Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное 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 №7
по дисциплине «Анализ алгоритмов»

Тема Графовые модели
Студент Звягин Д.О.
Группа ИУ7-53Б
Преподаватель Волкова Л.Л., Строганов Д.В.

Москва, 2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Графовые модели позволяют представлять алгоритмы и их выполнение в виде графов [1].

Таким образом, графовой моделью называется конечный орграф, вершины которого являются некоторыми командами или участками исполняемого кода, а дуги выражают некоторое отношение между этими участками. С помощью графовой модели могут быть выражены отношения последовательности, зависимости по данным и другие.

Графовые модели могут использоваться для поиска участков программы, которые могут быть выполнены параллельно.

Целью данной работы является построение четырёх графовых моделей для фрагмента кода, а также определение применимости графовых моделей к задаче анализа программного кода.

Для достижения этой цели нужно выполнить следующие задачи:

— выбрать фрагмент кода из лабораторной номер 5;

— построить графовые модели;

— сделать вывод о применимости графовых моделей к задаче анализа программного кода.

1 Фрагмент кода

Для выполнения работы, был выбран фрагмент кода из лабораторной работы номер 5. Код представлен на листинге 1.1.

Листинг 1.1 — Фрагмент кода из лабораторной работы номер 5

```
std::vector<std::string> Parser::getSteps() {
    auto candidates = filterNodes(GUMBO_TAG_DIV);          // 1
    for (const GumboNode *node : candidates) {             // 2
        if(!hasClass(node, "instructions"))                // 3
            continue;

        auto spans = filterNodes(node, GUMBO_TAG_SPAN); // 4
        if (spans.empty())                                 // 5
            continue;

        std::vector<std::string> res;                      // 6
        res.resize(spans.size());                          // 7

        for (const GumboNode *span : spans) {              // 8
            std::string tmptext = extractText(span);       // 9
            if (tmptext.size())                            // 10
                res.emplace_back(tmptext);                 // 11
        }

        return res;                                        // 12
    }

    return {};                                             // 13
}

std::string extractText(const GumboNode *node) {
    GumboNode *extracted_text =                            // 14
        static_cast<GumboNode *>(node->v.element.children.data[0]);
    if (extracted_text->type == GUMBO_NODE_TEXT) {         // 15
        return extracted_text->v.text.text;                // 16
    }

    return {};                                             // 17
}
```

Функция Parser::getSteps() вычленяет шаги из html-файла рецепта с сайта `chiefs.kz`, а

затем складывает полученные строки в вектор.

Функция extractText вычленяет из объекта типа GumboNode текст и возвращает его в формате std::string

Комментарии с числами указывают номера вершин, соответствующих определённым строкам.

В коде используются только стандартные и библиотечные (GumboNode) типы данных.

Функции hasClass и filterNodes являются пользовательскими, однако включают в себя множество рекурсивных вызовов, выполняя при этом функции, которая могли бы быть предложены библиотекой. В целях упрощения построения графов, данные функции будут восприниматься как библиотечные, а соответственно не будет представлен их полный код. Такое действие оправдано, так как они не изменяют входные данные, а следовательно, при распараллеливании могут быть вызваны любым потоком при условии, что один объект GumboNode обрабатывается одним потоком.

2 Построение графов

В данном разделе будут построены четыре графа для фрагмента кода из листинга 1.1.

Граф управления — описание передачи управления в программе. Дуги этого графа показывают, какие команды могут исполняться непосредственно друг за другом.

Информационный граф описывает передачу данных между командами. Дуги этого графа описывают, какие именно данные передаются от команды к команде.

Операционная история — граф, описывающий отношение управления в контексте конкретного запуска программы. Вершины в этом графе имеют не больше одной входящей и исходящей дуги.

Информационная история — граф, содержащий информацию о том, какие команды требовали информацию от других команд и какую именно информацию они требовали в процессе конкретного запуска программы.

2.1 Граф управления

Граф управления представляет собой описание передачи управления в программе. Дуга, проходящая из одной вершины в другую предполагает, что непосредственно после выполнения первой вершины может быть выполнена вторая (а может — любая из других вершин, к которым идут дуги из первой).

Граф управления для фрагмента кода представлен на рисунке 2.1.

2.2 Информационный граф

Информационный граф описывает передачу данных между командами. На нём описываются какие данные передаются от команды к командам.

