ПРОБЛЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

Том 60 **2024** **Вып. 4**

УДК 510.51 : 004.932 : 519.6 : 519.171

© 2024 г. **Д.Д. Казимиров, Д.П. Николаев, Е.О. Рыбакова, А.П. Терехин**

БЫСТРЫЙ АЛГОРИТМ ВЫЧИСЛЕНИЯ ПРЕОБРАЗОВАНИЯ ХАФА ДЛЯ ИЗОБРАЖЕНИЙ ПРОИЗВОЛЬНОГО РАЗМЕРА С ПЕРЕИСПОЛЬЗОВАНИЕМ ВЫДЕЛЕННОЙ ПАМЯТИ

In-place алгоритмы эффективно используют память, уже выделенную для входных данных, ограничиваясь лишь незначительным дополнительным объемом памяти для промежуточных вычислений. Для изображений ширины, равной степени двойки, известен in-place алгоритм, являющийся вариацией стандартного алгоритма Брейди – Ёна для вычисления преобразования Хафа. Однако этот алгоритм неприменим к изображениям с произвольной шириной, наиболее часто встречающимся на практике. Напротив, out-of-place алгоритм $FHT2DS$ может обрабатывать изображения различных размеров. В настоящей статье представлен in-place вариант алгоритма $FHT2DS$, названный $FHT2IDS$. Мы показываем, что алгоритм $FHT2IDS$ дает такие же результаты, как и алгоритм $FHT2DS$, но использует значительно меньше памяти на каждом шаге рекурсии. В частности, на каждом шаге рекурсии алгоритм $FHT2IDS$ требует массива размера не более $w+h$ (где w и h – ширина и высота изображения), в то время как алгоритм $FHT2DS$ требует массива размера wh. Экспериментальные результаты показывают, что алгоритм $FHT2IDS$, реализованный на C/C++, работает на 26% быстрее своего out-of-place аналога, алгоритма $FHT2DS$. Алгоритм $FHT2IDS$ также доступен на Python через открытый исходный код библиотеки *adrt*.

Ключевые слова: преобразование Хафа, быстрое преобразование Хафа, приближенное дискретное преобразование Радона, in-place, вычислительный граф.

DOI: 10.31857/S0555292324040065, **EDN:** VKDONT

§ 1. Введение

Преобразование Хафа (ПХ) широко используется в обработке изображений и машинном зрении. Оно обычно рассматривается как инструмент для робастной оценки параметров одной или нескольких прямых на дискретном изображении путем определения числа точек, лежащих на каждой из заранее заданных прямых. Этот метод был предложен Полом Хафом в 1959 году как способ обнаружения прямолинейных траекторий в экспериментах с пузырьковыми камерами [1]. Основной принцип преобразования Хафа заключается в накоплении “голосов” вдоль дискретизированных прямых в заданной параметризации и присвоении накопленного значения каждой прямой. Это накопленное значение увеличивается с вероятностью присутствия соответствующей прямой на изображении.

Исторически преобразование Хафа (ПХ) в первую очередь было известно благодаря своему применению для обнаружения контрастных прямых или их сегментов на изображениях [2–5]. Однако с течением времени область его применения значительно расширилась. Преобразование Хафа используется в различных областях, включая бинаризацию изображений [6] и сегментацию [7,8], а также для таких задач,

как автоматическое определение параметров аберрации оптической системы [9] и робастная ортогональная линейная регрессия на низкоразмерных гистограммах [10].

Преобразование Хафа требует использования быстрых вычислительных алгоритмов (алгоритмы быстрого преобразования Хафа, или БПХ). В 1992 году М. Брейди и В. Ён разработали метод для приближенного дискретного преобразования Радона [11], предназначенный для квадратных изображений размера $2^q \times 2^q, q \in \mathbb{N}$. В аннотации и заключении своей работы авторы взаимозаменяемо используют термины "преобразование Хафа" и "преобразование Радона". Их подход основывается на динамическом программировании, что позволяет исключить избыточные вычисления сумм для ранее обработанных сегментов при определении суммы вдоль соответствующих прямых. Это приводит к эффективному вычислению преобразования Хафа для изображения размера $n \times n$ за $\mathcal{O}(n^2 \log n)$ операций сложения. Оценка ортотропной ошибки аппроксимации прямых диадическими дискретизациями, используемыми в методе Брейди – Ёна, была подробно изучена в литературе [12]. Метод Брейди – Ёна стал стандартным подходом для практических применений преобразования Хафа, так как он предоставляет вычислительно эффективное решение, сохраняя при этом основную идею оригинальной техники [13–18].

Широкая применимость преобразования Хафа на практике накладывает строгие требования к производительности и объему памяти на алгоритмы БПХ, особенно во встроенных системах с ограниченными вычислительными ресурсами [13–16, 19, 20] или, как правило, в менее мощных периферийных устройствах [17, 21–23]. Эти требования к производительности и памяти становятся особенно важными в парадигме интернета вещей, в которой вычисления выполняются локально на процессорах, а не на серверных машинах [21, 24, 25].

На практике наиболее часто встречаются изображения с шириной, не являющейся степенью двойки. Стандартный алгоритм Брейди – Ёна неприменим к таким изображениям. В литературе были предложены обобщения алгоритма Брейди – Ёна для изображений произвольного размера [26–28]. Однако эти подходы все еще не могут быть внедрены в системы с жесткими аппаратными ограничениями и требованиями к реальному времени. Предложенные алгоритмы неэффективно используют память и не соответствуют строгим требованиям по скорости и объему выделяемой памяти [21, 24, 25, 29, 30]. Избыточная дополнительная память, требуемая алгоритмами для их корректного исполнения, остается преградой для применения этих алгоритмов к большим изображениям на портативных устройствах, что подчеркивает критическую необходимость разработки быстрых in-place алгоритмов для вычисления БПХ.

In-place алгоритм по определению не использует дополнительную память для обработки входных данных, но может требовать небольшого объема памяти, который может быть как постоянным $\mathcal{O}(1)$, так и непостоянным, например, $\mathcal{O}(\log n)$, для работы с входными данными размера n [31, 32]. Иногда допускается использование памяти, меньшей, чем $\mathcal{O}(n)$. In-place алгоритмы часто работают быстрее, чем их out-of-place аналоги [32], поскольку они оптимизируют использование памяти [33, 34], снижают накладные расходы на копирование данных [32, 33] и повышают эффективность работы с кэшем [32, 35]. Изменяя данные непосредственно в исходном месте в памяти, in-place алгоритмы уменьшают необходимость в дополнительном выделении памяти, что может привести к снижению количества операций с памятью и улучшению локальности данных в кэше. Последнее, в свою очередь, снижает задержки доступа и ускоряет выполнение алгоритма [32, 35]. In-place подход переиспользования исходно выделенной памяти для промежуточных вычислений особенно выгоден в условиях ограниченности ресурсов, когда минимизация объема используемой памяти имеет решающее значение. Out-of-place алгоритмы могут обращаться к памяти чрезмерное количество раз и могут привести к существенно-

му замедлению выполнения алгоритма из-за обмена данными между оперативной памятью и диском [36]. Кроме того, in-place алгоритмы избегают сложностей, связанных с управлением множественными копиями данных, что снижает накладные расходы на выделение памяти, инициализацию и очистку [32]. Среди известных примеров in-place алгоритмов можно выделить алгоритм Кули – Тьюки для вычисления одномерного быстрого преобразования Фурье (БПФ) [37], а также in-place версию алгоритма Брейди – Ёна для БПХ для изображений с ширинами, являющимися степенями двойки [38].

В статье [27] исследуются два алгоритма БПХ для изображений произвольного размера – алгоритмы $FHT2DS$ и $FHT2DT$. Показано, что с точки зрения количества выполняемых суммирований алгоритм $FHT2DS$, первоначально предложенный в работе [26], является более быстрым. Использование in-place модификации алгоритма $FHT2DS$ может привести к дополнительным улучшениям как в вычислительной скорости, так и в объеме вспомогательной памяти, что видится шагом на пути к его широкому использованию во встраиваемых устройствах или системах в интернете вещей. В настоящей статье предлагается алгоритм $FHT2IDS$ – in-place модификация алгоритма $FHT2DS$. Представлены теоретические и экспериментальные исследования свойств данного алгоритма.

Статья имеет следующую структуру. Параграф 2 посвящен описанию алгоритма $FHT2DS$ и перечислению его ключевых характеристик. В § 3 вводится графовое представление алгоритма $FHT2DS$, в § 4 рассматривается in-place свойство алгоритмов с точки зрения теории графов. Доказательства важных свойств вычислительного графа алгоритма $FHT2DS$, основанные на понятиях и определениях из предыдущих параграфов, изложены в § 5. Доказанные свойства позволяют описать in-place алгоритм $FHT2IDS$ в § 6. Далее, в § 7 приводятся экспериментальные оценки производительности предложенного алгоритма. Наконец, в § 8 представлено обсуждение результатов, с выделением открытых вопросов и проблем, связанных с in-place алгоритмами, подобными $FHT2IDS$, которые требуют дальнейшего решения. Заключение представлено в § 9.

§ 2. Описание алгоритма $FHT2DS$ и его характеристики

Алгоритм $FHT2DS$ рекурсивно делит входное изображение

$$I = I_{w\times h}\colon \mathbb{Z}_w \times \mathbb{Z}_h \to \mathbb{A}$$

размера $w \times h$ на левое и правое подызображения I_L и I_R размеров $w_L \times h$ и $w_R \times h$ соответственно, где $w_L = \lfloor w/2 \rfloor$ и $w_R = w - \lfloor w/2 \rfloor$ [26–28]. Здесь $\mathbb{A}$ обозначает произвольную абелеву группу. На каждом шаге рекурсии вычисляется преобразование Хафа для подызображений I_L и I_R, – результаты вычислений обозначены J_L и J_R. Затем Хаф-образы J_L и J_R объединяются (сливаются) для формирования полного преобразования Хафа изображения I. Вектор $J(t,:)$ вычисляется как поэлементная сумма векторов $J_L(t_L,:)$ и $Concat(J_R(t_R, s:h), J_R(t_R, 0:s))$, где

$$s = (t - t_R) \bmod h, \quad t_L = [k_L t], \quad t_R = [k_R t],$$
$$k_L = (w_L - 1)/(w - 1), \quad k_R = (w_R - 1)/(w - 1).$$

Здесь для обращения к поддиапазонам векторов используется нотация среза, т.е. $n_1 : n_2$ обозначает диапазон от n_1 до n_2 (включая n_1, не включая n_2). Отсутствие обоих индексов указывает на полный диапазон. Функция $[\cdot]$ округляет вещественное число x до ближайшего целого числа $[x]$, при этом $[m + 1/2] = m$ для $m \in \mathbb{Z}$. В рамках алгоритмов, представленных в статье, Хаф-образ J использует так называемую st-параметризацию [26]: значение пикселя $J(t, s)$ равняется аппроксимации

интеграла входного одноканального изображения вдоль прямой

$$y(x) = s + \frac{t}{w-1}x, \quad (t,s) \in \mathbb{Z}_w \times \mathbb{Z}_h,$$

в декартовой системе координат Oxy, где оси Ox и Oy направлены вдоль ширины и высоты изображения соответственно [12].

Псевдокод алгоритма $FHT2DS$ представлен ниже [26–28]. В данной статье всякий алгоритм понимается как совокупность правил, которым необходимо следовать при выполнении вычислений, а псевдокод служит средством спецификации этого набора упорядоченных правил. Алгоритм однозначно определяет вычислительный граф, однако один и тот же вычислительный граф может соответствовать множеству различных алгоритмов, воплощающих одну и ту же идею, приводящих к одинаковому результату, но работающих с входными данными различными способами и следуя разным последовательностям правил. После того как алгоритм A представлен, его сложность по объему вспомогательной памяти, обозначенная через $M_A(n)$, где n – размер входных данных, определяется как минимально возможный размер массива, который должен быть выделен при выполнении алгоритма для получения корректного результата (в частности, результаты новых вычислений не должны перезаписывать массив, который все еще используется в вычислениях). При этом предполагается, что как только массив перестает использоваться в вычислениях (например, после завершения обработки рекурсивного вызова), он очищается, и память, необходимая для его хранения, освобождается. В данной статье свойства алгоритма в общем случае отличаются от свойств его конкретных реализаций, которые не рассматриваются. Так, например, при обсуждении сложности по объему вспомогательной памяти реализации, в отличие от сложности рекурсивного алгоритма, следует учитывать память, необходимую для хранения стека рекурсивных вызовов.

