МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания 1

Тема:

Оценка сложности и определение эффективности алгоритма

Выполнил студент Цемкало А. Р.

Фамилия И.О.

группа ИКБО-10-20

Москва 2021

СОДЕРЖАНИЕ

Задание 1

1.1. Постановка задачи

Определить эффективный алгоритм из двух предложенных, используя оценку теоретической сложности каждого из алгоритмов и емкостную сложность, решения следующей задачи: дан массив из n элементов целого типа, удалить из массива все значения равные заданному.

Алгоритм 1

1.2. Модель решения поставленной задачи

a) Описание выполнения алгоритма

Алгоритм проходит по всем элементам в массиве по порядку. Если встречается элемент, равный key, удаляется данный элемент. Удаление происходит таким образом, что все последующие элементы сдвигаются влево (то есть под номером удаляемого элемента теперь стоит его сосед справа). После сдвига влево n уменьшается на единицу.

b) Определение инварианта цикла для внешнего цикла (доказательство корректности цикла)

Инвариант внешнего цикла: «После выполнения i-го шага i-ый элемент либо удаляется, сдвигая последние n-i элементов массива влево, либо, если элемент удалять не нужно, i увеличивается, таким образом переходя к следующему шагу (к проверке следующего элемента).»

```
void delFirstMethod(int& n, int key, int* x) {
	int i = 0;
	int k_moving = 0, k_comparing = 0;
	// Перед началом цикла i < n, так как i = 0, а n - длина массива
	while (i < n) {
		k_comparing++;
		if (x[i] == key) {
			for (int j = i; j < n - 1; j++) {
				x[j] = x[j + 1];
				k_moving++;
			}
			n--;
		}
		else {
			i++;
		}

		// Если элемент удалён из массива, то элементы от i до n-1 становятся равны элементам от i+1 до n, то есть "соседу справа",
		// а n уменьшается на 1, таким образом последний лишний элемент отсекается.

		// Если же элемент не удалён, то i увеличивается на 1, чтобы перейти к анализу следующего элемента.
	}

	// Цикл завершается, когда i = n-1, то есть цикл проверил все элементы массива от 0 до n-1, ненужные - удалены.

	cout << "Number of movements: " << k_moving << endl;
```

```
        cout << "Number of comparisons: " << k_comparing << endl;
}
```

c) Определение вычислительной сложности алгоритма, используя теоретический подход

Все данные алгоритма, для вывода функции роста представлены в табл.1.

Таблица 1

Номер оператора	Оператор	Время выполнения одного оператора	Кол-во выполнений оператора в строке
	1	2	3
1	int i = 0;	C1	1 раз
2	while (i < n) {	C2	n +1 раз
3	if (x[i] == key) {	C3	n раз
4	for (int j = i; j < n - 1; j++) {	C4	$\sum_{i=1}^{n} t_i$
5	x[j] = x[j + 1];	C5	$\sum_{i=1}^{n-1} t_i$
6	}		
7	n--;	C6	n раз
8	}		
9	else {		
10	i++;	C7	n раз
11	}		
12	}		

Выведем функцию роста для времени выполнения алгоритма:

$$T(n) = C1 + C2 * (n + 1) + C3 * n + C4 * \sum_{i=1}^{n} t_i + C5 * \sum_{i=1}^{n-1} t_i +$$

$$+C6 * n + C7 * n \qquad (1)$$

Определим порядок роста *в лучшем случае*, т.е. когда тело вложенного цикла не выполняется (удаляемых элементов нет). Тогда количество выполнений оператора 3 по-прежнему равна n, но операторы 4, 5 и 6 выполняться не будут. 10-ый выполнится n раз.

Подставим эти значения в формулу (1):

$T(n) = C1 + C2 * (n + 1) + C3 * n + C7 * n = (C2 + C3 + C7) * n + (C1 + +C2) = An + B$

Порядок роста времени в зависимости от n наилучшем случае линейный, т.е. $T(n)=\Theta(n)$.

Определим порядок роста *в худшем случае*, т.е. когда оператор 4 выполняется полное количество раз (массив состоит только из удаляемых элементов).

В этом случае в сумме $\sum_{i=1}^{n} t_i$ значение t_i равно i.

Тогда значение $\sum_{i=1}^{n} t_i = \sum_{i=1}^{n} i = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$.

Значение $\sum_{i=1}^{n-1} t_i = \frac{(n-1)(n-1)}{2} = \frac{n^2-2n+1}{2}$.

Подставим эти значения в формулу (1):

$$T(n) = C1 + C2 * (n+1) + C3 * n + C4 * \frac{n^2-n}{2} + C5 * \frac{n^2-2n+1}{2} + C6 * n +$$

$$+C7 * n = (C4 + C5) * \frac{n^2}{2} + \left(C2 + C3 - \frac{C4}{2} - C5 + C6 + C7\right) * n + \left(C1 + C2 + \right.$$

$$\left. + \frac{C5}{2}\right) = An^2 + Bn + C$$

Пренебрегаем константой C. Получаем $T(n)= An^2+Bn$. Функция n^2 имеет порядок роста выше, чем функция n. Таким образом, в выражении $T(n)= An^2+Bn$, доминирующей функцией является n^2, и она определяет порядок роста для алгоритма в худшем случае. Т.е. $T(n)=\Theta(n^2)$.

