



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: Боргачев Т.М.

Группа: ИНБО-10-23

Москва – 2024

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ.....	6
1 ФОРМУЛИРОВКА ЗАДАЧИ	7
1.1 Задание 1.....	7
1.2 Задание 2 – Индивидуальная задача	8
2 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ.....	8
2.1 Задание 1.....	8
2.1.1 Математическая модель решения задачи	8
2.1.2 Реализация алгоритма в виде функции.....	11
2.1.3 Реализация функций	14
2.1.4 Тестирование алгоритма в различных ситуациях	14
2.1.5 Оценка ёмкостной сложности алгоритма	16
2.2 Задание 2.....	17
2.2.1 Математическая модель решения задачи	17
2.2.2 Реализация алгоритма в виде функции.....	19
2.2.3 Тестирование алгоритма.....	20
2.2.4 Практическая оценка сложности алгоритма для больших n	21
2.2.5 Оценка ёмкостной сложности алгоритма	22
3 ВЫВОДЫ	22
4 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ.....	23

1 ФОРМУЛИРОВКА ЗАДАЧИ

Цель: приобретение практических навыков:

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

1.1 Задание 1

Требуется 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 x 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.

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

x-массив, n – количество элементов в массиве, key – удаляемое значение	
Алгоритм 1: delFirstMetod(x,n,key) { $i \leftarrow 1$ while ($i \leq n$) do if $x[i]=key$ then //удаление for $j \leftarrow i$ to $n-1$ do $x[j] \leftarrow x[j+1]$ od $n \leftarrow n-1$ else $i \leftarrow i+1$ endif od }	Алгоритм 2: delOtherMetod(x,n,key) { $j \leftarrow 1$ for $i \leftarrow 1$ to n do $x[j] \leftarrow x[i];$ if $x[i] \neq key$ then $j++$ endif od $n \leftarrow j$ }

Рисунок 1 – Алгоритмы сортировки

1.2 Задание 2 – Индивидуальная задача

Выполнить разработку алгоритма в соответствии с задачей варианта 5: Обход матрицы по спирали (по часовой стрелке: первая строка, последний столбец, нижняя строка, первый столбец).

2 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ

2.1 Задание 1

2.1.1 Математическая модель решения задачи

Первый алгоритм проверяет каждый элемент массива, идя по циклу, на совпадение со значением key и, при совпадении, начинает заменять каждый элемент, включая текущий, на следующий. Работа алгоритма представлена в виде блок-схемы на рис. 2.

