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: Величко В.Д.

Группа: ИКБО-74-23

Москва – 2024

СОДЕРЖАНИЕ

1 ЦЕЛЬ

Приобретение практических навыков:

- Эмпирическому определению вычислительной сложности алгоритмов на теоретическом и практическом уровнях;
- Выбору эффективного алгоритма решения вычислительной задачи из нескольких.
- Разработка собственного алгоритма в соответствии с задачей.

2 ЗАДАНИЕ 1

Формулировка задания: выбрать эффективный алгоритм вычислительной задачи из двух предложенных, используя теоретическую и практическую оценку вычислительной сложности каждого из алгоритмов, а также его емкостную 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.

2.1 Первый алгоритм

2.1.1 Описание математической модели

С помощью цикла идем по массиву с первого элемента до n-ного, где n – размер массива. Если текущий элемент равен заданному значению, то смещаем все следующие значения в массиве на 1 позицию влево, тем самым заменяя и удаляя требуемый элемент. Переменную n, отвечающую за размер массива, уменьшаем на 1.

2.1.2 Блок-схема алгоритма, доказательство корректности циклов

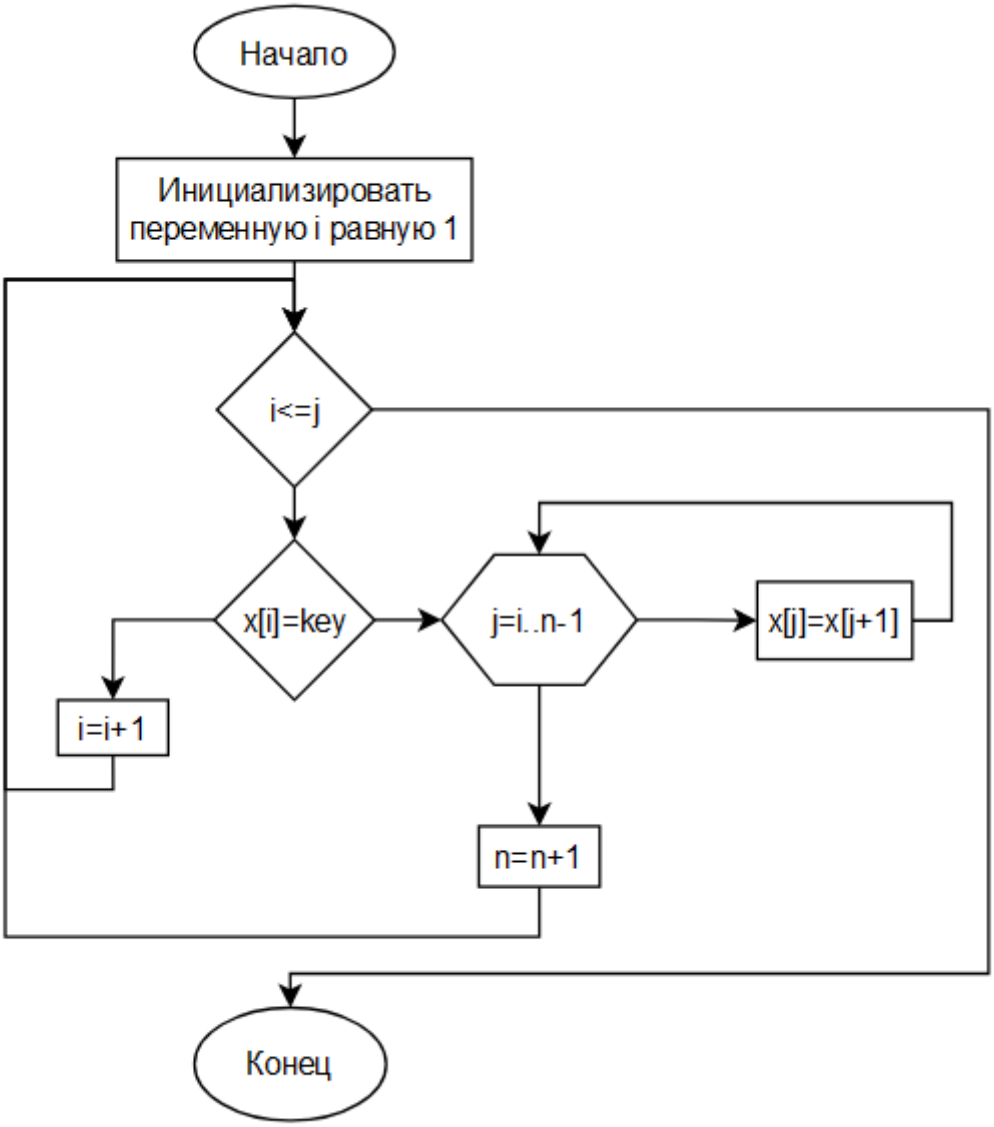

Рисунок 1 – Блок-схема для первого алгоритма

Касательно доказательства корректности циклов, можно рассмотреть инвариантные условия.

