МИНОРБНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Отчёт по выполнению практического задания № 1.1
Тема:
«Оценка вычислительной сложности алгоритма»
Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Фамилия И.О.
Фамилия И.О.

Группа: АААА-00-00
Номер группы

Москва 2023

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ 4
1.1 Задание 1 4
1.2 Задание 2. Индивидуальная задача 5
2 РЕШЕНИЕ ЗАДАЧИ 1 ПО АЛГОРИТМУ 1 6
2.1 Формулировка задачи 6
2.1 Математическая модель решения 6
2.3 Реализация алгоритма в виде функции и отладка на массиве при n=10, n=100 7
2.4 Реализация функций заполнения массива датчиком случайных чисел, вывод массива на экран монитора 9
2.5 Тестирования алгоритма в разных ситуациях 10
2.6 Результаты тестирования 10
2.7 Оценка ёмкостной сложности алгоритма 11
3 РЕШЕНИЕ ЗАДАЧИ 1 ПО АЛГОРИТМУ 2 12
3.1 Формулировка задачи 12
3.2 Математическая модель решения 12
3.3 Реализация алгоритма в виде функции и отладка на массиве при n=10, n=100 13
3.4 Тестирование алгоритма в разных ситуациях 15
3.5 Результаты тестирования 15
3.6 Оценка ёмкостной сложности алгоритма 16
4 ВЫВОД ПО ЗАДАНИЮ 1 17
5 ЗАДАНИЕ 2 18
5.1 Формулировка задачи 18
5.2 Математическая модель решения 18
5.3 Реализация алгоритма 20
5.4 Тестирование алгоритма 21
5.5 Оценка ёмкостной сложности алгоритма 22

5.6 Оценка ёмкостной сложности алгоритма..... .23
6 ВЫВОДЫ..... .24

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Задание 1.

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

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. Необходимо реализовать два представленных алгоритма, оценить их вычислительную сложность теоретически и практически и сделать вывод об их эффективности.

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 - Алгоритм 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
	}

1.2 Задание 2. Индивидуальная задача

17 Дана целочисленная прямоугольная матрица размером n*m. Определить номер строки, в которой находится самая длинная серия одинаковых элементов. Пример с серией из четырех чисел 3: 1 2 3 3 3 3 5.

2 РЕШЕНИЕ ЗАДАЧИ 1 ПО АЛГОРИТМУ 1

2.1 Формулировка задачи

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

2.1 Математическая модель решения

Производится перебор элементов. Если конкретный элемент равен удаляемому key, то необходимо удалить этот элемент, сдвинув в цикле часть массива, идущую после удаленного элемента, влево. После сдвига длину массива необходимо уменьшить.

