УДК 519.72

Д.В. Поляков, А.И. Попов, С.А. Дузькрятченко

СТРУКТУРА ДАННЫХ ДЛЯ СОЗДАНИЯ АССОЦИАТИВНОГО МАССИВА НА ОСНОВЕ МОНОТОННЫХ ХЕШ-ФУНКЦИЙ

Тамбовский государственный технический университет

Цель: Снижение сложности алгоритмов решения задач поиска ближайшего объекта, интервального поиска, путём разработки структуры данных для создания хеш-таблицы на основе монотонных хеш-функций.
Подход: Методика исследований основана на теории множеств, теории информационного поиска и теории алгоритмов.
Результаты: Предложена структура данных для создания ассоциативного массива на основе монотонных хеш-функций, алгоритмы работы с ней и проведён сравнительный анализ предложенной и существующих структур данных для хеш-таблиц.
Ограничения исследований: Исследования не затрагивают вопроса выбора хеш-функции, требуя от неё только монотонности. Также в работе очерчен круг необходимых в дальнейшем теоретических и экспериментальных исследований, для выбора параметров предложенной структуры и условий рехеширования.
Оригинальность / значение: Предлагаемая структура данных и алгоритмы работы с ней показывают хорошие результаты при решении задач о близости и интервального поиска в сравнении с другими обобщениями индексных массивов для хеш-таблиц.

Ключевые слова: хеш-таблица, хеш-функция, ассоциативный массив, задача поиска идентичных объектов, задача о близости, задача интервального поиска, интервальный поиск, информационный поиск.

Хранение и поиск информации, безусловно, являются ключевыми процессами любой информационной системы (ИС). Их эффективная реализация рассматривается в двух важнейших областях современной информатики: теории информационного поиска и теории баз данных. Однако задачи информационного поиска крайне разнообразны. Так, Гасанов и Кудрявцев в своей работе «Теория хранения и поиска информации» [1] выделяют базовые задачи информационного поиска. Они охарактеризованы наличием таких объектов, как «запрос», представляющий собой некий минимальный элемент, задающий семантику интересов пользователя; «запись» – поисковый образ данных, представленный обычно в виде множества полей базы данных, и «отображение», позволяющее определить подмножество записей, которое необходимо вернуть в ответ на соответствующий запрос. Таким образом, по Гасанову, задача информационного поиска, формализуется кортежем

$$<X,Y,\rho:X\rightarrow P(Y)>, \quad (1)$$

где X – это универсум запросов, Y – множество записей, ρ – отображение, а $P(Y)$ – булеан (множество всех подмножеств) Y.

Вместе с тем, существует широкий класс задач, в котором само построение отображения ρ нетривиально. Например, задачи поиска текстовых сведений ставят вопрос о релевантности и пертинентности результатов информационного поиска. Под релевантностью понимается соответствие результатов информационного поиска запросу пользователя, а под пертинентностью – соответствие полученных результатов его информационной потребности [2]. Эти понятия были рассмотрены в работах Ландэ, Санарского, Безсудного, Робертсона и других [3-10]. По сути, определение множества документов, релевантных запросу пользователя или пертинентных его информационным потребностям, и есть построение отображения ρ.

На сегодняшний день разработано множество механизмов определения релевантности и пертинентности: начиная с расширенной булевой и пространственно-векторной моделей, разработанных Солтоном, Фоксом, By [11-14], заканчивая методами *text meaning*, компьютерной лингвистики и анализа коллокаций термов [15-21].

Однако при построении отображения ρ или же, имея его в тривиальном виде, мы

сталкиваемся с задачей информационного поиска (1). Существует её тривиальное решение, заключающиеся в последовательном обходе множества Y и проверки каждого $y \in Y$ на принадлежность результату отображения запроса. Такой подход вполне применим, однако крайне неэффективен с позиции алгоритмической сложности решения задач информационного поиска.

Вместе с тем, даже несмотря на постоянный рост вычислительной мощности комплектующих ИС, эффективные с позиции времени решения задач информационного поиска являются крайне важными для современных ИС. Это происходит из-за того, что увеличение объёмов обрабатываемой информации существенно опережает рост вычислительной мощности устройств. На сегодняшний день существует широкий класс задач, например, задачи мониторинга или учёта, предполагающих статистическую или интеллектуальную обработку накопленных данных и потому требующих выполнения огромного числа различного вида поисковых запросов. Кроме того, работа с данными зачастую происходит по сети, что приводит к дополнительным временным потерям. Такая ситуация свидетельствует о несомненной актуальности исследований в области структур данных и алгоритмов, позволяющих эффективно решать задачи информационного поиска вида (1).

Рассмотрим основные подвиды задачи (1) более подробно:

- задачи поиска с коротким ответом [1] – подвид задачи (1), в которых результат работы отображения ρ содержит ограниченное подмножество Y. Задачи с коротким ответом, в свою очередь, разделяются на следующие типы:

a) задача поиска идентичных объектов, которая заключается в нахождении объекта идентичного объекту-запросу – она рассмотрена в работах Гасанова, Ландиса, Белобродского, Решетникова, Фергюсона и других [1, 22-26];

b) задача о близости, заключающаяся в поиске на множестве с заданным линейным бинарным отношением порядка или предпорядка объекта, ближайшего к объекту-запросу и исследованная в работах Гасанова, Ювала, Фредмана, Брукхарда, Минского и других [1, 26-29];

- задача интервального поиска [1], заключающаяся в поиске на множестве с заданными линейными бинарными отношениями порядка или предпорядка элементов находящихся в интервале, определённом поисковым запросом. Эти задачи также были исследованы во многих работах [1, 30-38].

Для решения рассмотренных задач информационного поиска используется два вида структур данных с алгоритмами их обработки: деревья и хеш-таблицы [39, 40]. Под деревьями, в данном случае, понимаются связные ациклические графы. Такие структуры данных, повышающие скорость поиска хранимых в них элементов, называют ассоциативными массивами [39].

Хранение данных с помощью деревьев в большинстве случаев представлено такой структурой, как бинарное дерево и множеством обслуживающих его алгоритмов. Различные алгоритмы обработки бинарных деревьев определяют разные виды построенных на них ассоциативных массивов, таких как, например, АВЛ-деревья, вероятностные деревья, красно-чёрные деревья [39, 40].

Сложность поиска идентичных объектов и решения задачи о близости на эффективных бинарных деревьях как в среднем, так и в худшем случае, пропорциональна логарифму от количества хранимых записей. Вместе с тем, сложность задачи интервального поиска может меняться и, зачастую, довольно высока из-за того, что переход к следующему элементу является нетривиальной задачей с неконстантной (зависящей от количества хранимых записей) алгоритмической сложностью.

При использовании хеш-таблиц можно добиться константного (не зависящего от количества хранимых записей) в среднем и даже в худшем случае времени поиска элементов [39, 40]. Вместе с тем, в хеш-таблицах нетривиальной задачей является добавление элементов, а также решение задач о близости и интервального поиска. Кроме того, при организации хеш-таблиц, зачастую, неэффективно используется память.

В данной работе предлагается структура данных, которая, в случае использования её для построения хеш-таблицы, при определённых условиях позволит улучшить средние и худшие показатели скорости решения задач о близости и интервального поиска.

Сравнительный анализ хеш-таблиц

Идея хранения и поиска данных на основе хеш-таблицы состоит в том, чтобы построить отображение из множества Y на множество $\{n \in N_0 \mid n<m\}$, где m – некоторое число, ограничивающие область допустимых значений отображения, а N_0 – множество натуральных чисел и 0. Такое отображение называется хеш-функцией. Для рассматриваемого множества допустимых значений хеш-функции, ограниченного сверху натуральным числом m обозначим хеш-функцию как $h_m: Y \to \{n \in N_0 \mid n<m\}$.

В качестве структуры данных для создания хеш-таблицы обычно используется индексный массив или некоторое его обобщение.

Индексный массив представляет собой коллекцию записей расположенных в *RAM*-памяти последовательно, что позволяет, зная индекс, вычислить адрес записи, и в силу свойств *RAM*-памяти обратиться к записи за константное время.

Пусть A_m – индексный массив, а $A_m[i]$ – его i-тый элемент, $0 \le i < m$. При добавлении записи в хеш-таблицу вычисляется с помощью хеш-функции индекс массива и соответствующему его элементу присваивается запись. То есть добавление записи $y \in Y$ выглядит как $A_m[h_m(y)] := y$, где символ := означает присваивание.

Тогда решение задачи поиска идентичного объекта будет тривиальным: если для нашего запроса $x \in X$ $A_m[h_m(x)] = x$, то элемент найден, в противном случае данный элемент в хеш-таблице отсутствует.

Для сравнительной оценки эффективности работы ассоциативных массивов исследуем потребляемые ими ресурсы, такие как сложность алгоритмов решения тех или иных задач и объём требуемой ассоциативному массиву памяти. Вместе с тем, оценка данных величин в абсолютных значениях крайне затруднительна, ибо они зависят от технических характеристик устройств, на которых запускаются алгоритмы, а для хеш-таблиц ещё и от самого множества записей – Y. В качестве способа оценки сложности алгоритмов и используемой хеш-таблицей памяти воспользуемся классическим подходом: оценкой асимптотической сложности с помощью O-нотации. Этот подход широко используется, а его адекватность хорошо исследована, например, в работах Кнута, Кормена, Лейзерсона, Ривеста, Штайна [39, 41] и других.

