Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Фундаментальные Науки»
КАФЕДРА ФН-12 «Математическое моделирование»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ
по дисциплине «Типы и структуры данных»
Тема: «Сравнительный анализ рекурсивного и нерекурсивного алгоритмов»

Выполнил студент группы ФН12-31Б: Егоров К.Ю.

дата, подпись Ф.И.О.

Проверил преподаватель: Строганов Ю.В.

дата, подпись Ф.И.О.

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Разработчики часто сталкиваются с дилеммой выбора между рекурсивными и итеративными подходами.

Цель работы

Провести сравнительный анализ рекурсивного и нерекурсивного алгоритмов. Для достижения поставленной цели требуется выполнить следующие задачи:

1) разработать рекурсивный и нерекурсивный алгоритмы решения задачи, согласно варианту;

2) описать средства разработки и инструменты замера процессорного времени выполнения реализации алгоритмов;

3) реализовать разработанные алгоритмы;

4) выполнить тестирование реализации алгоритмов;

5) выполнить теоретическую оценку затрачиваемой реализацией каждого алгоритма памяти(для рекурсивного алгоритма — на материале анализа высоты дерева рекурсивных вызовов и оценки затрачиваемой на один вызов функции памяти);

6) выполнить замеры процессорного времени выполнения реализации алгоритмов в зависимости от варьируемого входа(одна точка на графике получается делением времени выполнения k идентичных расчетов на k, k >= 100);

7) сделать выводы из сравнительного анализа реализации рекурсивного и нерекурсивного алгоритмов решения одной и той же задачи, заданной вариантом, по критериям ёмкостной эффективности(на материале теоретической оценки пиковой временной эффективности и на материале результатов измерений).

1 Аналитическая часть

Рекурсивный алгоритм — это алгоритм, который решает задачу, вызывая сам себя один или несколько раз для решения тесно связанных подзадач. Эти вызовы называются рекурсивными вызовами [1].

Итеративный алгоритм — это алгоритм, который реализуется с помощью циклов(таких как for и while), которые повторяют выполнение блока операторов до достижения некоторого условия завершения. В отличие от рекурсии, итерация не требует организации стека вызовов и обычно более эффективна по использованию памяти [1].

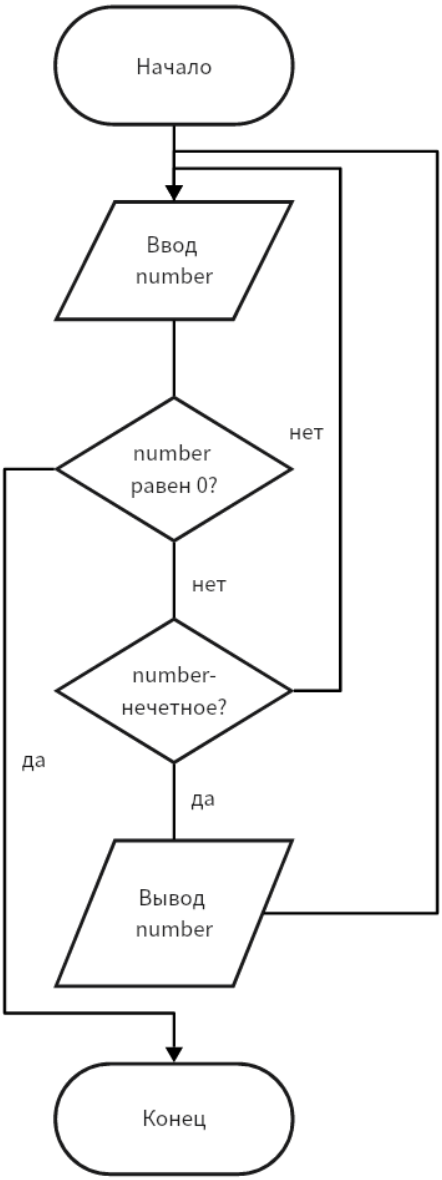
Описание варианта для разработки алгоритмов: Дана последовательность натуральных чисел(одно число в строке), завершающаяся числом 0. Выведите все нечетные числа из этой последовательности, сохраняя их порядок. В этой задаче нельзя использовать глобальные переменные и передавать какие-либо параметры в рекурсивную функцию. Функция получает данные, считывая их с клавиатуры. Функция не возвращает значение, а сразу же выводит результат на экран. Основная программа должна состоять только из вызова этой функции.

