ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

УДК 519.72

Д.В. Поляков, А.И. Попов

ГЕНЕРАТОР МОНОТОННЫХ ХЕШ-ФУНКЦИЙ ДЛЯ АССОЦИАТИВНОГО МАССИВА

ФГБОУ ВПО «Тамбовский государственный технический университет»

Цель: Получение генератора хеш-функций с хорошими значениями асимптотических характеристик алгоритмов решения основных поисковых задач.
Подход: Методика исследований основана на теории множеств и теории информационного поиска.
Результаты: Представлен подход к генерации хеш-функций. Предложен метод получения хеш-функции на основе рассматриваемого генератора.
Ограничения исследований: Предложена и обоснована гипотеза о высоких характеристиках алгоритмов для решения основных поисковых задач при использовании хеш-функций, полученных с помощью предложенного метода. Ожидается, что дальнейшие исследования подтвердят гипотезу и позволят детально определить характеристики этих алгоритмов.
Оригинальность / значение: Предлагаемый генератор задаёт семейство хеш-функций для новых эффективных ассоциативных массивов, основанных на хеш-таблицах.

*Ключевые слова***:** хеш-функция, хеш-таблица, ассоциативный массив, минимальная хеш-функция, монотонная хеш-функция, совершенная хеш-функция, генерация хеш-функций, поиск идентичного значения, задача о близости, интервальный поиск.

Введение

В наше время информационные технологии играют в обществе одну из ключевых ролей. И их непрерывное развитие сопровождается ростом количества данных, которые необходимо обрабатывать. К примеру, осуществление мониторинга в различных отраслях человеческой деятельности приводит к накоплению огромных объёмов данных, нуждающихся в статистической, а иногда и интеллектуальной обработке. Решение задач мониторинга, составление планов и расписаний, автоматизация производства и процесса оказания услуг – всё это и многое другое предполагают хранение и обработку большого количества данных. В итоге, практически каждая современная информационная система содержит в себе базу данных.

Для работы с данными, порядок расположения которых не важен, используются так называемые ассоциативные массивы – абстрактные типы данных для хранения коллекций элементов, поддерживающие, как минимум, три операции: поиск элемента, добавление элемента и удаление элемента [1].

Способы организации ассоциативного массива разделяются на две основные группы: хеш-таблицы и бинарные деревья [1-2]. Основные показатели эффективности конкретного способа организации ассоциативного массива – это время, затрачиваемое на операции поиска, добавления и удаления элемента, а также расход памяти [1, 3].

Данные показатели зачастую[1-3] оцениваются не в абсолютных величинах, а с помощью, так называемой *O*-нотации, то есть асимптотической сложности относительно числа хранимых элементов. Асимптотическая сложность представляет собой оценку роста времени работы алгоритма при росте объёма входных данных. Эта мера хорошо себя зарекомендовала и стала классической для оценки сложности алгоритмов [1].

Вместе с тем, если рассматриваемые показатели носят случайный характер, то их целесообразно задавать двумя численными характеристиками: средним значением и значением в худшем случае.

Современные реалии требуют оценивать сложность решения не одной, а, как минимум, трёх поисковых задач: поиск идентичных объектов, решение задачи о близости и интервальный поиск [3].

Поиск элемента – тривиальная и очевидная задача, состоящая в поиске в ассоциативном массиве объекта, идентичного объекту-запросу. Решение задачи о близости – поиск ближайшего элемента к объекту-запросу. Обычно задача о близости появляется в связи с тем, что заданный объект в массиве не найден и возникает необходимость найти ближайший к нему объект. Интервальный поиск – это поиск набора хранимых элементов в интервале, который собственно и задаётся поисковым запросом. Такая задача распространена в теории баз данных, статистике [4] и автоматизации проектирования [5].

Основные определения и обозначения

В данной работе исследуются способы организации ассоциативного массива на основе хеш-функций. Такие массивы называются хеш-таблицами [1-3].

Хеш-таблицы представляют собой в некотором роде обобщение индексных массивов с их прямой адресацией к элементам [1]. Обращение к i-му элементу индексного массива (a_i) происходит за константное время благодаря тому, что вычисляется адрес данного элемента по формуле

$$\&a_i = a + iw, \qquad (1)$$

где a – адрес первого байта выделенной под массив области памяти, w – размер в байтах ячейки массива, а $\&a_i$ – адрес i-того элемента, то есть адрес a_i. Представленная формула становится очевидной, если вспомнить, что байт – минимальная адресуемая ячейка памяти, а все байты индексного массива по определению расположены по порядку. Таким образом, если элементы массива содержатся в *RAM* (памяти, позволяющей получить доступ к любой ячейке по её адресу, за одинаковое время), то вычисление адреса позволяет обратиться к соответствующему элементу за константное, то есть не зависящее от числа элементов в индексном массиве время. Такое время на языке O-нотации записывается как $O(1)$, что означает независимость времени обращения к элементу массива от его номера n.

Основная идея подхода к организации хеш-таблиц в том, чтобы выбрать некоторую функцию (хеш-функцию), которая будет вычислять адрес ячейки для хранения некоторого элемент x на основе самого x. Строго говоря, хеш-функция вычисляет не адрес элемента, а его индекс (i) в некотором индексном массиве, после чего адрес уже вычисляется по формуле (1).

Определим строго понятие хеш-функции для ассоциативного массива.

