

|  |  |
| --- | --- |
| **Министерство образования и науки**  **Российской Федерации**  **Государственное образовательное учреждение высшего образования «МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ**  **ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  **(МАДИ)»** |  |

**Кафедра «Высшая математика»**

**Отчет по дисциплине «Структуры и алгоритмы обработки данных»**

**Практическая работа №4**

**«Методы поиска и модификация двумерных массивов»**

**Выполнил:**

Учебная группа 1бПМ\_\_\_\_\_\_\_\_\_  
ФИО Греча К. П. \_\_

**Принял:**

Должность Старший преподаватель

Звание \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ФИО \_ Кутейников И. А.\_\_\_\_\_

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023г.

**Цель:**

Требуется определить двумерный массив целых чисел (например,

размера 30), заполнить его случайными числами (в диапазоне от A до B) или

ввести его элементы с клавиатуры и определить его характеристики в

соответствии с вариантом (В частично заполненном значением 1 логическом квадратном массиве имеются свободные участки, моделируемые значением 0 Определить, можно ли разместить на них квадрат из значений 1 размером M. Если возможно, то напечатать массив со вставленным квадратом на экране).

1 Инициализация элементов двумерного массива числами, вводимыми

с клавиатуры;

2 Инициализация элементов двумерного массива случайными числами;

3 Вывод двумерного массива на экран;

4 Сортировка двумерного массива по строкам;

4.1 Сортировка в соответствии с вариантом;

4.2 Собственная реализация быстрой сортировки;

4.3 Сортировка встроенной функцией сортировки;

4.4 Сравнение времени работы сортировок;

5 Поиск значения в строке массива;

4.1 Последовательный поиск;

4.2 Бинарный поиск;

4.3 Фибоначчиев поиск;

4.4 Интерполяционный поиск;

4.5 Поиск встроенной функцией

4.4 Сравнение времени алгоритмов поиска;

6 Решение задачи уровня В;

7 Решение задачи уровня С (опционально);

8 Выход из программы.

Реализовать заданные методы поиска значений в числовом двумерном

массиве в соответствии с индивидуальным заданием. Оценить время работы

каждого алгоритма поиска.

**Ключевые методы кода:**

public static void fillByHand() {  
 System.*out*.println("Введите кол-во строк: ");  
 int str = *scanner*.nextInt();  
 System.*out*.println("Введите кол-во столбцов: ");  
 int stb = *scanner*.nextInt();  
 *array* = new int[str][stb];  
 System.*out*.println("Введите элементы массива: ");  
 for (int i = 0; i < str; i++) {  
 System.*out*.println("Строка № " + (i + 1) + " :");  
 for (int j = 0; j < stb; j++) {  
 System.*out*.println("Элемент № " + (j + 1) + ": ");  
 *array*[i][j] = *scanner*.nextInt();  
 }  
 }  
 }  
  
  
 public static void fillByRandom() {  
 System.*out*.println("Введите кол-во строк: ");  
 int str = *scanner*.nextInt();  
 System.*out*.println("Введите кол-во столбцов: ");  
 int stb = *scanner*.nextInt();  
 *array* = new int[str][stb];  
 for (int i = 0; i < str; i++) {  
 for (int j = 0; j < stb; j++) {  
 *array*[i][j] = *random*.nextInt(2001) - 1000; // заполняет случайными значениями от -1000 до 1000  
 }  
 }  
 }  
  
  
 public static void printArray() {  
 System.*out*.println("Двумерный массив: ");  
 if (*array*.length == 0) {  
 System.*out*.println("\n Массив не существует");  
 } else {  
 for (int i = 0; i < *array*.length; i++) {  
 for (int j = 0; j < *array*[i].length; j++) {  
 System.*out*.print(*array*[i][j] + " ");  
 }  
 System.*out*.println();  
 }  
 }  
 }  
  
  
 static int[][] shellSort() {  
 int rows = *array*.length;  
// Проходим по каждой строке  
 for (int i = 0; i < rows; i++) {  
 int[] row = *array*[i];  
 int n = row.length;  
 int d = n / 2;  
//Сортируем текущую строку  
 while (d > 0) {  
 for (int j = d; j < n; j++) {  
 int temp = row[j];  
 int k = j;  
  
