Динамические базы данных, основывающиеся на хешировании методом цепочек

И. С. Лапшов

В работе исследуется сложность таких основных операций в динамических базах данных, как поиск, вставка и удаление записей. Предложены такая структура базы данных и соответствующие ей алгоритмы, которые с помощью двух параллельных вычислительных процессов позволяют совершать основные операции над базой данных в среднем за константное время.

Введение

Объектом исследования являются динамические базы данных, то есть структуры, хранящие информацию и предоставляющие пользователю набор операций для поиска и обработки этой информации.

В качестве представления динамической базы данных (БД) используется информационно-графовая модель данных [1]. В этой модели состоянию базы данных в каждый момент времени сопоставляется ориентированный граф с нагруженными ребрами и вершинами. Основным операциям, к которым будем относить поиск, вставку и удаление записи, соответствуют процедуры, выполняющие перемещения по ребрам графа и модифицирующие его структуру.

Исследовалась сложность основных операций как время выполнения соответствующих процедур в среднем и в худшем случаях.

Существует много работ, посвященных изучению алгоритмов поиска, вставки и удаления. Часть из них связаны с методом хеширования, который является объектом исследования данной работы.

Хеширование предполагает наличие хеш-функции. Хеш-функция $h(y)$ определена на множестве записей Y и переводит его в множество $\{1, 2, \ldots, m\}$, где m — параметр хеш-функции.

Обычно используется метод хеширования, предложенный А. Думи [2] и называемый методом цепочек. Этот метод предполагает наличие m списков. Тогда для поступившего поискового запроса x (он же запись) вычисляется значение хеш-функции $h(x)$, и если оно равно i, то просматривается i-ый список и в нем ищется запись x. Аналогично происходит и процедура вставки и удаления записи.

Методы хеширования хороши тем, что при удачном выборе хеш-функции, равномерно рассеивающей поступающие записи, время основных операций в среднем будет очень малым.

Однако, большинству методов хеширования присущи общие недостатки. Во-первых, пользуясь словами Д. Кнута [3, с. 642], «... нужно свято верить в теорию вероятностей, ибо они (методы хеширования) эффективны лишь в среднем, а худший случай просто ужасен!».

Во-вторых, часто довольно трудно распределить память под хеш-таблицу. Периодически, когда записей в БД накапливается большое количество, требуется тягостное «рехеширование». В то же время, если выделить памяти слишком много, то это может быть расточительно.

В данной работе предлагается алгоритм основных операций, который в среднем эффективен как хорошие методы хеширования, то есть обеспечивает мгновенное решение. С другой стороны, предложенный алгоритм не обладает вторым недостатком: «рехеширование» не влияет на порядок сложности основных операций.

1. Основные понятия и формализация модели

Для исследования необходимо построить формальную модель.

Назовем некоторое множество Y *множеством записей*, а элементы этого множества — *записями*. Будем считать, что информация, хранящаяся в базе данных, представляет собой конечное подмножество V множества записей. Назовем это подмножество *библиотекой*.

Будем рассматривать три вида операций над базами данных: поиск записи, вставка записи и удаление записи.

Для выполнения поиска необходим *поисковый запрос* — объект из множества X, которое мы назовем *множеством поисковых запросов*. В теории поиска информации вводится также бинарное отношение $\rho \subseteq X \times Y$, называемое отношением поиска. Запись $y \in Y$ удовлетворяет запросу $x \in X$, если $x \,\rho\, y$. Тройку $\langle X, Y, \rho \rangle$ называют *типом поиска*. Операция поиска по запросу $x \in X$ состоит в отыскании всех записей $y \in V$, таких, что $x \,\rho\, y$.

Далее будем рассматривать поиск типа $\langle X, Y, \rho \rangle$, где $X = Y$, а ρ — отношение равенства на $Y \times Y$ — так называемый *поиск идентичных объектов*.

В качестве модели базы данных будем использовать информационные графы (ИГ).

Структура ИГ

Пусть F — множество одноместных предикатов, заданных на множестве X. Пусть G — множество функций, определенных на X, области значений которых являются отрезками натурального ряда. Назовем их *переключателями*.