Пусть U – это множество элементов, для хранения которых предназначен ассоциативный массив. В самом общем случае U задаётся типом данных элементов, для хранения которых предназначена хеш-таблица. Вместе с тем, при решении конкретных задач U удобно ограничить. Например, для ассоциативного массива хранящего элементы четырёхбайтного целочисленного типа $U = \{ u \in N, -2^{31} \le u \le 2^{31} - 1 \}$, однако, если по условию поставленной задачи данный массив используется для хранения возраста людей, целесообразно ограничить U, задав его следующим образом: $U = \{ u \in N, 0 \le u \le 150 \}$.

Определим хеш-функцию как отображение h_m: $U \to \{ i \in N, i = \overline{0, m-1}, m \in N \}$.Создадим индексный массив из m элементов и обозначим его A_m. Будем считать, что индексы элементов данного массива – целые числа от 0 до m–1. Элемент массива A_m с индексом $i = \overline{0, m-1}$ будем обозначать $A_m[i]$. Вычисление адреса $A_m[i]$ будем осуществлять по формуле (1). Тогда хеш-таблицу для хранения элементов множества U зададим как $< h_m, A_m >$.

Очевидно, что добавление произвольного элемента $u \in U$ в хеш-таблицу на практике

реализуется присваиванием значения u элементу массива $A_m[h_m(u)]$, а поиск элемента $u \in U$ в хеш-таблице путём его сравнения с элементом, хранящимся в $A_m[h_m(u)]$. Так как если элемент $u \in U$ содержится в хеш-таблице, то он содержится в ячейке $A_m[h_m(u)]$, операция его удаления очевидно. Сложность таких операций определяется сложностью вычисления хеш-функции, а она асимптотически оценивается как $O(1)$, так как вычисление хеш-функции не зависит от числа элементов в массиве.

Таким образом, операции поиска, добавления и удаления элементов из хеш-таблицы оказываются гораздо быстрее, чем аналогичные в ассоциативных массивах, реализованных на основе бинарных деревьев и имеющих зачастую логарифмическую сложность. Вместе с тем, на практике при работе с заранее неизвестным набором объектов невозможно выбрать функцию h_m так, чтобы она не отображала никакие два элемента из U в один и тот же индекс. Такая ситуация называется коллизией. Существует множество методов разрешения коллизий, например, методы линейного зондирования, цепочек переполнения, универсальное хеширование и др. [1-2].

Другой неприятной ситуацией является, как правило, имеющаяся в A_m избыточная память. Впрочем, если таковая отсутствует, то любая операция вставки приведёт к коллизии.

Таким образом, избыточная память для хеш-таблицы является своеобразным краеугольным камнем. С одной стороны, если выделить много памяти, то это будет крайне расточительным, а с другой, если избыточной памяти мало, то число коллизий возрастёт, что пагубно скажется на скорости работы и в конечном итоге быстро приведёт к переполнению и как результату – рехешированию [1, 6] – процессу перераспределения памяти. Этот процесс обладает огромной трудоёмкостью по сравнению с алгоритмами поиска элементов, вставки, избегания коллизий. В контексте асимптотической сложности он оценивается как $O(n)$, где n – количество элементов в хеш-функции.

Вместе с тем константа при n крайне велика. То есть данный алгоритм при равных характеристиках вычислительных машин будет работать гораздо медленнее, чем многие другие с такой же асимптотической сложностью. Необходимость рехеширования возникает в рамках алгоритма добавления элемента и является вероятностным событием. Она неизбежно наступает для любых хеш-таблиц. Множество работ посвящено снижению этой вероятности [1-3, 6].

Для хеш-таблиц скорость поиска элемента при удачном выборе хеш-функции, равномерно рассеивающий поступающие объекты хранения, в среднем будет высока. Вместе с тем, большинство хеш-таблиц не способны эффективно решить задачи о близости и интервального поиска. Наконец, при хорошем математическом ожидании эффективности поиска худшие случаи поиска обладают сложностью выше сложности поиска полным перебором, так как приходится перебирать пустые ячейки [6]. Очевидно, что хеш-таблицы имеют огромный потенциал. И на сегодняшний день крайне важно развивать данное направление исследований. Необходимо ослаблять и устранять вышеозначенные недостатки, а также формировать общую теорию хеш-таблиц.

Рассмотрим некоторые понятия, связанные с хеш-функцией. Пусть X – множество элементов, хранимых в $< h_m, A_m >$. Очевидно, что $X \subset U$.

Хеш-функция называется совершенной [7-8], если она инъективна [9] на X. То есть при хранении элементов X не возникло коллизий.

Хеш-функция называется минимальной [10-11], если она сюръективна [9] на X. То есть, нет элементов массива A_m, которые бы пустовали, а значит, нет и избыточной памяти.

Понятия совершенной и минимальной хеш-функций, а также алгоритмы их построения подробно обсуждаются во многих научных работах [7, 8, 10, 11] и учебной литературе [1-2].

Минимальная совершенная хеш-функция обладает свойствами инъективности и сюръективности, а значит [9] является биекцией X на $\overline{0, m-1}$. Такая функция позволяет осуществлять поиск и удаление за одну итерацию, а также не требует избыточной памяти, вместе с тем, операция добавления элементов будет всегда вызывать коллизию и некоторые ме-

ханизмы разрешения коллизий, например, линейное зондирование, будут неприменимы. Минимальные совершенные хеш-функции удобны, когда набор хранимых элементов известен заранее [11] и операций добавления и удаления не предусматривается.