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

```
void delFirstMethod(int &n, int key, int* x) {
	int i = 0;
	int k_moving = 0, k_comparing = 0;

	while (i < n) {
		k_comparing++;
		if (x[i] == key) {
			for (int j = i; j < n - 1; j++) {
				x[j] = x[j + 1];
				k_moving++;
			}
			n--;
		}
		else {
			i++;
		}
	}

	cout << "Number of movements: " << k_moving << endl;
	cout << "Number of comparisons: " << k_comparing << endl;
}


int main() {
	int key = 2;
	int n = 10;
	int* x = new int[n] { 1, 2, 3, 2, 5, 2, 2, 5, 7, 2 };

	//int n = 100;
	//int* x = new int[n] { 1, 2, 3, 2, 5, 2, 2, 5, 7, 2, 1, 2, 3, 2, 5, 2, 2, 5, 7, 2,
	1, 2, 3, 2, 5, 2, 2, 5, 7, 2, 1, 2, 3, 2, 5, 2, 2, 5, 7, 2, 1, 2, 3, 2, 5, 2, 2, 5,
	7, 2, 1, 2, 3, 2, 5, 2, 2, 5, 7, 2, 1, 2, 3, 2, 5, 2, 2, 5, 7, 2, 1, 2, 3, 2, 5, 2,
	2, 5, 7, 2, 1, 2, 3, 2, 5, 2, 2, 5, 7, 2, 1, 2, 3, 2, 5, 2, 2, 5, 7, 2 };
	cout << "Old array: ";
```

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

	delFirstMethod(*&n, key, x);
	cout << "New array (" << key << " was deleted): ";
	for (int i = 0; i < n; i++) {
		cout << x[i] << " ";
	}
	cout << endl;
}
```

```
Old array: 1 2 3 2 5 2 2 5 7 2
Number of movements: 26
Number of comparisons: 10
New array (2 was deleted): 1 3 5 5 7
```

Рисунок 1 - Отладка функции delFirstMethod на массиве при n=10 (key=2)

```
Old array: 1 2 3 2 5 2 2 5 7 2 1 2 3 2 5 2 2 5 7 2 1 2 3 2 5 2 2 5 7 2 1 2 3 2 5 2 2 5 7 2 1 2 3 2 5 2 2 5 7 2 1 2
Number of movements: 2510
Number of comparisons: 100
New array (2 was deleted): 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7
```

Рисунок 2 - Отладка функции delFirstMethod на массиве при n=100 (key=2)

1.4. Реализация функции: заполнение массива датчиком случайных чисел, вывод массива на экран монитора

```
#include <iostream>
using namespace std;

void delFirstMethod(int &n, int key, int* x) {
	int i = 0;
	int k_moving = 0, k_comparing = 0;

	while (i < n) {
		k_comparing++;
		if (x[i] == key) {
			for (int j = i; j < n - 1; j++) {
				x[j] = x[j + 1];
				k_moving++;
			}
			n--;
		}
		else {
			i++;
		}
	}

	cout << "Number of movements: " << k_moving << endl;
	cout << "Number of comparisons: " << k_comparing << endl;
}

int main() {
	int key = 24;
	int n = 10;
	int* x = new int[n];
	cout << "Old array: ";
	for (int i = 0; i < n; i++) {
		x[i] = rand() % 100;
		cout << x[i] << " ";
```

```
	}
	cout << endl;

	delFirstMethod(*&n, key, x);
	cout << "New array (" << key << " was deleted): ";
	for (int i = 0; i < n; i++) {
		cout << x[i] << " ";
	}
	cout << endl;
}
```

```
Old array: 41 67 34 0 69 24 78 58 62 64
Number of movements: 4
Number of comparisons: 10
New array (24 was deleted): 41 67 34 0 69 78 58 62 64
```

Рисунок 3 - Отладка функции delFirstMethod на массиве, заполненном датчиком случайных чисел, вывод массива на экран монитора (key=24)

1.5. Представление результатов тестирования с указанием количества операций согласно теоретическим расчетам и полученным при выполнении алгоритма

Для массива { 1, 2, 3, 2, 5, 2, 2, 5, 7, 2 } кол-во перемещений должно быть равно 8+6+4+3=21.

Количество сравнений = 10.

