ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 31 за 2024 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

В.А. Судаков, А.Д. Шаблий

Разрешение циклических зависимостей графовой модели взаимосвязи требований к программному обеспечению

Рекомендуемая форма библиографической ссылки: Судаков В.А., Шаблий А.Д. Разрешение циклических зависимостей графовой модели взаимосвязи требований к программному обеспечению // Препринты ИПМ им. М.В.Келдыша. 2024. № 31. 13 с.
https://doi.org/10.20948/prepr-2024-31
https://library.keldysh.ru/preprint.asp?id=2024-31

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

В.А. Судаков, А.Д. Шаблий

Разрешение циклических зависимостей графовой модели взаимосвязи требований к программному обеспечению

Москва — 2024

Судаков В.А., Шаблий А.Д.

Разрешение циклических зависимостей графовой модели взаимосвязи требований к программному обеспечению

Получение сведений о трассируемости требований к программному обеспечению на файлы исходного кода приложения является сложной прикладной проблемой. Особенно при установлении необходимого объема верификационных процедур, выполнение которых необходимо для подтверждения выполнимости требований при изменении одного или нескольких файлов исходного кода. Разработана модель, которая обеспечивает анализ связности файлов исходного кода между собой и их соответствие требованиям. Предложен метод разрешения циклических зависимостей и формирования графа трассируемости требований на файлы исходного кода. Предложена программная реализация модели, в состав которой включены опциональные модули для исследования ее оптимальной конфигурации. Исследована зависимость времени работы от реализации способа хранения данных в разработанном программном решении.

Ключевые слова: требования, трассируемость, исходный код, графовые модели, ориентированный граф, циклы.

Sudakov Vladimir Anatolyevich, Shabliy Alexey Denisovich

Resolving cyclic dependencies of a graph model of software requirements relationships

Obtaining information about the traceability of software requirements to application source code files is a complex application problem. Especially when establishing the required scope of verification procedures, the implementation of which is necessary to confirm the feasibility of requirements when changing one or more source code files. A model has been developed that provides analysis of the connectivity of source code files between themselves and the requirements. A method is proposed for resolving cyclic dependencies and generating a traceability graph of requirements for source code files. A software implementation of the model is proposed, which includes optional modules for studying its optimal configuration. The dependence of operating time on the implementation of the data storage method in the developed software solution has been studied.

Key words: requirements, traceability, source code, graph models, directed graph, cycles.

Введение

Трассируемость требований программного обеспечения (ПО) – это взаимосвязь между требованиями к ПО и файлами исходного кода, которая определяет происхождения, порождения или зависимости между ними. Получение сведений о такой трассируемости является сложной прикладной проблемой. Зачастую такие сведения необходимы для анализа эффективности проведенных работ по проектированию приложения, оценке стоимости его тестирования [1], а также для управления его конфигурацией [2]. В данной работе предложен способ построения схемы, описывающей, в каких файлах исходного кода реализуется заданный объем требований к ПО, и адаптированной для получения сведений о зависимостях файлов исходного кода между собой.

Для решения задач, связанных с оптимизацией ПО [3], получением данных о загруженности ресурсов вычислительной системы [4], иерархическим описанием зависимостей компонентов [5], широкое применение нашли графовые модели. Это обусловлено простотой описания актуальных для программирования проблем с использованием графовых моделей [6].

При описании работы потоков управления, потоков данных, связи составных частей распределенной системы очень важно указывать направление потока или связи [7], [8]. При организации связей всегда есть источник и есть потребитель. Иногда один компонент системы, обозначаемый на графе вершиной, является одновременно источником и потребителем. Кроме того, он может быть источником для нескольких потребителей и потребителем от нескольких источников. В ряде задач одна вершина может быть источником и потребителем для самой себя, например в случае наличия обратных связей в описываемой модели. Для описания таких связей используются ориентированные графы, в которых по направлению дуг можно судить о принадлежности вершины к числу источников или приемников.

