][N], int x)
{
   int r = M-1;
   int c = 0;
   while (r >= 0 && c < N)
     if (a[r][c] == x) return true;
     else if (a[r][c] < x) c = c + 1;
     else r = r - 1;
   return false;
}
```

Лесно се съобразява, че реализацията на алгоритъма Saddleback може да се опише и като се започне от най-горния десен елемент на масива (индекси [1][N]), а придвижването ще се извършва надолу (нарастване на номера на реда) и наляво (намаляване на номера на колонката).

2.2. Последователно-двоично търсене по редове (стълбове)

Понеже редовете (колонките) на частично наредените масиви са наредени едномерни масиви, то за всеки от тях може да се приложи алгоритъмът за двоично търсене. Това означава обединение на последователното търсене с двоичното търсене в една процедура, която накратко е наречена *последователно-двоично* търсене. Следва описанието на алгоритъма като функция (`alg2`), в която съществено се използва функцията `bSearchR` за двоично търсене в произволен ред r от масива a, $r = 1, 2, \ldots, M$.

```
bool alg2(int a[][N], int x)
{
   for (int r = 0; r < M; r++)
    if (bSearchR(a, r, 0, N-1, x)) return true;
   return false;
}
```

Функцията `bSearchR` ще се изпълни не повече от M пъти, което означава, че времето за изпълнение на тази функция е const$x M x lg(N)$. Същата идея може да се

приложи и по колонки. Една предварителна проверка за стойностите на M и N ще реши кой от двата варианта да бъде използван в конкретния случай – търсене по редове или търсене по колонки.

2.3. Двумерен вариант на двоичното търсене

В този алгоритъм „средният“ елемент ще бъде някъде в „средата“ на масива с индекси [M/2][N/2]. Чрез средния елемент масивът се „разбива“ на четири подмасива, всеки от които е частично нареден масив (свойство P3). Един пример е даден на фиг. 1, в който $M = 9$, $N = 10$, $M/2 = 4$, $N/2 = 5$. Средният елемент в масива е $a[4][5] = 19$, а четирите области A, B, C и D са определени с номерата на двойките индекси по редове и колонки, както следва: подмасив A: ([1..4], [1..5]), подмасив B: ([1..4], [6..10]), подмасив C: ([5..9], [1..4]) и подмасив D: ([5..9], [6..10]).

Интересни за разглеждане са следните случаи:

a) Ако $a[M/2][N/2] = x$, елементът е намерен и търсенето завършва с успех.
b) Ако $x < a[M/2][N/2]$, търсеният елементът не може да бъде в подмасив D. (свойство P1) и търсенето трябва да продължи в подмасивите A, B и C.
c) Ако $a[M/2][N/2] < x$, търсеният елементът не може да бъде в подмасив A (свойство P1) и търсенето следва да продължи в подмасивите B, C и D.
d) Този процес на „разбиване“ на подмасиви продължава до намирането на търсения елемент или до достигане на подмасив, в който единият или и двата му размера са равни на нула.

Специално ще отбележим, че подмасивите B и C присъстват и в двата случая b) и c), което трябва се има предвид при реализацията на алгоритъма (`alg3Rec`). Параметрите $r1$, $r2$, $c1$ и $c2$ се използват като индекси, с които на всяка стъпка се определят подмасивите A, B, C и D на текущия масив.

	1	2	3	4	5	6	7	8	9	10
1	2	5	6	8	10	14	14	16	19	42
2	3	6			11	18	2		4	44
3	7	8		11	14	22	2		6	48
4	11	12	13	14	**19**	24	31	33	38	50
5	12	14	15	15	20	27	36	37	40	45
6	12	15	16	18	23	27	37	47	48	55
7	14	2		5	27	30	4		9	56
8	18	24	30	31	33	37	40	50	51	58
9	30	32	38	40	44	46	47	52	54	60

Фигура 1. Масив, „разбит“ на четири подмасива от средния елемент a[4][5] = 19

```
bool alg3(int a[][N], int x)
{
  return alg3Rec(a, 0, M-1, 0, N-1, x);
}

// Функция, реализираща алгоритъма
   „Двумерен вариант на двоично търсене“
// Обръщение към тази функция извършва функцията alg3
bool alg3Rec(int a[][N], int r1, int r2, int c1, int c2, int x)
{
  // Случаи, при които търсенето завършва с неуспех
  if (r1 > r2 || c1 > c2) return false;
  if (r1 == r2 && c1 == c2 && a[r1][c1] != x) return false;
  if ((x < a[r1][c1]) || (x > a[r2][c2])) return false; // Свойство P1

