Теоретические вопросы информатики, прикладной математики, компьютерных наук и когнитивно-информационных технологий

https://doi.org/10.25559/SITITO.021.202502.166-175
УДК 004.023

Генерация перестановок частично упорядоченного множества с учетом зависимости в элементах

И. В. Андреев*, К. И. Владимиров

ФГАОУ ВО «Московский физико-технический институт (национальный исследовательский университет)», г. Долгопрудный, Российская Федерация

Адрес: 141701, Российская Федерация, Московская область, г. Долгопрудный, Институтский переулок, д. 9

* andreev.iv@phystech.edu

Оригинальная статья

Аннотация

Данная работа посвящена разработке методов, позволяющих существенно сократить количество линейных порядков, необходимых для анализа частично упорядоченных множеств. Такой анализ играет ключевую роль при тестировании многопоточных систем, где важно учитывать все допустимые варианты межпоточного взаимодействия. В настоящее время для тестирования подобных систем применяются как направленные литмус-тесты, специально сконструированные для проверки конкретных эффектов, так и генераторы последовательностей инструкций, порождающие рандомизированные, но простые и удобные для проверки, сценарии исполнения с существенными ограничениями. Применение сложных случайных входных данных затруднено из-за экспоненциального роста числа линейных порядков для анализа. Предлагаемый в данной работе алгоритм позволяет существенно сократить число рассматриваемых линейных порядков за счёт учёта зависимостей между элементами множества при их переупорядочивании. Полученные результаты делают возможным более эффективное тестирование многопоточных систем и расширяют область применения методов анализа частичного порядка.

Ключевые слова: линейные порядки, частично упорядоченные множества, зависимость операций, модели памяти, многопоточные программы, litmus-тесты

Конфликт интересов: авторы заявляют об отсутствии конфликта интересов.

Для цитирования: Андреев И. В., Владимиров К. И. Генерация перестановок частично упорядоченного множества с учетом зависимости в элементах // Современные информационные технологии и ИТ-образование. 2025. Т. 21, № 2. С. 166-175. https://doi.org/10.25559/SITITO.021.202502.166-175

Theoretical Questions of Computer Science, Computational Mathematics, Computer Science and Cognitive Information Technologies

Generation of Permutations of a Partially Ordered Set with Element Dependencies Taken into Account

I. V. Andreev*, K. I. Vladimirov

Moscow Institute of Physics and Technology
(National Research University), Dolgoprudny,
Russian Federation

Address: 9 Institutskiy per., Dolgoprudny 141701,
Moscow Region, Russian Federation

* andreev.iv@phystech.edu

Original article

Abstract

This paper is devoted to the development of methods that make it possible to substantially reduce the number of linear orders required for the analysis of partially ordered sets. Such analysis plays a key role in testing multithreaded systems, where it is important to take into account all admissible variants of inter-thread interaction. At present, testing of such systems relies both on directed litmus tests specifically designed to check particular effects and on instruction-sequence generators that produce randomized yet simple and easy-to-verify execution scenarios with significant constraints. The use of complex random input data is hindered by the exponential growth in the number of linear orders to be analyzed. The algorithm proposed in this paper makes it possible to significantly reduce the number of considered linear orders by taking into account dependencies between set elements when they are reordered. The obtained results enable more efficient testing of multithreaded systems and broaden the scope of methods for partial-order analysis.

Keywords: linear orders, partially ordered sets, operation dependencies, memory models, multithreaded programs, litmus tests

Conflict of interests: The authors declares no conflict of interest.

For citation: Andreev I.V., Vladimirov K.I. Generation of Permutations of a Partially Ordered Set with Element Dependencies Taken into Account. *Modern Information Technologies and IT-Education.* 2025;21(2):166-175. https://doi.org/10.25559/SITITO.021.202502.166-175

Введение

Актуальной задачей в анализе многопоточных программ является порождение различных порядков выполнения операций, возможных при их исполнении. Это имеет важное значение как для доказательства свойств многопоточных алгоритмов [1, 2], так и для проверки корректности компьютерных систем, реализующих такие алгоритмы [3-5].

В общем случае для некоторого набора операций с памятью (N) может наблюдаться экспоненциально большое количество возможных состояний (до $N!$, как показано в [6]). Поэтому при тестировании большого числа операций вводятся строгие ограничения. Например, в работах [7, 8] применяются псевдослучайные тесты [9], содержащие тысячи операций, однако недетерминизм результата исполнения таких тестов существенно снижается за счёт требования уникальности каждой пары значение/адрес. По считанному значению можно однозначно определить, какая операция его записала.

При меньшем числе операций с памятью эти ограничения можно существенно ослабить. Небольшие литмус-тесты [10, 11] предназначены для проверки корректности реализации модели памяти путём выявления допустимых и недопустимых межпоточных взаимодействий в соответствии со спецификацией.

В данной статье мы сосредоточим внимание на построении общего подхода, позволяющего решать задачи анализа как для больших тестов с ограничениями, так и для небольших литмус-подобных тестов без каких-либо ограничений на операции с памятью. При этом анализ многопоточных программ будет сводиться к исследованию частично упорядоченных множеств.

Частично упорядоченные множества

Частично упорядоченное множество [12] – множество из n элементов, упорядоченных антирефлексивным, асимметричным, транзитивным отношением $<_P$. Частично упорядоченное множество может быть представлено в виде ориентированного графа $G = \{V, E\},\ V = \{x \mid x \in P\},\ E = \{(x, y) \mid x, y \in V, x <_p y\}$. Вершины графа соответствуют элементам множества, а ориентированные рёбра отражают отношения порядка между ними.