В алгоритме 12 функция $CreateZeroedImage(w,h)$ инициализирует изображение нулями. Функция $Concat(v_1,v_2)$ выполняет конкатенацию двух векторов v_1 и v_2.

Алгоритм 12 Алгоритм $FHT2DS$ для вычисления преобразования Хафа для изображений произвольного размера [26–28]

Input: $w > 0$, $h > 0$, изображение $I = I_{w\times h}$
Output: Хаф-образ $J = J_{w\times h}$
if $w > 1$ **then**
    $w_L \leftarrow \lfloor w/2 \rfloor$
    $w_R \leftarrow w - w_L$
    $I_L \leftarrow I(0 : w_L, :)$ ▷ I_L – область изображения I, память не выделяется
    $I_R \leftarrow I(w_L : w, :)$ ▷ I_R – область изображения I, память не выделяется
    $J_L \leftarrow FHT2DS(w_L, h, I_L)$ ▷ J_L является изображением
    $J_R \leftarrow FHT2DS(w_R, h, I_R)$ ▷ J_R является изображением
    $J \leftarrow CreateZeroedImage(w,h)$ ▷ для изображения J выделяется массив размера wh
    $k_L \leftarrow (w_L - 1)/(w-1)$
    $k_R \leftarrow (w_R - 1)/(w-1)$
    for $t \leftarrow 0$ **to** $w-1$ **do**
        $t_L \leftarrow [t\,k_L]$
        $t_R \leftarrow [t\,k_R]$
        $s \leftarrow (t - t_R) \bmod h$
        $J(t,:) \leftarrow J_L(t_L,:) + Concat(J_R(t_R, s:h), J_R(t_R, 0:s))$
else
    $J \leftarrow I$
return $J = J_{w\times h}$

Деля на каждом шаге рекурсии обрабатываемое изображение на левое и правое подызображения алгоритм $FHT2DS$ следует схеме "разделяй и властвуй" – это обеспечивает быстрое вычисление преобразования Хафа. На самом деле, было доказано, что алгоритм $FHT2DS$ демонстрирует следующую вычислительную сложность $T_{DS}(w,h)$ [27, 28], которая рассчитывается как количество выполненных операций в группе $\mathbb{A}$:

$$T_{DS}(w,h) = (\lfloor \log w \rfloor + 2)wh - 2^{\lfloor \log w \rfloor + 1}h \leqslant \frac{5\log_3 2}{3}\, wh\log w < 1{,}052\, wh\log w,$$

где $\dfrac{5\log_3 2}{3}$ – это точная константа в асимптотике вида $\mathrm{const}\cdot wh\log w$, т.е.

$$\sup_{w,h} \frac{T_{DS}(w,h)}{wh\log w} = \frac{5\log_3 2}{3},$$

$\log_2 x \equiv \log x$. Более того, имеет место асимптотическая эквивалентность [27, 28]:

$$T_{DS}(w,h) \sim wh\log w$$

при $w \to \infty$, т.е.

$$\lim_{w\to\infty} \frac{T_{DS}(w,h)}{wh\log w} = 1$$

для любой зависимости $h = h(w)$. Поскольку асимптотическая вычислительная сложность алгоритма Брейди–Ёна $T_{BY}(w,h) = wh\log w$, при $w = 2^q$, $q \in \mathbb{N}$, является оптимальной и не может быть улучшена [39], алгоритм $FHT2DS$ представляет собой обобщение алгоритма Брейди–Ёна для изображений с шириной, не являющейся степенью двойки, и это обобщение невозможно модифицировать с точки зрения асимптотической вычислительной сложности.

На каждом шаге рекурсии алгоритм $FHT2DS$ требует аллоцирования массива размера wh для вычисления Хаф-образа J с помощью J_L и J_R, вместо повторного использования массивов J_L и J_R. Иными словами, при каждом рекурсивном вызове алгоритм $FHT2DS$ требует выделения массива размера, равного размеру Хаф-образа, который вычисляется в данном вызове. Это приводит к тому, что сложность по вспомогательной памяти $M_{DS}(w,h)$ алгоритма $FHT2DS$ при обработке изображения размера $w \times h$ составляет $M_{DS}(w,h) = \mathcal{O}(wh\log w)$. Данный факт представляет собой препятствие для вычисления преобразования Хафа с использованием алгоритма $FHT2DS$ на устройствах с ограниченной памятью.

Наша цель – разработать in-place модификацию алгоритма $FHT2DS$ под названием $FHT2IDS$, которая, сохраняя рекурсивную схему алгоритма $FHT2DS$, требует выделения массива размера не более $w + h$ на каждом шаге рекурсии и при этом сохраняет вычислительную сложность $T_{IDS}(w,h) = T_{DS}(w,h)$ алгоритма $FHT2DS$.

§ 3. Вычислительный граф алгоритма $FHT2DS$

На каждом шаге рекурсии мы ассоциируем процедуру суммирования вектор-столбцов двух отдельных Хаф-образов, описанную в строках 13–17 алгоритма 12 и условно именуемую merge-этапом (merge-операцией или merge-процедурой в рамках одного рекурсивного вызова) алгоритма $FHT2DS$, с диаграммой потока данных (вычислительным графом), которая представлена ориентированным ациклическим графом

$$\Gamma = (V_\Gamma, E_\Gamma).$$

Множество вершин

$$V_\Gamma = \{L(i)\}_{i\in\mathbb{Z}_{w_L}} \cup \{R(i)\}_{i\in\mathbb{Z}_{w_R}} \cup \{C(i)\}_{i\in\mathbb{Z}_w}$$

состоит из вершин $L(i)$ (*левая* вершина) и $R(i)$ (*правая* вершина), хранящих векторы $J_L(i,:)$ и $J_R(i,:)$, а также вершин $C(t)$, в которых сохраняется объединенный вектор, полученный из $J_L(t_L,:)$ и $J_R(t_R,:)$,

$$t_L = [k_L t], \quad t_R = [k_R t], \quad k_L = (w_L - 1)/(w-1), \quad k_R = (w_R - 1)/(w-1),$$
$$w_L = \lfloor w/2 \rfloor, \quad w_R = w - \lfloor w/2 \rfloor.$$

Операции сложения векторов-столбцов производятся согласно направлениям ребер

$$E_\Gamma = \{(L(t_L), C(t))\}_{t\in\mathbb{Z}_w} \cup \{(R(t_R), C(t))\}_{t\in\mathbb{Z}_w}, \tag{1}$$
$$t_L = t_L(t) = [k_L t], \quad t_R = t_R(t) = [k_R t], \tag{2}$$
$$k_L = (w_L - 1)/(w-1), \quad k_R = (w_R - 1)/(w-1), \tag{3}$$
$$w_L = \lfloor w/2 \rfloor, \quad w_R = w - \lfloor w/2 \rfloor. \tag{4}$$

Отметим, что Γ является двудольным графом, поскольку все ребра $e \in E_\Gamma$ начинаются в $\{L(i)\}_{i\in\mathbb{Z}_{w_L}} \cup \{R(i)\}_{i\in\mathbb{Z}_{w_R}}$ и заканчиваются в $\{C(i)\}_{i\in\mathbb{Z}_w}$. Примеры описанного графа Γ, который управляет порядком исполнения merge-процедур в алгоритме $FHT2DS$, приведены на рис. 1.

Уже можно отметить, что вычислительный граф алгоритма $FHT2DS$ для четных значений ширины w напоминает вычислительный граф алгоритма Кули – Тьюки для БПФ по основанию 2 с прореживанием по времени (radix-2 decimation-in-time fast Fourier transform FFT, или radix-2 DIT FFT) [37], который используется для быстрого вычисления одномерного преобразования Фурье. Вычислительный граф алгоритма $FHT2DS$ для четных w представляет собой объединение непересекающихся подграфов, называемых бабочками, что аналогично структуре вычислительного графа для БПФ с основанием 2. Далее в § 6 будет дано точное определение графа типа бабочки, а также объяснение того, что вычислительный граф алгоритма $FHT2DS$ для четных значений w изоморфен вычислительному графу алгоритма Кули – Тьюки для БПФ с основанием 2. Это объяснение основывается на теоремах, представленных в § 5.

§ 4. Графовая интерпретация in-place свойства для алгоритма $FHT2DS$

Алгоритм $FHT2DS$ на каждой итерации предварительно выделяет память для массива J размера $w \times h$, в котором будут храниться суммы векторов $J_L(t_L,:)$ и $J_R(t_R,:)$ после исполнения merge-процедуры. Вместо предварительного выделения массива J мы можем хранить суммы векторов непосредственно в уже аллоцированных вершинах $\{L(i)\}_{i\in\mathbb{Z}_{w_L}} \cup \{R(i)\}_{i\in\mathbb{Z}_{w_R}}$, которые далее уже не будут задействованы при вычислении сумм остальных векторов. Такой подход потребует меньших затрат дополнительной рабочей памяти при условии, что нам известен порядок векторов, хранимых в $\{L(i)\}_{i\in\mathbb{Z}_{w_L}} \cup \{R(i)\}_{i\in\mathbb{Z}_{w_R}}$, а также порядок вершин $C(i)$, в которые последовательно будем записывать результаты суммирования векторов. Правильный порядок вычисления вершин $C(i)$ является принципиально важным для разработки модификации in-place алгоритма $FHT2DS$.

Математически задачу правильного обхода вершин $C(t)$ в in-place алгоритме можно сформулировать следующим образом. Рассмотрим биекцию

$$K\colon \{C(i)\}_{i\in\mathbb{Z}_w} \to \{L(i)\}_{i\in\mathbb{Z}_{w_L}} \cup \{R(i)\}_{i\in\mathbb{Z}_{w_R}}$$

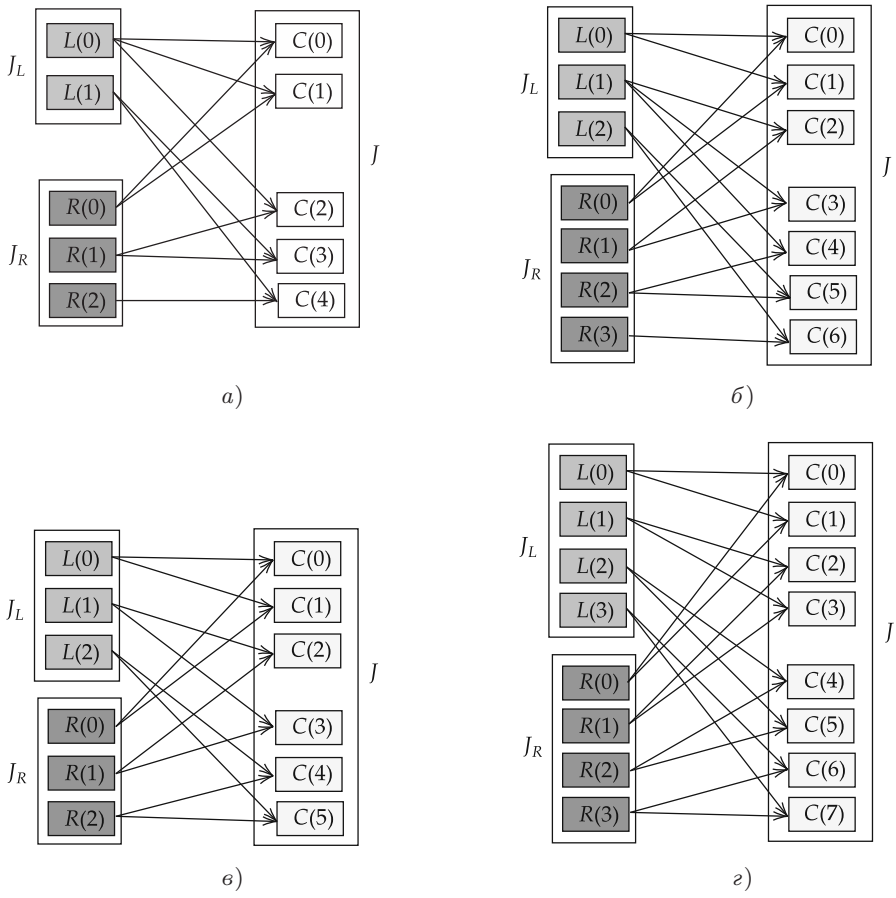


Рис. 1. Примеры вычислительных графов Γ алгоритма *FHT2DS* для различных значений ширины входного изображения w: *а*) $w = 5$, *б*) $w = 6$, *в*) $w = 7$, *г*) $w = 8$

