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 Шахнович Дмитрий Сергеевич

Группа ИУ7-52Б

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

Москва, 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Графовая модель – ориентированный конечный граф, вершины которого являются некоторыми командами или участками исполняемого кода, а дуги выражают некоторое отношение между этими участками. Таким образом с помощью графовой модели могут быть выражены отношения последовательности, зависимости по данным и другие. Таким модели могут использоваться для анализа разработанных программ, в частности для поиска участков программы, которые могут быть выполнены параллельно.

Целью данной работы является разработка графовых моделей для заданного участка кода программы.

Для достижения цели необходимо выполнить следующие задачи:

— рассмотрение участка кода;

— разработка 4-х графовых моделей: графа управления, информационного графа, операционной истории и информационной истории.

1 Используемая программа

В листинге 1.1 представлена анализируемая программа, написанная на языке Go.

Листинг 1.1 — Данные, передающиеся между частями конвейера

```
package lab7

import (
"golang.org/x/net/html"
)

type TokenNode struct {
  StartTag *html.Token
  Children []*TokenNode
  EndTag   *html.Token
}

type TokenForest struct {
  Trees []*TokenNode
}

func (forest *TokenForest) Find(tag string, attrs map[string]string) *
   TokenNode { // 1
  var result *TokenNode
  for _, tree := range forest.Trees { // 2
```

```
      result = tree.Find(tag, attrs) // 3
      if result != nil {             // 4
        return result // 5
      }
    }
    return nil // 6
  }

  func (root *TokenNode) Find(tag string, attrs map[string]string) *
     TokenNode { // 7
    if root.StartTag.Data == tag { // 8
      if attrs == nil { // 9
        return root // 10
      }
      for _, attr := range root.StartTag.Attr { // 11
        if attrs[attr.Key] == attr.Val { // 12
          return root // 13
        }
      }
    }
    for _, child := range root.Children { // 14
      result := child.Find(tag, attrs) // 15
      if result != nil {               // 16
        return result // 17
      }
    }
    return nil // 18
  }
```

Программа 1.1 описывает две структуры:

— $TokenNode$ – структура, описывающая узел синтаксического дерева html. Состоит из начально и конечного синтаксических единиц, а также из указателей на узлы, которые находятся внутри текущего;

— $TokenForest$ – лес html деревьев. Состоит из массива $TokenNode$ – корней деревьев.

Также в программе представлены два метода:

— $Find$ структуры $TokenNode$ – поиск первого тега в дереве, содержащего хотя бы один из атрибутов заданных ключом и значением;

— $Find$ структуры $TokenNode$ – такой же поиск, но в лесу деревьев.

Комментарии в методах после определённых строк представляют номера команд, которые будут использованы при построении графовых моделей.

2 Графовые модели

2.1 Граф управления

Граф управления представляет собой описание передачи управления в программе. Таким образом дуги в таком описании показывают, какие команды могут исполняться непосредственно друг за другом. граф управления для программы 1.1 представлен на рисунке 2.1.

