



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 «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 3 по дисциплине «Анализ алгоритмов»

Тема Графовые модели алгоритмов

Студент Ильченко Е. А.

Группа ИУ7-54Б

Преподаватели Волкова Л.Л.

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 Аналитическая часть	6
1.1 Рекурсия	6
1.1.1 Определение рекурсии	6
1.2 Графовые модели программ	6
1.2.1 Типы отношений в графовых моделях	6
1.2.2 Основные модели программ	7
1.3 Параллелизм в алгоритмах и программах	7
1.3.1 Закон Амдала	7
2 Конструкторская часть	9
2.1 Требования к реализации	9
2.2 Разработка алгоритмов	9
3 Технологическая часть	13
3.1 Средства реализации	13
3.2 Реализация алгоритмов	13
4 Исследовательская часть	14
4.1 Граф управление	14
4.1.1 Нерекursивный алгоритм	14
4.1.2 Рекурсивный алгоритм	14
4.2 Информационный граф	14
4.2.1 Нерекursивный алгоритм	14
4.2.2 Рекурсивный алгоритм	15
4.3 Операционная история	15
4.3.1 Нерекursивный алгоритм	15
4.3.2 Рекурсивный алгоритм	15
4.4 Информационная история	16
4.4.1 Нерекursивный алгоритм	16

4.4.2	Рекурсивный алгоритм	16
4.5	Распараллеливание	16
4.5.1	Нерекурсивный алгоритм	16
4.5.2	Рекурсивный алгоритм	16
ЗАКЛЮЧЕНИЕ		18
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ		19

ВВЕДЕНИЕ

Целью данной работы является на материале графовых моделей алгоритмов выделить участки программ, которые могут быть исполнены параллельно.

Для достижения поставленной цели необходимо решить следующие задачи:

- описать два алгоритма, согласно индивидуальному варианту, – рекурсивный и нерекурсивный;
- описать реализации двух алгоритмов четырьмя графовыми моделями – графом управления, информационным графом, операционной историей, информационной историей;
- указать участки каждой программы, которые могут быть исполнены параллельно, или отсутствие таковых.

В рамках данной работы рассматривается задача подсчёта количества единиц в последовательности из нулей и единиц, завершающейся числом 2. Анализируется последовательный код. В данной работе не требуется разрабатывать код, решающий задачу в параллельном режиме.

1 Аналитическая часть

1.1 Рекурсия

1.1.1 Определение рекурсии

Рекурсия – это метод определения функций или решения задач, при котором функция вызывает саму себя непосредственно или через другие функции [1].

Рекурсивная функция f определяется через [1, 2]:

- **базовый случай** (терминальное условие): $f(x_0) = c$, где x_0 – терминальное значение;
- **рекурсивный случай**: $f(x) = g(x, f(h(x)))$, где $h(x)$ – функция, уменьшающая задачу.

Корректность рекурсивного алгоритма обеспечивается выполнением условий [1]:

- **существование базового случая**: функция завершается при обнаружении терминатора 2 или пустой последовательности;
- **сходимость**: на каждом шаге размер задачи уменьшается ($|rest(S)| < |S|$);
- **правильность**: результат вычисляется как сумма вклада текущего элемента и результата для оставшейся последовательности.

1.2 Графовые модели программ

Программы можно представлять с помощью ориентированных графов, состоящих из набора вершин и множества соединяющих их направленных дуг. В зависимости от уровня детализации, вершины могут представлять различные сущности [3]:

- если вершины соответствуют итерациям циклов, то каждая вершина представляет операторы тела цикла, выполненные на одной и той же итерации;
- если вершины соответствуют срабатываниям операторов, то каждая вершина представляет один из операторов тела цикла, выполненный на некоторой итерации.

1.2.1 Типы отношений в графовых моделях

Дуги в графовых моделях программ отражают связь (отношение) между вершинами. Выделяют два основных типа отношений:

- 1) **операционное отношение**: две вершины A и B соединяются направленной дугой тогда и только тогда, когда вершина B может быть выполнена сразу после вершины A . Операционное отношение называют отношением по передаче управления;
- 2) **информационное отношение**: две вершины A и B соединяются направленной дугой тогда и только тогда, когда вершина B использует в качестве аргумента некоторое значение, полученное в вершине A . Информационное отношение называют отношением по

передаче данных.