1.1 Вывод

Были приведены все основные термины и вариант для разработки алгоритмов.

2 Конструкторская часть

Блок-схема обоих алгоритмов



2.1 Вывод

Была приведена блок-схема, описывающая вариант, разработанный в виде итеративного алгоритма и рекурсивного алгоритма.

3 Технологическая часть

3.1 Выбор средств реализации

Для программной реализации поставленных задач использовалась среда разработки *Visual Studio Code*, используемый язык программирования — C++. Для компиляции кода использовался компилятор $g++$. Исследование проводилось на ноутбуке с такими техническими составляющими:

1) оперативная память — DDR4 16ГБ (скорость 3200 МТ/с);
2) процессор — 12th Gen Intel(R) Core(TM) i7-12700H (2.30 ГГц);
3) тип системы — 64-разрядная операционная система, процессор x64;
5) операционная система — Windows 11 Pro (сборка 26100.6584);
6) диск — SSD 954ГБ (NVMe).

3.2 Примеры работы программной реализации

Использование первого режима

На рисунке 1 представлен запуск программной реализации с такими входными данными для обоих алгоритмов: пять различных натуральных чисел с 0 в качестве шестого числа, среди которых будет два нечетных числа(на экране помимо введенных чисел должны быть продублированы нечетные числа).

```
Выберите режим работы
Введите соответствующий номер
1) Пользовательский ввод последовательностей
2) Автоматический массированный замер времени работы алгоритмов
1
Введите последовательность натуральных чисел, завершающуюся 0
Рекурсивная версия:
1
1
3
3
4
6
8
0
Введите последовательность еще раз
Классическая версия:
1
1
3
3
4
6
8
0
```

Рисунок 1 — Пример 1

На рисунках 2 и 3 представлены запуски программной реализации для обоих алгоритмов с такими входными данными: вводится некоторое количество натуральных чисел, после чего вводится отрицательное число(в этом случае программа должна выдать ошибку).

```
Выберите режим работы
Введите соответствующий номер
1) Пользовательский ввод последовательностей
2) Автоматический массированный замер времени работы алгоритмов
1
Введите последовательность натуральных чисел, завершающуюся 0
Рекурсивная версия:
1
1
3
3
4
-1
Ошибка: введено отрицательное число
```

Рисунок 2 — Пример 2

```
Выберите режим работы
Введите соответствующий номер
1) Пользовательский ввод последовательностей
2) Автоматический массированный замер времени работы алгоритмов
1
Введите последовательность натуральных чисел, завершающуюся 0
Рекурсивная версия:
1
1
3
3
4
0
Введите последовательность еще раз
Классическая версия:
1
1
3
3
4
-1
Ошибка: введено отрицательное число
```

Рисунок 3 — Пример 3

Использование второго режима

На рисунках 4 и 5 представлен вход во второй режим работы, а также окончание работы этого режима.

```
Выберите режим работы
Введите соответствующий номер
1) Пользовательский ввод последовательностей
2) Автоматический массированный замер времени работы алгоритмов
2
Количество тестов: 100
Длина последовательности: 2
701
701
473
473
933
933
485
485
341
341
981
981
945
945
735
735
523
523
95
95
431
431
655
655
615
615
773
773
539
539
481
481
555
555
239
239
953
953
73
73
161
161
145
145
489
489
67
67
```

Рисунок 4 — Начало

```
825
825
947
947
577
577
489
489
853
853
287
287
571
571
419
419
235
235
407
407
859
859
657
657
237
237
17
17
947
947
475
475
847
847
Результаты замера времени
Рекурсивная версия: 0(мс)
Классическая версия: 0(мс)
```

Рисунок 5 — Окончание

3.3 Вывод

Была выполнена программная реализация двух режимов взаимодействия. Также были приведены результаты запуска программной реализации этого режима со всеми возможными исходами.