Отношение покрытия $<_p$ возникает между элементами x и y в том случае, если не существует такого элемента z, что $x <_p z$ и одновременно $z <_p y$. Это можно интерпретировать как отношение ближайшего соседства в частичном порядке.

Линейный порядок элементов — это биекция $\sigma: P \to [n]$ такая, что $x <_p y \to \sigma_x < \sigma_y$. Иначе говоря, это топологическая сортировка графа [13], задаваемого отношениями частичного порядка. Основной целью данной работы является построение линейных порядков. Элемент x, для которого $\nexists y <_p x$, называется **минимальным** в частично упорядоченном множестве.

$$l(P) = \sum_{x \in min(P)} l(P \setminus x) \qquad (1)$$

Количество линейных порядков $l(P)$ задаётся формулой 1 (см. например [14]).

Классы эквивалентности перестановок

Рассмотрим всевозможные линейные перестановки S_n. Две перестановки в S_n считаются эквивалентными, если одна может быть преобразована в другую путём замены подпоследовательности элементов теми же элементами, переставленными определённым образом. Обобщённая теория представлена в работах [15-17].

Например, перестановка $12\mathbf{345}6$ может быть преобразована в $12\mathbf{543}6$ путём замены подпоследовательности $\mathbf{345}$ на подпоследовательность $\mathbf{543}$. Можем сказать, что 123456 эквивалентна 125436 относительно замены $123 \leftrightarrow 321$. Или, используя нотацию множеств: перестановки эквивалентны относительно $\{123, 321\}$.

Пусть $\pi \in S_n$ и пусть $B = \{B_1, \dots, B_t\}$ – заданное разбиение S_k, где $k \le n$. Каждый B_l является шаблоном длиной k, относительно которого можно переставлять элементы. Назовём две перестановки -эквивалентными, если одна может быть получена из другой путём последовательных перестановок, где каждая перестановка σ_i шаблона производится с теми же элементами σ_j шаблона, а σ_i и σ_j принадлежат одному B_l. Тогда $Eq(\pi, B)$ – множество перестановок эквивалентных π относительно B. Таким образом, $125436 \in Eq(123456, \{\{123,321\}\})$.

Независимости в элементах частично упорядоченного множества

В работе [18] было представлено построение всевозможных порядков с учётом независимости элементов. Напомним, что два элемента частично упорядоченного множества $x, y \in P$ называются **независимыми** если $\{xy, yx\} \in B^{|}$. Для дальнейших рассуждений будем рассматривать разбиение $B^{|}$, где $k = 2$.

```
Input:  P – частично упорядоченное множество.
Input:  B| – разбиение из независимых элементов.
Input:  B = {Bn}, где Bn – множество из разбиений
на -ом шаге рекурсии. B(n) = Bn ∈ B
Input:  Path– текущая линейная перестановка.
Input:  y – вершина, обработанная на предыдущем
шаге рекурсии.
1:  function Traverse(P, B|, B, Path, y)
2:    n = |Path|
3:    B(n) = ∅
4:    Bn−2 = B(n − 2)
5:    for all x ∈ min(P) | {xy, yx} ∉ Bn−2
6:      Process(x)            ◁ Обработка нового
элемента в линейном порядке
7:      Traverse(P \ x, B|, B, Path + x, x)
8:      if {xy, yx} ∈ B| then
9:        Bn−2 = {xy, yx} ∪ Bn−2
```

Алгоритм 1. Обход частично упорядоченного множества с учётом независимости

Algorithm 1. Traversal of a partially ordered set taking independence into account

Алгоритм 1 реализует обход частично упорядоченного множества с учётом независимости, представленный ранее в работе [18]. Алгоритм устраняет избыточность в поиске путей, поддерживая для каждого уровня рекурсии n множество B_n. Это множество заполняется на глубине $n+2$ и содержит независимые элементы, уже использованные на этом уровне. При последующем обходе соседних ветвей на уровне n вершины из B_{n-2} исключаются из числа кандидатов, что гарантирует построение уникальных путей и предотвращает дублирующие вычисления [19].

Применение цепочки последовательных эквивалентных перестановок порождает перестановку, эквивалентную данной. Рассмотрим это на примере. Пусть $B^{|} = \{\{01,10\},\{02,20\}\}$ и $\pi = 012$. Тогда $012 \overset{Eq(\pi,B^{|})}{\Longleftrightarrow} 102 \overset{Eq(\pi,B^{|})}{\Longleftrightarrow} 120$. Таким образом, $012 \overset{Eq(\pi,B)}{\Longleftrightarrow} 120$, где $B = \{012,120\} \cup B^{|}$.

Построение и анализ множества B, генерируемого из $B^{|}$ последовательностью перестановок, является отдельной задачей и в данной работе не рассматривается. Далее будет рассматриваться только множество $B^{|}$.

Зависимости в элементах частично упорядоченного множества

Назовём два элемента частично упорядоченного множества $x, y \in P$ **зависимыми**, если $\{xy, yx\} \notin B^{|}$. То есть, для этих элементов существует такой относительный порядок, который порождает перестановку, не принадлежащую классу эквивалентности оригинальной перестановки. Множество пар зависимых элементов будем обозначать как D. Пусть $\{x,y\} \in D, \{a,x\}, \{a,y\} \notin D$. Тогда

- $B = \{\{ax, xa\}, \{ay, ya\}\}$;
- $xay, xya \in Eq(axy, B)$;
- $yax, yxa \in Eq(ayx, B)$.