Вернёмся к рассмотрению хеш-таблиц. Из рассмотренных алгоритмов следует, что время поиска и добавления элемента в хеш-таблицу не зависит от количества хранимых элементов, то есть оно константно, что на языке O-нотации обозначается как $O(1)$. Это хорошая оценка, так как большинство ассоциативных массивов, работающих на основе деревьев, имеют асимптотическую сложность для тех же операций $O(\log n)$.

Вместе с тем, при решении задачи поиска идентичного объекта у хеш-таблиц имеют место два существенных недостатка:

- в массиве A_m присутствуют пустые ячейки, что в отличие от ассоциативных массивов на основе деревьев, приводит к существенному перерасходу памяти;
- существует возможность коллизий – ситуаций, когда две разных записи хеш-функция отображает в один и тот же индекс.

Стоит отметить, что зачастую, в случае использования хеш-функций с хорошими характеристиками, верно правило: чем меньше в хеш-таблице лишней памяти, тем больше в ней коллизий.

Для решения проблемы коллизий существует множество подходов. Рассмотрим подробнее основные [39, 40].

1. Линейное зондирование – метод, при котором в случае возникновения коллизии осуществляется линейных проход по массиву с поиском незанятых ячеек для вставки эле-

мента. Очевидным плюсом линейного зондирования является снижение объёма лишней памяти. Однако при поиске записи, в случае, если она была вставлена в массив путём линейного зондирования время работы алгоритма, которому в процессе поиска приходится повторять весь путь до искомого объекта, деградирует вплоть до $O(n)$. А $O(n)$ соответствует оценке простейшего алгоритма полного перебора.

2. Двойное хеширование – метод, который обрабатывает коллизии путём взятия новой хеш-функции, задающей циклический сдвиг по A_m. Поиск осуществляется путём того же циклического сдвига. Плюсы и минусы двойного хеширования аналогичны линейному зондированию, с той лишь разницей, что удачный выбор второй хеш-функции позволит уменьшить число повторных коллизий в сравнении с линейным зондированием. С другой стороны, применяемые на практике в качестве отображений для повторного хеширования, квадратичные функции [40] могут привести к тому, что добавление элемента будет невозможно, при наличии в A_m большого количества пустых ячеек. Такая ситуация может произойти из-за того, что циклический сдвиг в общем случае не будет обходить все элементы массива, в отличие от процесса поиска свободного места на основе линейного зондирования.

3. Использование вместо индексного массива некоторого его обобщения, которое позволяет в одну и ту же ячейку добавлять несколько записей. Такие обобщения используют следующие алгоритмы разрешения коллизий:

- метод цепочек переполнения, заключающийся в том, что каждый элемент A_m является началом линейного списка, в который и добавляются записи, попавшие в данную ячейку;
- подход, предлагающий использование бинарных деревьев вместо списков, что позволяет ускорить поиск среди элементов, попавших в одну ячейку;
- механизм разрешения коллизий, основанный на использовании для каждой ячейки своей отдельной хеш-таблицы.

Основным недостатком хеш-таблиц, рассмотренных в третьем пункте списка, при решении задачи поиска идентичного объекта является, то, что в отличие от линейного зондирования и двойного хеширования, в ассоциативном массиве хранится большой объём пустых ячеек. При этом, хотя поиск идентичного объекта в среднем осуществляется за $O(1)$, благодаря выбору хорошей хеш-функции, минимизирующей коллизии, в случае их возникновения решаемая задача сводится к поиску среди хранимых записей, составляющих с коллизию запросом. Сложность такого поиска определяется структурой данных, разрешающей коллизию (список/дерево/хеш-таблица) относительно мощности множества записей, составляющих коллизию с объектом-запросом. В дальнейшем такое множество будем называть *коллизионным доменом* запроса.

Пусть C – некоторый алгоритм разрешения коллизий. Тогда хеш-таблица задаётся набором элементов

$$<h_m, A_m, C>. \quad (2)$$

Рассмотрим теперь эффективность представленных хеш-таблиц при решении двух оставшихся поисковых задач, исследуемых в данной работе.

В общем случае, хеш-таблицы не предназначены для решения задач о близости и интервального поиска. Действительно, хеш-функция, в общем случае, может отобразить близкие значения в произвольную пару индексов. Вместе с тем, для того, чтобы решать задачу о близости с помощью хеш-таблиц, достаточно потребовать от используемых в них хеш-функций монотонность.

Отметим, что требование монотонности к хеш-функциям при решении задач о близости или интервального поиска корректны, так как в формулировке обеих задач присутствует условие наличия отношения порядка или предпорядка на множестве Y. Более того, в рамках информатики данное условие никак не ограничивает общность поисковой задачи в силу того, что на множестве любых, хранящихся в памяти компьютера, объектов можно установить отношение порядка. Действительно, если даже объекты не имеют естественного подхода к

установлению такого бинарного отношения, как, например, лексикографический порядок на множестве строк или порядок, задаваемый отношениями «<» и «>», на любом числовом множестве, они всё равно представляют собой битовые последовательности, которые можно интерпретировать как числа в двоичной системе счисления.

На сегодняшний день предложено и исследовано большое количество монотонных хеш-функций. Эти исследования представлены в работах Гасанова, Белазегуа, Болди, Дроссоса и других [1, 42-44]. Кроме того, монотонные хеш-функции представлены и в работах [45-48], например, в статье «Генератор монотонных хеш-функций для ассоциативного массива» был предложен метод получения хеш-функции на основе генератора с использованием произвольных непрерывных строго возрастающих биекций из множества хранимых записей на отрезок [0, 1] [48].

Для оценки сложности решения задачи о близости и интервального поиска сформулируем некоторые определения и утверждение, докажем его, а также введём в рассмотрение необходимые величины.

Пусть имеем задачу информационного поиска вида (1), которая решается посредствам хеш-таблицы вида (2). Пусть $|Y| = n$, то есть n определяет количество хранимых записей.

Активной вершиной назовём элемент $Am[i]$, если $(\exists y \in Y)(h_m(y) = i)$. Другими словами, активными вершинами будем называть все ячейки массива Am, в которые попадают записи Y при построении отображения этих записей с помощью хеш-функции.

Пассивной вершиной назовём элемент $Am[i]$, если $(\forall y \in Y)(h_m(y) \neq i)$. Другими словами пассивными вершинами будем называть все ячейки массива Am, в которые в результате построения отображения с помощью хеш-функции для элементов множества Y не попало ни одной записи.

Пустыми вершинами назовём все ячейки массива Am, которые не хранят в себе записи. *Заполненными вершинами* назовём все ячейки массива Am, которые хранят в себе запись и не являются активными.

Отметим, что в данных ранее определениях ячейки массива Am названы вершинами. Это было из-за того, что предлагаемую в данной статье структуру данных удобно рассматривать как граф, частью вершин которого будут ячейки массива Am. В дальнейшем будем называть как вершинами, так и ячейками в зависимости от контекста.

Может показаться, что понятия пустой и пассивной вершины синонимичны, однако это не так. Пустые вершины, безусловно, всегда являются пассивными. В то время как пассивные вершины могут содержать записи, попавшие туда в результате работы алгоритма разрешения коллизий (C). Так, если C представляет собой методы двойного хеширования или линейного зондирования, множества пассивных и пустых вершин не будут совпадать в случае появления и обработки хотя бы одной коллизии. Вместе с тем, для группы методов разрешения коллизий, использующих дополнительные структуры, таких, например, как метод цепочек переполнения, множества пассивных и пустых вершин всегда совпадает.

Утверждение: из монотонности хеш-функции следует, что ближайшая к искомой (x) запись попала либо в ту же вершину, что и x, либо в одну из соседних по A_m активных вершин. Докажем утверждение. Предположим противное: ближайшая запись к запросу x не попала ни в ячейку $A_m[h_m(x)]$ ни в соседние к ней активные вершины. Представим соседние непустые элементы как $A_m[h_m(x) - \delta_1]$ и $A_m[h_m(x) + \delta_2]$, где δ_1 и δ_2 – некоторые натуральные числа. Обозначим искомую запись y_x, $y_x \in Y$. Тогда

$$h_m(y_x) \in \overline{0, h_m(x) - \delta_1 - 1} \cup \overline{h_m(x) + \delta_2 + 1, m}. \quad (3)$$

Действительно, в противном случае, если бы $h_m(y_x) \in \overline{h_m(x) - \delta_1, h_m(x) + \delta_2}$, запись y_x попала бы в $A_m[h_m(x)]$ или $A_m[h_m(x) - \delta_1]$ или $A_m[h_m(x) + \delta_2]$, что невозможно из-за нашего

предположения, либо в пассивные вершины между $A_m[h_m(x)]$ и $A_m[h_m(x)-\delta_1]$ или $A_m[h_m(x)]$ и $A_m[h_m(x)+\delta_2]$, а это невозможно по определению пассивных вершин.

Разобьём рассмотрение условия (3) на два случая и покажем ошибочность изначального предположения в обоих.

1. $h_m(y_x)\in\overline{0,h_m(x)-\delta_1-1}$. Рассмотрим произвольный $y^*\in Y$, такой что $h_m(y^*)=h_m(x)-\delta_1$. Хотя бы одна такая запись точно существует в силу того, что $A_m[h_m(x)-\delta_1]$ активная вершина по построению. Относительно y^* очевидно неравенство

$$h_m(y_x) < h_m(y^*) < h_m(x).$$

Тогда в силу монотонности $h_m(.)$ получим

$$y_x < y^* < x,$$

что противоречит нашему предположению о том, что y_x – ближайшая запись к искомому объекту x.