Например:

1. Для цикла while: Инвариант - до текущего индекса i включительно все элементы массива отличны от искомого элемента. Инвариант сохраняется на каждой итерации цикла.

2. Для цикла for (при нахождении искомого элемента): Инвариант - сдвиг происходит корректно, остальные элементы не меняются местами. Инвариант также поддерживается на каждой итерации.

2.1.3 Определение вычислительной сложности алгоритма

Номер строки	Алгоритм, записанный на псевдокоде	Количество повторений действия в зависимости от объема входных данных n
1	delFirstMetod(x,n,key){	
2	i←1	1
3	while (i<=n) **do**	n+1
4	if x[i]=key then	n
5	for j←i to n-1 do	$\sum (n-1)\,(n-2)\,\dots\,2\,1$
6	x[j] ←x[j+1]	$\sum (n-1)\,(n-2)\,\dots\,2\,1$
7	od	
8	n←n-1	n
9	else	
10	i←i+1	
11	endif	
12	**od**	
13	}	

Количество повторений действия в строке 6 представляет собой арифметическую прогрессию. Найдем ее сумму $\sum (n-1)\,(n-2)\,\dots\,2\,1 = \frac{n-1+1}{2} * (n-1) = 0.5n^2 - 0.5n.$

Тогда общая вычислительная сложность алгоритма в худшем случае определяется функцией $T(n) = n^2 + 2n + 2$. То есть алгоритм имеет квадратичный порядок роста времени вычисления.

В лучшем случае, когда ни один элемент удалять не нужно, сложность определяется функцией T(n)=2n+2.

2.1.4 Реализация алгоритма на C++

```cpp
#include <iostream>
#include <random>

using namespace std;
/*объявляет функцию с именем delFirstMethod,
которая принимает три аргумента:
указатель на массив целых чисел,
ссылку на целое число (количество элементов в массиве)
и целое число (искомый элемент).
Функция возвращает значение типа void*/
void delFirstMethod(int* array, int& n, int key)
{
    setlocale(LC_ALL, "RUS");
    int count = 0;
    /*Это будет индекс текущего элемента массива.*/
    int i = 0;
    /*увеличивает значение переменной на 1*/
    /*Инкремент*/
    count++;
```

Рисунок 2.1 – Первый алгоритм

```
20
21      /*запускает цикл, который продолжается,
22      пока значение переменной i меньше значения переменной n*/
23      while (i < n)
24      {
25          /*Инкремент*/
26          count++;
27          /*проверяет, равен ли текущий элемент массива
28          (доступный через array с индексом i)
29          искомому значению key*/
30          if (array[i] == key)
31          {
32              /*запускает цикл с переменной j,
33              которая начинается с текущего индекса i
34              и продолжается до предпоследнего элемента массива
35              (n-1), увеличивая j на 1 каждый раз*/
36              for (int j = i; j < n - 1; j++)
37              {
38                  /*увеличивает значение на 2,
39                  чтобы учесть две операции в этом цикле:
40                  увеличение j и доступ к элементу массива.*/
41                  count += 2;
42                  /*меняет местами текущий элемент массива и следующий элемент,
43                  используя индексы j и j+1*/
44                  array[j] = array[j + 1];
45              }
46              /*декремент,
47              чтобы удалить только что перемещенный элемент из массива.*/
48              n--;
49              count++;
50          }
51          /*если условие if не выполняется,
52          то увеличивается значение i на 1 и значение operation_number на 1
53          для учёта этих операций.*/
54          else
55          {
56              i++;
57              count++;
58          }
59      }
60
61      count++;
62      cout << "Операций: " << count << endl;
63  }
```

Рисунок 2.2 – Первый алгоритм

```
64   /*Главная функция программы*/
65   int main()
66   {
67       setlocale(LC_ALL, "RUS");
68       int n = 10; //размер
69       int key = 0; //что убрать
70       cout << "Введите число, которое нужно удалить: ";
71       cin >> key;
72
73       cout << "Исходный массив: ";
74       int* array = new int[n]; // Массив
75       mt19937 gen(random_device{}());
76       uniform_int_distribution<int> dist(1, 10);
77       for (int j = 0; j < n; j++) {
78           array[j] = dist(gen); // Заполнение массива случайными значениями от 1 до 10
79           cout << array[j] << " ";
80       }
81       cout << endl;
82       delFirstMethod(array, n, key);
83
84       cout << "Измененный массив: ";
85       for (int i = 0; i < n; i++) {
86           cout << array[i] << " ";
87       }
88       cout << endl;
89       //размер конечного массива
90       cout << "n = " << n << endl;
91
92       return 0;
93   }
```

Рисунок 2.3 – Первый алгоритм

2.1.5 Тестирование

```
Введите число, которое нужно удалить: 7
Исходный массив: 7 8 3 1 6 8 7 5 8 9
Операций: 46
Измененный массив: 8 3 1 6 8 5 8 9
n = 8
```