Пару $\mathcal{F} = \langle F, G \rangle$ будем называть *базовым множеством*.

Пусть нам дан произвольный ориентированный граф (возможно, пустой). Выделим в нем одну вершину (если граф непустой) и назовем ее корнем ИГ. Также выделим произвольное подмножество вершин графа и назовем их *листьями* (корень может быть листом). Вершины графа, не являющиеся листьями, будем называть *внутренними вершинами*.

Сопоставим каждому листу графа некоторую запись из множества Y. Это соответствие назовем *нагрузкой листьев*.

Выделим некоторое подмножество внутренних вершин, которые назовем *точками переключения*. Каждой точке переключения v с полустепенью исхода m поставим в соответствие переключатель $g_v(x) \in G$, у которого область значений содержит не менее m чисел. Занумеруем все дуги, исходящие из v последовательными числами из области значений g_v. Такие дуги назовем *переключательными*.

Внутренние вершины, не являющиеся точками переключения, назовем предикатными. Дуги, не являющиеся переключательными, также назовем *предикатными*. Каждой предикатной дуге сопоставим символ из F.

Полученный граф назовем *информационным графом* над базовым множеством $\mathcal{F}$ с библиотекой V, где V — множество записей, приписанных листьям ИГ.

Поиск в ИГ

Каждому ИГ U можно сопоставить процедуру поиска. Предполагается, что эта процедура хранит в своей (внешней) памяти структуру U. Входными данными является запрос $x \in X$. Выходными данными является подмножество библиотеки V.

Пусть на вход процедуре поступил запрос $x \in X$. Если ИГ пустой, то процедура поиска завершается с пустым выходным множеством. В противном случае введем понятие активного множества вершин и внесем в него в начальный момент корень ИГ U и помечаем его. Далее по очереди просматриваем вершины из активного множества и для каждой из них проделываем следующее:

- если рассматриваемая вершина — лист, то запись, приписанную вершине, включаем в ответ;
- если рассматриваемая вершина — точка переключения, то вычисляем на запросе x переключатель, соответствующий данной вершине, и если дуга, нагрузка которой равна значению переключателя существует, и ее конец — не помеченная вершина, то помечаем конец дуги и включаем его в множество активных вершин;
- если рассматриваемая вершина предикатная, то просматриваем по очереди исходящие из нее дуги и вычисляем значения предикатов, приписанных этим дугам на запросе x. Концы дуг, которым соответствуют предикаты со значениями, равными 1, если они непомеченные, помечаем и включаем в множество активных вершин;
- исключаем рассматриваемую вершину из активного множества.

Процедура завершается по исчерпании активного множества.

Скажем, что ИГ U *решает задачу поиска*, если множество, полученное на выходе процедуры, будет содержать все те и только те записи библиотеки V, которые удовлетворяют запросу x.

Сложность поиска в ИГ

Пусть нам дан ИГ U над базовым множеством $\mathcal{F}$. Каждому символу $g \in G$ ($f \in F$) поставим в соответствие неотрицательное число $t(g)$ ($t(f)$) — *сложность переключателя g (предиката f)*. Тогда понятие базового множества функций можно расширить, рассматривая функции из F и G вместе со своими сложностями. Будем записывать это как $\langle F, G, t\rangle$.

Сложностью поиска по запросу x в ИГ U назовем сумму сложностей всех вычисленных в ходе процедуры поиска переключателей и предикатов. Обозначим сложность поиска $T(U, x)$.

Сложность ИГ U можно вводить двумя способами. Во-первых, как максимальную сложность на запросе

$$\widehat{T}(U) = \max_{x \in X} T(U, x).$$

Эту величину будем называть *верхней сложностью* ИГ U, или *сложностью в худшем случае*.

Во-вторых, можно вводить среднюю сложность ИГ U. Для этого будем считать, что x — случайная величина с неким заданным вероятностным распределением на множестве X. Тогда *средняя сложность поиска* в ИГ U будет определяться как математическое ожидание величины $T(U, x)$, то есть число

$$T(U) = \mathrm{M}_x T(U, x).$$

Вставка и удаление записи в ИГ

Назовем вставкой записи y в ИГ U такое преобразование U, при котором библиотека V, приписанная листьям ИГ U, преобразуется в $V \cup \{y\}$ и U остается ИГ, решающим задачу поиска для библиотеки $V \cup \{y\}$.

