

МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение высшего образования*

***«МИРЭА – Российский технологический университет»***

**РТУ МИРЭА**

Отчет по выполнению практического задания № 1.1

**Тема:**

«Оценка вычислительной сложности алгоритма»

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Смирнов А. Ю.

Фамилия И.О.

Группа: ИКБО-52-23

Номер группы

Москва – 2024

СОДЕРЖАНИЕ

[1 ПЕРВОЕ ЗАДАНИЕ 3](#_Toc161174516)

[1.1 Постановка задачи 3](#_Toc161174517)

[1.2 Анализ алгоритмов 3](#_Toc161174518)

[1.2.1 Алгоритм 1 3](#_Toc161174519)

[1.2.2 Алгоритм 2 5](#_Toc161174520)

[1.3 Ёмкостная сложность 6](#_Toc161174521)

[1.4 Тестирование алгоритмов 6](#_Toc161174522)

[1.5 Вывод 8](#_Toc161174523)

[2 ВТОРОЕ ЗАДАНИЕ 9](#_Toc161174524)

[2.1 Постановка задачи 9](#_Toc161174525)

[2.2 Математическая модель решения задачи 9](#_Toc161174526)

[2.3 Разработка эффективного алгоритма 10](#_Toc161174527)

[2.4 Реализация алгоритма 11](#_Toc161174528)

[2.5 Тестирование алгоритма 12](#_Toc161174529)

[2.6 Оценка сложности алгоритма 13](#_Toc161174530)

[2.7 Вывод 13](#_Toc161174531)

# 1 ПЕРВОЕ ЗАДАНИЕ

**1.1 Постановка задачи**

Выбрать эффективный алгоритм вычислительной задачи из двух предложенных, используя теоретическую и практическую оценку вычислительной сложности каждого из алгоритмов, а также его ёмкостную сложность. Пусть имеется вычислительная задача:

– дан массив х из n элементов целого типа; удалить из этого массива все значения равные заданному (ключевому) key.

Удаление состоит в уменьшении размера массива с сохранением порядка следования всех элементов, как до, так и следующих после удаляемого. Например, необходимо удалить из массива все значения равные 2. Исходный массив (n=10): 1 2 3 2 2 2 5 2 2 2. Результат: n=3; 1 3 5.

**1.2 Анализ алгоритмов**

**1.2.1 Алгоритм 1**

Данный алгоритм проходит по каждому элементу массива и сравнивает его с key. Если значение элемента совпадает со значением key, то данный элемент удаляется, а все остальные справа от него смещаются на одну позицию влево. Иначе алгоритм переходит к следующему элементу.

Инвариант: на каждом шаге внешнего цикла алгоритм сохраняет или удаляет элемент массива, поэтому инвариант внешнего цикла: «После выполнения i-ого шага цикла первые i элементов массива [0; i), не равные key, со- хранены и стоят в том же порядке, что и до начала выполнения алгоритма». Во внутреннем цикле происходит смещение всех элементов, расположенных справа от i-ого. Тогда инвариант внутреннего цикла: «Элемент на i-ом месте удаляется, а следующие за ним элементы массива [i; n) смещаются влево на 1 позицию».

Когда i = n, в массиве остаются элементы, не равные key.

Эффективность алгоритма:

В худшем случае, где все элементы равны key, внутренний цикл совершит максимально возможное число итераций, причём оно зависит от i и n. В лучшем случае, ни один элемент не равен key, внутренний цикл выполняться не будет вообще.

Определим количество выполнений вложенного цикла в худшем случае: При i=0, оператор выполнится n раз.

При i=1, оператор выполнится n-1 раза.

При i=k, оператор выполнится n-k раз.

При i=n-2, оператор выполнится 1 раз.

При i=n-1, оператор выполнит сравнение и цикл закончится. Количество операций внутри цикла равно n (см. таблицу 1).

Таблица 1 – Таблица эффективности алгоритма 1

|  |  |  |
| --- | --- | --- |
| Оператор | Кол-во выполнений оператора в строке | |
| В лучшем случае | В худшем случае |
| int i = 0; | 1 | 1 |
| while (i < n) { | n + 1 | n + 1 |
| if (x[i] == key) { | n | n |
| for (int j = i; j < n - 1; j ++) | 0 | n |
| x[j] = x[j + 1]; | 0 | n2 |
| n--; | 0 | n |
| else i ++; | n | 0 |

Сложность:

Лучший случай: 3n + 2 (линейная); Худший случай: n2 + 3n + 2 (квадратичная).

В среднем случае внутренний цикл, как и в худшем, всегда будет выполняться, из-за чего сложность будет квадратичной.

**1.2.2 Алгоритм 2**

j-ому элементу массива x присваивается i-ый элемент до тех пор, пока не встретится отличный от значения key, в результате которого программа переходит к следующему элементу. Оператор if анализирует количество отличных от key значений, записанных в счётчик j, в конце это количество присваивается переменной n.