и граф

$$\Gamma/K = (V_{\Gamma/K}, E_{\Gamma/K}),$$

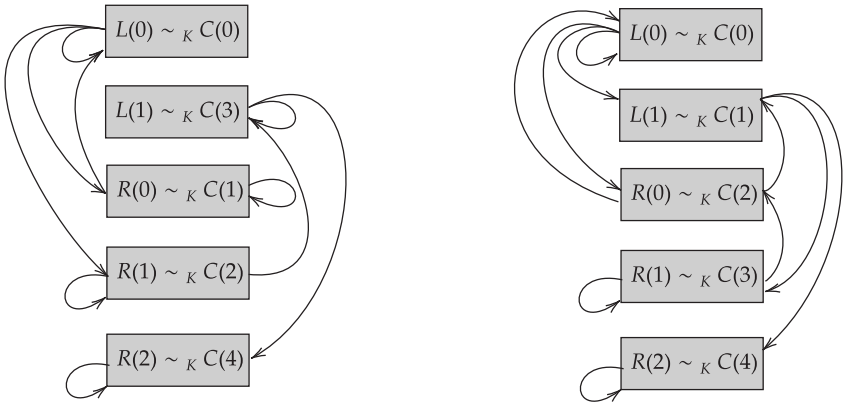
факторизованный следующим отношением эквивалентности вершин графа Γ

$$\forall v_1, v_2 \in V_\Gamma : \; v_1 \sim_K v_2 \iff v_1 = K(v_2) \vee v_2 = K(v_1).$$

Эквивалентно, мы получаем вершины $V_{\Gamma/K}$ после склеивания вершин V_Γ согласно отображению K, а ребра $E_{\Gamma/K}$ наследуются от ребер E_Γ:

$$V_{\Gamma/K} = \{v \in V_\Gamma \mid \forall v_1, v_2 \in V : \; v_1 \sim_K v_2 \Rightarrow v_1 = v_2\},$$
$$E_{\Gamma/K} = \{e \in E_\Gamma \mid \forall v_1', v_2', v_1'', v_2'' \in V_\Gamma : \; v_1' \sim_K v_1'', \; v_2' \sim_K v_2'' \; \Rightarrow \; (v_1', v_2') = (v_1'', v_2'')\}.$$

Пусть $\pi_\alpha \colon \mathbb{Z}_w \to 2^{\mathbb{Z}_w}$ – разбиение (ранга α) множества $\{0, 1, \ldots, w-1\}$ на подмножества, каждое из которых содержит не более $\alpha \in \mathbb{Z}_w$, $\alpha > 0$, элементов: для любого $i \in \mathbb{Z}_w$ выполняется $0 \leqslant |\pi_\alpha(i)| \leqslant \alpha$, $\bigcup \operatorname{Im} \pi_\alpha = \{0, 1, \ldots, w-1\}$ и $\pi_\alpha(i) \cap \pi_\alpha(j) = \varnothing$ для любых $i \neq j$.



а) *б*)

Рис. 2. Примеры вычислительных *FHT2DS* фактор-графов Γ/K для различных биекций $K \in \{K_{(a)}, K_{(b)}\}$ и соответствующих эквивалентностей $\sim_K$ на множестве вершин V_Γ, $w = 5$: *а*) Граф Γ с биекцией $K_{(a)}$, обладающий in-place свойством ранга 2: соответствующая функция разбиения задана как $\pi_2(0) = \{4\}$, $\pi_2(1) = \{3\}$, $\pi_2(2) = \{2\}$, $\pi_2(3) = \{0, 3\}$, $\pi_2(4) = \varnothing$. Пара $(K_{(a)}, \pi_2)$ обеспечивает in-place свойство ранга 2 для графа Γ; *б*) Пример графа Γ, который не обладает in-place свойством ранга 2 с биективной функцией $K_{(b)}$, т.е. не существует разбиения π_2 ранга 2, такого что пара $(K_{(b)}, \pi_2)$ обеспечивает in-place свойство ранга 2 для графа Γ. Заметим, что пара $(K_{(b)}, \pi_4)$ с $\pi_4(0) = \{4\}$, $\pi_4(1) = \{0, 1, 2, 3\}$, $\pi_4(2) = \pi_4(3) = \pi_4(4) = \varnothing$ гарантирует in-place свойство ранга 4 для графа Γ

Мы ищем пару (K, π_α), обеспечивающую следующее *in-place свойство* (*ранга* α) для графа Γ: для графа Γ/K и для любого $m \in \mathbb{Z}_w$, после удаления вершин

$$C(i) \in V_{\Gamma/K}, \quad i \in \bigcup \{\pi_\alpha(j) \mid 0 \leqslant j \leqslant m\},$$

и ребер из $E_{\Gamma/K}$, заканчивающихся в этих вершинах, результирующее множество оставшихся вершин Γ/K остается подграфом исходного графа Γ/K. В этом случае граф Γ, мы будем говорить, *обладает in-place свойством* (*ранга* α), *обеспеченным* (*гарантированным*) *парой* (K, π_α) (см. примеры на рис. 2). Когда индекс α не указан, его значение подразумевается равным 2.

Практически in-place свойство ранга α ($\alpha > 1$) для графа Γ означает, что алгоритм с вычислительным графом Γ может быть выполнен в in-place режиме, требующем выделения массива размера не более $(\alpha - 1)h$ для проведения промежуточных вычислений:

1. Последовательно для $0 \leqslant i \leqslant w - 1$ рассматриваются вершины $K(\mathrm{prev}(C(j))) \subset \subset V_\Gamma$, где $j \in \pi_\alpha(i)$, которые составляют образ множества $\mathrm{prev}(C(j))$ при отображении K;
2. Если $\pi_\alpha(i) \neq \varnothing$, выполняется суммирование векторов (merge-процедура), хранящихся в вершинах $K(\mathrm{prev}(C(j)))$ (при этом $|\mathrm{prev}(C(j))| = 2$ согласно уравнению (1));
3. После суммирования (merge-процедуры) векторов производится сохранение результата в вершинах $K(C(j)) \in V_\Gamma$, $j \in \pi_\alpha(i)$; поскольку $|\mathrm{prev}(C(j))| \leqslant \alpha$, эти вычисления могут быть выполнены не более чем с $\alpha - 1$ дополнительными временными массивами длины h.

Здесь, а также далее для произвольного ориентированного графа $\gamma = (V_\gamma, E_\gamma)$, если $C \in V_\gamma$, то

$$\mathrm{prev}(C) = \{v \in V_\gamma \mid (v, C) \in E_\gamma\}.$$

Также

$$\mathrm{next}(C) = \{v \in V_\gamma \mid (C, v) \in E_\gamma\}.$$

Таким образом, для построения in-place модификации алгоритма $FHT2DS$ необходимо найти соответствующую биективную функцию K и соответствующую функцию разбиения π_α. Описание таких функций требует более детального рассмотрения структуры вычислительного графа $FHT2DS$, что и является основным вопросом следующего параграфа.

§ 5. Анализ структуры вычислительного графа алгоритма $FHT2DS$

Здесь мы доказываем несколько замечательных свойств вычислительного графа алгоритма $FHT2DS$ (см. § 3). Эти свойства служат основой для обоснования корректности далее предложенного in-place алгоритма $FHT2IDS$.

Теорема 1. *Для любого* $t_L \in \mathbb{Z}_{w_L}$

$$\mathrm{next}(L(t_L)) = \left\{ C(t) \;\middle|\; \left\lfloor \frac{t_L - 1/2}{k_L} + 1 \right\rfloor \leqslant t \leqslant \left\lfloor \frac{t_L + 1/2}{k_L} \right\rfloor \wedge t \in \mathbb{Z}_w \right\}, \tag{5}$$

где $k_L = (w_L - 1)/(w - 1)$, $w_L = \lfloor w/2 \rfloor$, $w > 3$.

Также для любого $t_R \in \mathbb{Z}_{w_R}$

$$\mathrm{next}(R(t_R)) = \left\{ C(t) \;\middle|\; \left\lfloor \frac{t_R - 1/2}{k_R} + 1 \right\rfloor \leqslant t \leqslant \left\lfloor \frac{t_R + 1/2}{k_R} \right\rfloor \wedge t \in \mathbb{Z}_w \right\}, \tag{6}$$

где $k_R = (w_R - 1)/(w - 1)$, $w_R = w - \lfloor w/2 \rfloor$, $w > 2$.

Доказательство. Если $C(t) \in \mathrm{next}(L(t_L))$ для некоторого $t \in \mathbb{Z}_w$, $t_L \in \mathbb{Z}_{w_L}$, можем записать

$$t_L = [k_L t]$$

согласно устройству алгоритма $FHT2DS$ (см. § 2). Отсюда следует, что

$$t_L - 1/2 < k_L t \leqslant t_L + 1/2 \iff \frac{t_L - 1/2}{k_L} < t \leqslant \frac{t_L + 1/2}{k_L},$$

$k_L > 0$ при $w > 3$. Так как $t \in \mathbb{Z}_w$ – целое число, мы заключаем, что

$$\left\lfloor \frac{t_L - 1/2}{k_L} + 1 \right\rfloor \leqslant t \leqslant \left\lfloor \frac{t_L + 1/2}{k_L} \right\rfloor,$$

что и доказывает равенство (5).

В силу соотношения $t_R = [k_R t]$ равенство (6) доказывается аналогично. ▲

Далее мы будем обозначать

$$\deg^+(v) = |\{e \in E_\Gamma \mid e = (v, x) \wedge x \in V_\Gamma\}|$$

– *выходная степень* (*или степень исхода*) вершины $v \in V_\Gamma$ в ориентированном графе Γ,

$$\deg^-(v) = |\{e \in E_\Gamma \mid e = (x, v) \wedge x \in V_\Gamma\}|$$

будет обозначать *входную степень* (*или степень захода*) вершины $v \in V_\Gamma$ в ориентированном графе Γ, а

$$\deg(v) = \deg^+(v) + \deg^-(v)$$

– (*полная*) *степень* вершины $v \in V_\Gamma$.

Теорема 2. *Для всякого $t_L \in \mathbb{Z}_{w_L}$ справедливо неравенство*

$$2 \leqslant \deg^+(L(t_L)) \leqslant 3. \tag{7}$$

Более того, для всякого $t_R \in \mathbb{Z}_{w_R}$ имеет место неравенство

$$1 \leqslant \deg^+(R(t_R)) \leqslant 2. \tag{8}$$

Отметим, что для всякого $t \in \mathbb{Z}_w$

$$\deg^-(C(t)) = 2. \tag{9}$$

Доказательство. Поскольку вычислительный граф алгоритма *FHT2DS* не содержит кратных ребер, для доказательства неравенства (7) достаточно показать, что для всех $t_L \in \mathbb{Z}_{w_L}$ выполняется следующее неравенство:

$$2 \leqslant |\mathrm{next}(L(t_L))| \leqslant 3.$$

Для неравенства (8) необходимо показать, что, другими словами,

$$1 \leqslant |\mathrm{next}(R(t_R))| \leqslant 2.$$

Рассмотрим отношение $t_L = [k_L t]$, которое означает $C(t) \in \mathrm{next}(L(t_L))$. Заметим, что

$$k_L = \frac{\lfloor w/2 \rfloor - 1}{w - 1} < 1/2$$

при $w > 3$. Это означает, что любое множество вида $(k-1/2, k+1/2]$, $k \in \mathbb{Z}$, содержит хотя бы два различных элемента из последовательности $\{k_L n\}_{n \in \mathbb{Z}_w}$. Следовательно, $|\mathrm{next}(L(t_L))| \geqslant 2$ для $w > 3$. Построив графы Γ для случаев $w \in \{1, 2, 3\}$, можно убедиться, что последнее неравенство справедливо для $w \geqslant 1$.