Удалением записи y из ИГ U назовем преобразование U, при котором библиотека V, приписанная листьям ИГ U, преобразуется в $V \setminus \{y\}$ и U остается ИГ, решающим задачу поиска для библиотеки $V \setminus \{y\}$.

Будем рассматривать вставку (удаление) записи в ИГ как процедуру, хранящую во внешней памяти структуру ИГ и выполняющуюся по некому условному алгоритму A. На вход процедуре подается элемент $y \in Y$, а результатом ее работы является преобразование U.

Алгоритм вставки/удаления представляет собой последовательность преобразований вершин, дуг графа, их пометок, которые мы договоримся считать элементарными, то есть на выполнение которых тратится небольшое и константное время. Набор конкретных преобразований будем называть *базовым множеством преобразований* и обозначать как $\mathcal{H} = \langle H \rangle$, где $H = \{h_1, \dots h_m\}$ — множество элементарных операций.

Пусть нам дан пустой ИГ. Осуществим некоторое количество последовательных вставок и удалений записей $y_1, \dots, y_l$, $l \in \mathbb{N}_0$. Полученный ИГ назовем *построенным по набору* $(y_1, \dots, y_l)$. Обозначим через $\mathcal{U}(A, n)$ множество ИГ с n листьями, построенных по алгоритму A по всевозможным наборам из Y. Введем также обозначение

$$\mathcal{U}(A) = \bigcup_{n \in \mathbb{N}} \mathcal{U}(A, n).$$

Вставка/удаление записи $y \in Y$ в ИГ U по алгоритму A имеет смысл только если $U \in \mathcal{U}(A)$.

Сложность вставки/удаления записи в ИГ

Сопоставим каждому $h \in H$ величину $t(h) \geqslant 0$ и назовем ее сложностью элементарной операции h. Значения таких величин будем обговаривать отдельно. Тогда можно расширить понятие базового множества элементарных операций, рассматривая каждую операцию вместе со своей сложностью. Обозначим это как $\mathcal{H} = \langle H, t \rangle$.

Любой алгоритм A вставки/удаления строится над некоторым базовым множеством функций $\mathcal{F} = \langle F, G, t \rangle$ и базовым множеством преобразований $\mathcal{H} = \langle H, t \rangle$.

Сложность вставки записи $y \in Y$ в ИГ U по алгоритму A будет равна сумме сложностей элементарных операций, произведенных в ходе выполнения процедуры по этому алгоритму. Обозначим ее $R_A(U, y)$.

Аналогично, сложность удаления записи определим как сумму сложностей соответствующих элементарных операций и обозначим как $S_A(U, y)$.

Величину

$$\widehat{R}_A(U) = \max_{y \in Y} R_A(U, y)$$

назовем верхней сложностью (или сложностью в худшем случае) вставки в ИГ U.

Величину

$$\widehat{S}_A(U) = \max_{y \in Y} S_A(U, y)$$

назовем верхней сложностью (или сложностью в худшем случае) удаления в ИГ U.

Считая y случайной величиной, среднюю сложность вставки записи y в ИГ U можно определять по формуле

$$R_A(U) = \mathrm{M}_y R_A(U, y);$$

Аналогично, среднюю сложность удаления записи y из ИГ U определим по формуле

$$S_A(U) = \mathrm{M}_y S_A(U, y).$$

Объемом $Q(U)$ ИГ U назовем число ребер в ИГ U. Количество листьев в ИГ U обозначим как $N(U) = |V|$, где V — библиотека ИГ U.

2. Хеширование методом цепочек

В [2] описан константный в среднем алгоритм поиска записей в БД. Этот алгоритм основан на хешировании методом цепочек. Алгоритм хорошо работает в среднем при условии статичности данных в БД, то есть, когда единственным типом запросов к БД является запрос на поиск в базе элемента x. В данном разделе будет показано,

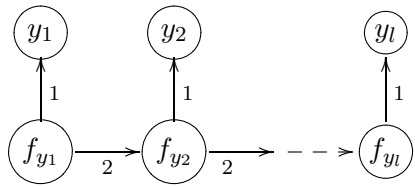
что на его основе можно организовать БД, предоставляющую пользователям также возможность вставки и удаления записи, не нарушая порядок величины сложности поиска.