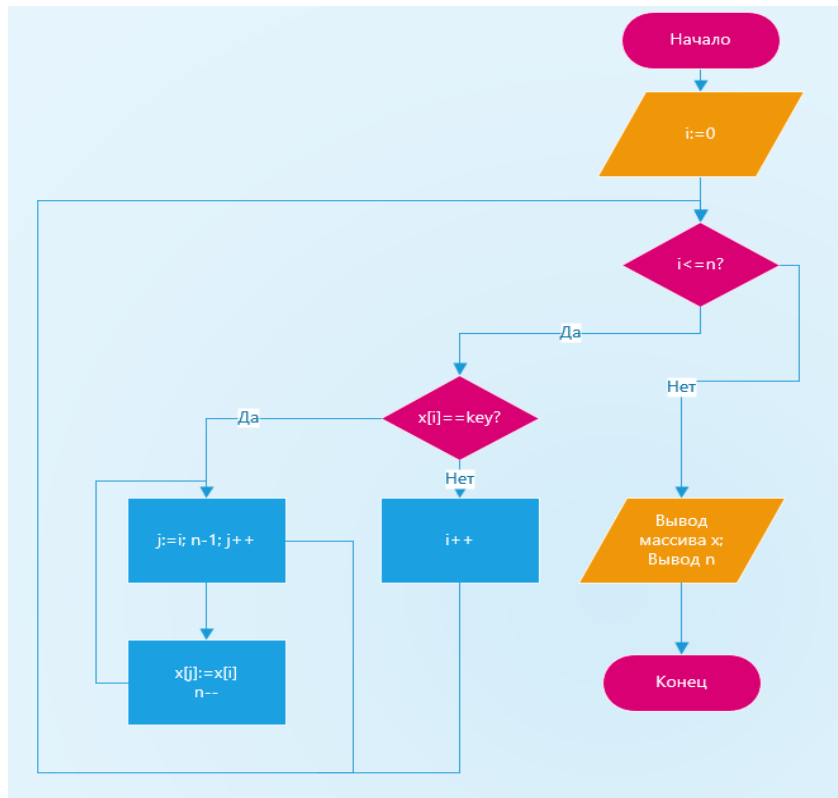


Рисунок 2 – Блок-схема первого алгоритма

Второй алгоритм основан на одном цикле, в котором каждый элемент, в зависимости от совпадения своего значения с key , остается в массиве (при несовпадении) либо заменяется на следующий (при совпадении), тем самым удаляется из массива. Конечное количество элементов совпадает со значением первой переменной-счетчика j . Работа алгоритма представлена в виде блок-

схемы на рис. 3.

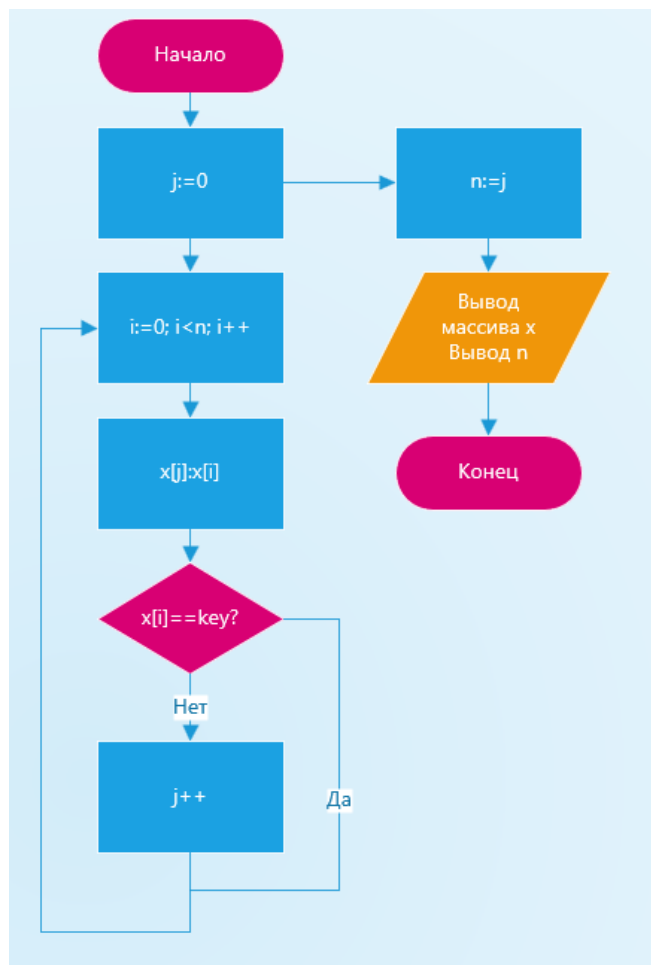


Рисунок 3 – Блок-схема второго алгоритма

Инвариантом цикла первого алгоритма можно назвать то, что количество элементов массива x всегда больше или равно переменной i . То есть: $n \geq i$. Докажем что цикл корректен:

1. При инициализации $n \geq 1$, $i=0$ – нет противоречия.
2. Пусть $x = [1, 2, 3, 5, 4, 3, 3, 2]$, $key = 3$, $i=0$. Каждый раз когда $x[i] \neq key$ – i увеличивается на 1. Каждый раз когда $x[i] = key$ – n уменьшается на 1. При ситуации $i == n$ – цикл завершается. Ситуации, когда n уменьшается и i увеличивается одновременно, нет. Следовательно, нет противоречия.
3. Завершение цикла происходит при $i == n$.
4. Цикл завершится, как только будут удалены все элементы, значения которых совпадают с key .

Инвариантом цикла второго алгоритма можно назвать $i \geq j$. Докажем что

цикл корректен:

1. При инициализации $j=0$, $i=0$ – нет противоречия
2. Значение i увеличивается за каждое выполнение тела цикла, значение j увеличивается за каждое выполнение тела цикла, если $x[i] \neq \text{key}$, случая когда значение j увеличивается, а значение i не увеличивается – нет. Следовательно, нет противоречия.
3. Цикл завершается при $i == n$, значение i увеличивается, значение n либо уменьшается, либо не меняется. Следовательно, нет противоречия.
4. За каждое прохождение цикла значение i приближается к значению n , так как изначально $n \geq 1$. Следовательно, цикл конечен.