Аналогично, рассматривая соотношение $t_R = [k_R t]$, оценка

$$k_R = \frac{w - \lfloor w/2 \rfloor - 1}{w - 1} \leqslant \frac{1}{2}$$

при $w > 3$ обосновывает тот факт, что любое множество вида $(k - 1/2, k + 1/2]$, $k \in \mathbb{Z}$, содержит хотя бы один элемент из последовательности $\{k_R n\}_{n \in \mathbb{Z}_w}$. Следовательно, $|\mathrm{next}(R(t_R))| \geqslant 1$ для $w > 3$. Построив графы Γ для случаев $w \in \{1, 2, 3\}$, можно убедиться, что неравенство $|\mathrm{next}(R(t_R))| \geqslant 1$ остается верным также для $w \in \{1, 2, 3\}$.

Далее зафиксируем $t_L \in \mathbb{Z}_{w_L}$, и пусть $t = t' \in \mathbb{Z}_{w-3}$ удовлетворяет соотношению $t_L = [k_L t]$. Мы докажем, что $t = t' + 3$ не может быть решением уравнения $t_L = [k_L t]$. Это привело бы к неравенству $|\mathrm{next}(L(t_L))| \leqslant 3$ (здесь используется монотонность функции $t_L(t) = [k_L t]$, т.е. $t'' > t' + 3$ не может быть решением уравнения $t_L = [k_L t]$, если $t' + 3$ не является его решением). Действительно, из $t_L = [k_L t']$ получаем

$$k_L t' > t_L - 1/2.$$

Далее, для $t = t' + 3$ имеем

$$k_L(t' + 3) > t_L + 3k_L - 1/2. \tag{10}$$

Заметим, что

$$t_L + 3k_L - 1/2 \geqslant t_L + 1/2 \iff k_L = \frac{\lfloor w/2 \rfloor - 1}{w-1} \geqslant 1/3 \tag{11}$$

при $w \geqslant 10$. Из уравнений (10) и (11) получаем

$$k_L(t'+3) > t_L + 1/2,$$

что означает

$$[k_L(t'+3)] \geqslant t_L + 1,$$

т.е. $t = t' + 3$ не является решением уравнения $t_L = [k_L t]$, и $|\mathrm{next}(L(t_L))| \leqslant 3$ при $w \geqslant 10$. Это неравенство остается верным и для $w < 10$, что подтверждается непосредственным построением графов Γ для $w < 10$.

Далее мы докажем, что $|\mathrm{next}(R(t_R))| \leqslant 2$ при $t_R \in \mathbb{Z}_{w_R}$.

Отметим, что выполняются следующие неравенства:

$$\frac{t}{2} - \frac{1}{2} < k_R t \leqslant \frac{t}{2}, \quad t < w-1, \tag{12}$$

и

$$\frac{t}{2} - \frac{1}{2} \leqslant k_R t \leqslant \frac{t}{2}, \quad t = w-1. \tag{13}$$

Действительно, оценки сверху в неравенствах (12) и (13) верны, коль скоро $k_R \leqslant \frac{1}{2}$.

Теперь поясним оценки снизу в неравенствах (12) и (13). Когда $w = 1 \bmod 2$, имеем $k_R = \frac{1}{2}$, и следовательно,

$$k_R t = \frac{t}{2} > \frac{t}{2} - \frac{1}{2}.$$

Когда $w = 0 \bmod 2$, $k_R < \frac{1}{2}$, имеем следующую цепочку неравенств для $t < w-1$:

$$\frac{t}{2} - k_R t < \frac{w-1}{2} - k_R(w-1) = \frac{w-1}{2} - (w_R - 1) = \frac{w-1}{2} - w_R + 1 =$$
$$= \frac{w}{2} - w_R + \frac{1}{2} = \frac{1}{2},$$

так как функция $f(t) = \frac{t}{2} - k_R t$ является возрастающей, а $w_R = \frac{w}{2}$.

Когда $w = 0 \bmod 2$, для $t = w-1$ имеем

$$\frac{w-1}{2} - \frac{1}{2} = \frac{w}{2} - 1 = w_R - 1 = k_R(w-1).$$

Таким образом, неравенства (12) и (13) доказаны. Из них следует, что каждый полуинтервал вида $(k/2 - 1/2, k/2]$, $k \in \mathbb{Z}_{w-1}$, содержит ровно одно число вида $k_R t$, а полуинтервал $((w-1)/2 - 1/2, (w-1)/2]$ содержит либо два числа вида $k_R t$ (когда $k_R < 1/2$ и $w = 0 \bmod 2$), либо одно число вида $k_R t$ (когда $k_R = 1/2$ и $w = 1 \bmod 2$). Поэтому в каждом множестве вида $(t_R - 1/2, t_R + 1/2]$, $t_R \in \mathbb{Z}_{w_R}$, содержится не более двух различных чисел вида $k_R t$, и следовательно, уравнение $t_R = [k_R t]$, $t_R \in \mathbb{Z}_{w_R}$, не имеет более двух различных решений, а значит, $|\mathrm{next}(R(t_R))| \leqslant 2$.

Таким образом, неравенства (7) и (8) доказаны.

Наконец, уравнение (1) объясняет соотношение (9). Теорема 2 полностью доказана. ▲

Следующие следствие и теорема характеризуют количество левых вершин $L(t_L)$ с выходной степенью 3 и правых вершин $R(t_R)$ с выходной степенью 1.

Следствие 1. *Число левых вершин с выходной степенью* 3 *равно числу правых вершин с выходной степенью* 1:

$$|\{L(t_L) \mid \deg^+(L(t_L)) = 3 \wedge t_L \in \mathbb{Z}_{w_L}\}| = \\ = |\{R(t_R) \mid \deg^+(R(t_R)) = 1 \wedge t_R \in \mathbb{Z}_{w_R}\}|. \tag{14}$$

Доказательство. Для графа Γ выполняется следующее стандартное тождество:

$$\sum_{v \in V_\Gamma} \deg(v) = 2|E_\Gamma|, \tag{15}$$

т.е. сумма степеней всех вершин графа равна удвоенному числу ребер.

Уравнение (15) можно переписать в следующем виде:

$$\sum_{t_L \in \mathbb{Z}_{w_L}} \deg^+(L(t_L)) + \sum_{t_R \in \mathbb{Z}_{w_R}} \deg^+(R(t_R)) + \sum_{t \in \mathbb{Z}_w} \deg^-(C(t)) = 2|E_\Gamma| \iff \tag{16}$$

$$\iff \sum_{t_L \in \mathbb{Z}_{w_L}} \deg^+(L(t_L)) + \sum_{t_R \in \mathbb{Z}_{w_R}} \deg^+(R(t_R)) + 2w = 4w \iff \tag{17}$$

$$\iff \sum_{t_L \in \mathbb{Z}_{w_L}} \deg^+(L(t_L)) + \sum_{t_R \in \mathbb{Z}_{w_R}} \deg^+(R(t_R)) = 2w. \tag{18}$$

Здесь мы воспользовались соотношением (9), а также тем фактом, что согласно архитектуре алгоритма *FHT2DS* справедливы следующие равенства для входных и выходных степеней вершин:

$$\deg^-(L(t_L)) = \deg^-(R(t_R)) = \deg^+(C(t)) = 0.$$

Кроме того,

$$|E_\Gamma| = 2|\{C(t) \mid t \in \mathbb{Z}_w\}| = 2w.$$

В силу теоремы 2 и уравнения (18), если $\deg^+(L(t_L)) = 3$ для некоторого $t_L \in \mathbb{Z}_{w_L}$, то по принципу Дирихле существует $t_R \in \mathbb{Z}_{w_R}$, для которого $\deg^+(R(t_R)) = 1$. И обратное утверждение также верно: как только $\deg^+(R(t_R)) = 1$ для некоторого $t_R \in \mathbb{Z}_{w_R}$, то по принципу Дирихле существует $t_L \in \mathbb{Z}_{w_L}$, для которого $\deg^+(L(t_L)) = 3$. Таким образом, тождество (14) доказано. ▲

Теорема 3. *Справедливы следующие утверждения*:

1. *Пусть* $w = 0 \bmod 2$. *Тогда все левые и правые вершины имеют выходную степень, равную* 2:

$$\deg^+(L(t_L)) = \deg^+(R(t_R)) = 2, \quad t_L \in \mathbb{Z}_{w_L}, \quad t_R \in \mathbb{Z}_{w_R}, \quad w = 0 \bmod 2. \tag{19}$$

2. *Пусть* $w = 1 \bmod 2$. *Тогда существует единственная левая вершина* $L([w/4]-1)$ *с выходной степенью* 3 *и единственная правая вершина* $R(w_R - 1)$ *с выходной степенью* 1. *Все остальные левые и правые вершины имеют выходную степень, равную* 2:

$$\deg^+(L([w/4] - 1)) = 3, \quad \deg^+(L(t_L)) = 2, \quad t_L \in \mathbb{Z}_{w_L} \setminus \{[w/4] - 1\}, \tag{20}$$

$$\deg^+(R(w_R - 1)) = 1, \quad \deg^+(R(t_R)) = 2, \quad t_R \in \mathbb{Z}_{w_R} \setminus \{w_R - 1\}, \tag{21}$$

$$w_R = w - \lfloor w/2 \rfloor, \quad w = 1 \bmod 2. \tag{22}$$

Доказательство. Во-первых, пусть $w = 0 \bmod 2$. Из неравенств (12) и (13) следует, что каждый полуинтервал $(k/2 - 1/2, k/2]$, $k \in \mathbb{Z}_{w-1}$, содержит одно число вида $k_R t$, в то время как полуинтервал $((w-1)/2 - 1/2, (w-1)/2]$ содержит ровно два числа вида $k_R t$. Следовательно, каждое множество вида $(t_R - 1/2, t_R + 1/2]$, $t_R \in \mathbb{Z}_{w_R}$, содержит ровно два числа из последовательности $\{k_R t\}_{t \in \mathbb{Z}_w}$, т.е. для любого $t_R \in \mathbb{Z}_{w_R}$ уравнение $t_R = [k_R t]$ имеет ровно два решения. Это эквивалентно тому, что $\deg^+(R(t_R)) = 2$ для всех $t_R \in \mathbb{Z}_{w_R}$, что, согласно следствию 1, также влечет $\deg^+(L(t_L)) = 2$ для всех $t_L \in \mathbb{Z}_{w_L}$. Первая часть текущей теоремы доказана.

Теперь положим $w = 1 \bmod 2$. В этом случае $k_R = 1/2$, и из неравенств (12) и (13) следует, что каждый полуинтервал $(k/2 - 1/2, k/2]$, $k \in \mathbb{Z}_w$, содержит ровно одно число вида $k_R t$, $t \in \mathbb{Z}_w$. Следовательно, полуинтервалы $(t_R - 1/2, t_R + 1/2]$, $t_R \in \mathbb{Z}_{w_R - 1}$, содержат два числа вида $k_R t$, а последний полуинтервал $(t_R - 1/2, t_R + 1/2]$, $t_R = w_R - 1$, содержит только одно такое число $w_R - 1$; $t \in \mathbb{Z}_w$. Другими словами, уравнение $t_R = [k_R t]$ имеет два решения, когда $t_R \neq w_R - 1$. Это доказывает, что существует только одна правая вершина $R(w_R - 1)$ графа Γ с выходной степенью 1. Согласно следствию 1 существует также единственная левая вершина с выходной степенью 3. Необходимо доказать, что ее порядковый номер равен $[w/4] - 1$.