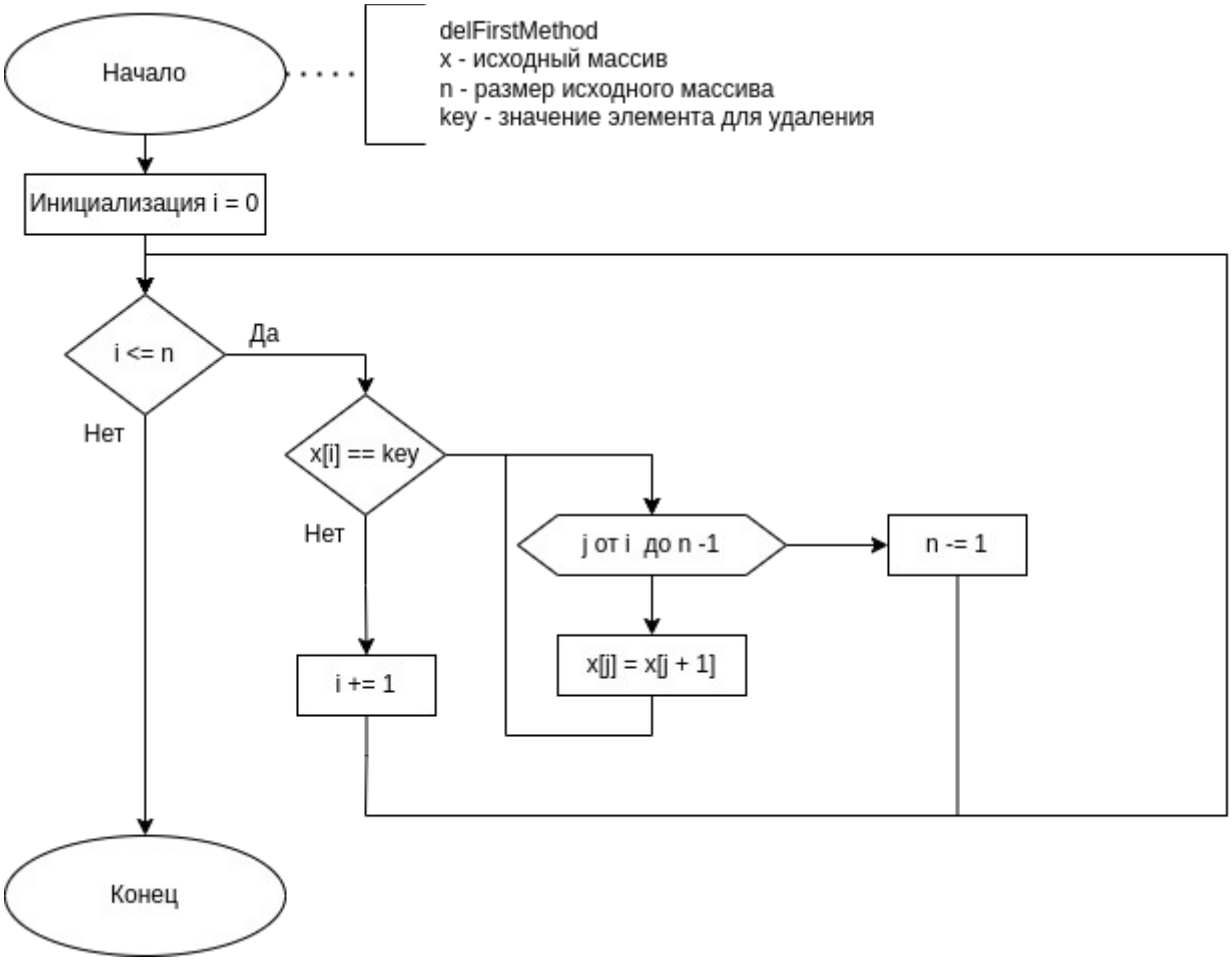

Рисунок 1 - Блок схема алгоритма 1

Инвариант цикла: с каждой итерацией удаляется (или не удаляется) элемент массива равный (или не равный) значению *key* и уменьшается (или не уменьшается) длина массива на единицу. Следовательно, на *i*-ой итерации среди элементов с индексами с 0 по *i*-ый нет элементов со значением *key*.

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; j++)	0	$n^2 + n$
x[j] = x[j + 1];	0	n
n--;	0	n
i++;	n	0
return n;	1	1

Согласно табл.2:

В лучшем случае: $T(n) = 3n + 3 \Rightarrow T(n) = n$. Линейная зависимость.

В худшем случае: $T(n) = n^2 + 5n + 3 \Rightarrow T(n) = n^2$. Квадратичная зависимость.

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

Реализация алгоритма представлена на рис.2

```
int delFirstMethod(int *x, int n, int key)
{
    int i = 0;
    int op_count = 2; // инициализация переменной i и сравнение в цикле
    while (i < n)
    {
        op_count++; // условие цикла
        op_count++; // сравнение в условном операторе
        if (x[i] == key)
        {
            op_count++; // инициализация переменной j
            for (int j = i; j < n; j++)
            {
                op_count++; // сравнение в цикле
                x[j] = x[j + 1];
                op_count++; // перемещение элементов
            }
            n -= 1;
            op_count++; // уменьшение n
        }
        else
        {
            i += 1;
            op_count++; // увеличение i
        }
    }

    op_count++; // возврат n
    cout << "delFirstMethod кол-во операций: " << op_count << endl;

    return n;
}
```