Определим вычислительную сложность первого алгоритма, для этого посчитаем количество выполнений каждого оператора. Для подсчета используем табл. 1.

Таблица 1 – Подсчет количества выполнений операций первого алгоритма

Оператор	Кол-во выполнений оператора в строке	
	в лучшем случае	в худшем случае
<code>int i = 0;</code>	1	1
<code>while (i < n){</code>	1	n
<code>if (x[i] == key) {</code>	0	n
<code>for (int j = i; j < n - 1; j++) {</code>	0	$n + (n-1)^2$
<code>x[j] = x[j + 1]; }</code>	0	$(n-1)^2$
<code>n--; }</code>	0	n
<code>else { i++; }</code>	1	n
<code>for (int i = 0; i < n; i++) {</code>	2	$n+1$
<code>cout << x[i] << " "; }</code>	1	n
<code>cout << "\n" << n << endl;</code>	1	1

Таким образом, функция роста первого алгоритма в лучшем случае примет вид $T(n) = 7$, а в худшем случае $T(n) = 2n^2 + 3n + 5$. Отсюда можно

сделать вывод: время работы алгоритма линейно зависит от n^2 .

Определим вычислительную сложность второго алгоритма. Для подсчета используем табл. 2.

Таблица 2 – Подсчет количества выполнений операций второго алгоритма

Оператор	Кол-во выполнений оператора в строке	
	в лучшем случае	в худшем случае
<code>int j = 0;</code>	1	1
<code>for (int i = 0; i < n; i++) {</code>	2	$n+1$
<code> x[j] = x[i];</code>	1	n
<code> if (x[i] != key) {</code>	0	n
<code> j++; }</code>	0	n
<code>}</code>		
<code>n = j;</code>	1	1
<code>for (int i = 0; i < n; i++) {</code>	0	$n+1$
<code> cout << x[i] << " "; }</code>	0	n
<code>cout << "\n" << n << endl;</code>	1	1

Таким образом, функция роста второго алгоритма в лучшем случае примет вид $T(n) = 6$, а в худшем случае $T(n) = 6n + 5$. Отсюда можно сделать вывод: время работы алгоритма линейно зависит от n .

На основе полученной линейной зависимости количества операций от n входных данных для обоих алгоритмов, сделаем вывод, что первый алгоритм сложнее относительно количества выполняемых операций, поэтому второй алгоритм наиболее эффективен.

2.1.2 Реализация алгоритма в виде функции

Реализация первого алгоритма в виде функции с подсчетом суммарного количества выполненных сравнений и перемещений элементов при решении задачи на языке программирования C++ представлена на рис. 4:

```

void delFirstMetod(int* x, int n, int key) {
    int i = 0, oper=0; // 1
    while (i < n) { // n
        if (x[i] == key) { // n
            for (int j = i; j < n - 1; j++) { // n + (n-1)^2
                x[j] = x[j+1]; // (n-1)^2
                oper++;
            }
            n--; // n
            oper++;
        }
        else {
            i++; // n
            oper++ ;
        }
        oper++;
    }
    for (int i = 0; i < n; i++) { // n+1
        cout << x[i] << " "; // n
        oper++;
    }
    cout << "\n" << "Конечное количество элементов массива: " << n << endl; //1
    cout << "Количество сравнений и перемещений равно: " << oper << endl;
    // Функция роста T(n) = 2n^2 + 3n + 5
}

```

Рисунок 4 – Реализация первого алгоритма в виде функции

Реализация второго алгоритма в виде функции с подсчетом суммарного количества выполненных сравнений и перемещений элементов при решении задачи на языке программирования C++ представлена на рис. 5:

```

void delOtherMetod(int* x, int& n, int key) {
    int j = 0, oper=0; // 1
    for (int i = 0; i < n; i++) { // n + 1
        x[j] = x[i]; // n
        oper+=2;
        if (x[i] != key) { // n
            j++; // n
            oper++;
        }
    }
    n = j; // 1
    for (int i = 0; i < n; i++) { // n + 1
        cout << x[i] << " "; // n
        oper++;
    }
    cout << "\n" << "Конечное количество элементов массива: " << n << endl; //1
    cout << "Количество сравнений и перемещений равно: " << oper << endl;
    // Функция роста T(n) = 6n + 5
}