Инвариант: i + j ≥ 0.

В переменную n записано кол-во элементов, отличных от значения переменной key, среди первых i элементов [0; i) массива:

* в начале первой итерации исследованная область представляет собой единственную точку 1, а область неопределенности составляет [2, n);
* на втором шаге область неопределенности сокращается до [3, n);
* на третьем – до [4, n) и т. д., пока не превратится в пустое множество. Цикл конечен.

Эффективность. В данном алгоритме всего один цикл, который проходит через все элементы массива. Тогда число итераций цикла, считая выход, равно n + 1. Тело цикла, будет выполнено n раз. В лучшем случае инкрементирующий оператор не выполнится никогда, в худшем – n раз.

Данные алгоритма представлены в табл. 2.

Таблица 2 – Таблица эффективности алгоритма 2

|  |  |  |
| --- | --- | --- |
| Оператор | Количество выполнений оператора в строке | |
| В лучшем случае | В худшем случае |
| int j = 0; | 1 | 1 |
| for (int i = 0; i < n; i ++) { | n + 1 | n + 1 |
| x[j] = x[i]; | n | n |
| if (x[i] != key) | n | n |
| j ++; | 0 | n |
| n = j | 1 | 1 |

Сложность:

Лучший случай: 3n + 3 (линейная);

Худший случай: 4n + 3 (линейная).

Сложность в худшем и лучшем случаях линейная. Значит, в среднем случае будет аналогично.

**1.3 Ёмкостная сложность**

У обоих алгоритмов ёмкостная сложность равна n, так как они не требуют дополнительного массива для решения задачи.

**1.4 Тестирование алгоритмов**

Посмотрим, сколько итераций понадобится алгоритмам в разных ситуациях. Произведём подсчет суммарного количества выполненных сравнений и перемещений элементов. Приведём таблицу 3 сравнения работы двух алгоритмов.

Таблица 3 – Сравнение алгоритмов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Длина массива (n) | Количество итераций | | | | |
|  | | Случайные числа | Все элементы удалены | Ни один элемент не удалён | Расчёт | |
|  | | | | | Лучший случай | Худший случай |
| 1 | 10 | 27 | 58 | 10 | n | 50,9n |
| 100 | 869 | 5098 | 100 |
| 2 | 10 | 20 | 20 | 20 | 2n | 2n |
| 100 | 200 | 200 | 200 |

Как видно из таблицы, второй алгоритм по большей части совершает меньшее число итераций, чем первый, причём их количество остаётся неизменным при постоянном размере массива, т. е. время работы алгоритма 2 не зависит от значений элементов массива.

В алгоритме 1 если в массиве все элементы равны key, то алгоритм будет выполняться максимально долго; если же все элементы не равны key, то - наоборот. Однако стоит отметить, что в массив вводились случайным образом элементы от 1 до 10, но если этот интервал увеличить, то в таком случае вероятность наличия конкретного числа будет уменьшаться, из-за чего число итераций первого алгоритма будет стремиться к ситуации «Ни один элемент не удалён», и тогда первый алгоритм будет чаще более быстрым по времени.

**1.5 Вывод**

Наиболее эффективным можно назвать алгоритм 2 по следующим причинам:

- Линейная сложность;

- Всего один цикл;

- Скорость работы не зависит от значений элементов массива;

# 2 ВТОРОЕ ЗАДАНИЕ

**2.1 Постановка задачи**

Определить, симметрична ли матрица относительно главной диагонали.

**2.2 Математическая модель решения задачи**

Данный алгоритм считывает с клавиатуры size – размер матрицы. Затем, в двойном цикле, повторяющемся (size) раз, заполняется значениями матрица. После чего, в двойном цикле, i = 0 повторятся (size) раз, j = (i+1) повторяется (size) раз, если матрица[ i ][ j ] != матрица[ j ][ i ], то матрица является не симметричной относительно главной диагонали, иначе она симметрична.

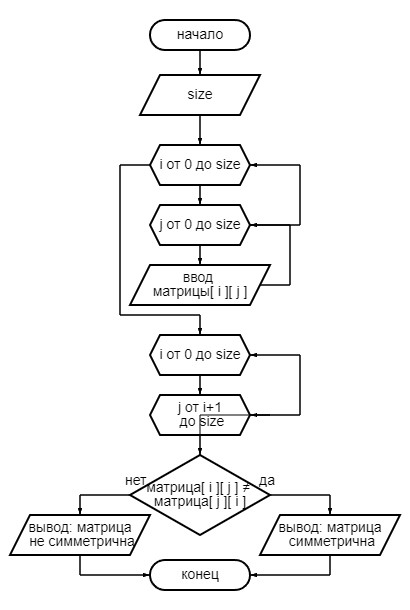
 Блок-схема алгоритма представлена ниже:

Рис. 1. Блок-схема

**2.3 Разработка эффективного алгоритма**

Инвариант: "В начале i-й итерации в переменной symetric хранится значение, которое указывает на то, является ли матрица симметричной относительно главной диагонали для первых i строк и столбцов. Если symetric равно true, то все предыдущие строки и столбцы были симметричны. Если symetric становится false, это означает наличие несимметричного элемента относительно главной диагонали."