  // Декомпозиция на общия случай на четири подобни подслучая
  int r = (r1 + r2)/2;                          // Индекси на
  int c = (c1 + c2)/2;                          // средния елемент
  if (a[r][c] == x) return true;
  else if (r1 == r2 && c1 == c2) return false;

  if (alg3Rec(a, r1, r, c+1, c2, x))
    return true;                                // Подмасив B
  if (alg3Rec(a, r+1, r2, c1, c, x))
    return true;                                // Подмасив C
  if (x < a[r][c])
    return alg3Rec(a, r1, r, c1, c, x);         // Подмасив A
  else
    return alg3Rec(a, r+1, r2, c+1, c2, x);     // Подмасив D
}
```

Началните стойности на параметрите $r1$, $r2$, $c1$ и $c2$, с които функцията `alg3Rec` се изпълнява, са съответно $[r1..r2] = [0..M\text{-}1]$ и $[c1..c2] = [0..N\text{-}1]$. Те определят границите на първоначалния масив.

2.4. Алгоритъм за търсене в средния ред на масива

Вместо да се избира елемент в средата на масива, търсенето в този алгоритъм започва с двоично търсене в средния ред на масива. След това се определя подходящата средна колонка. Ето основните стъпки в алгоритъма:

1. Извършва се търсене в средния ред $r = [M/2]$ на масива. Ако търсеният елемент е намерен, търсенето завършва с успех.
2. Ако търсеният елемент не е в ред r, тогава:

– Ако $a[r][1] > x$ търсенето продължава в подмасив $A1$, определен от редове $[r1, r-1]$.
– Ако $a[r][N] < x$ търсенето продължава в подмасив $A2$, определен от редове $[r+1, r2]$.

3. Ако търсеният елемент не е в ред r, но в него има позиция q, такава че: $a[r][q] < x < a[r][N]$. В този случай търсенето продължава в два подмасива B и C.
   – Масивът B е определен с редове $[r1, r-1]$ и колонки $[q+1, c2]$.
   – Масивът C е с редове $[r1+1, r2]$ и колонки $[c1, q]$.
4. За всеки от подмасивите $A1$, $A2$, B и C търсенето продължава от т. 1.

По-долу следва кодът на рекурсивната функция с име `alg4Rec`, реализираща алгоритъма. Параметрите $r1$, $r2$, $c1$и $c2$ се използват като индекси, с които на всяка стъпка се определят подмасивите A и B на текущия масив.

```
bool alg4(int a[][N], int x)
{
  return alg4Rec(a, 0, M-1, 0, N-1, x);
}

// Функция, реализирана по метода „Търсене в средния ред на масива“
bool alg4Rec(int a[][N], int r1, int r2, int c1, int c2, int x)
{
  // Случаи, при които търсенето завършва с неуспех
  if (r1 > r2 || c1 > c2) return false;
  if (r1 == r2 && c1 == c2 && a[r1][c1] != x) return false;
  if ((x < a[r1][c1]) || (x > a[r2][c2])) return false;// Свойство P1

  // Декомпозиция на общия случай на два подобни подслучая
  int q;
  int r = (r1 + r2)/2;                    // Индекс на средния ред
  if (bSearchR(a, r, c1, c2, x, q)) return true;