Важным свойством некоторых хеш-функций является их монотонность [10-12]. Она позволяет относительно быстро решать задачу о близости и осуществлять интервальный поиск, что являлось крайне асимптотически сложной задачей для немонотонных хеш-функций. Действительно, если функция монотонна, то либо найденное, либо ближайшее ненулевое слева или справа в A_m значение являются решением задачи о близости. А проверка на близость трёх элементов производится за константное время. Таким образом, для монотонных хеш-функций сложность решения задачи о близости асимптотически совпадает с решением поисковой задачи и является $O(1)$.

Задача интервального поиска для монотонных хеш-функций решается нахождением двух ближайших значений к границам запроса-интервала и перечислением всех ненулевых значений A_m между ними. При использовании немонотонной хеш-функции такие задачи привели бы к обходу всего массива A_m, число элементов которого может быть существенно и даже асимптотически больше числа хранимых элементов.

Отметим также, что если монотонная хеш-функция является ещё и совершенной, то сложность интервального поиска равна его минимальной границе [3] – сложности перечисления всех найденных элементов.

Метод генерации монотонных хеш-функций

Далее, без ограничения общности, будем рассматривать U, как множество числовой природы. Более подробно данное допущение пояснено в литературе [1, 6]. Здесь же просто примем это как условие поставленной задачи хранения и поиска объектов.

Пусть $X=\{x_1, x_2, x_3, \ldots x_n\}$, $X \subset [x_1, x_n] \subset U, x_1 < x_2, \ldots < x_n$. Строгость неравенства объясняется тем, что хранение одинаковых элементов реализуется на основе хранения одного из них и учёта их количества. Построим совершенную монотонную хеш-функцию для множества X.

Рассмотрим множество $F\{ f \mid f$ – непрерывная, строго возрастающая биекция на U и f: $[x_1, x_n] \rightarrow [0, 1] \}$. Назовём F образующим множеством. Выберем некоторую функцию $f \in F$, назовём её образующей и рассмотрим образы элементов X. Из биективности и возрастания f на U следует, что:

$$0 = f(x_1) < f(x_2) \ldots < f(x_n) = 1. \tag{2}$$

Введём в рассмотрение показатель $\delta_{\max}$, вычисляемый по следующей формуле:

$$\delta_{\max} = \min_{i=\overline{2,n}} \left(f(x_i) - f(x_{i-1}) \right). \tag{3}$$

Для построения хеш-таблицы возьмём произвольное число $\delta \le \delta_{\max}$ и вычислим m – размер индексного массива A_m хеш-таблицы:

$$m = \left\lfloor \frac{1}{\delta} \right\rfloor + 1, \tag{4}$$

где $\lfloor \ \rfloor$ – округление в меньшую сторону.

Рассмотрим множество полуинтервалов $y_i = [i \cdot \delta; (i+1) \cdot \delta), i = \overline{0, m-1}$.

Заметим, что:

$$\bigcup_{i=\overline{0,m-1}} y_i = [0 \ ; \delta \cdot m] \supset [0 \ ; 1] \tag{5}$$

Действительно, для $\forall i = \overline{0, m-2}$ правая выколотая граница полуинтервала y_i совпадает с левой включённой границей полуинтервала y_{i+1}. Таким образом, объединение всех полу-

интервалов приведёт к образованию одного полуинтервала с левой включённой границей y_0 и правой выколотой y_{m-1}, то есть к $\left[0\ ;\delta\cdot m\right)$. Осталось показать, что $\delta\cdot m>1$. Для этого проведём следующие равносильные преобразования на основе (4):

$$\left\lfloor\frac{1}{\delta}\right\rfloor=m-1,$$

$$m-1\le\frac{1}{\delta}<m$$

Помножим обе части неравенства на δ:

$$\delta\cdot(m-1)\le1<\delta\cdot m.$$

Что и требовалось доказать.

Зададим хеш-функцию следующим образом:

$$h_m(x)=\left\lfloor\frac{f(x)}{\delta}\right\rfloor \tag{6}$$

Индексный массив A_m определяется числом его элементов, которое было вычислено в (3). Покажем, что h_m – монотонная хеш-функция на отрезке $[x_1,x_n]$ и совершенная монотонная на множестве X. Для этого докажем следующие леммы.

Лемма 1. Пусть $x\in[x_1,x_n]$. Тогда для того, чтобы $f(x)\in y_i, i=\overline{0,m-1}$, необходимо и достаточно $h_m(x)=i$.

Доказательство. Покажем необходимость. Возьмём и зафиксируем произвольное $i=\overline{0,m-1}$ и произвольный $x\in[x_1,x_n]$. Пусть $f(x)\in y_i$, это означает, что

$$i\cdot\delta\le f(x)<(i+1)\cdot\delta. \tag{7}$$

Разделим все части неравенства (7) на δ:

$$i\le\frac{f(x)}{\delta}<(i+1)\cdot \tag{8}$$

Из (8) следует, что $\left\lfloor\frac{f(x)}{\delta}\right\rfloor=i$ или, согласно (6) $h_m(x)=i$. Необходимость доказана.

Покажем достаточность. Для этого предположим противное, то есть $\exists i=\overline{0,m-1}$ и $x\in[x_1,x_n]$, такие что $h_m(x)=i$ и $f(x)\notin y_i$. Так как $x\in[x_1,x_n]$ и в силу (5) $f(x)\in[0,1]\subset\bigcup\limits_{j=\overline{0,m-1}}y_j$.

Следовательно $\exists j=\overline{0,m-1}$, такое, что $f(x)\in y_j$. Тогда в силу доказанной ранее необходимости $h_m(x)=j$, но по нашему предположению $h_m(x)=i$, то есть $i=j$, а значит $f(x)\in y_i$. Получили противоречие. Наше предположение неверно, а достаточность доказана.