Для решения задач, связанных с оптимизацией многопоточного ПО [9], необходим механизм преобразования графа из исходного вида к целевому. Преобразование происходит по заранее сформулированным правилам и может осуществляться за несколько итераций. Условие окончания проведения итераций преобразования также определено заранее. Для решения задачи построения графа трассируемости требований к ПО на файлы исходного кода необходимо, чтобы вне зависимости от очередности вершин графа, к которым применяются действия по преобразованию, результирующий граф всегда формировался бы одинаково. Описанный в статье [9] способ не гарантирует этого.

Изложенный в работе [10] подход к формированию графа с применением алгоритмов нейронных сетей [11] не учитывает ограничение, что вершины результирующего графа должны быть двух категорий. В то же время приведенный способ формирования результирующего графа может быть

доработан для решения задачи построения оптимальной декомпозиции компонентов приложения с целью максимизации вариантов комплектаций его поставки при заданном трассировании требований к ПО на файлы исходного кода с учетом разрешенных циклических зависимостей и выполняя свою работу на уже предварительно преобразованном графе.

Описанные в [9] и [12] приемы преобразования ориентированного графа и видоизменения его в результате итерационно выполняемых действий нацелены на построение такого графа, который бы упрощал поиск цепочки задействованных в одном сценарии работы ПО вершин графа. Такие приемы требуют доработки и адаптации для решения задачи построения графа трассируемости требований к ПО на файлы исходного кода с учетом возможности наличия циклических зависимостей у файлов исходного кода между собой.

Кроме того, при описании способов преобразования исходного графа необходимо учитывать временные издержки, которые появляются в ходе выполнения операций над исходным графом. Объем издержек возрастает, если в исходном графе присутствует значительное число требований к ПО и файлов исходного кода.

Описание модели

Исходными данными для модели являются: информация о составе файлов исходного кода, сведения об их зависимостях между друг другом, а также трассируемость требований на них.

Перед началом работы модели необходимо определить:

1. множество файлов исходного кода $F_{sc}^{*} = \{F_{sc}^{1}, F_{sc}^{2}, \dots, F_{sc}^{n}\}$;
2. множество требований к ПО $R^{*} = \{R^{1}, R^{2}, \dots, R^{*}\}$;
3. матрицу бинарных отношений трассируемости требований к ПО на файлы исходного кода Mt размерности $n \times m$;
4. матрицу бинарных отношений зависимостей файлов исходного кода Md размерности $n \times n$.

Значение элемента $a_{i,j}^{Mt}$ матрицы Mt определяется следующим образом:

$$a_{i,j}^{Mt} = \begin{cases} 1, & \text{если файл } F_{sc}^{i} \text{ реализует требование } R^{j} \\ 0, & \text{если файл } F_{sc}^{i} \text{ не реализует требование } R^{j} \end{cases}.$$

Значение элемента $a_{i,j}^{Md}$ матрицы Md определяется следующим образом:

$$a_{i,j}^{Md} = \begin{cases} 1, & \text{если файл } F_{sc}^{i} \text{ имеет зависимость на файл } F_{sc}^{j} \\ 0, & \text{если файл } F_{sc}^{i} \text{ не имеет зависимость на файл } F_{sc}^{j} \text{ или } i = j \end{cases}.$$

Примеры исходных матриц Mt и Md приведены в таблицах 1 и 2 соответственно.

Таблица 1

Пример Mt

	R^1	R^2	…	R^m
F_{sc}^1	1	0	…	0
F_{sc}^2	0	1	…	0
F_{sc}^3	0	1	…	0
…	…	…	…	…
F_{sc}^n	0	0	…	1

Таблица 2

Пример Md

	F_{sc}^1	F_{sc}^2	F_{sc}^3	…	F_{sc}^n
F_{sc}^1	0	0	1	…	0
F_{sc}^2	0	0	0	…	0
F_{sc}^3	1	0	0	…	0
…	…	…	…	…	…
F_{sc}^n	0	0	0	…	0

Модель используется для выполнения следующих этапов:

1. считывание F_{sc}^*, R^*, Mt и Md;
2. построение исходного графа G;
3. преобразование исходного графа G в результирующий G'.