  return alg4Rec(a, r + 1, r2, c1, q, x) || // Област C
         alg4Rec(a, r1, r, q + 1, c2, x);   // Област B
}
```

Началните стойности на параметрите $r1$, $r2$, $c1$и $c2$, с които функцията `alg4Rec` се изпълнява, са съответно $[r1..r2] = [0..M-1]$ и $[c1..c2] = [0..N-1]$.

3. Алгоритъм за търсене чрез редица от концентрични подмасиви

Водеща идея в този алгоритъм е конструирането на редица от подмасиви с намаляваща размерност, която има следните две свойства:

– [C1] Всеки подмасив от редицата, с изключение на първоначалния, изцяло се съдържа в предхождащия го. Тази е и причината редицата от подмасиви да бъде наречена редица от *концентрични* масиви.
– [C2] Ако първоначалният масив съдържа елемент с търсената стойност x, то този елемент се съдържа във всеки подмасив от редицата.

Тъй като размерността на всеки следващ подмасив в редицата е намаляваща, то поне една от размерностите на всеки следващ подмасив от редицата ще е по-малка от съответната размерност на предхождащия го подмасив. Като следствие от това и от свойствата C1 и C2, редицата от подмасиви ще клони към подмасив, съдържащ само един елемент. Стойността на този елемент ще определи крайния резултат от търсенето. Ако стойността му е равна на x, търсенето завършва с успех. В противен случай първоначалният масив не съдържа елемент с търсената стойност.

3.1. Построяване на редицата от концентрични подмасиви

В алгоритъма, представен по-долу, процедурата за двоично търсене се прилага четирикратно за определянето на всеки следващ подмасив от редицата.

1. Началният масив (подмасив) съдържа редовете $[r1..r2] = [1..M]$ и колоните $[c1..c2] = [1..N]$.
2. В интервала от редовете $[r1..r2]$ и колонката $c1$ се търси ред с номер $p1$ такъв, че елементът $a[p1][c1]$ да е равен на x или $a[p1][c1]$ да е най-голямото число в колонката $c1$, което да не е по-голямо от x (фиг. 2).
3. В реда $p1$ между колонките $[c1..c2]$ се търси колонка $q1$ такава, че $a[p1][q1]$ да е равен на x или $a[p1][q1]$ да е най-голямото число в колонката $q1$, което не е по-голямо от x. По този начин се определят номерата на реда $r2 = p1$ и на колонката $c1 = q1$ на новия подмасив.
4. В интервала от колонките $[c1..c2]$ и в реда $r1$ се търси колонка с номер $q2$ такава, че елементът $a[r1][q2]$ да е равен на x или $a[r1][q2]$ да е най-голямото число в реда $r1$, което не е по-голямо от x.
5. В колонката $q2$ между редовете $[r1..p]$ се търси ред $p2$ такъв, че $a[p2][q2]$ да е равен на x или $a[p2][q2]$ да е най-голямото число в колонката $q2$, което не е по-голямо от x. По този начин се определя номерът на горния ред $r1 = p2$ и номерът на дясната колонка $c2 = q2$ на новия подмасив.

От стъпки 2 и 3 на алгоритъма се определя долният ляв елемент на новия подмасив: ред $r2 = p1 - 1$ и колонка $c1 = q1 + 1$. От стъпки 4 и 5 се определя горният десен елемент на новия подмасив: ред $r1 = p2 + 1$ и колонка $c2 = q2 - 1$.

С описаната процедура от текущия подмасив се отстраняват областите, несъдържащи търсения елемент. Търсенето ще продължи в нов подмасив, в който е възможно да се съдържа търсеният елемент. Фиг. 2 илюстрира опеделянето на

новия подмасив. Маркираната област от масива е новият подмасив от редицата.

Ясно е, че конструираният нов масив е подмасив на дадения. Ще докажем, че ако x е елемент на първоначалния масив, то той се съдържа и в подмасива.

Доказателство. Начинът, по който са пресметнати $p1$, $q1$ и $p2$, $q2$, елеминира търсенето в подмасивите ($[r1..r2]\ [c1..q1]$) и ($[r1..p2][c1..c2]$).

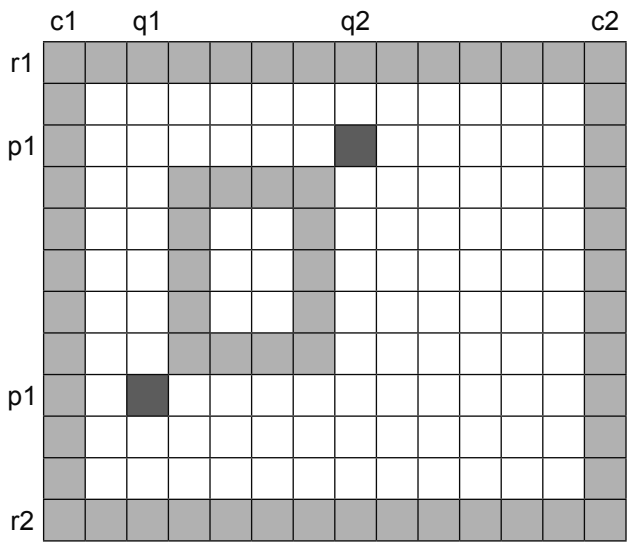
В сила са и следните неравенства:

$a[p1][c1] < x < a[p1+1][c1] \le a[i][j]$ за $i = p1+1, p1+2, \ldots, M; j = 1, 2, \ldots, N$.

Първите две неравенства следват от избора на $p1$, а последното – от свойство P1. Аналогично могат да се запишат и неравенствата:

$a[r1][q2] < x < a[r1][q2+1] \le a[i][j]$ за $i = 1, 2, \ldots, M; j = q2+1, q2+2, \ldots, N$.

С тези две системи от неравенства се елиминират подмасивите, определени с интервалите от индекси ($[p1+1..r2][c1..c2]$) и ($[r1..r2][q2+1..c2]$). Всичко това показва, че ако търсеният елемент x е в първоначалния масив, той остава и във всеки следващ подмасив на редицата.



Фигура 2. Илюстрация на алгоритъма, използващ редица от подмасиви

Следва кодът на функцията `alg5`, реализираща описания алгоритъм. Както се вижда, централна роля в нея играят функциите за двоично търсене по редове `bSearchR` и по колонки `bSearchC`. Всяка от тези функции се прилага по два пъти за определянето на координатите на всеки следващ подмасив от редицата.

3.2. Програмен код на алгоритъма

```
bool alg5(int a[][N], int x)
{
   // Начален масив
   int r1 = 0, r2 = M-1;            // Редове  [0 .. M-1]
   int c1 = 0, c2 = N-1;            // Колонки [0 .. N-1]
   int p, q;

   while ((r1 <= r2) && (c1 <= c2))
   {
      // Двоично търсене в колонка c1 между редове [r1, r2]
      if (bSearchC(a, c1, r1, r2, x, p)) return true;
      else
      { // Двоично търсене в ред p между колонки [c1, c2]
        if (bSearchR(a, p, c1, c2, x, q)) return true;
        r2 = p - 1;                 // Нов ред
        c1 = q + 1;                 // Нова колонка
      }

      // Двоично търсене в ред r1 между колонки [c1, c2]
      if (bSearchR(a, r1, c1, c2, x, q)) return true;
      else
      { // Двоично търсене в колонка q между редове [r1, p]
        if (bSearchC(a, q, r1, p, x, p)) return true;
        c2 = q - 1;                 // Нова колонка
        r1 = p + 1;                 // Нов ред
      }