Лемма 2. $\left(\forall i,j=\overline{1,n}\right)\left(h_m(x_i)\ne h_m(x_j)\right)$

Доказательство. Предположим противное, то есть $\left(\exists i,j=\overline{1,n}\right)\left(h_m(x_i)=h_m(x_j)\right)$. Пусть $h_m(x_i)=h_m(x_j)=k, k=\overline{0,m-1}$, тогда в силу леммы 1 $f(x_i)\in y_k$ и $f(x_j)\in y_k$. Положим для определённости, что $f(x_i)<f(x_j)$(строгость знака сравнения обусловлена (2)).Отсюда получаем

$$k\cdot\delta\le f(x_i)<f(x_j)<(k+1)\cdot\delta. \tag{9}$$

Из (9) и выбора $\delta\le\delta_{\max}$ следует, что $f(x_j)-f(x_i)<\delta\le\delta_{\max}$. С другой стороны из (2) и (3) следует, что $f(x_j)-f(x_i)\ge f(x_{i+1})-f(x_i)\ge\delta_{\max}\ge\delta$. Наше предположение неверно. Лемма доказана.

Утверждение 1. Функция h_m, заданная выражениями (2)–(6), является монотонной на $[x_1, x_n]$ и совершенной на X хеш-функцией, задающей хеш-таблицу из m элементов.

Доказательство. Согласно выбору функции f и выражению (5) $(\forall x \in [x_1, x_n])(f(x) \in y_k, k = \overline{0, m-1})$, что по лемме 1 равносильно $h_m(x) = k, k = \overline{0, m-1}$. Таким образом, показано, что h_m является хеш-функцией на $[x_1, x_n]$, задающей хеш-таблицу из m элементов.

Возьмём и зафиксируем произвольные $\tilde{x}, \hat{x} \in [x_1, x_n]$, $\tilde{x} < \hat{x}$. В силу (5) $(\exists k = \overline{0, m-1})(f(\tilde{x}) \in y_k)$, тогда по лемме 1 $h_m(\tilde{x}) = k$, а в силу строго возрастания f её значение в $\hat{x}$ будет строго больше того её значения в $\tilde{x}$. Из этого делаем вывод, что $f(\hat{x})$ принадлежит либо полуинтервалу y_k, либо любому другому полуинтервалу, расположенному на числовой оси правее y_k. То есть, $f(\hat{x}) \in y_s, k \le s$. Тогда по лемме 1 $k \le h_m(\hat{x})$ или $h_m(\tilde{x}) \le h_m(\hat{x})$. Таким образом для произвольных $\tilde{x}, \hat{x} \in [x_1, x_n]$, $\tilde{x} < \hat{x}$ показано, что $h_m(\tilde{x}) \le h_m(\hat{x})$. Монотонность h_m на $[x_1, x_n]$ доказана.

Таким образом показано, что функция h_m является монотонной хеш-функцией на $[x_1, x_n]$, а, следовательно, она монотонная хеш-функция и на подмножестве $[x_1, x_n] - X$. Совершенство h_m на X следует из определения, так как инъективность h_m на X доказана леммой 2.

Утверждение доказано.

Отметим, что предложенный подход к генерации хеш-функций предполагает наличие уже некоторого объёма хранимых данных – множества X. И поэтому он удобен лишь при рехешировании, так как в начале работы с хеш-таблицей X равен пустому множеству.

Процесс получения хеш-функции начинается с построения F – образующего множества функций, которое строится на основе X. Далее происходит выбор образующей функции $f \in F$. Он произволен и от него зависят свойства хеш-функции. После этого по формуле (3) находим $\delta_{\max}$. Выбор же δ из полуинтервала $[\delta_{\max}, 0)$ произволен, и от него также зависят свойства хеш-функции. Таким образом, некоторую хеш-функцию, сгенерированную на основе предложенного подхода, удобно задать в виде

$$<X, f(x), \delta>, \tag{10}$$

а соответствующую ей хеш-таблицу

$$<X, f(x), \delta, A_m, L, R>, \tag{11}$$

где L и R – ячейки, отвечающие соответственно за поиск и добавление элементов, находящихся в полуинтервалах $[\inf U; x_1)$ и $(x_n; \sup U]$. Таким образом, становятся возможны поиск и добавление в хеш-таблицу (11) произвольного элемента множества U. Естественно, при проведении операции добавления элемента могут возникать коллизии. В данной статье не рассматриваются алгоритмы разрешения коллизий. Единственное требование, которое к ним предъявляется – это сохранение упорядоченности элементов. Так, например, разрешение коллизий методом цепочек[1-3] удовлетворяет предъявленным требованиям, а большинство механизмов разрешения коллизий основанных на открытой адресации[1-2] могут нарушить упорядоченность данных, располагающихся в хеш-таблицах.

Отметим, что в самом общем случае δ может быть больше $\delta_{\max}$. Действительно, неравенство $\delta \le \delta_{\max}$ используется только в доказательстве леммы 2, которая в свою очередь используется только для того, чтобы показать совершенство (10) на X. Таким образом, если в (10) не соблюдено неравенство $\delta \le \delta_{\max}$, то утверждение 1 верно за исключением совершенства хеш-функции на X.

Может показаться, что ограничения, наложенные на множество образующих функций, довольно строгие. Подробное обоснование выбора именно такого множества представлено в [13]. Способ построения хеш-функции (3) – (6) назовём генератором хеш-функций.

Исследование свойств генерируемых хеш-функций

Для исследования свойств сгенерированных с помощью (3) – (6) хеш-функций введём некоторые обозначения.