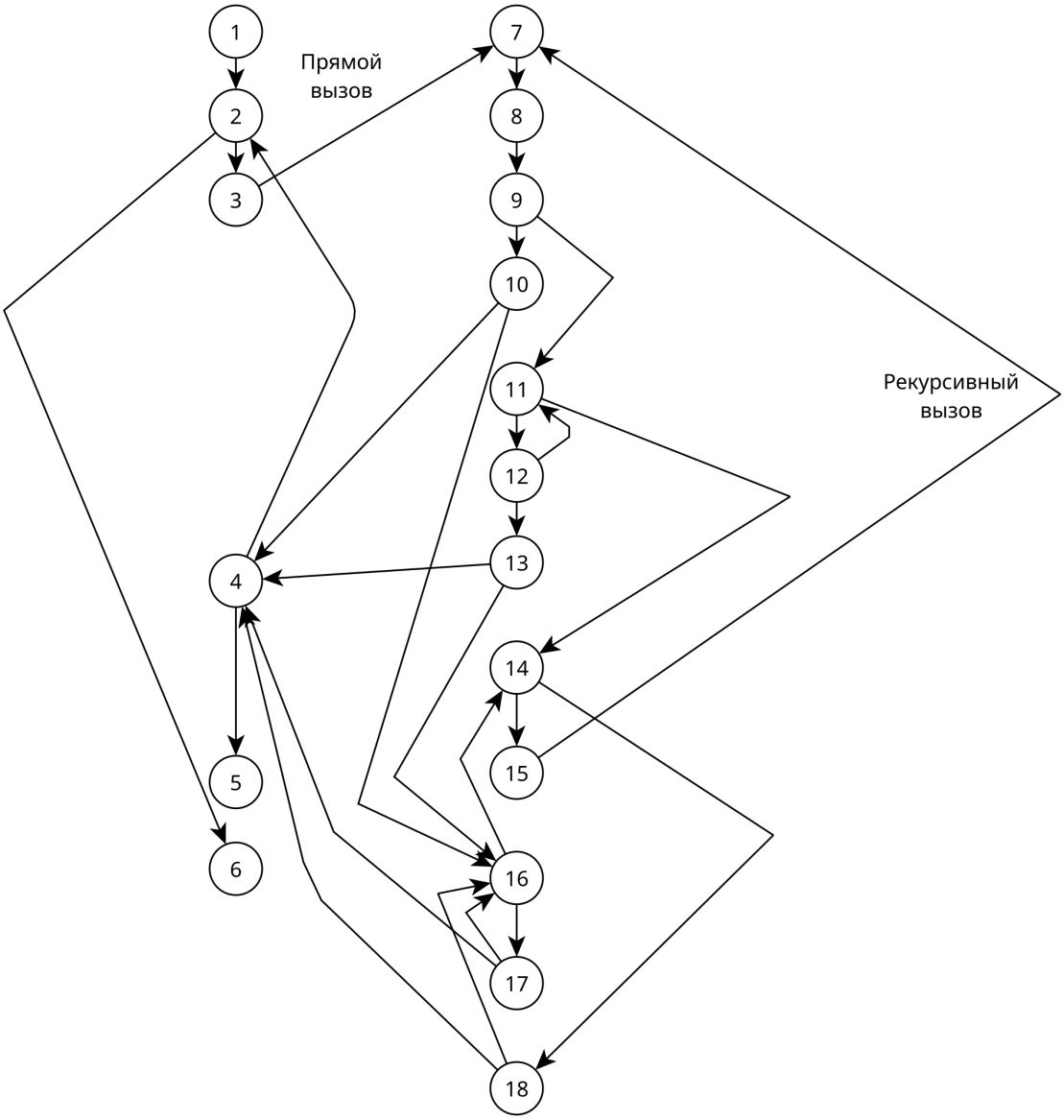

Рисунок 2.1 — Граф управления

2.2 Информационный граф

Информационный граф описывает передачу данных между командами. На нём описываются какие данные передаются от команды к командам. Информационный граф для программы 1.1 представлен на рисунке 2.2.