2. $h_m(y_x)\in\overline{h_m(x)+\delta_2+1,m}$. Рассмотрим произвольный $y^*\in Y$, такой, что $h_m(y^*)=h_m(x)+\delta_2$. Хотя бы одна такая запись точно существует в силу того, что $A_m[h_m(x)+\delta_2]$ активная вершина по построению. Относительно y^* очевидно неравенство

$$h_m(x) < h_m(y^*) < h_m(y_x).$$

Тогда в силу монотонности $h_m(.)$ получим

$$x < y^* < y_x,$$

что противоречит нашему предположению о том, что y_x – ближайшая запись к искомому объекту x.

Из рассмотренных случаев, очевидно, что при выполнении условия (3) неизбежно получаем противоречие, а значит наше предположение неверно. Утверждение доказано.

Так как в данной работе не рассматриваются конкретные реализации хеш-функций и статистические характеристики хранимых элементов, для оценки сложности алгоритмов решения задач о близости и интервального поиска исследуемыми хеш-таблицами введём величины:

- k – математическое ожидание количества коллизий в одном элементе массива;
- l – средняя длина непрерывной последовательности пустых вершин;
- c – число элементов в искомом интервале;
- m_k – математическое ожидание числа элементов хеш-таблицы, использующейся для разрешения коллизий.

Легко заметить, что методы линейного зондирования и двойного хеширования не очень подходят для решения задачи о близости, так как записи каждого коллизионного домена могут оказаться в любом элементе A_m. Вследствие этого, поиск ближайшего элемента можно осуществить лишь полным обходом массива A_m за $O(m)$, что хуже, чем поиск по линейному списку записей за $O(n)$, так как для хеш-таблиц с рассматриваемыми алгоритмами разрешения коллизий $n\le m$.

Аналогично дело обстоит и с интервальным поиском, который можно осуществить лишь полным обходом A_m. Средний и худший случаи решения задачи о близости в данном случае совпадают.

В случаях разрешения коллизий методами линейного зондирования и двойного хеширования все n записей хранятся в массиве Am. Отсюда легко оценить перерасход памяти как $O(m\text{-}n)$.

В научной литературе [39] отмечается, что «при вполне обоснованных допущениях математическое ожидание поиска элемента в хеш-таблице составляет $O(1)$». В данном случае, под «вполне обоснованными допущениями» понимается эффективность h_m и величина

массива m. Как уже отмечалось ранее, не выбор h_m, не величина массива A_m в рамках данной работы не рассматривается. Потому и сравнение хеш-таблиц по среднему времени поиска идентичного объекта проводиться не будет, так как оно зависит лишь от выбора m и h_m.

Рассмотрим теперь эффективность работы хеш-таблиц, которые используют обобщения индексных массивов, позволяющие добавлять в одну и ту же ячейку массива A_m несколько записей, благодаря тому, что за этой самой ячейкой скрывается другая структура данных. Назовём эту структуру вспомогательной и введём для её оценки величину $T(k)$, которая определяет сложность разрешения задачи о близости в соответствующей структуре. Так, при использовании метода цепочек переполнения вспомогательной структурой является неупорядоченный список и $T(k) = O(k)$. Если же в качестве такой структуры выбрано бинарное дерево, то $T(k) = O(log(k))$.

Решение задач о близости для рассматриваемых хеш-таблиц в силу доказанного утверждения равносильно получению трёх ячеек: основной за $O(1)$, соседних активных ячеек, путём обхода A_m за $O(l)$; и решению для каждой из полученных ячеек задачи о близости за $T(k)$. Таким образом, оценим сложность решения задачи о близости как $O(l) + T(k)$.

Решение задачи интервального поиска состоит из решения двух задач о близости, соответствующих левой и правой границам интервала и обходу хеш-таблицы. Так как по итогам обхода будет найдено c значений, количество активных ячеек A_m в интервале будет равно $\frac{c}{k}$. Тогда, для решения задачи интервального поиска необходимо обойти $\frac{c}{k} \cdot l$ ячеек A_m и $\frac{c}{k}$ коллизионных домена из k элементов.

Таким образом, оценка сложности решения задачи интервального поиска (T_i) для хеш-таблиц, которые используют обобщения индексных массивов, позволяющие добавлять в одну и ту же ячейку массива A_m несколько записей, имеет вид

$$T_i = O(l) + T(k) + O\left(\frac{c}{k} \cdot l\right) + O\left(\frac{c}{k} \cdot t_k\right),$$

где t_k - сложность обхода коллизионного домена из k элементов при заданном методе разрешения коллизий, или

$$T_i = O(l) + T(k) + O\left(\frac{c}{k} \cdot \left(l + t_k\right)\right). \quad (4)$$

Так как в случае исследуемой группы методов разрешения коллизий – все элементы коллизионного домена хранятся в отдельных структурах данных, лишнюю память оценим как общее число вершин за вычетом активных вершин, то есть объём неиспользуемой памяти оценивается как $O\left(m - \frac{n}{k}\right)$. А в случае, когда дополнительная структура представляет собой хеш-таблицу, тратится ещё больше памяти, так как каждая дополнительная хеш-таблица имеет пустые ячейки. В среднем каждая вспомогательная хеш-таблица имеет $m_k - k$ незанятых ячеек, в то время как общее число вспомогательных хеш-таблиц оценивается как $\frac{n}{k}$. С учётом ячеек, оставшихся свободными в основной таблице, оценка неиспользуемой памяти принимает вид: $O\left(m - \frac{n}{k} + \left(m_k - k\right) \cdot \frac{n}{k}\right)$.

Однако, приняв во внимание то, что $\left(m_k - k\right)$ в среднем не может быть меньше 1, но и асимптотически больше k. Это означает, что асимптотическая оценка объёма неиспользуемой памяти корректно оценить как $O(m)$.

Полученные оценки асимптотической сложности алгоритмов и требуемой памяти для различных способов разрешения коллизий приведены в табл. 1.

Таблица 1

Метод разрешения коллизии	Средняя сложность решения		Средний перерасход памяти
	Задача о близости	Интервальный поиск	
Линейное зондирование	$O(m)$	$O(m)$	$O(m-n)$
Двойное хеширование	$O(m)$	$O(m)$	$O(m-n)$
Цепочки переполнения	$O(l+k)$	$O\left(\left(\frac{c}{k}+1\right)\cdot(l+k)\right)$	$O\left(m-\frac{n}{k}\right)$
Бинарные Деревья	$O(l+\log(k))$	$O\left(\left(\frac{c}{k}+1\right)\cdot(l+\log(k))\right)$	$O\left(m-\frac{n}{k}\right)$
Вторичные хеш-таблицы	$O(l)$	$O\left(\left(\frac{c}{k}+1\right)\cdot(l+m_k)\right)$	$O(m)$

Предлагаемая структура данных для хеш-таблицы

Базовым элементом предлагаемой структуры является классический узел двусвязных списков. Он представляет собой объединение хранимой записи и двух указателей на такие же узлы. Произвольный узел a_i формализуем в виде

$$a_i = <l_i, y_i, r_i>, \quad (5)$$

где l_i, и r_i – указатели; y_i – запись – элемент множества Y.

Так как указатели позволяют реализовать связи между узлами структуры данных, образуя, по сути, некоторый граф, будем в дальнейшем называть узлы вида (5) вершинами.

Предлагаемая структура состоит из связанных между собой посредством указателей вершин вида (5). Построим массив A_m из данных вершин и назовём эти вершины *базовыми*. Остальные, не принадлежащие A_m вершины будем называть *дополнительными.*

Рис. 1. Условные обозначения вершин разных видов

Базовую активную вершину a_i назовём *индексной*, если l_i = *NULL* – специальное значение, которое свидетельствует о том, что в указателе не содержится адреса объекта, то есть

l_i ни на что не указывает. В противном случае базовую активную вершину a_i назовём *списочной.* Списочная вершина представляет собой первый элемент циклического двусвязного списка. В предлагаемой структуре данных списочные вершины будут пересечениями циклических двусвязных списков и индексного массива A_m.

Для удобства представления предлагаемой структуры данных в виде графа введём некоторые обозначения. На рис. 1 изображены условные обозначения разных видов вершин.

Пусть a_i – некоторая активная вершина, а a_j – соседняя справа активная вершина. Вершину a_i и все пассивные вершины A_m, расположенные между a_i и a_j назовём *подмассивом* вершины a_i. Возможно, что две активные вершины расположены непосредственно рядом друг с другом. Тогда считаем что подмассив левой активной вершины пустой. Пример пустого и непустого подмассивов изображены на рис. 2.

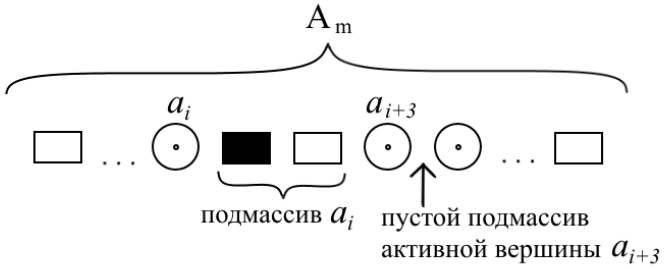


Рис. 2. Виды подмассивов

В качестве базовой структуры для хеш-таблицы в данной работе предлагается граф с вершинами, представленными на рис. 1 и обладающий следующими свойствами:

- любая пассивная вершина a_i с помощью l_i указывает на ближайшую левую активную, а с помощью r_i на ближайшую правую активную;
- в случае отсутствия коллизий дополнительные вершины в структуре данных отсутствуют, все пассивные вершины – пустые, активные – индексные и с помощью r_i указывают на самих себя;
- в случае возникновения коллизий, происходят изменения на основании алгоритма A, который будет рассмотрен в данной работе далее.