      // Проверка дали елементът е намерен
      if ((r1 == r2) && (c1 == c2) && (a[r1][c1] == x))
        return true;
   }
   return false;
}
```

4. Обратна на основната задача

За извършването на експерименти с представените алгоритми е необходимо да се генерират 2D частично наредени масиви със случайни числа. Това налага решаването на още една задача.

4.1. Дефиниция и алгоритъм на задачата

Задача. Дадени са целите положителни числа M, N, p, q, x, y и z, за които са изпълнени следните условия: $1 \le p \le M$, $1 \le q \le N$ и $y \le x \le z$. Да се генерира 2D масив a с размерност MxN, имащ свойствата:

- Масивът a е частично нареден, а елементите му са случайни цели числа в интервала $[y .. z]$.
- $a[1][1] = y$, $a[M][N] = z$ и $a[p][q] = x$.

Всеки 2D масив може да се разглежда като едномерен, в който редовете (колонките) на двумерния масив се поставят една след друга, т.е. след елементите на първия ред следват елементите на втория и т.н. Пресмятането на позицията (индекса) k на елемент от едномерния масив, съответен на елемент от двумерния масив с индекси $[p][q]$, може да се извърши с линейна функция, която ще зависи от p, q и N. Тук трябва да се има предвид, че индексът на първия елемент в езиците C/C++ е нула. Тогава общият вид на функцията *address*, с която се пресмята адресът на елемента $a[p][q]$ в едномерното представяне на двумерния масив, ще бъде:

$$k = address(p, q, \mathrm{N}) = \mathrm{N}p + q$$

Като се използват направените по-горе уточнения, решението на задачата може да се формулира така:

1. Дефинира се едномерен масив b с MxN елемента;
2. Генерират се MxN случайни числа в интервала $[y..z]$ като елементи на масива масива b.
3. Масивът b се сортира в нарастващ ред на елементите му.
4. Масивът b се разглежда като едномерно представяне на двумерния масив a по редове.

Определеният по този начин масив a с помощта на едномерния масив b е частично нареден, защото масивът b е сортиран. Това означава, че: $b[k1] \le b[k2]$ за $k1 \le k2$, т.е. 2D масивът a е нареден по редове и по колонки.

Остава нерешен още един въпрос – елементът $a[p][q]$ трябва да е със стойност x. Това лесно се постига, като интервалът от индекси $[0 .. MN\text{-}1]$ се раздели на две части $[0.. Np + q\text{ -}1]$ и $[Np + q, MN\text{ -}1]$. За елементите от масива b с индекси от първия интервал ще се генерират случайни числа в интервала $[y .. x]$, а за елементите от втория интервал ще се генерират случайни числа в интервала $[x .. z]$.

4.2. Програмен код за генериране на 2D частично наредени масиви

Генерирането на случайни числа – елементи на частично нареден 2D масив, се извършва с трите функции, чиито дефиниции следват по-долу.

```
void syn2Darray(int a[][N])
{
   int size = M*N;
   for (int k = 0; k < size; k++)     // Генериране на MxN
     a[k/N][k%N] = rNumb();           // случайни числа

// „Бързо“ сортиране на масива с алгоритъм от STL библиотеката
   qsort(a, size, sizeof(int), compare);
}

// Функция, генерираща случайно число в интервала
  [MINVAL .. MAXVAL]
int rNumb()
{
   const double DBLMXRD = double(RAND_MAX) + 1;
   double r = double(rand())/DBLMXRD;
   return int(r * (MAXVAL - MINVAL + 1)) + MINVAL;
}

// Дефиниция на функцията „compare“ за сравняване на две
// числа. Използва се от полиморфната функция qsort от
// библиотеката с алгоритми STL
int compare (const void * a, const void * b)
{
  return ( *(int*)a - *(int*)b );
}
```

5. Експерименти. Измерване на времето със синтетично генерирани частично наредени 2D масиви

С разработените алгоритми са направени разнообразни експерименти за пресмятане на времето им за изпълнение. Част от тези данни са представени по-долу в таблици и графики. Дадена е и частта от управляващия модул на програмата, с която са извършени експериментите. Основни параметри на експериментите са различните стойности на размерностите M и N на масивите. Те включват и трите случая: $M < N$, $M > N$ и $M = N$. За всички тези случаи са извършени два основни вида експерименти, когато търсеното число е елемент на масива и когато търсеното число не е елемент на масива. Времената за изпълнение на всички алгоритми са получени от изпълнението им с едни и същи 2D частично наредени масиви и едни и същи числа x за търсене в тях. Максималният размер на масивите, с

които са извършени изпълненията, е 250,000 елемента. За всяка двойка (*M*, *N)* са генерирани по 10 масива. За стойност на числото *x*, което се търси, е използван всеки един от елементите на масива. Това означава, че всеки от алгоритмите е изпълнен общо 10x*M*x*N* пъти при търсенето, което е завършвало с успех за всяка двойка (*M*, *N*). Същият брой експерименти е извършен и в случаите, при които числото не е елемент на масива.

Графичната илюстрация на времената за изпълнение на алгоритмите за определени стойности на *M* и *N* е получена в средата на програмиране MATLAB.[6]

5.1. Текст на управляващия модул на програмата

Петте алгоритъма са реализирани като функции с еднакви сигнатури. Това позволява да се разглеждат като масив от функции `{alg1, alg2, alg3, alg4, alg5}`. С това се постига унифицирано обръщение към всички функции с единствен оператор. Той се намира в тялото на най-вътрешния цикъл на управляващия модул.