Рисунок 3.1 - Тестирование при 10 элементах, key=7. Случайная ситуация.

```
Введите число, которое нужно удалить: 1
Исходный массив: 8 10 5 5 1 9 1 4 1 9 7 2 7 6 5 6 9 3 6 3 5 4 3 6 1 9
10 6 8 10 10 9 3 5 2 5 2 4 2 3 2 9 1 3 2 10 7 9 3 5 6 10 6 1 6 5 4 2 2
 7 8 2 10 1 3 5 7 10 3 6 5 1 1 5 7 1 6 7 7 4 5 7 6 7 5 3 7 10 1 9 6 4
4 3 2 2 2 6 9 5
Операций: 1368
Измененный массив: 8 10 5 5 9 4 9 7 2 7 6 5 6 9 3 6 3 5 4 3 6 9 10 6 8
 10 10 9 3 5 2 5 2 4 2 3 2 9 3 2 10 7 9 3 5 6 10 6 6 5 4 2 2 7 8 2 10
3 5 7 10 3 6 5 5 7 6 7 7 4 5 7 6 7 5 3 7 10 9 6 4 4 3 2 2 2 6 9 5
n = 89
```

Рисунок 3.2 - Тестирование при 100 элементах, key=1. Случайная ситуация.

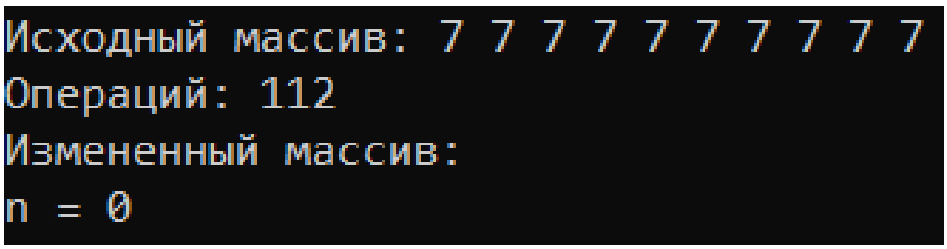

Рисунок 3.3 - Тестирование при 10 элементах и удаления всех.

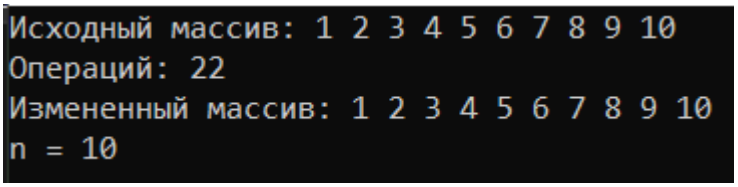

Рисунок 3.4 - Тестирование при 10 элементах и ничего не удаляется

2.2 Второй алгоритм

2.2.1 Описание математической модели

С помощью цикла идем по массиву с первого элемента до n-ного, где n – размер массива. В переменной i хранится номер рассматриваемого элемента исходного массива, в переменной j хранится номер размещаемого в данный момент элемента в конечном массиве. В j-тый элемент постоянно помещаем i-тый, но увеличиваем j на 1 (то есть размещаем теперь в следующий элемент конечного массива) только если текущий не равен искомому значению.