Множество $X_U = \left\{x_i \in U \mid i = \overline{1,n}, \inf U = x_1 < x_2, ... < x_n = \sup U\right\}$.

Функция P_X – функция распределения случайной величины X. Здесь предполагается, что в хеш-таблицу записываются случайные величины, подчинённые некоторому закону распределения. Этому закону подчиняется как множество X, так и новые элементы, записываемые в хеш-таблицу. Плотность вероятности случайной величины X обозначим $\rho(x)$.

Введём функционал $G_i(f), i = \overline{2,n}, f \in F$, такой что $G_i(f) = f(x_i) - f(x_{i-1})$. Назовём функцию f_{opt} оптимальной, если

$$f_{opt} = \arg\left(\max_{f \in F}\left(\min_{i=\overline{2,n}} G_i(f)\right)\right). \tag{12}$$

Частные случаи хеш-функций вида (10) рассматривались в литературе. Например, хеш-функция $< X, x, \delta_{\max} >$ представлена в работах Э.Э. Гасанова и Л.П. Луговской [3, 14]. В них показано [14], что операция поиска производится за 6 элементарных операций, а m в среднем при равномерном законе распределения X равно n^2. Вместе с тем, в данных работах не было исследовано никакого механизма разрешения коллизий.

В «Теории хранения и поиска информации» Э.Э. Гасанова и В.Б. Кудрявцева (теорема 13, примеры 1 и 2) [3] представлены в качестве элементов предложенных алгоритмов хеш-функции $<X, \frac{x - x_1}{x_n - x_1}, \frac{1}{k}>$, где k – некоторое число и подробно исследованы, а в [15] предложен алгоритм вида $<X_U, P_X, \frac{1}{k}>$. Все алгоритмы, базирующиеся на хеш-функциях, предложенных в вышеупомянутых работах [3, 14-15], имеют высокую оценку математического ожидания скорости поиска (1-2 итерации) и нижнюю логарифмическую асимптотическую оценку в худшем случае.

Рассмотрим некоторые хеш-функции. Но для начала предложим критерии их оценки. Хеш-функции не являются хеш-таблицами, они не предусматривают алгоритмов разрешения коллизий, и потому по ним, в отрыве от конкретной реализации ассоциативного массива невозможно оценить время решения поисковых задач, а также добавления и удаления элементов. В качестве критерия оценки хеш-функции предлагается «равномерность рассеивания» [3] «равномерность распределения записей в массиве» [2] или «равномерность хеширования» [1]. Все эти термины означают одно и тоже, а именно то, что для каждого добавляемого объекта хранения, его попадание в тот или иной элемент A_m приближённо равновероятен. Будем называть этот показатель хеш-функции – равномерность хеширования.

Рассмотрим хеш-функцию

$$<X_U, P_X, \delta>. \tag{13}$$

Во-первых, отметим, что заданная хеш-функция корректна, а именно $P_X \in F$. Действительно, P_X непрерывная, строго возрастающая, и в $x_1 = \inf X$ она равна 0, $x_n = \sup U$ – единице. Вместе с тем, рассматриваемая хеш-функция удовлетворяет утверждению 1. Какова же её равномерность хеширования? Для ответа на этот вопрос возьмём и зафиксируем произвольный индекс $i = \overline{0, m-1}$ и рассмотрим $x \in U \mid h_m(x) = i$. Согласно лемме 1 это равносильно тому, что $P_X(x) \in [i \cdot \delta; (i+1) \cdot \delta)$. Тогда, в силу того, что любая биективная функция обратима, имеет место неравенство:

$$P_X^{-1}(i \cdot \delta) \le x < P_X^{-1}((i+1) \cdot \delta). \tag{14}$$

То есть все x, значение хеш-функции (13) которых равно i, удовлетворяют неравен-

ству (14). А вероятность p_i – того что произвольный элемент множества U попадёт в полуинтервал, заданный неравенством (14) вычисляется согласно расчёту геометрической вероятности следующим образом:

$$p_i = P_X\left(P_X^{-1}((i+1)\cdot\delta)\right) - P_X\left(P_X^{-1}(i\cdot\delta)\right) = (i+1)\cdot\delta - i\cdot\delta = i\cdot\delta + \delta - i\cdot\delta = \delta$$

или

$$p_i = \delta \ . \tag{15}$$

То есть для хеш-функции (13) получение любого значения равновероятно, она обладает максимальной равномерностью хеширования. Однако на практике возникает ряд проблем. Зачастую неизвестны ни P_X, ни $\inf U$, ни $\sup U$. Более того, из-за необходимости разбивать на маленькие отрезки образ U появляется потребность в больших дополнительных объёмах памяти.

В работе «Оптимизационная задача построения отображения на адресное пространство для модели хранения данных с константным временем поиска» [13] исследовались хеш-функции вида:

$$<X, f_{opt}, \delta_{\max}>. \tag{16}$$

Для таких функций было доказано, что они являются минимальными, монотонными и совершенными на X. В случае если базирующаяся на (16) хеш-таблица предполагает операции добавления, достаточно взять любое $\delta < \delta_{\max}$. Оно не изменит совершенности и монотонности хеш-функции, но добавит достаточное количество пустых элементов. Также очевидно, что изменение δ в полуинтервале $(0, \delta_{\max}]$ фактически представляет собой управление размером дополнительной памяти. Поэтому перейдём к рассмотрению следующей хеш-функции:

$$<X, f_{opt}, \delta>, \ 0 < \delta \le \delta_{\max}. \tag{17}$$

Исследуем свойства хеш-функции (17) подробнее.