```
const int NRALG = 5;          // Брой на алгоритмите
const int NRARR = 10;         // Брой на генерираните масиви
const int MINVAL = 1;         // Минимално число в масива
const int MAXVAL = 10000000; // Максимално число в масива
const int M = ...;            // Брой на редовете на масива
const int N = ...;            // Брой на колонките на масива

// Масив с имената на алгоритмите
const string algName[NRALG] = {
   “Algorithm 1”, // Алгоритъм Saddleback
   “Algorithm 2”, // Последователно-двоично търсене по редовете
   “Algorithm 3”, // Двумерен вариант на двоичното търсене
   “Algorithm 4”, // Търсене в средния ред на масива
   “Algorithm 5”  // Алгоритъм с концетрични подмасиви
};

// Прототипи на функциите, реализиращи петте алгоритъма
bool alg1(int a[][N], int x);
bool alg2(int a[][N], int x);
bool alg3(int a[][N], int x);
bool alg4(int a[][N], int x);
bool alg5(int a[][N], int x);

// Прототип на функция (масив от функции), която извършва
// обръщение към функциите, които се изследват. Масивът f
```

```
  // съдържа имената на функциите, декларирани по-горе
  bool (*f[NRALG])(int [][N], int x) = {alg1, alg2, alg3, alg4,
alg5};

  // Рекурсивна функция, реализираща Алгоритъм 3.
  // Обръщение към нея извършва в alg3.
  bool alg3Rec(int a[][N], int r1, int r2, int c1, int c2, int x);

  // Рекурсивна функция, реализираща Алгоритъм 4.
  // Обръщение към нея извършва в alg4.
  bool alg4Rec(int a[][N], int r1, int r2, int c1, int c2, int x);

  // Следват прототипи на помощни функции
  // . . .

  int main()
  {
     double totalTime[NRALG] = {0.0}; // Масив за натрупване
                                      // на времената
                          // за изпълнение на отделните алгоритми
     int a[M][N];         // 2D частично нареден масив
     bool algResult;
     clock_t start, end; // Променливи, необходими за измерване на
                         // времето за изпълнение на алгоритмите
     srand(unsigned(time(0)));  // Стартиране на генератора
                                // на случайни числа

     // Печат на основните параметри на експеримента
     cout << "Number of algorithms = " << NRALG << endl;
     cout << "Number of arrays     = " << NRARR << endl;
     cout << "Minimum number       = " << MINVAL << endl;
     cout << "Maximum number       = " << MAXVAL << endl;
     cout << "Number of rows       = " << M << endl;
     cout << "Number of columns    = " << N << endl;

     for (int nrArr = 0; nrArr < NRARR; nrArr++)// Всеки от
                                                // алгоритмите се
     {                                          // изпълнява с
                                                // NRARR масива
```

```
      syn2Darray(a);                  // Генериране на синтетичен
                                      // 2D частично нареден масив
      for (int alg = 0; alg < NRALG; alg++) // Цикъл за изпълнение
                                            // на всеки
      {                                     // алгоритъм
        start = clock();                    // Стартиране на
                                            // часовника за alg
        for (int i = 0; i < M; i++)         // Алгоритъмът alg
                                            // се изпълнява MxN
          for (int j = 0; j < N; j++)       // пъти с всеки
                                            // елемент на масива
           algResult = (*f[alg])(a, a[i][j]);// Обръщение към
                                            // алгоритъм alg[1..5]
        end = clock();            // Спиране на часовника за alg
        // Сумарно време за изпълнение на алгоритъма alg [1..5]
        totalTime[alg] += (double)(end-start)/CLOCKS_PER_SEC;
      }
    }