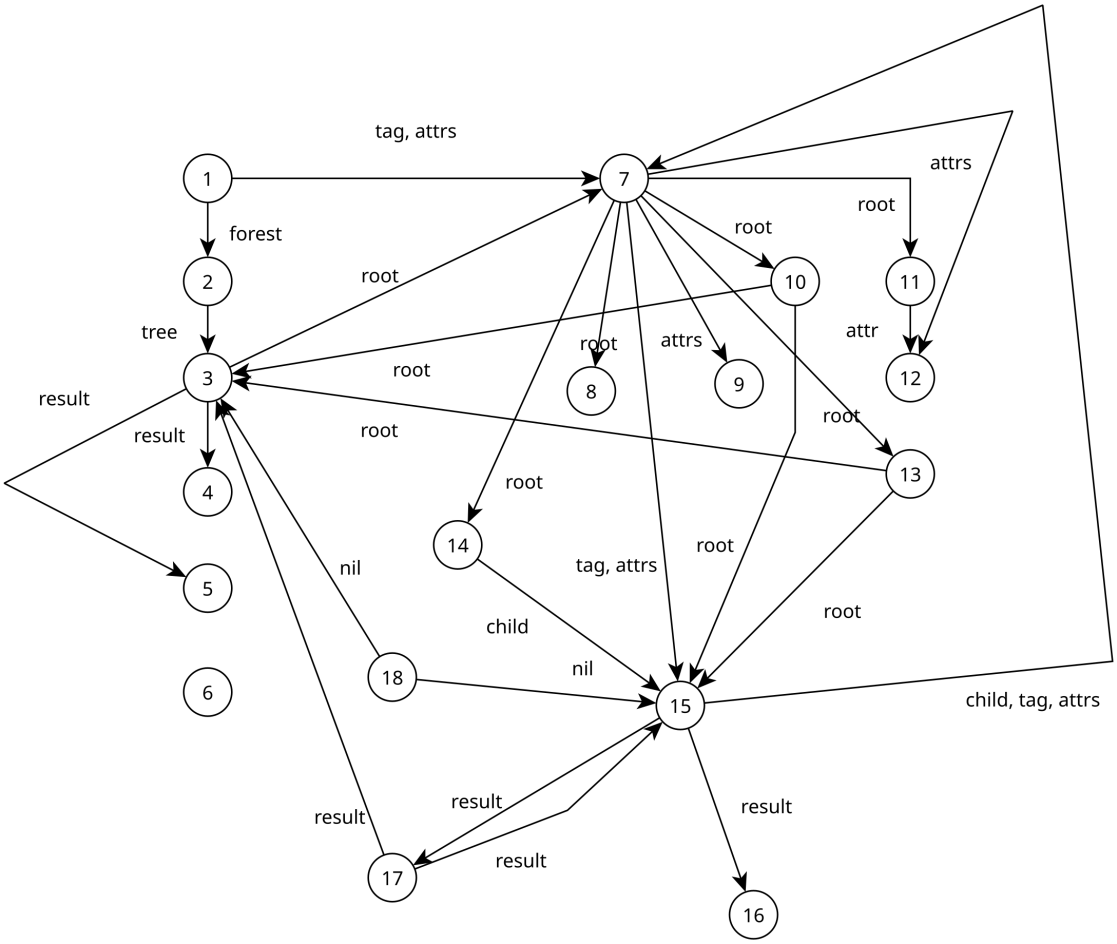

Рисунок 2.2 — Информационный граф

2.3 Операционная история

Операционная история, как и граф управления, описывает отношение управления, однако в отличие от него операционная история содержит информацию о некотором запуску программы, где каждая вершина имеет не более одного входа и одного выхода. Операционная история для программы 1.1 представлен на рисунке 2.3.

