**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** Жаринов М. А.

**Группа** ИУ7-52Б

**Преподаватели** Волкова Л. Л., Строганов Д. В.

Москва, 2025

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Цель данной лабораторной работы — на материале графовых моделей алгоритмов выделить участки программ, которые могут быть исполнены параллельно. Для достижения данной цели необходимо выполнить задачи:

1) описать два алгоритма вычисления среднего значения элементов последовательности, окачивающейся нулём, — рекурсивный и нерекурсивный;

2) описать реализации двух алгоритмов четырьмя графовыми моделями — графом управления, информационным графом, операционной историей, информационной историей;

3) указать участки каждой программы, которые могут быть исполнены параллельно, или отсутствие таковых.

# 1 Аналитическая часть

## 1.1 Основные определения

*Рекурсия* — способ описания функций или процессов через самих себя. *Рекурсивная функция* — подпрограмма, которая обращение к самой себе. *Хвостовая рекурсия* — рекурсия, при которой рекурсивный вызов является последней операцией перед возвратом из функции. Хвостовая рекурсия позволяет компилятору выполнить оптимизацию хвостового вызова. Вместо добавления нового кадра стека для каждого рекурсивного вызова, компилятор может переиспользовать текущий кадр, эффективно превращая рекурсию в цикл [1].

*Граф* — математический объект, состоящий из двух множеств. Одно из них — любое конечное множество, его элементы называются графа. Другое множество состоит из пар вершин, эти пары называются графа. Если множество вершин графа обозначено буквой $V$, множество рёбер — буквой $E$, а сам граф — буквой $V$, то граф можно обозначить как $G = (V, E)$. Если рёбра — упорядоченные пары, то такой граф называется *ориентированным* (сокращённо орграф), если же неупорядоченные, то *неориентированным*. Ребро $(a, b)$ соединяет вершины $a$ и $b$. В графе может быть не более одного ребра, соединяющего две данные вершины. Ребро типа $(a, a)$, то есть соединяющее вершину с ней же самой, называют петлей [2].

## 1.2 Графовые модели программ

Между действиями программы устанавливаются два типа отношений. Если одно действие выполняется непосредственно за другим, то между двумя этими действиями установлена *связь по управлению* или *операционная связь*. Если одно действие использует в качестве аргументов результаты выполнения другого действиями, то между ними установлена *информационная связь*. Оба типа отношений в общем случае вводят на множестве действий частичный порядок [3].

*Граф управления* — это ориентированный граф, построенный по исходному тексту программы следующим образом: каждой операторной конструкции программы ставится в соответствие вершина графа, между вершинами проводится направленная дуга от вершины, соответствующей предшественнику, к вершине, соответствующей последователю, если между ними установлена операционная связь. Граф управления полностью определяется структурой программы и не зависит от её входных данных: для каждой программы множества вершин и дуг фиксированы и однозначно задают единственный граф [3].

*Операционная история* — это ориентированный граф, построенный на основе выполнения программы при заданных входных данных, в котором каждое срабатывание оператора (не обязательно уникальное) представлено отдельной вершиной, а дуги отражают непосредственную последовательность срабатываний операторов; такой граф представляет собой единственный путь от начальной вершины к конечной и полностью определяется конкретными входными данными [3].

*Информационный граф* — это ориентированный граф, вершины которого соответствуют операторам исходной программы, а дуги отражают теоретически возможные отношения информационной связи между ними; данный граф не зависит от конкретных входных данных и представляет собой статическую модель программы, в которой могут присутствовать дуги, не реализующиеся при фактическом выполнении [3].

*Информационная история* — это ориентированный граф, в котором каждое срабатывание оператора при выполнении программы на заданном наборе входных данных фиксируется отдельной вершиной, а дуги соединяют эти вершины в порядке передачи информации между последовательно выполняемыми операторами; такой граф представляет конкретный путь выполнения программы и полностью определяется входными данными [3].

## Вывод

В аналитической части были рассмотрены определения рекурсии, рекурсивной функции, графов и графовых моделей алгоритмов.

# 2 Конструкторская часть

## 2.1 Разработка алгоритмов

На рисунке 2.1 показана схема итеративного алгоритма вычисления среднего значения элементов последовательности.