Обозначим l_P – линейный порядок множества P. Q, W, E – подмножества P, $a \in P$. Q, E могут быть $\emptyset$. И выполнены условия

- $Q \cap W \cap E \cap \{a\} = \emptyset$;
- $\forall x \in W \vDash \{x, a\} \notin D$.

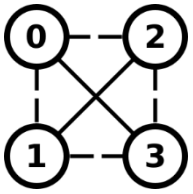
Тогда

$$\{\boldsymbol{a} l_W, l_W \boldsymbol{a}\} \in B$$
$$l_Q l_W \boldsymbol{a} l_E \in Eq\left(l_Q \boldsymbol{a} l_W l_E, B\right) \quad (2)$$

Другими словами, если переставить элемент $\boldsymbol{a}$ относительно любого порядка независимых элементов, то результатом будет порядок, принадлежащий тому же классу эквивалентности. Отношение $B^{|}$ строится на основе рёбер независимости между элементами. Если два элемента x и y независимы, то существует ребро независимости между ними, и в этом случае $\{xy, yx\} \in B^{|}$.

Отношения зависимости и независимости элементов множества могут быть представлены в виде графов, при этом оба графа будут иметь общее множество вершин. Множество вершин: $V = \{x \mid x \in P\}$. Рёбра зависимости: $E_{dep} = \{\{x,y\} \mid \{x,y\} \in D\}$, рёбра независимости – $E_{indep} = \{\{x,y\} \mid \{xy, yx\} \in B^{|}\}$. Таким образом, граф зависимости будет представлен как $G(V, E_{dep})$, а граф независимости как $G(V, E_{indep})$. В связи с анализом только пар независимых операций, то есть только соседних в текущем линейном порядке, множество переупорядочивания может по-прежнему содержать эквивалентные перестановки.

$$B \supseteq B^{|} \quad (3)$$



Р и с. 1. Частично упорядоченное множество с рёбрами независимости (пунктирная линия) и рёбрами зависимости (сплошная линия)
F i g. 1. Partially ordered set with independence edges (dashed lines) and dependency edges (solid lines)

Источник: здесь и далее в статье все рисунки составлены авторами.
Source: Hereinafter in this article all figures were drawn up by the authors.

Рассмотрим тот же пример, что был представлен в работе [18]. На рисунке 1 изображено частично упорядоченное множество, в котором отсутствуют элементы, связанные отношением порядка, но присутствуют отношения зависимости и независимости.

Полное множество разбиений B (см. 4), которое можно построить с учётом множества пар зависимых элементов D, содержит всевозможные эквивалентные разбиения. Множество разбиений $B^{|}$, построенное с учётом отношения независимости (см. 5), обладает значительно меньшим размером по сравнению с полным множеством разбиений, не теряя при этом информации, необходимой для анализа.

$$B = \{\{01,10\},\{02,20\},\{31,13\},\{23,32\},$$
$$\{031,103\},\{032,203\},\{120,012\},\{123,312\},$$
$$\{301,130\},\{302,230\},\{210,021\},\{213,321\}\} \quad (4)$$

$$B^{|} = \{\{01,10\},\{02,20\},\{31,13\},\{23,32\}\} \quad (5)$$

```
1 :   0123  |  0123
2 :   0213  |  0213
3 :   1230  |  1230
4 :   2130  |  1203 (0123)
5 :         |  1302 (1230)
6 :         |  2103 (0213)
7 :         |  2130
8 :         |  2301 (2130)
9 :         |  3012 (1302)
10:         |  3021 (2301)
```

Листинг 1. Линейные порядки для B (слева) и $B^{|}$ (справа)
Listing 1. Linear orders for B (left) and $B^{|}$ (right)

Всё множество линейных порядков с учётом эквивалентности $Eq(\pi, B)$ представлено слева, а с учётом $Eq(\pi, B^{|})$ – справа на листинге 1. Справа в скобках показаны порядки, принадлежащие тому же классу $Eq(\pi, B)$. На этом примере видна избыточность вычислений при учёте только соседних элементов перестановки.

Построение всевозможных линейных порядков с учётом зависимости

Рассмотрим частично упорядоченное множество P и множество зависимых элементов D. За основу обхода частично упорядоченного множества возьмём топологическую сортировку. На каждом шаге при этом будем строить два новых частично упорядоченных множества $P^{\rightarrow}$ и $P^{\leftarrow}$, а также обновлять множество зависимых элементов D^* по правилам 6.

$$\begin{aligned} D^* &= D \setminus \{x, y\} \\ P^{\rightarrow} &= P \cup \left(x <_p y\right) \\ P^{\leftarrow} &= P \cup \left(y <_p x\right) \end{aligned} \tag{6}$$

Изменим шаг рекурсии при построении нового линейного порядка, а именно — выбор минимального элемента. Все минимальные элементы частично упорядоченного множества P формируют **множество минимальных элементов** $min(P)$. Шаг рекурсии будет выглядеть следующим образом: $P^* = P \setminus min(P)$.