Рисунок 2 - Реализация первого алгоритма на языке C++

На рис.3 показаны результаты отладки функции на массиве из 10 элементов. Мы видим, что было выполнено 55 операций, и все элементы, равные 2, были удалены.

```
Тестирование для n = 10, key = 2
3 2 4 8 7 8 8 3 2 2
delFirstMethod кол-во операций: 55
n = 7: 3 4 8 7 8 8 3
```

Рисунок 3 - Тестирование первого алгоритма для n = 10

На рис.4 показаны результаты отладки функции на массиве из 100 элементов. Мы видим, что было выполнено 1049 операций, и все элементы, равные 2, были удалены.

```
Тестирование для n = 100, key = 2
n = 100: 8 3 8 8 6 7 4 3 6 8 3 1 1 9 2 4 8 1 1 1 8 4 1 1 6 6 8 1 3 4 9 8 5 1 6 1 1 9 9 4 3 1 2 9 2 2 9 8 4 5 2 5
9 7 3 6 8 5 5 3 7 2 9 4 6 2 4 3 8 3 7 1 2 7 1 5 7 8 6 5 4 8 1 6 8 4 1 5 8 1 4 6 6 5 8 7 9 2 2 6

delFirstMethod кол-во операций: 1049
n = 90: 8 3 8 8 6 7 4 3 6 8 3 1 1 9 4 8 1 1 1 8 4 1 1 6 6 8 1 3 4 9 8 5 1 6 1 1 9 9 4 3 1 9 9 8 4 5 5 9 7 3 6 8 5
 5 3 7 9 4 6 4 3 8 3 7 1 7 1 5 7 8 6 5 4 8 1 6 8 4 1 5 8 1 4 6 6 5 8 7 9 6
```

Рисунок 4 - Тестирование первого алгоритма для n = 100

2.4 Реализация функций заполнения массива датчиком случайных чисел, вывод массива на экран монитора.

На рис. 5 представлен код вывода массива на экран монитора. На рис. 6 представлен код функции заполнения массива датчиком случайных чисел.

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

Рисунок 5 - Функция вывода массива на экран

```
int rnd()
{
    random_device dev;
    mt19937 rng(dev());
    uniform_int_distribution<mt19937::result_type> d(1, 9);

    return d(rng);
}

void genRandomArray(int *x, int n)
{
    for (int i = 0; i < n; i++)
    {
        x[i] = rnd();
    }
}
```

Рисунок 6 - Функция генерации случайного массива

2.5 Тестирования алгоритма в разных ситуациях

Тестирования алгоритма в разных ситуация представлены на рис. 7.

Мы видим, что лучшим случаем является ситуация неудаления ни одного элемента, ей соответствует 33 операции. Худшим случаем является ситуация удаления всех элементов, которой соответствует 153 операции. В случае заполнения массива случайными числами количество операций было равно 38.

