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 ІМ-42
Туров Андрій Володимирович
номер у списку групи: 29

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 29

Задано квадратну матрицю дійсних чисел $A[n;\ n]$. Визначити присутність у побічній діагоналі матриці будь-якого з чисел діапазону $[0;\ 5]$ і його місцезнаходження (координати) методом двійкового пошуку (Алгоритм №1), якщо елементи цієї діагоналі впорядковані за незменшенням.

Текст програми

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

// Amount of indices in the antidiagonal to repeat, as a coefficient
const float REPEATING_CHANCE = 0.5;

// The first antidiagonal value determines the others.
// It is MIN_FIRST + random_float(MIN_RANDOM, MAX_RANDOM)
// All the following are between MIN and MAX, but guaranteed to be less or equal.

// These are good for testing no suitable elements.
const float MIN_FIRST  = 5;
const float MIN_RANDOM = -9.99;
const float MAX_RANDOM = 10;

/*// The first suitable element is probably in the middle*/
/*const float MIN_FIRST  = 5;*/
/*const float MIN_RANDOM = 0;*/
/*const float MAX_RANDOM = 20;*/

/*// Good chance that none will be suitable, but some towards the end can be*/
/*const float MIN_FIRST  = 49;*/
/*const float MIN_RANDOM = -5;*/
/*const float MAX_RANDOM = 50;*/

/*// First 1-3 are suitable*/
/*const float MIN_FIRST  = 10;*/
/*const float MIN_RANDOM = -10;*/
/*const float MAX_RANDOM = -5;*/

float random_float(float a, float b) {
```

```
	// If the first part is 0, we get a;
	// If 1, we get b - a + a = b;
	return ((float) rand() / (float) RAND_MAX)
		* (b - a) + a;
}

char contains(int array[], int size, int needle) {
	char result = 0;
	for (int i = 0; i < size; i++) {
		if (array[i] == needle) {
			result = 1;
			break;
		}
	}

	return result;
}

// Searches for an element that satisfies `(x >= 0) && (x <= 5)`
int asc_binary_search(int size, float array[size][size]) {
	int left_x_bound = 0;
	int right_x_bound = size - 1;
	float current_element;
	int final_x = -1;

	while (left_x_bound <= right_x_bound) {
		int x = (left_x_bound + right_x_bound) / 2;
		current_element = array[size - x - 1][x];
		printf("Checking %5.2f (%d %d): ", current_element, size - x - 1, x);

		if ((current_element >= 0) && (current_element <= 5)) {
			final_x = x;
			printf("found\n");
			break;
		} else if (current_element > 5) {
			right_x_bound = x - 1;
			printf("more than needed\n");
		} else {
			left_x_bound = x + 1;
			printf("less than needed\n");
		}
	}

	return final_x;
}

void generate_data(int size, float matrix[size][size]) {
	// Amount of indices in the antidiagonal to repeat
	const int REPEATING_SIZE = REPEATING_CHANCE * size;

	// Randomly generating indices that will have repeated values
	// (not guaranteed to be unique, but I'm fine with that)
	int repeating_indices[REPEATING_SIZE];
	for (int i = 0; i < REPEATING_SIZE; i++) {
		repeating_indices[i] = 1 + random_float(0, size - 2); // we only need
```

```
the whole part
	}

	float antidiagonal[size];
	// Randomly generating the antidiagonal.
	// The largest number goes first (top-to-bottom), and some of them have to
repeat
	antidiagonal[0] = MIN_FIRST + random_float(MIN_RANDOM, MAX_RANDOM);
	for (int i = 1; i < size; i++) {
		if (contains(repeating_indices, REPEATING_SIZE, i)) {
			antidiagonal[i] = antidiagonal[i - 1];
		} else {
			do {
			antidiagonal[i] = random_float(MIN_RANDOM, MAX_RANDOM);
			} while (antidiagonal[i] >= antidiagonal[i - 1]);
		}
	}

	// Filling the matrix with antidiagonal and arbitrary random values
	for (int y = 0; y < size; y++) {
		for (int x = 0; x < size; x++) {
			if (x == size - y - 1) {
			matrix[y][x] = antidiagonal[y];
			if ((antidiagonal[y] >= 0) && (antidiagonal[y] <= 5)) {
					printf("\033[34m"); // print in blue
			} else {
					printf("\033[31m"); // print in red
			}
			} else {
			matrix[y][x] = ((float) rand() / (float) RAND_MAX) * 99.0;
			}
			printf("%5.2f\033[0m ", matrix[y][x]);
		}
		printf("\n");
	}
}

int main(int argc, char **argv) {
	srand(clock());

	int size;
	if (argc == 1) {
		printf("Enter the matrix size (n*n): ");
		scanf("%d", &size);
	} else {
		size = atoi(argv[1]);
	}

	for (;;) {
		system("clear");

		float matrix[size][size];

		generate_data(size, matrix);
```

```
            int x = asc_binary_search(size, matrix);

            if (x == -1) {
                  printf("No element could be found\n");
            } else {
                  printf("The first found element is (%d %d)\n", size - x - 1, x);
            }

            printf("\x1b[2m\n\nPress Enter to regenerate...\n\x1b[0m");
            getchar();
      }


      return 0;
}
```

Результати тестування програми

(В моєму варіанті елементи побічної діагоналі в порядку незменшення, але я під час написання програми спирався на верхній елемент як на найбільший, тому він в мене скрізь в коді “перший”, адже далі зростає Y-координата. Результат не змінюється.)

Верхній елемент від -4.99 до 15.00, усі інші менші, але в межах $[-9.99,\ 10.00]$ (перший закоментований набір констант)

