Information Processing Letters 20 (1985) 13–17
North-Holland

2 January 1985

SELECTION IN X + Y AND MATRICES WITH SORTED ROWS AND COLUMNS *

A. MIRZAIAN and E. ARJOMANDI

Department of Computer Science, York University, Downsview, Ontario M3J 1P3, Canada

Communicated by L. Boasson
Received February 1984
Revised June 1984

Let A be an $n \times n$ matrix of reals with sorted rows and columns and k an integer, $1 \leqslant k \leqslant n^2$. We present an O(n) time algorithm for selecting the k*th* smallest element of A. If X and Y are sorted n-vectors of reals, then the Cartesian sum $X + Y$ is such a matrix as A. One application of selection in $X + Y$ can be found in statistics. The algorithm presented here is based on a new divide-and-conquer technique, which can be applied to similar order related problems as well. Due to the fact that the algorithm has a relatively small constant time factor, this result is of practical as well as theoretical interest.

Keywords: Matrix, sorting, divide-and-conquer

1. Introduction

In this paper we consider the selection problem in matrices with sorted rows and columns. Selection in $X + Y$, where X and Y are sorted vectors, is a special case of this problem.

Let $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$ be two vectors of real numbers. The Cartesian sum $X + Y$ is the $n \times n$ matrix with ij*th* entry $x_i + y_j$. If X and Y are sorted, then $X + Y$ is a matrix with sorted rows and columns. Selection and other related problems in $X + Y$ have received considerable attention, due to their application in statistics and operations research [2,3,4,5,6,7,8,10]. $X + Y$ order related problems arise in some VLSI layout problems as well [9].

Jefferson, Shamos and Tarjan [10] present an O(n log n) time algorithm for selecting the median of $X + Y$. Johnson and Mizoguchi [8] give an O(n log n) algorithm for selecting the k*th* smallest element in $X + Y$. Both algorithms in [10] and [8] sort the vectors X and Y before the algorithms may proceed. Despite the time required to sort X and Y, both these algorithms still require O(n log n) time. Frederickson and Johnson [2] consider selection in matrices with sorted columns. Their algorithm for selecting the k*th* largest element of $X + Y$, $1 \leqslant k \leqslant \frac{1}{2}n^2$, runs in $O(\max\{n, n\log(k/n)\})$ time. They also give an O(n) time algorithm for selection in matrices with sorted rows and columns [3].

Let A be an $n \times n$ matrix of real numbers with sorted rows and columns and let k be an integer, $1 \leqslant k \leqslant n^2$. We present an O(n) time algorithm to select the k*th* smallest element of A. The algorithm presented in this paper applies an elegant divide-and-conquer technique. This method may be applied to similar order related problems. For instance, we have used this technique to obtain a linear time algorithm for the optimum offset problem of channel routing in VLSI [9]. Although Frederickson and Johnson's algorithm [3] has a similar time bound, the algorithm presented in this paper is simpler. Also the technique used in our algorithm is of practical as well as theoretical interest.

* This work was supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Grants A5516 and A4304.

2. Terminology

Let A be an $n \times n$ matrix of reals. The elements of A are not necessarily distinct. We assume rows and columns of A to be indexed $1, 2, \ldots, n$. We call A *ordered* if elements in each row are in nonincreasing order, and elements in each column are in nondecreasing order. Let $\bar{n} = \lceil \frac{1}{2}(n+1) \rceil$. Submatrix $\bar{A}$ of A is an $\bar{n} \times \bar{n}$ matrix and is defined to be the submatrix of A consisting of the odd indexed rows and columns, plus the last row and column of A in case n is even. Let L be a list of reals and a be a real number. We define $rank^+$ and $rank^-$ of a in L as follows:

$$rank^+(L, a) = |\{x \in L \mid x > a\}|, \tag{2.1}$$

$$rank^-(L, a) = |\{x \in L \mid x < a\}|. \tag{2.2}$$

Suppose $1 \leqslant k \leqslant |L|$. Then a is defined to be the k*th* smallest element of L if and only if $rank^-(L, a) \leqslant k - 1$ and $rank^+(L, a) \leqslant |L| - k$. For simplicity we use the term k*th element of* L to mean k*th smallest element of* L throughout this paper.

3. The main observation

The following theorem is the basis for our selection algorithm.

Theorem 3.1. *Let* A *be an* $n \times n$ *ordered matrix and* $\bar{A}$ *be the submatrix of* A *as defined earlier. Then, for any real number* a, *the following inequalities hold*:

(i) $rank^-(A, a) \leqslant 4\, rank^-(\bar{A}, a)$,

(ii) $rank^+(A, a) \leqslant 4\, rank^+(\bar{A}, a)$.