 while (k >= d && row[k - d] > temp) {  
 row[k] = row[k - d];  
 k -= d;  
 }  
  
 row[k] = temp;  
 }  
 // уменьшаем шаг для следующей итерации  
 d /= 2;  
 }  
 }  
 return *array*;  
 }  
  
 public static void quickSortMatrix() {  
 int rows = *array*.length;  
  
 for (int i = 0; i < rows; i++) {  
 *array*[i] = *quickSortRow*(*array*[i]);  
 }  
 }  
  
 public static int[] quickSortRow(int[] row) {  
 if (row.length <= 1) {  
 return row;  
 }  
  
 int pivot = row[row.length - 1];  
 int i = -1;  
  
 for (int j = 0; j < row.length - 1; j++) {  
 if (row[j] < pivot) {  
 i++;  
  
 int temp = row[i];  
 row[i] = row[j];  
 row[j] = temp;  
 }  
 }  
  
 int temp = row[i + 1];  
 row[i + 1] = row[row.length - 1];  
 row[row.length - 1] = temp;  
  
 int[] leftPart = Arrays.*copyOfRange*(row, 0, i + 1);  
 int[] rightPart = Arrays.*copyOfRange*(row, i + 2, row.length);  
  
 int[] result = new int[row.length];  
 for (int j = 0; j < leftPart.length; j++) {  
 result[j] = leftPart[j];  
 }  
  
 result[leftPart.length] = pivot;  
  
 for (int j = 0; j < rightPart.length; j++) {  
 result[leftPart.length + 1 + j] = rightPart[j];  
 }  
  
 return result;  
 }  
  
  
 public static void builtInSort() {  
 int rows = *array*.length;  
  
 for (int i = 0; i < rows; i++) {  
 Arrays.*sort*(*array*[i]);  
 }  
 }  
  
  
 public static void compareSortingTime() {  
 long shellSortTime = *measureSortingTime*(Main::*shellSort*);  
 long quickSortTime = *measureSortingTime*(Main::*quickSortMatrix*);  
 long builtInSortTime = *measureSortingTime*(Main::*builtInSort*);  
  
 System.*out*.println("Сортировка Шелла заняла " + shellSortTime + " наносекунд.");  
 System.*out*.println("Быстрая сортировка заняла " + quickSortTime + " наносекунд.");  
 System.*out*.println("Встроенная сортировка заняла " + builtInSortTime + " наносекунд.");  
 }  
  
 private static long measureSortingTime(Runnable sortingMethod) {  
 long startTime = System.*nanoTime*();  
 sortingMethod.run();  
 long endTime = System.*nanoTime*();  
 return endTime - startTime;  
 }  
  
  
//CASE 5  
public static void sequentialSearchInSTR() {  
 int key = *getSearchValueInSTR*(); // Теперь используем новый метод для ввода значения  
 boolean found = false;  
  
 for (int i = 0; i < *array*.length; i++) {  
 for (int j = 0; j < *array*[i].length; j++) {  
 if (*array*[i][j] == key) {  
 System.*out*.println("Значение " + key + " найдено в строке " + (i + 1) + ", столбце " + (j + 1));  
 found = true;  
 break;  
 }  
 }  
 if (found) {  
 break;  
 }  
 }  
  
 if (!found) {  
 System.*out*.println("Значение " + key + " не найдено в матрице");  
 }  
}  
 // Метод для ввода значения для поиска в строке матрицы  
 public static int getSearchValueInSTR() {  
 System.*out*.print("Введите значение для поиска в строке: ");  
 return *scanner*.nextInt();  
 }  
  
  
  
 public static boolean binarySearchInSTR(int[] sortedSTR, int targetValue) {  
 int left = 0;  
 int right = sortedSTR.length - 1;  
 boolean valueFound = false;  
  
 while (left <= right) {  
 int mid = left + (right - left) / 2;  
  
 if (sortedSTR[mid] == targetValue) {  
 valueFound = true;  
 break;  
 } else if (sortedSTR[mid] < targetValue) {  
 left = mid + 1;  
 } else {  
 right = mid - 1;  
 }  
 }  
  
 if (valueFound) {  
 System.*out*.println("Значение " + targetValue + " найдено в строке");  
 } else {  
 System.*out*.println("Значение " + targetValue + " не найдено в строке");  
 }  
return valueFound;  
 }  
  