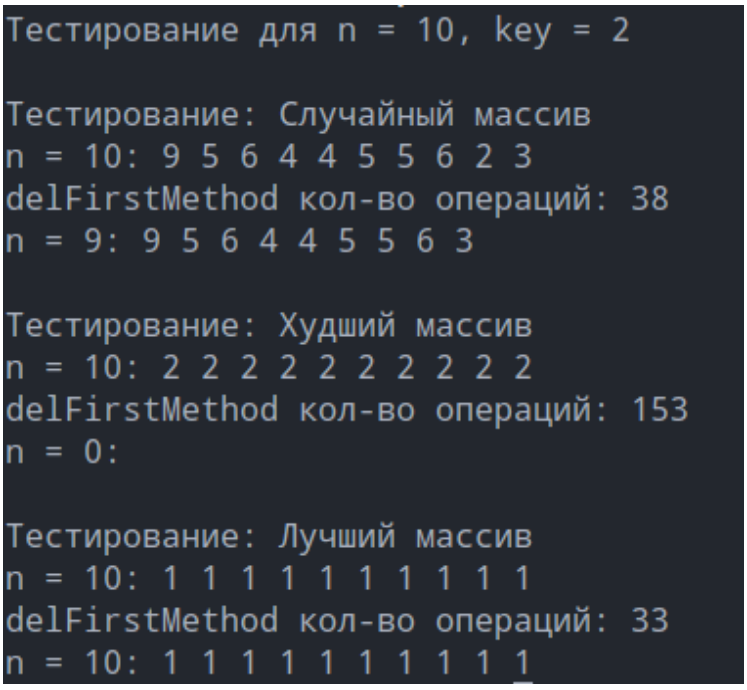

Рисунок 7 - Тестирование первого алгоритма при различных ситуациях

2.6 Результаты тестирования

По результатам тестирования составим таблицу 4 и сравним результаты тестирования с предсказанными теоретическими результатами. Теоретические расчеты приведены как значение T(n) для n=10 для каждого случая.

Таблица 3 - Сравнение результатов тестирования алгоритма

Ситуация	Теоретический расчёт	Результат тестирования
Удаление всех элементов	153	153
Неудаление ни одного элемента	33	33
Случайное заполнение массива	Не определено	38

2.7 Оценка ёмкостной сложности алгоритма

Количество переменных, создаваемых в функции, не зависит от входных данных. Поэтому V(n) – это константа.

3 РЕШЕНИЕ ЗАДАЧИ 1 ПО АЛГОРИТМУ 2

3.1 Формулировка задачи

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

3.2 Математическая модель решения

Производится инициализация счетчика j. Производится перебор элементов. Перебираемый элемент перемещается на j-ую позицию. Если перебираемый элемент не равен удаляемому key, то j увеличивается на 1. Если перебираемый элемент равен удаляемому key, то j не увеличивается на 1, следовательно, следующий элемент, не равный удаляемому key, будет перемещен на позицию удаляемого элемента.

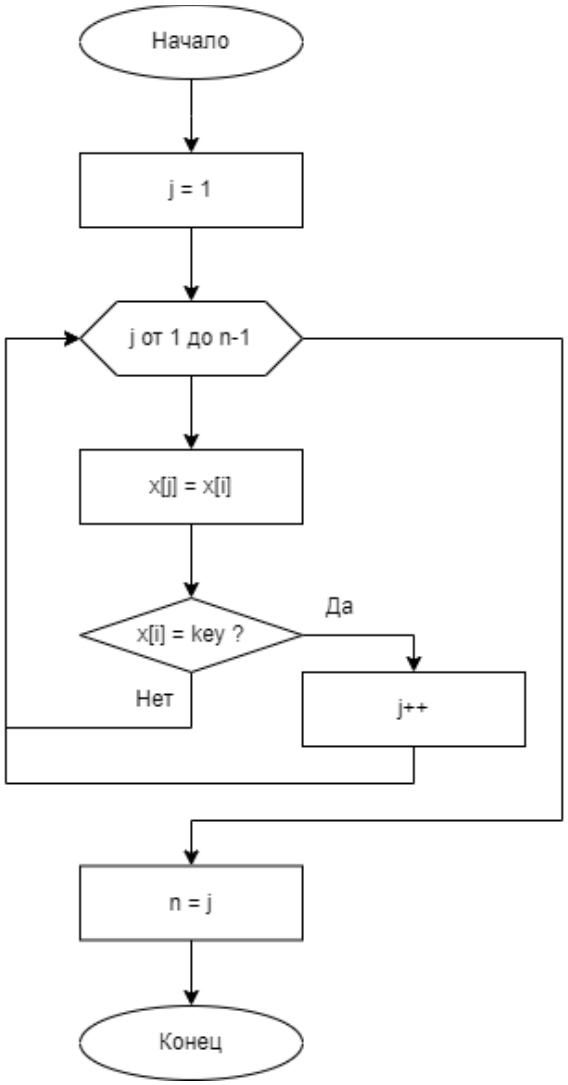

Рисунок 8 - Блок схема второго алгоритма

Инвариант цикла: с каждой итерацией удаляется (или не удаляется) элемент массива равный (или не равный) значению *key* и уменьшается (или не уменьшается) длина массива на единицу. Следовательно, на *i*-ой итерации среди элементов с индексами с 0 по *i*-ый нет элементов со значением *key*.

