ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

DOI 10.34216/2587-6147-2020-2-48-26-28

УДК 004.021
Сонин Ярослав Юрьевич
магистрант
Костромской государственный университет, г. Кострома, Россия
Орлов Александр Валерьевич
кандидат технических наук, доцент
Костромской государственный университет, г. Кострома, Россия
zhjckfd123456789987654321@mail.ru, aorlov@list.ru

ЗАДАЧА ОПРЕДЕЛЕНИЯ ПОРЯДКА ЗАПУСКА МОДУЛЕЙ В АЛГОРИТМЕ СОРТИРОВКИ ГРАФА ПРИМЕНИТЕЛЬНО К РАСЧЕТУ ФИЗИЧЕСКИХ ВЕЛИЧИН

В данной статье анализируются методы решения задачи определения порядка вызова взаимосвязанных компонентов программы с помощью нисходящего и восходящего алгоритмов определения порядка запуска модулей. В ходе исследования были рассмотрены три подхода к определению порядка запуска модулей: известные алгоритмы топологической сортировки Кана и Тарьяна, а также предложен «восходящий» алгоритм, базирующийся на поиске модулей, способных рассчитать недостающие величины. Для проверки работы алгоритмов была создана их пробная реализация на языке C#, позволяющая проверить корректность их работы, способность обнаруживать различные виды ошибок, определить скорость работы при большом числе модулей. По итогам тестирования пробной реализации был сделан вывод, что восходящий алгоритм в большинстве случаев работает быстрее и является более удобным в использовании.
Ключевые слова: *модульность, алгоритм, граф, зависимость, расчет, переменная, производительность, топологическая сортировка.*

В настоящее время все еще остается актуальной задача определения механических характеристик текстильных материалов. Для решения этой задачи создавались информационные системы, основанные на различных методах регистрации и обработки данных [1].

Тем не менее, в ходе создания подобных систем чаще всего встает задача расчета одного набора физических величин на базе набора других, уже известных величин. С архитектурной точки зрения имеет смысл оформить расчет отдельных величин в виде отдельных программных модулей [2].

При этом порядок расчета величин и порядок запуска рассчитывающих их модулей тесно связаны между собой. Модули могут требовать для своей работы величины, вычисляемые другими модулями. Ведь изначально доступен только набор некоторых базовых величин.

Возникает необходимость в определении порядка вызова модулей, при котором на момент вызова каждого модуля все необходимые ему величины уже были рассчитаны. Иными словами, требуется определить зависимости того или иного модуля. Для этого требуется знать, какие величины являются входными и выходными для каждого модуля. Также зачастую является желательным определение списка модулей, подлежащих вызову для расчета заданного набора величин.

Данная задача может быть сведена к топологической сортировке графа, вершинами которого являются модули, а ребрами – вычисляемые ими величины.

Хорошо известны два алгоритма топологической сортировки. Суть алгоритма Кана [3, с. 220–224] применительно к поставленной задаче можно описать следующим образом. Нам известен список требуемых модулей. На каждой итерации алгоритма определяем, какие модули могут быть запущены для текущего набора доступных величин. При успешном завершении алгоритма все требуемые модули были упорядочены. При неудаче алгоритма были обнаружены модули, которые нельзя вызвать из-за нехватки значений требуемых величин.

Этот алгоритм прост в понимании и реализации, но он требует от пользователя знания всех промежуточных величин, необходимых для рас-

чета искомых, а также знания распределения вычисляемых величин по доступным модулям.

Алгоритм Тарьяна [4, с. 649–651] производит рекурсивный поиск в глубину, пытаясь для каждого модуля определить, какие модули следует вызывать строго позднее данного. Однако, поскольку алгоритм описан в терминах теории графов, перед началом работы потребуется составить полный граф взаимных зависимостей модулей, что может увеличить время работы.

Этот алгоритм также обладает недостатком, связанным с необходимостью предварительного выделения подграфа, рассчитывающего искомые величины.

Можно предложить альтернативный подход, который мы обозначим как «восходящий алгоритм». Изначально известен список доступных для вызова модулей и список величин, которые нужно вычислить. На каждой итерации определяем, какие модули могут вычислить недостающие величины. При успешном завершении алгоритма будет найдено упорядоченное подмножество модулей, вызов которых приведет к расчету требуемых величин. Неудача алгоритма означает, что на одной из итераций не удалось найти ни одного модуля, способного рассчитать одну или более из требуемых величин.