G представляет собой модифицированное И/ИЛИ-дерево, которое помимо стандартных взаимоотношений между структурными элементами по типам И/ИЛИ [13] включает в себя также:

1. отношение множественного ИЛИ;
2. отношение обязательного включения дочернего компонента;
3. отношение опционального включения дочернего компонента;
4. отношения, связывающие между собой компоненты, не являющиеся по отношению друг к другу родительскими или дочерними.

Кроме того, в графе G выделяются два слоя:

1. слой файлов исходного кода;
2. слой требований к ПО.

Пример G, построенного по входным данным, приведен на рисунке 1.

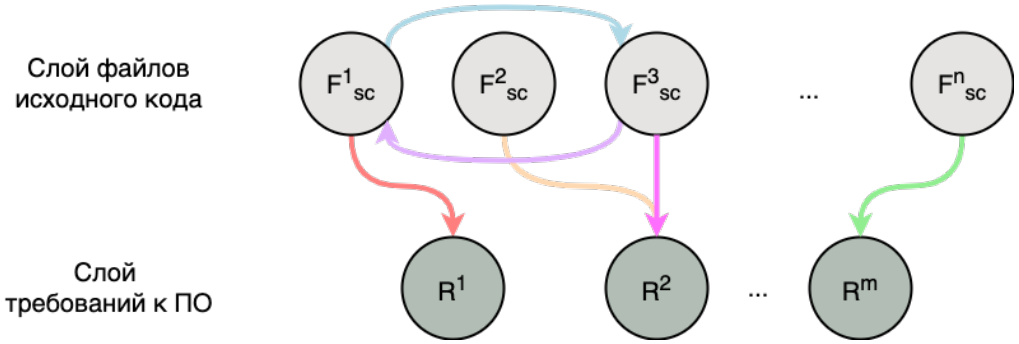

Рис. 1. Пример $\boldsymbol{G}$

Для преобразования G в G' итерационно выполняется коррекция, предложенная в работе [14], до тех пор, пока в G присутствуют циклы из произвольного числа узлов [15]. Коррекция выполняется методом слияния [4] входящих в цикл вершин, образуя группу. При этом:

1. дуги между сливающимися вершинами удаляются;
2. образованная группа является новой вершиной графа, все входящие и исходящие дуги замыкаются на нее. Пример слияния приведен на рисунке 2.

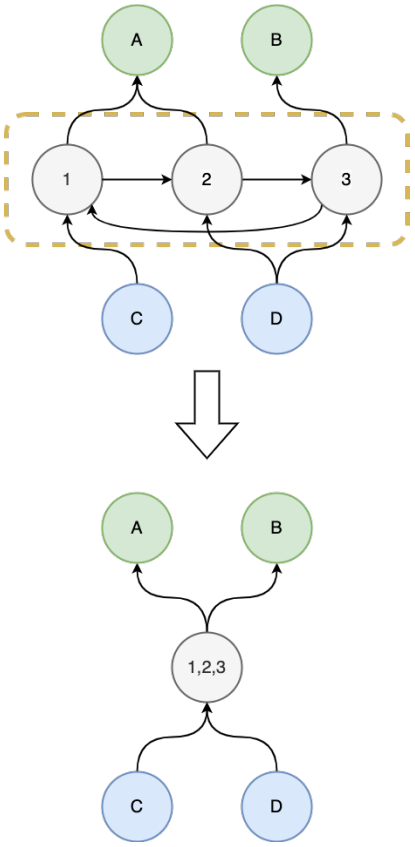

Рис. 2. Пример слияния входящих в цикл вершин

На последующей итерации поиска циклов в графе обозначенные выше группы учитываются как обычные вершины и, таким образом, могут участвовать в образовании новых групп по вышеуказанным правилам.

В приведенном примере (рис. 1) вершины F_{sc}^1 и F_{sc}^3 образуют цикл. Результат слияния приведен на рисунке 3.