Таблица 4 - Подсчет количества операторов во втором алгоритме

Оператор	Кол-во выполнения оператора в строке	
	В лучшем случае (все элементы массива равны key)	В худшем случае (в массиве нет элементов key)
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
return n;	1	1

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

Реализация алгоритма показана на рис. 9.

На рис.10 показаны результаты отладки функции на массиве из 10 элементов. Мы видим, что было выполнено 42 операции, и все элементы, равные 2, были удалены.

```cpp
int delOtherMethod(int *x, int n, int key)
{
    int j = 0;
    int op_count = 2; // инициализация переменной i и j
    for (int i = 0; i < n; i++)
    {
        op_count += 1; // сравнение в цикле
        x[j] = x[i];
        op_count += 1; // перемещение элементов

        op_count += 1; // условынй оператор
        if (x[i] != key)
        {
            op_count += 1; // увеличивание j
            j++;
        };
    }

    op_count += 1; // присваивание
    n = j;

    op_count += 1; // возврат n

    cout << "delOtherMethod кол-во операций: " << op_count << endl;
    return n;
}
```

Рисунок 9 - Реализация второго алгоритма на языке C++

```
Тестирование для n = 10, key = 2

Тестирование: Случайный массив
n = 10: 7 6 4 2 6 8 2 7 7 9
delOtherMethod кол-во операций: 42
n = 8: 7 6 4 6 8 7 7 9
```

Рисунок 10 - Тестирование алгоритма на массиве n = 10

На рис.11 показаны результаты отладки функции на массиве из 100 элементов. Мы видим, что было выполнено 397 операций, и все элементы, равные 2, были удалены.

```
Тестирование для n = 100, key = 2

Тестирование: Случайный массив
n = 100: 4 8 2 7 3 1 5 6 6 5 5 2 8 6 6 3 8 1 5 2 6 9 1 9 8 6 8 9 7 9 5 5 1 1 1 1 1 6 6 3 3 7 4 5 3 9 1 2 7 8 6 3
3 1 3 9 1 5 1 3 3 3 5 8 5 4 1 2 6 9 3 2 3 7 9 5 5 3 5 9 4 5 4 3 8 4 4 4 3 9 5 3 2 8 9 5 1 9 3 5
delOtherMethod кол-во операций: 397
n = 93: 4 8 7 3 1 5 6 6 5 5 8 6 6 3 8 1 5 6 9 1 9 8 6 8 9 7 9 5 5 1 1 1 1 1 6 6 3 3 7 4 5 3 9 1 7 8 6 3 3 1 3 9 1
 5 1 3 3 3 5 8 5 4 1 6 9 3 3 7 9 5 5 3 5 9 4 5 4 3 8 4 4 4 3 9 5 3 8 9 5 1 9 3 5
```

Рисунок 11 - Тестирование алгоритма на массиве n = 100

3.4 Тестирование алгоритма в разных ситуациях

Тестирование алгоритма в разных ситуациях продемонстрировано на рис.12. Мы видим, что лучшим случаем является ситуация удаления всех элементов, ей соответствует 34 операции. Худшим случаем является ситуация неудаления ни одного элемента, которой соответствует 44 операции. В случае заполнения массива случайными числами количество операций было равно 43.

