

МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение высшего образования*

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

**РТУ МИРЭА**

Отчет по выполнению практического задания № 1

**Тема:**

«Оценка вычислительной сложности алгоритма»

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Моисенко М. О.

Фамилия И.О.

Группа: ИКБО-00-22

Номер группы

Москва – 2023

**СОДЕРЖАНИЕ**

# ПОСТАНОВКА ЗАДАЧИ

Цель: приобретение практических навыков:

* эмпирическому определению вычислительной сложности алгоритмов на теоретическом и практическом уровнях;
* выбору эффективного алгоритма решения вычислительной задачи из нескольких.

## Условие задания 1

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.

Можно предложить два подхода к решению данной задачи, т.е. два алгоритма. Они представлены в табл. 1. Необходимо реализовать эти алгоритмы, оценить их вычислительную сложность теоретически и практически и сделать вывод об их эффективности.

Таблица 1 – Два алгоритма решения задачи

|  |  |
| --- | --- |
| *x* – массив, *n* – количество элементов в массиве, *key* – удаляемое значение | |
| Алгоритм 1:  **delFirstMetod** (x, n, key) {  i ← 1  **while** (i <= n) **do**  **if** x[i] = key **then**  //удаление  **for** j ← i **to** n - 1 **do**  x[j] ← x[j+1]  **od**  n ← n - 1  **else**  i ← i + 1  **endif**  **od**  } | Алгоритм 2:  **delOtherMetod** (x, n, key) {  j ← 1  **for** i ← 1 **to** n **do**  x[j] ← x[i]  **if** x[i] != key **then**  j++  **endif**  **od**  n ← j  } |

## Условие задания 2

Выполнить разработку алгоритма в соответствии с задачей варианта.

Задача варианта 6: “Найти максимальный элемент в части матрицы, расположенной над главной диагональю.”

# ЗАДАНИЕ 1

## Формулировка задачи

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

## Математическая модель решения задачи (алгоритм №1)

Создается числовая переменная i, чье значение будет равно индексу текущего положения в массиве. Ей присваивается значение, равное первому индексу массива. Пока переменная i меньше или равна переменной n, которая содержит в себе информацию о количестве элементов в массиве, текущий элемент сравнивается с заданной переменной key. Если их значения совпадают, элемент удаляется, иначе к значению i прибавляется один. Элемент удаляется через присваивание его ячейки значения ячейки с индексом на 1 больше, чем у ячейки совпавшего элемента, аналогичное происходит и с ячейками с индексами с i+1 по n-1, а n уменьшается на 1.

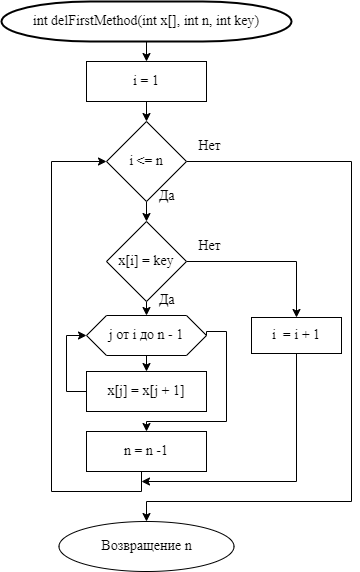


Рисунок 1 – Блок-схема алгоритма 1

Инвариант цикла:

Для любой i-ой итерации среди элементов массива с 0 по i-1 нет элементов со значением key. Размер массива это целое натуральное число, и мы либо увеличиваем переменную i на один, либо уменьшаем n на один, что через некоторое количество иттераций выводит из цикла => цикл конечен.

Таблица 2 – Подсчет количества операторов в алгоритме 1

|  |  |  |
| --- | --- | --- |
| Оператор | Кол-во выполнения оператора в строке | |
| в лучшем случае  (в массиве нет элементов key) | в худшем случае  (все элементы массива равны key) |
| 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 | (n2+n) / 2 |
| x[j] = x[j + 1]; | 0 | (n2+n) / 2 |
| n--; | 0 | n |
| i++; | n | 0 |
| return n; | 1 | 1 |

Из табл. 2 делаем выводы:

T(n) (в лучшем) = 3n + 2 <=> T(n) = n => Линейная зависимость.