Блок-схема данного алгоритма приведена на рисунке.

Алгоритм:

1) инициализируем множество изначально доступных величин теми величинами, которые известны заранее;
2) инициализируем множество требуемых величин;
3) инициализируем список доступных модулей всеми доступными модулями;
4) удаляем из списка недостающих величин те, которые входят в список изначально доступных. Если список требуемых величин пуст, переходим к пункту 13;
5) перебираем список модулей. Если нашли модуль, множество выходных величин которого пересекается с множеством требуемых величин, то переходим к пункту 6. Если такой модуль не найден, переходим к пункту 14. Обозначим этот модуль как «модуль-кандидат»;
6) ищем в списке вызова модулей модуль, у которого множество входных величин пересекается с множеством выходных величин модуля-кандидата;
7) добавляем модуль-кандидат в список вызываемых модулей на позицию перед найденным в пункте 6 модулем. Если модуль не был найден в пункте 6, добавляем модуль-кандидат в конец списка вызова;

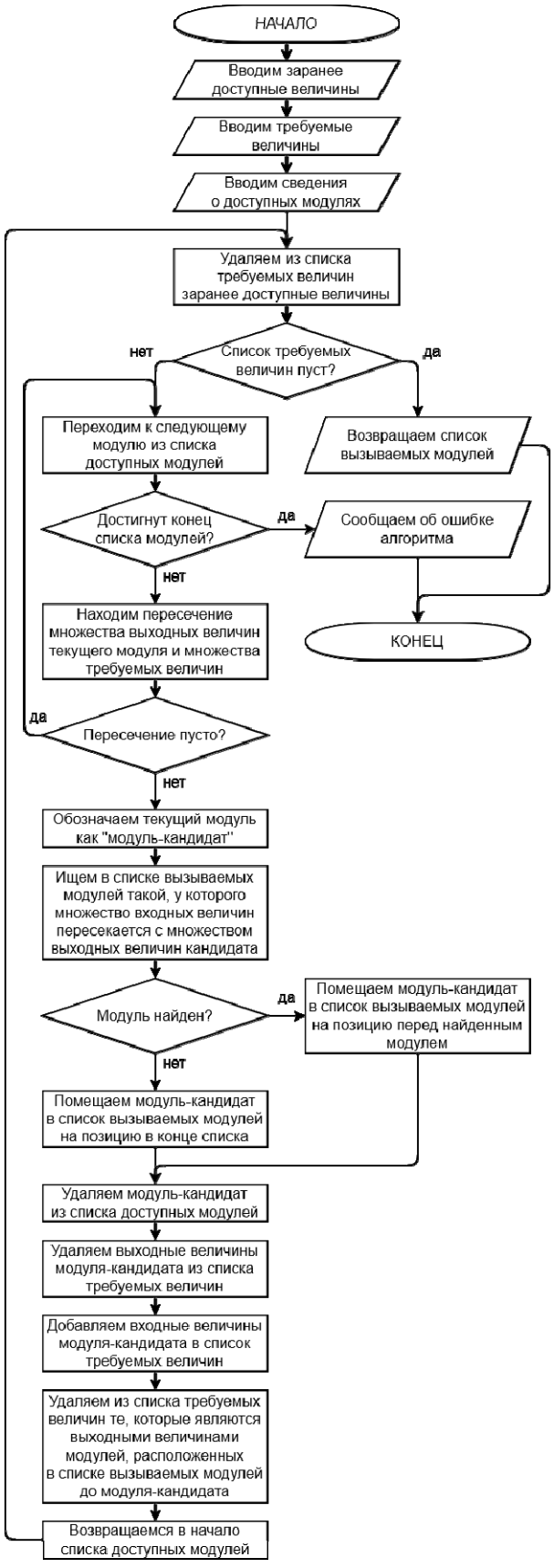

Рис. Блок-схема восходящего алгоритма

8) убираем модуль-кандидат из списка доступных модулей, чтобы предотвратить его повторный вызов;