Proof. We only prove (i). Part (ii) may be proved similarly. Let $\bar{A}_L$ consist of the elements of $\bar{A}$ that are less than a. Thus

$$|\bar{A}_L| = rank^-(\bar{A}, a). \tag{3.1}$$

Let A_L be the portion of A that consists of:

(a) $\bar{A}_L$, and

(b) for each element $A_{ij} \in \bar{A}_L$, its neighboring elements $A_{i,j-1}$, $A_{i+1,j-1}$ and $A_{i+1,j}$ (if they exist) from $A - \bar{A}$. Since the matrix A is ordered, A_L includes all the elements of A that are less than a. Thus

$$|A_L| \geqslant rank^-(A, a). \tag{3.2}$$

By the construction of A_L from $\bar{A}_L$ we conclude

$$|A_L| \leqslant 4|\bar{A}_L|. \tag{3.3}$$

From (3.1), (3.2) and (3.3) we have $rank^-(A, a) \leqslant 4\, rank^-(\bar{A}, a)$. □

4. The selection algorithm

Before describing the details of the selection algorithm we present an O(n) time algorithm to compute $rank^-$ of a real number a in an $n \times n$ ordered matrix A. $rank^+$ may be computed similarly.

The function in Fig. 1 computes $rank^-(A, a)$ in O(n) time. Let $pick(L, k)$ be a function which takes a list L and an integer k, $1 \leqslant k \leqslant |L|$, and returns the k*th* element of L in $O(|L|)$ time. For such an algorithm, see [1]. Functions *pick*, $rank^-$ and $rank^+$ are used in our selection algorithm.

The idea behind our selection algorithm is to recursively select two elements a and b, $a \geqslant b$, from $\bar{A}$ so that the following hold:

(1) The k*th* element of A is between a and b.

(2) The number of elements of A which are less than a and greater than b is O(n).

The main result of this paper is that the function *select*(A, k), presented in Fig. 2, computes the k*th* element of an $n \times n$ ordered matrix A in O(n) time. The function *select* calls the recursive function *biselect*(n, A, k_1, k_2) with $k_1 \geqslant k_2$, which returns (x, y), where x is the k_1th and y is the k_2th element of the $n \times n$ matrix A. Let $\bar{k}_1$ and $\bar{k}_2$ be defined as follows:

$$\bar{k}_1 = \begin{cases} n + 1 + \lceil \frac{1}{4}k_1 \rceil & \text{if n is even,} \\ \lceil \frac{1}{4}(k_1 + 2n + 1) \rceil & \text{if n is odd;} \end{cases} \tag{4.1}$$

$$\bar{k}_2 = \lfloor \tfrac{1}{4}(k_2 + 3) \rfloor. \tag{4.2}$$

$\bar{k}_1$ is chosen to be the smallest integer such that

```
function rank⁻(A, a);
  begin
    j := 1; x := 0;
    for i := 1 to n do begin
      while j ≤ n and A_ij ≥ a do j := j+1;
      x := x+n−j+1
    end;
    return x
  end;
```

Fig. 1. A ranking algorithm.

the $\bar{k}_1$th element of $\bar{A}$ is at least as large as the k_1th element of A. $\bar{k}_2$ is chosen to be the largest integer such that the $\bar{k}_2$th element of $\bar{A}$ is no larger than the k_2th element of A. In the algorithm, the phrase *ith of* A is shorthand for *ith element of* A. The first parameter n of the function *biselect* is the dimension of the submatrix which appears as the second parameter of the function (and not necessarily the dimension of the main matrix). We assume that either the matrix A is present in the memory before the computation begins, or the elements of A can be computed as they are needed. If A is of the form X + Y, then only the vectors X and Y need to be present in the memory.

```
function select(A, k);
  begin
    (x, y) := biselect(n, A, k, k);
    return x
  end select;

function biselect(n, A, k1, k2);
    begin
1.  if n ≤ 2
2.  then (x, y) := (k1th of A, k2th of A)
    else begin
3.    (a, b) := biselect(n̄, Ā, k̄1, k̄2);
4.    ra⁻ := rank⁻(A, a);
5.    rb⁺ := rank⁺(A, b);
6.    L := {A_ij | a > A_ij > b};
7.    if ra⁻ ≤ k1 − 1 then x := a
8.      else if k1 + rb⁺ − n² ≤ 0 then x := b
9.        else x := pick(L, k1 + rb⁺ − n²);
10.   if ra⁻ ≤ k2 − 1 then y := a
11.     else if k2 + rb⁺ − n² ≤ 0 then y := b
12.       else y := pick(L, k2 + rb⁺ − n²)
      end;
13.   return (x, y)
    end biselect;
```

Fig. 2.