Отметим, что любая хеш-таблица имеет период работы без коллизий, к примеру, при добавлении одного элемента, поэтому задание структуры данных с помощью перечисленных трёх свойств корректно.

Кроме того, очевидно, что для того, чтобы выполнялись условия – необходимо наличие у каждой пассивной вершины левой и правой активных вершин. Поэтому, при практической реализации хеш-таблицы рекомендуется искусственно добавить два объекта – активные вершины – в начало и конец A_m. Такая практика, кроме того, что позволит выполнить требуемые условия, ещё и удобна с точки зрения выявления первого и последнего элементов структуры данных.

Пример предлагаемой структуры данных до появления коллизий представлен на рис. 3.

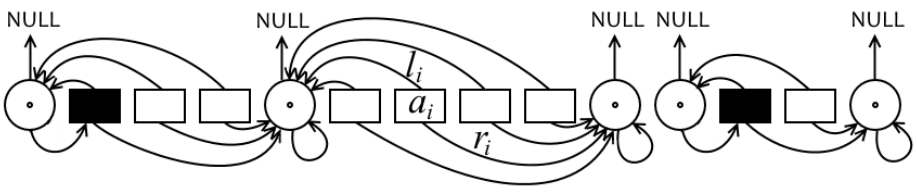


Рис. 3. Пример предлагаемой структуры данных до появления коллизий

Для удобства визуализации предлагаемой структуры посредствам графов указатели l_i произвольной базовой вершины a_i будут формализованы стрелками, располагающимися над последовательностью вершин A_m, а указатели r_i стрелками, находящимися под ней.

Рассмотрим вспомогательную процедуру, которая по индексу вершины определяет её тип. Пусть стоит задача найти тип некоторой вершины $a_i = <l_i, y_i, r_i>$. Представим алгоритм данной процедуры.

Шаг 1. Проверить содержит ли l_i константу *NULL*. Если содержит, то a_i – индексная вершина и конец алгоритма.

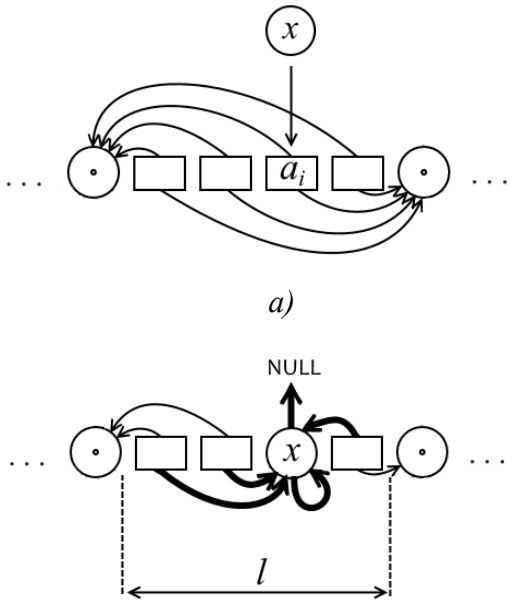
Шаг 2. Проверить указывает ли l_i на базовый элемент. Если указывает, то a_i – пассивная вершина. В противном случае a_i – списочная вершина. Конец алгоритма..

Для удобства назовём процедуру определения типа вершины на основе рассмотренного алгоритма *H*.

Отметим, что процедуры *H* достаточно для определения пустой вершины. Действительно, пустая вершина по определению является пассивной и не содержит записи. С помощью процедуры *H* определяем, что вершина пассивная и проверяем: есть ли в ней запись. На практике хеш-таблицы реализуются таким образом, что всегда можно определить, содержит ли вершина запись. Это может быть реализовано с помощью специальной зарезервированной записи, означающей отсутствие настоящей, либо использованием в качестве y_i указателей или ссылок.

Кроме того, важно отметить, что шаг 2 осуществляется за константное время. Действительно, если у нас есть массив, значит, мы имеем адрес его первого элемента и размер массива. Эти величины и позволяют составить неравенство, определяющее является ли рассматриваемая вершина базовой, то есть, по определению, принадлежащей массиву.

В дальнейшем для простоты будет указано, что при необходимости определения типа вершины в различных алгоритмах будет использоваться процедура *H*. Вместе с тем, очевидно, что до внесения каких-либо изменений в отношении любой вершины процедура *H* должна быть применена единожды. В рамках парадигмы объектно-ориентированного программирования это реализуется крайне просто. Рассматриваемая процедура реализована в виде метода итератора, который кэширует полученное значение и при повторном вызове не проводит аналогичные действия, а просто возвращает ранее сохранённый тип вершины.



а)

б)

Рис. 4. Добавление нового элемента в структуру данных, не вызывающее коллизию:
а – изначальный вид, необходимо добавить *x* в a_i;
б – вид после добавления, жирными линиями выделены новые связи

Рассмотрим процесс добавления записи в вершину в случае отсутствия коллизии. Очевидно, что не вызвать коллизию может только добавление в пустую вершину. Данный процесс представлен на рис. 4.

Изменение структуры графа, представленное на рис. 4, с алгоритмической точки зрения тривиально и имеет асимптотическую сложность $O(l)$. Действительно, пустая вершина является пассивной, а значит, согласно свойствам исследуемого графа, содержит в себе указатели на ближайшие левую и правую активные вершины. А для представленного на рис. 4 изменения структуры графа достаточно пройти по массиву между двумя этими активными вершинами и соответствующим образом поменять связи. Средняя асимптотическая сложность вытекает из средней величины массива между двумя активными вершинами, которая ранее была обозначена l.

Алгоритмы разрешения коллизий на предлагаемой структуре данных

Рассмотрим теперь задачу разрешения коллизий на предлагаемой структуре данных. В общем виде алгоритм разрешения коллизий A, заявленный в свойствах рассматриваемого графа, состоит из двух вспомогательных алгоритмов: A_1 и A_2. Алгоритм A_1 используется для разрешения коллизий при попадании новой записи в активную вершину, а алгоритм A_2 решает аналогичную задачу, при попадании новой записи в заполненную вершину. Алгоритм A заключается в процедуре H и запуске алгоритма A_1, если по итогам H выяснилось, что вершина активная (индексная или списочная), либо запуске алгоритма A_2, если вершина пассивная.

На данном этапе рассмотрения предлагаемой структуры данных может сложиться неверное мнение, что для неё множества пустых и пассивных вершин совпадает. На самом деле заполненные пассивные вершины появляются в результате коллизий на активных вершинах. Появление заполненных пассивных вершин, в свою очередь, приводит к появлению на них коллизий. Поэтому целесообразно рассмотреть сначала разрешение коллизий на активных вершинах (алгоритм A_1), а уже потом исследовать аналогичные алгоритмы на пассивных вершинах (алгоритм A_2).

Для начала наглядно представим алгоритм разрешения коллизий A_1. На рис. 5 представлены различные случаи разрешения коллизий в предлагаемом графе с помощью алгоритма A_1. Для предотвращения загромождения иллюстрации лишними построениями показаны связи только активных вершин.

Формализуем строго алгоритм A_1. Пусть a_i – активная вершина, в которую по результатам работы хеш-функции необходимо поместить некоторый объект x. Рассмотрим алгоритм A_1 по разрешению коллизии в вершине a_i.

Шаг 1. Если подмассив a_i не содержит пустые вершины перейти к шагу 3.

Шаг 2. Добавить x в подмассив a_i с сохранением порядка элементов в нём. В качестве новой вершины взять первую пустую, следующую за непустыми вершинами подмассива. Сдвинуть в A_m указатель r_i на одну вершину вправо, если a_i –индексная вершина. Конец алгоритма.

Шаг 3. Присвоить значение последней вершины подмассива a_i в переменную y. Если $x>y$, то поменять их местами. Пометить последнюю вершину подмассива a_i как пустую. Сдвинуть в A_m указатель r_i на одну вершину влево, если a_i – индексная вершина.

Шаг 4. Если вершина a_i индексная, то преобразовать её в списочную.

Шаг 5. Добавить y в циклический список при списочной вершине a_i, с сохранением упорядоченности элементов списка и перейти к шагу 2.

По сути, алгоритм разрешения коллизий A_1 предлагает сохранять все элементы коллизионного домена в подмассиве соответствующей активной вершины в упорядоченном виде. А когда пустые вершины подмассива заканчиваются – активная вершина из индексной преобразуется в списочную, представляющую собой начало двусвязного циклического списка. При этом новые элементы коллизионного домена добавляются в упорядоченном виде уже в этот список.