2.2.2 Блок-схема алгоритма, доказательство корректности циклов

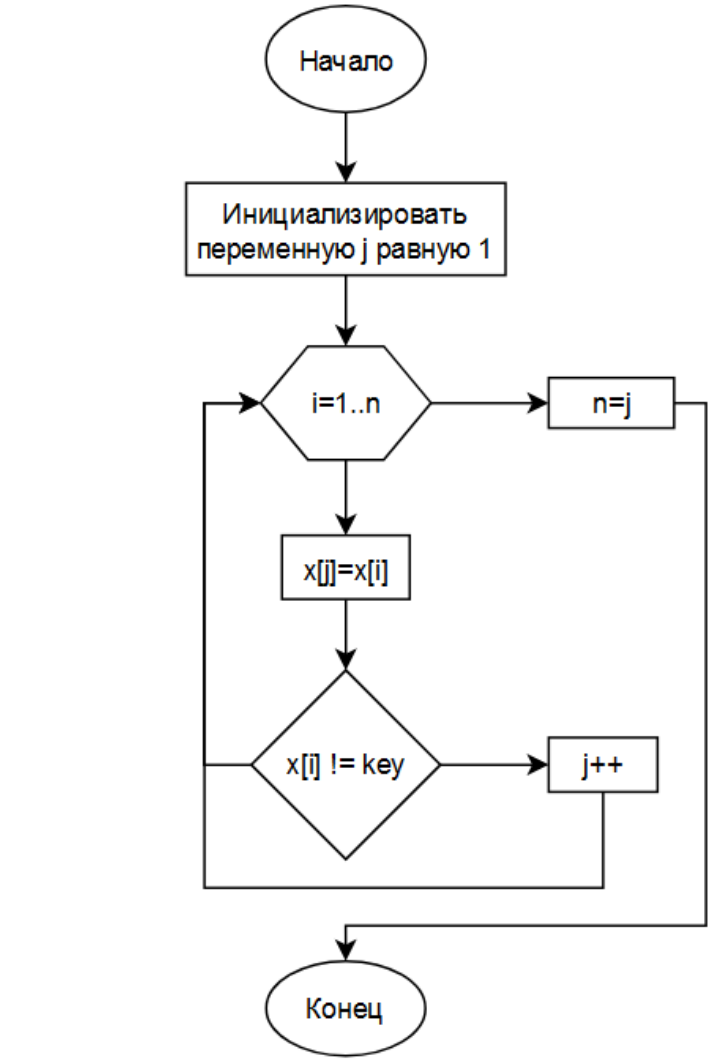

Рисунок 4 – Блок-схема второго алгоритма

Доказательство конечности цикла: при каждой итерации область неопределённости сужается на 1 элемент. До начала цикла не просмотрено n элементов, после первой итерации n-1, после второй n-2 и так далее. После n-ной итерации будет не просмотрено n-n=0 элементов, следовательно цикл завершится.

Таким образом, цикл алгоритма корректен, а значит и сам алгоритм, корректен.

2.2.3 Определение вычислительной сложности алгоритма

Номер строки	Алгоритм, записанный на псевдокоде	Количество повторений действия в зависимости от объема входных данных n
1	delOtherMetod(x,n,key){	
2	j←1	1
3	for i←1 to n **do**	n+1
4	x[j]=x[i];	n
5	if x[i]!=key then	n
6	j++	n
7	endif	
8	**od**	
9	**n**←j	1
10	}	

Общая вычислительная сложность алгоритма в худшем случае определяется функцией $T(n) = 4n + 3$. То есть алгоритм имеет линейны1 порядок роста времени вычисления.

В лучшем случае, когда все нужно удалять, сложность определяется функцией T(n)=2n+3.

2.2.4 Реализация алгоритма на языке C++

```cpp
#include <iostream>
/* используемый для генерации случайных чисел*/
#include <random>

using namespace std;
void delOtherMethod(int array[], int& n, int key) {
    int j = 0;
    int count = 0;
    for (int i = 0; i < n; i++)
    {
        /*опирование элемента массива с индексом i
        в элемент с индексом j*/
        array[j] = array[i];
        /* условие, проверяющее, равен ли текущий элемент массива
        ключу. Если нет, то элемент копируется
        в следующую позициюмассива*/
        if (array[i] != key)
        {
            j++;//инкремент
            count++;
        }
    }
    /*обновление размера массива после удаления элементов*/
    n = j;
    count++;
    cout << "Операций: " << count << endl;
}
```

Рисунок 5.1 – Второй алгоритм

```cpp

/*Главная функция программы*/
int main()
{
    setlocale(LC_ALL, "RUS");
    int n = 100; //размер
    int key = 0; //что убрать
    cout << "Введите число, которое нужно удалить: ";
    cin >> key;

    cout << "Исходный массив: ";
    int* array = new int[n]; // Массив
    mt19937 gen(random_device{}());
    uniform_int_distribution<int> dist(1, 10);
    for (int j = 0; j < n; j++) {
        array[j] = dist(gen); // Заполнение массива случайными значениями от 1 до 10
        cout << array[j] << " ";
    }
    cout << endl;
    delOtherMethod(array, n, key);

    cout << "Измененный массив: ";
    for (int i = 0; i < n; i++) {
        cout << array[i] << " ";
    }
    cout << endl;
    //размер конечного массива
    cout << "n = " << n << endl;

    return 0;
}

```

Рисунок 5.2 – Второй алгоритм

2.2.5 Тестирование

```
Введите число, которое нужно удалить: 5
Исходный массив: 9 6 7 8 8 7 2 1 2 6
Операций: 11
Измененный массив: 9 6 7 8 8 7 2 1 2 6
n = 10
```

Рисунок 6.1 - Тестирование при 10 элементах. Случайная ситуация.

```
Введите число, которое нужно удалить: 9
Исходный массив: 1 7 1 7 8 9 9 1 1 8 5 5 10 7 5 3 1 2 5 9 2 10 5 3 8 9 6 10 6 3 3
3 2 3 1 6 8 8 6 8 4 5 8 6 5 10 6 10 3 4 6 2 2 8 1 6 3 1 4 5 4 7 3 3 8 8 2 10 5 6 1
0 6 7 2 10 1 10 4 5 5 7 4 1 3 4 1 7 10 9 8 1 2 6 3 4 6 9 5 2 4
Операций: 95
Измененный массив: 1 7 1 7 8 1 1 8 5 5 10 7 5 3 1 2 5 2 10 5 3 8 6 10 6 3 3 3 2 3
1 6 8 8 6 8 4 5 8 6 5 10 6 10 3 4 6 2 2 8 1 6 3 1 4 5 4 7 3 3 8 8 2 10 5 6 10 6 7
2 10 1 10 4 5 5 7 4 1 3 4 1 7 10 8 1 2 6 3 4 6 5 2 4
n = 94
```

