

**Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки**

Лабораторна робота №5

з дисципліни
«Алгоритми і структури даних»

Виконав:

Студент групи IM-41
Димура Ілля Олександрович
Номер у списку групи: 7

Перевірила:

Молчанова А. А.

Київ 2024

Завдання

1. Написати програму розв'язання задачі пошуку (за варіантом) у двовимірному масиві (матриці) методом двійкового пошуку. Алгоритм двійкового пошуку задається варіантом завдання.
2. Розміри матриці m та n взяти самостійно у межах від 7 до 10.
3. При тестуванні програми необхідно підбирати такі вхідні набори початкових значень матриці, щоб можна було легко відстежити коректність виконання пошуку і ця коректність була б протестована для всіх можливих випадків. З метою тестування дозволяється використовувати матриці меншого розміру

Варіант 7:

Задано матрицю дійсних чисел $A[m,n]$. Окремо у останньому рядку і першому стовпчику матриці визначити присутність заданого дійсного числа X і його місцезнаходження (координати) методом двійкового пошуку (Алгоритм №1), якщо елементи цього рядка і стовпчика впорядковані за небільшеннем.

Текст програми

```
#include <stdio.h>

const int ROWS = 7;
const int COLUMNS = 7;

void last_row(const float matrix[ROWS][COLUMNS], const int m, const
int n, const float target) {

    // check order

    for (int i = 0; i < n - 1; i++) {
        if (matrix[m - 1][i] < matrix[m - 1][i + 1]) {
            printf("The last row is not sorted in descending order,
skipping... \n");
            return;
        }
    }
}
```

```
int idx = -1;
int left = 0;
int right = n - 1;

while (left <= right) {
    int mid = (left + right) / 2;
    if (matrix[m - 1][mid] == target) {
        idx = mid;
        break;
    }
    if (matrix[m - 1][mid] > target) left = mid + 1;
    else right = mid - 1;
}

if (idx == -1) printf("Value %.3f was not found in the last row\n",
target);
else printf("Value %.3f was found in the last row, index = %d\n",
target, idx);
}

void first_column(const float matrix[ROWS][COLUMNS], const int m,
const float target) {
    // check order
    for (int i = 0; i < m - 1; i++) {
        if (matrix[i][0] < matrix[i + 1][0]) {
            printf("The last row is not sorted in descending order,
skipping...\n");
            return;
        }
    }
}
```

```
}
```

```
int idx = -1;
```

```
int left = 0;
```

```
int right = m - 1;
```

```
while (left <= right) {
```

```
    int mid = (left + right) / 2;
```

```
    if (matrix[mid][0] == target) {
```

```
        idx = mid;
```

```
        break;
```

```
}
```

```
    if (matrix[mid][0] > target) left = mid + 1;
```

```
    else right = mid - 1;
```

```
}
```

```
if (idx == -1) printf("Value %.3f was not found in the first column\n", target);
```

```
else printf("Value %.3f was found in the first column, index = %d\n", target, idx);
```

```
}
```

```
int main(void) {
```

```
    int m = ROWS;
```

```
    int n = COLUMNS;
```

```
    float matrix[m][n];
```

```
    for (int i = 0; i < m; i++) {
```

```
        printf("%d row of matrix 7x7 (space separated): ", i + 1);
```

```
        for (int j = 0; j < n; j++) {
            scanf("%f", &matrix[i][j]);
        }
    }

printf("input matrix:\n");
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        printf("%.3f\t\t", matrix[i][j]);
    }
    printf("\n");
}

float target;
printf("enter target for the first column: ");
scanf("%f", &target);

first_column(matrix, m, target);

printf("enter target for the last row: ");
scanf("%f", &target);

last_row(matrix, m, n, target);

return 0;
}
```

Оточення:

```
→ lab05 gcc --version
```

```
Apple clang version 16.0.0 (clang-1600.0.26.4)
```

```
Target: arm64-apple-darwin24.1.0
```

```
Thread model: posix
```

```
→ lab05 gcc -o bin01 main.c
```

```
→ lab05 ls bin*
```

```
bin01
```