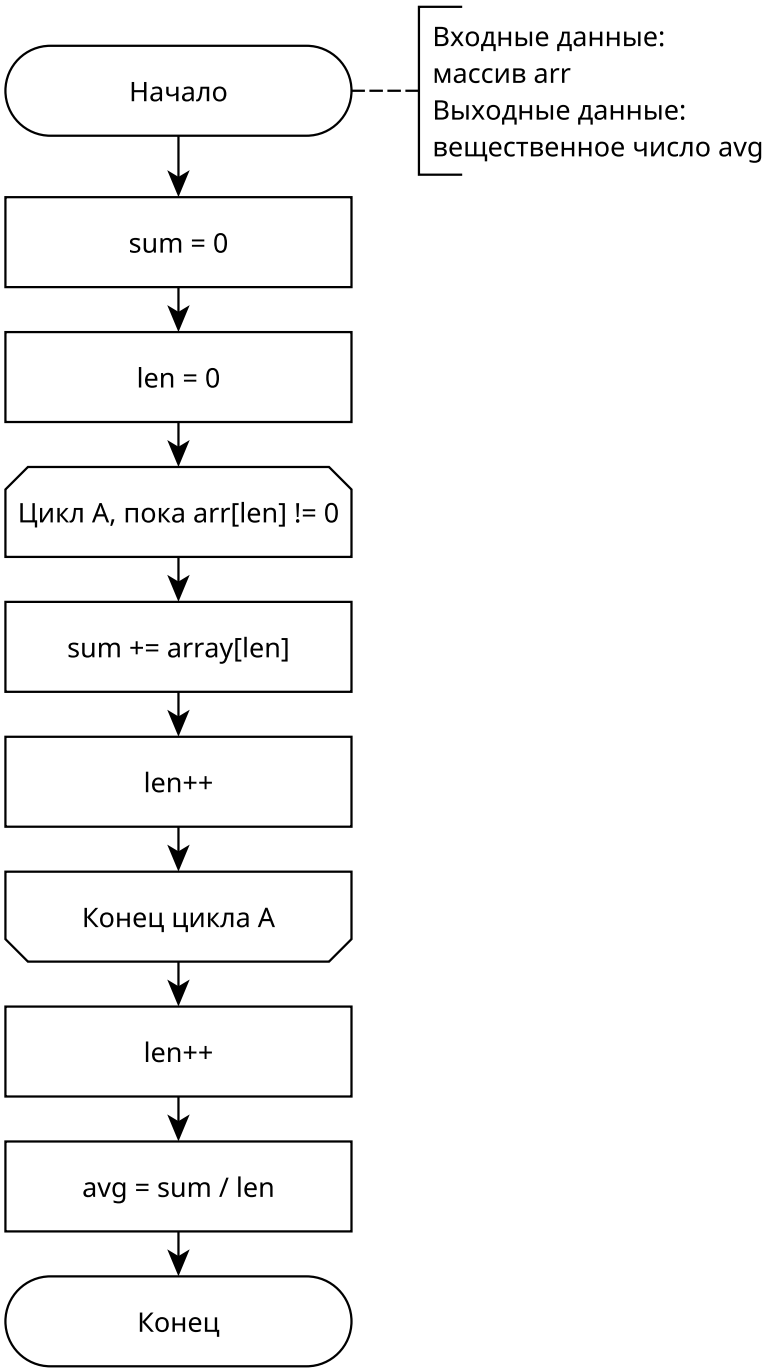


Рисунок 2.1 — Схема итеративного алгоритма вычисления среднего значения элементов последовательности

Схема рекурсивного алгоритма вычисления среднего значения элементов последовательности показана на рисунке 2.2.