Рисунок 6.2 - Тестирование при 100 элементах, key=7. Случайная ситуация.

```
Исходный массив: 1 2 3 4 5 6 7 8 9 10
Операций: 11
Измененный массив: 1 2 3 4 5 6 7 8 9 10
n = 10
```

Рисунок 6.3 - Тестирование при 10 элементах и ничего не удаляется

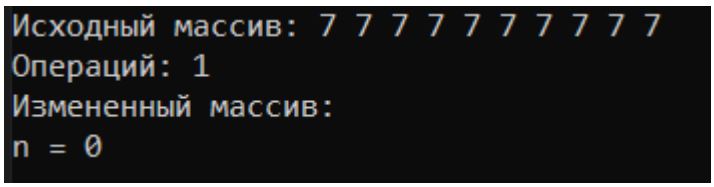

Рисунок 6.4 - Тестирование при 10 элементах и удаления всех.

2.3 Вывод по первому заданию

Основываясь на полученных результатах, можно сделать вывод, что второй алгоритм и в среднем и худшем случае требует намного меньше действий для выполнения.

Поскольку результаты теоретического расчета сложности практически совпадают с экспериментально полученными, можно заявить, что расчеты выполнены верно.

3 ЗАДАНИЕ 2

3.1 Описание математической модели

Математическая модель задачи представляет собой матрицу чисел MxN, с границами t, b, l, r. Цикл продолжается пока t <= b и l <= r, в каждой итерации выводятся элементы строки сверху слева направо и столбца справа сверху вниз. Границы t и r изменяются после каждой итерации.