Тести програми:

```
→ lab05 gcc -o bin01 main.c && ./bin01
1 row of matrix 7x7 (space separated): 70 2 3 4 5 6 7
2 row of matrix 7x7 (space separated): 60 4 2 1 7 5 6
3 row of matrix 7x7 (space separated): 50 3 6 2 1 3 2
4 row of matrix 7x7 (space separated): 40 5 8 6 2 4 3
5 row of matrix 7x7 (space separated): 30 8 9 3 3 6 1
6 row of matrix 7x7 (space separated): 20 1 2 8 2 7 5
7 row of matrix 7x7 (space separated): 10 9 8 7 6 5 4
input matrix:
70.000      2.000      3.000      4.000      5.000      6.000      7.000
60.000      4.000      2.000      1.000      7.000      5.000      6.000
50.000      3.000      6.000      2.000      1.000      3.000      2.000
40.000      5.000      8.000      6.000      2.000      4.000      3.000
30.000      8.000      9.000      3.000      3.000      6.000      1.000
20.000      1.000      2.000      8.000      2.000      7.000      5.000
10.000      9.000      8.000      7.000      6.000      5.000      4.000
enter target for the first column: 40
Value 40.000 was found in the first column, index = 3
enter target for the last row: 6
Value 6.000 was found in the last row, index = 4
```

```
→ lab05 gcc -o bin01 main.c && ./bin01
1 row of matrix 7x7 (space separated): 70 2 3 4 5 6 7
2 row of matrix 7x7 (space separated): 60 4 2 1 7 5 6
3 row of matrix 7x7 (space separated): 50 3 6 2 1 3 2
4 row of matrix 7x7 (space separated): 40 5 8 6 2 4 3
5 row of matrix 7x7 (space separated): 30 8 9 3 3 6 1
6 row of matrix 7x7 (space separated): 20 1 2 8 2 7 5
7 row of matrix 7x7 (space separated): 10 9 8 7 6 5 4
input matrix:
70.000      2.000      3.000      4.000      5.000      6.000      7.000
60.000      4.000      2.000      1.000      7.000      5.000      6.000
50.000      3.000      6.000      2.000      1.000      3.000      2.000
40.000      5.000      8.000      6.000      2.000      4.000      3.000
30.000      8.000      9.000      3.000      3.000      6.000      1.000
20.000      1.000      2.000      8.000      2.000      7.000      5.000
10.000      9.000      8.000      7.000      6.000      5.000      4.000
enter target for the first column: 11
Value 11.000 was not found in the first column
enter target for the last row: 10
Value 10.000 was found in the last row, index = 0
```

```
value 10.000 was found in the last row, index = 0
→ lab05 gcc -o bin01 main.c && ./bin01
1 row of matrix 7x7 (space separated): 70 2 3 4 5 6 7
2 row of matrix 7x7 (space separated): 60 4 2 1 7 5 6
3 row of matrix 7x7 (space separated): 50 3 6 2 1 3 2
4 row of matrix 7x7 (space separated): 40 5 8 6 2 4 3
5 row of matrix 7x7 (space separated): 30 8 9 3 3 6 1
6 row of matrix 7x7 (space separated): 20 1 2 8 2 7 5
7 row of matrix 7x7 (space separated): 10 9 8 7 6 5 4
input matrix:
70.000      2.000      3.000      4.000      5.000      6.000      7.000
60.000      4.000      2.000      1.000      7.000      5.000      6.000
50.000      3.000      6.000      2.000      1.000      3.000      2.000
40.000      5.000      8.000      6.000      2.000      4.000      3.000
30.000      8.000      9.000      3.000      3.000      6.000      1.000
20.000      1.000      2.000      8.000      2.000      7.000      5.000
10.000      9.000      8.000      7.000      6.000      5.000      4.000
enter target for the first column: 20
Value 20.000 was found in the first column, index = 5
enter target for the last row: 6
Value 6.000 was found in the last row, index = 4
```

```
→ lab05 gcc -o bin01 main.c && ./bin01
1 row of matrix 7x7 (space separated): 70 2 3 4 5 6 7
2 row of matrix 7x7 (space separated): 60 4 2 1 7 5 6
3 row of matrix 7x7 (space separated): 50 3 6 2 1 3 2
4 row of matrix 7x7 (space separated): 40 5 8 6 2 4 3
5 row of matrix 7x7 (space separated): 30 8 9 3 3 6 1
6 row of matrix 7x7 (space separated): 20 1 2 8 2 7 5
7 row of matrix 7x7 (space separated): 10 9 8 7 6 5 4
input matrix:
70.000      2.000      3.000      4.000      5.000      6.000      7.000
60.000      4.000      2.000      1.000      7.000      5.000      6.000
50.000      3.000      6.000      2.000      1.000      3.000      2.000
40.000      5.000      8.000      6.000      2.000      4.000      3.000
30.000      8.000      9.000      3.000      3.000      6.000      1.000
20.000      1.000      2.000      8.000      2.000      7.000      5.000
10.000      9.000      8.000      7.000      6.000      5.000      4.000
enter target for the first column: 10
Value 10.000 was found in the first column, index = 6
enter target for the last row: 9
Value 9.000 was found in the last row, index = 1
```

```
→ lab05 gcc -o bin01 main.c && ./bin01
1 row of matrix 7x7 (space separated): 70 2 3 4 5 6 7
2 row of matrix 7x7 (space separated): 60 4 2 1 7 5 6
3 row of matrix 7x7 (space separated): 50 3 6 2 1 3 2
4 row of matrix 7x7 (space separated): 40 5 8 6 2 4 3
5 row of matrix 7x7 (space separated): 30 8 9 3 3 6 1
6 row of matrix 7x7 (space separated): 20 1 2 8 2 7 5
7 row of matrix 7x7 (space separated): 10 8 8 8 8 8 8
input matrix:
70.000      2.000      3.000      4.000      5.000      6.000      7.000
60.000      4.000      2.000      1.000      7.000      5.000      6.000
50.000      3.000      6.000      2.000      1.000      3.000      2.000
40.000      5.000      8.000      6.000      2.000      4.000      3.000
30.000      8.000      9.000      3.000      3.000      6.000      1.000
20.000      1.000      2.000      8.000      2.000      7.000      5.000
10.000      8.000      8.000      8.000      8.000      8.000      8.000
enter target for the first column: 55
Value 55.000 was not found in the first column
enter target for the last row: 8
Value 8.000 was found in the last row, index = 3
```

```

→ Lab05 gcc -o bin01 main.c && ./bin01
1 row of matrix 7x7 (space separated): 73.19 9.91 3.63 3.11 6.33 8.19 3.9
2 row of matrix 7x7 (space separated): 61.47 3.74 5.67 4.44 6.15 2.47 1.3
3 row of matrix 7x7 (space separated): 51.25 8.52 6.51 7.22 5.16 9.25 1.8
4 row of matrix 7x7 (space separated): 40.31 8.13 2.44 6.33 4.02 5.31 0.8
5 row of matrix 7x7 (space separated): 30.92 2.29 7.56 1.99 5.07 4.92 0.2
6 row of matrix 7x7 (space separated): 20.12 2.21 2.72 8.11 7.02 1.12 0.2
7 row of matrix 7x7 (space separated): 10.57 6.75 5.35 4.55 4.01 3.57 0.6
input matrix:
73.190      9.910      3.630      3.110      6.330      8.190      3.900
61.470      3.740      5.670      4.440      6.150      2.470      1.300
51.250      8.520      6.510      7.220      5.160      9.250      1.800
40.310      8.130      2.440      6.330      4.020      5.310      0.800
30.920      2.290      7.560      1.990      5.070      4.920      0.200
20.120      2.210      2.720      8.110      7.020      1.120      0.200
10.570      6.750      5.350      4.550      4.010      3.570      0.600
→ enter target for the first column: 51.25
Value 51.250 was found in the first column, index = 2
→ enter target for the last row: 0.6
→ Value 0.600 was found in the last row, index = 6