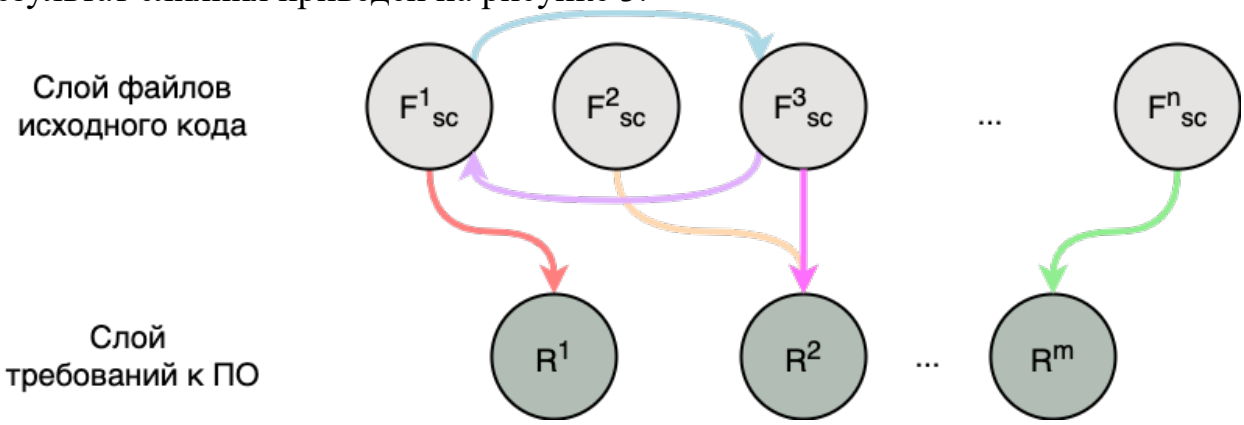

Рис. 3. Пример $\boldsymbol{G'}$

Результирующий граф может быть использован при оценке объема верификационных процедур, которые необходимо выполнить при внесении изменений в файлы исходного кода. Так, при внесении изменений в файл исходного кода F_{sc}^1 следует выполнить верификационные процедуры реализации требований R^1 и R^2.

Программная реализация

Описанная модель реализована на языке программирования Java. Реализация сочетает достоинства применения как парадигмы ООП, так и функционального программирования за счет активного использования лямбда-выражений.

При проектировании программной реализации было выявлено, что на производительность решения существенное влияние оказывает способ хранения информации о вершинах и дугах. Вариативность способов хранения этой информации зависит от выбранного языка программирования и объема подключаемых библиотек.

Используемый в работах [5], [16], [17] способ хранения информации в массивах является самым простым в реализации. Его использование позволяет наиболее эффективно осуществлять поиск элементов и обновление информации о наличествующих в графе вершинах и дугах. Как показано в работе [18], целесообразно изменять значение весовых коэффициентов. Данный способ наиболее эффективно использовать при работе на статическом графе, который не изменяется с течением времени. Использование его в динамически изменяющемся графе приведет к необходимости создания новых экземпляров массивов с последующим заполнением их значений.

В работе [19] приводится пример использования многомерных массивов. Однако, как описано ранее, данный подход эффективен при неизменяемом числе связей между компонентами графа. Например, если у каждой вершины зафиксировано число дуг, а в процессе преобразований графа моделью изменяется информация о том, какие именно дуги соединяют вершину с другими вершинами графа. При проектировании модели, в которой число дуг у вершины графа может изменяться, могут создаваться новые вершины и дуги, удаляться ранее созданные, эффективным представляется использование динамических структур данных.

Примером динамических структур данных являются связные списки. В работе [20] рассматривается пример их использования. Однако списки допускают хранение одинаковых элементов, что при проектировании описанной графовой модели недопустимо. Модель предусматривает наличие уникальных элементов. Вследствие этого было принято решение об использовании множеств уникальных элементов. В Java такая структура данных носит название Set.

Выбранная структура данных представляет коллекцию, которая может быть реализована по-разному. Выбор реализации влияет на время работы программной реализации модели. С целью проведения экспериментов и получения информации о зависимости времени работы программы от числа элементов графа в реализации коллекции Set предусмотрены два модуля для взаимодействия с данными, заданных пользователем:

1. модуль определения реализации коллекции Set;
2. модуль генерации исходных данных.