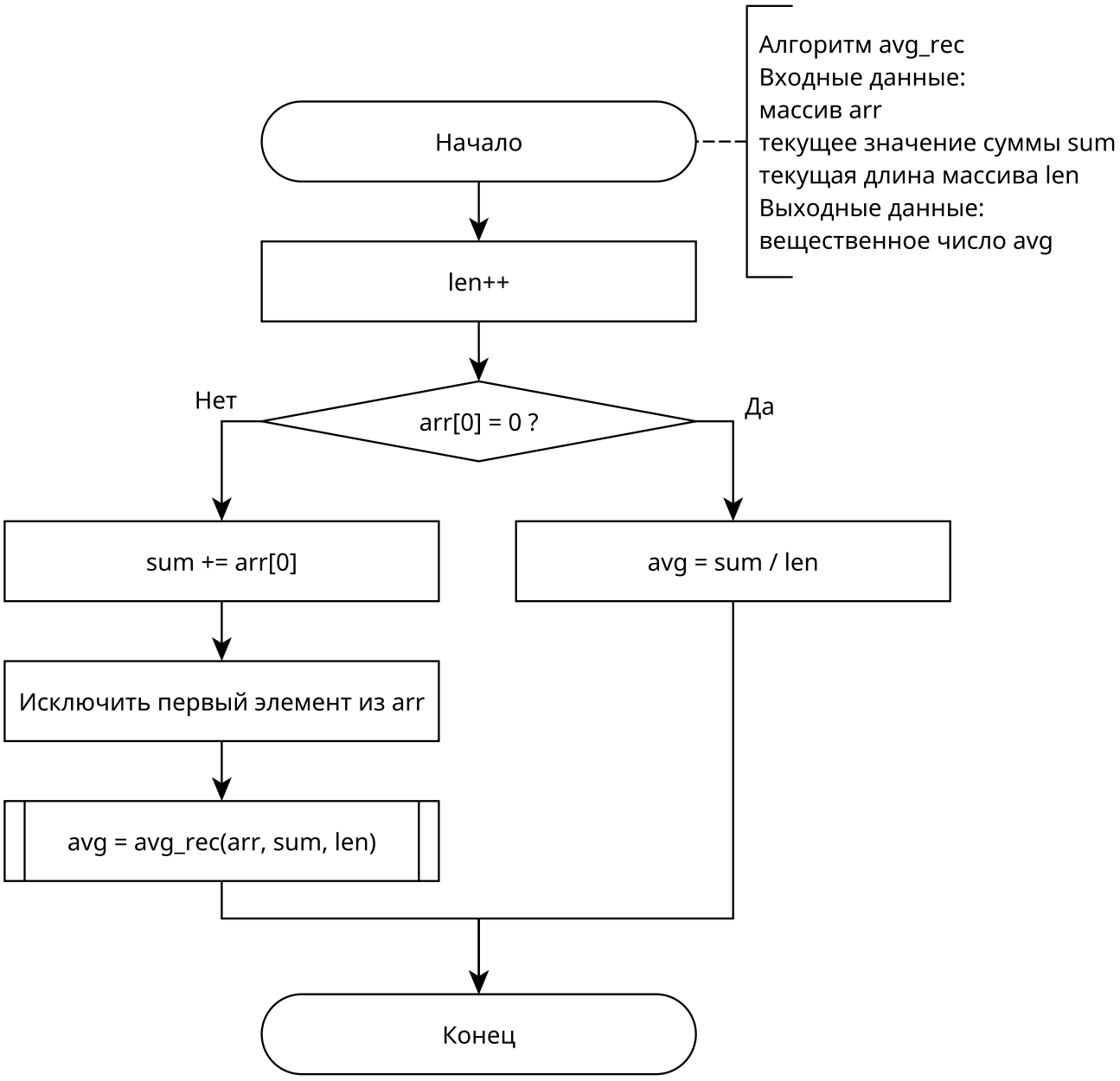


Рисунок 2.2 — Схема рекурсивного алгоритма вычисления среднего значения элементов последовательности

## Вывод

В конструкторской части были описаны итеративный и рекурсивный алгоритмы вычисления среднего значения элементов последовательности.

# 3 Технологическая часть

Для реализации алгоритмов был выбран язык C.

## 3.1 Реализации алгоритмов

В листинге 3.1 представлен исходный код реализации итеративного алгоритма вычисления среднего значения элементов последовательности.

```
double calc_avg_iter(int *array) {
  long long sum = 0;
  size_t len = 0;
  while (array[len] != 0) {
    sum += array[len];
    len++;
  }
  len++;
  double avg = (double)sum / (double)len;
  return avg;
}
```

Листинг 3.1 — Реализация итеративного алгоритма вычисления среднего значения

В листинге 3.2 представлен исходный код реализации рекурсивного алгоритма вычисления среднего значения элементов последовательности.

```
double calc_avg_rec(int *array, long long *sum, size_t *len) {
  double avg;
  (*len)++;
  if (*array == 0) {
    avg = (double)*sum / (double)*len;
    return avg;
  }
  *sum += *array;
  array++;
  avg = calc_avg_rec(array, sum, len);
  return avg;
}
```

Листинг 3.2 — Реализация рекурсивного алгоритма вычисления среднего значения

## Вывод

В технологической части определены необходимые средства реализации, и с их помощью реализованы алгоритмы вычисления среднего значения элементов последовательности.

# 4 Исследовательская часть

Для реализаций алгоритмов по листингам 3.1 и 3.2 построены 4 графовые модели:

1) граф управления;
2) информационный граф;
3) операционная история;
4) информационная история.

Для обозначения вершин используются номера строк из листингов.

## 4.1 Графовые модели для реализации итеративного алгоритма

Граф управления для реализации итеративного алгоритма приведён на рисунке 4.1.

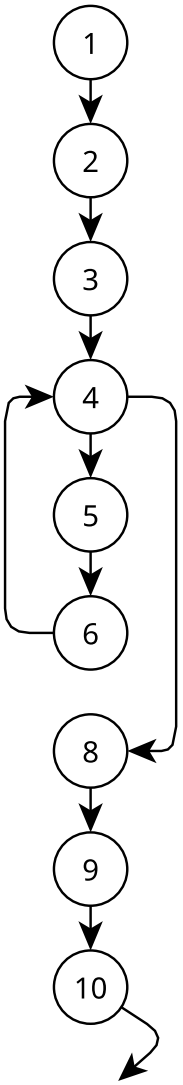


Рисунок 4.1 — Граф управления для реализации итеративного алгоритма

Информационный граф для реализации итеративного алгоритма приведён на рисунке 4.2.