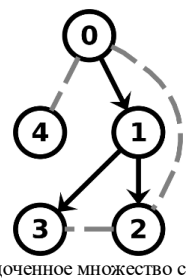
Все описанные шаги реализуются в алгоритме топологической сортировки с учётом зависимости, представленном на алгоритме 2.

```
Input:   P – частично упорядоченное множество.
Input:   D – множество зависимых элементов.
Input:   Path– текущая линейная перестановка.
1 :   function Traverse(P, D, Path)
2 :     X | ∀x ∈ X ⇒ x ∈ min(P)
3 :     if X = ∅, P ≠ ∅ then
4 :       return                                   ◁
Найден цикл
5 :     if ∀x ∈ X, y ∈ P ⊨ {x, y} ∉ D then
6 :       Process(X)                               ◁
Обработка множества X
7 :       Traverse(P \ X, D, Path + X)             ◁
Новый шаг рекурсии с меньшим P
8 :     for all x ∈ X, y ∈ P | {x, y} ∈ D do
9 :       D* = D \ {x, y}
10:       P→ = P ∪ (x <_p y)
11:       P← = P ∪ (y <_p x)
12:       Traverse(P→, D*, Path)
13:       Traverse(P←, D*, Path)
```

Алгоритм 2. Обход частично упорядоченного множества с учётом зависимости
Algorithm 2. Traversal of a partially ordered set taking dependency into account

На строке **6** алгоритма 2 порядок элементов из множества минимальных X не важен. Из условия $\forall x \in X, y \in P \vDash (x, y) \notin D$ следует, что $\{xy, yx\} \in B^{|}$. По формуле 2 любые перестановки элементов из X будут входить в один класс эквивалентных перестановок.

Пример

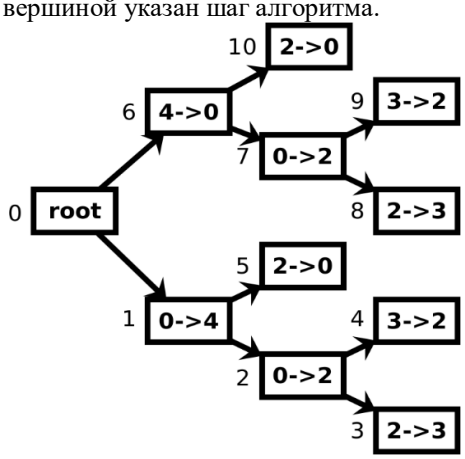


Р и с. 2. Частично упорядоченное множество с зависимыми элементами
F i g. 2. Partially ordered set with dependent elements

Работу алгоритма 2 рассмотрим на примере частично упорядоченного множества G (см. 7), представленного в виде смешанного графа на рисунке 2. Пронумерованные вершины графа соответствуют элементам множества, ориентированные рёбра отображают строгий порядок элементов, а неориентированные рёбра (изображённые пунктиром) – зависимые элементы.

$$\begin{aligned} G &= \left(V, E_{order}, E_{dep}\right), \\ V &= \{x \mid x \in P\}, \\ E_{order} &= \{(x, y) \mid x <_p y\}, \\ E_{dep} &= \{\{x, y\} \mid \{x, y\} \in D\}. \end{aligned} \tag{7}$$

На строках 11 и 12 алгоритма 2 формируются новые частично упорядоченные множества с дополнительным строгим порядком между зависимыми операциями: $x <_p y$ или $y <_p x$. Такой обход можно удобно представить в виде бинарного дерева[1], как показано на рисунке 3. Вершины дерева соответствуют выбору направления неориентированного ребра, а рядом с каждой вершиной указан шаг алгоритма.



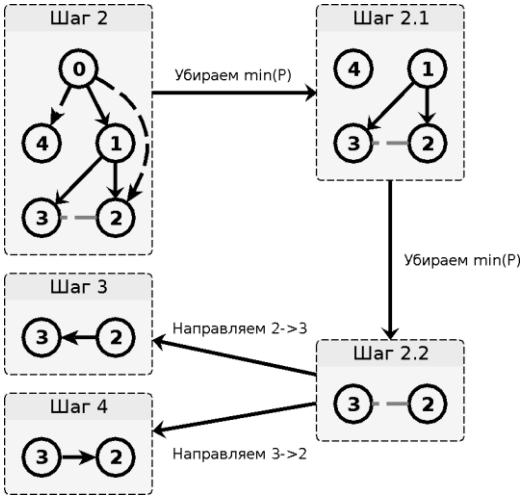
Р и с. 3. Бинарное дерево обхода частично упорядоченного множества, представленного на рис. 2
F i g. 3. Binary traversal tree of the partially ordered set shown in Fig. 2

Р и с. 4. Частично упорядоченное множество на шагах 5 и 10
F i g. 4. Partially ordered set at steps 5 and 10

На рисунке 3 представлено дерево, в котором можно заметить уникальные состояния частично упорядоченного множества на шагах 5 и 10. Во время этих шагов частично упорядоченное множество выглядит так, как показано на рисунке 4. Этому состоянию соответствует цикл [20] $0 \rightarrow 1 \rightarrow 2 \rightarrow 0$, что является некорректным состоянием. Поэтому направление ребра $2 \rightarrow 0$ невозможно, и нет смысла продолжать обход. На данном примере видно, что направление $2 \rightarrow 0$ заведомо неверное, так как $0 <_p 2$. Таким образом, множество D можно уменьшить, убрав из него подмножество D_{order} (см. 8).

$$D_{order} = \{\{x, y\} \in D \mid (x, y) \in E^+_{order} \vee (y, x) \in E^+_{order}\} \tag{8}$$

[1] Aho A.V., Hopcroft J.E., Ullman J.D. Data structures and algorithms. MA, USA: Addison-wesley Boston, 1983. Vol. 175. 180 p.



Р и с. 5. Обзор шага 2
F i g. 5. Overview of step 2

Разберём более детально отдельные шаги алгоритма на примере из рисунка 5. На втором шаге мы имеем частично упорядоченное множество, которое удовлетворяет условию $\forall x \in min(P), y \in P \vDash (x, y) \notin D$. Можно удалить минимальное множество $min(P) = \{0\}$ из частично упорядоченного множества и перейти к шагу 2.1. Здесь снова выполняется условие, что никакие элементы минимального подмножества $min(P) = \{1, 4\}$ не имеют зависимых операций, поэтому можно уменьшить частично упорядоченное множество. На шаге 2.2 $min(P) = \{2,3\}$, но $\{2,3\} \in D$, поэтому вначале необходимо направить неориентированное ребро. Направляем $2 \to 3$ и переходим к шагу 3. Направляем $3 \to 2$ и переходим к шагу 4.