Пусть множество записей X представляет собой полуотрезок $(0, 1]$.

Вначале определим вспомогательный тип информационных графов — *списки*. Базовым множеством будет $\mathcal{F}_0 = \langle \varnothing, \{f_a(x) \mid a \in X\}\rangle$, где

$$f_a(x) = \begin{cases} 1, & \text{если } x = a, \\ 2, & \text{если } x \neq a. \end{cases}$$

Будем считать, что $t(f_a) = t_f$ для любого a. Пусть, библиотека имеет вид $V = \{y_1, \ldots, y_l\}$. Тогда список $A(V)$ будет иметь следующий вид:



Видно, что $T(A(V), x) \leqslant N(A(V)) \cdot t_f$.

Вставку записи $y \in X$ в список A можно совершить следующим образом: сначала в списке производится поиск листа с пометкой y, и, если такого листа не найдено, осуществляется добавление в конец списка конструкции в виде переключательной вершины с пометкой f_y, листом с пометкой y и соответствующих дуг. Обозначим сложность вставки такой конструкции символом t_α. Получим

$$R(A(V), y) \leqslant N(A(V)) \cdot t_f + t_\alpha.$$

Удаление записи $y \in X$ в списке A состоит из поиска x и, возможно, последующего удаления листа вместе с соответствующей переключательной вершиной (обозначим эту сложность через t_β). Поэтому, для сложности удаления записи из списка справедливо неравенство:

$$S(A(V), y) \leqslant N(A(V)) \cdot t_f + t_\beta.$$

Через $]a[$ обозначим наименьшее целое не меньшее, чем a.

Определим базовое множество функций.

$$\begin{aligned} f_a(x) &= \begin{cases} 1, & \text{если } x = a \\ 2, & \text{если } x \neq a \end{cases}; \\ t(f_a) &= t_f \text{ для любого } a \in X; \\ g_i(x) &=]x \cdot i[; \\ t(g_i) &= t_g \text{ для любого } i \in \mathbb{N}; \\ \mathcal{F} &= \langle \varnothing, \{f_a(x) \mid a \in X\} \cup \{g_i(x) \mid i \in \mathbb{N}\}, t \rangle . \end{aligned}$$

Пусть m — натуральное число. Перейдем к определению ИГ U_m, соответствующего хешированию методом цепочек, где число m будем называть параметром хеш-функции.

Рассмотрим $\mathcal{X}_m = \{X_1, \ldots, X_m\}$ — разбиение X такое, что $X_i = \left(\frac{i-1}{m}, \frac{i}{m}\right]$, $m \in \{1, 2, \ldots, m\}$. Нетрудно видеть, что число $g_m(x)$ равно номеру элемента из разбиения, которому принадлежит x.

ИГ U_m над базисом $\mathcal{F}$ с библиотекой $V \subseteq X$ строится следующим образом. Возьмем вершину β_0 и объявим ее корнем графа U. Выпустим из β_0 m ребер, припишем им числа от 1 до m, объявим β_0 переключательной вершиной и припишем ей переключатель $g_m(x)$.

Пусть $V_i = X_i \cap V$, $l_i = |V_i|$, $i \in \{1, 2, \ldots, m\}$.

Конец ребра с номером i обозначим β_i.

Для всех таких i, что $V_i \neq \varnothing$, выпустим из вершины β_i список, то есть $A(V_i)$.

Очевидно, что дерево U_m решает задачу поиска.

Опишем алгоритм A_1^m вставки/удаления записи y в ИГ U_m. Вначале вычисляется значение переключателя $i = g_m(y)$ на запросе y. Затем в списке, выпущенном из вершины β_i, осуществляется вставка/удаление записи y.

Далее всюду поисковый запрос x, а также вставляемую запись y будем считать случайными величинами, распределенными равномерно на $(0, 1]$.