```
32.18 55.98 96.17 92.22 97.49 59.92 63.12 46.41 15.22 11.67
97.74 28.06 48.65 92.35 77.35 48.76 25.76 41.13 11.67 32.25
15.12 78.61 17.00 72.63 76.76  4.11 82.72  7.85 30.50 29.59
 3.38 69.37 45.91  4.40  2.55  2.89  7.85  1.57 94.77  1.39
61.49 58.89 47.79 76.71 57.63  7.85 75.85 26.36 50.98 54.20
75.12 76.75 95.33  8.36 -2.42 91.87 74.94 25.37 65.50 52.70
29.48 49.22 83.20 -2.42 59.07 52.60 53.57  5.98 57.00 56.13
 8.88 58.58 -7.95 51.90 10.26 21.07 11.79 58.06 97.78 69.42
34.91 -7.95 25.13 21.40 89.11  1.25 98.15 85.44  9.61 91.01
-9.78 61.38 34.98 57.51 15.07 64.46  7.73 98.28 24.53 60.33
Checking -2.42 (5 4): less than needed
Checking  7.85 (2 7): more than needed
Checking  7.85 (4 5): more than needed
No element could be found
```

```
70.90  9.81 66.38 20.53 42.33 97.63 98.39 94.37 69.69  0.45
 9.03 62.95 76.20 69.80 12.52 29.47 89.95 69.06  0.45 84.81
68.27 63.41  3.96 33.18 41.18 41.27 29.70  0.45  1.09  1.57
31.61 23.34 69.94 32.18 94.25 79.76 -4.16 98.56 15.78 23.08
97.19 15.17 18.45 67.88 24.21 -4.25 81.40 45.09 94.00 93.92
74.55 84.96 63.98 60.36 -9.42 54.23 28.39 64.32 87.41 69.58
 6.59 18.11 70.67 -9.42  8.16 49.72 94.01 78.11 81.90 89.25
58.86 81.47 -9.60  6.03 81.95 79.66 21.21  1.40 48.54 45.41
82.80 -9.60 93.63 40.42 77.72 69.19 26.37 42.71 30.55 80.60
-9.88 71.10 94.87 69.01 41.68  2.46 87.12 13.34 10.62 37.84
Checking -9.42 (5 4): less than needed
Checking  0.45 (2 7): found
The first found element is (2 7)
```

Верхній елемент від 5 до 25, усі інші менші, але в межах [0; 20] (другий набір констант)

```
32.08 82.10  6.44 67.51 67.86 52.16 19.70 79.15  8.92 18.88
 5.72 79.79 73.87 40.82 53.70 65.02 65.93  4.30  0.39 98.85
71.71 78.55 36.60 83.34 76.64 76.91 92.87  0.37 62.48 33.21
75.11 69.57 71.95 75.11  2.65 55.04  0.37 81.55 70.16 23.90
34.71 89.86  4.05 43.63 95.58  0.21 83.85 18.50 37.40 38.55
83.52  4.34 42.86 83.37  0.21 76.04 22.41 20.97 60.38  0.05
97.88 54.25 62.53  0.09 32.09 30.36 33.09  5.03  6.47 35.74
60.08 88.03  0.02  6.90 83.98 23.74 96.76 88.03 67.37 93.34
72.88  0.02 85.86 31.75 12.43 70.38 36.08 55.29 54.75 13.13
 0.00 77.70 75.72 73.51 77.75 74.60 28.76 41.28  7.69 59.12
Checking  0.21 (5 4): found
The first found element is (5 4)
```

```
50.15 42.69 20.28 91.77 83.06 76.41 54.24 57.52 25.88 24.15
30.23 94.26 68.07 29.93 58.87 20.43 17.29 97.85 14.78 76.04
50.13 88.64 71.65  4.03 97.25 56.47 18.40 14.08 79.96 58.49
83.85 89.56 78.07 84.05 40.71 21.76 14.08  5.34 33.48  5.81
81.75 87.73 63.33  8.63 18.95 14.08 58.59 76.70 48.88 18.46
97.13 66.17 17.31 74.17 14.08 17.30  6.95 46.81 21.34  5.21
 4.28 39.73 85.17  0.19 62.77 24.58 75.73 41.83  9.64 17.44
63.59 14.98  0.05 50.92 69.40 96.73 39.65 33.74  6.36 58.60
92.33  0.05 83.06  8.49 11.79 81.18 74.66 29.10 56.35 91.96
 0.04 36.06  4.16 14.30 41.26  8.44 54.03 27.43 71.21 78.62
Checking 14.08 (5 4): more than needed
Checking  0.05 (8 1): found
The first found element is (8 1)
```

Верхній елемент від 44 до 99, усі інші менші, але в межах $[-5; 50]$ (третій закоментований набір констант)