4 Исследовательская часть

4.1 Цель

Сделать теоретическую оценку трудоемкости каждого из алгоритмов и провести сравнение с результатами работы программной реализации, а также теоретическую оценку памяти каждого из алгоритмов.

4.2 Оценка затрачиваемой памяти алгоритмов

Для рекурсивной реализации каждый раз инициализируется одна целочисленная переменная. Однако, глубина вызова рекурсии равна n(где n — длина числовой последовательности). Таким образом, асимптотика по памяти для рекурсивной реализации будет $O(n)$, причем асимптотика для каждого отдельного вызова — $O(1)$.

В итеративной реализации единожды инициализируется целочисленная переменная. После чего в цикле больше не происходит инициализаций. Таким образом, асимптотика по памяти для итеративной реализации будет $O(1)$.

4.3 Оценка трудоемкости алгоритмов

Модель оценки в условных единицах:

1) операции стоимостью 1 — >(проверка: больше ли),==(проверка на равенство);

2) операции стоимостью 2 — %(нахождение остатка при делении);

Рассмотрим трудоемкость алгоритмов для последовательности длины n. Для обоих алгоритмов она будет одинаково вычисляться.

Рекурсивный и итеративный алгоритм

Если номер члена последовательности меньше n: $C = 1 + 3$.

Если номер члена последовательности равен n: $C = 1$.

Таким образом общая теоретическая оценка трудоемкости для обоих алгоритмов будет описываться таким рекуррентным соотношением: $F(n) = F(n-1) + C = n \cdot C$(где C — константа).

На рисунке 6 представлен график зависимости времени работы каждого из алгоритмов относительно длины последовательности.

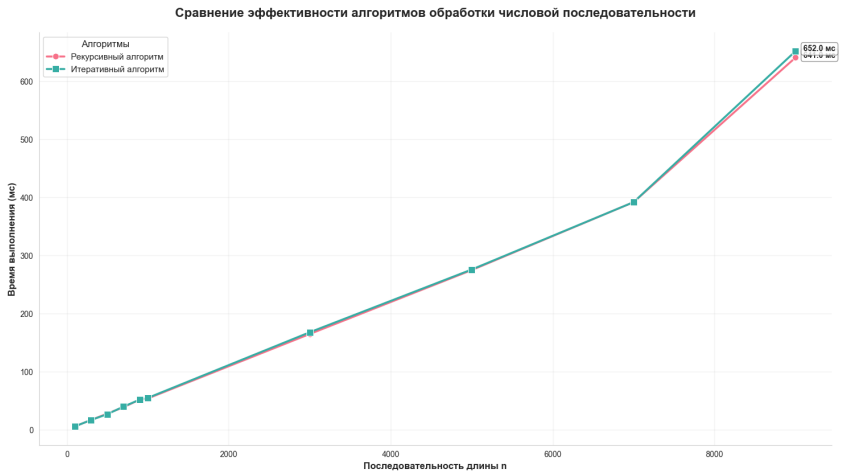


Рисунок 6 — График

4.4 Вывод

По теоретическим оценкам оба алгоритма одинаково оптимальны по времени. По результатам замеров времени чуть медленнее работает итеративная реализация. После же анализа затрат памяти было выявлено, что рекурсивная реализация затрачивает гораздо больше ресурсов.

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы была достигнута цель работы, заключавшаяся в проведении сравнительного анализа рекурсивного и нерекурсивного алгоритмов.

Также были выполнены все задачи:

1) разработать рекурсивный и нерекурсивный алгоритмы решения задачи, согласно варианту;

2) описаны средства разработки и инструменты замера процессорного времени выполнения реализации алгоритмов;

3) реализованы разработанные алгоритмы;

4) выполнено тестирование реализации алгоритмов;

5) выполнена теоретическая оценка затрачиваемой реализацией каждого алгоритма памяти(для рекурсивного алгоритма — на материале анализа высоты дерева рекурсивных вызовов и оценки затрачиваемой на один вызов функции памяти);

6) выполнены замеры процессорного времени выполнения реализации алгоритмов в зависимости от варьируемого входа(одна точка на графике получается делением времени выполнения k идентичных расчетов на k, k >= 100);

7) сделаны выводы из сравнительного анализа реализации рекурсивного и нерекурсивного алгоритмов решения одной и той же задачи, заданной вариантом, по критериям ёмкостной эффективности(на материале теоретической оценки пиковой временной эффективности и на материале результатов измерений).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кормен, Т. Алгоритмы: построение и анализ = Introduction to Algorithms / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн ; пер. с англ. под ред. И. В. Красикова. — 3-е изд. — Москва : Вильямс, 2022. — 1328 с. — ISBN 978-5-8459-2087-5.