  
  
  
 public static void fibonacciSearchInMatrix(int keyValue) {  
 int rows = *array*.length;  
 int foundRow = -1;  
 int foundColumn = -1;  
 boolean valueFound = false;  
  
 for (int i = 0; i < rows; i++) {  
 int[] currentRow = *array*[i];  
 int n = currentRow.length;  
  
 if (n > 0 && *fibonacciSearch*(currentRow, keyValue, n)) {  
 foundRow = i;  
 foundColumn = *findColumnInRow*(currentRow, keyValue);  
 valueFound = true;  
 break;  
 }  
 }  
  
 if (valueFound) {  
 System.*out*.println("Значение найдено в строке " + (foundRow + 1) + " столбце " + (foundColumn + 1));  
 } else {  
 System.*out*.println("Значение не найдено");  
 }  
 }  
  
 public static boolean fibonacciSearch(int[] array, int key, int n) {  
 int fib2 = 0;  
 int fib1 = 1;  
 int fib = fib2 + fib1;  
  
 while (fib < n) {  
 fib2 = fib1;  
 fib1 = fib;  
 fib = fib2 + fib1;  
 }  
  
 int offset = -1;  
  
 while (fib > 1) {  
 int i = Math.*min*(offset + fib2, n - 1);  
  
 if (array[i] < key) {  
 fib = fib1;  
 fib1 = fib2;  
 fib2 = fib - fib1;  
 offset = i;  
 } else if (array[i] > key) {  
 fib = fib2;  
 fib1 = fib1 - fib2;  
 fib2 = fib - fib1;  
 } else {  
 return true;  
 }  
 }  
  
 if (fib1 == 1 && array[offset + 1] == key) {  
 return true;  
 }  
  
 return false;  
 }  
 // Новый метод для поиска столбца в строке  
 public static int findColumnInRow(int[] row, int key) {  
 for (int j = 0; j < row.length; j++) {  
 if (row[j] == key) {  
 return j;  
 }  
 }  
 return -1; // Если не найдено, возвращаем -1  
 }  
  
  
  
 public static void interpolationSearchInMatrix() {  
 System.*out*.print("Введите значение для поиска: ");  
 int keyValue = *scanner*.nextInt();  
 *scanner*.nextLine();  
  
 int rows = *array*.length;  
 boolean found = false;  
  
 for (int i = 0; i < rows; i++) {  
 int[] sortedRow = *quickSortRow*(Arrays.*copyOf*(*array*[i], *array*[i].length));  
 int result = *interpolationSearch*(sortedRow, keyValue); // для каждой строки используется метод interpolationSearch  
  
 if (result != -1) {  
 System.*out*.println("Значение " + keyValue + " найдено в строке " + (i + 1) + ", столбце " + (result + 1));  
 found = true;  
 }  
 }  
  
 if (!found) {  
 System.*out*.println("Значение " + keyValue + " не найдено в матрице");  
 }  
 }  
 // Метод интерполяционного поиска для одномерного массива  
 public static int interpolationSearch(int[] array, int key) {  
 int low = 0;  
 int high = array.length - 1; // лоу и хайт границы поиска в массиве  
  
 while (low <= high && key >= array[low] && key <= array[high]) {  
 if (low == high) {  
 if (array[low] == key) {  
 return low; // если элемент найден он возвращается , если нет то границы обновляются  
 }  
 return -1;  
 }  
  
 int pos = low + ((key - array[low]) \* (high - low)) / (array[high] - array[low]);  
  
 if (array[pos] == key) {  
 return pos;  
 }  
  
 if (array[pos] < key) {  
 low = pos + 1;  
 } else {  
 high = pos - 1;  
 }  
 }  
 return -1;  
 }  
  