Модуль определения реализации коллекции Set выполнен для оптимальности применения различных реализаций коллекции Set в разработанной программной реализации. Благодаря использованию этого модуля достигается возможность задания конкретной реализации с последующим ее внедрением в исходный код программной реализации модели. Модуль позволяет внедрять как стандартные реализации коллекции Set, так и реализации, хранящиеся в подключаемых библиотеках или самостоятельно реализованных пользователем.

Модуль генерации исходных данных выполнен для сокращения времени на подготовку Mt и Md.

В качестве входных данных модуль принимает четыре параметра:

1. Fc – число файлов исходного кода;
2. Rc – число требований к ПО;
3. Mfd – максимальное число зависимостей у файла исходного кода;
4. Mfr – максимальное число реализуемых требований в одном файле.

Диаграмма работы выполненной программной реализации приведена на рисунке 4.

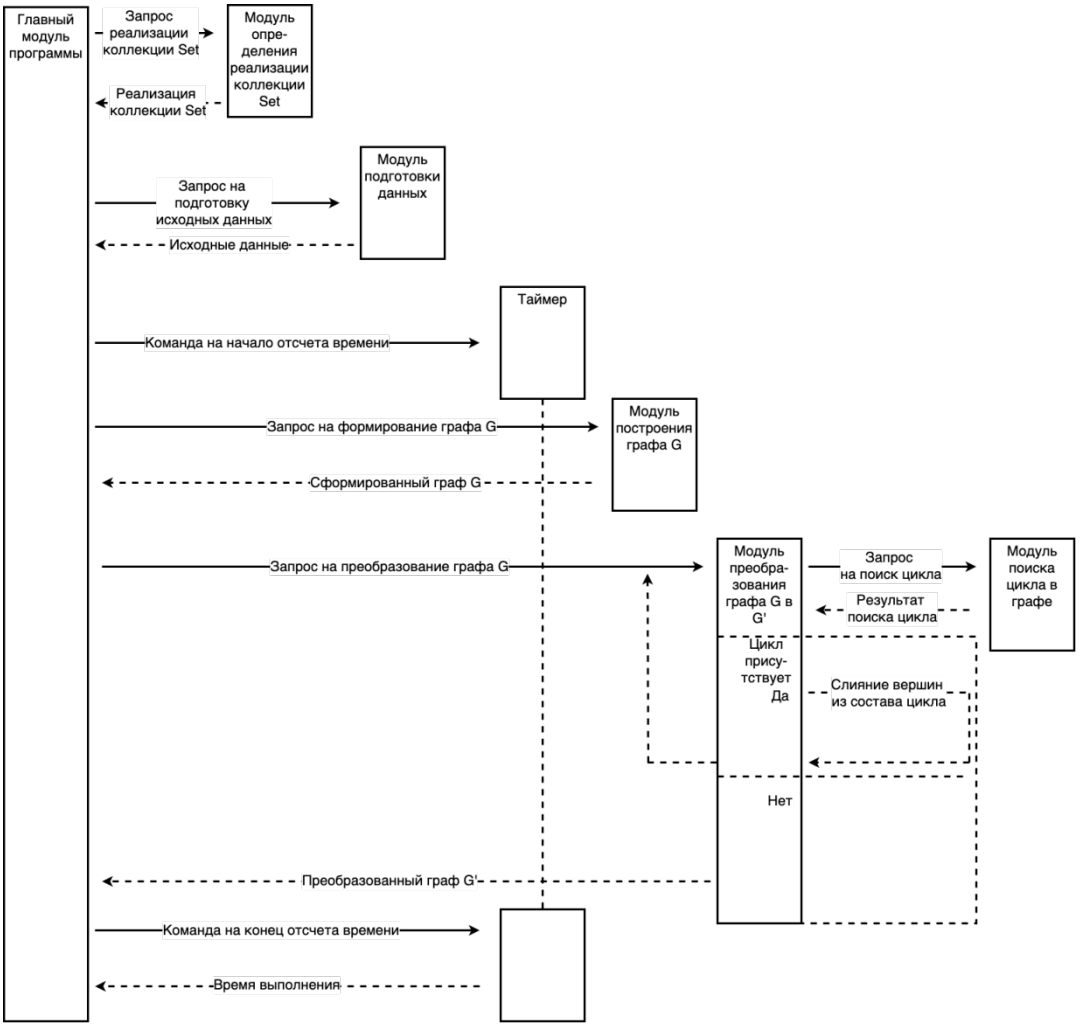

Рис. 4. Диаграмма работы выполненной программной реализации

Эксперименты и обсуждение