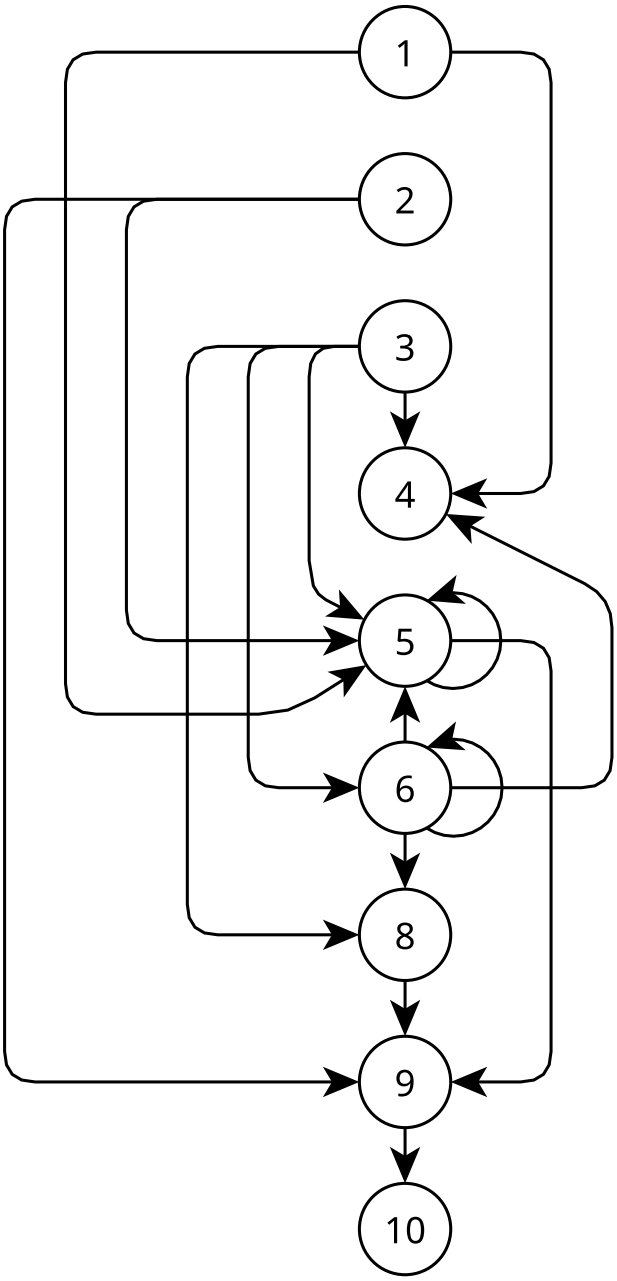


Рисунок 4.2 — Информационный граф для реализации итеративного алгоритма

Операционная история для реализации итеративного алгоритма приведена на рисунке 4.3.