Утверждение 1. *Пусть ИГ $U_m \in \mathcal{U}(A_1^m, n)$ (то есть U_m построен по алгоритму A_1^m и содержит n листьев). Тогда*

$$T(U_m) \leqslant t_g + \frac{n}{m} \cdot t_f, \; R(U_m) = t_g + \frac{n}{m} \cdot t_f + t_\alpha, \; S(U_m) \leqslant t_g + \frac{n}{m} \cdot t_f + t_\beta.$$

Доказательство. Рассмотрим произвольный поисковый запрос $x \in (0, 1]$. Пусть i — такое число, что $x \in X_i$. Тогда,

$$T(U_m, x) = t_g + T(A(V_i), x) \leqslant t_g + l_i \cdot t_f,$$

откуда имеем

$$T(U_m) = \mathrm{M}_x T(U_m, x) = \int\limits_X T(U_m, x)\mathrm{P}(dx) =$$
$$= \sum_{i=1}^{m} \int\limits_{X_i} T(U_m, x) \cdot \mathrm{P}(dx) \leqslant \sum_{i=1}^{m} (t_g + l_i \cdot t_f) \cdot \mathrm{P}(X_i) =$$
$$= t_g + \frac{n}{m} \cdot t_f.$$

Для вставки произвольной записи y в U_m сначала проходим через переключатель $g_m(y)$ в вершину β_i, а затем производим вставку y в список $A(V_i)$. Таким образом получаем:

$$R(U_m, y) \leqslant t_g + l_i \cdot t_f + t_\alpha,$$

откуда получаем

$$R(U_m) = \mathrm{M}_x R(U_m, x) = \int\limits_X R(U_m, x)\mathrm{P}(dx) =$$
$$= \sum_{i=1}^{m} \int\limits_{X_i} R(U_m, x) \cdot \mathrm{P}(dx) \leqslant \sum_{i=1}^{m} (t_g + l_i \cdot t_f + t_\alpha) \cdot \mathrm{P}(X_i) =$$
$$= t_g + \frac{n}{m} \cdot t_f + t_\alpha.$$

Аналогично, для сложности удаления записи в ИГ U_m получаем

$$S(U_m, y) \leqslant t_g + l_i \cdot t_f + t_\alpha,$$
$$S(U_m) \leqslant t_g + \frac{n}{m} \cdot t_f + t_\beta.$$

Замечание. В случае, если в описываемом ИГ U_m, в котором $N(U_m) = n$, выполняется $n \leqslant c \cdot m$, где c — некоторая константа,

то средняя сложность основных операций по величине не больше, чем константа:

$$T(U_m) \leqslant t_g + \frac{n}{m} \cdot t_f \leqslant t_g + c \cdot t_f;$$
$$R(U_m) \leqslant t_g + \frac{n}{m} \cdot t_f + t_\alpha \leqslant t_g + c \cdot t_f + t_\alpha;$$
$$S(U_m) \leqslant t_g + \frac{n}{m} \cdot t_f + t_\beta \leqslant t_g + c \cdot t_f + t_\beta.$$

Из замечания к утверждению 1 следует, что для того, чтобы основные операции сохраняли константную сложность, необходимо по мере роста числа записей в БД увеличивать значение параметра хеш-функции m и соответствующим образом перестраивать ИГ U_m. При этом, процедура, выполняющая преобразование U_m, не должна «тормозить» эти основные операции.

Будем считать, что операция перестройки ИГ выполняется независимо от операций поиска и вставки в ИГ и параллельно им. Это значит, что параллельно с основным процессом, выполняющим поиск и вставку записей, работает второй процесс, периодически совершающий перестройку ИГ. Этот «служебный» процесс будет блокировать доступ к какой-то части вершин, пока он перестраивает их. Тогда параллельный ему основной процесс, дойдя до заблокированной вершины, будет вынужден ждать ее разблокировки. Таким образом, сложность поиска и вставки записи (которая понимается нами как время выполнения некой абстрактной процедуры) может увеличиться.

Итак, для достижения константной верхней оценки сложности основных операций необходимо, чтобы:

1) процедура перестройки, меняющая значение параметра хеш-функции m на m', заканчивалась до того, как количество записей в БД достигало значения $c \cdot m'$, где c — константа на протяжении всего функционирования БД;
2) увеличение средней сложности основных операций (с учетом возможной блокировки обрабатываемых ими вершин) произошло не более, чем на константу.