```
Old array: 1 2 3 2 5 2 2 5 7 2
Number of movements: 21
Number of comparisons: 10
New array (2 was deleted): 1 3 5 5 7
```

Рисунок 4 - Отладка функции delFirstMethod на массиве { 1, 2, 3, 2, 5, 2, 2, 5, 7, 2 } (key=2)

Количество операций согласно теоретическим расчетам и полученных при выполнении алгоритма одинаково.

Алгоритм 2

1.6. Модель решения поставленной задачи

a) Описание выполнения алгоритма

Алгоритм проходит по всем элементам в массиве по порядку. Нужные элементы (не равные key) отправляются в начало массива. Число j - номер, под которым стоит элемент в новом массиве (с уже удалёнными элементами). В конце n становится равной j, таким образом, ненужная часть массива просто обрезается.

b) Определение инварианта цикла для внешнего цикла (доказательство корректности цикла)

Инвариант внешнего цикла: «После выполнения i-го шага i-ый элемент либо записывается после предыдущего не удаляемого элемента, либо с ним ничего не происходит, чтобы поверх него был записан «нужный» (если после

удаляемого элемента нет тех, которые нужно сохранить, j не будет увеличиваться, то есть часть массива будет отсечена). Поверх старого массива записывается новый.»

```
void delOtherMethod(int& n, int key, int* x) {
	int j = 0;
	// Перед началом цикла i < n, так как i = 0, а n - длина массива
	for (int i = 0; i < n; i++) {
		x[j] = x[i];
		if (x[i] != key) {
			j++;
		}
		// Если элемент не должен быть удалён, то j не увеличивается, чтобы при следующей итерации поверх этого элемента не был записан другой.
		// При этом увеличивается и длина будущего массива x, т.к. потом n будет равно j.
		// Если же элемент должен быть удалён, то значение j увеличено не будет, то есть при следующей итерации на место этого элемента будет записан следующий, который удалять не нужно.
	}
	// Цикл завершается, когда i = n-1, то есть цикл проверил все элементы массива от 0 до n-1.
	n = j;
}
```

c) Определение вычислительной сложности алгоритма, используя теоретический подход

Все данные алгоритма, для вывода функции роста представлены в табл.2.

Таблица 2

Номер оператора	Оператор	Время выполнения одного оператора	Кол-во выполнений оператора в строке
	1	2	3
1	int j = 0;	C1	1 раз
2	for (int i = 0; i < n; i++) {	C2	n +1 раз
3	x[j] = x[i];	C3	n раз
4	if (x[i] != key) {	C4	n раз
5	j++;	C5	n раз
6	}		
7	}		
8	n = j;	C6	1 раз

Описание результатов, представленных в столбце 3.

Оператор 1 выполняется один раз.

Оператор 2. Определяется количество выполнений условия цикла при просмотре всех значений.

Согласно циклу с предусловием: первый вход в цикл при i=0; последний вход в цикл при i=n-1; после последнего входа i=n, т.е. ещё одна проверка и завершение цикла. Считаем сколько раз выполнялся оператор i < n: n раз

обеспечивался вход в цикл и один раз при выходе из цикла, таким образом, всего n+1 раз за время работы.

Оператор 3. Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 4 (if). Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 5. Это оператор – блок оператора if. Выполняется столько раз, сколько и if (в худшем случае) – n раз.

Оператор 6. Этот оператор вне цикла, и он будет выполняться 1 раз.

Обозначим время выполнения алгоритма T(n) – функция, зависящая от n.

Определим время выполнения алгоритма - как сумму времени выполнения каждого оператора:

$$T(n) = C1 + C2 * (n + 1) + C3 * n + C4 * n + C5 * n + C6 \quad (1)$$

$$T(n) = (C2 + C3 + C4 + C5) * n + (C1 + C2 + C6) = An + B$$

Т.е. имеем линейную зависимость времени от размера входных данных.

Так как требуется определить порядок роста времени от n, то константами можно пренебречь.

В результате T(n) в худшем случае линейно зависит от n.

Рассмотрим лучший случай: удалить нужно все элементы. Цикл будет выполняться все равно n+1 раз. Тогда T(n) в лучшем случае линейно зависит от n.

Рассмотрим средний случай: удалить нужно половину элементов. Цикл будет выполняться все равно n+1 раз. Тогда T(n) в среднем случае линейно зависит от n.

Вывод. Порядок роста T(n)=Θ(n).

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

```
void delOtherMethod(int& n, int key, int* x) {
	int k_moving = 0, k_comparing = 0;
	int j = 0;
	for (int i = 0; i < n; i++) {
		if (i != j) {
			k_moving++;
		}
		x[j] = x[i];

		k_comparing++;
		if (x[i] != key) {
			j++;
		}
	}
	n = j;
	cout << "Number of movements: " << k_moving << endl;
	cout << "Number of comparisons: " << k_comparing << endl;
}

int main() {
```

```
	int key = 2;
	int n = 10;
	int* x = new int[n] { 1, 2, 3, 2, 5, 2, 2, 5, 7, 2 };
	//int* x = new int[n] { 1, 2, 3, 2, 5, 2, 2, 5, 7, 2, 1, 2, 3, 2, 5, 2, 2, 5, 7, 2,
1, 2, 3, 2, 5, 2, 2, 5, 7, 2, 1, 2, 3, 2, 5, 2, 2, 5, 7, 2, 1, 2, 3, 2, 5, 2, 2, 5, 7, 2, 1,
2, 3, 2, 5, 2, 2, 5, 7, 2, 1, 2, 3, 2, 5, 2, 2, 5, 7, 2, 1, 2, 3, 2, 5, 2, 2, 5, 7, 2, 1, 2,
3, 2, 5, 2, 2, 5, 7, 2, 1, 2, 3, 2, 5, 2, 2, 5, 7, 2 };
	cout << "Old array: ";
	for (int i = 0; i < n; i++) {
		cout << x[i] << " ";
	}
	cout << endl;

	delOtherMethod(*&n, key, x);
	cout << "New array (" << key << " was deleted): ";
	for (int i = 0; i < n; i++) {
		cout << x[i] << " ";
	}
	cout << endl;
}
```