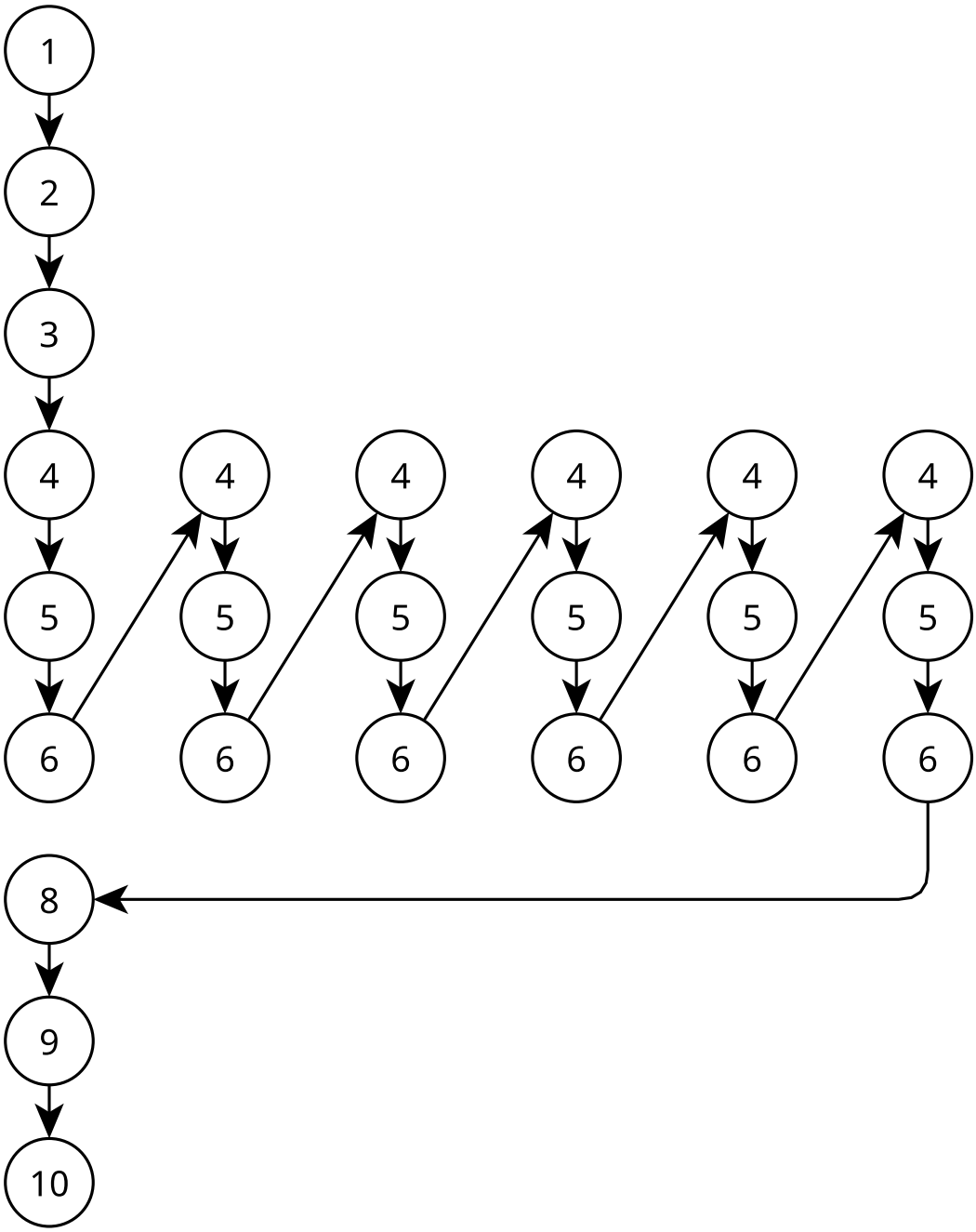


Рисунок 4.3 — Операционная история для реализации итеративного алгоритма

Информационная история для реализации итеративного алгоритма приведена на рисунке 4.4.