Опишем алгоритм выполнения такой перестройки. Пусть m_0 — значение параметра хеш-функции m на момент начала перестройки.

Пусть далее, m_1 — новое значение m такое, что $m_1 = k \cdot m_0$, где $k \in \mathbb{N}, k \geqslant 2$ — некоторое зафиксированное число, называемое коэффициентом расширения.

Возьмем новое разбиение отрезка $(0, 1]$:

$$\mathcal{X}_{m_1} = \{X'_1, \ldots, X'_{m_1}\};$$
$$X'_i = \left(\frac{i-1}{m_1}, \frac{i}{m_1}\right], i \in \{1, 2, \ldots, m_1\}.$$

Заметим, что это разбиение является измельчением для $\mathcal{X}_{m_0}$, то есть в каждом отрезке X_i содержится $k = m_1/m_0$ отрезков X'_j. Обозначим $V'_j = V \cap X'_j$.

Просмотрим по очереди каждую из вершин β_i, $i \in \{1, 2, \ldots, m_0\}$. Для каждой из них проделаем следующее.

Создадим вершину β'_i (не присоединяя ее к основному дереву — это будет корнем нового поддерева, которое обозначим U'_i). Выпустим из β'_i k дуг, помеченных числами $k(i-1)+1, \ldots, k(i-1)+k$, а вершины, в которые они ведут, обозначим соответственно $\gamma_{k(i-1)+1}, \ldots, \gamma_{k(i-1)+k}$. Припишем вершине β'_i переключатель $g_{m_1}(x)$. Можно считать, что перечисленные действия имеют сложность $t_\gamma \cdot k$, где t_γ — некоторая константа, обозначающая сложность создания одного листа.

Последующие вставки и удаления в полученном дереве будем осуществлять по тому же алгоритму, что и в основном ИГ.

После этого по очереди просмотрим листья списка $A(V_i)$ с корнем β_i и для каждого листа возьмем значение, приписанное ему и вставим это значение в дерево U'_i. В отличие от операции вставки записи в основное дерево, вставка записи y в U'_i будет иметь константную сложность, так как не требуется предварительно проверять существование листа с пометкой y. Заметим, что все перечисленные действия не требуют блокировки вершин в исходном дереве, то есть, параллельно в поддереве $A(V_i)$ может осуществляться поиск и вставка записей. Что касается удаления вершин в обрабатываемом списке, то наряду с удалением записи y из первоначального списка нужно удалять y и в построенном дереве U'_i.

Сложность такой операции будем считать равной $(t_f + t_g + t_\alpha) \cdot l'_i$, где l'_i — количество листьев i-го поддерева, обработанных в ходе пе-

рестройки[1]. Напомним, что t_f — сложность просмотра следующего листа в $A(V_i)$, t_g — сложность вычисления переключателя $g_{m_1}(x)$, а t_α — сложность вставки записи в список.

После построения дерева U_i' мы заменяем им дерево $A(V_i)$. Эту операцию можно считать элементарной[2], поэтому, припишем ей сложность t_ε. Заметим также, что выполнение этой операции требует блокировки поддерева с корнем β_i на время t_ε.

Наконец, когда перестройка завершена во всем дереве U_{m_0}, «склеим» все вершины β_i, а полученную вершину назначим корнем основного дерева (оставив приписанный ему переключатель $g_{m_1}(x)$) и в результате получим ИГ U_{m_1}. Мы также можем считать эту операцию элементарной и выполняемой за время t_δ[3].

Итак, если вся процедура перестройки ИГ U_{m_0} завершается (то есть «успевает» за ростом числа записей в базе), то, как нетрудно видеть, она имеет сложность не более, чем

$$t_\delta + \sum_{i=0}^{m_0} [kt_\gamma + (t_f + t_g + t_\alpha) l_i' + t_\varepsilon] \leqslant t_\delta + m_1 t_\gamma + m_0 t_\varepsilon + N'(U) \cdot (t_f + t_g + t_\alpha), \tag{1}$$

где $N'(U)$ — количество обработанных в процессе перестройки листьев в ИГ U_{m_0}.