```

Рисунок 5 – Реализация второго алгоритма в виде функции

Отладка алгоритмов при значении n=10 представлена на рис. 6 и 7.


```

int main()
{
    int n, key;
    setlocale(LC_ALL, "Rus");
    while (true) {
        cout << "Введите n: \n";
        cin >> n;
        cout << "Введите key: \n";
        cin >> key;
        srand(time(NULL));
        int temp;
        int* x = new int[n];
        int* y = new int[n];
        for (int i = 0; i < n; i++) {
            temp = rand() % 100; //генерация случайных значений для элементов массива
            x[i] = temp;
            y[i] = temp;
        }
        delFirstMetod(x, n, key);
        delete[] x; // Освобождаем память после использования
        delOtherMetod(y, n, key);
        delete[] y; // Освобождаем память после использования
    }
}

```

Рисунок 6 – Функция для отладки алгоритмов

```

Введите n:
10
Введите key:
2
Отсортированный массив x: 40 22 60 31 47 69 41 76 54
Конечное количество элементов массива: 9
Количество сравнений и перемещений равно: 32
Отсортированный массив x: 40 22 60 31 47 69 41 76 54
Конечное количество элементов массива: 9
Количество сравнений и перемещений равно: 38

```

Рисунок 7 – Вывод программы при n=10

Отладка алгоритмов при значении n=100 представлена на рис. 8.

```

Введите n:
100
Введите key:
3
Отсортированный массив x: 25 82 12 54 50 56 31 78 4 23 7 78 95 39 61 32 2 81 81 16 84 26 33 88 99 4 43 31 9 46 90 55 81 12 30 70 75 78 97 51 61 89 36 55 8 3
8 19 26 71 81 80 7 46 44 97 55 17 7 60 22 21 43 95 46 67 28 20 80 53 36 98 72 2 73 84 45 86 96 13 61 52 34 25 42 2 19 4 57 46 54 21 74 58 32 68 64 75 13
Конечное количество элементов массива: 98
Количество сравнений и перемещений равно: 484
Отсортированный массив x: 25 82 12 54 50 56 31 78 4 23 7 78 95 39 61 32 2 81 81 16 84 26 33 88 99 4 43 31 9 46 90 55 81 12 30 70 75 78 97 51 61 89 36 55 8 3
8 19 26 71 81 80 7 46 44 97 55 17 7 60 22 21 43 95 46 67 28 20 80 53 36 98 72 2 73 84 45 86 96 13 61 52 34 25 42 2 19 4 57 46 54 21 74 58 32 68 64 75 13
Конечное количество элементов массива: 98
Количество сравнений и перемещений равно: 396

```

Рисунок 8 – Вывод программы при n=100

2.1.3 Реализация функций

Заполнение массива датчиком случайных чисел представлено на рис. 9.

```
int* RandChisla(int n){
    srand(time(NULL));
    int* x = new int[n];
    for (int i = 0; i < n; i++) {
        x[i] = rand() % 100; //генерация случайных значений для элементов массива
    }
    return(x);
}
```

Рисунок 9 – Датчик случайных чисел для массива

Вывод массива на экран монитора представлен на рис. 10.

```
void Output(int*y,int n) {
    cout << "\nИзмененный массив x : ";
    for (int i = 0; i < n; i++) {
        cout << y[i] << " ";
    }
    cout << endl;
}
```

Рисунок 10 – Вывод массива на экран монитора

2.1.4 Тестирование алгоритма в различных ситуациях

Результаты работы алгоритмов при случайном заполнении массивов представлены на рис. 11.

```
Введите n:
30
Введите key:
3
Первоначальный массив x: 65 85 25 59 29 19 30 56 87 11 79 27 48 64 35 15 84 3 74 40 57 84 50 32 58 64 97 94 51 25
n = 29
Количество сравнений и перемещений равно: 72

Измененный массив x : 65 85 25 59 29 19 30 56 87 11 79 27 48 64 35 15 84 74 40 57 84 50 32 58 64 97 94 51 25
Затраченное время: 0.006
n = 29
Количество сравнений и перемещений равно: 89

Измененный массив x : 65 85 25 59 29 19 30 56 87 11 79 27 48 64 35 15 84 74 40 57 84 50 32 58 64 97 94 51 25
Затраченное время: 0.004
```