В целях проверки работоспособности программной реализации описанной модели проводилась серия экспериментов. В качестве входных данных использовались различные комбинации значений параметров для модуля генерации исходных данных.

Характеристики оборудования, используемого для проведения экспериментов:

1. операционная система – Ubuntu 23.04;
2. процессор – 2-ядерный процессор Intel Core i5 с тактовой частотой 1,8 GHz;
3. объем ОЗУ – 8 ГБ.

В проведенных экспериментах были собраны сведения о среднем времени выполнения операций по преобразованию G в G', которые были выполнены по 1000 раз каждый при заданных параметрах генератора исходных данных для следующих стандартных реализаций коллекции Set:

- HashSet;

- LinkedHashSet;
- TreeSet.

На рисунках 5 и 6 приводится графическая интерпретация результатов экспериментов при работе с числом файлов исходного кода в 100 и 1000 единиц соответственно.

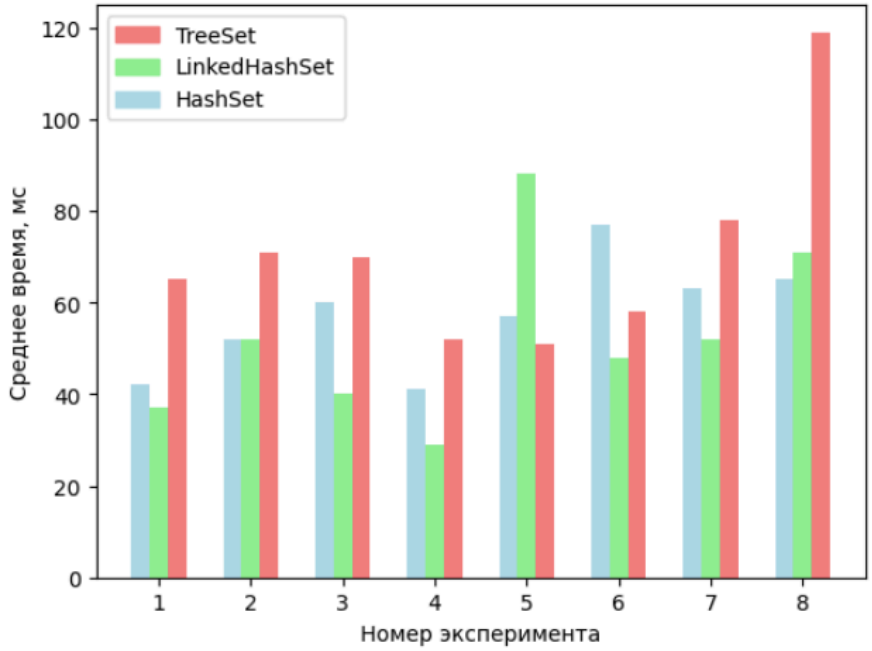

Рис. 5. Графическая интерпретация результатов экспериментов при работе с числом файлов исходного кода в 100 единиц