1.2.2 Основные модели программ

На основе различных комбинаций типов вершин и отношений выделяют четыре основные графовые модели программ [3]:

- 1) **граф управления программой**: вершины являются операторами, а дуги представляют операционные отношения. Данная модель отражает поток управления в программе;
- 2) **информационный граф программы**: вершины соответствуют операторам, а дуги представляют информационные отношения. Модель отражает зависимости по данным между операторами программы;
- 3) **операционная история программы**: вершины представляют срабатывания операторов, а дуги отражают операционные отношения между ними;
- 4) **информационная история программы**: вершины соответствуют срабатываниям операторов, а дуги представляют информационные отношения между ними.

1.3 Параллелизм в алгоритмах и программах

Для определения ресурса параллелизма в графе алгоритма используется ярусно параллельная форма графа алгоритма [4].

Свойства:

- 1) начальная вершина каждой дуги расположена на ярусе с номером меньшим, чем номер яруса конечной вершины;
- 2) между вершинами, расположенными на одном ярусе, не может быть дуг.

1.3.1 Закон Амдала

Закон Амдала – иллюстрирует ограничение роста производительности вычислительной системы с увеличением количества вычислителей [5].

$$\frac{T^1}{T^p} = S < \frac{1}{f + \frac{1-f}{p}}$$

где

- f – доля последовательных операций ($0 \leq f \leq 1$);
- p – число процессоров;
- T^1 – время работы программы на одном процессоре;
- T^p – время работы программы на системе из p процессоров.

Вывод

В аналитической части:

- представлено строгое математическое описание рекурсивного метода с выделением базового и рекурсивного случаев;
- рассмотрены графовые модели программ, включая четыре основных типа: граф управления, информационный граф, операционная история и информационная история программы;
- представлено определение ресурса параллелизма в графе алгоритма, используя ярусно параллельную форму графа алгоритма.

2 Конструкторская часть

2.1 Требования к реализации

К программе предъявлены ряд функциональных требований: входные данные – последовательность целых чисел (0, 1, 2), выходные данные – количество единиц в последовательности до первого вхождения числа 2.

К программе предъявлены ряд требований:

- наличие интерфейса для выбора действий;
- реализация рекурсивного алгоритма подсчёта единиц в последовательности;
- реализация нерекурсивного алгоритма подсчёта единиц в последовательности;
- наличие функциональности замера процессорного времени выполнения алгоритмов;
- поддержка ввода последовательности чисел с проверкой корректности данных.

Программа работает в трёх режимах:

- выполнение рекурсивного алгоритма;
- выполнение нерекурсивного алгоритма;
- замер процессорного времени выполнения алгоритмов.

При первом и втором режимах на вход поступает последовательность чисел (0, 1), завершающаяся числом 2, на выход – количество единиц в последовательности до первого вхождения числа 2.

При третьем режиме работы на вход поступают минимальный размер последовательности, максимальный размер, шаг изменения размера и число итераций k ($k \geq 100$), на выход – результаты замера времени в виде таблицы, где каждая точка получается делением времени выполнения k идентичных расчётов на k .

2.2 Разработка алгоритмов

Раздел содержит блок-схемы, описывающие следующие алгоритмы: рекурсивный алгоритм подсчёта единиц в последовательности на рисунке 2.1, нерекурсивный алгоритм подсчёта единиц в последовательности на рисунке 2.2.