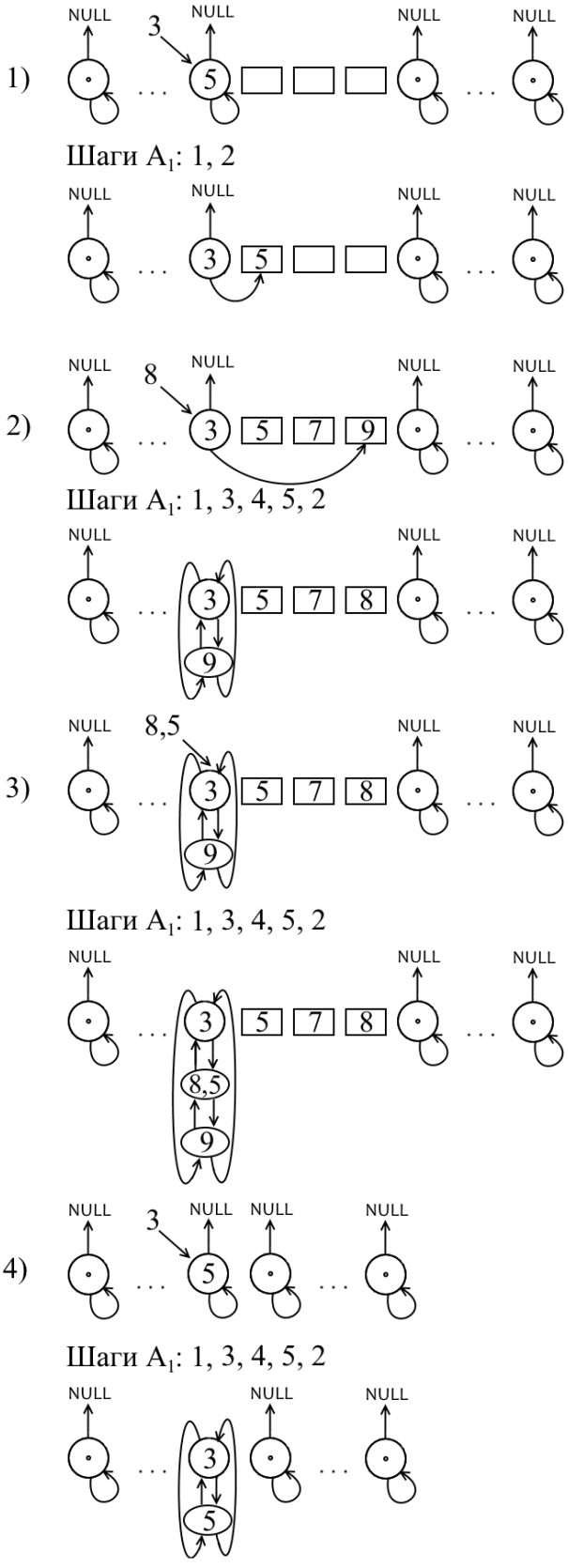


Рис. 5. Примеры разрешения коллизий с помощью алгоритма A_1

По итогам рассмотрения A_1 становится понятным, что предлагаемая структура данных представляет собой индексный массив, пересекающийся в списочных вершинах с циклическим двунаправленным списком.

Рассмотрим подробнее алгоритм A_1, пояснив, возможно неочевидные, операции и проведя оценку его асимптотической сложности.

Пусть a_j – ближайшая справа к a_i активная вершина. Рассмотрим вспомогательное утверждение о том, что зная a_i легко найти и a_j (здесь и далее под понятиями «знать вершину» и «найти вершину» будут пониматься соответствующие операции с её адресом). Действительно, для этого достаточно применить процедуру H к стоящей в A_m справа от a_i вершине. Если эта вершина активная, то она и есть a_j, в противном случае данная вершина является пассивной, а, значит, её правый указатель r_{i+1} содержит адрес a_j.

На шаге 1 осуществляется проверка наличия непустых вершин в подмассиве a_i. Из алгоритма A_1 следует, что, если вершина a_i индексная, то r_i – её правый указатель – содержит адрес последней заполненной вершины подмассива a_i. Поэтому достаточно проверить следующую вершину в A_m за последней заполненной. Если эта вершина a_j, то подмассив a_i не содержит пустых элементов. В том случае, если a_i – списочная вершина, то также очевидно, что её подмассив не содержит пустых элементов. Заметим, что процедура H, поиск ближайшей справа активной вершины и все операции шага 1 происходят за константное время.

На шаге 2 производится операция добавления элемента в упорядоченный массив с сохранением порядка. Примем среднее количество заполненных вершин в подмассиве p, тогда асимптотическая сложность данной операции равна $O(p)$. Отметим, что $p \le l$.

Все операции шага 3 стандартны и выполняются за константное время. Тем не менее, может показаться неочевидным то, что известна последняя вершина подмассива вершины a_i. Вместе с тем, переход к шагу 3 осуществляется, только если в подмассиве нет пустых ячеек. А это значит, что последняя ячейка подмассива вершины a_i находится слева в A_m от вершины a_j.

Шаги 4 и 5 содержат единственную операцию с неконстантной асимптотической сложностью – добавление элемента в упорядоченный список с сохранением порядка. Если положить среднюю длину списка s, то такая операция асимптотически оценивается как $O(s)$.

На основе проведённого анализа алгоритма A_1, легко видеть, что в нём присутствуют только две операции, осуществляемые за неконстантное время. Это добавление элемента в список, имеющее сложность $O(s)$ и добавление элемента в массив со сложностью $O(p)$. Таким образом, асимптотическая сложность рассматриваемого алгоритма (T_{A_1}) имеет вид: $T_{A_1} = O(p) + O(s) = O(p+s) = O(k)$.

Рассмотрим второй алгоритм разрешения коллизий – A_2, используемый, в случае если коллизия возникла в пассивной заполненной вершине.

Пусть a_i – пассивная заполненная вершина, в которую по результатам работы хеш-функции необходимо поместить некоторый объект x. Отметим, что a_i входит в подмассив ближайшей левой активной вершины a_{left}. Данный подмассив ограничен ближайшей активной вершиной справа как к a_i, так и к a_{left}. Назовём эту вершину a_{right}. Согласно второму свойству предлагаемой структуры данных именно на a_{left} и a_{right}. указывают l_i и r_i соответственно. Ранее мы показали, как, зная a_{left} и a_{right},найти первую пустую вершину или показать, что таковых, в исследуемом подмассиве, нет. Нахождение первой пустой вершины, означает нахождение последней заполненной, так как она, по факту, является предыдущей. Назовём последнюю заполненную вершину рассматриваемого подмассива a_{end}. Так как a_{left}, a_{right} и $a_{end} \in A_m$ и наименования *left*, *right* и *end* до сих пор были связаны только с семантикой и никак не привязаны к месторасположению вершин, положим без ограничения общности, что $a_{left} = A_m[left]$, $a_{right} = A_m[right]$ и $a_{end} = A_m[end]$.

Рассмотрим алгоритм A_2 по разрешению коллизии в вершине a_i.

Шаг 1. Находим *left*, *right* и *end*.

Шаг 2. Создаём связный список *tmp* вершин вида (5) из *end* – *left* + 1 элементов.

Шаг 3. Проходим параллельно массив A_m, начиная с $A_m[left]$ и заканчивая $A_m[end]$, и список *tmp*, присваивая соответствующий элемент массива списку и помечая элементы массива как пустые.

Шаг 4. Если вершина a_{left} индексная, то преобразовать её в списочную.

Шаг 5. Вставить *tmp* в циклический двусвязный список при вершине a_{left}.

Шаг 6. Провести процедуру вставки объекта *x* без возникновения коллизии. Конец алгоритма.

На рис. 6 изображён пример разрешения коллизий в предлагаемой структуре данных с помощью алгоритма A_2. Для предотвращения загромождения иллюстрации лишними построениями показаны связи только активных вершин.

Заметим, что асимптотическая сложность всех операций кроме шагов 3 и 6 $O(1)$. Так, значения вершин списка *tmp* после копирования их из массива упорядочены и наибольшая из них меньше наименьшего значения в циклическом списке при a_{left}, если такой существовал.

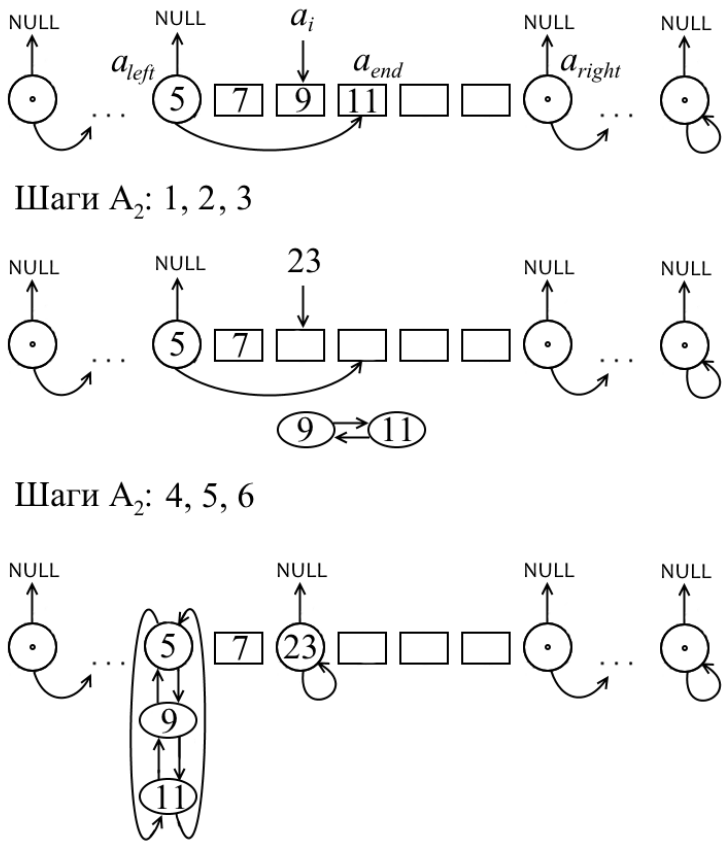


Рис. 6. Разрешение коллизий на основе алгоритма A_2 в случае, когда a_{left} – индексная вершина

Это неравенство имеет место из-за способа заполнения подмассива и списка при вершине a_{left}. Поэтому вставка списка *tmp* в циклический двусвязный список осуществляется путём перераспределения нескольких связей, а значит за константное время.

Асимптотическая сложность выполнения шага 3 равна $O(l)$, так как, по своей сути, рассматриваемый этап алгоритма представляет собой проход по подмассиву некоторой активной вершины, со средней длиной, равной *l*.