    cout << fixed << setprecision(3);
    for (int alg = 0; alg < NRALG; alg++)
      cout << algName[alg] << ", time = " << setw(8)// Печат на
                                                   // времето
           << totalTime[alg] << " seconds" << endl; // за всеки
                                                    // алгоритъм
    return 0;
}
```

5.2. Резултати от проведените експерименти

Част от извършените експерименти са обобщени в няколко таблици и графики. В експеримент 1 стойността на M е фиксирана на 250 реда, а броят на колонките се изменя от 100 до 1000 със стъпка 100 (Табл. 1). При експеримент 2 броят на колонките N е фиксиран и е равен на 250, а броят на редовете се променя от 100 до 1000 със стъпка 100 (Табл. 2). Може да се забележи, че и в двата експеримента алгоритмите 1, 4 и 5 имат сходни времена за едни и същи стойности на MxN. Вижда се също, че алгоритмите 2 и 3 дават големи отклонения при експеримент 2 ($M > N$) спрямо тези от експеримент 1, което е очаквано.

На фиг. 3 графично са илюстрирани времената от експериментите 1 и 2, но само на три от алгоритмите (1, 4 и 5). Времената за изпълнение на всички алгоритми в двата случая се различават, особено тези на Алгоритми 2 и 3 (Табл.1, 2),

но съотношенията на времената между всичките алгоритми и в двата случая се запазват.

Експеримент 1

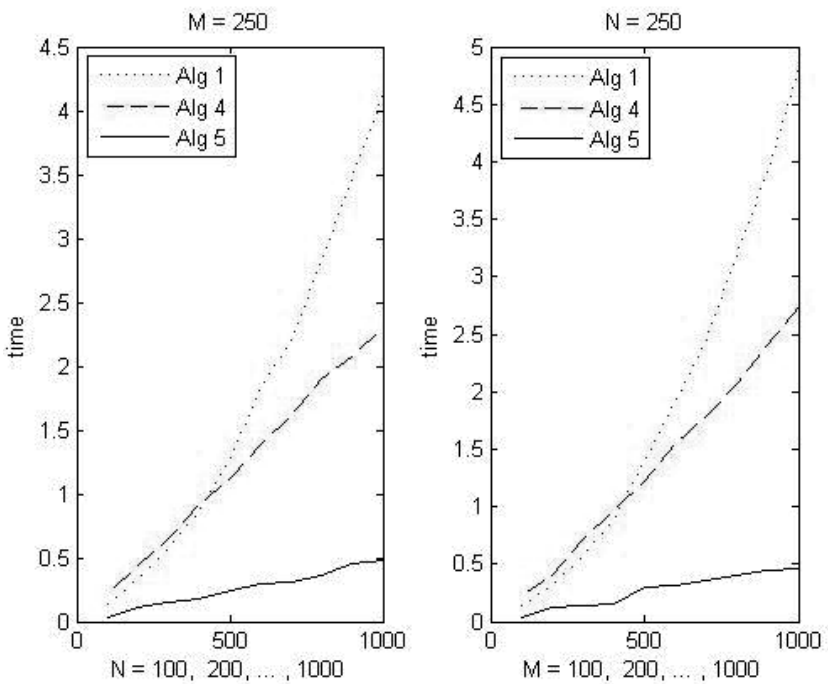
Алг. 1	Алг. 2	Алг. 3	Алг. 4	Алг. 5
0.136	1.994	1.863	0.215	0.035
0.329	4.320	4.010	0.437	0.110
0.578	7.035	6.145	0.657	0.155
0.859	9.546	8.143	0.921	0.172
1.280	11.807	10.202	1.126	0.249
1.839	15.212	12.231	1.388	0.296
2.199	17.597	14.181	1.606	0.311
2.825	20.217	16.208	1.902	0.359
3.461	22.682	18.317	2.074	0.454
4.134	25.117	20.264	2.309	0.481

Таблица 1. $M = 250$; $N = 100, 200, \ldots, 1000$

Експеримент 2

Алг. 1	Алг. 2	Алг. 3	Алг. 4	Алг. 5
0.140	0.891	0.949	0.221	0.032
0.312	3.510	3.542	0.406	0.125
0.560	8.035	7.891	0.703	0.141
0.905	13.822	13.496	0.965	0.157
1.403	21.670	19.281	1.216	0.297
1.902	31.280	27.032	1.544	0.314
2.465	42.042	34.866	1.794	0.359
3.198	55.209	42.853	2.075	0.406
3.915	70.482	50.934	2.402	0.452
4.837	86.345	58.064	2.747	0.465

Таблица 2. $M = 100, 200, \ldots, 1000$ и $N = 250$



Фигура 3. Графики на резултатите от изпълненията на три от алгоритмите (1, 4 и 5)

Експеримент 3

M = N	Алг 1	Алг 2	Алг 3	Алг 4	Алг 5
100	0.016	0.312	0.358	0.078	0.016
150	0.139	1.140	1.294	0.188	0.015
200	0.249	2.780	2.868	0.328	0.094
250	0.421	5.414	5.022	0.531	0.109
300	0.734	10.030	9.314	0.826	0.171
350	1.204	16.050	14.930	1.186	0.232
400	1.843	24.504	21.687	1.622	0.312
450	2.527	34.086	29.703	2.029	0.373
500	3.527	46.457	38.313	2.495	0.469

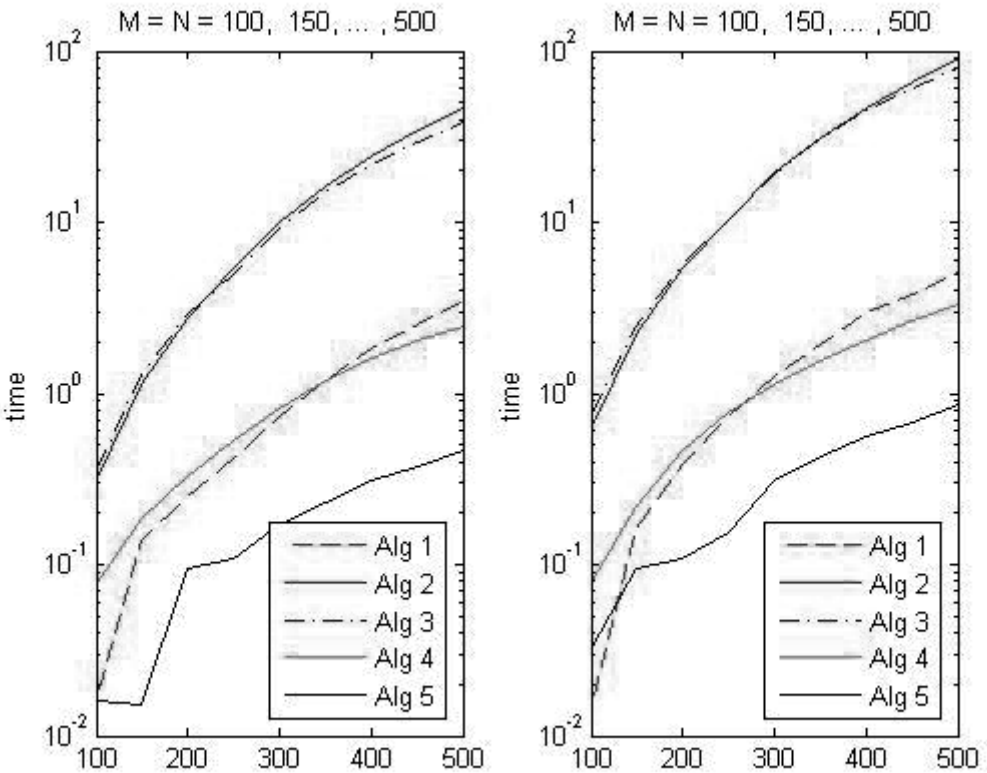
Таблица 3. M = N

При експеримент 3 са измерени времената на петте алгоритъма при $M = N$ с начална стойност 100, крайна стойност 500 и стъпка 50 в случая, когато търсеният елемент е елемент на масива (Табл. 3). Същият експеримент е извършен и в случая, когато търсеното число не е елемент на масива. Графично резултатите от двата експеримента са представени на фиг. 4. Поради голямата разлика във времената на Алгоритми 2 и 3 по отношение на останалите е използвана логаритмична скала по оста *Oy*. В лявата графика е представен случаят, когато търсеното число е елемент на масива, а дясната показва случая, когато числото не е елемент на масива. Въпреки че се забелязва известно нарастване на времената във втория случай, съотношението на времената между отделните алгоритми се запазава.

Експеримент 4

M	N	Алг. 1	Алг. 2	Алг. 3	Алг. 4	Алг. 5
1	250000	781.503	0.424	2.276	0.469	0.500
5	50000	155.017	0.984	2.137	0.952	0.482
10	25000	77.360	1.747	2.308	1.248	0.470
20	12500	38.750	3.120	2.916	1.453	0.466
250000	1	809.484	9598.542	2.813	2.974	0.464
50000	5	172.694	2384.915	2.729	3.106	0.421
25000	10	91.024	1381.944	3.776	3.228	0.436