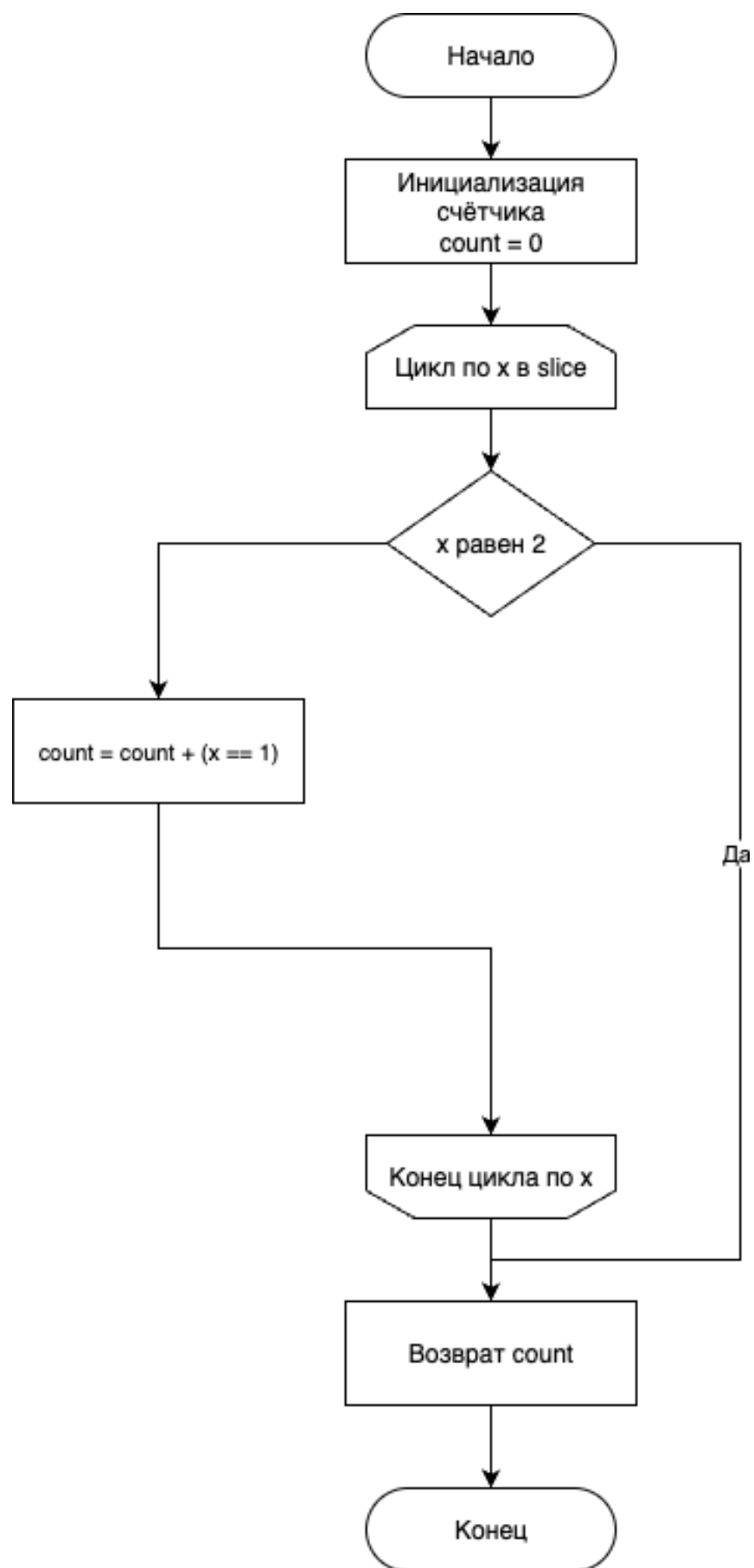


Рисунок 2.1 — Схема нерекурсивного алгоритма

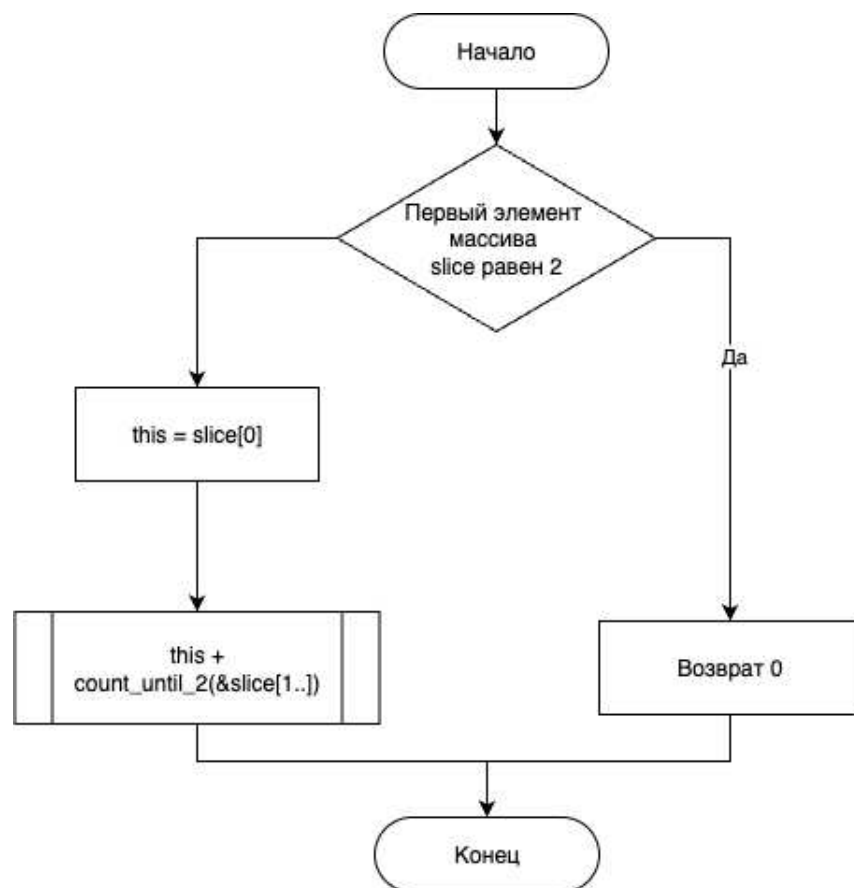


Рисунок 2.2 — Схема рекурсивного алгоритма

Вывод

В данном разделе были разработаны два алгоритма подсчёта единиц в последовательности: рекурсивный и нерекурсивный. Для каждого из них представлены блок-схемы, описывающие логику работы.

3 Технологическая часть

3.1 Средства реализации

Для реализации алгоритмов подсчёта единиц в последовательности был выбран язык программирования Rust, поскольку он соответствует требованиям лабораторной работы. Измерение времени выполнения алгоритмов осуществлялось с использованием структуры `Instant` из стандартной библиотеки `std::time`. Разработка программного обеспечения проводилась в интегрированной среде разработки `RustRover`.

3.2 Реализация алгоритмов

Были реализованы нерекурсивный алгоритм подсчёта единиц в последовательности (листинг 3.1), рекурсивный алгоритм подсчёта единиц в последовательности (листинг 3.2).

Листинг 3.1 — Нерекурсивный алгоритм

```
pub(crate) fn count_until_2(slice: &[i32]) -> usize {
    let mut count = 0; // 1
    for &x in slice { // 2
        if x == 2 { // 3
            break; // 4
        }
        count += (x == 1) as usize; // 5
    }
    count // 6
}
```

Листинг 3.2 — Рекурсивный алгоритм

```
pub(crate) fn count_until_2(slice: &[i32]) -> usize {
    if slice[0] == 2 { // 1
        0; // 2
    }
    let this = slice[0] as usize; // 3
    this + count_until_2(&slice[1..]) // 4
}
```

Вывод

В данном разделе были реализованы алгоритмы нерекурсивный алгоритм подсчёта единиц в последовательности и рекурсивный алгоритм подсчёта единиц в последовательности.