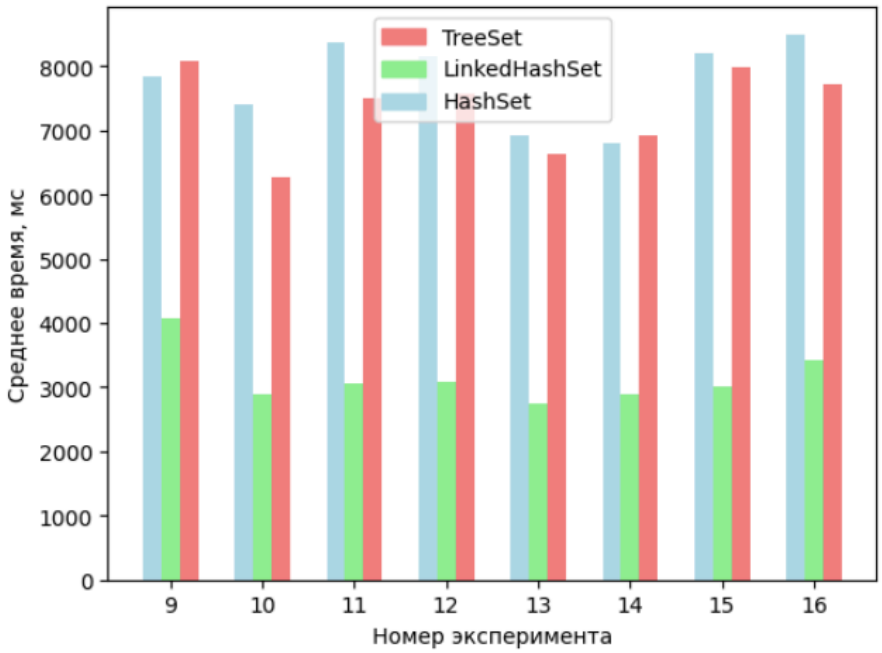

Рис. 6. Графическая интерпретация результатов экспериментов при работе с числом файлов исходного кода в 1000 единиц

Результаты экспериментов показывают, что резкое увеличение времени выполнения преобразования графа происходит при увеличении числа файлов

исходного кода, а значительное увеличение числа требований к ПО к такому эффекту не приводит.

Наибольшее значение среднего времени составляет 8496 мс, оно достигается при работе с реализацией HashSet. Наименьшее значение среднего времени достигается при работе с LinkedHashSet. При использовании этой реализации коллекции Set наибольшее значение среднего времени составляет 4061 мс, в то время как для TreeSet максимальное значение среднего времени достигает 8083 мс в экспериментах по взаимодействию с файлами исходного кода в количестве 1000 единиц.

В дальнейших исследованиях предполагается проведение сравнения предложенного подхода с методом факторного моделирования [21] как альтернативного представления вычислительных процедур на графах, без устранения циклов и позволяющих вычислять взаимное влияние вершин.

Заключение

В статье предложена модель, отображающая трассируемость требований к ПО на файлы исходного кода и позволяющая разрешать циклические зависимости между файлами исходного кода. Используя ее, можно получать информацию о степени связности файлов исходного кода друг с другом, на какую долю функционала оказывается влияние при внесении изменений в файлы исходного кода. Данные сведения могут быть полезны при оценке необходимого объема выполнения верификационных процедур после внесения изменений в один или несколько файлов исходного кода.

Библиографический список

1. Вигура А.Н. Анализ и тестирование программ на основе алгебраической модели, Информационные технологии // Вестник Нижегородского университета им. Н.И. Лобачевского. 2011. № 5 (1). с. 185–190.
2. Осипов В.П., Судаков В.А., Хахулин Г.Ф. Информационные технологии формирования этапной программы научно-прикладных исследований на российском сегменте Международной космической станции // Вестник компьютерных и информационных технологий. 2012. № 12(102). с. 24-28.
3. Штейнберг О.Б., Ивлев И.А. Применение преобразования циклов "Retiming" с целью уменьшения количества используемых регистров // Южный федеральный университет // Известия ВУЗов. Северо-Кавказский регион. Технические науки. 2017. № 3, с 76-80.
4. Недоводеев К.В. Метод генерации графов потоков данных, используемых при автоматическом синтезе параллельных программ для неоднородных многоядерных процессов // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. 2012. № 3, с. 47-52.