За один шаг ориентирования рёбер можно обработать сразу несколько элементов частично упорядоченного множества. При выборе элементов сохраняется строгий порядок: $0 \to 1$, $0 \to 4$, а любое переупорядочивание элементов без отношения зависимости друг с другом будет принадлежать одному классу эквивалентности. Поэтому будет достаточно рассмотреть только один порядок, например $0 \to 1 \to 4$.

Асимптотика

Так как порядок определяется путём выбора направления неориентированного ребра из E_{dep}, то количество направлений равно $2^{|D|}$, асимптотическая сложность алгоритма задаётся соотношением 9, где D — множество зависимых элементов, n — количество элементов частично упорядоченного множества.

$$O\left(2^{|D|}n\right) \quad (9)$$

Возможны случаи, когда $\exists\{x, y\} \in D \mid x <_p y$. То есть зависимые элементы обладают отношением порядка. В этом случае нет смысла строить $P^{\leftarrow} = P \cup \left(y <_p x\right)$, так как оно заведомо будет обладать циклом. Множество D можно минимизировать из условия $D_{min} = \{\{x, y\} \in D \mid x \not<_p y \wedge y \not<_p x\}$.

Общее количество перестановок не изменится, так как цикл обработается на строке 3 алгоритма 2, однако уменьшение множества D уменьшает показатель степени в формуле 9.

Результаты

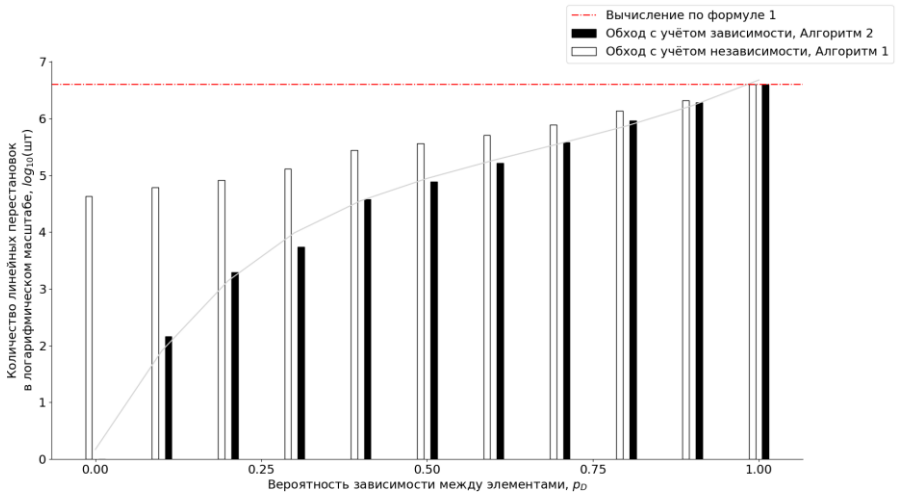
Обозначим p_D – вероятность зависимости между элементами множества P.

$$p_D = p_D(x, y) = \mathbb{P}[\{x, y\} \in D],\ \forall x, y \in P \quad (10)$$

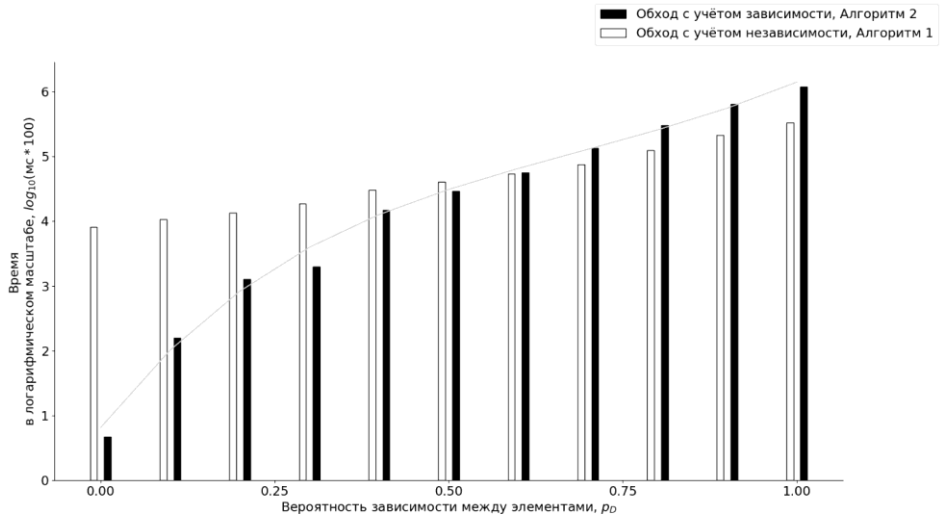
И p_P – вероятность отношения порядка между элементами множества P.

$$p_P = p_P(x, y) = \mathbb{P}\left[x <_p y\right],\ \forall x, y \in P \quad (11)$$

Из формулы 3 следует, что количество линейных порядков при обходе с учётом зависимости всегда меньше либо равно, чем при обходе с учётом независимости. Кроме того, по построению, количество линейных порядков всегда равно количеству классов эквивалентности, что означает минимальное возможное количество для данного частично упорядоченного множества.