Заметим, что на рис. 6 изображён пример разрешения коллизий с помощью алгоритма A_2 только в случае, когда коллизия произошла в подмассиве индексной вершины. На рис. 7 изображён пример разрешения коллизии с помощью алгоритма A_2, при условии, что a_{left} – списочная вершина.

Отметим, что шаг 6 не продемонстрирован на рис. 6 и рис. 7 в силу того, что мы отказались от построения связей пассивных вершин. Вместе с тем, изменения структуры данных на шаге 6 алгоритма A_2 продемонстрированы на рис. 4 и оценка асимптотической сложности данной процедуры $O(l)$.

Таким образом, асимптотическая сложность рассматриваемого алгоритма $\left(T_{A_2}\right)$ имеет вид: $T_{A_2} = O(1)+O(1)+O(l)+O(1)+O(1)+O(l) = O(l)$.

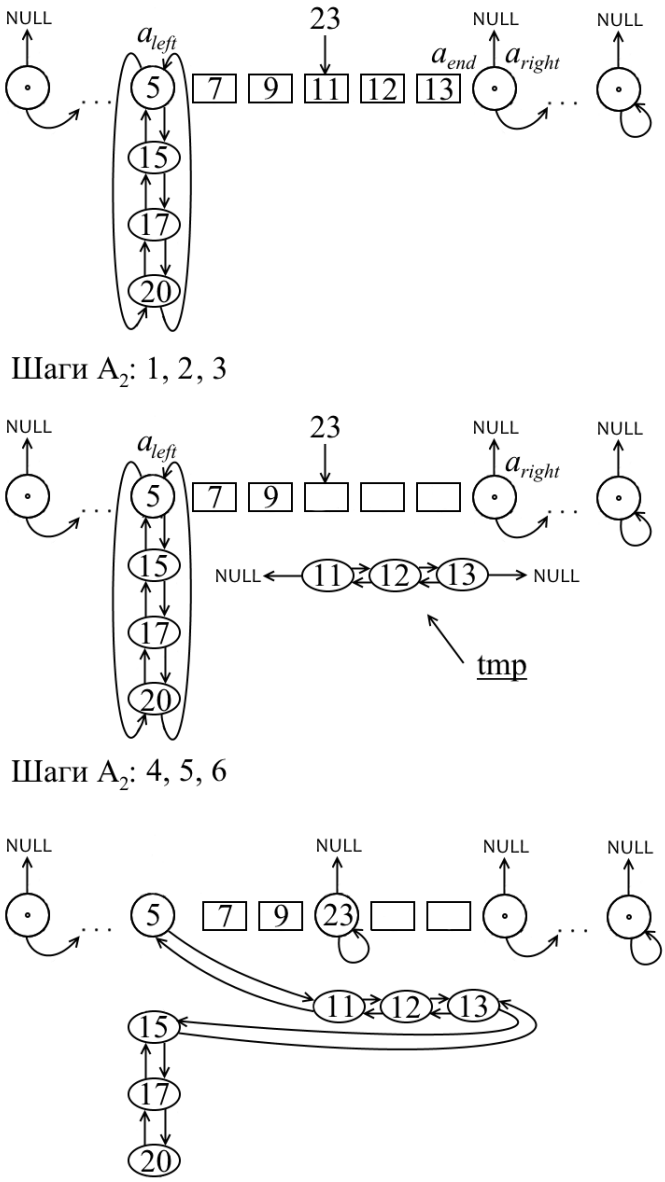


Рис. 7. Разрешение коллизий с помощью алгоритма A_2 в случае, когда a_{left} – списочная вершина

Очевидно, что для оценки асимптотической сложности алгоритма A, представляющего собой выполнение одного из алгоритмов A_1 или A_2, необходимо определить корреляцию между l и k, а также найти вероятность возникновения коллизии в активной и пассивной вершинах. Это возможно лишь при условии выбора конкретного вида h_m, поэтому в данной статье ограничимся оценкой сложности алгоритма разрешения коллизий $\left(T_A\right)$ в виде $T_A = O(\max\{k,l\})$.

Решение задач информационного поиска на предлагаемой структуре данных

Рассмотрим алгоритм поиска идентичных объектов на исследуемом графе. Пусть имеем задачу информационного поиска вида (1). К хеш-таблице был послан запрос $x \in X$. Найдём запись y из хеш-таблице, идентичную x.

Шаг 1. Вычисляем $i = h_m(x)$. Осуществляем процедуру H для вершины $A_m[i]$. Если $A_m[i]$ – пассивная вершина, то искомой записи в таблице нет и конец алгоритма.

Шаг 2. Если вершина $A_m[i]$ индексная, то переходим к шагу 5.

Шаг 3. Проверяем: лежит ли x в отрезке, задаваемом первым и последним элементом списка. Если x больше максимального элемента списка, то искомой записи в таблице нет и конец алгоритма, а если меньше минимального, то переходим к шагу 5.

Шаг 4. Производим поиск по двусвязному циклическому списку при вершине $A_m[i]$. В результате объект может быть найден или не найден. Результат считается результатом поиска x в Y. Конец алгоритма.

Шаг 5. Производим поиск по индексному массиву – части состоящей из заполненных элементов подмассива $A_m[i]$. Результат считается результатом поиска x в Y. Конец алгоритма.

Единственные шаги рассматриваемого алгоритма, осуществляющиеся за неконстантное время – это шаги 4 и 5. Заметим, что в результате работы алгоритма выполняется только один из этих шагов. Число вершин циклических двусвязных списков в предлагаемой структуре данных в среднем было обозначено s, тогда асимптотическая сложность шага 5 равна $O(s)$. Среднее количество заполненных вершин в подмассиве было обозначено p, тогда асимптотическая сложность шага 5 равна $O(log(p))$.

В среднем время поиска на хеш-таблице константно. Это происходит благодаря тому, что при выборе хорошей хеш-функции мощность абсолютного большинства коллизионных доменов равна 1. Это означало бы, что $s = 0$, а $p = k = 1$.

Вместе с тем, если сравнить алгоритм поиска в предлагаемой структуре данных с алгоритмами поиска при линейном зондировании, двойном хешировании, цепочками переполнения лишь в случаях возникновения коллизий, то становится ясно, что названные виды хеш-таблиц имеют сложность поиска искомой вершины в коллизионном домене со сложностью как минимум $O(k)$. В то время как предлагаемая структура обеспечивает частично логарифмический поиск по коллизионному домену. Если же разрешение коллизий происходит на основе бинарных деревьев и дополнительных хеш-таблиц, то сложность поиска в коллизионном домене будет логарифмической и константной соответственно. Эти показатели лучше, чем у предлагаемой структуры данных.

Заметим, что вместо циклических двусвязных списков можно использовать бинарные деревья, составленные из тех же узлов вида (5). При таком построении скорость поиска не будет падать ниже логарифмической относительно мощности коллизионного домена. Однако подобная модернизация усложнит процедуру разрешения коллизий в пассивных вершинах. Например, в случае, изображённом на последнем графе (рис. 7), осуществлённая там операция добавления списка была бы гораздо сложнее. Также сложнее были бы операции обхода хеш-таблицы. С другой стороны, замена циклического двусвязного списка на дерево может существенно ускорить все операции поиска, на хранимой в списках части коллизионного домена. В дальнейших работах планируется исследование аналогичных структур данных, использующих вместо циклического двусвязного списка различного рода бинарные деревья. А в данной работе ограничимся тем, что выбор хорошей хеш-функции приведёт к отсутствию слишком длинных списков.

Рассмотрим теперь решение задачи о близости на предлагаемой структуре данных. Пусть имеем задачу информационного поиска вида (1). К хеш-таблице был послан запрос $x \epsilon X$. Требуется найти запись, ближайшую к запросу.

Шаг 1. Вычисляем $i = h_m(x)$. Осуществляем процедуру H для вершины $A_m[i]$. Если $A_m[i]$ – пассивная вершина, то переходим к шагу 6.

Шаг 2. Если x меньше минимального элемента коллизионного домена $A_m[i]$, то переходим к шагу 7. А если x больше максимального элемента коллизионного домена $A_m[i]$, то переходим к шагу 8.

Шаг 2. Если вершина $A_m[i]$ индексная, то переходим к шагу 4.

Шаг 3. Осуществляем решение задачи о близости на циклическом двусвязном упорядоченном списке при вершине $A_m[i]$.

Шаг 4. Осуществляем решение задачи о близости на индексном массиве – части состоящей из заполненных элементов подмассива $A_m[i]$.

Шаг 5. Выбираем ближайший элемент к x из полученных в результате шага 3, если он имел место, и шага 4. Он и является решением задачи о близости. Конец алгоритма.

Шаг 6. Находим ближайшие к $A_m[i]$ слева и справа активные вершины: $A_m[left]$ и $A_m[right]$ соответственно. Сравниваем максимальный элемент коллизионного домена $A_m[left]$ и минимальный $A_m[right]$ по критерию близости с x. Ближайший из них и есть решение поставленной задачи о близости. Конец алгоритма.

Шаг 7. Находим ближайшую слева активную вершину $A_m[left]$. Решением задачи будет ближайший к x элемент из максимального элемента $A_m[left]$ и минимального элемента $A_m[i]$. Конец алгоритма.

Шаг 8. Находим ближайшую справа активную вершину $A_m[right]$. Решением задачи будет ближайший к x элемент из минимального элемента $A_m[right]$ и максимального элемента $A_m[i]$. Конец алгоритма.

Рассматриваемый алгоритм решения задачи о близости оперирует понятиями минимальный и максимальный элементы коллизионного домена активной вершины. Существование данных элементов определяется тем, что на основе определений активной вершины и коллизионного домена, очевидно, что последний у активных вершин не пуст и содержит хотя бы один элемент (это, кстати, означает отсутствие коллизии как таковой).