4 Исследовательская часть

4.1 Граф управление

4.1.1 Нерекурсивный алгоритм

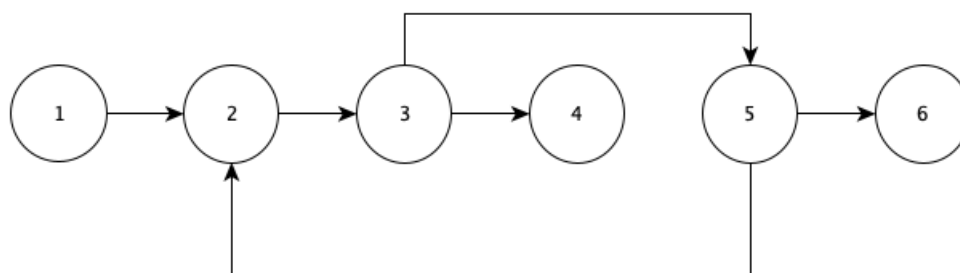


Рисунок 4.1 — Граф управления нерекурсивного алгоритма

4.1.2 Рекурсивный алгоритм

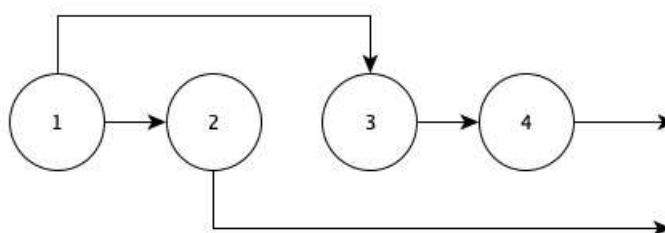


Рисунок 4.2 — Граф управления нерекурсивного алгоритма

4.2 Информационный граф

4.2.1 Нерекурсивный алгоритм

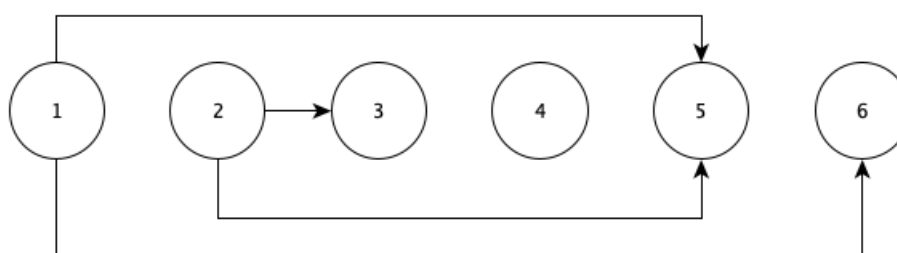


Рисунок 4.3 — Граф управления нерекурсивного алгоритма

4.2.2 Рекурсивный алгоритм

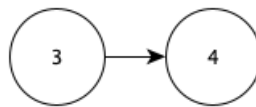


Рисунок 4.4 — Граф управления нерекурсивного алгоритма

4.3 Операционная история

4.3.1 Нерекурсивный алгоритм

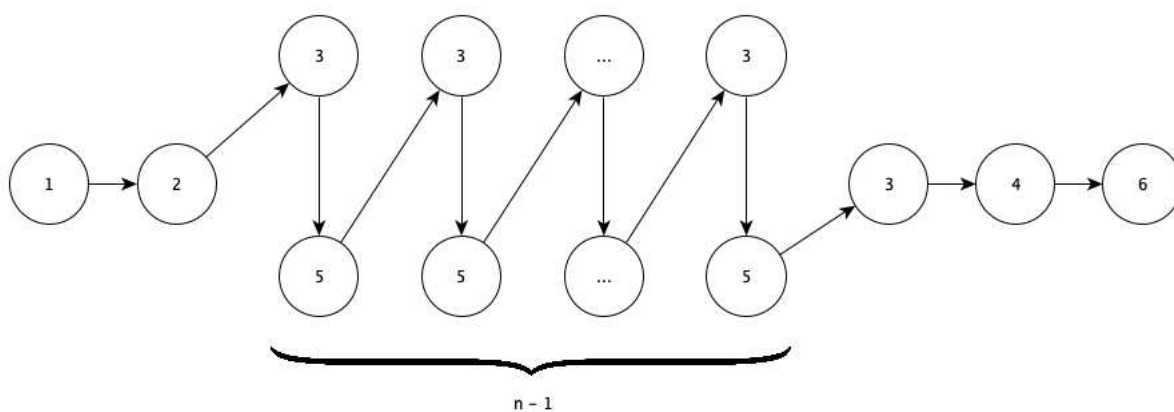


Рисунок 4.5 — Граф управления нерекурсивного алгоритма

4.3.2 Рекурсивный алгоритм

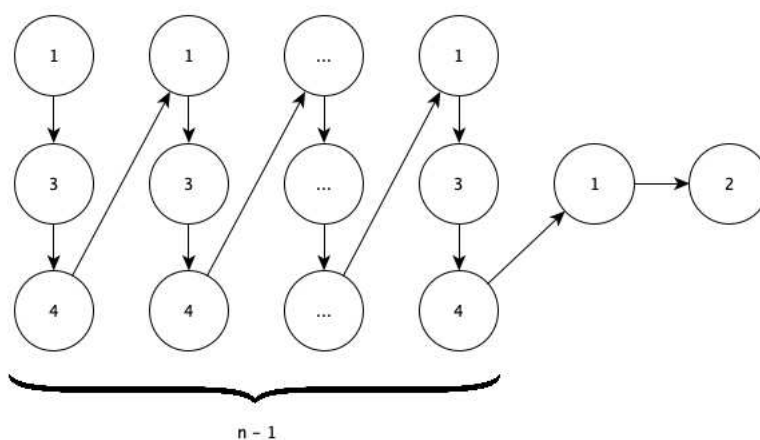


Рисунок 4.6 — Граф управления нерекурсивного алгоритма

4.4 Информационная история

4.4.1 Нерекursивный алгоритм

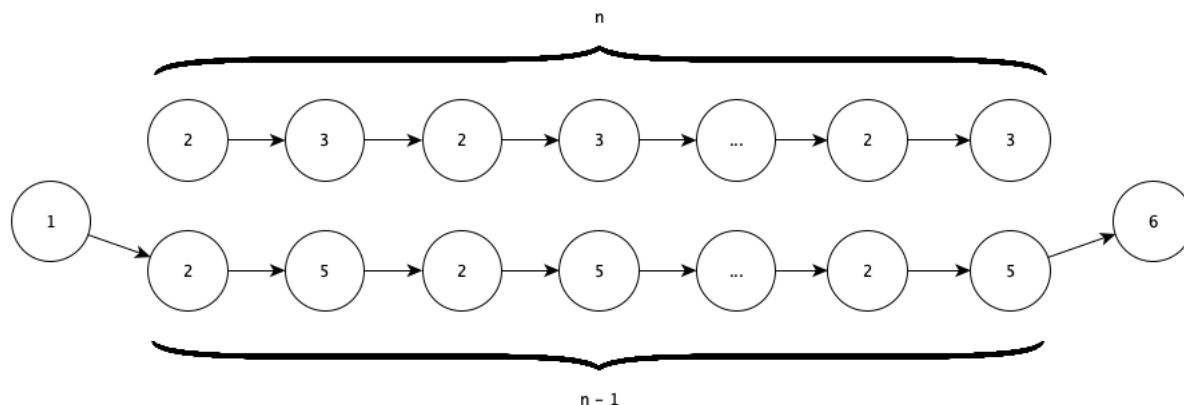


Рисунок 4.7 — Граф управления нерекурсивного алгоритма

4.4.2 Рекурсивный алгоритм

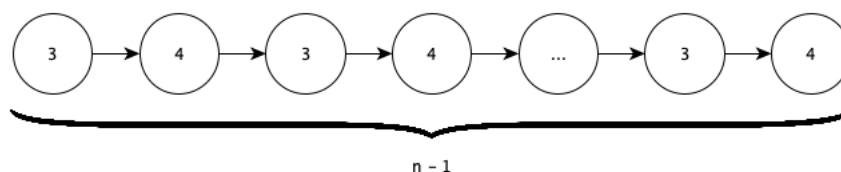


Рисунок 4.8 — Граф управления нерекурсивного алгоритма

4.5 Распараллеливание

4.5.1 Нерекursive алгоритм

Нерекursive алгоритм не получится сделать параллельным, поскольку каждая итерация цикла зависит от результатов предыдущих: обновление переменной `count` требует знания её текущего значения, а условие досрочного завершения `break` создаёт управляющую зависимость, которая делает невозможным предсказание количества выполняемых итераций. Таким образом, все итерации образуют строго последовательную цепочку.

4.5.2 Рекурсивный алгоритм