5. Proof of correctness

In order to prove the correctness and the claimed running time of the algorithm, we need the following lemma.

Lemma 5.1. *During the course of the algorithm, whenever biselect*(n, A, k_1, k_2) *is called, the following hold:*

(i) $n^2 \geq k_1 \geq k_2 \geq 1$,

(ii) $k_1 - k_2 \leq 4n - 4$.

Proof. We use induction on the number of times *biselect* is called. The function *biselect* is first called from *select* with $k_1 = k_2 = k$ and $n^2 \geq k \geq 1$. Therefore, (i) and (ii) obviously hold in this case. Otherwise, *biselect* is called from line 3. of the algorithm (see Fig. 2). In this case, by the induction hypothesis, we have $n^2 \geq k_1 \geq k_2 \geq 1$ and $k_1 - k_2 \leq 4n - 4$. Furthermore, $n \geq 3$. We consider two cases, depending whether n is even or odd.

Case 1 (n is even): Recall that, in this case, $\bar{n}$, the dimension of $\bar{A}$, is $\frac{1}{2}(n+2)$. Using formulas (4.1) and (4.2) it is easy to show that $\bar{k}_1 \leq \bar{n}^2$, $\bar{k}_2 \geq 1$ and $\bar{k}_1 - \bar{k}_2 \geq 0$. Therefore, (i) holds. Furthermore, $\bar{k}_1 - \bar{k}_2 \leq 2n = 4\bar{n} - 4$.

Case 2 (n is odd): This case is proved similarly.

We conclude that $\bar{n}^2 \geq \bar{k}_1 \geq \bar{k}_2 \geq 1$ and $\bar{k}_1 - \bar{k}_2 \leq 4\bar{n} - 4$ in both cases. This completes the proof. □

Theorem 5.2. *Let* A *be an* $n \times n$ *ordered matrix and* $n^2 \geq k_1 \geq k_2 \geq 1$. *Then biselect*$(n, A, k_1, k_2)$ *returns the* k_1*th and* k_2*th elements of* A.

Proof. Let x^* and y^* be the k_1th and k_2th elements of A, respectively. Notice that $x^* \geq y^*$. We show, by induction on n, that *biselect*(n, A, k_1, k_2) returns (x^*, y^*).

Basis ($n \leq 2$): Obvious.

Induction ($n > 2$): By (i) of Lemma 5.1 we have $\bar{n}^2 \geq \bar{k}_1 \geq \bar{k}_2 \geq 1$, $\bar{A}$ is an $\bar{n} \times \bar{n}$ ordered matrix, and $\bar{n} < n$. Therefore, by the induction hypothesis, at line 3. of the algorithm, a is the $\bar{k}_1$th and b the $\bar{k}_2$th element of $\bar{A}$. This implies

$$rank^+(\bar{A}, a) \leq \bar{n}^2 - \bar{k}_1$$

and

$$rank^-(\bar{A}, b) \leq \bar{k}_2 - 1.$$

Using Theorem 3.1, we have

$rank^+(A, a) \leqslant 4\bar{n}^2 - 4\bar{k}$

and

$rank^-(A, b) \leqslant 4\bar{k}_2 - 4.$

Now, by substituting the appropriate formulas for $\bar{n}^2$, $\bar{k}_1$ and $\bar{k}_2$, we obtain

$$rank^+(A, a) \leqslant n^2 - k_1, \tag{5.1}$$

$$rank^-(A, b) \leqslant k_2 - 1. \tag{5.2}$$

From (5.1) and (5.2) we conclude $a \geqslant x^* \geqslant y^* \geqslant b$. If x is assigned the value a at line 7., then $rank^-(A, a) = ra^- \leqslant k_1 - 1$. This, together with inequality (5.1), implies $x^* = a = x$. If x is assigned the value b at line 8., then $rank^+(A, b) = rb^+ \leqslant n^2 - k_1$. This, together with inequality (5.2) and the fact that $k_2 \leqslant k_1$, implies $x^* = b = x$. Finally, if the assignment at line 9. is executed, then $ra^- > k_1 - 1$ and $rb^+ > n^2 - k_1$. These imply $a > x^* > b$. Therefore, $x^* \in L$, and by definition we have $|L| = ra^- + rb^+ - n^2$. From this we conclude that $|L| \geqslant k_1 + rb^+ - n^2 \geqslant 1$. Therefore the assignment of line 9. is meaningful and after its execution we have

$$\begin{aligned} rank^-(A, x) &= rank^-(L, x) + (n^2 - rb^+) \\ &\leqslant (k_1 + rb^+ - n^2 - 1) + (n^2 - rb^+) \\ &= k_1 - 1, \end{aligned}$$

$$\begin{aligned} rank^+(A, x) &= rank^+(L, x) + (n^2 - ra^-) \\ &\leqslant (|L| - (k_1 + rb^+ - n^2)) \\ &\quad + (n^2 + ra^-) \\ &= n^2 - k_1. \end{aligned}$$