Т(n) (в худшем) = n2 + 4n + 3 <=> T(n) = n2 => Квадратичная.

Для среднего случая возьмем, что количество элементов key составляет половину количества всех элементов массива. Легко заметить, что зависимость остается также квадратичной за счет вложенного цикла.

## Реализация алгоритма 1 в виде функции и отладка на массиве при n=10, n=100

На рисунке 2 предоставлен код функции, который соответствует алгоритму 1 с подсчетом количества выполненных сравнений и перемещений элементов.



Рисунок 2 – код функции, реализующая первый алгоритм

На рис.3 продемонстрирована работа функции на массиве из 10 элементов, где ключевой элемент равнялся единице.

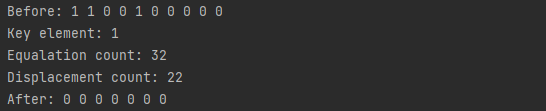


Рисунок 3 – Отладка функции на массиве из 10 элементов

Можно увидеть, что было запущено 32 операций сравнения и 22 операций переноса, что соответствует Т(n) = 5,4n.

На рис.4 продемонстрирована работа функции на массиве из 100 элементов, где ключевой элемент равнялся нулю.

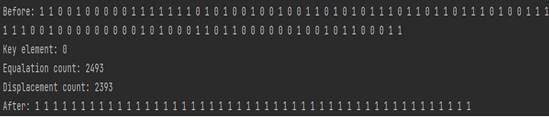


Рисунок 4 – Отладка функции на массиве из 100 элементов

Можно увидеть, что было запущено 2493 операций сравнения и 2393 операций переноса, что соответствует Т(n) = 48,86n.

## Тестирование алгоритма 1 в различных ситуациях

### Случайное заполнение массива

В пункте 2.2 уже демонстрировалась работа алгоритма 1 в случае заполнения массива случайными числами.

### Удаления элементов не происходит

На рис. 5 продемонстрирована работа алгоритма в случае, если ни один элемент не совпал с ключевым. Согласно теоретическим расчетам, количество операций должен отражать линейную зависимость, т. е. равным количеству элементов, что в данном случае равно 10.

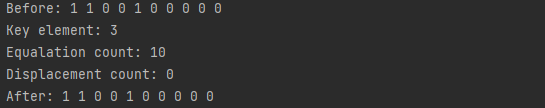


Рисунок 5 – Отладка на массиве, в котором ни один элемент не равен key

### Удаление всех элементов

Результаты тестирования алгоритма 1 в случае, если все элементы равны key и удаляются представлены на рис. 6. Согласно теоретическим расчетам, количество операций должен отражать квадратичную зависимость, т. е. равным квадрату количества элементов, что в данном случае равно 100.

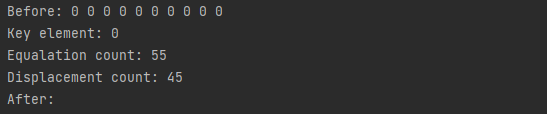


Рисунок 6 – Отладка на массиве, в котором все элементы удаляются

Чем больше элементов, равных key, тем больше и время работы алгоритма.

## Математическая модель решения задачи (алгоритм №2)

Задается переменная j = 1, обозначающая индекс текущего элемента. Алгоритм перебирает все индексы массива, передавая их значение переменной i, элементу с индексом j присваивается значение элемента с индексом i, и, если этот элемент с индексом i не равен ключевому значению, то переменная j увеличивается на 1. Таким образом все элементы, не равные ключевому значению, перемещаются в начало массива. В конце переменная n, хранящая в себе информацию о длине массива, принимает значение j.