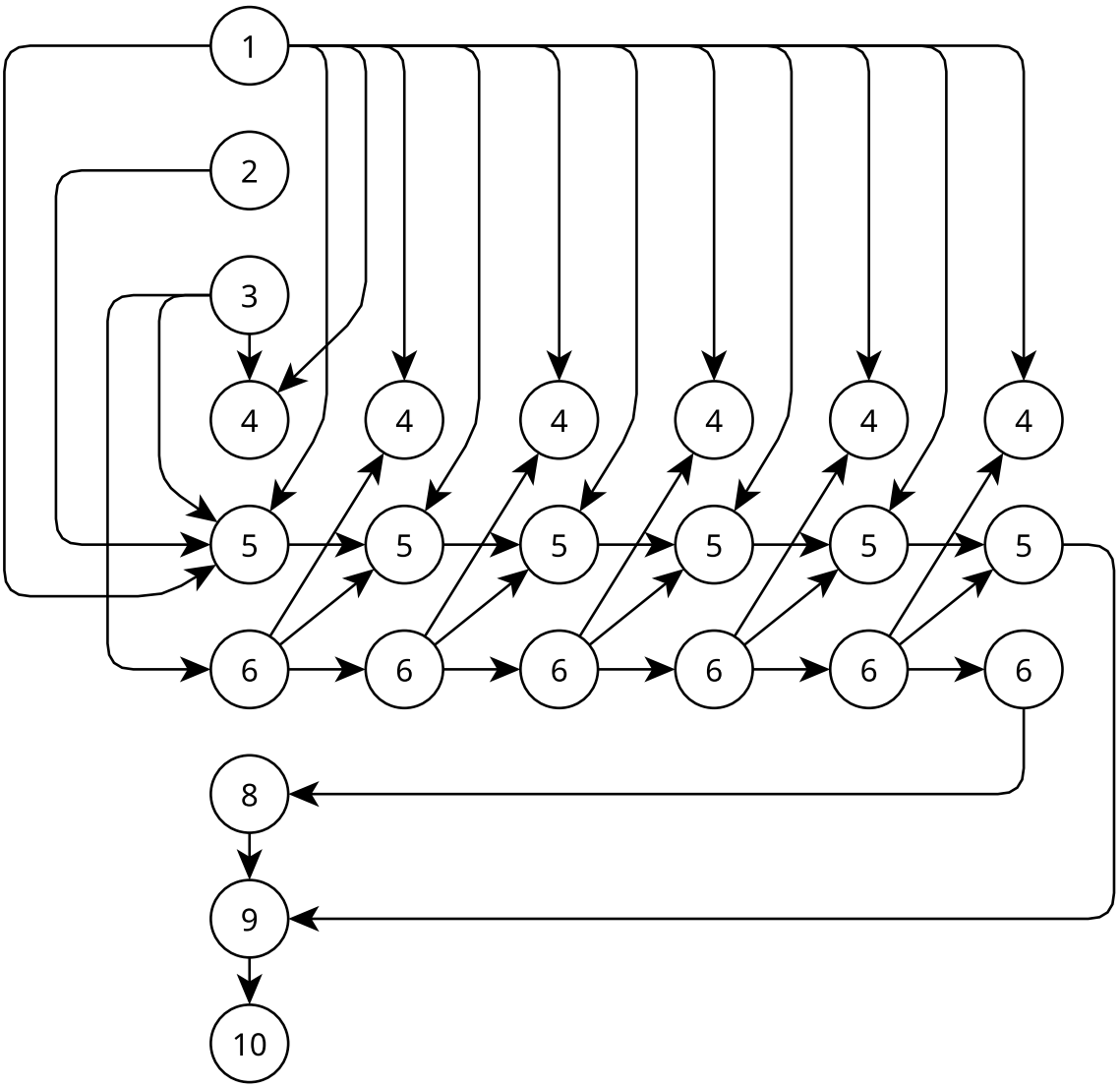


Рисунок 4.4 — Информационная история для реализации итеративного алгоритма

Анализ графовых моделей для рекурсивного алгоритма показывает, что данная реализация алгоритма не может быть выполнена параллельно, так как каждое действие увеличения суммы (вершина 5) информационно зависит от предыдущего.

## 4.2 Графовые модели для реализации рекурсивного алгоритма

Вершина 10 используется для обозначения рекурсивного вызова, а 10б — для обозначения присваивания его результата переменной.

Граф управления для реализации рекурсивного алгоритма приведён на рисунке 4.5.

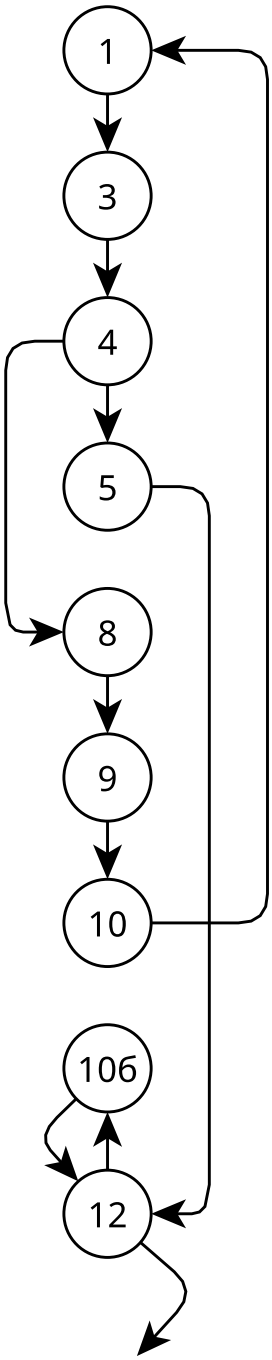


Рисунок 4.5 — Граф управления для реализации рекурсивного алгоритма

Информационный граф для реализации рекурсивного алгоритма приведён на рисунке 4.6.

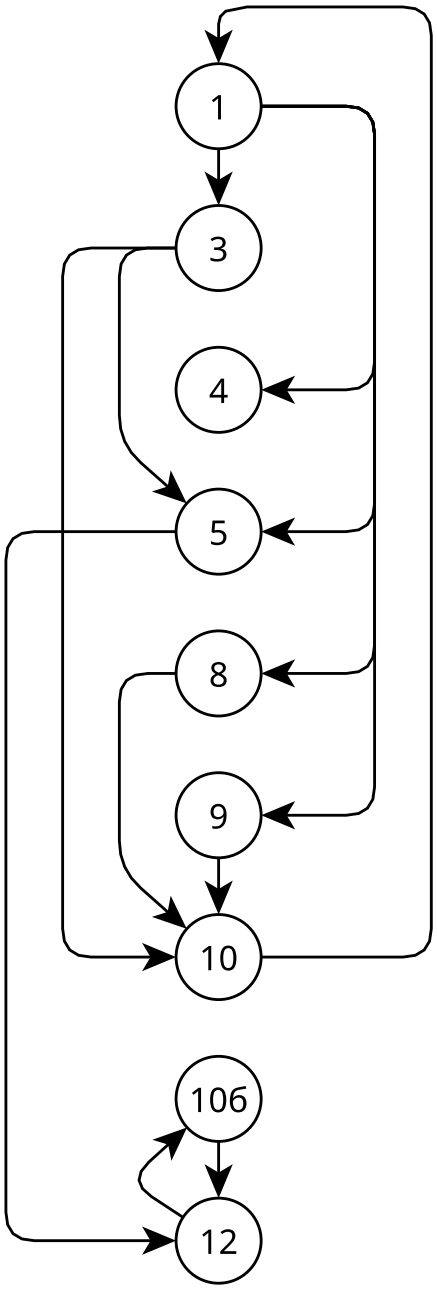


Рисунок 4.6 — Информационный граф для реализации рекурсивного алгоритма

Операционная история для реализации рекурсивного алгоритма приведена на рисунке 4.7.