Рисунок 11 – Случайное заполнение массивов

Результаты работы алгоритмов при удалении всех элементов массивов представлены на рис.12.

```

Первоначальный массив x: 3 3 3 3 3 3 3 3 3 3
n = 0
Количество сравнений и перемещений равно: 65

Измененный массив x :
Затраченное время: 0.001
n = 0
Количество сравнений и перемещений равно: 20

Измененный массив x :
Затраченное время: 0.001

```

Рисунок 12 – Удаление всех элементов

Результаты работы алгоритмов в случае, когда ни один элемент не удаляется, представлены на рис. 13.

```

Первоначальный массив x: 3 4 5 6 7 8 9 10 11 12
n = 10
Количество сравнений и перемещений равно: 20

Измененный массив x : 3 4 5 6 7 8 9 10 11 12
Затраченное время: 0.003
n = 10
Количество сравнений и перемещений равно: 30

Измененный массив x : 3 4 5 6 7 8 9 10 11 12
Затраченное время: 0.003

```

Рисунок 13 – Ни один элемент не удаляется

Таким образом, можно сделать вывод о том, что первому алгоритму нужна большая вычислительная мощность для удаления объектов из массива, нежели второму, но при отсутствии удаляемых элементов, первый алгоритм справляется лучше для малых значений n . А также время, затраченное на выполнение алгоритмов, либо равное, либо лучшее для второго алгоритма.

Проведем теоретические расчёты - вычислим количество операций для первого и второго алгоритма при помощи функции роста, воспользуемся табл. 3 и 4.

Таблица 3 – Количество операций первого алгоритма

Оператор	Кол-во выполнений оператора в строке
<code>int i = 0;</code>	1
<code>while (i < n) {</code>	n
<code>if (x[i] == key) {</code>	n
<code>for (int j = i; j < n - 1; j++) {</code>	$n + (n-1)^2$

Оператор	Кол-во выполнений оператора в строке
<code>x[j] = x[j + 1]; }</code>	$(n-1)^2$
<code>n--; }</code>	n
<code>else { i++; }</code>	n
<code>for (int i = 0; i < n; i++) {</code>	$n+1$
<code>cout << x[i] << " "; }</code>	n
<code>cout << "\n" << n << endl;</code>	1

Тогда: функция роста $T(n) = 2n^2 + 3n + 5$. Таким образом, получаем квадратичную зависимость количества операций от n^2 .

Таблица 4 – Количество операций второго алгоритма

Оператор	Кол-во выполнений оператора в строке
<code>int j = 0;</code>	1
<code>for (int i = 0; i < n; i++) {</code>	$n+1$
<code>x[j] = x[i];</code>	n
<code>if (x[i] != key) {</code>	n
<code> j++; }</code>	n
<code>}</code>	
<code>n = j;</code>	1
<code>for (int i = 0; i < n; i++) {</code>	$n+1$
<code> cout << x[i] << " "; }</code>	n
<code>cout << "\n" << n << endl;</code>	1

Тогда: функция роста $T(n) = 6n + 5$. Следовательно, получаем линейную зависимость количества операций от n .

2.1.5 Оценка ёмкостной сложности алгоритма

В качестве исходных данных, с которыми работает алгоритм, на вход поступает n элементов массива x , следовательно ёмкостная сложность алгоритма равна n .

2.2 Задание 2

2.2.1 Математическая модель решения задачи

В качестве исходных данных алгоритму предоставляется матрица x размера $m \times n$, где m – количество строк, n – количество столбцов. Алгоритм выводит элементы матрицы по спирали:

1. Выводит первую строку, увеличивая значение переменной-счетчика строк;
2. Выводит последний столбец, уменьшая значение n ;
3. Выводит нижнюю строку в обратном порядке, уменьшая значение m ;
4. Выводит первый столбец в обратном порядке, увеличивая значение переменной-счетчика столбцов.

Работа алгоритма в виде блок-схемы представлена на рис. 14.