Лемма 3. Для хеш-функции вида (17) $\delta_{\max} = \frac{1}{n-1}$.

Доказательство. Предположим, что $\delta_{\max} > \frac{1}{n-1}$. Тогда, согласно (3):

$$\left(\forall i = \overline{2,n}\right)\left(f(x_i) - f(x_{i-1}) \ge \delta_{\max} > \frac{1}{n-1}\right) \Rightarrow \sum_{i=2}^{n} f(x_i) - f(x_{i-1}) > \sum_{i=2}^{n} \frac{1}{n-1}$$

или

$$f(x_n) - f(x_1) > \frac{n-1}{n-1} \text{ или } 1 > 1.$$

Получили противоречие, следовательно, предположение неверно, значит $\delta_{\max}$ для любой хеш-функции не превосходит $\frac{1}{n-1}$.

Согласно построению (12) f_{opt} таково, что $\delta_{\max}$ – максимально для всех f_{opt}. Таким образом, для доказательства леммы достаточно показать существование $f^* \in F$, такой, что $\min\limits_{i=2,n}\left(f^*(x_i) - f^*(x_{i-1})\right) = \frac{1}{n-1}$.

В качестве f^* выберем любую функцию из F, удовлетворяющую системе:

$$\left(f^*(x_i) = \frac{i-1}{n-1}\right)\left(\forall i = \overline{2,n}\right). \tag{18}$$

Действительно, для таких функций $f^*(x_i) - f^*(x_{i-1}) = \frac{i-1}{n-1} - \frac{i-1}{n-1} = \frac{1}{n-1}$ для $\left(\forall i = \overline{2,n}\right)$, а значит и $\min\limits_{i=2,n}\left(f^*(x_i) - f^*(x_{i-1})\right) = \frac{1}{n-1}$.

Пусть F_{opt} – множество оптимальных функций на X_n.

Следствие 1. Для $\left(\forall f_{opt} \in F_{opt}\right)\left(\forall i = \overline{2,n}\right)\left(f_{opt}(x_i) - f_{opt}(x_{i-1}) = \frac{1}{n-1}\right)$.

Доказательство. Предположим противное. Учтём, что согласно лемме 3: $\delta_{\max} = \frac{1}{n-1}$. То есть: $\left(\forall f_{opt} \in F_{opt}\right)\left(\forall i = \overline{2,n}\right)\left(f_{opt}(x_i) - f_{opt}(x_{i-1}) \ge \frac{1}{n-1}\right)$. Тогда отрицание условия будет иметь вид:

$$\left(\exists f_{opt} \in F_{opt}\right)\left(\exists i = \overline{2,n}\right)\left(f_{opt}(x_i) - f_{opt}(x_{i-1}) > \frac{1}{n-1}\right). \tag{19}$$

Возьмём и зафиксируем f_{opt} из (19).Тогда для него верно, что

$$\sum_{i=2}^{n} f_{opt}(x_i) - f_{opt}(x_{i-1}) > \sum_{i=1}^{n} \frac{1}{n-1} \Leftrightarrow 1 > 1.$$

Получили противоречие. Следовательно, наше предположение неверно и следствие доказано.

Следствие 2.

$$F_{opt} = \left\{ f \in F \,|\, \left(f^*(x_i) = \frac{i-1}{n-1}\right)\left(\forall i = \overline{2,n}\right)\right\}. \tag{20}$$

Доказательство. В лемме 3 было показано, что произвольная функция из F_{opt} является оптимальной. Теперь покажем обратное, то есть любая оптимальная функция удовлетворяет (18). Возьмём и зафиксируем произвольное $i = \overline{2,n}$. Рассмотрим сумму (*S*):

$$S = \sum_{k=2}^{i} f_{opt}(x_k) - f_{opt}(x_{k-1}).$$

С одной стороны, $S = f_{opt}(x_i) - f_{opt}(x_1) = f_{opt}(x_i)$, а согласно следствию 1 $S = \frac{i-1}{n-1}$.Таким образом, для произвольного $i = \overline{2,n}$ получаем $f_{opt}(x_i) = \frac{i-1}{n-1}$. Что и требовалось доказать.

Под X_n будем понимать множество *X*, состоящее из *n* элементов и меняющееся путём добавления новых элементов, распределённых на *U* согласно закону P_x. Для простоты дальнейшей работы и корректности записей обозначим $X_n = \left\{x_1^n, x_2^n, ..., x_n^n\right\}$.

Лемма 4. При $n \to \infty$ $X_n \to X_U$.

Доказательство. Условие Леммы аналогично следующим двум условиям:

$$\begin{cases} \lim\limits_{n\to\infty} x_1^n = \inf U, \\ \lim\limits_{n\to\infty} x_n^n = \sup U. \end{cases} \tag{21}$$

Возьмём и зафиксируем произвольное $\varepsilon > 0$. Рассмотрим полуинтервалы $\left[\inf U, \inf U + \varepsilon\right)$ и $\left(\sup U - \varepsilon, \sup U\right]$. Вероятности попадания элемента в эти полуинтервалы: $P_x\left(\inf U + \varepsilon\right)$ и $P_x\left(\sup U\right) - P_x\left(\sup U - \varepsilon\right)$ являются ненулевыми, а потому с неограниченным возрастанием *n*, найдутся такие n_1 и n_2, что $x_1^{n_1} \in \left[\inf U, \inf U + \varepsilon\right)$, а $x_{n_1}^{n_1} \in \left(\sup U - \varepsilon, \sup U\right]$. Начиная с этого момента с возрастанием *n* x_1^n не увеличится, а x_n^n не уменьшится. То есть $\left(\forall n > n_1\right)\left(\left|x_1^n - \inf U\right| < \varepsilon\right)$ и $\left(\forall n > n_2\right)\left(\left|x_n^n - \sup U\right| < \varepsilon\right)$. То есть (21) верно, по определению предела.