3.2 Блок-схема алгоритма, доказательство корректности циклов

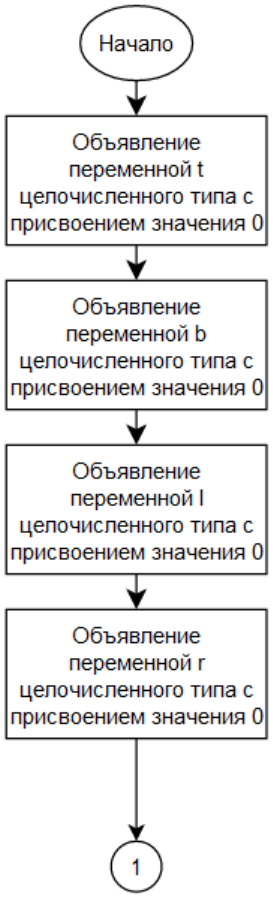

Рисунок 7.1 – Первая часть Блок-схемы

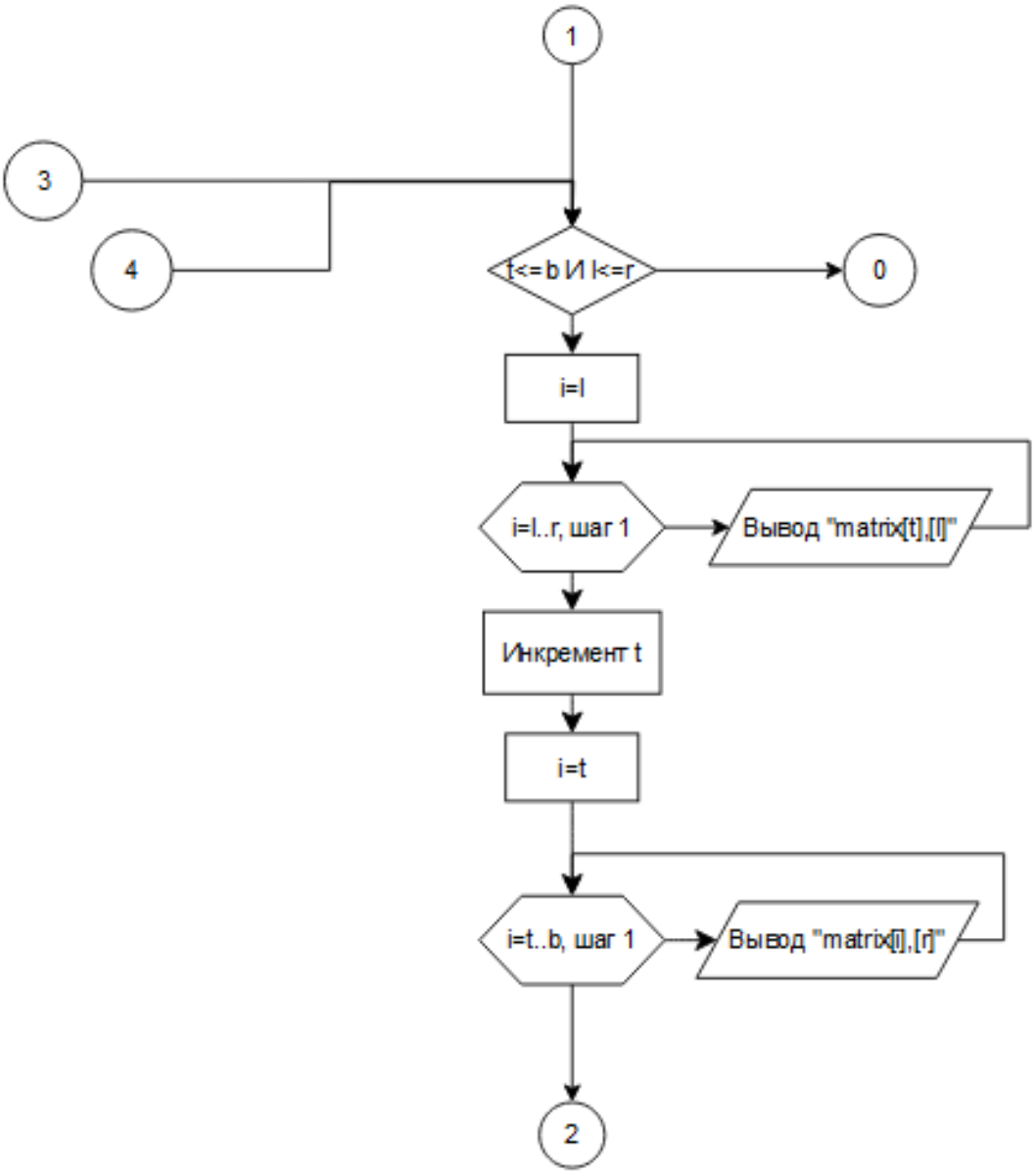

Рисунок 7.2 – Вторая часть Бок-схемы

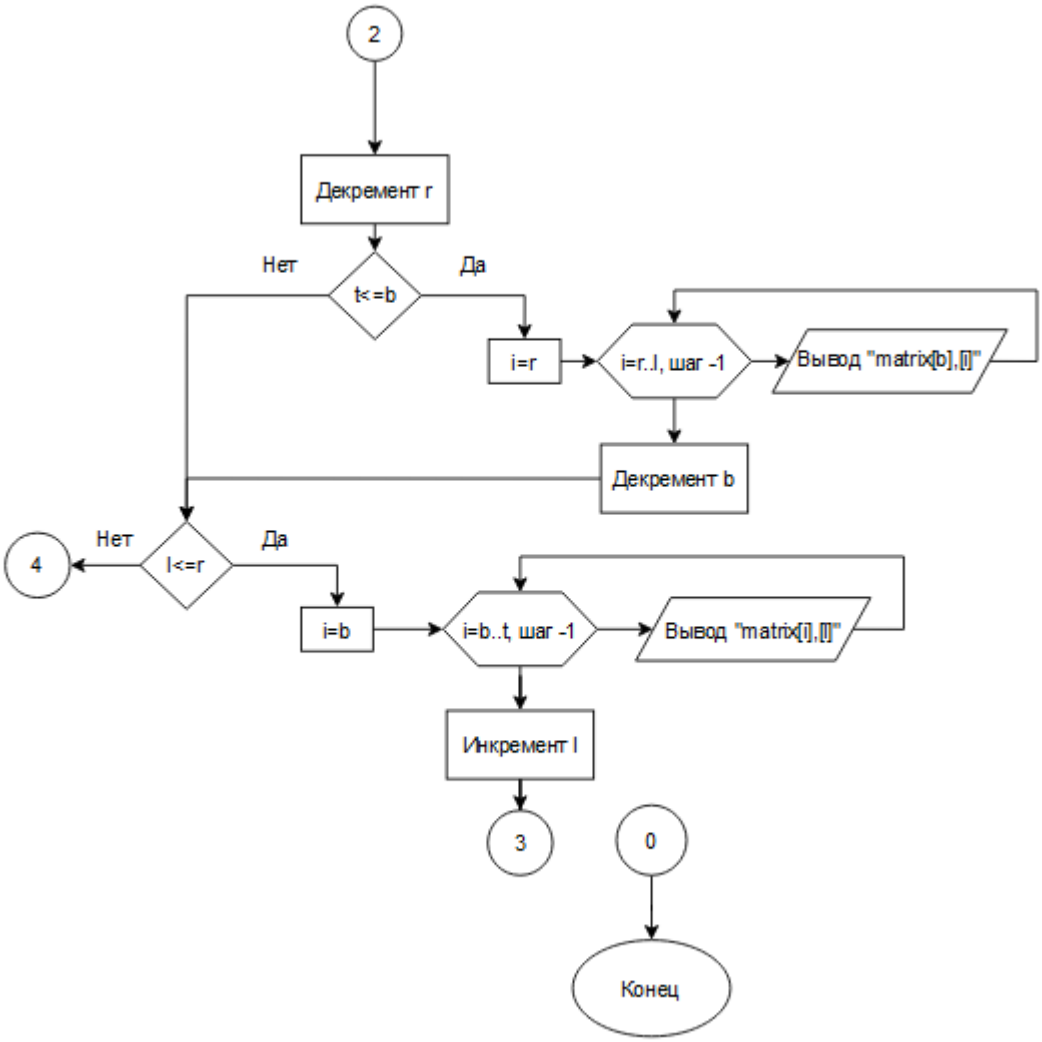

Рисунок 7.3 - Третья Блок-схема

Доказательство конечности циклов: В данной функции используется алгоритм спирали, который заключается в том, что элементы матрицы выводятся по спирали, начиная с верхнего левого угла и двигаясь вниз, затем вправо и так далее. Доказательство конечности циклов основывается на том, что размер матрицы конечен, а в каждом шаге цикла хотя бы один индекс изменяется. Поскольку индексы не могут превышать размер матрицы, то количество итераций цикла конечно и зависит от размера матрицы.