Therefore, $x = x^*$. With a similar argument, we have $y = y^*$. This completes the proof. □

Theorem 5.3. *Function select* (A, k) *correctly computes the* k*th element*, $1 \leqslant k \leqslant n^2$, *of an* $n \times n$ *ordered matrix* A.

Proof. The proof immediately follows from Theorem 5.2. □

6. Timing analysis

The following theorem and its corollary establish the claimed running time of the selection algorithm.

Theorem 6.1. *If* A *is an* $n \times n$ *ordered matrix and* $n^2 \geqslant k \geqslant 1$, *then select*(A, k) *computes the* k*th element of* A *in* O(n) *time.*

Proof. We first show that |L|, at line 6. of the algorithm, satisfies $|L| = O(n)$. At line 3. we have $a \geqslant b$. If $a = b$, then L is empty. Otherwise, from the definition of L we have

$|L| = rank^-(A, a) + rank^+(A, b) - n^2.$

Using Theorems 3.1 and 5.2 and Lemma 5.1(ii), we can show that

$|L| \leqslant 12n.$

From the above we conclude that lines 9. and 12. of the algorithm take O(n) time. Lines 4., 5. and 6. also take O(n) time, using an algorithm that resembles that of Fig. 1. Therefore, lines 4. to 13. take O(n) time. If T(n) is the time complexity of *biselect*, then

$T(n) = O(1)$ for $n \leqslant 2$,

and

$T(n) = T(\lceil \frac{1}{2}(n+1) \rceil) + O(n)$ for $n > 2$.

Therefore, $T(n) = O(n)$. This implies *select* takes O(n) time. □

Corollary 6.2. *The* k*th element of an* $n \times n$ *ordered matrix* A *can be found in* $O(\min\{n, k, n^2 - k\})$ *time.*

Proof. Theorem 6.1 shows that selection in A can be done in O(n) time. If $k < n$, then consider the $k \times k$ submatrix B of A, consisting of the last k columns and first k rows of A. The k*th* element of A is also the k*th* element of B which can be found in O(k) time. A similar argument holds for the case $n^2 - k < n$. □

7. Conclusion

In this paper we have presented an efficient and practical algorithm for selection in ordered matrices. The algorithm is based on a new divide-and-conquer technique which may be used in other order related problems as well. For instance, we have used this technique to obtain a linear time algorithm for the optimum offset problem of channel routing in VLSI [9].

References

[1] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest and R.E. Tarjan, Time bounds for selection, J. CSS 7 (4) (1973) 448–461.

[2] G.N. Frederickson and D.B. Johnson, The complexity of selection and ranking in $X + Y$ and matrices with sorted columns, J. CSS 24 (1982) 197–208.

[3] G.N. Frederickson and D.B. Johnson, Generalized s tion and ranking: Sorted matrices, SIAM J. Compu (1) (1984) 14–30.

[4] M.L. Fredman, Two applications of probabilistic se technique: Sorting $X + Y$ and building balanced se trees, Proc. 7th ACM Symp. on Theory of Compu (1975) 240–244.

[5] L.H. Harper, T.H. Payne, J.E. Savage and E. Str Sorting $X + Y$, Comm. ACM 18 (6) (1975) 347–349.

[6] J.L. Hodges and E.L. Lehmann, Estimates of loca based on rank tests, Ann. Math. Statist. 34 (1963) 598–

[7] D.B. Johnson and S.D. Kashdan, Lower bounds for s tion in $X + Y$ and other multisets, J. ACM 25 (5) (1 556–570.

[8] D.B. Johnson and T. Mizoguchi, Selecting the k th eler in $X + Y$ and $X_1 + X_2 + \cdots + X_m$, SIAM J. Comput. (1978) 147–153.

[9] A. Mirzaian, Channel routing in VLSI, Proc. 16th A ACM Symp. on Theory of Computing (1984) 101–10

[10] M.I. Shamos, Geometry and statistics: Problems at interface, in: J.F. Traub, ed., Algorithms and Comple New Directions and Recent Results (Academic Press, York, 1976) 251–280.