```
Old array: 1 2 3 2 5 2 2 5 7 2
Number of movements: 8
Number of comparisons: 10
New array (2 was deleted): 1 3 5 5 7
```

Рисунок 5 - Отладка функции delOtherMethod на массиве при n=10 (key=2)

```
Old array: 1 2 3 2 5 2 2 5 7 2 1 2 3 2 5 2 2 5 7 2 1 2 3 2 5 2 2 5 7 2 1 2 3 2 5 2 2 5 7 2 1 2 3 2 5 2 2 5 7 2 1 2 3 2 5 2 2 5 7 2 1 2 3 2 5 2 2 5 7 2 1 2 3 2 5 2 2 5 7 2
Number of movements: 98
Number of comparisons: 100
New array (2 was deleted): 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7 1 3 5 5 7
```

Рисунок 6 - Отладка функции delOtherMethod на массиве при n=100 (key=2)

1.8. Реализация функции: заполнение массива датчиком случайных чисел, вывод массива на экран монитора

```
#include <iostream>
using namespace std;
void delOtherMethod(int& n, int key, int* x) {
	int k_moving = 0, k_comparing = 0;
	int j = 0;
	for (int i = 0; i < n; i++) {
		if (i != j) {
			k_moving++;
		}
		x[j] = x[i];

		k_comparing++;
		if (x[i] != key) {
			j++;
		}
	}
	n = j;
	cout << "Number of movements: " << k_moving << endl;
	cout << "Number of comparisons: " << k_comparing << endl;
}

int main() {
	int key = 8;
	int n = 10;
	int* x = new int[n];
```

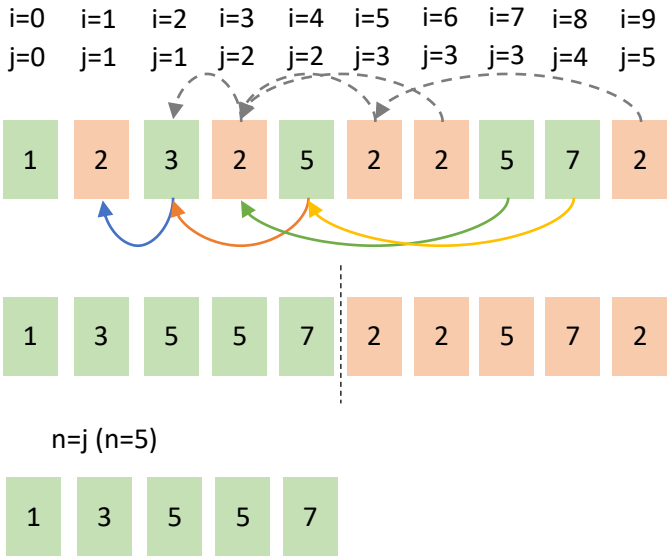
```
	cout << "Old array: ";
	for (int i = 0; i < n; i++) {
		x[i] = rand() % 10;
		cout << x[i] << " ";
	}
	cout << endl;
	delOtherMethod(*&n, key, x);
	cout << "New array (" << key << " was deleted): ";
	for (int i = 0; i < n; i++) {
		cout << x[i] << " ";
	}
	cout << endl;
}
```

```
Old array: 1 7 4 0 9 4 8 8 2 4
Number of movements: 3
Number of comparisons: 10
New array (8 was deleted): 1 7 4 0 9 4 2 4
```

Рисунок 7 - Отладка функции delOtherMethod на массиве, заполненном датчиком случайных чисел, вывод массива на экран монитора (key=24)

1.9. Представление результатов тестирования с указанием количества операций согласно теоретическим расчетам и полученным при выполнении алгоритма

Для массива { 1, 2, 3, 2, 5, 2, 2, 5, 7, 2 } кол-во перемещений должно быть равно 8:



Количество сравнений = 10 (всегда равно n).