ПРИЛОЖЕНИЕ А

В листинге 1 приведена программная реализация поставленных задач.

Листинг 1 — Программная реализация

```
#include <chrono>
#include <iostream>
#include <fstream>
#include <vector>
#include <random>
#include <sstream>

using namespace std;
using namespace chrono;

struct InputError : public exception {
        string message;
        InputError(const string& msg) : message(msg) {}
        const char* what() const noexcept override {
                return message.c_str();
        }
};

struct FileError : public exception {
        string message;
        FileError(const string& msg) : message(msg) {}
        const char* what() const noexcept override {
                return message.c_str();
        }
};

struct IntTypeError : public exception {
        string message;
        IntTypeError(const string& msg) : message(msg) {}
        const char* what() const noexcept override {
                return message.c_str();
        }
};

void recursive() {
        int number;
        cin >> number;
        if (cin.fail()) {
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                throw InputError("Ошибка: неправильный ввод");
        }
        else if (number < 0) {
                throw IntTypeError("Ошибка: введено отрицательное число");
```

```
        }
        else if (number > 0) {
                if (number % 2 == 1) {
                        cout << number << endl;
                }
                recursive();
        }
}

void classic() {
        int number;
        while (1) {
                cin >> number;
                if (cin.fail()) {
                        cin.clear();
                        cin.ignore(numeric_limits<streamsize>::max(), '\n');
                        throw InputError("Ошибка: неправильный ввод");
                }
                else if (number == 0) {
                        break;
                }
                else if (number < 0) {
                        throw IntTypeError("Ошибка: введено отрицательное
   число");
                }
                else if (number % 2 == 1) {
                        cout << number << endl;
                }
        }
}

void benchmark() {
        const int NUM_TESTS = 100;
        const int SEQUENCE_LENGTH = 9000;

        ofstream file("data.txt", ios::app);
        if (!file.is_open()) {
                throw FileError("Ошибка: не удалось открыть файл");
        }

        random_device rand;
        mt19937 generate(rand());
        uniform_int_distribution<int> dist(1, 1000);

        vector<long long> recursive_times;
        vector<long long> classic_times;

        cout << "Количество тестов: " << NUM_TESTS << endl;
        cout << "Длина последовательности: " << SEQUENCE_LENGTH+1 << endl;

        for (int test = 0; test < NUM_TESTS; ++test) {
                vector<int> test_data;
```

```
        for (int i = 0; i < SEQUENCE_LENGTH; ++i) {
                test_data.push_back(dist(generate));
        }
        test_data.push_back(0);

        {
                auto orig_cin = cin.rdbuf();

                stringstream test_stream;
                for (int num : test_data) {
                        test_stream << num << " ";
                }

                cin.rdbuf(test_stream.rdbuf());

                auto start = high_resolution_clock::now();
                recursive();
                auto end = high_resolution_clock::now();

                cin.rdbuf(orig_cin);

                duration<double, milli> time = end - start;
                recursive_times.push_back(time.count());
        }

        {
                auto orig_cin = cin.rdbuf();

                stringstream test_stream;
                for (int num : test_data) {
                        test_stream << num << " ";
                }

                cin.rdbuf(test_stream.rdbuf());

                auto start = high_resolution_clock::now();
                classic();
                auto end = high_resolution_clock::now();

                cin.rdbuf(orig_cin);

                duration<double, milli> time = end - start;
                classic_times.push_back(time.count());
        }
}

long long recursive_avg = 0, classic_avg = 0;
for (int i = 0; i < NUM_TESTS; ++i) {
        recursive_avg += recursive_times[i];
        classic_avg += classic_times[i];
}
recursive_avg /= NUM_TESTS;
```

```
        classic_avg /= NUM_TESTS;

        cout << "Результаты замера времени" << endl;
        cout << "Рекурсивная версия: " << recursive_avg << "мс()" << endl;
        cout << "Классическая версия: " << classic_avg << "мс()" << endl;
        file << recursive_avg << " " << classic_avg << endl;
}

int main() {
        cout << "Выберите режим работы\n" << "Введите соответствующий
   номер" << endl;
        cout << "1) Пользовательский ввод последовательностей\n" << "2)
   Автоматический массированный замер времени работы алгоритмов" << endl;

        try {
                int n;
                cin >> n;
                if (n == 1) {
                        cout << "Введите последовательность натуральных
   чисел, завершающуюся 0" << endl;

                        cout << "Рекурсивная версия:" << endl;
                        recursive();

                        cout << "Введите последовательность еще раз" <<
   endl;
                        cout << "Классическая версия:" << endl;
                        classic();
                }
                else if (n == 2) {
                        benchmark();
                }
                else {
                        throw InputError("Ошибка: неправильный ввод");
                }
        }
        catch (const InputError& e) {
                cerr << e.what() << endl;
                return 1;
        }
        catch (const IntTypeError& e) {
                cout << e.what() << endl;
                return 1;
        }
        catch (const exception& e) {
                cerr << "Неизвестная ошибка: " << e.what() << endl;
                return 1;
        }
}
```