Эквивалентно, проверим, что уравнение $[w/4] - 1 = [k_L t]$ имеет три различных решения для $t \in \mathbb{Z}_w$. Действительно, эти решения выражаются как

$$t \in \{2([w/4] - 1) + n \mid n = 0, 1, 2\}.$$

Если $w = 4k+1$, $k \in \mathbb{Z}$, $k > 0$, имеем $[w/4]-1 = k-1$, $w_L = \lfloor w/2 \rfloor = 2k$, $k_L = \dfrac{2k-1}{4k}$ и

$$k - \frac{3}{2} < \frac{(2k-1)(2k-2)}{4k} \leqslant k_L t = \frac{2k-1}{4k}(2k + n - 2) \leqslant k - \frac{1}{2} \Longrightarrow$$
$$\Longrightarrow [k_L t] = k - 1 = [w/4] - 1.$$

Если же $w = 4k - 1, k \in \mathbb{Z}$, $k > 0$, тогда $[w/4] - 1 = k - 1$, $w_L = \lfloor w/2 \rfloor = 2k - 1$, $k_L = \dfrac{k-1}{2k-1}$ и

$$k - \frac{3}{2} < \frac{(k-1)(2k-2)}{2k-1} \leqslant k_L t = \frac{k-1}{2k-1}(2k + n - 2) \leqslant \frac{2k(k-1)}{2k-1} \leqslant k - \frac{1}{2} \Longrightarrow$$
$$\Longrightarrow [k_L t] = k - 1 = [w/4] - 1.$$

Таким образом, $t \in \{2([w/4]-1)+n \mid n = 0, 1, 2\}$ действительно являются решениями уравнения $t_L = [k_L t]$, и можем заключить, что $\deg^+(L([w/4] - 1)) = 3$. Вторая часть теоремы 3 полностью доказана. ▲

Утверждение теоремы согласуется с рис. 1, на котором изображены вычислительные графы Γ алгоритма $FHT2DS$ для $w \in \{5, 6, 7, 8\}$.

В следующем параграфе, основываясь на доказанных структурных свойствах вычислительного графа алгоритма $FHT2DS$, мы представим его in-place модификацию – алгоритм $FHT2IDS$. Обоснование нового алгоритма также приведено в следующем параграфе.

§ 6. Описание in-place алгоритма *FHT2IDS*

Мы приводим описание нашего алгоритма $FHT2IDS$ – in-place модификации алгоритма $FHT2DS$, который был предложен ранее [26–28] и предназначен для вычисления БПХ для изображений произвольной ширины.

В алгоритме $FHT2IDS$ вершины $C(t)$ вычисляются в ходе выполнения последовательных merge-процедур, т.е. суммирования векторов, хранимых в левых $L(t_L)$ и правых $R(t_R)$ вершинах (см. § 3). Однако в отличие от алгоритма $FHT2DS$ алгоритм $FHT2IDS$ сохраняет результат либо в левой, либо в правой вершине, т.е. в той области памяти, где хранилось входное изображение I. При этом вектор, который ранее был сохранен в этой вершине, стирается и больше не может быть использован для дальнейших вычислений. Поэтому в алгоритме $FHT2IDS$ каждый результат вычислений должен быть сохранен в вершине, которая впоследствии не используется для вычислений других $C(t)$. Это требование накладывает существенные ограничения на порядок вычисления вершин $C(t)$. Важно отметить, что при таком обходе вершин столбцы результата преобразования Хафа могут вычисляться не по порядку (заданном st-параметризацией), в отличие от алгоритма $FHT2DS$. Однако даже при таком порядке обхода можно переставить столбцы выходного изображения так, чтобы оно совпало с результатом работы алгоритма $FHT2DS$.

Мы опишем правильный порядок обхода вычисляемых вершин $C(t)$, а также порядок сохранения результатов слияний (в ходе merge-процедур) векторов (т.е. столбцов Хаф-образов) из левых и правых вершин.

Рассмотрим случай четной ширины $w = 0 \bmod 2$. Согласно ключевой теореме 3 из предыдущего § 5 каждая левая вершина $L(t_L)$ и правая вершина $R(t_R)$ вычислительного графа имеют выходную степень, равную 2, в то время как входная степень каждой вершины $C(t)$ также равна 2. Кроме того, ребра, инцидентные левой/правой вершине с большим индексом t_L или t_R, входят в вершину $C(t)$ с большим индексом t. Таким образом, вычислительный граф алгоритма $FHT2DS$ в случае четной ширины w распадается на $w/2$ так называемых бабочек (изображенных на рис. 3). Более точно, бабочка здесь определяется как подграф $\mathcal{B}(k)$, $k \in \mathbb{Z}_{w/2}$, графа Γ и имеет следующий вид:

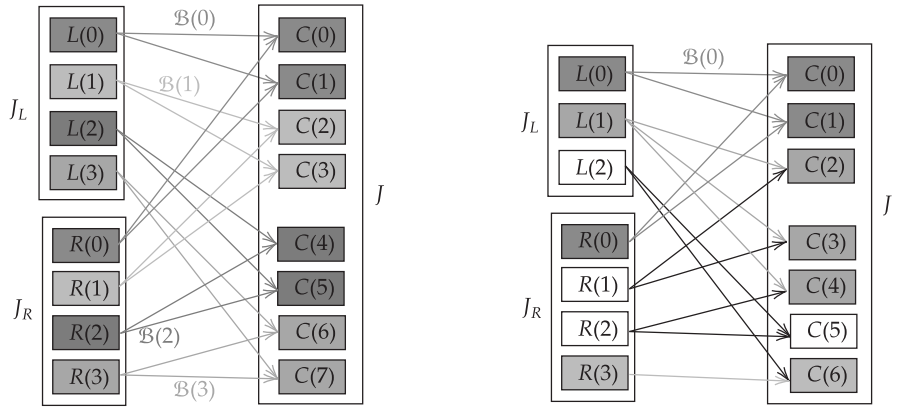
$$\mathcal{B}(k) = (V_{\mathcal{B}(k)}, E_{\mathcal{B}(k)}) \subset \Gamma,$$
$$V_{\mathcal{B}(k)} = \{L(k), R(k), C(2k), C(2k+1)\},$$
$$E_{\mathcal{B}(k)} = \{(L(k), C(i)), (R(k), C(i)) \mid i = 2k, 2k+1\}.$$

Для случая четной ширины w мы предлагаем обрабатывать все бабочки вычислительного графа Γ последовательно. При обработке отдельной бабочки $\mathcal{B}(k)$, $k \in \mathbb{Z}_{w/2}$, используется дополнительный временный массив размера $1 \times h$ (обозначаемый через tmp):

1. Сначала вычисляется вершина $C(k)$ (соответствующая некоторому значению параметра $t = 2k$ в алгоритме 13) по вершинам $L(k)$ и $R(k)$ (которые хранят вектора $J_L(k_L, :)$ и $J_R(k_R, :)$ в алгоритме 13 соответственно), и результат сохраняется во временном массиве tmp (см. алгоритм 13 для обработки структуры бабочки, строка 9);
2. Затем вычисляется вершина $C(2k+1)$ (соответствующая значению $t = 2k+1$) по вершинам $L(k)$ и $R(k)$ (которые содержат вектора $J_L(k_L, :)$ и $J_R(k_R, :)$ в алгоритме 13 соответственно), и результат сохраняется в $R(k)$ (эта вершина соответствует $J_R(k_R, :)$, см. алгоритм 13, строка 10);
3. Наконец, содержимое tmp записывается обратно в $L(k)$ (соответствующий вектор есть $J_L(k_L, :)$, алгоритм 13, строка 11).

Алгоритм, вычисляющий вершины в структуре отдельной бабочки, приведен в виде псевдокода 13.

Переформулировав алгоритм 13 в терминах, введенных в § 4 с графовой интерпретацией in-place свойства, можно заключить, что функция K, которая определяет левые и правые вершины для последовательной записи вычисляемых вершин $C(t)$, $t \in \mathbb{Z}_w$, отображает вершины $C(2k)$ в вершины $L(k)$, а вершины $C(2k+1)$ – в вер-



а) *б*)

Рис. 3. Пример структуры бабочек в вычислительном графе Γ алгоритма *FHT*2*DS*: *а*) $w = 8$; *б*) $w = 7$. Кроме бабочки $\mathcal{B}(0)$, также выделены вершины со степенями 3 и 1

Алгоритм 13 Алгоритм *ProcessButterfly*

Input: изображения J_L и J_R, их ширины w_L и w_R, они имеют равную высоту h, массивы индексов K_L и K_R, по которым восстанавливается правильный порядок столбцов изображений J_L и J_R (индуцированный *st*-параметризацией), индекс n бабочки $\mathcal{B}(n)$, которую необходимо вычислить, массив K, по которому восстанавливается правильный порядок столбцов изображения J (индуцированный *st*-параметризацией)
$t \leftarrow 2k$
$t_L \leftarrow k$
$t_R \leftarrow k$
$s_L \leftarrow (t - t_R) \bmod h$
$s_R \leftarrow (t + 1 - t_R) \bmod h$
$k_L \leftarrow K_L(t_L)$
$k_R \leftarrow K_R(t_R)$
$tmp \leftarrow J_L(k_L, :) + Concat(J_R(k_R, s_L : h), J_R(k_R, 0 : s_L))$ ▷ выделение массива размера h
$J_R(k_R, :) \leftarrow J_L(k_L, :) + Concat(J_R(k_R, s_R : h), J_R(k_R, 0 : s_R))$
$J_L(k_L, :) \leftarrow tmp$
$K(t) \leftarrow k_L$
$K(t + 1) \leftarrow w_R + k_R$

шины $R(k)$:

$$K(C(2k)) = L(k), \quad K(C(2k + 1)) = R(k), \quad k \in \mathbb{Z}_{w/2}. \tag{23}$$

Кроме того, разбиение π_2 ранга 2 множества $\mathbb{Z}_w$, которое определяет порядок перезаписи левых и правых вершин, для нашего алгоритма выражается следующей формулой:

$$\pi_2(k) = \{2k, 2k + 1\}, \quad k \in \mathbb{Z}_{w/2},$$
$$\pi_2(k) = \varnothing, \quad k \in \mathbb{Z}_w \setminus \mathbb{Z}_{w/2}.$$

Для наглядности порядок вычислений (merge-операций) в алгоритме *FHT*2*IDS* и порядок вершин, используемых для хранения результатов вычислений, определенный парой (K, π_2), приведены на рис. 4.

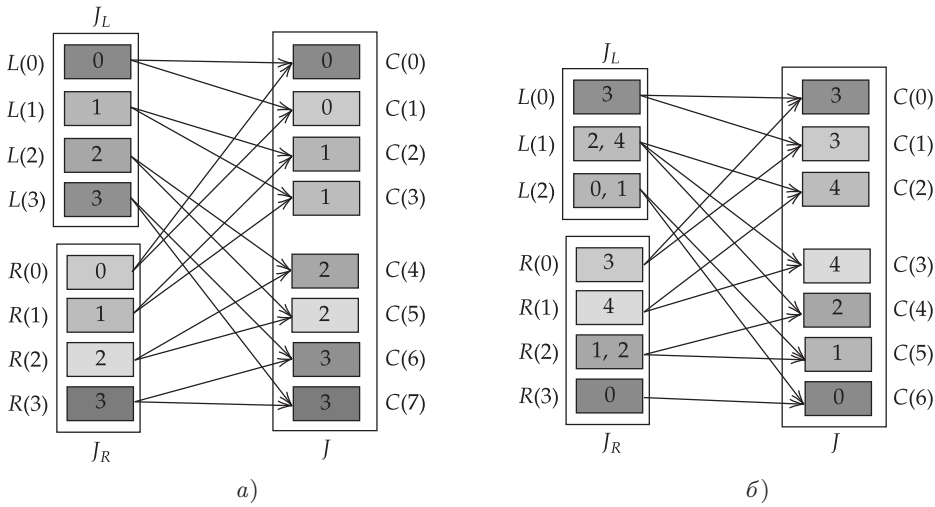


Рис. 4. Порядок merge-операций и сохранений в алгоритме $FHT2IDS$. Номера вершин указывают на этапы алгоритма, на которых они используются: *а*) $w = 8$. Биекция K задается следующим образом: $K(C(0)) = L(0)$, $K(C(1)) = R(0)$, $K(C(2)) = L(1)$, $K(C(3)) = R(1)$, $K(C(4)) = L(2)$, $K(C(5)) = R(2)$, $K(C(6)) = L(3)$, $K(C(7)) = R(3)$; *б*) $w = 7$. Биекция K задается следующим образом: $K(C(0)) = L(0)$, $K(C(1)) = R(0)$, $K(C(2)) = L(1)$, $K(C(3)) = R(1)$, $K(C(4)) = R(2)$, $K(C(5)) = L(2)$, $K(C(6)) = R(3)$

Теоремы из предыдущего § 5 доказывают, что такая пара (K, π_2) обеспечивает in-place свойство (ранга 2) для графа Γ: результаты вычислений алгоритмов $FHT2IDS$ и $FHT2DS$ для идентичных входных изображений, с учетом перестановки K их столбцов, совпадают. Каждый рекурсивный вызов (без обработки дочерних вызовов) выделяет массив размера не более чем $w' + h$: массив размера $w' \leqslant w$ для хранения массива для K и отдельный массив размера h для обработки всех бабочек.