```
Old array: 1 2 3 2 5 2 2 5 7 2
Number of movements: 8
Number of comparisons: 10
New array (2 was deleted): 1 3 5 5 7
```

Рисунок 9 - Отладка функции на массиве { 1, 2, 3, 2, 5, 2, 2, 5, 7, 2 } (key=2)

Количество операций согласно теоретическим расчетам и полученных при выполнении алгоритма одинаково.

1.10. Тестирование алгоритма в случаях: все элементы должны быть удалены, ни один элемент не удаляется.

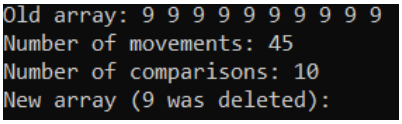


Рисунок 9 – Функция delFirstMethod, все элементы должны быть удалены

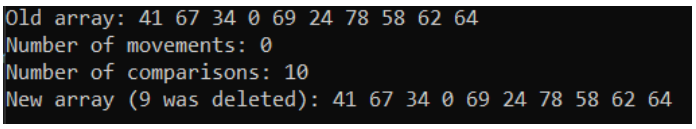


Рисунок 10 – Функция delFirstMethod, ни один элемент не удаляется

Теоретическая сложность алгоритма delFirstMethod в случае, когда все элементы должны быть удалены: $\Theta(n^2)$.

Теоретическая сложность алгоритма delFirstMethod в случае, когда ни один элемент не удаляется: $\Theta(n)$.

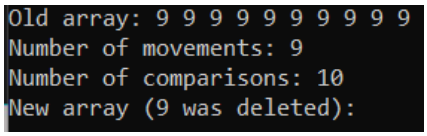


Рисунок 11 – Функция delOtherMethod, все элементы должны быть удалены

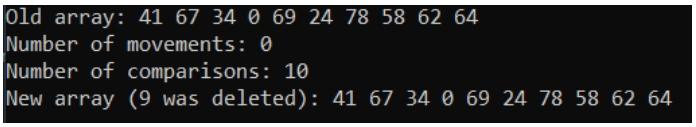


Рисунок 12 – Функция delOtherMethod, ни один элемент не удаляется

Теоретическая сложность алгоритма delOtherMethod в случае, когда все элементы должны быть удалены: $\Theta(n)$.

Теоретическая сложность алгоритма delOtherMethod в случае, когда ни один элемент не удаляется: $\Theta(n)$.

Таким образом, теоретическая сложность (в худшем случае) алгоритма delOtherMethod меньше, чем у алгоритма delFirstMethod.

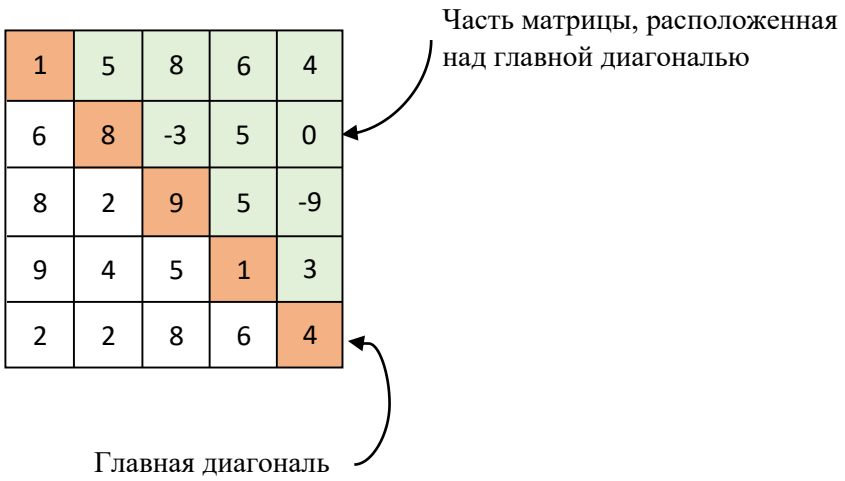
Задание 2

2.1 Постановка задачи

Вариант 6.

Найти максимальный элемент в части матрицы, расположенной над главной диагональю.

2.2 Модель решения



Часть матрицы, расположенная над главной диагональю – часть, в которой номер столбца больше номера строки.

Для поиска максимального числа в массиве нужно создать переменную maximum, в которую оно будет записываться. Если элемент больше переменной maximum, её значение становится равным этому элементу.

2.3 Разработка эффективного алгоритма

а) Разработка алгоритма

1. Создать матрицу.
2. Создать переменную maximum для записи максимального числа.
3. Двумя циклами (один из них вложенный) пройтись по всем элементам с итераторами i и j, где i – номер строки, j – номер столбца, причём j > i, так как нам нужны только элементы выше главной диагонали.
4. Во внутреннем цикле проверить: если текущий элемент больше переменной maximum, то в maximum записывается значение этого элемента.
5. Последнюю строчку проходить необязательно, так как там нет элемента, находящегося выше главной диагонали.

б) Определение инвариант

После выполнения каждого шага цикла в переменной maximum записан максимум из элементов массива [0, i][i+1, j].

в) Доказательство корректности циклов в алгоритме

```
	// Внешний цикл выполняется для i от 0 до n-2, проходя все строки, кроме последней.
	for (int i = 0; i < n - 1; i++) {
		// Во внутреннем цикле проверяются только те элементы, в которых номер столбца больше номера строки.
		// Таким образом, мы не проходим через элементы массива, которые ниже главной диагонали.
		for (int j = i + 1; j < n; j++) {
			if (matrix[i][j] > maximum) {
				maximum = matrix[i][j];
			}
```