Утверждение 2. При $n \to \infty$ $f_{opt} \to P_X$.

Доказательство. Согласно (20) и в силу строгого возрастания f_{opt}

$$\left(\forall x \in \left(x_{i-1}, x_i\right]\right)\left(\forall f_{opt} \in F_{opt}\right)\left(\frac{i-2}{n-1} < f_{opt}(x) \le \frac{i-1}{n-1}\right) \tag{22}$$

Рассмотрим множество $Z_m = \left\{ z_k, k = \overline{0,m} \mid z_k = P_X^{-1}\left(\frac{k}{m}\right) \right\}$. Вероятность того, что случайная величина *x* попадёт в отрезок $[z_{k-1}, z_k]$, равна: $P_X\left(P_X^{-1}\left(\frac{k}{m}\right)\right) -$

$$- P_X\left(P_X^{-1}\left(\frac{k-1}{m}\right)\right) = \frac{k}{m} - \frac{k-1}{m} = \frac{1}{m}, \forall k = \overline{1,m.}$$

Пусть w_k – число точек множества *X*, попавших в $[z_{k-1}, z_k]$. Тогда, согласно статистическому определению вероятности [16]

$$\lim_{n\to\infty} \frac{w_k}{n} = \frac{1}{m}, \forall k = \overline{1,m.} \tag{23}$$

Возьмём и зафиксируем некоторое число $k = \overline{1,m.}$. Рассмотрим такие целые числа *M* и *N*, что $(x_M \le z_{k-1} < z_k \le x_N) \wedge (x_{M+1} > z_{k-1}) \wedge (X_{N-1} < z_k)$.

Тогда, согласно нашим обозначениям

$$M = \sum_{l=1}^{k-1} w_l, N = M = \sum_{l=1}^{k} w_l. \tag{24}$$

Из (24) и (22) легко видеть, что

$$(\forall x \in [z_{k-1}, z_k])(\forall f_{opt} \in F_{opt})\left(\frac{\sum_{l=1}^{k-1} w_l - 2}{n-1} < f_{opt}(x) \le \frac{\sum_{l=1}^{k} w_l - 1}{n-1} \right)$$

или

$$(\forall x \in [z_{k-1}, z_k])(\forall f_{opt} \in F_{opt})\left(\sum_{l=1}^{k-1} \frac{w_l}{n-1} - \frac{2}{n-1} < f_{opt}(x) \le \sum_{l=1}^{k} \frac{w_l}{n-1} - \frac{1}{n-1} \right). \tag{25}$$

Возьмём пределы от каждого члена неравенства в (25) при $n \to \infty$. Получим

$$(\forall x \in [z_{k-1}, z_k])(\forall f_{opt} \in F_{opt})\left(\lim_{n\to\infty}\left(\sum_{l=1}^{k-1} \frac{w_l}{n-1} - \frac{2}{n-1} \right) \le \lim_{n\to\infty}\left(f_{opt}(x)\right) \le \lim_{n\to\infty}\left(\sum_{l=1}^{k} \frac{w_l}{n-1} - \frac{1}{n-1} \right) \right)$$

или

$$(\forall x \in [z_{k-1}, z_k])(\forall f_{opt} \in F_{opt})\left(\sum_{l=1}^{k-1} \lim_{n\to\infty}\left(\frac{w_l}{n-1} \right) \le f_{opt}(x) \le \sum_{l=1}^{k} \lim_{n\to\infty}\left(\frac{w_l}{n-1} \right) \right).$$

Тогда в силу (23) получаем:

$$(\forall x \in [z_{k-1}, z_k])(\forall f_{opt} \in F_{opt})\left(\frac{k-1}{m} \le f_{opt}(x) \le \frac{k}{m} \right).$$

Из чего в силу построения множества Z_m делаем вывод

$$(\forall x \in [z_{k-1}, z_k])(\forall f_{opt} \in F_{opt})\left(P_X(z_{k-1}) \le f_{opt}(x) \le P_X(z_k)\right). \tag{26}$$

Согласно построению множества $Z_m, z_k = P_X^{-1}\left(\frac{k}{m}\right), \forall k = \overline{0,m}$. Тогда (26) перепишем в виде:

$$(\forall x \in [z_{k-1}, z_k])(\forall f_{opt} \in F_{opt})\left(P_X\left(P_X^{-1}\left(\frac{k-1}{m}\right)\right) \le f_{opt}(x) \le P_X\left(P_X^{-1}\left(\frac{k}{m}\right)\right) \right)$$

или

$$(\forall x \in [z_{k-1}, z_k])(\forall f_{opt} \in F_{opt})\left(\frac{k-1}{m} \le f_{opt}(x) \le \frac{k}{m} \right). \tag{27}$$

С другой стороны:

$$x \in [z_{k-1}, z_k] \Leftrightarrow (z_{k-1} \le x \le z_k) \Rightarrow$$
$$P_X(z_{k-1}) \le P_X(x) \le P_X(z_k) \Leftrightarrow$$
$$P_X\left(P_X^{-1}\left(\frac{k-1}{m}\right)\right) \le P_X(x) \le P_X\left(P_X^{-1}\left(\frac{k}{m}\right)\right) \Leftrightarrow$$
$$\frac{k-1}{m} \le P_X(x) \le \frac{k}{m}. \quad (28)$$

В силу (27) и (28) получаем:

$$(\forall x \in [z_{k-1}, z_k])(\forall f_{opt} \in F_{opt})\left(f_{opt}(x), P_X(x) \in \left[\frac{k-1}{m}, \frac{k}{m}\right]\right). \quad (29)$$

Выбор m произволен. Это позволяет нам неограниченно увеличивать m, сохраняя при этом условие $m<<n$, так как n устремлён к бесконечности. В случае неограниченного увеличения m, отрезок $\left[\frac{k-1}{m}, \frac{k}{m}\right]$ неограниченно уменьшается, что приводит в силу (29) к неограниченному сближению $f_{opt}(x), P_X(x)$. Что и требовалось доказать.