3.3 Определение вычислительной сложности алгоритма

Номер строки	Оператор	Количество повторений действия в зависимости от объема входных данных n
1	Void spiral	
2	t = 0	1
3	b = M - 1	1
4	l = 0	1
5	r = N - 1	1
6	while (t <= b && l <= r)	n
7	`for (int i = l; i <= r; i++)`	n(n+1)=n^2+n
8	{	
9	`cout << matrix[t][i] << " ";`	n-1
10	}	
11	`t++;`	n
12	`p++;`	
13	`for (int i = t; i <= b; i++)`	n(n+1)
14	{	
15	`cout << matrix[i][r] << " ";`	n-1
16	}	
17	`r--;`	n
18	`p++;`	
19		n-1
20	if (t <= b)	n
21	`for (int i = r; i >= l; i--)`	n(n+1)
22	{	
23	`cout << matrix[b][i] << " ";`	n-1
24	}	
25	`b--;`	n
26	`p++;`	
27		n-1
28	if (l <= r)	n
29	`for (int i = b; i >= t; i--)`	n(n+1)

30	{	
31	cout << matrix[i][l] << " ";	n-1
32	}	
33	l++;	n
34	p++;	
35		n-1
36		1

Вычислительная сложность алгоритма в худшем, лучшем, среднем случае не будет отличаться своим значением функции так как при любых значениях элементов матрицы обход по спирали будут выполняться все циклы в программе, в результате чего получается квадратичная функция роста алгоритма

T(n)= 5+3n^2+3n+n-1+n-1+2n = 3n^2+6n+3

3.4 Реализация алгоритма на языке C++

```
1   #include <iostream> /* Подключаем заголовочный файл <iostream>,
2   который позволяет работать с потоками ввода-вывода */
3   using namespace std; /* Это нужно, чтобы не писать std::
4   перед стандартными функциями и объектами */
5
6    //Объявляем макросы для количества строк и столбцов в матрице
7    #define M 4 // Количество строк
8    #define N 4 // Количество столбцов
9
10  //b-bottom-низ
11   //l-left-лево
12   //r-right-право
13   //t-top-верх
14   //matrix-матрица
```

Рисунок 8.1 – Программа на C++

```cpp

    while (t <= b && l <= r) /*Запускаем цикл,
ПОКА верхняя граница(t) не достигнет нижней(b), и(&&) левая(l) не достигнет правой(r) */
    {
        /*Циклы for внутри while обходят матрицу по часовой стрелке,
        выводя элементы в нужном порядке (t, последний столбец, b, первый столбец) */
        // Вывод верхней строки
        for (int i = l; i <= r; i++) //выводит элементы первой строки по порядку
        {
            p++;
            cout << matrix[t][i] << " "; //вверхняя строка i столбец
        }
        t++; //граница увеличивается
        p++;

        // Вывод последнего столбца
        for (int i = t; i <= b; i++) //выводит элементы последнего столбца по порядку
        {
            p++;
            cout << matrix[i][r] << " "; //i строка правый столбец
        }
        r--; // граница уменьшается
        p++;

        // Вывод нижней строки
        if (t <= b)
        {
            for (int i = r; i >= l; i--) /*выводит элементы нижней строки,
                но если левая граница (l) меньше или равна правой границе (r)
                (строка 37 - то самое условие) */
            {
                p++;
                cout << matrix[b][i] << " "; //нижняя строка i столбец
            }
            b--; // граница уменьшается
            p++;
        }
        p++;

        // Вывод первого столбца
        if (l <= r)
        {
            for (int i = b; i >= t; i--)
            {
                p++;
                cout << matrix[i][l] << " "; /*i строка левый столбец
                это все нужно для извлечения соответствующих элементов
                во время обхоба матрицы по часовой стрелке*/
            }
            l++; //граница увеличивается
            p++;
        }
        p++;
    }
    /*после каждого шага внутри цикла while,
    границы либо уменьшаются, либо увеличиваются.
    Это нужно, чтобы переместиться к следующей строке или столбцу внутри матрицы*/
    p++;
    cout << "Операций: " << p << endl;
}
```