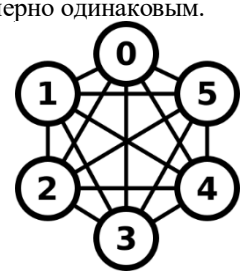
Р и с. 6. График зависимости количества линейных порядков в логарифмическом масштабе от вероятности зависимости элементов для обходов с учётом зависимости и независимости. $|P| = 13, p_P = 0.3$.
F i g. 6. Plot of the number of linear orders on a logarithmic scale versus the probability of element dependency for dependency-aware and independence-aware traversals. $|P| = 13, p_P = 0.3$.



Р и с. 7. График зависимости времени построения линейных порядков в логарифмическом масштабе от вероятности зависимости элементов для обходов с учётом зависимости и независимости.
F i g. 7. Plot of the time required to construct linear orders on a logarithmic scale versus the probability of element dependency for dependency-aware and independence-aware traversals.
$|P| = 13$, $p_P = 0.3$.

На графике 6 показано, что количество линейных порядков для одной и той же конфигурации частично упорядоченного множества меньше при обходе с учётом зависимостей, по сравнению с обходом с учётом независимостей.

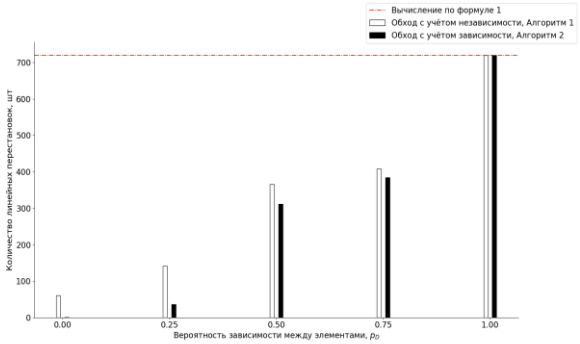
Для обхода с учётом зависимости необходим анализ множества D, а для обхода с учётом независимости — множества $B^{|}$. Из графика 7 видно, что при малом p_D обход с учётом независимости значительно уступает обходу с учётом зависимости. Это объясняется тем, что $E_{dep} = \overline{E_{indep}}$[21]. Обход $G(V, E_{dep})$ более выгоден, когда $|D| < |B^{|}|$, и наоборот. В граничном случае, когда $|B| \sim |B^{|}|$, количество линейных порядков при обоих обходах становится примерно одинаковым.



Р и с. 8. Частично упорядоченное множество. |P| = 6, |D| = 15. Без рёбер порядка. С рёбрами зависимости.
F i g. 8. Partially ordered set. |P| = 6, |D| = 15. Without order edges. With dependency edges.

На графике 9 можно пронаблюдать граничный случай с $p_D = 1.0$, при котором частично упорядоченное множество без отношений порядка, но с полным графом [22] зависимостей. Такое множество представлено на рисунке 8. Каждый элемент зависит от каждого. В случае, если разреженное частично упорядоченное множество можно разделить на несвязные подмножества A и B (произвольным образом), то формулу 1 можно преобразовать в вид 12, см. работу [12].

$$l(P) = l(A) * l(B) * \binom{|A|}{|B|} \qquad (12)$$



Р и с. 9. График зависимости количества линейных порядков от вероятности зависимости элементов для обходов с учётом зависимости и независимости. |P |=6, p_P=0.0
F i g. 9. Plot of the number of linear orders versus the probability of element dependency for dependency-aware and independence-aware traversals. |P |=6, p_P=0.0

Для множества, представленного на рисунке 8, формула упрощается до $l(P) = |P|! = 6! = 720$, что подтверждается результатом на графике 9. При $p_D \sim 1$, как видно из графика 7, время построения линейных порядков при анализе с учётом зависимостей оказывается больше, чем при анализе с учётом независимостей. Это связано с построением и обработкой ситуаций, когда частично упорядоченное множество содержит цикл, как на строчке 6 алгоритма 2. Оба алгоритма сводятся к топологической сортировке, и для обхода с учётом зависимости становится невозможным обработать сразу несколько элементов множества за один раз.

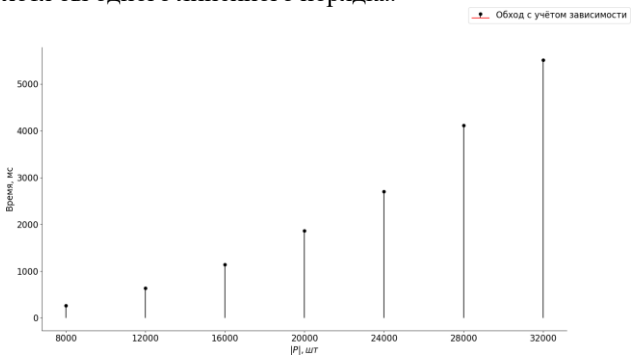
Применение обхода с учётом зависимости к тестированию многопоточных систем

В работе [8] генерировался псевдослучайный тест со следующими ограничениями:

- невозможно частичное пересечение адресов обращений в память;
- пары адрес/значения уникальны для каждой записи в память.

Проверка трассы заключается в упорядочивании операций $M[x] == v$ и $M[x] := v$ и построении топологической сортировки.

Частично упорядоченное множество определяется моделью памяти[2] [23-25]. Если результат операции b не может быть наблюдаем до выполнения операции a, то $a < b$. Пара адрес/значение задаёт отношение зависимости между операциями. Условие уникальности пар адрес/значение при верификации с использованием алгоритма 2 однозначно определяет порядок между операциями на момент выбора направления (либо $P^{\rightarrow}$, либо $P^{\leftarrow}$). В этом случае проверка состояния трассы сводится к однозначному упорядочиванию всех зависимых пар элементов, что эквивалентно поиску хотя бы одного линейного порядка.