Рекурсивный алгоритм не получится сделать параллельным, поскольку каждый рекурсивный вызов зависит от результата предыдущего: вычисление суммы требует полного завершения

всех последующих рекурсивных вызовов. Таким образом, все вызовы образуют строго последовательную цепочку, где каждый шаг должен дождаться результата следующего перед тем как вернуть своё значение.

Вывод

В данном разделе были построены граф управления, информационный граф, операционная история и информационная история нерекурсивного и рекурсивного алгоритмов. Была проанализирована возможность параллельных вычислений для нерекурсивного и рекурсивного алгоритмов.

ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы была успешно достигнута основная цель.

Все поставленные задачи выполнены в полном объёме:

- разработаны рекурсивный и нерекурсивный алгоритмы решения задачи;
 - описаны реализации двух алгоритмов четырьмя графовыми моделями – графом управления, информационным графом, операционной историей, информационной историей;
 - указаны участки каждой программы, которые могут быть исполнены параллельно.
- Было выявлено, что оба алгоритма сделать параллельным нельзя.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Головешкин В.А., Ульянов М.В. Теория рекурсии для программистов. – М.: ФИЗМАТЛИТ, 2006. – 296 с.
2. Баррон Д. Рекурсивные методы в программировании. – М.: Мир, 1974. – 79 с.
3. Федотов И.Е. Параллельное программирование. Модели и приёмы / И.Е. Федотов. – Москва : Солон-пресс, 2018. - 390 с.
4. Ахо А.В., Ульман Дж.Д. Теория синтаксического анализа, перевода и компиляции. Том 1. – М.: Мир, 1978. – 612 с.
5. Amdahl G.M. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities // AFIPS Conference Proceedings. – 1967. – Vol. 30. – P. 483-485.