В листинге 2 приведена вспомогательная программная реализация для

получения графика зависимости времени работы каждого из алгоритмов относительно длин последовательностей.

Листинг 2 — Вспомогательная программная реализация на языке программирования Python

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

def graphics(data):
df = pd.DataFrame(data)

df_melted = df.melt(id_vars='sequence_length',
var_name='Алгоритм',
value_name='Время мс()')

sns.set_style("whitegrid")
sns.set_palette("husl", 2)

plt.figure(figsize=(12, 8))

lineplot = sns.lineplot(data=df_melted,
x='sequence_length',
y='Время мс()',
hue='Алгоритм',
style='Алгоритм',
markers={'Рекурсивный алгоритм': 'o', 'Итеративный алгоритм': 's'},
dashes=False,
markersize=8,
linewidth=2.5)

plt.title('Сравнение эффективности алгоритмов обработки числовой
   последовательности',
fontsize=16, fontweight='bold', pad=20)
plt.xlabel('Последовательность длины n', fontsize=12, fontweight='bold')
plt.ylabel('Время выполнения мс()', fontsize=12, fontweight='bold')

plt.legend(title='Алгоритмы', title_fontsize=12, fontsize=11)

plt.grid(True, alpha=0.3)

last_size = df['sequence_length'].iloc[-1]
for algo in ['Рекурсивный алгоритм', 'Итеративный алгоритм']:
last_point = df[algo].iloc[-1]
plt.annotate(f'{last_point:.1f} мс',
xy=(last_size, last_point),
xytext=(10, 0),
textcoords='offset points',
fontsize=10,
fontweight='bold',
```

```
bbox=dict(boxstyle="round,pad=0.3",
facecolor="white",
alpha=0.8,
edgecolor='gray'))

sns.despine()
plt.tight_layout()

plt.show()

data_1 = {
        'sequence_length': [],
        'Рекурсивный алгоритм': [],
        'Итеративный алгоритм': []
}

with open('data.txt', 'r') as f:
plus = 200
i = 100
while i <= 10**3:
a = f.readline().split()

data_1['sequence_length'].append(i)
data_1['Рекурсивный алгоритм'].append(float(a[0]))
data_1['Итеративный алгоритм'].append(float(a[1]))
i += plus
plus = 2000
i = 1000
while i <= 10**4:
a = f.readline().split()

data_1['sequence_length'].append(i)
data_1['Рекурсивный алгоритм'].append(float(a[0]))
data_1['Итеративный алгоритм'].append(float(a[1]))
i += plus

graphics(data_1)
```