Р и с. 10. График времени построения одного линейного порядка от количества элементов множества. $p_P = 0.001$

F i g. 10. Plot of the time required to construct one linear order versus the number of elements in the set. p_P = 0.001

Промоделируем эксперимент, представленный в работе [8]. Предположим, что половина всех операций с памятью – это чтение, а другая половина – запись. Во время верификации цикл между операциями $M[x] == v$ и $M[x] := v$ возможен только в случае некорректного поведения тестируемой системы. Алгоритм 2 допускает циклы, если возможны другие порядки чтения/записи, однако в контексте построения псевдослучайного теста такие порядки невозможны. Для моделирования отсутствия циклов используем сильно разряженное частично упорядоченное множество с параметром $p_P = 0.001$. Уникальность пар адрес/значение моделируется наличием только одного ребра между чтением и записью.

На графике 10 представлены результаты построения одного порядка в зависимости от количества элементов множества. Как видно, обход с учётом зависимости можно использовать для проверки нескольких тысяч операций с памятью. Также, при отсутствии строгих ограничений на операции, можно верифицировать небольшие литмус-подобные тесты с высокой интерференцией между операциями.

Заключение

В данной работе был продемонстрирован алгоритм построения линейных порядков частично упорядоченного множества с учётом зависимости элементов множества. Особое внимание уделено сравнению времени построения линейных порядков при обходе с учётом зависимости и независимости. Для значений $p_D \lesssim 0.5$ было показано существенное улучшение в производительности алгоритма при учёте зависимости, что значительно ускоряет процесс построения порядков по сравнению с обходом с учётом независимости.

Дополнительно ограничением алгоритма для практического применения является максимальный размер входных данных. Например, при $p_D = 0.1$, для практического использования $|P|$ ограничен числом порядка $50 - 100$, тогда как для $p_P \sim p_D \sim 0.001$ уже можно обрабатывать тысячи элементов (т.е. метод лучше всего подходит для обработки разреженных графов).

Предложенный алгоритм позволяет расширить область применения на большие частично упорядоченные множества и может быть использован для верификации как малых тестов без искусственных ограничений, так и более масштабных тестов с тысячами операций с памятью. Работу можно рассматривать как шаг к более эффективным методам анализа подсистемы памяти в многоядерных вычислительных устройствах, где зависимости между операциями играют ключевую роль.

Данный подход демонстрирует свою расширяемость и потенциал для дальнейших улучшений, включая оптимизацию для ещё более сложных систем и задач, а также для более крупных и разнообразных тестов с высокой интерференцией между операциями. В дальнейшем исследования могут быть направлены на улучшение алгоритма, его оптимизацию для работы с более сложными моделями памяти, а также на разработку практических инструментов на основе данного подхода для анализа и верификации многопоточных программ.

[2] Sorin D. J., Hill M. D., Wood D. A. A Primer on Memory Consistency and Cache Coherence. Morgan & Claypool Publishers, 2011. 210 p.