9) убираем выходные величины модуля-кандидата из множества требуемых величин;
10) добавляем входные величины модуля-кандидата во множество требуемых величин;
11) удаляем из множества требуемых величин те, которые входят в множество заранее доступных величин;
12) удаляем из множества требуемых величин те, которые являются выходными величинами модулей, расположенных в списке вызова модулей до модуля-кандидата;
13) если множество требуемых величин пусто, то сортировка успешно завершена, завершаем алгоритм. Иначе переходим к пункту 4;
14) отсутствие модуля-кандидата означает, что одна или более из требуемых величин не рассчитываются ни одним модулем. Такая ситуация может возникнуть, если неизвестная величина была запрошена пользователем либо является входной для одного из модулей. Сообщаем о неудаче и завершаем алгоритм.

Восходящий алгоритм позволяет пользователю указать нужные ему величины без знания названий модулей, вычисляющих их. Поэтому он, несмотря на большую сложность разработки, является более предпочтительным вариантом с точки зрения простоты использования.

Нами создана пробная реализация алгоритмов на языке C# [5], в рамках которой проверялась скорость их работы при большом числе (N = 1000) модулей. Проверка производилась при различном порядке расположения модулей в исходном списке: лучший сценарий (модули уже отсортированы в нужном порядке), худший сценарий (модули отсортированы в обратном порядке) и случайный порядок модулей (одинаковый для всех алгоритмов). Результаты приведены в таблице.

По итогам тестирования пробной реализации установлено, что восходящий алгоритм не только является более удобным в использовании, но и работает быстрее, особенно в ситуациях, когда порядок модулей в исходном списке далек от оптимального.

ВЫВОДЫ

1. Рассмотрены различные методы решения задачи определения порядка вызова взаимосвязанных компонентов программы.

2. Рассмотрен альтернативный подход, более соответствующий поставленной задаче.

3. Создана пробная реализация рассмотренных решений, проверена ее работа как в условиях корректности поступающих данных, так и при наличии в них ошибок.

4. По итогам анализа работы пробной реализации рекомендуется использовать восходящий алгоритм сортировки.

Таблица

Время выполнения пробной реализации (усредненное на 100 сортировок), мс

Сценарий	Восходящий алгоритм	Алгоритм Кана	Алгоритм Тарьяна
Лучший	3	< 1	74
Случайный порядок	20	30	76
Худший	35	50	74

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Вихарев С. М., Федосова Н. М., Батьков Н. В. Информационно-измерительная система на базе разрывной машины РМП-1 // Вестник Костромского государственного технологического университета. – 2008. – № 17. – С. 78–80.
2. Паттерны проектирования / Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс. – СПб. : Питер, 2011. – 670 с. – (Head First).
3. Левитин А. Алгоритмы. Введение в разработку и анализ. – М. : Вильямс, 2006. – 576 с.
4. Алгоритмы: построение и анализ (Introduction to Algorithms) / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. – 3-е изд. – М. : Вильямс, 2013. – 1328 с.
5. Троелсен Э., Джепикс Ф. Язык программирования C# 7 и платформы .NET и .NET Core. – М. : Диалектика, 2018. – 1328 с.

REFERENCES

1. Viharev S. M., Fedosova N. M., Bat'kov N. V. Informacionno-izmeritel'naya sistema na baze razryvnoj mashiny RMP-1 // Vestnik Kostromskogo gosudarstvennogo tekhnologicheskogo universiteta. – 2008. – № 17. – S. 78–80.
2. Patterny proektirovaniya / E. Frimen, E. Frimen, K. S'erra, B. Bejts. – SPb. : Piter, 2011. – 670 s. – (Head First).
3. Levitin A. Algoritmy. Vvedenie v razrabotku i analiz. – M. : Vil'yams, 2006. – 576 s.
4. Algoritmy: postroenie i analiz (Introduction to Algorithms) / T. Kormen, Ch. Lejzerson, R. Rivest, K. Shtajn. – 3-e izd. – M. : Vil'yams, 2013. – 1328 s.
5. Troelsen E., Dzhepiks F. YAzyk programmirovaniya C# 7 i platformy .NET i .NET Core. – M. : Dialektika, 2018. – 1328 s.