```
                // Если значение элемента больше значения, записанного в maximum,
                // то maximum становится равным значению этого элемента.
            }
            // Цикл завершается, когда все строки, кроме последней, пройдены.
        }
```

г) Определение вычислительной сложности алгоритма на основе теоретического подхода

Все данные алгоритма, для вывода функции роста представлены в табл.1.

Таблица 1

Номер оператора	Оператор	Время выполнения одного оператора	Кол-во выполнений оператора в строке
	1	2	3
1	int n = 5;	C1	1 раз

2	int matrix[5][5] = {
	{1, 5, 8, 6, 4},
	{6, 8, -3, 5, 0},
	{8, 2, 9, 5, -9},
	{9, 4, 5, 1, 3},
	{2, 2, 8, 6, 4}
	};
		C2	1 раз

3	int maximum = INT_MIN;	C3	1 раз
4	for (int i = 0; i < n - 1; i++) {	C4	n+1 раз
5	for (int j = i + 1; j < n; j++) {	C5	$\sum_{j=1}^{n} t_j$
6	if (matrix[i][j] > maximum) {	C6	$\sum_{j=1}^{n-1} t_j$
7	maximum = matrix[i][j];	C7	$\sum_{j=1}^{n-1} t_j$
8	}		
9	}		
10	}		
11	cout << maximum << endl;	C8	1 раз

Рассмотрим кол-во выполнений оператора C5.

При j=1, оператор 5 выполнится n-1 раз.

При j=2, оператор 5 выполнится n-2 раза.

……...

При j=k, оператор 5 выполнится n-k раз.

……..

При j=n-2, оператор 5 выполнится 2 раза.

При j=n-1, оператор 5 выполнится 1 раза.

При j=n, оператор 5 выполнит еще одно сравнение и цикл завершиться.

Выведем функцию роста для времени выполнения алгоритма:

$$T(n) = C1 + C2 + C3 + C4 * (n + 1) + C5 * \sum_{j=1}^{n} t_j + C6 * \sum_{j=1}^{n-1} t_j + C7 * \sum_{j=1}^{n-1} t_j + C8 \quad (1)$$

Определим порядок роста в лучшем случае, т.е. операция внутри условия не выполняется (когда все элементы матрицы равны минимально возможному значению). Тогда оператор 7 выполняться не будет. Количество выполнений остальных операторов не поменяется.

В этом случае в сумме $\sum_{j=1}^{n} t_j$ значение t_j равно j.

Тогда значение $\sum_{j=1}^{n} t_j = \sum_{j=1}^{n} j = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$.

Значение $\sum_{j=1}^{n-1} t_j = \frac{(n-1)(n-1)}{2} = \frac{n^2-2n+1}{2}$.

Подставим эти значения в формулу (1):

$$T(n) = C1 + C2 + C3 + C4 * (n + 1) + C5 * \frac{n^2-n}{2} + C6 * \frac{n^2-2n+1}{2} + C8 = \left(\frac{C5}{2} + \frac{C6}{2}\right) * n^2 + \left(C4 - \frac{C5}{2} - C6\right) * n + \left(C1 + C2 + C3 + C4 + \frac{C6}{2} + C8\right) = An^2 + +Bn + C$$

Пренебрегаем константой C. Получаем $T(n)= An^2+Bn$. Функция n^2 имеет порядок роста выше, чем функция n. Таким образом, в выражении $T(n)= An^2+Bn$, доминирующей функцией является n^2, и она определяет порядок роста для алгоритма в лучшем случае. Т.е. $T(n)=Θ(n^2)$.

Определим порядок роста *в худшем случае*, т.е. когда оператор 7 выполняется полное количество раз (с каждым шагом значение элемента возрастает).

В этом случае в сумме $\sum_{j=1}^{n} t_j$ значение t_j равно j.

Тогда значение $\sum_{j=1}^{n} t_j = \sum_{j=1}^{n} j = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$.

Значение $\sum_{j=1}^{n-1} t_j = \frac{(n-1)(n-1)}{2} = \frac{n^2-2n+1}{2}$.

Подставим эти значения в формулу (1):

$$T(n) = C1 + C2 + C3 + C4 * (n + 1) + C5 * \frac{n^2-n}{2} + C6 * \frac{n^2-2n+1}{2} + C7 * * \frac{n^2-2n+1}{2} + C8 = \left(\frac{C5}{2} + \frac{C6}{2} + \frac{C7}{2}\right) * n^2 + \left(C4 - \frac{C5}{2} - C6 - C7\right) * n + \left(C1 + +C2 + C3 + C4 + \frac{C6}{2} + \frac{C7}{2} + C8\right) = An^2 + Bn + C$$

Пренебрегаем константой C. Получаем $T(n)= An^2+Bn$. Функция n^2 имеет порядок роста выше, чем функция n. Таким образом, в выражении $T(n)= An^2+Bn$, доминирующей функцией является n^2, и она определяет порядок роста для алгоритма в худшем случае. Т.е. $T(n)=\Theta(n^2)$.

Вывод: порядок роста $T(n)=\Theta(n^2)$.

2.4 Реализация алгоритма варианта в виде одной функции (без декомпозиции на другие функции).