Информационный граф для фрагмента кода представлен на рисунке 2.2.

Зелёными стрелками отмечена передача константы. То есть те ситуации, когда переданная информация точно не будет изменена.

Пунктирная стрелка от 9 к 14 означает комментарий. Она не является частью информационного графа.

Знаком “=>” отмечено изменение имени переменной при передаче её между функциями.

Это также сделано для упрощения чтения модели.

Дуги из вершин 16 и 17 возвращаются в дугу 9, так как работает оператор = на возвращённое из функции значение.

Вершина 13 означает возврат пустого массива. Ей не нужны никакие данные, так как это возврат константы.

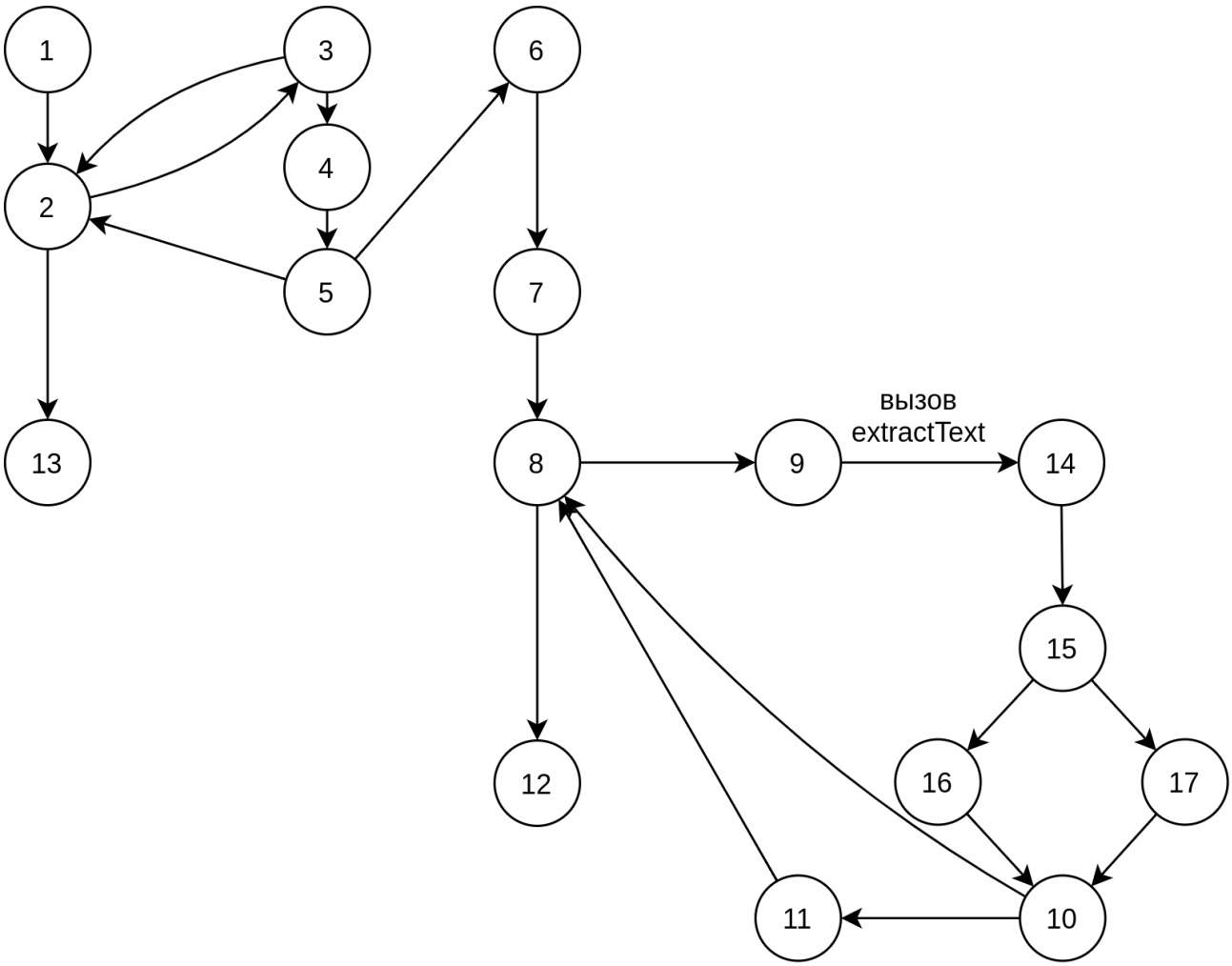

Рисунок 2.1 — Граф управления для фрагмента кода

2.3 Операционная история

Операционная история, описывает отношение управления в контексте некоторого определённого запуска программы.