Рисунок 8.2 – Программа на C++

```
70
71   //int main() это главная функция программы. Заполняется матрица любыми числаим
72   int main()
73   {
74       setlocale(LC_ALL, "RUS"); //для использования русского языка
75       int matrix[M][N] = { {8, 6, 3, 1}, //первая строка матрицы
76                            {48, -5, 7, 3}, // вторая строка матрицы
77                            {9, 1, 2, 63}, // третья строка матрицы
78                            {13, 21, 72, 12} }; // четвертая строка матрицы
79       /*если захочется поменять кол - во строк или кол - во столбцов,
80       то нужно вернуться на строки 4 и 5 и менять значения*/
81
82       cout << "Обход матрицы по спирали (по часовой стрелке): " << endl;
83       spiral(matrix); // вызов функции spiral для обхода матрицы по часовой стрелке
84
85       return 0; // это чтобы показать успешное завершение программы
86   }
```

Рисунок 8.3 – Программа на C++

3.5 Тестирование

MxN	Вводные данные	Ожидаемый ответ	Ответ
4x4	{{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}}	1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10	1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
3x3	{{8,6,4},{2,5,3},{7,1,9}}	8 6 4 3 9 1 7 2 5	8 6 4 3 9 1 7 2 5
2x3	{{1,2,3},{4,5,6}}	1 2 3 6 5 4	1 2 3 6 5 4
5x2	{{1,6},{2,7},{3,8},{4,9},{5,10}}	1 6 7 8 9 10 5 4 3 2	1 6 7 8 9 10 5 4 3 2

Рисунок 9 – Тестовые данные

```
Обход матрицы по спирали (по часовой стрелке):
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10 Операций: 30
```

Рисунок 10.1 – Тест для 4x4

```
Обход матрицы по спирали (по часовой стрелке):
8 6 4 3 9 1 7 2 5 Операций: 21
```

Рисунок 10.2 – Тест для 3x3

```
Обход матрицы по спирали (по часовой стрелке):
1 2 3 6 5 4 Операций: 14
```

Рисунок 10.3 – Тест для 2x3

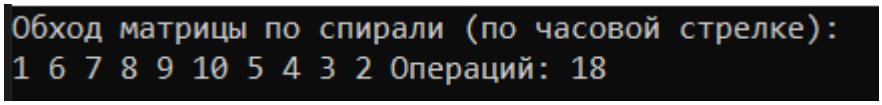

Рисунок 10.4 – Тест для 5x2

Практическая оценка сложности алгоритма обхода матрицы по спирали для больших n будет зависеть от размеров матрицы. Наилучшая, наихудшая и средняя временная сложность алгоритма сохраняется и для различных размеров матриц.

1) Наилучший случай: В лучшем случае, когда матрица представляет собой квадратную матрицу, размером n x n, алгоритм обойдет матрицу за O(n^2) времени. В этом случае, все элементы матрицы будут пройдены без дополнительных условий.

2) Наихудший случай: В худшем случае, когда матрица представляет собой прямоугольную матрицу, размером m x n, где m ≠ n, алгоритм обойдет матрицу за O(mn) времени. В этом случае, алгоритм должен выполнить дополнительные проверки для обхода всех элементов.

3) Средний случай: - В среднем случае, для случайных матриц, также время работы алгоритма будет O(mn).

3.6 Выводы по заданию 2

В ходе работы был разработан алгоритм в соответствии с индивидуальным вариантом, оценена его сложность теоретическим и практическим методами. Основываясь на полученных данных, можно утверждать, что алгоритм по обходу матрицы по спирали (по часовой стрелке) имеет зависимость от размера матрицы.

4 ВЫВОДЫ

В ходе работы отработаны навыки определению:

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

Разработан собственный алгоритм решения задачи и оценена его эффективность. Тестирование подтвердило правильность решения задачи алгоритмом, а также правильность теоретического расчета вычислительной сложности алгоритмов.

5 ЛИТЕРАТУРА

1. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона, 2010.
2. Кнут Д. Искусство программирования. Тома 1-4, 1976-2013.
3. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для про-граммистов и любопытствующих, 2017.
4. Кормен Т.Х. и др. Алгоритмы. Построение и анализ, 2013.
5. Лафоре Р. Структуры данных и алгоритмы в Java. 2-е изд., 2013.
6. Макконнелл Дж. Основы современных алгоритмов. Активный обучающий метод. 3-е доп. изд., 2018.
7. Скиена С. Алгоритмы. Руководство по разработке, 2011.
8. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2017.
9. Гасфилд Д. Строки, деревья и последовательности в алгоритмах. Инфор-матика и вычислительная биология, 2003.

По языку C++:

10. Страуструп Б. Программирование. Принципы и практика с использовани-ем C++. 2-е изд., 2016.
11. Павловская Т.А. C/C++. Программирование на языке высокого уровня, 2003.
12. Прата С. Язык программирования C++. Лекции и упражнения. - 6-е изд., 2012.
13. Седжвик Р. Фундаментальные алгоритмы на C++, 2001-2002
14. Хортон А. Visual C++ 2010. Полный курс, 2011.
15. Шилдт Г. Полный справочник по C++. 4-е изд., 2006.