5. Штейнберг О. Б. Минимизация количества временных массивов в задаче разбиения циклов // Известия ВУЗов, Северо-Кавказский регион, естественные науки. 2011. № 5, с. 31-35.
6. Огородов С.В. Обоснование линейно-упорядоченного представления графовых моделей программ // Известия Томского политехнического университета. 2008. Т. 312. № 5, с. 85-89.
7. Чертков А.А., Каск Я.Н., Очина Л.Б. Маршрутизация потоковой сети на основе модификации алгоритма Беллмана – Форда // Вестник Государственного университета морского и речного флота имени адмирала С. О. Макарова. 2022, № 14(4), с. 615-627. https://doi.org/10.21821/2309-5180-2022-14-4-615-627
8. Сахаров В.В., Чертков А.А., Очина Л.Б. Маршрутизация сетей с отрицательными весами звеньев в пакете оптимизации MATLAB // Вестник Государственного университета морского и речного флота имени адмирала С.О. Макарова. 2019. № 2 (11). с. 230-242.
9. Корячко В.П., Скворцов С.В. Иерархическая модель глобальной оптимизации у параллельных объектных программ // Электронный журнал "Инженерное образование". 2006. № 8.
10. Карпов Ю.Л., Волкова И.А., Вылиток А. А., Карпов Л. Е., Сметанин Ю. Г. Проектирование интерфейсов классов графовой модели нейронной сети // Труды ИСП РАН. 2019. т. 31, № 4. с. 97-112.
11. Тарков М.С. Об эффективности построения гамильтоновых циклов в графах распределенных вычислительных систем рекуррентными нейронными сетями // Управление большими системами. Выпуск 43. М.: ИПУ РАН, 2013. с. 157-171.
12. Каленкова А.А. Оптимизация потоков работ по времени выполнения, основанная на удалении избыточных потоков управления // Труды МФТИ. 2009. Т. 1, № 2, с. 160-174.
13. Евсеева Ю.И., Бождай А.С. Метод структурно-параметрического синтеза адаптивных программных компонентов виртуальной образовательной среды // Известия высших учебных заведений. Поволжский регион. Технические науки. 2016. № 3(39). с. 84–92. DOI: 10.21685/2072- 3059-2016-3-8.
14. Четверина О.А. Методы коррекции профильной информации в процессе компиляции // Труды ИСП РАН. 2015. т. 27, вып. 6, с. 49-65.
15. Емельченков Е.П., Мунерман В.И., Мунерман Д.В., Самойлова Т.А. Один метод построения циклов в графе // Современные информационные технологии и ИТ-образование. 2021. Т. 17, № 4. с. 814-823;
16. Баглий А.П., Кривошеев Н.М., Штейнберг Б.Я., Штейнберг О.Б. Преобразования программ в оптимизирующей распараллеливающей системе для распараллеливания на распределенную память // Инженерный вестник Дона. 2022. №12.
17. Попов А.Ю. Применение вычислительных систем с многими потоками команд и одним потоком данных для решения задач оптимизации // Вестник МГТУ им. Н.Э. Баумана. Сер. "Приборостроение". 2012. №4. с. 146-154.

18. Фролов А.С., Семенов А.С. Обзор проблемно-ориентированных языков программирования для параллельного анализа статических графов // Computational nanotechnology. 2017. № 1. с. 27-32.
19. Шульженко А.М. Автоматическое определение циклов ParDo в программе // Известия высших учебных заведений. Северо-Кавказский регион. Серия: Естественные науки. 2005. № 11. с. 77-87.
20. Кошелев В.К., Игнатьев В.Н., Борзилов А.И. Инфраструктура статического анализа программ на языке C# // Труды ИСП РАН, том 28, вып. 1, 2016 г., с. 21-40.
21. Четверушкин Б.Н., Судаков В.А. Факторное моделирование для инновационно-активных предприятий // Математическое моделирование. 2020. Т. 32, № 3. с. 115-126. DOI: 10.20948/mm-2020-03-07.

Оглавление

Введение 3
Описание модели 4
Программная реализация 7
Эксперименты и обсуждение 9
Заключение 11
Библиографический список 11