В результате подключения второго, «служебного» процесса, получим новый алгоритм A_2, по которому производятся основные операции.

Каждая из рассматриваемых трех основных операций: поиска, вставки и удаления записи x, состоит в

1) вычислении переключателя $g_{m_0}(x)$, приписанного корню;

[1]Эта величина существует, если операция перестройки «успевает» за ростом количества записей в БД.

[2]В практической реализации такой операции может соответствовать простое изменение значения указателя на дочернюю вершину.

[3]Такое предположение основано на том, что в практической реализации предложенной модели набору вершин β_i и набору вершин β_j' логично сопоставить массив указателей на поддеревья. Тогда операция «склейки» всего лишь заключается в замене хеш-функции $g_{m_0}(x)$ на $g_{m_1}(x)$ и замене в ней адреса начала массива.

2) поиска, вставки или удаления в поддереве с номером $i = g_{m_0}(x)$. Здесь возможны следующие варианты:
   а) «служебный» процесс не дошел до поддерева с номером i, либо не закончил обработку этого поддерева. Как отмечалось выше, никакой блокировки основного процесса не происходит. Таким образом, соответствующая операция производится в списке $A(V_i)$.
   Если же рассматриваемой операцией является удаление записи и «служебный» процесс работает на текущем поддереве, то в случае удачного удаления записи в списке $A(V_i)$ необходимо также удалить x из U_i', что подразумевает вычисление переключателя $g_{m_1}(x)$ и удаление x из соответствующего списка;
   б) «служебный» процесс заблокировал (завершает обработку) поддерева i. Тогда основной процесс ждет завершения обработки, затем вычисляет переключатель $i' = g_{m_1}(x)$, и наконец производит поиск, вставку или удаление в списке $A(V_{i'}')$;
   в) «служебный» процесс уже обработал поддерево с номером i. Тогда производится вычисление переключателя $i' = g_{m_1}(x)$, а затем соответствующая операция в списке $A(V_{i'}')$.

Таким образом, можно сформулировать следующее утверждение, касающееся средней сложности основных операций в построенной модели.

Утверждение 2. *Пусть ИГ U построен по алгоритму A_2 и имеет n листьев. Пусть также параллельно работает служебный процесс, перестраивающий U. m_0 — значение параметра хеш-функции на момент начала перестройки. Тогда*

$$T(U) \leqslant 2t_g + t_\varepsilon + \frac{n}{m_0} \cdot t_f,$$
$$R(U) \leqslant 2t_g + t_\varepsilon + t_\alpha + \frac{n}{m_0} \cdot t_f,$$
$$S(U) \leqslant 2t_g + t_\varepsilon + 2t_\beta + 2\frac{n}{m_0} \cdot t_f.$$

Доказательство. Посчитаем сложность поиска в построенной БД. Пусть x — поисковый запрос. Как и в утверждении 1, будем считать, что i — номер отрезка X_i, в который попадает x, а $l_i = |V_i|$ — количество записей, лежащих на этом отрезке. Тогда для сложности поиска имеем

$$T(U, x) \leqslant 2t_g + l_i t_f + t_\varepsilon.$$

Для сложности вставки справедливо

$$R(U, x) \leqslant 2t_g + l_i t_f + t_\alpha + t_\varepsilon.$$

Наконец, для сложности удаления:

$$S(U, x) \leqslant 2t_g + 2l_i t_f + 2t_\beta + t_\varepsilon.$$

Тогда, используя выкладки, аналогичные изложенным в утверждении 1, получим:

$$\begin{aligned} T(U) &= \mathrm{M}_x T(U, x) \leqslant 2t_g + t_\varepsilon + \frac{n}{m_0} \cdot t_f; \\ R(U) &= \mathrm{M}_x R(U, x) \leqslant 2t_g + t_\varepsilon + t_\alpha + \frac{n}{m_0} \cdot t_f; \\ S(U) &= \mathrm{M}_x S(U, x) \leqslant 2t_g + t_\varepsilon + 2t_\beta + 2\frac{n}{m_0} \cdot t_f. \end{aligned}$$