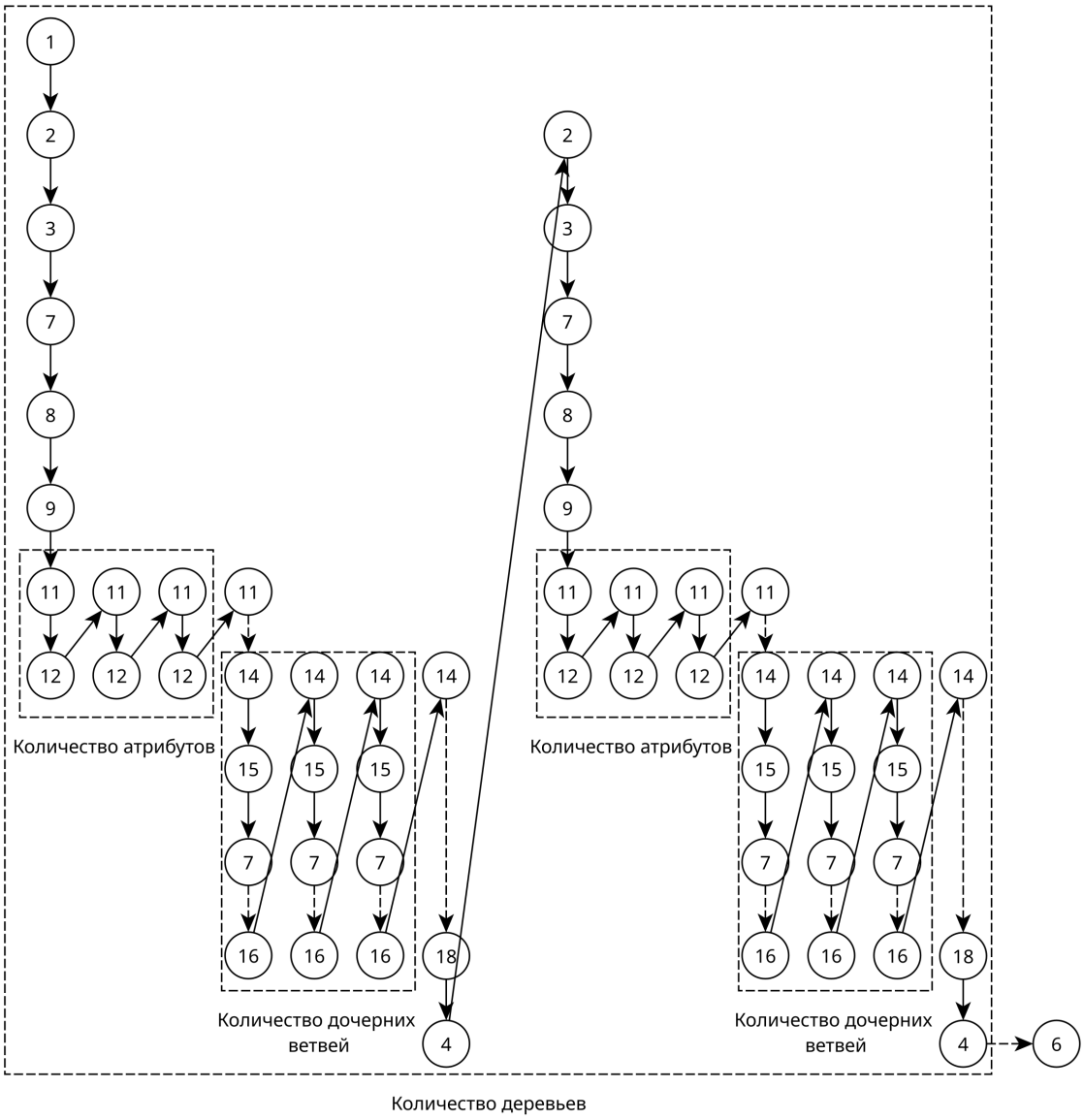

Рисунок 2.3 — операционная история

2.4 Информационная история

Информационная история, как и операционная история представляют собой некоторый запуск программы, однако он содержит информацию о последовательности команд, но содержит информацию о том, какие команды требуют информацию от других команд. Информационная история для программы 1.1 представлен на рисунке 2.4.