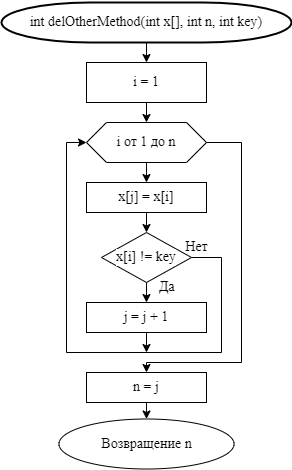


Рисунок 7 – Блок-схема алгоритма 2

Инвариант цикла:

Для любой i-ой итерации среди элементов массива с 0 по i-1 нет элементов со значением key. Число элементов в массиве конечно, цикл идет от 1 до n => цикл конечный.

Таблица 3 – Подсчет количества операторов в алгоритме 1

|  |  |  |
| --- | --- | --- |
| Оператор | Кол-во выполнения оператора в строке | |
| в лучшем случае  (все элементы массива равны key) | в худшем случае  (в массиве нет ни одного элемента key) |
| int j = 0; | 1 | 1 |
| for (int i = 0; i < n; i++) | n | n |
| x[j] = x[i]; | n | n |
| if (x[i] != key) | n | n |
| j++; | 0 | n |
| n = j; | 1 | 1 |
| return n; | 1 | 1 |

Из табл. 3 делаем выводы:

T(n) (в лучшем) = 3n + 3 <=> T(n) = n => Линейная зависимость.

Т(n) (в худшем) = 4n + 3 <=> T(n) = n=> Линейная зависимость.

Следовательно, Т(n) (в среднем) = n. Линейная зависимость.

## Реализация алгоритма 2 в виде функции и отладка на массиве при n=10, n=100

На рисунке 8 предоставлен код функции, который соответствует алгоритму 2 с подсчетом количества выполненных сравнений и перемещений элементов.



Рисунок 8 – код функции, реализующая первый алгоритм

На рис.9 продемонстрирована работа функции на массиве из 10 элементов, где ключевой элемент равнялся единице.

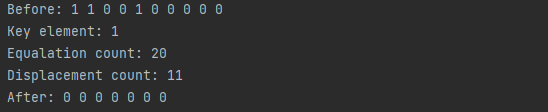


Рисунок 9 – Отладка функции на массиве из 10 элементов

Можно увидеть, что было запущено 20 операций сравнения и 11 операций переноса, что соответствует Т(n) = 3,1n.

На рис.10 продемонстрирована работа функции на массиве из 100 элементов, где ключевой элемент равнялся нулю.

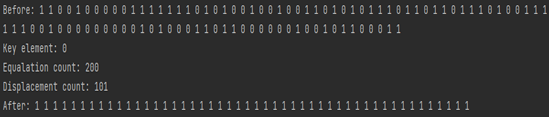


Рисунок 10 – Отладка функции на массиве из 100 элементов

Можно увидеть, что было запущено 200 операций сравнения и 101 операций переноса, что соответствует Т(n) = 3,01n.

## Тестирование алгоритма 2 в различных ситуациях

### Случайное заполнение массива

В пункте 2.5 уже демонстрировалась работа алгоритма 2 в случае заполнения массива случайными числами.

### Удаления элементов не происходит

На рис. 11 продемонстрирована работа алгоритма в случае, если ни один элемент не совпал с ключевым. Согласно теоретическим расчетам, количество операций должен отражать линейную зависимость, т. е. равным количеству элементов, что в данном случае равно 10.

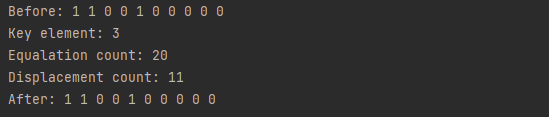


Рисунок 11 – Отладка на массиве, в котором ни один элемент не равен key

### Удаление всех элементов

Результаты тестирования алгоритма 2 в случае, если все элементы равны key и удаляются представлены на рис. 12. Согласно теоретическим расчетам, количество операций должен отражать линейную зависимость, т. е. равным количеству элементов, что в данном случае равно 10.