```
#include <iostream>
using namespace std;

int main() {
	int n = 5;
	int matrix[5][5] = {
		{1, 5, 8, 6, 4},
		{6, 8, -3, 5, 0},
		{8, 2, 9, 5, -9},
		{9, 4, 5, 1, 3},
		{2, 2, 8, 6, 4}
	};
	int maximum = INT_MIN;
	for (int i = 0; i < n - 1; i++) {
		for (int j = i + 1; j < n; j++) {
			if (matrix[i][j] > maximum) {
				maximum = matrix[i][j];
			}
		}
	}
	cout << maximum << endl;
	return 0;
}
```

2.5 Проведение тестирований алгоритма на массиве из 10 чисел.

Таблица 1: Таблица тестов

Матрица	Ожидаемые выходные данные	Фактические выходные данные

`int matrix[10][10] = {`
`{25, 5, 8, 6, 4, 8, 2, 9, 5, 4},`
`{ 6, 8, 3, 5, 6, 1, -15, 8, 6, 4 },`
`{ 8, 2, 9, 5, 4, 6, 8, 3, 5, 6 },`
`{ 9, 4, 5, 1, 3, 2, 2, 10, 6, 4 },`
`{ 2, 2, 8, 6, 4, 9, 4, 5, 1, 3 },`
`{ 1, 30, 8, 6, 4, 8, 2, 9, 5, 4 },`
`{ 6, 8, 3, 5, 6, 1, 5, -8, 6, 4 },`
`{ 8, -20, 9, 5, 4, 6, 8, 3, 5, 6 },`
`{ 9, 4, 5, 1, 3, 2, 2, 8, 6, 4 },`
`{ 2, 2, 8, 6, 4, 9, 4, 5, 1, 3 }};`
	10	10
		`Martix:`
		`25 5 8 6 4 8 2 9 5 4`
		`6 8 3 5 6 1 -15 8 6 4`
		`8 2 9 5 4 6 8 3 5 6`
		`9 4 5 1 3 2 2 10 6 4`
		`2 2 8 6 4 9 4 5 1 3`
		`1 30 8 6 4 8 2 9 5 4`
		`6 8 3 5 6 1 5 -8 6 4`
		`8 -20 9 5 4 6 8 3 5 6`
		`9 4 5 1 3 2 2 8 6 4`
		`2 2 8 6 4 9 4 5 1 3`
		`max: 10`

`int matrix[10][10] = {`
`{25, 1, 1, 1, 1, 1, 1, 1, 1, 1},`
`{ 50, 25, 1, 1, 2, 1, 1, 1, 1, 1 },`
`{ 50, 50, 25, 1, 1, 1, 1, 1, 1, 1},`
`{ 50, 50, 50, 25, 1, 1, 1, 1, 1 ,1 },`
`{ 50, 50, 50, 50, 25, 1, 1, 1, 1, 1 },`
	2	2

{ 50, 30, 50, 50, 50, 25, 1, 1, -10, 1 }, { 50, 50, 50, 50, 50, 50, 25, 1, 1, 1 }, { 50, 50, 50, 50, 50, 50, 50, 25, 1, 1 }, { 50, 50, 50, 50, 50, 50, 50, 50, 25, 1 }, { 50, 50, 50, 50, 50, 50, 50, 50, 50, 25 } };		Martix: 25 1 1 1 1 1 1 1 1 1 50 25 1 1 2 1 1 1 1 1 50 50 25 1 1 1 1 1 1 1 50 50 50 25 1 1 1 1 1 1 50 50 50 50 25 1 1 1 1 1 50 30 50 50 50 25 1 1 -10 1 50 50 50 50 50 50 25 1 1 1 50 50 50 50 50 50 50 25 1 1 50 50 50 50 50 50 50 50 25 1 50 50 50 50 50 50 50 50 50 25 max: 2

int matrix[10][10] = {
{25, -10, -10, -10, -10, -10, -10, -10, -10, -10},
{ 50, 25, -10, -10, -10, -10, -10, -10, -10, -10 },
{ 50, 50, 25, -10, -10, -10, -10, -10, -10, -10},
{ 50, 50, 50, 25, -10, -10, -10, -10, -10 ,-10 },
{ 50, 50, 50, 50, 25, -10, -10, -10, -10, -10 },
{ 50, 30, 50, 50, 50, 25, -10, -10, -10, -10 },
{ 50, 50, 50, 50, 50, 50, 25, -10, -10, -10 },
{ 50, 50, 50, 50, 50, 50, 50, 25, -10, -10 },
{ 50, 50, 50, 50, 50, 50, 50, 50, 25, -10 },
{ 50, 50, 50, 50, 50, 50, 50, 50, 50, 25 }
};
	-10	-10
		Martix:
		25 -10 -10 -10 -10 -10 -10 -10 -10 -10
		50 25 -10 -10 -10 -10 -10 -10 -10 -10
		50 50 25 -10 -10 -10 -10 -10 -10 -10
		50 50 50 25 -10 -10 -10 -10 -10 -10