В таком графе каждая вершина имеет не более одного входа и одного выхода.

Операционная история для фрагмента кода представлена на рисунке 2.3.

Оказалось, что в данном фрагменте кода расположен не двойной вложенный цикл, а два цикла: один для фильтрации подходящих элементов и выбора первого из них, а второй — для обработки его содержимого. Хотя технически в коде один цикл for вложен во второй.

С одной стороны, выбор фрагмента кода не верен, но с другой — это один из случаев, в котором использование графовых моделей алгоритма позволило выявить спорное архитектурное решение.

2.4 Информационная история

Информационная история содержит информацию о том, какие команды требовали информацию от других команд в контексте некоторого определённого запуска программы.

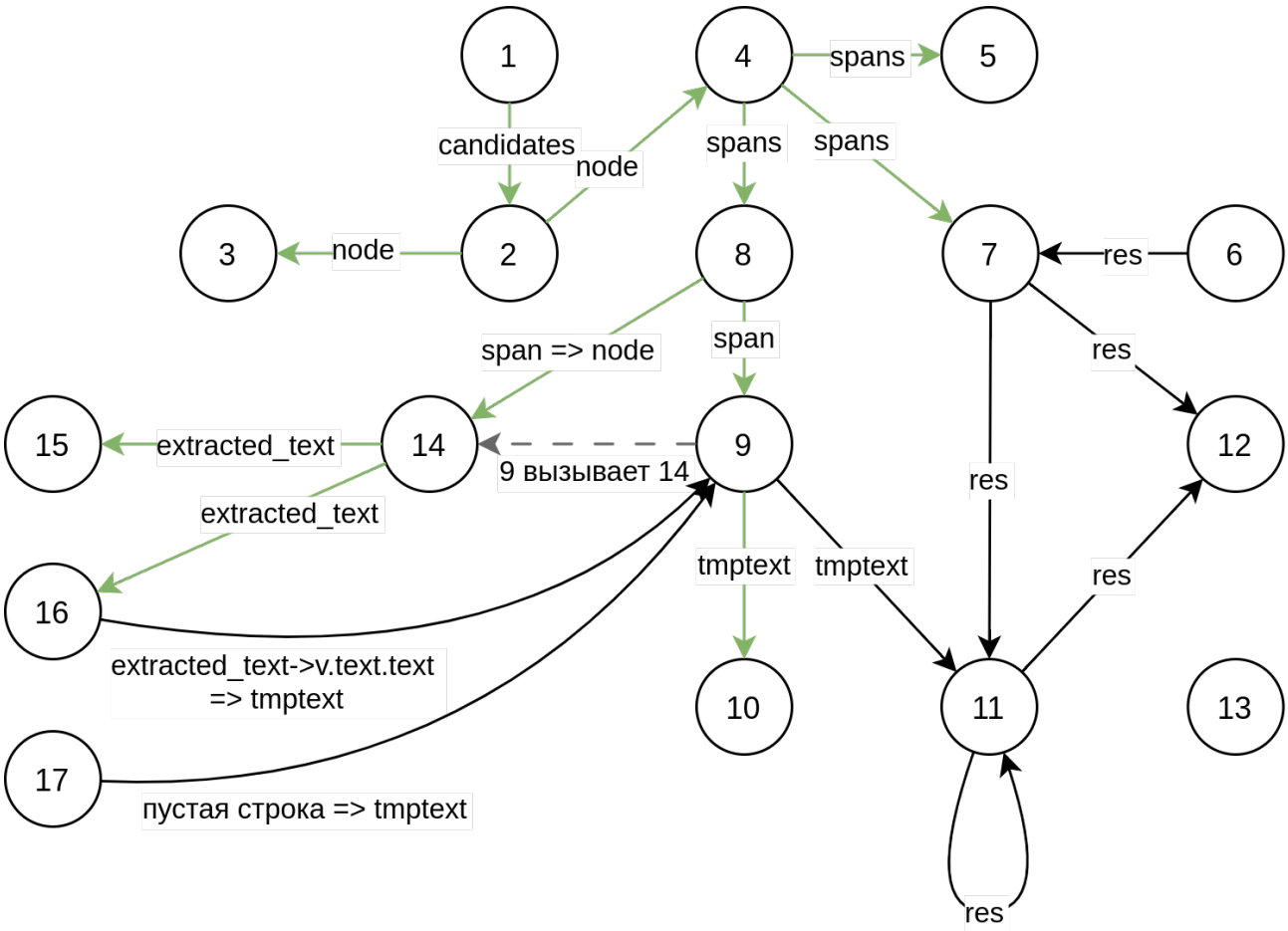

Рисунок 2.2 — Информационный граф для фрагмента кода

Информационная история для фрагмента кода представлена на рисунке 2.4.

На этом рисунке пунктирные квадраты означают комментарии. Пунктирные стрелки — циклы.

Как в информационном графе, зелёные дуги — передача константы.