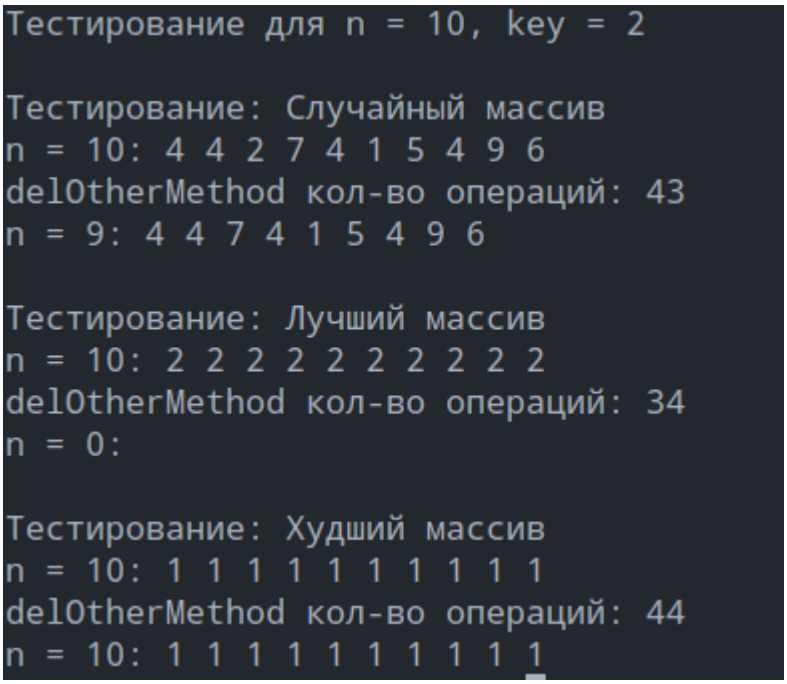

Рисунок 12 - Тестирование алгоритма в разных ситуациях

3.5 Результаты тестирования

По результатам тестирования составим таблицу 4 и сравним результаты тестирования с предсказанными теоретическими результатами. Теоретические расчеты приведены как значение T(n) для n=10 для каждого случая.

Таблица 5 - Сравнение результатов тестирования второго алгоритма

Ситуация	Теоретический расчёт	Результат тестирования
Удаление всех элементов	34	34
Неудаление ни одного элемента	44	44
Случайное заполнение массива	Не определена	43

3.6 Оценка ёмкостной сложности алгоритма

Количество переменных, создаваемых в функции, не зависит от входных данных. Поэтому V(n) – это константа.

4 ВЫВОД ПО ЗАДАНИЮ 1

В лучшем случае количество операций в первом алгоритме меньше, чем во втором, но во всех других случаях второй алгоритм показывает результаты гораздо лучше, чем первый. Порядок его роста в худшем случае линейный, а у первого – квадратичный. Следовательно, второй алгоритм является более оптимизированным для общего случая.

5 ЗАДАНИЕ 2

5.1 Формулировка задачи

Дана целочисленная прямоугольная матрица размером n*m. Определить номер строки, в которой находится самая длинная серия одинаковых элементов.

5.2 Математическая модель решения

Объявляем переменную *line* для записи номера конечной строки, инициализируем переменную *mx* = 0 для хранения значения длины максимальной серии одинаковых чисел. Проходим по матрице двумерным циклом, инициализируя во внешнем цикле переменную *cur*, для хранения текущей длины серии чисел. В каждый проход внутреннего цикла (по строке) сравниваем текущий элемент и следующий, если они совпадают, то увеличиваем *cur*, иначе *cur* = 1. Если элементы совпали, то сравниваем значение *cur* с *mx*, если оно больше, записываем номер текущей строки в *line* и записываем *cur* в *mx*.

Инвариант цикда: с каждой итерацией происходит увеличение *i* на 1, следовательно на *n*-ой итерации цикл завершится.