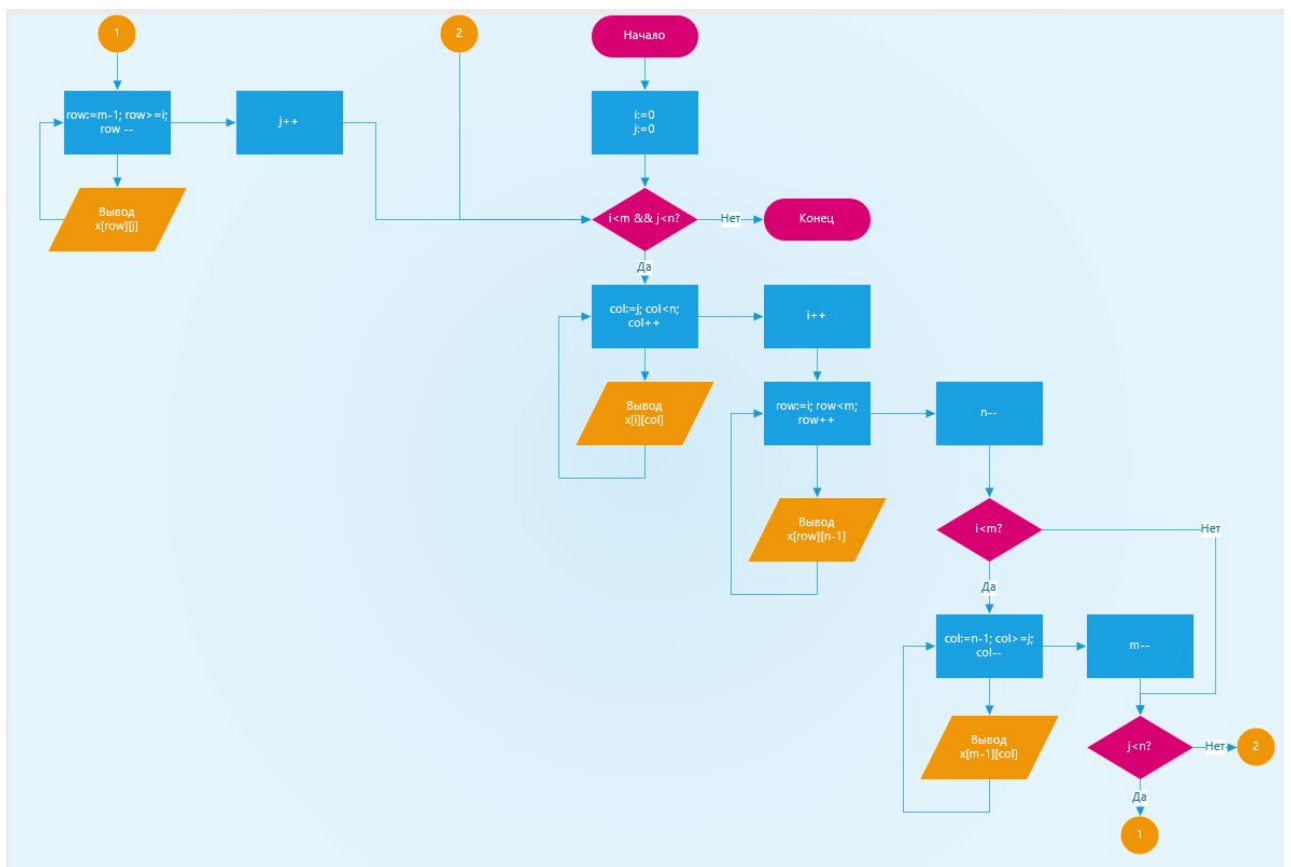


Рисунок 14 – Блок-схема алгоритма

Инвариантом цикла можно считать условие $m \geq i$ and $n \geq j$.

Для того чтобы доказать, что условие является инвариантом цикла while, давайте проверим его выполнение на всех этапах цикла:

Истинность при входе: при входе в цикл i и j инициализируются нулевыми значениями, а m и n имеют значения размеров матрицы. Таким образом, условие $m \geq i \ \&\& \ n \geq j$ выполняется при входе в цикл.

Сохранение в итерации: на каждой итерации цикла либо увеличиваются значения i и j , либо изменяются значения m и n . Однако, выражение $m \geq i \ \&\& \ n \geq j$ остается неизменным. Если оно было истинным на предыдущей итерации, оно останется истинным и на следующей.