Следствие. При $n \to \infty$. (17)$\to$(13).

Доказательство. Действительно при $n \to \infty$ в силу леммы 4 $f_{opt} \to P_X$, а в силу утверждения 2 $f_{opt} \to P_X$. Это и означает $<X, f_{opt}, \delta> \to <X_U, P_X, \delta>$ или (17)$\to$(13).

Заключение

Следствие к утверждению 2 приводит к гипотезе о том, что на больших значениях n хеш-функция вида (17) обладает высокой равномерностью хеширования. Однако возможно, что высокая равномерность достигается только на очень больших n. И использование (17) для построения хеш-таблиц на практике может быть неоправданно. Для подтверждения или опровержения гипотезы необходимо провести вычислительные эксперименты. При постановке данных экспериментов целесообразно в качестве f_{opt} использовать соответствующий линейный сплайн [17]. Сплайн не обеспечивает высокой скорости поиска, так как его вычисление имеет логарифмическую асимптотическую оценку. Однако он представляет собой f_{opt}, а целью экспериментов является исследования поведения f_{opt} при росте n. Также с помощью сплайнов удобно задать верхние и нижние ограничения для f_{opt}. Проведение этих исследований позволит определить области применения и дальнейшие пути развития, предложенного в данной работе генератора хеш-функций.

Библиографический список

1. **Кормен, Т**. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. – 2-е изд. – М.: Вильямс, 2011. – 1290 с.
2. **Кузнецов, С.Д.** Методы сортировки и поиска / С.Д. Кузнецов. – ИСП РАН, Центр Информационных Технологий Режим доступа: http://citforum.ru/programming/theory/sorting/sorting2.shtml.
3. **Гасанов, Э.Э.** Теория хранения и поиска информации / Э.Э. Гасанов, В.Б. Кудрявцев. – М.: ФИЗМАТЛИТ, 2002. – 288 с.
4. **Loftsgaarden, D.** A nonparametric density function / D.O. Loftsgaarden C.P. Queensberry. – Ann. Math. Stat. 36, 1965. С. 1049–1051.
5. **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. С. 241–247.
6. **Кнут, Д.** Искусство программирования для ЭВМ Т. 3. Сортировка и поиск / Д. Кнут. – М.: Мир, 1978.

7. **Zbigniew, J.C.** An optimal algorithm for generating minimal perfect hash functions / J.C. Zbigniew, G Havas, B.S. Majewski. – Informational processing letters. 1992. №43(5). С. 257–264.
8. **Pescio, C.** Minimal perfect hashing. – Dr. Dobb's Journal. 1996. № 249.
9. **Верещагин, Н.К.** Лекции по математической логике и теории алгоритмов. Ч. 1. Начала теории множеств. – 2-е изд., испр. / Н.К. Верещагин, А. Шень. – М.: МЦНМО, 2002. – 128 с.
10. **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.
11. **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.
12. Математическая энциклопедия / под ред. И. М. Виноградова. – М.: Советская энциклопедия, 1977–1985.
13. **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.
14. **Гасанов, Э.Э.** Константный в худшем случае алгоритм поиска идентичных объектов / Э.Э. Гасанов, Ю.П. Луговская // Дискретная математика, 1999. Т.11. №4. С. 139–144.
15. **Поляков, Д.В.** Алгоритм поиска идентичных объектов на непрерывном множестве // Методы управления потоками в транспортных системах. МАДИ. 2009. С. 114–121.
16. **Лаговский, А.Ф.** Теория вероятности: учеб. пособие / А.Ф. Лаговский. – Калининград: Изд.-во Калининградского университета, 1997. – 103 с.
17. **Алберг, Дж.** Теория сплайнов и её приложения / Дж. Алберг, Э. Нильсон, Дж. Уолш. – М.: Мир, 1972. – 320 с.

Дата поступления
в редакцию 16.04.2015

D.V. Polyakov, A.I. Popov

GENERATOR OF THE MONOTONE HASH FUNCTION FOR AN ASSOCIATIVE ARRAY

Tambov state technical university

Purpose: Generator of hash functions with a good value of asymptotic characteristics of algorithms solving of basic search problems is offered.
Approach: The methodology is based on the set theory and theory of information retrieval.
Findings: The approach to the generation of hash functions is presented. Offered the method for obtaining a hash function based on the considered generator.
Research limitations: Proposed and substantiated the hypothesis of high characteristics of the algorithms for solving basic search problems by hash functions obtained with the proposed method. Expected that the further researches will confirm the hypothesis and will specify it’s characteristics.
Originality/value: Proposed generator defined the family of hash functions for new effective associative arrays based on hash table.

Key words: hash function, hash table, an associative array, minimal hashing, monotone hashing, perfect hashing, generation of hash functions, search of an identical object, the problem of proximity, interval search.