Следует отметить, что тот факт, что граф Γ в случае четной ширины изображения является дизъюнктным объединением бабочек, делает наш алгоритм $FHT2IDS$ схожим с алгоритмом Кули – Тьюки для быстрого вычисления одномерного преобразования Фурье. Вычислительный граф алгоритма Кули – Тьюки для случая БПФ с декомпозицией по основанию 2 и с прореживанием по времени также представляется как объединение бабочек [37]. Таким образом, для случая четной ширины w вычислительный граф алгоритма $FHT2DS$ изоморфен вычислительному графу алгоритма Кули – Тьюки для одномерного БПФ с основанием 2 (в частности, степени всех вершин в соответствующих вычислительных графах алгоритмов совпадают).

Теперь опишем алгоритм $FHT2IDS$ для изображений с нечетной шириной $w = 1 \bmod 2$. Согласно ключевой теореме 3 из предыдущего параграфа, граф Γ имеет одну левую вершину $L(\lfloor w/4 \rfloor - 1)$ с выходной степенью 3 и одну правую вершину $R(w_R - 1)$ с выходной степенью 1, в то время как все остальные левые и правые вершины имеют выходную степень 2. Поскольку вершина $R(w_R - 1)$ участвует только в одном вычислении вершины $C(w - 1)$, мы предлагаем рассматривать ее первой и записывать результат merge-операции вершин $L(w_L - 1)$ и $R(w_R - 1)$ в нее. После этого мы мысленно удаляем вершины $R(w_R - 1)$ и $C(w - 1)$, а также ребра $(L(w_L - 1), C(w - 1))$ и $(R(w_R - 1), C(w - 1))$ из графа Γ (см. рис. 5, шаг 1).

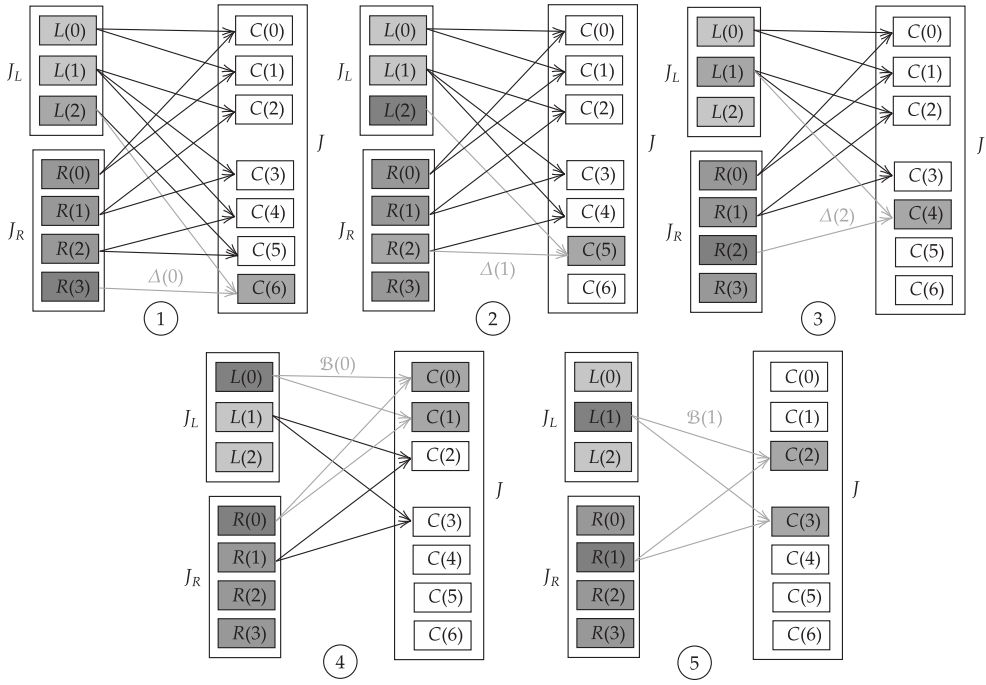


Рис. 5. Последовательные преобразования графа Γ в алгоритме *FHT2IDS* для случая $w = 7$. На каждом шаге рассматриваются вершины треугольника или бабочки. Результаты вычислений записываются в вершины, составляющие основание треугольника или бабочки

Процедура, выполняемая при обработке следующего подграфа типа треугольника

$$\Delta(0) = (V_{\Delta(0)}, E_{\Delta(0)}) \subset \Gamma,$$
$$V_{\Delta(0)} = \{L(w_L - 1), R(w_R - 1), C(w - 1)\},$$
$$E_{\Delta(0)} = \{(L(w_L - 1), C(w - 1)), (R(w_R - 1), C(w - 1))\},$$

будет называться *ProcessTriangle* (см. алгоритм 14). На этом конкретном шаге алгоритма, в ходе исполнения *ProcessTriangle* применительно к $\Delta(0)$, результат вычислений сохраняется в правой вершине $R(w_R - 1)$.

После этого, мысленно удалив ребра, входящие в вершину $C(w - 1)$, выходная степень вершины $L(w_L - 1)$ станет равной либо 1, либо 2. Если она станет равной 1, выполняется процедура *ProcessTriangle* для вершин треугольника

$$\Delta(1) = (V_{\Delta(1)}, E_{\Delta(1)}) \subset \Gamma,$$
$$V_{\Delta(1)} = \{L(w_L - 1), R(w_R - 2), C(w - 2)\},$$
$$E_{\Delta(1)} = \{(L(w_L - 1), C(w - 2)), (R(w_R - 2), C(w - 2))\},$$

и результат вычисления $C(w-2)$ записывается в левую вершину $L(w_L-1)$ (см. шаг 2 на рис. 5). После этого действия степень исхода вершины $R(w_R-2)$ станет равной 1, и мы можем повторить шаги, описанные выше, для правой вершины $R(w_R - 2)$ (см. шаг 3 на рис. 5).

Алгоритм 14 Алгоритм *ProcessTriangle*

Input: Изображения J_L и J_R, их ширины w_L и w_R, их одинаковая высота h, массивы индексов K_L и K_R, которые позволяют восстановить правильный порядок столбцов изображений J_L и J_R, индекс n треугольника $\Delta(n)$, подлежащего обработке, и массив K, который позволяет восстановить правильный порядок столбцов изображения J

$t \leftarrow w - 1 - n$

$t_L \leftarrow w_L - 1 - \lfloor n/2 \rfloor$

$t_R \leftarrow w_R - 1 - \lfloor (n+1)/2 \rfloor$

$s \leftarrow (t - t_R) \bmod h$

$k_L \leftarrow K_L(t_L)$

$k_R \leftarrow K_R(t_R)$

if $n = 0 \bmod 2$ **then** ▷ результат сохраняется в правую вершину

    $J_R(k_R, :) \leftarrow J_L(k_L, :) + Concat(J_R(k_R, s:h), J_R(k_R, 0:s))$

    $K(t) \leftarrow w_R + k_R$

else ▷ результат сохраняется в левую вершину

    $J_L(k_L, :) \leftarrow J_L(k_L, :) + Concat(J_R(k_R, s:h), J_R(k_R, 0:s))$

    $K(t) \leftarrow k_L$

Таким образом, снизу вверх по вычислительному графу обрабатывая последовательные треугольники, при этом чередуя процесс записи merge-результатов в левые и правые вершины (шаги 1–3), мы в конечном итоге дойдем до вершины $L([w/4]-1)$, которая будет иметь степень исхода, равную 2 (хотя изначально ее степень исхода была равна 3). На этом этапе степень исхода всех оставшихся (не удаленных из рассмотрения) левых и правых вершин будет равна 2, и следовательно, оставшийся граф раскладывается на бабочки (шаг 4 на рис. 5, также см. рис. 3,*б*)). В дальнейшем остается последовательно обработать полученные бабочки с помощью повторных применений процедуры *ProcessButterfly*, как это описано выше для изображений с четной шириной (см. алгоритм 13). Схематично алгоритм *FHT2IDS* проиллюстрирован на рис. 5.

Следуя алгоритму *FHT2IDS*, можно уточнить, что для нечетной ширины w граф Γ состоит из $[w/4]$ последовательных бабочек $\{\mathcal{B}(k)\}_{k\in\mathbb{Z}_{[w/4]}}$ и $w - 2[w/4]$ треугольников $\{\Delta(k)\}_{k\in\mathbb{Z}_{w-2[w/4]}}$. Треугольник $\Delta(k)$ с номером k задается следующим образом:

$$\Delta(k) = (V_{\Delta(k)}, E_{\Delta(k)}) \subset \Gamma, \quad k \in \mathbb{Z}_{w-2[w/4]},$$
$$V_{\Delta(k)} = \{L(w_L - 1 - \lfloor k/2 \rfloor), R(w_R - 1 - \lfloor (k+1)/2 \rfloor), C(w-1-k)\},$$
$$E_{\Delta(k)} = \{(v, C(w-1-k)) \mid v = L(w_L - 1 - \lfloor k/2 \rfloor) \vee$$
$$\vee\, v = R(w_R - 1 - \lfloor (k+1)/2 \rfloor)\}.$$

Важно отметить, что стадия обработки любого треугольника не требует выделения дополнительной памяти и выполняется исключительно в пределах существующей памяти. В то же время, для бабочек, как было отмечено при рассмотрении случая с четной шириной w, необходимо выделить только один массив размера h. Это приводит к тому, что $M_{IDS}(w,h) = h$.

Используя графовую интерпретацию из § 4, в терминах пары отображений (K, π_α) при $\alpha = 2$ мы предлагаем следующие отображения для случая нечетной ширины w:

$$K(C(w-1-k)) = R(w_R - 1 - \lfloor (k+1)/2 \rfloor), \quad k = 0 \bmod 2, \quad k \in \mathbb{Z}_{w-2[w/4]}, \quad (24)$$
$$K(C(w-1-k)) = L(w_L - 1 - \lfloor k/2 \rfloor), \quad k = 1 \bmod 2, \quad k \in \mathbb{Z}_{w-2[w/4]}, \quad (25)$$
$$K(C(2k)) = L(k), K(C(2k+1)) = R(k), \quad k \in \mathbb{Z}_{[w/4]}, \quad (26)$$

и

$$\pi_2(k) = \{w-1-k\}, \quad k \in \mathbb{Z}_{w-2[w/4]},$$
$$\pi_2(k+w-2[w/4]) = \{2k, 2k+1\}, \quad k \in \mathbb{Z}_{[w/4]},$$
$$\pi_2(k) = \varnothing, \quad k < w \wedge k \geqslant w - [w/4].$$

Для нечетного w структура отображения K наглядно проиллюстрирована на рис. 4,*б*). Образ целого числа $n \in \mathbb{Z}_w$ при отображении π_2 есть множество индексов t вершин $C(t)$, помеченных на рис. 4 числом n.

Согласно представленным рассуждениям, основанным на теоремах из § 5, пара (K, π_2) обеспечивает in-place свойство (ранга 2) для вычислительного графа Γ и в конечном итоге на теоретическом уровне доказывает, что на каждом шаге рекурсии алгоритм *FHT2IDS* требует аллоцирования дополнительного массива памяти размера не более $w + h$. Действительно, каждый рекурсивный вызов (без обработки дочерних вызовов) требует аллоцирования массива размера не более $w' + h$, точнее, массива размера $w' \leqslant w$ для хранения массива K (строка 10 алгоритма 15) и массива размера h для обработки всех бабочек (строка 9 алгоритма 13). Кроме того, доказано, что результаты работы алгоритмов *FHT2IDS* и *FHT2DS* совпадают, с точностью до известной перестановки K столбцов Хаф-образа, и вычислительная сложность (т.е. количество выполненных суммирований) обоих алгоритмов одинакова.