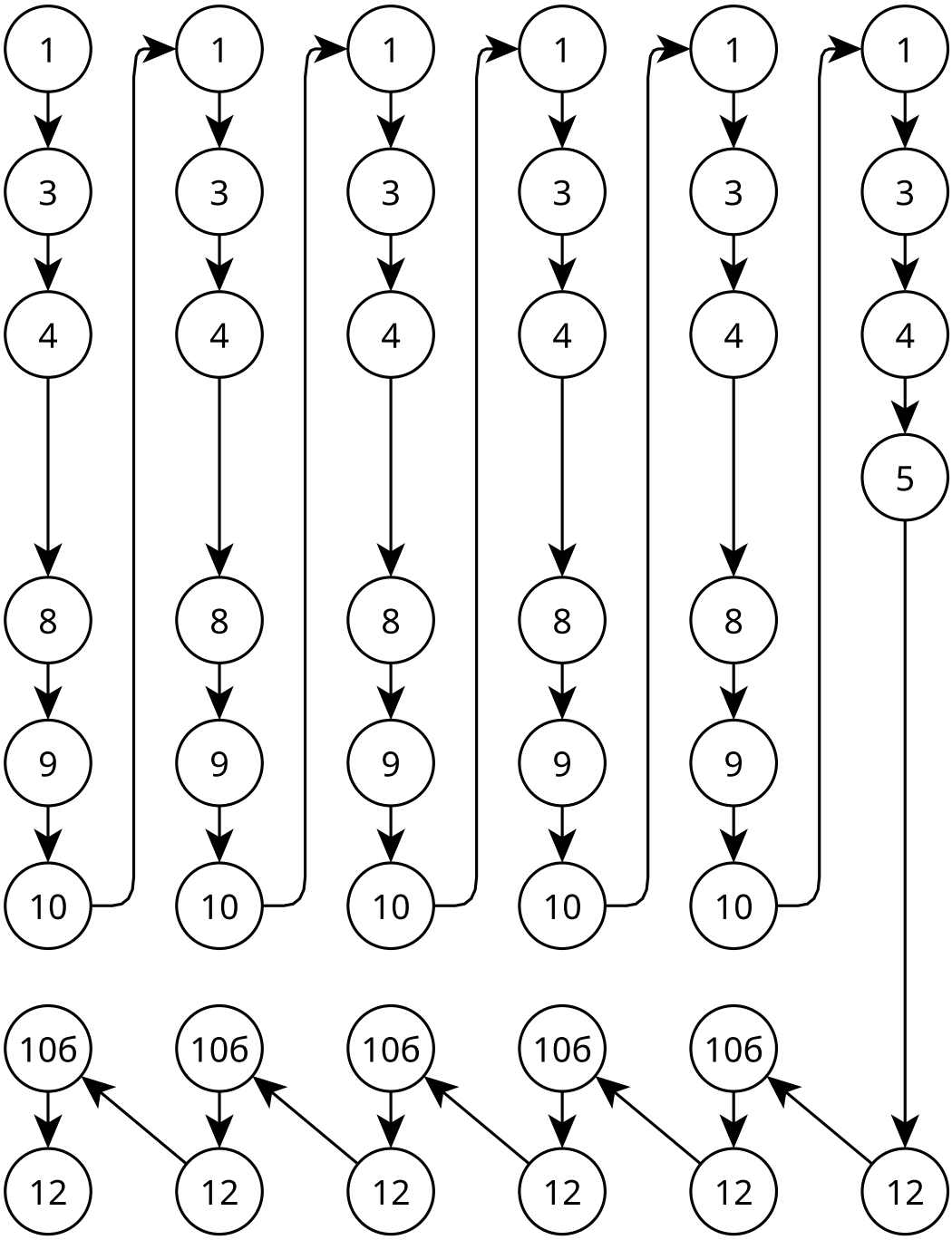


Рисунок 4.7 — Операционная история для реализации рекурсивного алгоритма

Информационная история для реализации рекурсивного алгоритма приведена на рисунке 4.8.