Теорема 1. *Пусть существует такое положительное число ϵ, что операция вставки производится не чаще, чем один раз за время $t_g + t_\alpha + t_\gamma + t_f + \epsilon$. Пусть далее, операция перестройки начинается каждый раз, когда количество листьев в ИГ достигает значения параметра хеш-функции, и для коэффициента расширения выполнено*

$$k \geqslant \left\lceil \frac{t_g + t_\alpha + t_\gamma + t_f + t_\varepsilon + t_\delta + \epsilon}{\epsilon} \right\rceil. \tag{2}$$

Тогда в любой момент функционирования БД для сложностей основных операций справедливы неравенства

$$\begin{aligned} T(U) &\leqslant 2t_g + t_\varepsilon + k \cdot t_f, \\ R(U) &\leqslant 2t_g + t_\varepsilon + t_\alpha + k \cdot t_f, \\ S(U) &\leqslant 2t_g + t_\varepsilon + 2t_\beta + 2k \cdot t_f. \end{aligned}$$

Доказательство. Согласно (1) время перестройки не превышает величины

$$t \leqslant t_\delta + m_1 t_\gamma + m_0 t_\varepsilon + n(t_g + t_f + t_\alpha),$$

где n — количество листьев, обработанных в результате выполнения процедуры перестройки. Очевидно, n не может превышать количества записей в библиотеке до начала перестройки либо вставленных в U во время выполнения перестройки. Заметим, что n тем более не может превышать количества записей в БД к моменту окончания перестройки. С другой стороны, согласно требованию о частоте операции вставки, на то, чтобы количество таких записей достигло n, требуется времени

$$t \geqslant (t_g + t_\alpha + t_f + t_\gamma + \epsilon)(n - n_0),$$

где n_0 — количество записей в БД на момент начала перестройки. Получаем неравенство:

$$(t_g + t_\alpha + t_f + t_\gamma + \epsilon)(n - n_0) \leqslant t_\delta + m_1 t_\gamma + m_0 t_\varepsilon + n(t_g + t_f + t_\alpha),$$

следовательно

$$(t_\gamma + \epsilon)n \leqslant (t_g + t_\alpha + t_f + t_\gamma + \epsilon)n_0 + t_\delta + k m_0 t_\gamma + m_0 t_\varepsilon.$$

Заметим также, что, так как перестройка началась как только число записей достигло значения параметра хеш-функции, то $n_0 = m_0$. Тогда для количества записей после окончания перестройки получим:

$$n \leqslant m_0 \frac{t_g + t_\alpha + t_f + t_\gamma + t_\varepsilon + t_\delta + \epsilon + k t_\gamma}{t_\gamma + \epsilon}.$$

Из (2) можно вывести: $t_g + t_\alpha + t_f + t_\gamma + t_\varepsilon + t_\delta + \epsilon \leqslant k\epsilon$, тогда имеем:

$$n \leqslant m_0 \frac{k\epsilon + k t_\gamma}{\epsilon + t_\gamma} = m_0 \cdot k = m_1.$$

Последнее неравенство означает, что перестройка всегда заканчивается до того, как количество записей в БД достигает значения m_1 и нужно совершать очередную перестройку.

Это значит, что в любой момент времени $N(U) \leqslant k \cdot m$. Сравнив полученное неравенство и утверждение 2, получим

$$T(U) \leqslant 2t_g + t_\varepsilon + k \cdot t_f,$$
$$R(U) \leqslant 2t_g + t_\varepsilon + t_\alpha + k \cdot t_f,$$
$$S(U) \leqslant 2t_g + t_\varepsilon + 2t_\beta + 2k \cdot t_f,$$

что и составляет утверждение данной теоремы.

Итак, построенная модель динамической базы данных имеет константную среднюю сложность всех основных операций.

Однако, следует отметить, что сложность поиска и удаления записи в худшем случае вообще говоря линейна, что является существенным недостатком построенной модели.

Список литературы

[1] Гасанов Э. Э., Кудрявцев В. Б. Теория хранения и поиска информации. М.: ФИЗМАТЛИТ, 2002.

[2] Dumey A. Indexing for Rapid Random Access Memory Systems // Computers and Automation. (1956). 4, no. 12. 6–9.

[3] Кнут Д. Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск. М.: Мир, 1978.