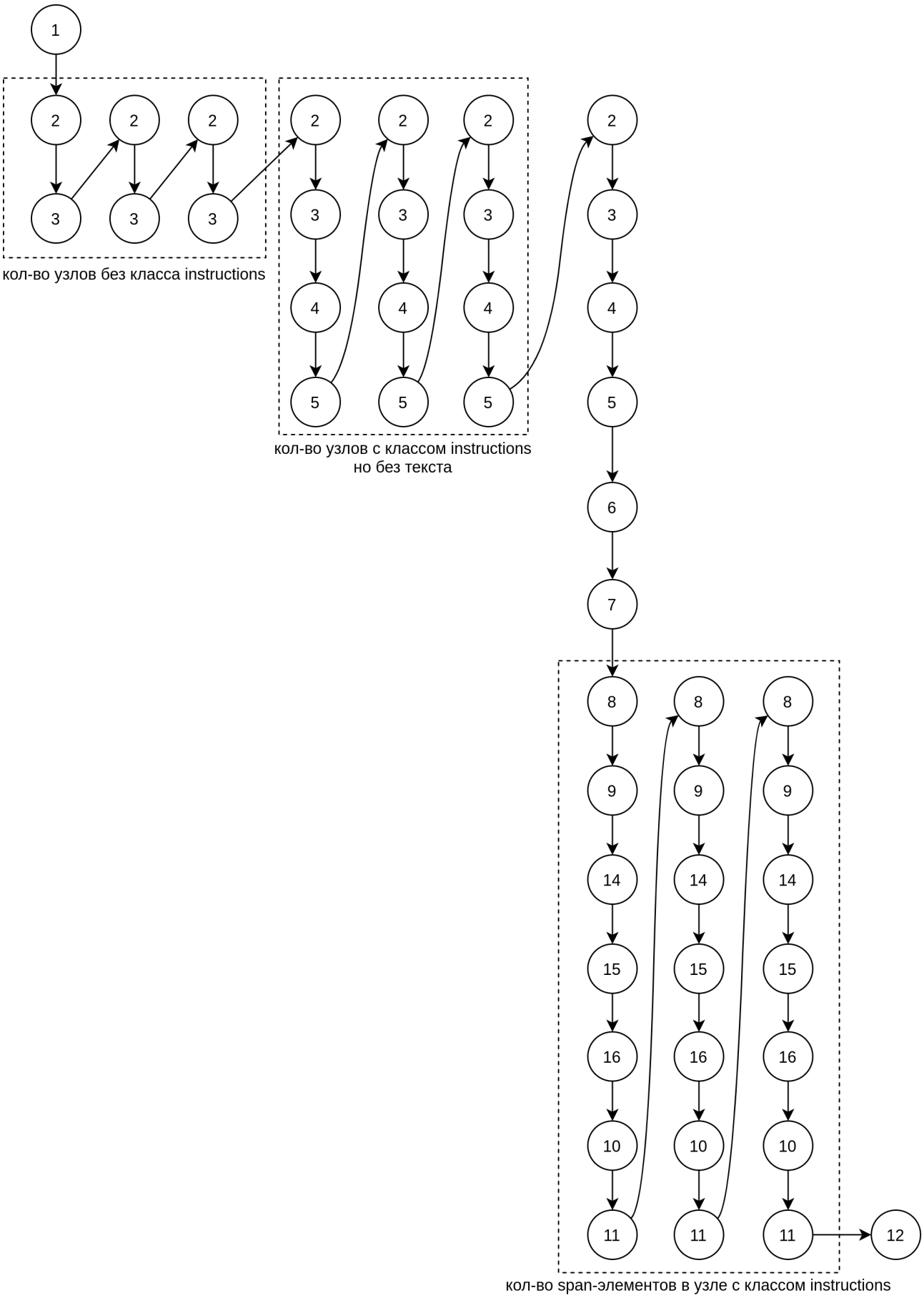

Рисунок 2.3 — Операционная история для фрагмента кода

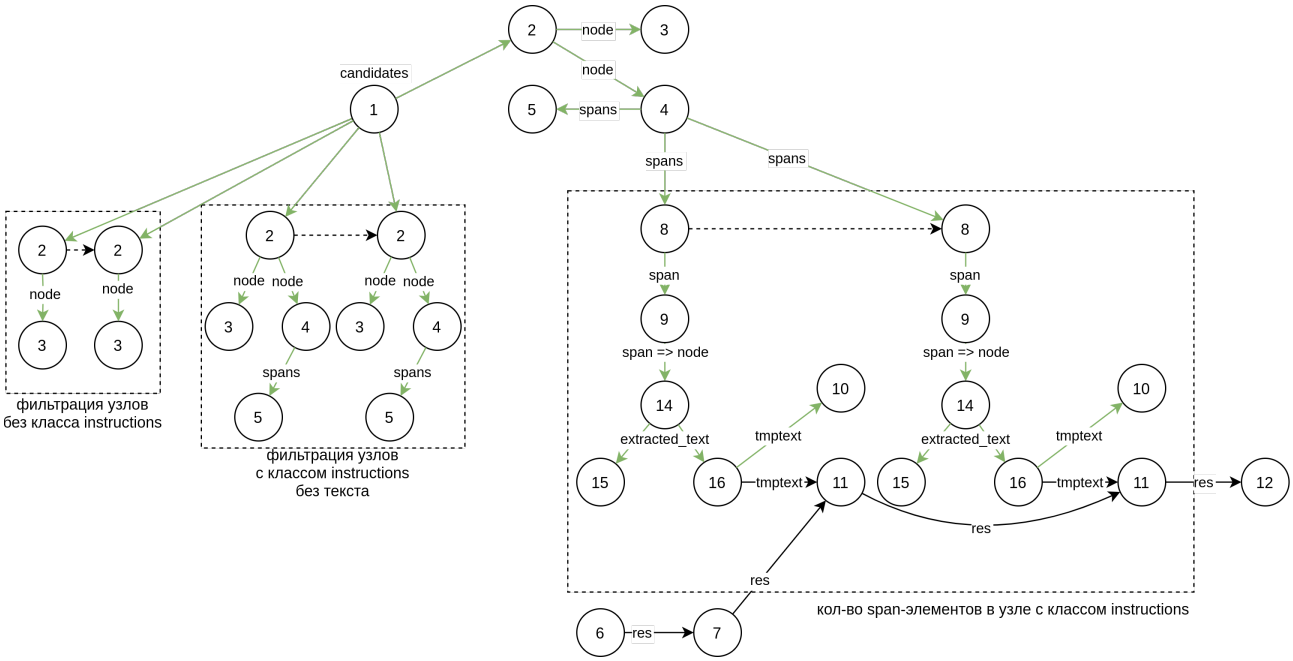

Рисунок 2.4 — Информационная история для фрагмента кода

ЗАКЛЮЧЕНИЕ

Цель данной лабораторной работы была достигнута, а именно: были построены четыре графовые модели для фрагмента кода из лабораторной работы номер 5, а также определена применимость графовых моделей к задаче анализа программного кода.

Графовые модели действительно упрощают задачу анализа кода и позволяют выявлять различные связи участков кода. В ходе данной работы, при построении операционной истории было выявлено то, что фрагмент кода не содержит настоящий вложенный цикл, а при должном опыте, это можно было бы выявить ещё при построении графа управления.

Для достижения цели были выполнены следующие задачи:

— был выбран фрагмент кода из лабораторной работы номер 5;

— были построены графовые модели для выбранного фрагмента кода;

— были сделаны выводы о применимости графовых моделей к задаче анализа программного кода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лекции по курсу «Анализ алгоритмов» МГТУ им. Н.Э. Баумана, кафедры «Программное обеспечение ЭВМ и информационные технологии» 2024 год.