  
  
 public static void builtInSearchInMatrix() {  
 System.*out*.print("Введите значение для поиска: ");  
 int keyValue = *scanner*.nextInt();  
 *scanner*.nextLine();  
  
 boolean found = false;  
  
 for (int i = 0; i < *array*.length; i++) {  
 for (int j = 0; j < *array*[i].length; j++) { // цикл фор проходит по каждому столбцу и строке  
 if (*array*[i][j] == keyValue) { // в каждой иттерации проверяется найдено значение или нет  
 System.*out*.println("Значение " + keyValue + " найдено в строке " + (i + 1) + ", столбце " + (j + 1));  
 found = true;  
  
 }  
 }  
 if (found) {  
 }  
 }  
 if (!found) {  
 System.*out*.println("Значение " + keyValue + " не найдено в матрице");  
 }  
 }  
  
  
  
 public static void compareSearchMethodsTime() {  
 int targetValue = *getSearchValueInSTR*();  
  
 long startTime;  
 long endTime;  
  
 // Sequential Search  
 startTime = System.*nanoTime*();  
 *sequentialSearchInSTR*();  
 endTime = System.*nanoTime*();  
 System.*out*.println("Последовательный " + (endTime - startTime) + " наносекунд");  
  
 // Binary Search  
 int[] sortedStr = *quickSortRow*(Arrays.*copyOf*(*array*[0], *array*[0].length));  
 startTime = System.*nanoTime*();  
 *binarySearchInSTR*(sortedStr, targetValue);  
 endTime = System.*nanoTime*();  
 System.*out*.println("Бинарный поиск: " + (endTime - startTime) + " наносекунд");  
  
 // Fibonacci Search  
 startTime = System.*nanoTime*();  
 *fibonacciSearchInMatrix*(targetValue);  
 endTime = System.*nanoTime*();  
 System.*out*.println("Поиск Фибоначчи: " + (endTime - startTime) + " наносекунд");  
  
 // Interpolation Search  
 int[] sortedRowInterpolation = *quickSortRow*(Arrays.*copyOf*(*array*[0], *array*[0].length));  
 startTime = System.*nanoTime*();  
 *interpolationSearch*(sortedRowInterpolation, targetValue);  
 endTime = System.*nanoTime*();  
 System.*out*.println("Интерполяционный поиск:" + (endTime - startTime) + " наносекунд");  
  
 // Built-in Search  
 startTime = System.*nanoTime*();  
 *builtInSearchInMatrix*();  
 endTime = System.*nanoTime*();  
 System.*out*.println("Встроенный поиск: " + (endTime - startTime) + " наносекунд");  
 }  
  
  
  
 public static void solveTask10() {  
  
 System.*out*.print("Введите размер квадрата: ");  
 int N = *scanner*.nextInt(); // Размер квадратного массива  
 int[][] array = new int[N][N];  
  
 // Заполнение массива частично случайными значениями (1 или 0)  
 for (int i = 0; i < N; i++) {  
 for (int j = 0; j < N; j++) {  
 array[i][j] = (int) (Math.*random*() \* 2);  
 }  
 }  
  
 System.*out*.print("Введите размер квадрата для поиска: ");  
 int M = *scanner*.nextInt(); // Размер искомого квадрата  
  
 *checkAndPrintSquare*(array, M);  
 }  
  
 public static void checkAndPrintSquare(int[][] array, int M) {  
 int N = array.length;  
 boolean squareFound = false;  
  
 for (int i = 0; i <= N - M; i++) {  
 for (int j = 0; j <= N - M; j++) {  
 // проверка возможности размещения квадрата MxM в массиве  
 boolean squareFits = true;  
 for (int k = i; k < i + M; k++) {  
 for (int l = j; l < j + M; l++) {  
 if (array[k][l] != 1) { // проверяем, что значение равно 1  
 squareFits = false;  
 break;  
 }  
 }  
 if (!squareFits) {  
 break;  
 }  
 }  
 // если квадрат размещается, вывести этот квадратный массив и измененный квадрат  
 if (squareFits) {  
 squareFound = true;  
 // Вывод изначального массива  
 System.*out*.println("Изначальный массив:");  
 for (int k = 0; k < N; k++) {  
 for (int l = 0; l < N; l++) {  
 System.*out*.print(array[k][l] + " ");  
 }  
 System.*out*.println();  
 }  
 // Вывод квадрата размером MxM и его позиции  
 System.*out*.println("Квадрат размером " + M + "x" + M + " на позиции [" + i + ";" + j + "]:");  
 for (int k = i; k < i + M; k++) {  
 for (int l = j; l < j + M; l++) {  
 System.*out*.print(array[k][l] + " ");  
 }  
 System.*out*.println();  
 }  
 }  
 }  
 }  
 if (!squareFound) {  
 System.*out*.println("Невозможно разместить квадрат размером " + M + " в данном массиве");  
 }  
 }  
}