Минимальный элемент коллизионного домена всегда содержится в вершине $A_m[i]$. А на максимальный указывает либо её правый указатель r_i, если $A_m[i]$ – индексная вершина, либо левый l_i, если $A_m[i]$ вершина списочная. Нахождение ближайших слева и справа активных вершин к известной рассматривалось ранее и осуществляется за O(1). Решение задач о близости на упорядоченных списке и массиве, осуществляемое на 3-м и 4-м шагах соответственно общеизвестны. Асимптотическая сложность 3-го и 4-го шагов $O(s)$ и $O(log(p))$ соответственно.

Опираясь на оценки, представленные в табл. 1, можно утверждать, что решение задачи о близости на предлагаемой структуре обладает меньшей асимптотической сложностью, чем в хеш-таблицах с линейным зондированием, двойным хешированием и цепочками переполнения. Что же касается хеш-таблиц осуществляющих разрешение коллизий путём организации коллизионного домена с помощью бинарных деревьев и дополнительных хеш-таблиц, то на данном этапе исследования можно сказать, что скорости решения задачи о близости сравнимы. Более точное сравнение возможно лишь с учётом характеристик h_m и планируется в дальнейших исследованиях.

Решение задачи интервального поиска представляет собой решение двух задач о близости, позволяющих найти границы интервала и проход по записям между левой и правой границами найденного интервала. Отметим, что предлагаемая структура данных позволяет пройти по записям, минуя пустые вершины. Действительно, ранее было показано, что для любой ячейки A_m можно за константное время найти левую соседнюю активную вершину, которая связана с упорядоченным списком, если таковой присутствует в её коллизионном домене, и подмассивом, конец непустых ячеек которого также легко найти за константное время. Кроме того, известно:

- для каждой пары соседних активных вершин все элементы коллизионного домена левой меньше элементов коллизионного домена правой;
- все записи, хранящиеся в списке активной вершины больше записей хранящихся в её подмассиве;

- все записи, хранящиеся в списке и в подмассиве активной вершины, упорядочены.

Так как алгоритмы последовательного прохождения индексных массивов и двусвязных списков в обе стороны тривиальны, то и алгоритм обхода интервала в любую сторону очевиден. Если положить асимптотическую сложность решения задачи о близости за T, то та же сложность решения задачи интервального поиска определяется как $T + c$, где c – мощностная нижняя оценка сложности решения задачи интервального поиска, определяемая количеством записей в искомом интервале. Такое выражение для сложности означает, что понизить её могут лишь ассоциативные массивы с лучшими характеристиками решения задачи о близости.

В сравнении с другими хеш-таблицами, асимптотическая сложность интервального поиска в которых представлена в табл. 1, решение задачи о близости лучше только у ассоциативного массива, использующего для разрешении коллизий дополнительные хеш-таблицы. Вместе с тем, подобный подход приводит к тому, что перечисление всех записей в интервале предполагает не только обход пустых ячеек A_m, но и пустых ячеек базовых массивов всех дополнительных хеш-функций, разрешающих коллизии внутри искомого интервала. Кроме того, предполагается, что все дополнительные хеш-функции также обладают свойством монотонности, поэтому при хорошей h_m и малых размерах списков в предлагаемой структуре данных, она показывает лучшие показатели на интервальном поиске.

Выводы

В статье предлагается структура данных для использования в качестве базовой при проектировании хеш-таблиц. Эта структура состоит из однотипных элементов вида (5), и представляет собой индексный массив, пересекающийся с двусвязными циклическими списками. При этом элементы самого массива связаны друг с другом и с узлами списков определённым образом.

Показано, что в случае задачи поиска идентичных объектов при условии, данный объект составляет коллизию с несколькими хранимыми записями, предлагаемая структура показывает более низкую асимптотическую сложность, чем хеш-таблицы с линейным зондированием, двойным хешированием и цепочками переполнения, но более высокую чем хеш-таблицы с методами разрешения коллизий, основанными на построениях бинарных деревьев и дополнительных хеш-таблиц.

При решении задачи о близости лучшие показатели, в сравнении с предлагаемой структурой данных, наблюдаются только при использовании метода разрешения коллизий путём использования дополнительных хеш-таблиц.

При переходе к задачам интервального поиска предлагаемая структура данных составляет конкуренцию и последнему методу. Это происходит из-за того, что метод использования дополнительных хеш-таблиц для разрешения коллизий приводит к выделению большого объёма неиспользованной памяти, которую приходится обходить при интервальном поиске. Вместе с тем, тот же поиск в хеш-таблицах, основанных на предлагаемой структуре, не предполагает обхода пустых ячеек памяти.

Если сравнивать по критерию перерасхода памяти, то ситуация обратная.

Наибольший перерасход и соответственно худшие показатели демонстрирует метод разрешения коллизий путём дополнительных хеш-таблиц. Это происходит, так как кроме пустых ячеек в A_m появляются пустые ячейки и в дополнительных хеш-таблицах, разрешающих коллизии.

За ним следуют методы цепочек переполнения и использования бинарных деревьев для разрешения коллизий. Эти методы содержат весь коллизионный домен в дополнительной памяти.

Предложенная в данной работе структура содержит часть коллизионного домена в дополнительной памяти и часть в самом массиве A_m и поэтому имеет лучшие показатели по использованию памяти в сравнении с методами разрешения коллизий, основанными на цепочках переполнения и бинарных деревьях.

Лучшими по критериям использования памяти являются хеш-таблицы, использующие такие методы разрешения коллизий, как линейное зондирование и двойное хеширование: они хранят все объекты коллизионного домена в ячейках массива A_m, но, как было показано раньше, абсолютно не приспособлены для решения задач о близости и интервального поиска.

Таким образом, можно сделать предварительный вывод о том, что предложенную в статье структуру данных целесообразно использовать при необходимости часто осуществлять интервальный поиск и, в меньшей степени, решать задачу о близости в условиях дефицита памяти.

В дальнейших исследованиях планируется оценка асимптотической сложности решения базовых поисковых задач предложенной структурой данных с группой современных монотонных хеш-функций. Также необходимо рассмотреть сложность аналогичной структуры данных, использующей вместо циклических двусвязных списков бинарные деревья. А также возможную гибридизацию этих структур, которая, с одной стороны, не влечёт ухудшение в среднем сложности разрешения коллизий на пассивных вершинах и интервального поиска, а с другой, не позволяет образовывать длинные списки, преобразуя их в бинарные деревья.

Кроме того, предполагается провести вычислительные эксперименты, подтверждающие теоретические оценки на практике и более строго очерчивающие границы на множестве задач информационного поиска, при которых предложенная структура данных показывает лучшие результаты, чем другие известные хеш-таблицы.

Библиографический список

1. **Гасанов, Э.Э.** Теория хранения и поиска информации / Э.Э. Гасанов, В.Б. Кудрявцев. – М.: ФИЗМАТЛИТ, 2002. – 288 с.
2. ГОСТ по пертинентности и релевантности. Система стандартов по информации, библиотечному и издательскому делу. Поиск и распространение информации. Термины и определения. - Взамен ГОСТ 7.27-80; введ. 31.03.1997. – Минск, 2001. – Режим доступа: http://www.docload.ru/Basesdoc/6/6316.
3. **Ландэ, Д.В.** Интернетика: Навигация в сложных сетях: модели и алгоритмы / Д.В. Ландэ, А.А. Санарский, И.В. Безсуднов. – М.: ЛИБРОКОМ, 2009. – 264 с.
4. **Белобродский, А.В.** Об одном способе поиска релевантных векторов. / А.В. Белобродский, В.Н. Решетников // Вопросы оптимизации и управления. – М.: Изд-во Моск. ун-та, 1979. – С. 59–63.
5. **Robertson, S.E.** Simple, proven approaches to text retrieval / S.E. Robertson, K.S. Jones // Cambridge Technical Report. – 1997. – С. 21–23.
6. **Поляков, Д.В.** Нечеткий подход к определению пертинентности результатов поиска и выбору оптимального запроса / Ю.Ю. Громов, Д.В. Поляков, О.Г. Иванова, В.Е. Дидрих // Вестник Воронежского института ФСИН России. г. Воронеж. ООО ИПЦ «Научная книга». – 2011. – №2 – С. 49–55.
7. **Поляков, Д.В.** Использование математического аппарата нечеткой логики для определения пертинентности результатов поиска текстовых сведений / Д.В. Полякови [и др.] // Математические методы и информационно-технические средства: Труды VIII Всероссийской научно-практической конференции, Краснодарский университет МВД России. – 2012. – С. 163.