12500	20	53.821	774.307	9.218	3.167	0.453

Таблица 4. Времена на алгоритмите с няколко екстремални стойности на (M,N)



Фигура 4. Графики на резултатите от изпълненията на петте алгоритъма при $M = N$

Интересно е да се разгледат и резултатите от Табл. 4, в която има няколко екстремални случая, като например $M = 1$, $N = 250000$. В този случай двумерният масив се изражда в едномерен, състоящ се само от един ред. Аналогичен е и другият случай, при който масивът се изражда само в една колонка, $M = 250000$ и $N = 1$.

Във всички разгледани случаи Алгоритъм 5 се е изпълнил за най-кратко време. Изпълненията му не са повлияни от това, дали $M < N$, $M > N$ или $M = N$. Това му поведение се запазва и в случаите, когато двумерният масив е с големи различия в стойностите на M и N, включително и при изграждането на двумерния масив до едномерен. От Табл. 4 се вижда, че времената на Алгоритъм 1 са твърде високи при големи разлики в стойностите на M и N. Интересни, но не изненадващи, са резулатите на Алгоритъм 2. При $M = 1$ и $N = 250000$ времето му за изпълнение е най-добро от всички алгоритми за всички случаи на $M \times N = 250000$.

Направените по-горе коментари към времевите характеристики на разгледаните алгоритми не са математически обосновани. Те са основават на данните, получени експериментално, но резултатите от експериментите са достатъчно ясни, за да подсказват идеи за формулиране на хипотези, валидността на които следва да бъде доказана.

6. Проект в развитие

Като близки до основната задача, разгледана в статията, могат да се формулират редица други задачи за частично наредени масиви. Списък от шест такива задачи е даден по-долу. За всяка от тях се иска проектиране и разработка на съответен алгоритъм. Времевата характеристика на алгоритмите не е без значение. Реализирането на алгоритмите на определен език за програмиране и извършването на съответни експерименти ще внесе допълнителна яснота относно практическата им стойност.

Задача 1. Свойство P2 дава идея за алгоритм, който започва с разглеждането на най-големия елемент на масива, $a[M][N]$. Ето основните случаи, които трябва да се разглеждат в този алгоритъм:

a) Ако $a[M][N] = x$, елементът е намерен и търсенето завършва с успех.
b) Ако $x > a[M][N]$, масивът не съдържа елемент със стойност x и търсенето завършва с неуспех (съгласно P1).
c) Ако $a[M\text{-}1][N\text{-}1] = x$, елементът е намерен и търсенето завършва с успех.
d) Ако $a[M\text{-}1][N\text{-}1] < x$, търсенето има смисъл да продължи само в M-тия ред и N-та колонка (съглано P2).
e) Ако $x < a[M\text{-}1][N\text{-}1]$, търсенето продължава в подмасива на първоначалния масив, в който са отстранени последният ред и последната колонка. Новият подмасив е 2D частично нареден (съгласно P3).

Да се постри алгоритъм по описаната идея и да се извършат екперименти, подобни на описаните.

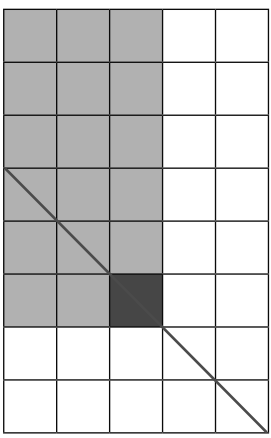
Задача 2. Тази задача е обобщение на задача 1, в която се прилага и свойство P4. На фиг. 5 и 6 са дадени два случая на правоъгълни масиви ($M \quad N$), в които с права линия са посочени елементите от диагонала, започващ от елемента с индекси $[M][N]$. Съгласно свойство P4 всички елементи от диагонала са наредени в ненамаляващ ред. В случай че масивът е квадратен, посочените диагонали представляват главния диагонал в съответния масив.

Ето основните случаи, които трябва да се разгледат:

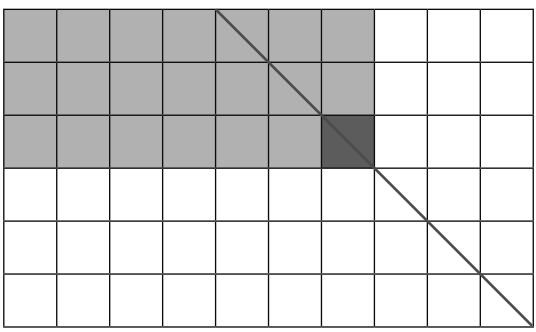
a) Ако $a[M][N] = x$, елементът е намерен и търсенето завършва с успех.
b) Ако $x > a[M][N]$, масивът не съдържа елемент със стойност x и търсенето завършва с неуспех (съгласно P1).