Завершение: цикл завершится, когда i станет равно m или j станет равно n , что означает, что мы завершили прохождение всех элементов.

Следовательно, $m \geq i \ \&\& \ n \geq j$ является корректным инвариантом для цикла `while` в данной функции.

Для определения вычислительной сложности алгоритма воспользуемся табл. 5, а также введем переменную $k = \min(m, n)$.

Таблица 5 – Вычислительная сложность алгоритма

Оператор	Кол-во выполнений оператора в строке	
	в лучшем случае	в худшем случае
<code>int i = 0, j = 0;</code>	2	2
<code>while (i < m && j < n) {</code>	1	k
<code> for (int col = j; col < n;</code> <code> col++) {</code>	2	$k * n + 1$
<code> cout << x[i][col] << " "; }</code>	1	$k * n$
<code> i++;</code>	1	k
<code> for (int row = i; row < m;</code> <code> row++) {</code>	1	$k * m + 1$
<code> cout << x[row][n-1] << " ";</code> <code> }</code>	0	$k * m$
<code> n--;</code>	1	k
<code>if (i < m) {</code>	1	k

Оператор	Кол-во выполнений оператора в строке	
	в лучшем случае	в худшем случае
<code>for (int col = n - 1; col >= j; col--) {</code>	0	$k * m + 1$
<code>cout << x[m - 1][col] << "</code> <code>"; }</code>	0	$k * m$
<code>m--; }</code>	0	k
<code>if (j < n) {</code>	1	k
<code>for (int row = m - 1; row >=</code> <code>i; row--) {</code>	0	$k * m + 1$
<code>cout << x[row][j] << " "; }</code>	0	$k * m$
<code>j++; }</code>	0	k

Таким образом, функция роста данного алгоритма в худшем случае: $T(n) = 4kn + 4km + 11k + 6$, а в лучшем случае $T(n) = 11$.

2.2.2 Реализация алгоритма в виде функции

Код на языке программирования C++, реализующий алгоритм в виде функции представлен на рис. 15.

```

void Obhod0(int** x, int m, int n) {
    int i = 0, j = 0; // 2
    while (i < m && j < n) { // min(m, n)
        // Обход верхней строки
        for (int col = j; col < n; col++) { // min(m, n) * (n+1) + 1
            cout << x[i][col] << " "; // n * min(m, n)
        }
        i++; // min(m, n)
        // Обход последнего столбца
        for (int row = i; row < m; row++) { // (m+1) * min(m, n) + 1
            cout << x[row][n - 1] << " "; // m * min(m, n)
        }
        n--; // min(m, n)
        // Обход нижней строки
        if (i < m) { // min(m, n)
            for (int col = n - 1; col >= j; col--) { // (n+1) * min(m, n) + 1
                cout << x[m - 1][col] << " "; // n * min(m, n)
            }
            m--; // min(m, n)
        }
        // Обход первого столбца
        if (j < n) { // min(m, n)
            for (int row = m - 1; row >= i; row--) { // (m+1) * min(m, n) + 1
                cout << x[row][j] << " "; // m * min(m, n)
            }
            j++; // min(m, n)
        }
        // Пусть k=min(m, n)
        // Тогда:
        // Функция роста T(n) = 2k*(n+1) + 2k*(m+1) + 2k*m + 2k*n + 7k + 6
        // T(n) = 4kn + 4km + 11k + 6
    }
}

```

Рисунок 15 – Реализация алгоритма в виде функции

2.2.3 Тестирование алгоритма

Результаты тестирования алгоритма на матрице 3x3 со значениями, заданными вручную представлены на рис. 16 и 17.

```

// Введем значения от 1 до 9 включительно, получим матрицу:
// 1 2 3
// 4 5 6
// 7 8 9
// Ожидаемый результат работы алгоритма:
// 1 2 3 6 9 8 7 4 5

```

Рисунок 16 – Ожидаемый результат тестирования алгоритма

```

Введите через enter 9 значений элементов матрицы x
1
2
3
4
5
6
7
8
9
1 2 3 6 9 8 7 4 5

```

Рисунок 17 – Полученный результат работы алгоритма

Результат тестирования алгоритма на матрице 7x5 со случайными значениями элементов представлен на рис. 18.

```
Введите кол-во строк:
7
Введите кол-во столбцов:
5
80 32 18 36 13
50 25 5 64 97
77 20 77 83 79
16 37 25 86 3
47 21 40 83 24
73 1 23 90 4
75 51 47 4 34
80 32 18 36 13 97 79 3 24 4 34 4 47 51 75 73 47 16 77 50 25 5 64 83 86 83 90 23 1 21 37 20 77 25 40
```

Рисунок 18 – Результат тестирования алгоритма на случайных значениях

Таким образом, можно сделать вывод о том, что алгоритм работает корректно.