**Результат:**

Ввод: Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Вывод:

1) Ввод матрицы в ручную

2) Ввод матрицы рандомно

3) Вывод матрицы на экран

4) Сортировка матрицы по строкам

5) Поиск значения в строке матрицы

6) Решение задачи №10

7) Выход из программы

Выбрать пункт меню: 2

Введите кол-во строк:

5

Введите кол-во столбцов:

5

Выбрать пункт меню: 3

Двумерный массив:

**-943 479 948 -284 421**

**-83 -155 917 55 716**

**-711 853 -270 896 -868**

**809 775 -257 -500 -436**

**-56 -75 -930 -136 -871**

Выбрать пункт меню: 4

Методы сортировки по строкам:

1. Сортировка Шелла:

2. Быстрая сортировка:

3. Встроенная сортировка:

4. Сравнение времени сортировок:

5. Вернуться назад

Выберите метод поиска подстроки: 1

Двумерный массив:

**-943 -284 421 479 948**

**-155 -83 55 716 917**

**-868 -711 -270 853 896**

**-500 -436 -257 775 809**

**-930 -871 -136 -75 -56**

Выбрать пункт меню: 5

Методы поиска значений в строке матрицы:

1 Последовательный поиск:

2 Бинарный поиск:

3 Фибоначчиев поиск:

4 Интерполяционный поиск:

5 Поиск встроенной функцией:

6 Сравнение времени алгоритмов поиска:

7 Вернуться назад

Выберите метод поиска подстроки: 4

Введите значение для поиска: 55

**Значение 55 найдено в строке 2, столбце 3**

1) Ввод матрицы в ручную

2) Ввод матрицы рандомно

3) Вывод матрицы на экран

4) Сортировка матрицы по строкам

5) Поиск значения в строке матрицы

6) Решение задачи №10

7) Выход из программы

Выбрать пункт меню: 6

Введите размер квадрата: 15

Введите размер квадрата для поиска: 3

Изначальный массив:

**1 0 0 0 1 1 1 1 1 1 0 0 1 0 1**

**0 1 0 1 0 1 0 0 0 0 1 0 0 0 0**

**1 1 0 1 0 1 0 1 1 0 0 0 1 0 0**

**1 1 1 1 0 1 0 0 0 0 0 1 1 0 1**

**0 0 1 0 0 0 1 0 1 0 0 0 0 0 0**

**0 0 0 1 0 1 0 0 0 0 1 1 0 1 1**

**1 0 0 0 1 1 1 1 0 1 1 0 0 0 1**

**1 1 1 1 1 1 0 1 0 0 1 1 1 1 1**

**1 0 1 1 1 0 0 0 0 0 0 1 1 0 1**

**1 0 1 1 1 1 0 0 1 0 1 1 0 0 0**

**0 0 1 0 0 1 0 0 1 1 1 0 0 0 1**

**1 1 1 1 0 1 1 1 1 0 1 0 1 1 0**

**1 1 0 0 1 0 1 1 1 1 1 1 0 1 0**

**0 1 0 1 0 1 1 1 0 0 1 0 1 0 1**

**1 1 0 0 0 1 1 0 1 1 1 1 0 1 0**

Квадрат размером 3x3 на позиции [7;2]:

**1 1 1**

**1 1 1**

**1 1 1**

**Заключение:**

В процессе выполнения лабораторной работы были изучены методы сортировки и поиска, такие как: сортировка Шелла, поиск Фибоначчи, Интерполяционный поиск. Были освоены умение работы с двумерными массивами (матрицами), такие как: замена переменных в массиве, вывод на экран массива.