References

1. Rinard M. Analysis of Multithreaded Programs. In: Cousot P. (eds) Static Analysis. SAS 2001. *Lecture Notes in Computer Science*. Vol. 2126. Berlin, Heidelberg: Springer; 2001. p. 1-19. https://doi.org/10.1007/3-540-47764-0_1
2. Madduri K., Feo J., Cong G., Bader D.A. Design of Multithreaded Algorithms for Combinatorial Problems. In: Rajasekaran S., Reif J.H. (eds) Handbook of Parallel Computing – Models, Algorithms and Applications. Chapman and Hall/CRC; 2007. p. 787-816. https://doi.org/10.1201/9781420011296.ch31
3. Petrochenkov M., Stotland I., Mushtakov R. Approaches to Stand-alone Verification of Multicore Microprocessor Caches. *Trudy ISP RAN* = Proceedings of the Institute for System Programming of the RAS. 2016;28(3):161-172. https://doi.org/10.15514/ISPRAS-2016-28(3)-10
4. Bennett A.J., Field T., Harrison P. Modelling and validation of shared memory coherency protocols. *Performance evaluation*. 1996;27-28:541-563. https://doi.org/10.1016/S0166-5316(96)90045-0
5. Petrot F., Greiner A., Gomez P. On Cache Coherency and Memory Consistency Issues in NoC Based Shared Memory Multiprocessor SoC Architectures. In: Proceedings of the 9th EUROMICRO Conference on Digital System Design (DSD '06). IEEE Computer Society, USA; 2006. p. 53-60. https://doi.org/10.1109/DSD.2006.73
6. Björner A., Wachs M.L. Permutation statistics and linear extensions of posets. *Journal of Combinatorial Theory, Series A*. 1991;58(1):85-114. https://doi.org/10.1016/0097-3165(91)90075-R
7. Hangal S., Vahia D., Manovit C., Lu J.-Y.J. TSOtool: A Program for Verifying Memory Systems Using the Memory Consistency Model. *ACM SIGARCH Computer Architecture News*. 2004;32(2):114. https://doi.org/10.1145/1028176.1006710
8. Naylor M., Moore S.W., Mujumdar A. A consistency checker for memory subsystem traces. In: Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design (FMCAD '16). Austin, Texas: FMCAD Inc; 2016. p. 133-140.
9. Wagner, Chin, McCluskey. Pseudorandom Testing. *IEEE Transactions on Computers*. 1987;C-36(3):332-343. https://doi.org/10.1109/TC.1987.1676905
10. Mador-Haim S., Alur R., Martin M.M.K. Litmus tests for comparing memory consistency models: how long do they need to be? In: Proceedings of the 48th Design Automation Conference (DAC '11). New York, NY, USA: Association for Computing Machinery; 2011. p. 504-509. https://doi.org/10.1145/2024724.2024842
11. Mador-Haim S., Alur R., Martin M.M.K. Generating Litmus Tests for Contrasting Memory Consistency Models. In: Touili T., Cook B., Jackson P. (eds.) Computer Aided Verification. CAV 2010. *Lecture Notes in Computer Science*. Vol. 6174. Berlin, Heidelberg: Springer; 2010. p. 273-287. https://doi.org/10.1007/978-3-642-14295-6_26
12. Kangas K., Hankala T., Niinimaki T., Koivisto M. Counting Linear Extensions of Sparse Posets. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI'16). New York, New York, USA: AAAI Press, 2016. p. 603-609.
13. Kahn A.B. Topological sorting of large networks. *Communications of the ACM*. 1962;5(11):558-562. https://doi.org/10.1145/368996.369025
14. Brightwell G., Winkler P. Counting linear extensions. *Order*. 1991;8:225-242. https://doi.org/10.1007/BF00383444
15. Knuth D. Permutations, matrices, and generalized Young tableaux. *Pacific Journal of Mathematics*. 1970;34(3):709-727.
16. Haiman M.D. Dual equivalence with applications, including a conjecture of Proctor. *Discrete Mathematics*. 1992;99(1-3):79-113.
17. Robinson G. de B. On the representations of the symmetric group. *American Journal of Mathematics*. 1938:745-760.
18. Andreev I.V., Vladimirov K.I. Efficient Reordering of Multiple Memory Operations in a Multithreaded Program. *Modern Information Technologies and IT-Education*. 2024;20(1):149-156. (In Russ., abstract in Eng.) https://doi.org/10.25559/SITITO.020.202401.149-156
19. Andreev I.V., Vladimirov K.I. Efficient Reordering of Multiple Memory Operations in a Multithreaded Program. *Modern Information Technologies and IT-Education*. 2024;20(1):149-156. (In Russ., abstract in Eng.) https://doi.org/10.25559/SITITO.020.202401.149-156
20. Hendry G.R.T. Extending cycles in graphs. *Discrete Mathematics*. 1990;85(1):59-72. https://doi.org/10.1016/0012-365X(90)90163-C
21. Corneil D.G., Lerchs H., Burlingham L.S. Complement reducible graphs. *Discrete Applied Mathematics*. 1981;3(3):163-174.
22. Beineke L.W., Harary F. The thickness of the complete graph. *Canadian Journal of Mathematics*. 1965;17:850-859.
23. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE transactions on computers*. 1979;C-28(9):690-691. https://doi.org/10.1109/TC.1979.1675439
24. Oberhauser J. Store Buffer Reduction in the Presence of Mixed-Size Accesses and Misalignment. In: Piskac R., Rümmer P. (eds.) Verified Software. Theories, Tools, and Experiments. VSTTE 2018. *Lecture Notes in Computer Science*. Vol. 11294. Cham: Springer; 2018. p. 322-344. https://doi.org/10.1007/978-3-030-03592-1_19

25. Cohen E., Schirmer B. From Total Store Order to Sequential Consistency: A Practical Reduction Theorem. In: Kaufmann M., Paulson L.C. (eds.) Interactive Theorem Proving. ITP 2010. *Lecture Notes in Computer Science*. Vol. 6172. Berlin, Heidelberg: Springer; 2010. p. 403-418. https://doi.org/10.1007/978-3-642-14052-5_28

Поступила 15.03.2025; одобрена после рецензирования 20.04.2025; принята к публикации 12.06.2025.

Submitted 15.03.2025; approved after reviewing 20.04.2025; accepted for publication 12.06.2025.

Об авторах:

Андреев Илья Витальевич, аспирант кафедры микропроцессорных технологий в интеллектуальных системах факультета радиотехники и кибернетики, ФГАОУ ВО «Московский физико-технический институт (национальный исследовательский университет)» (141701, Российская Федерация, Московская область, г. Долгопрудный, Институтский переулок, д. 9), **ORCID: https://orcid.org/0000-0001-6450-7917,** andreev.iv@phystech.edu

Владимиров Константин Игоревич, старший преподаватель кафедры микропроцессорных технологий в интеллектуальных системах факультета радиотехники и кибернетики, ФГАОУ ВО «Московский физико-технический институт (национальный исследовательский университет)» (141701, Российская Федерация, Московская область, г. Долгопрудный, Институтский переулок, д. 9), **ORCID: https://orcid.org/0000-0003-0925-1300,** konstantin.vladimirov@gmail.com

Все авторы прочитали и одобрили окончательный вариант рукописи.

About the authors:

Ilya V. Andreev, Postgraduate Student of the Chair of Microprocessor Technologies in Intelligent Systems, Department of Radio Engineering and Cybernetics, Moscow Institute of Physics and Technology (National Research University) (9 Institutskiy per., Dolgoprudny 141701, Moscow Region, Russian Federation), **ORCID: https://orcid.org/0000-0001-6450-7917,** andreev.iv@phystech.edu

Konstantin I. Vladimirov, Senior Lecturer of the Chair of Microprocessor Technologies in Intelligent Systems, Department of Radio Engineering and Cybernetics, Moscow Institute of Physics and Technology (National Research University) (9 Institutskiy per., Dolgoprudny 141701, Moscow Region, Russian Federation), **ORCID: https://orcid.org/0000-0003-0925-1300,** konstantin.vladimirov@gmail.com

All authors have read and approved the final manuscript.