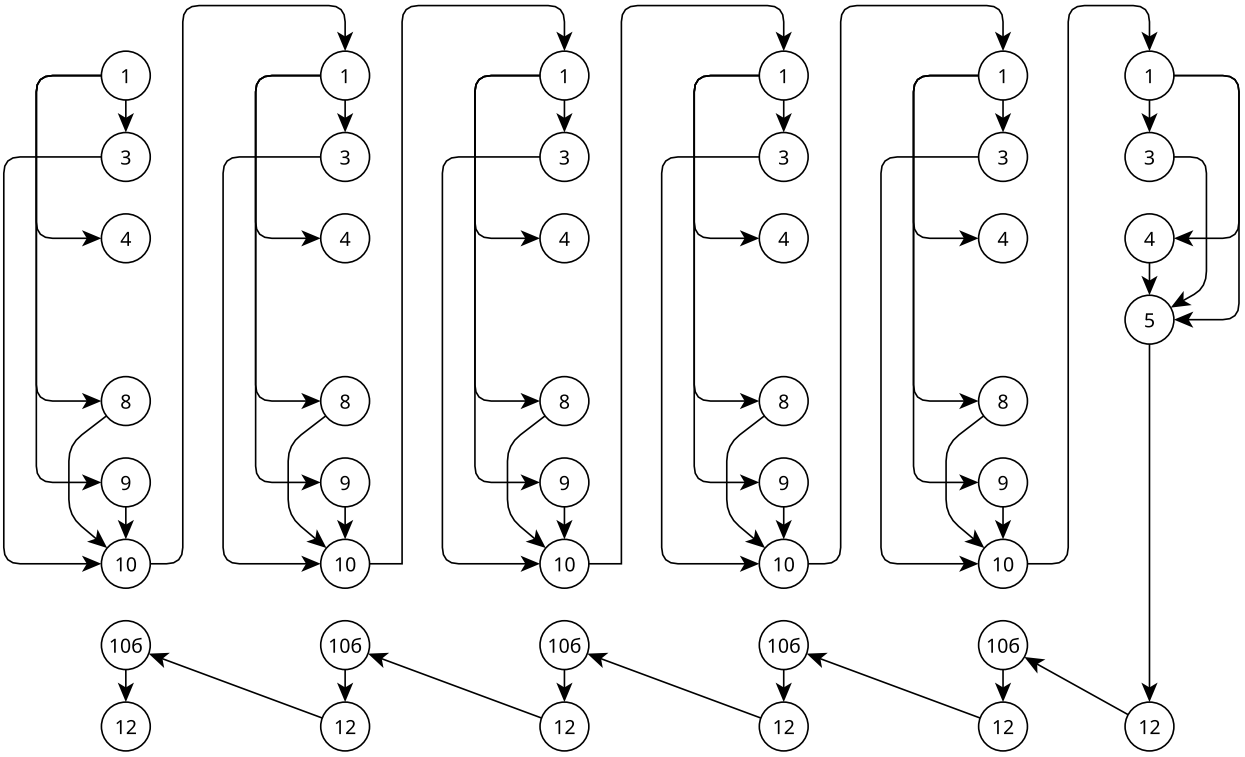


Рисунок 4.8 — Информационная история для реализации рекурсивного алгоритма

Анализ графовых моделей для рекурсивного алгоритма показывает, что данная реализация алгоритма не может быть выполнена параллельно, так как отсутствуют участки кода, не связанные друг с другом информационно и операционно. Так, несмотря на то, что вершина 9 не связана с вершиной 8 информационно, код вершины 8 обязательно должен быть исполнен до вершины 9, так как он использует значение переменной `array`, которое изменяется в вершине 9.

## Вывод

В исследовательской части были описаны реализации двух алгоритмов вычисления среднего значения элементов последовательности четырьмя графовыми моделями — графом управления, информационным графом, операционной историей, информационной историей и указано отсутствие участков каждой программы, которые могут быть исполнены параллельно.

# ЗАКЛЮЧЕНИЕ

Цель данной лабораторной работы была достигнута: на материале графовых моделей алгоритмов показано отсутствие участков программ, которые могут быть исполнены параллельно. Все задачи решены:

1) описаны два алгоритма вычисления среднего значения элементов последовательности, окачивающейся нулём, — рекурсивный и нерекурсивный;

2) описаны реализации двух алгоритмов четырьмя графовыми моделями — графом управления, информационным графом, операционной историей, информационной историей;

3) указано отсутствие участков каждой программы, которые могут быть исполнены параллельно.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рекурсивные методы в программировании / Д. Баррон — Москва: Мир, 2003 - 79 с.

2. ТЕОРИЯ ГРАФОВ: Учебное пособие / В. Е. Алексеев, Д. В. Захарова — Нижний Новгород: Нижегородский госуниверситет, 2017 - 119 с.

3. Параллельные вычисления / В. В. Воеводин, Вл. В. Воеводин — СПб.: БХВ-Петербург, 2002. - 608 с.