```
29.73 12.00 65.04 80.68 10.71 74.11 13.19 57.90 36.88 89.92
46.71 16.89 64.75 13.38 76.77  5.09 35.61 66.58 -1.40 25.33
29.47 28.08 51.94 10.97 68.39 14.22 48.14 -1.40 77.09 30.67
76.58 17.29 80.90 76.72 47.02 92.90 -1.40 42.76 28.69  4.60
17.87 41.88 62.50 54.75 88.59 -4.82 79.39 20.50  2.98 57.16
25.59 38.59 24.74 50.93 -4.87 68.06 52.81  3.86 79.03 22.20
18.08 28.17  0.29 -4.87 48.75  5.75 17.58 30.65 82.48 64.59
24.55 26.23 -4.87 93.29 29.15 44.10 36.17 91.65 98.85 25.76
72.04 -4.87 20.35 28.73 30.20 45.95 67.32 54.94 96.87 36.38
-4.92  8.75  1.74 16.41 30.95 19.82 44.58 31.24 68.57 50.34
Checking -4.87 (5 4): less than needed
Checking -1.40 (2 7): less than needed
Checking -1.40 (1 8): less than needed
Checking 89.92 (0 9): more than needed
No element could be found
```

```
71.65 15.38 81.81 93.97 72.32 63.44  0.30 71.36 35.26 57.62
45.79 74.26 28.16 33.07  3.32 98.02 25.88 36.94 57.62 85.87
10.80 58.22 89.48 66.86 82.73 53.59 38.60 57.62 40.00 14.89
93.44 14.75 92.99  4.00 86.40  9.37 30.06 85.80 81.37 81.69
50.25 81.67 54.05 85.50 28.46 26.26 29.31 14.66 61.53 32.64
13.68 87.41 69.58  0.55 26.26 98.21 28.80 90.03 66.07 12.53
44.62  5.67 52.53 25.47 59.51  0.11 67.28 53.51  4.10 54.68
62.88 89.91 25.47 37.05 45.57 41.15 19.72  0.62 27.65 48.18
29.94 25.47 42.31 10.71 62.57 55.99 98.12 33.15 56.54 97.33
 0.31 61.95 47.57 64.40 74.48 92.19 70.07 28.01 52.70 70.18
Checking 26.26 (5 4): more than needed
Checking 25.47 (8 1): more than needed
Checking  0.31 (9 0): found
The first found element is (9 0)
```

Верхній елемент від 0 до 5, усі інші менші, але в межах $[-10; -5]$

```
21.09 10.15 47.43 44.59 61.70 53.15 38.52 82.11 69.07  0.00
72.85 44.86 44.54 47.95 17.53 33.01  2.14 81.23 -9.63  3.93
17.27 67.04 56.40  1.72 31.11 10.95 89.08 -9.79  3.33 91.37
23.50 86.85 52.89 24.10  8.94 63.04 -9.79 71.53 53.53 25.74
25.68 92.04  8.85 94.75 65.89 -9.79 53.71 40.30 14.84 71.23
73.31 16.98 53.46 77.24 -9.85 34.25 21.50 34.64 35.97 52.62
45.58 26.04 55.95 -9.85 37.95 49.55 43.80 90.83 73.65 52.74
54.87 46.18 -9.85  7.26 80.61 71.86  0.31 89.46 67.61 66.20
44.17 -9.95  8.91 81.04 16.40 82.21 98.02 69.86 60.45 33.27
-9.97 91.37 95.09 69.24 44.98 41.67 95.28  1.93 79.62 45.83
Checking -9.85 (5 4): less than needed
Checking -9.79 (2 7): less than needed
Checking -9.63 (1 8): less than needed
Checking  0.00 (0 9): found
The first found element is (0 9)
```

Висновки

Розв’язав задачу на бінарний пошук у матриці (вона там була декорацією, в принципі, бо я рандомно усе одно окремо генерував діагональ, слідкуючи за тим, щоб наступні числа були меншими за попереднє, а потім підставляв і виконував пошук відносно другої координати, знаходячи першу як $size - x - 1$.

Якби дані зчитувались з файлу або напряму запитувались у користувача, мені б було легше демонструвати правильну роботу програми і я б витратив набагато менше часу, але натомість я показав:

- Що вмію генерувати набір випадкових даних, що відповідає певним критеріям.
- Як структурував програму, винісши логіку в окремі функції, а параметри для випадкової генерації у константи, щоб можна було впливати на хід програми.
- Що 20 разів перезапустити програму на випадкових даних менш запарно, ніж вручну придумувати та вписувати кожне число, тому Ви тепер читаєте звіт, де 2.5 аркуша займає С-код.
- Що я хотів написати складнішу і цікавішу програму просто. ¯_(ツ)_/¯
- Що я не додумався винести генерацію в окремий скрипт, тому такий підхід буде лише у шостій лабі.

upd 04:47 30.11: тепер іще більше коду, і ще набір констант