8. **Поляков, Д.В.** Построение пертинентного запроса к информационно-поисковой машине на основе математического аппарата нечеткой логики / Д.В. Поляков [и др.] // Математические методы и информационно-технические средства: Труды VIII Всероссийской научно-практической конференции, Краснодарский университет МВД России. – 2012. – С. 167.
9. **Поляков, Д.В.** Определение пертинентности результатов запроса с ис-пользованием нечеткой логики / Д.В. Поляков [и др.] // Приборы и системы. Управление, контроль, диагностика. – 2012. – №3 – С. 29–33.
10. **Поляков, Д.В.** Формализация информационной потребности с помощью коллокаций на основе теории нечётких множеств для пертинентного поиска текстовых сведений / Д.В. Поляков [и др.] // Информация и безопасность. Воронеж: Издательско-полиграфический центр Воронежского государственного университета. – 2012. – Т. 15. – №2 – С. 213–-218.
11. **Salton, G.** Extended Boolean information retrieval / G. Salton, E. Fox, H. Wu // Communications of the ACM. – 2001. – V. 26. № 4. – С. 35–43.
12. **Salton, G.** A Vector Space Model for Automatic Indexing / G. Salton, A. Wong, C. Yang. // Communications of the ACM. – 1975. – С. 613–620.
13. **Salton, G.** Selective Text Traversal / G. Salton, A. Singhal. – Department of Computer Science, Cornell University, Ithaca – 1995. – С. 131–144.
14. **Salton, G.** Automatic Information Retrieval / G. Salton. – Cornell University. – 1980. – С. 41–54.
15. **Ермаков, А.Е.** Компьютерная лингвистика и интеллектуальные технологии: труды Международной конференции Диалог'2008 // Автоматизация онтологического инжиниринга извлечения знаний из текста. – М., 2008.
16. **Пивоварова, Л.М.** Извлечение и классификация терминологических коллокаций на материале лингвистических научных текстов (предварительные наблюдения) / Л.М. Пивоварова, Е.В. Ягунова // Материалы симпозиума "Терминология и знание". – М., 2010.
17. **Захаров, В.П.** Анализ эффективности статистических методов выявления коллокаций в текстах на русском языке. [Электронный ресурс] / В.П. Захаров, М.В. Хохлова. – Санкт-Петербургский государственный университет, Институт лингвистических исследований РАН, Санкт-Петербург, 2010. – Режим доступа: http://www.dialog-21.ru/dialog2010/materials/html/22.htm.
18. **Савина, А.Н.** Исследование коллокаций с помощью экспериментов с информантами / А.Н. Савина, Е.В. Ягунова // Корпусная лингвистика–2011: труды международной конференции. – СПб.: С.-Петербургский гос. университет, Филологический факультет. – 2011. – С. 1–7.
19. **Недошивина, Е.В.** Учёт синтаксических связей при поиске коллокаций // Natural Language Processing. – 2008. – С. 1–3.
20. **Поляков, Д.В.** Формализация информационной потребности с помощью коллокаций на основе теории нечётких множеств для пертинентного поиска текстовых сведений / Д.В. Поляков, [и др.] // Информация и безопасность. Воронеж: Издательско-полиграфический центр Воронежского государственного университета. – 2012. – Т. 15. – №2 – С. 213–218.
21. **Поляков, Д.В.** Кластеризация текстовых коллекций на основе нечеткого описания коллокаций / О.Г. Иванова, Д.В. Поляков, А.Ю. Громова, В.Е. Дидрих // Информация и безопасность. – Воронеж: Издательско-полиграфический центр Воронежского государственного университета. – 2011. – №3 – С. 459–462.
22. **Адельсон-Вельский, Г.М.** Алгоритм организации информации / Г.М. Адельсон-Вельский, Е.М. Ландис // ДАН СССР – 1962. – Т. 146. – С. 263–266.
23. **Белобродский, А.В.** Об одном способе поиска релевантных векторов / А.В. Белобродский, В.Н. Решетников // Вопросы оптимизации и управления. – М.: Изд-во Моск. ун-та, 1979. – С. 59–63.
24. **Ben-Or, M.** Lower bounds for algebraic computation trees // Proc. Ibth ACM Annu. Symp. Theory Comput. –1983. – С. 80–86.
25. **Ferguson, D.E.** Fibonaccian searching // C ACM. – 1960. – V. 3. – С. 648.
26. **Yuval, G.** Finding nearest neighbours // Inform Processing Lett. – 1976. – V. 5. – С. 63–65.

27. **Fredman, M.L.** An algorithm for finding best match in logarithmic expected time / M.L. Fredman, J.L. Bentley, R.A. Finkel // ACM Trans. Math. Software. – 1977. – V. 3. – C. 209–226.
28. **Fredman, M.L.** An algorithm for finding nearest neighbors. / M.L. Fredman, F. Baskett, J. Shustek. // IEEE Trans. Comput. – 1975. – C. 1000–1006.
29. **Burkhard, W.A.** Some Approaches to best match file searching / W.A Burkhard, R.M. Keller // Commun. Ass. Comput. Mach. – 1973. – V. 16. – C. 230–236.
30. **Loftsgaarden, D.** A nonparametric density function / D.O. Loftsgaarden C.P. Queensberry. – Ann. Math. Stat. 36, 1965 – C. 1049–1051.
31. **Lauter, U.** 4-dimensional binary search trees as a means to speed up associative searches in design verification of integrated circuits // Journal of Design Automation and Fault Tolerant Computing, 2 1978. №3. – C. 241–247.
32. **Bentley, J.L.** Multidimensional binary search trees used for asso- associative searching / J.L. Bentley // Commun. Ass. Comput. Mach. – 1975. – V. 18. – C. 509–517.
33. **Bentley, J.L.** Data structures for range searching / J.L. Bentley, J.H. Friedman // Comput. Surveys. – 1979. – V. 11. – C. 397–409.
34. **Bentley, J.L.** Efficient worst-case data structures for range searching / J.L. Bentley, H.A. Maurer // Acta Inform. – 1980. – V. 13. – C. 155–168.
35. **Bentley, J.L.** A problem in multivariate statistics: Algorithms, data structure and applications / J.L. Bentley, M.I. Shamos // Proc. 15th Allerton Conf. Commun., Contr., Comput. – 1977. — C. 193-201.
36. **Bentley, J.L.** Analysis of range range searching in quad trees / J.L. Bentley, D.F. Stanat // Inform. Processing Lett. – 1975. – V. 3. – C. 170–173.
37. **Lee, D.T.** Worst case analysis for region and partial region searches in multidimensional binary search trees and bal- ansed quad trees / D.T. Lee, C.K. Wong. // Ada Informatica. – 1977. – V. 9. – C. 23–29.
38. **Lee, D.T.** Quintari trees: A file structures for multi- multidimensional database system / D.T. Lee, C.K. Wong // ACM Trans. Database Syst. – 1980. – C. 339–353.
39. **Кормен, Т.** Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн.– 2-е изд. – М.: Вильямс, 2011. – 1290 с.
40. **Кузнецов, С.Д.** Методы сортировки и поиска / С.Д. Кузнецов. – ИСП РАН, Центр информационных технологий Режим доступа: http://citforum.ru/programming/theory/sorting/sorting1.shtml.
41. **Кнут, Д. Э.** Искусство программирования. Т. 1. Основные Алгоритмы / Д. Э. Кнут. – 3-е изд. – М.: Вильямс, 2006. – 720 с.
42. **Belazzougui, D.** Monotone Minimal Perfect Hashing: Searching a Sorted Table with O(1) Accesses / D. Belazzougui, P. Boldi, R. Pagh, S. Vigna. –Proceedings of the 20th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA), New York, 2009. ACM Press.
43. **Belazzougui, D.** Theory and Practise of Monotone Minimal Perfect Hashing / D. Belazzougui, P.Boldi, R. Pagh, S. Vigna. – In Proceedings of the 11th Workshop on Algorithm Engineering and Experiments, ALENEX '09. Society for Industrial and Applied Mathematics, 2009.
44. **Drossos, L.** 2-D Monotone spatial indexing scheme with optimal update time / L. Drossos, S. Sioutas, K. Tsichlas, K. Ioannou. – WSEAS Int. Conf. on REMOTE SENSING, Venice, Italy, November 2-4. 2005. – P. 121–125.
45. **Поляков, Д.В.** Алгоритм поиска идентичных объектов на непрерывном множестве // Методы управления потоками в транспортных системах. – М: Изд-во МАДИ. – 2009, С. 114–121.
46. **Яковлев, А.В.** Оптимизационная задача построения отображения на адресное пространство для модели хранения данных с константным временем поиска / А.В. Яковлев [и др.]. – Воронеж: Приборы и системы. Управление, контроль, диагностика. 2013. №12. С. 36–41.
47. **Поляков, Д.В.** К вопросу о построении ассоциативных массивов для высокопроизводительных вычислений / Д.В. Поляков, А.И. Попов, В.А. Батуров // Информационные системы и технологии «ИСТ-2015»: материалы XXI Международной научно-технической конференции

– Нижний Новгород: Издательство Института радиоэлектроники и информационных технологий НГТУ. – 2015. – С. 269–270.

48. **Поляков, Д.В.** Генератор монотонных хеш-функций для ассоциативного массива / Д.В. Поляков, А.И. Попов // Труды НГТУ им. Р.Е. Алексеева. – Нижний Новгород. – 2015. №2 (109). – С. 70–82.

Дата поступления
в редакцию 22.10.2015

D.V. Polyakov, A.I. Popov, S.A. Duzkryatchenko

DATA STRUCTURES TO CREATE AN ASSOCIATIVE ARRAY BASED ON MONOTONE HASHING

Tambov state technical university

Purpose: Reducing the complexity of algorithms for solving problems of searching of the nearest object, searching of the interval, by developing of the data structure to create a hash table based on a monotone hash function.
Approach: The research is based on set theory, the theory of information retrieval and the theory of algorithms.
Findings: The structure of the data to create an associative array based on a monotone hash function and algorithms for this structure are suggested. Besides, provides a comparative analysis of the proposed and existing data structures for hash tables.
Research limitations: The research does not address problem of the choice of hash function, requiring only the its monotony. Also in this article is described the number of future researches, that needed for selection the parameters of the proposed structure and conditions of the re-hashing.
Originality / value: The proposed data structure and algorithms, working with it, show good results in solving problems of searching of the nearest object and searching of the interval, in comparison with other data structures, using in hash tables.

Key words: hash table, hash function, an associative array, the problem of searching for identical objects, the problem of searching of the nearest object, the problem of searching of the interval.