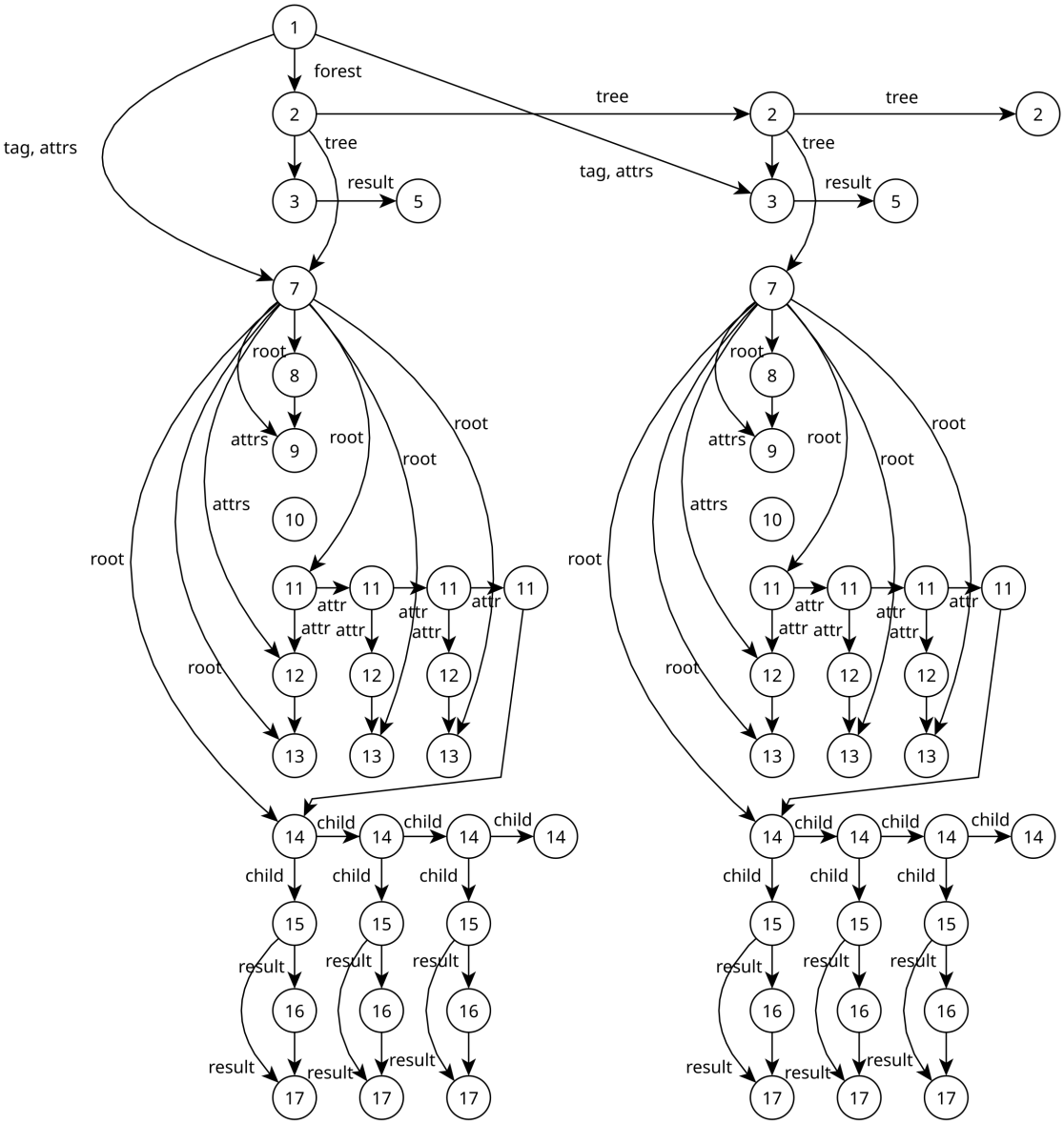

Рисунок 2.4 — Информационная история

ЗАКЛЮЧЕНИЕ

Цель – разработка графовых моделей для заданного участка кода программы – была выполнена.

В ходе работы была описана используемая программа, а также разработаны 4 графовые модели, описывающие её: граф управления, информационный графа, операционная история и информационная история.

Разработанные графовые модели действительно упрощают анализ программ. Граф управления и информационный граф помогают обнаружить скрытые связи между участками кода, а операционная история и информационная история позволяют обнаружить независимые части, которые потенциально могут выполняться параллельно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лекции курса «Анализ алгоритмов» за 2024 год. МГТУ им. Н.Э. Баумана, кафедра «Программное обеспечение ЭВМ и информационные технологии»