Когда i = size, цикл завершается. Если symetric равно true, то в этот момент все строки и столбцы были проверены, и матрица симметрична относительно главной диагонали. Поскольку изменение i не происходит внутри цикла, и исходное значение i равно 0, а предел итераций равен n, количество итераций ограничено, что гарантирует конечность цикла.

Вычислительная сложность алгоритма:

Таблица 1 – Таблица эффективности алгоритма

|  |  |  |
| --- | --- | --- |
| Оператор | Кол-во выполнений оператора в строке | |
| В лучшем случае | В худшем случае |
| bool symetric = true | 1 | 1 |
| for (int i=0; i < size; i ++){ | 1 | size + 1 |
| for (int j=i+1; i < size; i ++){ | 1 | Size-1 |
| If (matrix[i][j]!=matrix[j][i]){ | (size-1)/2 | (size-1)/2 |
| symetric = false | 0 | 1 |
| Cout << “…”  }  } | 0 | (size-1)/2 |
| If (!symmetric){ | 1 | 1 |
| Cout << “…”  }  } | 1 | 1 |

Определим время выполнения алгоритма - как сумму времени выполнения каждого оператора в лучшем случае (size = n):

T(n) = 1+1+1+(n-1)/2+1+1

T(n) = 5+(n-1)/2, т.е имеем линейную зависимость.

Теперь в худшем случае:

T(n) = 1+n+1+n-1+(n-1)/2+(n-1)/2+1+1+1

T(n) =3 + 3\*n, т.е имеем линейную зависимость.

**2.4 Реализация алгоритма**

Ниже представлена программа на языке C++

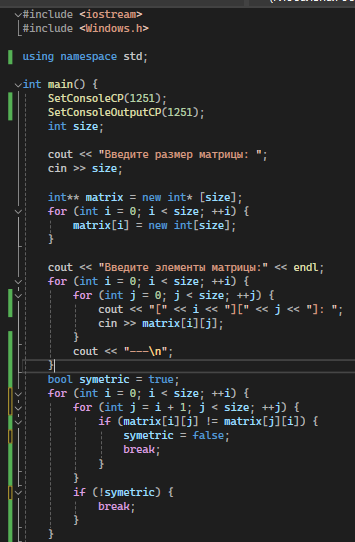


Рис.2 – Алгоритм на языке C++ (ч.1)

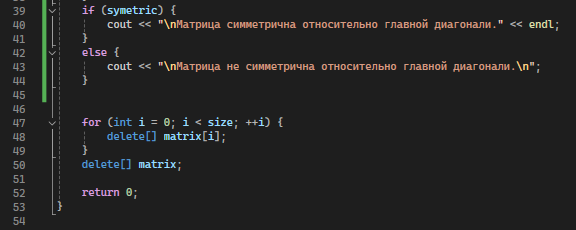
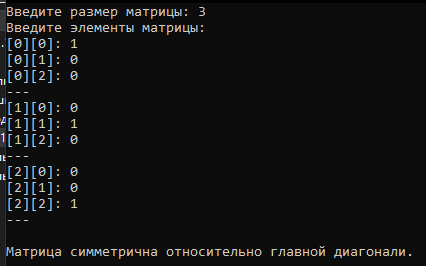
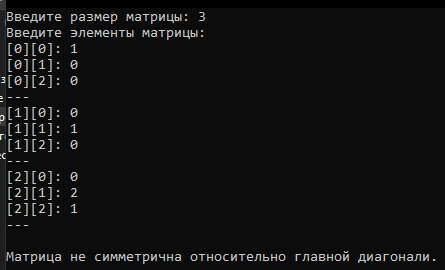


Рис.3 – Алгоритм на языке C++ (ч.2)

**2.5 Тестирование алгоритма**

Ниже представлена работа программы на различных входных данных.

****

**2.6 Оценка сложности алгоритма**

В таблице приведены данные о количестве итераций на разных входных значений size. При лучшем случае и худшем случае результат разный.

Таблица 3 – Сравнение алгоритмов

|  |  |  |  |
| --- | --- | --- | --- |
| N | Длина массива  (size) | Количество итераций | |
| Лучший случай | Худший случай |
| 1 | 11 | 10 | 36 |
| 2 | 101 | 55 | 306 |

**2.7 Вывод**

Разработан и протестирован алгоритм, определяющий симметричность матрицы относительно главной диагонали.