		50 50 50 50 25 -10 -10 -10 -10 -10
		50 30 50 50 50 25 -10 -10 -10 -10
		50 50 50 50 50 50 25 -10 -10 -10
		50 50 50 50 50 50 50 25 -10 -10
		50 50 50 50 50 50 50 50 25 -10
		50 50 50 50 50 50 50 50 50 25
		max: -10

int matrix[10][10] = {
{25, -10, -10, -10, -10, -10, -10, -10, -10, -10},
{ 50, 25, -10, -10, -10, -10, -10, -10, -10, -10 },
{ 50, 50, 25, -10, -10, -10, -10, -5, -10, -10},
{ 50, 50, 50, 25, -10, -10, -10, -10, -10 ,-10 },
{ 50, 50, 50, 50, 25, -10, -10, -10, -10, -10 },
{ 50, 30, 50, 50, 50, 25, -10, -10, -10, -10 },
{ 50, 50, 50, 50, 50, 50, 25, -10, -10, -10 },
{ 50, 50, 50, 50, 50, 50, 50, 25, -10, -10 },
{ 50, 50, 50, 50, 50, 50, 50, 50, 25, -10 },
{ 50, 50, 50, 50, 50, 50, 50, 50, 50, 25 }
};
	-5	-5
		Martix:
		25 -10 -10 -10 -10 -10 -10 -10 -10 -10
		50 25 -10 -10 -10 -10 -10 -10 -10 -10
		50 50 25 -10 -10 -10 -10 -5 -10 -10
		50 50 50 25 -10 -10 -10 -10 -10 -10
		50 50 50 50 25 -10 -10 -10 -10 -10
		50 30 50 50 50 25 -10 -10 -10 -10
		50 50 50 50 50 50 25 -10 -10 -10
		50 50 50 50 50 50 50 25 -10 -10
		50 50 50 50 50 50 50 50 25 -10
		50 50 50 50 50 50 50 50 50 25
		max: -5

Все требования выполняются

2.6 Выполнение практической оценки сложности алгоритма для больших n.

Для n = 100, т.е. когда размер матрицы 100*100, алгоритм выполнен за 18 миллисекунд.

Худший случай: Алгоритм с массивом из чисел, заполненных таким образом, что с каждым шагом число увеличивается, выполнен за 20 миллисекунд.

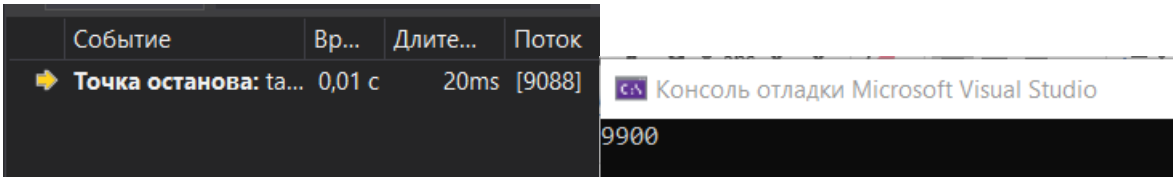


Рисунок 2, 2 - Скриншоты результатов тестирования в худшем случае

Лучший случай: Алгоритм с массивом, заполненным минимально возможными числами, выполнен за 19 миллисекунд.

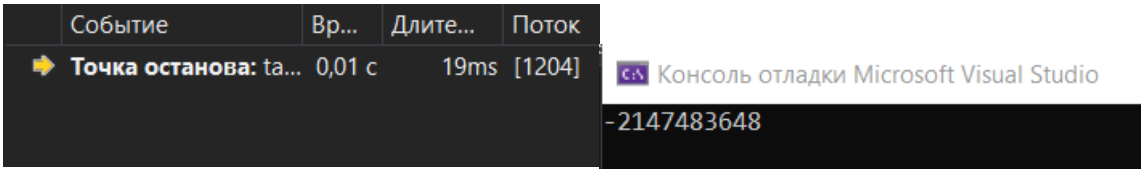


Рисунок 3, 4 - Скриншоты результатов тестирования в лучшем случае

ВЫВОДЫ

Приобретены практические навыки по определению:

1. сложности алгоритмов на теоретическом и практическом уровнях
2. эффективного алгоритма решения задачи из нескольких алгоритмов

Используя оценку теоретической сложности каждого из алгоритмов и ёмкостную сложность, из двух предложенных эффективным определён второй алгоритм (функция delOtherMethod).

Выполнено индивидуальное задание в соответствии с вариантом. Разработан эффективный алгоритм по нахождению максимального элемента в части матрицы, расположенной над главной диагональю.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Процедурное программирование Языки программирования - Сайт lizochekk! [Электронный ресурс]: URL: https://lizochekk.jimdofree.com/программирование/
2. DevDocs – C++ documentation [Электронный ресурс] URL: https://devdocs.io/cpp/