```

Висновок:

Для вирішення задачі був адаптований алгоритм двійкового пошуку для обробки як рядків, так і стовпців матриці, що дозволило ефективно зменшити кількість операцій. За умовою, елементи останнього рядка і першого стовпця впорядковані за спаданням. Відповідно до цього було змінено напрям пошуку: якщо $X > A[i]$, то переходимо в ліву частину. Аналогічно, якщо $X < A[i]$, то переходимо в праву.

Логарифмічна складність, а також можливість змінювати напрямок і умови пошуку зробили цей алгоритм поширеним у сфері обробки даних. Наприклад, одним із основних індексів у базах даних є індекси, побудовані на принципі бінарного пошуку (B-дерево, B+ дерево тощо). Вони також мають складність (акумульовану) $T = O(\log_2 n)$ для операцій вставки, читання та видалення. Основною причиною цього є те, що дуже часто ми працюємо з впорядкованими даними: час та ідентифікатори у базах даних (тип SERIAL) монотонно збільшуються. Тому бінарний пошук є дуже ефективним, коли ми шукаємо записи з точним співпадінням ($=$) або за діапазонами ($>$, $<$, \geq , \leq , BETWEEN).

Проте, якщо пошук виконується виключно за точним співпадінням, є сенс порівняти швидкість виконання з алгоритмами, побудованими на основі хеш-таблиць (hash map).