c) Ако случаите a) и b) не са в сила, тогава се прилага двоично търсене в диагоналния ред за намиране на елемент със стойност x. Ако такъв елемент се намери, търсенето завършва с успех. Ако няма такъв елемент, тогава се търси елемент, чиято стойност е най-голямата в диагонала, но не по-голяма от x. На двете фигури това е защрихованият елемент. Ако индексите му са $[p][q]$, тогава търсенето продължава в подмасивите $[p+1..M][1..N]$ и $[1..p][q+1..N]$, за които се прилага процедурата от по-горе.

d) Елементът с индекси [1][1] и защрихованият елемент определят подмасив, в който със сигурност няма елемент със стойност x.



Фигура 5. Правогъгълен масив, $M > N$



Фигура 6. Правогъгълен масив, $M < N$

Да се постри алгоритъм по описаната идея и да се извършат екперименти, подобни на описаните в т. 5.

Задача 3. Да се построи алгоритъм, с който се намира най-малкото число в 2D частично нареден масив, което не е елемент на масива.

Задача 4. Да се построи алгоритъм за намиране на всичките елементи на 2D частично нареден масив с дадена стойност.

Задача 5. Да се построи алгоритъм, с който се пресмята броят на елементите в 2D частично нареден масив, които са в даден инервал.

Задача 6. Да се построи алгоритъм за търсене в 3D частично нареден масив.

БЕЛЕЖКИ

1. Dijkstra E. (1985) The Saddleback Search. Note EWD-934. Available at http://www.cs.utexas.edu/users/EWD/index09xx.html
2. Gidney C. Searching a Sorted Matrix Faster. http://twistedoakstudios.com/blog/Post5365_searching-a-sorted-matrix-faster
3. Searching a 2D Sorted Matrix (Parts I, II, and III)

http://leetcode.com/2010/10/searching-2d-sorted-matrix.html

4. E. Dijkstra е удостоен през 1972 г. с Turing Award (присъждана от ACM) за фундаментален принос в областта на езиците за програмиране.
5. Gries D. Saddleback Search.

http://www.cs.geneseo.edu/~baldwin/math-thinking/saddleback.html

http://www.cs.cornell.edu/Courses/cs6110/2012sp/notes/griesLectureOnAlgorithms.pdf

6. MATLAB. Latest Release R2013b. http://www.mathworks.com/

ЛИТЕРАТУРА

Agarwal, P. & Sharir, M. (1998). Efficient Algorithms for Geometric Optimization. *ACM Computing Surveys*, vol. 30, No. 4, 412 – 458

Bird, R. (2006). Improving saddleback search: a lesson in algorithm design. *Mathematics of Program Construction*. Springer, LNCS 4014, 82 – 89.

Bird R. (2010). Improving on saddleback search. *Pearls of Functional Algorithm Design*. Cambridge University Press, 12 – 20.

Linial, N & Saks, M. (1985). Searching ordered structures. *Journal of Algorithms*. Vol 6, Issue 1, 86 – 103.

2D ARRAYS: SEARCH ALGORITHMS AND EXPERIMENTS

Abstract. The paper presents search algorithms applied to 2D partially ordered numeric arrays, i.e. the elements of the arrays are nonincreasing in both row and column. Four existing algorithms are discussed in the text. One of them is the Saddleback algorithm. A new *search algorithm through concentric subarrays* is offered by the author. The main idea of this algorithm consists of the construction of a series of subarrays with a decreasing number of their rows and columns. Each next subarray is fully contained in the previous subarray. This means that if the searched number is in the initial array, it belongs to all next subarrays. The last subarray of this series will consist of one single element which equals the searched number. All presented five algorithms

are implemented in the programming environment of Microsoft Visual C++. Additional experiments were conducted to find out the respective execution time for each of these five algorithms by selecting a variety of values for the number of rows and columns. Some of the obtained results are presented in tables, included in the text of the paper, and are also accompanied by the corresponding charts. The best values for the execution time are established for the search algorithm through concentric subarrays.

✉ **Dr. Pavel Azalov, Assoc. Prof.**
Pennsylvania State University
Hazleton campus
U.S.A.
E-mail: pk10@psu.edu