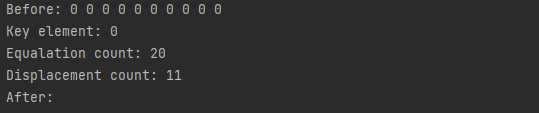


Рисунок 12 – Отладка на массиве, в котором все элементы удаляются

Количество операций не зависит от содержимого массива, во всех случаях общее их количество равно T(n) = 3n + 1 <=> T(n) = n.

## Реализация функций: заполнение массива случайными числами и вывод массива на экран

На рисунке 13 показана реализация функции, которая заполняет массив случайными числами в пределе от 0 до 1. На вход принимает массив и его размер.

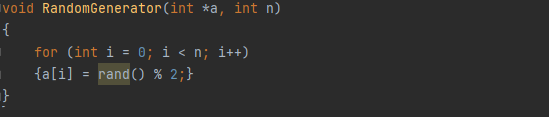


Рисунок 13 – Код функции, заполняющая массив случайными значениями

На рисунке 14 продемонстрирован код функции, которая выводит в консоль массив. На вход принимает массив и его размер.

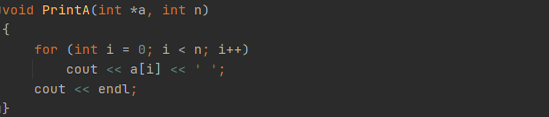


Рисунок 14 – Код функции, выводящей массив в консоль

## Вывод задания 1

Алгоритм 2 в любом случае имеет линейную сложность, которая не зависит от состава массива, что делает его более эффективным, чем алгоритм 1, который имеет в среднем квадратичную сложность и зависит от количества ключевых элементов в массиве.

# ЗАДАНИЕ 2

## Формулировка задачи

Моя индивидуальная задача представлена под номером 6: “Найти максимальный элемент в части матрицы, расположенной над главной диагональю.” Пусть матрица является квадратной и заполнена целыми числами.

## Математическая модель решения задачи

Задается переменная max\_a, которая будет содержать максимальное значение, и первоначально ей присваивается значение элемента матрицы с индексом [0,1]. Затем алгоритм перебирает значения от 0 до n – 2, передавая их значение переменной i, и значения от i+1 до n-1, передавая это переменной j. По завершению, в переменной max\_a хранится максимальное значение в части матрицы, которая находится над главной диагональю.

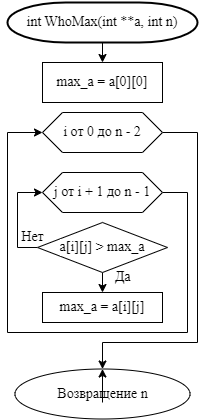


Рисунок 15 – Блок-схема алгоритма решения второй задачи

Оба цикла конечны, так как перебирают значения до n – k, где k = , в n содержится размер массива, что является конечным числом и не изменяется за время работы алгоритма.

Инвариант цикла:

Для любой i-той итерации в части матрицы, находящейся над главной диагональю, до элемента a[k][n], который соответствует i-той итерации, все элементы меньше или равны переменной max\_a.

Сложность этого алгоритма в любом случае будет линейной (T(n) = n), если говорить относительно элементов массива, так как алгоритм в любом случае проходит по всем значениям той части матрицы, что находится над главной диагональю.

## Реализация алгоритма решения задачи в виде одной функции

Реализация алгоритма представлена на рис. 16.



Рисунок 16 – Реализация алгоритма решения задачи на языке C++

## Тестирование алгоритма

Слова мои слова.

Таблица 5 – Тестирование алгоритма на разных входных данных

|  |  |  |
| --- | --- | --- |
| Входной параметр | Количество операций | Время работы, с |
| 2 |  |  |
| 10 |  |  |
| 100 |  |  |
| 1000 |  |  |

Результаты тестирования показаны на рис. 17 и рис. 18.