2.2.4 Практическая оценка сложности алгоритма для больших n

Практическая оценка сложности алгоритма в лучшем случае представлена на рис. 18.

```
Введите кол-во строк:
1
Введите кол-во столбцов:
1
33
33
Количество сравнений и перемещений данных: 11
Затраченное время: 0.001
```

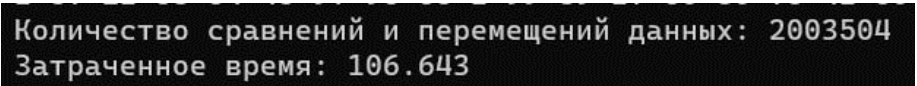
Рисунок 18 – Лучший случай

Практическая оценка сложности алгоритма в среднем случае представлена на рис. 19.

```
Введите кол-во строк:
5
Введите кол-во столбцов:
5
10 10 73 40 13
22 22 11 12 19
30 45 77 46 51
24 74 77 9 88
34 96 34 85 98
10 10 73 40 13 19 51 88 98 85 34 96 34 24 30 22 22 11 12 46 9 77 74 45 77
Количество сравнений и перемещений данных: 73
Затраченное время: 0.004
```

Рисунок 19 – Средний случай

Практическая оценка сложности алгоритма при больших значениях n и m представлена на рис. 20.



```
Количество сравнений и перемещений данных: 2003504
Затраченное время: 106.643
```

Рисунок 20 – Случай, когда $n=m=1000$

2.2.5 Оценка ёмкостной сложности алгоритма

В качестве исходных данных, с которыми работает алгоритм, на вход поступает $n * m$ элементов матрицы x , следовательно ёмкостная сложность алгоритма равна $n * m$.

3 ВЫВОДЫ

По завершении практики были разработаны и протестированы различные алгоритмы для двух заданий: удаление элементов из массива и вывод элементов матрицы по спирали.

Для каждого алгоритма были выведены инварианты, теоретически рассчитаны сложности алгоритмов и оценены практически.

В ходе выполнения первого задания также, на основе анализа, был сделан вывод о том, что:

1. Первому алгоритму нужна большая вычислительная мощность для удаления объектов из массива, нежели второму;
2. При отсутствии удаляемых элементов, первый алгоритм справляется лучше для малых значений n . А также время, затраченное на выполнение алгоритмов, либо равное, либо лучшее для второго алгоритма.
3. При увеличении значений n второй алгоритм справляется быстрее первого, требуя меньшего объема вычислительной мощности, следовательно второй алгоритм эффективнее первого.

4 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ

1. Сартаков М.В., ПР-1.1 (Теоретическая сложность алгоритма)М., МИРЭА — Российский технологический университет – 12 с. - URL: https://online-edu.mirea.ru/pluginfile.php?file=%2F1042738%2Fmod_assign%2Fintroattachment%2F0%2FПР1.1%20%28Теоретическая%20сложность%20алгоритма%29.pdf&forcedownload=1 (дата обращения: 15.02.2024). - Режим доступа: Электронно-облачная система – Cloud MIREA РТУ МИРЭА. - Текст: электронный.
2. Рысин М.Л., Сартаков М.В., Туманова М.Б., Введение в структуры и алгоритмы обработки данных. Ч. 1 - учебное пособие, 2022, МИРЭА – Российский технологический университет. – 2022, 109с. – URL: <file:///C:/Users/borga/Downloads/Рысин%20М.Л.%20и%20др.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Ч.%201%20-%20учебное%20пособие,%202022.pdf> (дата обращения: 15.02.2024). – Режим доступа: Электронно-облачная система – Cloud MIREA РТУ МИРЭА. - Текст: электронный.