Обобщая вышеизложенное, можно сказать, что предложенный нами алгоритм *FHT2IDS* состоит из двух основных этапов: редукции вычислительного графа до вида объединения бабочек путем удаления треугольников (этот этап отсутствует для случая четного w), а затем последовательной обработки бабочек. Общий псевдокод алгоритма, который использует процедуры *ProcessButterfly* и *ProcessTriangle*, представлен в виде алгоритма 15.

Алгоритм 15 Алгоритм *FHT2IDS*

1: **Input:** изображение $I\colon \mathbb{Z}_w \times \mathbb{Z}_h \to \mathbb{A}$, его ширина w и высота h
2: **Output:** изображение $I\colon \mathbb{Z}_w \times \mathbb{Z}_h \to \mathbb{A}$, массив $K\colon \mathbb{Z}_w \to \mathbb{Z}_w$ индексов столбцов изображения J для их естественного переупорядочения (согласованного с результатом выполнения алгоритма *FHT2DS*)
3: **if** $w > 1$ **then**
4: $w_L \leftarrow \lfloor w/2 \rfloor$
5: $w_R \leftarrow w - w_R$
6: $I_L \leftarrow I(0 : w_L, :)$ ▷ выделение памяти не производится
7: $I_R \leftarrow I(w_L : w, :)$ ▷ выделение памяти не производится
8: $\langle I_L, K_L \rangle \leftarrow FHT2IDS(I_L, w_L, h)$
9: $\langle I_R, K_R \rangle \leftarrow FHT2IDS(I_R, w_R, h)$
10: $K \leftarrow CreateZeroedArray(w)$ ▷ выделяется память для массива размера w
11: **if** $w = 0 \bmod 2$ **then**
12: **for** $n \leftarrow 0$ **to** $w/2 - 1$ **do** ▷ последовательная обработка бабочек
13: $ProcessButterfly(I_L, I_R, w_L, w_R, h, K_L, K_R, n, K)$
14: **else** ▷ последовательная обработка треугольников и затем – бабочек
15: **for** $n \leftarrow 0$ **to** $w - 2[w/4] - 1$ **do**
16: $ProcessTriangle(I_L, I_R, w_L, w_R, h, K_L, K_R, n, K)$
17: **for** $n \leftarrow 0$ **to** $[w/4] - 1$ **do**
18: $ProcessButterfly(I_L, I_R, w_L, w_R, h, K_L, K_R, n, K)$
19: **else**
20: $K(0) \leftarrow 0$
return I, K

§ 7. Экспериментальная оценка предложенного алгоритма *FHT2IDS*

Чтобы сравнить время работы алгоритмов *FHT2DS* и *FHT2IDS*, мы реализовали их на языке C++ и протестировали на квадратных изображениях с линейным размером от 1 до 4100, заполненных случайными числами от 0 до 255. Тип входного изображения был 32-битным, с плавающей запятой или целым беззнаковым. Эксперименты проводились на процессоре Intel Core i9-9900KF с архитектурой x86_64. В реализации алгоритма *FHT2IDS* задействовалась операция векторного суммирования. Мы использовали интринсики семейства SSE. Операции выполнялись на 128-битных регистрах, которые могут хранить либо 4 числа с плавающей запятой одинарной точности, либо 4 беззнаковых целых.

Результаты представлены на рис. 6 и 7. Видно, что in-place версия алгоритма *FHT2DS* оказалась быстрее остальных. В среднем, во всем рассматриваемом диапазоне ширин, выигрыш в скорости от использования in-place версии составил около 26%. При этом чем больше линейный размер изображения, тем существеннее эффект от использования алгоритма *FHT2IDS*. В табл. 1 приведено среднее время работы для изображения размера 2048. Видно, что для этого размера ускорение составило около 28% для каждого типа данных входного изображения. Использование целочисленного типа незначительно повлияло на время работы: наиболее частая арифметическая операция в этих алгоритмах, сложение, выполняется практически одинаково быстро как на целых числах, так и на числах с плавающей запятой.

§ 8. Обсуждение

Предложенный алгоритм *FHT2IDS*, хотя и является in-place алгоритмом, не является in-order алгоритмом, что означает, что Хаф-образ, возвращаемый этим алгоритмом, в общем случае отличается от Хаф-образа, возвращаемого алгоритмом

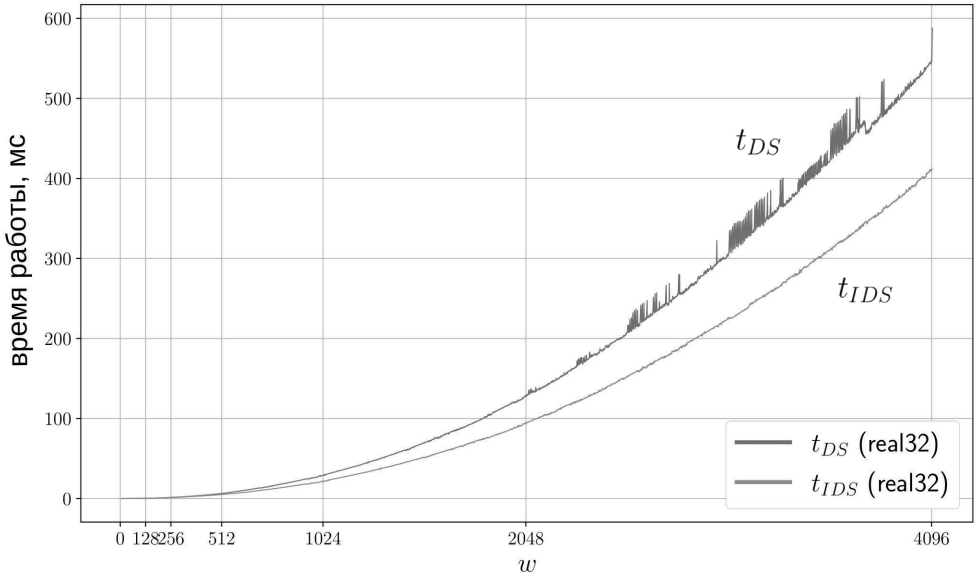


Рис. 6. График времени работы алгоритмов *FHT2DS* и *FHT2IDS* на изображениях 32-битного типа данных с плавающей запятой с линейным размером от 1 до 4100

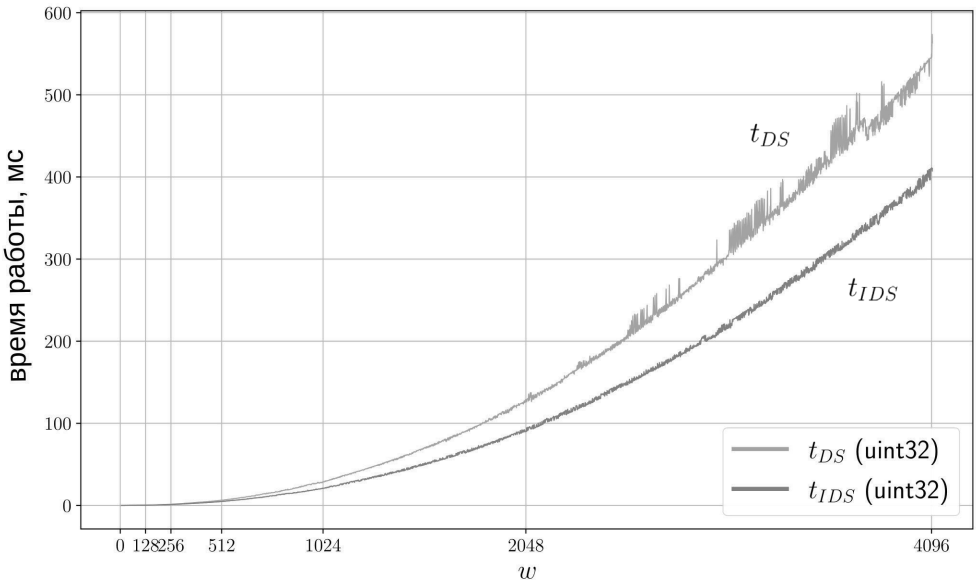


Рис. 7. График времени работы алгоритмов $FHT2DS$ и $FHT2IDS$ на изображениях 32-битного беззнакового целочисленного типа данных с линейным размером от 1 до 4100

Таблица 1

Среднее время работы алгоритмов $FHT2DS$ и $FHT2IDS$ на изображении размера 2048

	$\bar{t}_{DS}$	$\bar{t}_{IDS}$
real32	129 мс	93 мс
uint32	129 мс	93 мс

$FHT2DS$, перестановкой столбцов согласно отображению K. Для in-place алгоритма Кули – Тьюки для вычисления одномерного БПФ с основанием 2 и прореживанием по времени, как показано в работе [37], результирующий массив частот записывается в порядке битовой обратной записи, когда размер входного массива является степенью двойки. Поскольку, как было показано, вычислительные графы алгоритма $FHT2DS$ и алгоритма Кули – Тьюки для БПФ с основанием 2 и прореживанием по времени изоморфны для изображений четной ширины, можно заключить, что для ширин, являющихся степенями двойки, функция K также определяет порядок битовой обратной записи индексов столбцов Хаф-образа. Для произвольных ширин w порядок, определяемый функцией K, вычисляется в соответствии с уравнениями (23) и (24). Переупорядочение столбцов Хаф-образа в алгоритме $FHT2IDS$ требует дополнительных вычислительных затрат, и с целью оптимизации ПХ актуальна разработка не только in-place, но и in-order алгоритмов для вычисления БПХ [40].

Другим важным направлением для улучшения алгоритмов вычисления БПХ является разработка in-place модификаций более точных алгоритмов. В частности, алгоритм $FHT2DT$, предложенный в работах [27, 28], несколько медленнее алгоритма $FHT2DS$, но представляет более точный метод вычисления преобразования Хафа. Вычислительный граф алгоритма $FHT2DT$ имеет более сложную структу-

ру, и его эффективный анализ требует расширения результатов, приведенных в § 5. В этом направлении представляется возможным разработать более быстрые алгоритмы ПХ, сохраняя при этом высокую точность. Однако это, вероятно, потребует разработки более тонкого теоретического фундамента.

§ 9. Заключение

В настоящей статье предложен in-place алгоритм *FHT2IDS* для вычисления БПХ для изображений произвольной ширины. Предложенный алгоритм предстает in-place модификацией ранее предложенного в литературе алгоритма *FHT2DS* [26]. Нами приведено теоретическое обоснование корректности алгоритма *FHT2IDS*, показано, что его вывод совпадает с результатом работы алгоритма *FHT2DS* для изображений произвольной ширины. Это обоснование основано на анализе структуры вычислительного графа исходного алгоритма. Используемые методы доказательства могут быть полезны читателю при выводе in-place модификаций других алгоритмов, аналогичных тем, что применяются для вычисления БПХ или БПФ.

Продемонстрировано, что на каждом шаге рекурсии, при условии, что входное изображение имеет размерность $w \times h$, предложенный алгоритм *FHT2IDS* требует выделения массива размера не более $w + h$, что значительно меньше массива размера wh, который выделяется на каждом шаге рекурсии в рамках алгоритма *FHT2DS*. Снижение сложности алгоритма по объему вспомогательной памяти, в связи с оптимизацией управления кэшированием, а также уменьшением накладных расходов на копирование данных, приводят к значительному увеличению скорости при применении алгоритма *FHT2IDS* к большим изображениям по сравнению с алгоритмом *FHT2DS*. Экспериментальные результаты показывают, что алгоритм *FHT2IDS*, реализованный на языке C/C++, превосходит его out-of-place аналог *FHT2DS* на 26% по скорости. Алгоритм *FHT2IDS* реализован на языке Python и в настоящее время является частью общедоступной библиотеки *adrt* [41], которую читатель может найти полезной для собственных исследований. Мы рекомендуем читателю использовать разработанный алгоритм *FHT2IDS*, так как он существенно быстрее и значительно более экономен в смысле оперативной памяти по сравнению с алгоритмом *FHT2DS*.