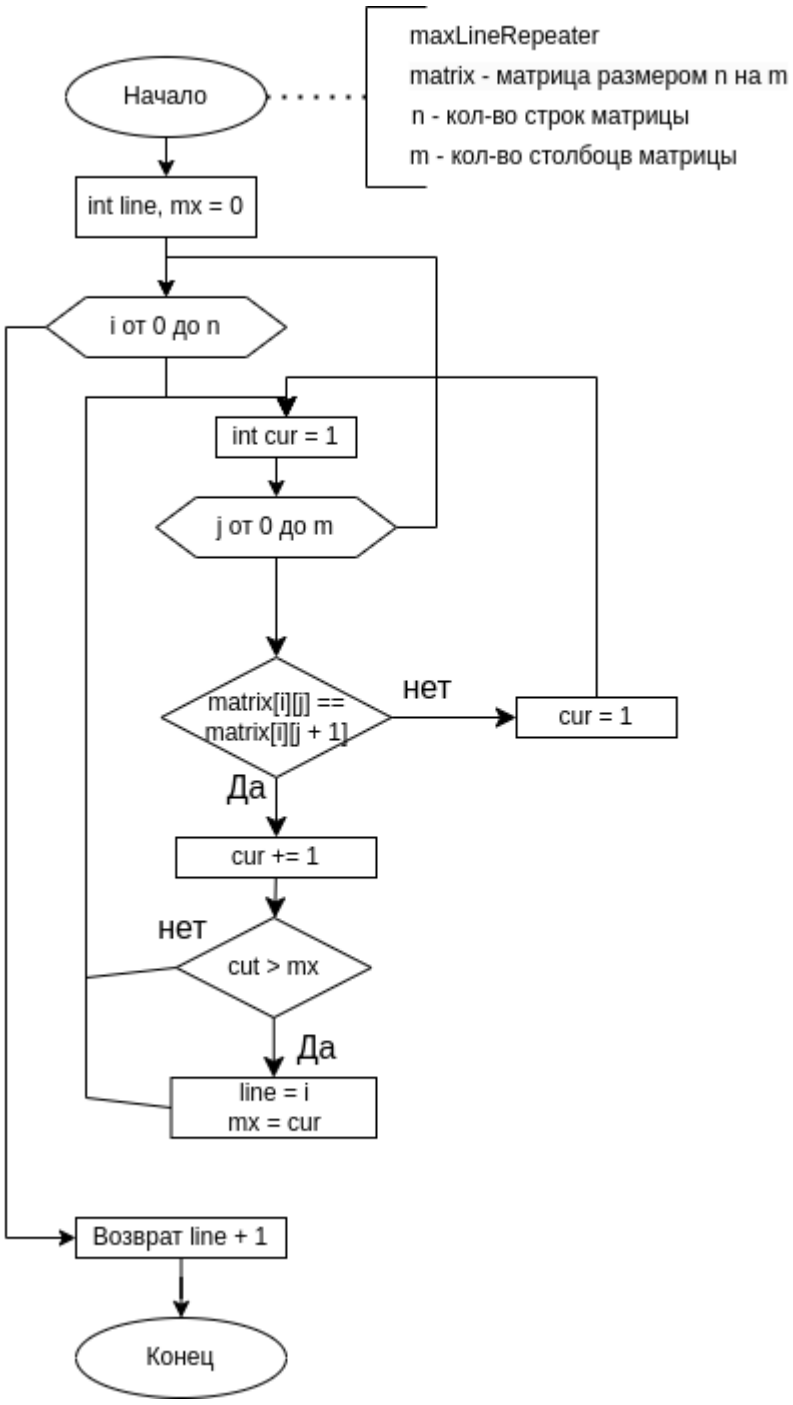

Рисунок 13 - Блок-схема алгоритма второй задачи

Таблица 6 - Подсчёт количества операторов в алгоритме

Оператор	Кол-во выполнения оператора в строке	
	В лучшем случае (в строке все элементы разные)	В худшем случае (все элементы строки одинаковые)
int line, mx = 0;	2	2
for (int i = 0; i < n; i++)	n + 1	n + 1
int cur = 1	n + 1	n + 1
for (int j = 0; j < m - 1; j++)	nm	nm
matrix[i][j] == matrix[i][j + 1]	nm	nm
cur += 1	0	n(m - 1)
if (cur > mx)	0	m - 1
line = i	0	m - 1
mx = cur	0	m - 1
cur = 1	n(m - 1)	0
return line + 1	2	2

В лучшем случае: T(nm) = 3nm + 3n + 5 => T(nm) = nm — кол-во элементов => линейная зависимость.