СПИСОК ЛИТЕРАТУРЫ

1. *Hough P.V.C.* Machine Analysis of Bubble Chamber Pictures // Proc. 2nd Int. Conf. on High-Energy Accelerators and Instrumentation (HEACC 1959). CERN, Geneva, Switzerland. Sept. 14–19, 1959. P. 554–558.
2. *Rahmdel P.S., Comley R., Shi D., McElduff S.* A Review of Hough Transform and Line Segment Detection Approaches // Proc. 10th Int. Conf. on Computer Vision Theory and Applications (VISAPP 2015). Berlin, Germany. Mar. 11–14, 2015. V. 2. P. 411–418. https://doi.org/10.5220/0005268904110418
3. *Mukhopadhyay P., Chaudhuri B.B.* A Survey of Hough Transform // Pattern Recognit. 2015. V. 48. № 3. P. 993–1010. https://doi.org/10.1016/j.patcog.2014.08.027
4. *Illingworth J., Kittler J.* A Survey of the Hough Transform // Comput. Vision Graph. Image Process. 1988. V. 44. № 1. P. 87–116. https://doi.org/10.1016/S0734-189X(88)80033-1
5. *Xu Z., Shin B., Klette R.* Accurate and Robust Line Segment Extraction Using Minimum Entropy with Hough Transform // IEEE Trans. Image Process. 2014. V. 24. № 3. P. 813–822. https://doi.org/10.1109/TIP.2014.2387020
6. *Алиев М.А., Николаев Д.П., Сараев А.А.* Построение быстрых вычислительных схем настройки алгоритма бинаризации Ниблэка // Тр. ИСА РАН. 2014. Т. 64. № 3. С. 25–34.
7. *Nikolaev D.P., Nikolayev P.P.* Linear Color Segmentation and Its Implementation // Comput. Vis. Image Und. 2004. V. 94. № 1–3. P. 115–139. https://doi.org/10.1016/j.cviu.2003.10.012

8. *Saha S., Basu S., Nasipuri M., Basu D.* A Hough Transform Based Technique for Text Segmentation // J. Comput. 2010. V. 2. № 2. P. 134–141.
9. *Кунина И.А., Гладилин С.А., Николаев Д.П.* Слепая компенсация радиальной дисторсии на одиночном изображении с использованием быстрого преобразования Хафа // Компьютерная оптика. 2016. Т. 40. № 3. С. 395–403. https://doi.org/10.18287/2412-6179-2016-40-3-395-403
10. *Асватов Е.Н., Ершов Е.И., Николаев Д.П.* Робастная ортогональная линейная регрессия для маломерных гистограмм // Сенсорные системы. 2017. Т. 31. № 4. С. 331–342.
11. *Brady M.L., Yong W.* Fast Parallel Discrete Approximation Algorithms for the Radon Transform // Proc. 4th Ann. ACM Symp. on Parallel Algorithms and Architectures (SPAA'92). San Diego, California, USA. June 29 – July 1, 1992. P. 91–99. https://doi.org/10.1145/140901.140911
12. *Карпенко С.М., Ершов Е.И.* Исследование свойств диадического паттерна быстрого преобразования Хафа // Пробл. передачи информ. 2021. Т. 57. № 3. С. 102–111. https://doi.org/10.31857/S0555292321030074
13. *Jahan R, Suman P., Singh D.K.* Lane Detection Using Canny Edge Detection and Hough Transform on Raspberry Pi // Int. J. Adv. Res. Comput. Sci. 2018. V. 9. № 2. P. 85–89.
14. *Thongpan N., Rattanasiriwongwut M., Ketcham M.* Lane Detection Using Embedded System // Int. J. Comput. Internet Manag. 2020. V. 28. № 2. P. 46–51.
15. *Panfilova E., Shipitko O.S., Kunina I.* Fast Hough Transform-Based Road Markings Detection For Autonomous Vehicle // 13th Int. Conf. on Machine Vision (ICMV 2020). Rome, Italy. Nov. 2–6, 2020. Proc. SPIE. V. 11605. P. 671–680. https://doi.org/10.1117/12.2587615
16. *Котов А.А., Коноваленко И.А., Николаев Д.П.* Прослеживание объектов в видеопотоке, оптимизированное с помощью быстрого преобразования Хафа // ИтиВС. 2015. № 1. С. 56–68.
17. *Tropin D.V., Ilyuhin S.A., Nikolaev D.P., Arlazarov V.V.* Approach for Document Detection by Contours and Contrasts // Proc. 25th Int. Conf. on Pattern Recognition (ICPR 2020). Milan, Italy. Jan. 10–15, 2021. P. 9689–9695. https://doi.org/10.1109/ICPR48806.2021.9413271
18. *Bezmaternykh P.V., Nikolaev D.P.* A Document Skew Detection Method Using Fast Hough Transform // 12th Int. Conf. on Machine Vision (ICMV 2019). Amsterdam, Netherlands. Nov. 16–18, 2020. Proc. SPIE. V. 11433. P. 132–137. https://doi.org/10.1117/12.2559069
19. *Min-Allah N., Qureshi M.B., Alrashed S., Rana O.F.* Cost Efficient Resource Allocation for Real-Time Tasks in Embedded Systems // Sustainable Cities And Society. 2019 V. 48. Article No. 101523. https://doi.org/10.1016/j.scs.2019.101523
20. *Gupta R.K., De Micheli G.* Specification and Analysis of Timing Constraints for Embedded Systems // IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 1997. V. 16. № 3. P. 240–256. https://doi.org/10.1109/43.594830
21. *Surianarayanan C., Lawrence, J.J., Chelliah P.R., Prakash E., Hewage C.* A Survey on Optimization Techniques for Edge Artificial Intelligence (AI) // Sensors. 2023. V. 23. № 3. Papaer No. 1279 (33 pp.). https://doi.org/10.3390/s23031279
22. *Ranjith M.S., Parameshwara S., Pavan Yadav A., Hegde S.* Optimizing Neural Network for Computer Vision Task in Edge Device, https://arxiv.org/abs/2110.00791 [cs.CV], 2021.
23. *Sharmila B.S., Santhosh H.S., Parameshwara S., Swamy M.S., Baig W.H., Nanditha S.V.* Optimizing Deep Learning Networks for Edge Devices with an Instance of Skin Cancer and Corn Leaf Disease Dataset // SN Comput. Sci. 2023. V. 4. Article No. 793. https://doi.org/10.1007/s42979-023-02239-5
24. *Comeagă A.-M., Marin I.* Memory Management Strategies for an Internet of Things System, https://arxiv.org/abs/2311.10458 [cs.SE], 2023.
25. *Almutairi R., Bergami G., Morgan G.* Advancements and Challenges in IoT Simulators: A Comprehensive Review // Sensors. 2024. V. 24. № 5. Paper No. 1511 (35 pp.), https://doi.org/10.3390/s24051511

26. *Anikeev F.A., Raiko G.O., Limonova E.E., Aliev M.A., Nikolaev D.P.* Efficient Implementation of Fast Hough Transform Using CPCA Coprocessor // Program. Comput. Soft. 2021. V. 47. № 5. P. 335–343. https://doi.org/10.1134/S0361768821050029
27. *Kazimirov D., Nikolaev D., Rybakova E., Terekhin A.* Generalization of Brady–Yong Algorithm for Fast Hough Transform to Arbitrary Image Size, https://arxiv.org/abs/2411.07351 [cs.CV], 2024.
28. *Kazimirov D., Nikolaev D., Rybakova E., Terekhin A.* Generalization of Brady–Yong Algorithm for Fast Hough Transform to Arbitrary Image Size // Proc. 5th Symp. on Pattern Recognition and Applications (SPRA 2024). Istanbul, Turkey. Nov. 11–13, 2024 (to appear).
29. *Gava M.A., Rocha H.R.O., Faber M.J., Segatto M.E.V., Wörtche H., Silva J.A.L.* Optimizing Resources and Increasing the Coverage of Internet-of-Things (IoT) Networks: An Approach Based on LoRaWAN // Sensors. 2023. V. 23. № 3. Paper No. 1239 (17 pp.). https://doi.org/10.3390/s23031239
30. *Almurshed O., Meshoul S., Muftah A., Kaushal A.K., Almoghamis O., Petri I., Auluck N., Rana O.* A Framework for Performance Optimization of Internet of Things Applications // Euro-Par 2023: Parallel Processing Workshops (Euro-Par 2023 Int. Workshops. Limassol, Cyprus. Aug. 28 – Sept. 1, 2023. Revised Selected Papers, Part I). Lect. Notes Comput. Sci. V. 14351. Cham: Springer, 2024. P. 165–176. https://doi.org/10.1007/978-3-031-50684-0_13
31. *Chakraborty S., Mukherjee A., Raman V., Satti S.R.* A Framework for In-place Graph Algorithms // 26th Annu. Europ. Symp. on Algorithms (ESA 2018). Helsinki, Finland. Aug. 20–22, 2018. Leibniz Int. Proc. Inform. (LIPIcs). V. 112. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany: Dagstuhl Publ., 2018. P. 13:1–13:16. https://doi.org/10.4230/LIPIcs.ESA.2018.13
32. *Axtmann M., Witt S., Ferizovic D., Sanders P.* Engineering In-place (Shared-Memory) Sorting Algorithms // ACM Trans. Parallel Comput. 2022. V. 9. № 1. P. 1–62. https://doi.org/10.1145/3505286
33. *Gu Y., Obeya O., Shun J.* Parallel In-place Algorithms: Theory and Practice // 2nd Symp. on Algorithmic Principles of Computer Systems (APOCS 2020). Virtual Conf. Jan. 13, 2021. P. 114–128. https://doi.org/10.1137/1.9781611976489.9
34. *Brönnimann H., Chan T.M., Chen E.Y.* Towards In-place Geometric Algorithms and Data Structures // Proc. 12th Annu. Symp. on Computational Geometry (SCG'04). Brooklyn, New York, USA. June 8–11, 2004. P. 239–246. https://doi.org/10.1145/997817.997854
35. *Kuszmaul W., Westover A.* Cache-Efficient Parallel-Partition Algorithms Using Exclusive-Read-and-Write Memory // Proc. 32nd ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'20). Virtual Event, USA. July 15–17, 2020. P. 551–553. https://doi.org/10.1145/3350755.3400234
36. *Bramas B., Bramas Q.* On the Improvement of the In-place Merge Algorithm Parallelization, https://arxiv.org/abs/2005.12648 [cs.DC], 2020.
37. *Cooley J.W., Tukey J.W.* An Algorithm for the Machine Calculation of Complex Fourier Series // Math. Comp. 1965. V. 19. № 90. P. 297–301. https://doi.org/10.2307/2003354
38. *Brady M.L., Yong W.* Fast Parallel Discrete Approximation Algorithms for the Radon Transform // Proc. 4th Ann. ACM Symp. on Parallel Algorithms and Architectures (SPAA'92). San Diego, California, USA. June 29 – July 1, 1992. P. 91–99. https://doi.org/10.1145/140901.140911
39. *Khanipov T.* Computational Complexity Lower Bounds of Certain Discrete Radon Transform Approximations, https://arxiv.org/abs/1801.01054 [cs.CC], 2018.
40. *Johnson H., Burrus C.* An In-order, In-place Radix-2 FFT // Proc. ICASSP'84: IEEE Int. Conf. on Acoustics, Speech, and Signal Processing. San Diego, CA, USA. Mar. 19–21, 1984. P. 473–476. https://doi.org/10.1109/ICASSP.1984.1172660
41. IITP Vision Lab. adrt: Approximate Discrete Radon Transform. GitHub repository, accessed 21.10.2024. https://github.com/iitpvisionlab/adrt

Казимиров Данил Дмитриевич
Институт проблем передачи информации
им. А.А. Харкевича РАН, Москва
Московский государственный университет
им. М.В. Ломоносова, Москва
ООО “Смарт Энджинс Сервис”, Москва
d.kazimirov@smartengines.com
Николаев Дмитрий Петрович
ООО “Смарт Энджинс Сервис”, Москва
Федеральный исследовательский центр
“Информатика и управление” РАН, Москва
d.p.nikolaev@smartengines.com
Рыбакова Екатерина Олеговна
Московский государственный университет
им. М.В. Ломоносова, Москва
ООО “Смарт Энджинс Сервис”, Москва
e.rybakova@smartengines.com
Терехин Арсений Павлович
Институт проблем передачи информации
им. А.А. Харкевича РАН, Москва
ars@iitp.ru

Поступила в редакцию
07.12.2024
После доработки
18.12.2024
Принята к публикации
25.12.2024