В худшем случае: T(nm) = 3nm + 6m + 2n + 6 => T(nm) = nm — кол-во элементов => линейная зависимость.

В среднем случае: T(nm) = (3nm + 3n + 5 + 3nm + 6m + 2n + 6) / 2 = 3nm + 2,5n + 3m + 5,5 => T(nm) = nm — кол-во элементов => линейная зависимость.

5.3 Реализация алгоритма

Реализация алгоритма представлена на рис.14

```
int maxLineRepeater(vector<vector<int>> matrix, int n, int m) {

    int line;
    int mx = 0;
    int op_count = 3; // инициализация line, mx и i
    for (int i = 0; i < n; i++)
    {
        op_count += 3; // инициализация cur и j и сравнение в цикле
        int cur = 1;
        for (int j = 0; j < m - 1; j++)
        {
            op_count += 2; // сравнение в цикле и условный оператор
            if (matrix[i][j] == matrix[i][j + 1])
                cur += 1;
                op_count += 2; // инкремент cur и условный оператор
                if (cur > mx)
                {
                    line = i;
                    mx = cur;
                    op_count += 2; // два присваивания line и mx
                }
            else
            {
                cur = 1;
                op_count += 1; // присваивание cur
            }
        }
    }

    op_count += 2; // инкремент и возврат line
    cout << "maxLineRepeater T(nm) = " << op_count << endl;
    return line + 1;
}
```

Рисунок 14 - Реализация алгоритма

5.4 Тестирование алгоритма

Составим табл. 7 для определения параметров тестов и оценки количества сравнений и операций для матрицы 3x3.

Ситуация	Теоретический расчёт	Результат тестирования
Все элементы в строках разные	45	45
Все элементы в строках одинаковые	46	46
Случайное заполнение матрицы	Не определено	43

Результаты тестирования показаны на рис. 15

```
3 3
Тестирование: Все элементы в строках разные
1 2 3
1 2 3
1 2 3
maxLineRepeater T(nm) = 45

Строка номер: 1
3 3
Тестирование: Все элементы в строках одинаковые
3 3 3
3 3 3
3 3 3
maxLineRepeater T(nm) = 46

Строка номер: 1
3 3
Тестирование: Случайный массив
1 4 3
2 2 3
3 2 3
maxLineRepeater T(nm) = 46

Строка номер: 2
```

Рисунок 15 - Тестирование алгоритма

5.5 Оценка ёмкостной сложности алгоритма

Тестирование алгоритма для больших значений n = 10 и m = 10.

```
10 10
Тестирование: Большие n
1 2 3 1 2 3 1 2 3 4
3 8 1 9 2 3 4 7 2 3
4 3 1 9 3 3 3 2 1 3
4 3 3 1 4 1 4 1 2 3
4 9 8 9 8 8 8 8 8 8
1 3 3 3 3 1 1 2 1 2
1 3 5 2 4 6 2 3 2 5
3 3 2 2 1 5 6 5 4 4
9 9 9 9 3 1 2 1 2 1
3 1 3 1 2 1 2 1 2 1
maxLineRepeater T(nm) = 488

Строка номер: 3
```

Рисунок 16 - Тестирование для n, m = 10

Таблица 7 - Результаты тестирования при больших n и m

Кол-во элементов	Результат тестирования
n = 10 m = 10	488

5.6 Оценка ёмкостной сложности алгоритма

Количество переменных, создаваемых в функции, не зависит от входных данных. Поэтому V(n) – это константа.

6 ВЫВОДЫ

В процессе выполнения работы были получены знания определения сложности и зависимости алгоритма, его реализации на языке программирования высокого уровня, тестирования его работы и описания работы собственного алгоритма с построением инварианта.