Динамическая база данных, допускающая параллельную обработку произвольных потоков запросов

А. А. Плетнев

В данной работе рассматривается решение динамической задачи поиска идентичного объекта для произвольного потока запросов. Сложность параллельной обработки запросов над общей структурой данных заключается в избежании конфликтов, которые возникают при изменении одного и того же участка памяти. При этом считаем, что считывать из одного участка памяти нескольким процессам допустимо. Решение получено с помощью динамического информационного графа [1], [2], который в свою очередь является обобщением информационного графа [3] на случаи вставки и удаления записи.

Ключевые слова: динамические базы данных, информационный граф, конечный автомат, потоки запросов, параллельная обработка данных.

1. Введение

Исследования в области параллельных вычислений широко известны. Основная сложность параллельных вычислений заключается в считывании и записывании данных в одну и ту же область памяти. Различают несколько типов систем для параллельной обработки данных. Они различаются способами обращения к общей памяти:

- одновременное чтение, одновременная запись (ОЧОЗ). В данной системе разрешается одновременное чтение и одновременная запись разными процессами в общую память;
- одновременное чтение, эксклюзивная запись (ОЧЭЗ). В данной системе разрешается одновременное чтение из общей памяти, при этом одновременная запись разными процессами в общую память запрещена;

- эксклюзивное чтение, эксклюзивная запись (ЭЧЭЗ). В данной системе запрещается писать и читать из общей памяти разным процессам.

Подробно эти системы описаны в [4], [5]. Применительно к задаче поиска идентичных объектов, было предложено множество различных алгоритмов, использующих параллельные вычисления. Они основываются на таких известных структурах данных как сбалансированное бинарные дерево, красно-черное дерево, 2-3 дерево. В основном цель всех исследований заключается в добавлении несколько записей к структуре данных, используя параллельные вычисления, за наименьшее время. Более детально о полученных результатах можно прочитать в [6], [7].

В данной работе будем рассматривать немного другую постановку задачи. Под потоком запросов будем понимать бесконечную последовательность запросов на поиск, вставку и удаление к базе данных, которые поступают через равные промежутки времени — такты. При этом в один такт может поступить не более одного запроса. Так же считаем, что за один такт может быть выполнено любое локальное преобразование структуры базы данных, предназначенное для поддержания базы данных в актуальном состоянии. Другими словами каждый процесс, который перестраивает или ищет запрос, может сделать за один такт, только одно преобразование над базой данных, при этом на следующий такт к базе данных может поступить уже новый запрос. Кроме того будем считать, что у этой структуры данных есть сколь угодно большое количество доступных процессов. Каждый процесс обрабатывает один запрос, который может быть одним из трех типов: поиск, вставка или удаление. Ниже будет предложена ОЧЭЗ структура данных, решающая динамическую задачу поиска идентичного объекта для любого потока запросов.

Идея построения искомой структуры данных основывается на информационно-графовом подходе [3], в частности решение получено при помощи динамического информационного графа [1].

Автор благодарит профессора Э. Э. Гасанова за постановку задачи и помощь в работе.

2. Основные понятия и определения

В данной работе будут рассматриваться применение динамической информационно-графовой модели к задаче поиска идентичного объекта. Для ее реализации не будет нужно полное описание информационного графа (ИГ) и динамического информационного графа (ДИГ). Это позволит не загромождать работу излишним определениями и выкладками. Подробное описание ДИГ и ИГ можно найти в [1], [2] и [3].

Пусть X — множество запросов, Y — множество записей (объектов поиска), в данной работе будем считать, что $Y = X$. Пусть V — библиотека, причем $V \subseteq Y, |V| < \infty$. Будем говорить, что структура данных решает задачу поиска идентичного объекта, если для любого запроса x, $x \in X$, в ответ выдается множество $\{x\}$, если $x \in V$, и пустое множество в противном случае.

В дальнейшем под предикатом будем понимать функции, которые определены на множестве запросов X и принимают два значения 0 и 1. Понятие ИГ над множеством предикатов F определяется следующим образом. Берется конечная многополюсная ориентированная сеть. В ней выбирается некоторый полюс, который называется корнем. Остальные полюса называются листьями и им приписываются записи из Y, причем разным листьям могут быть приписаны одинаковые записи. Ребрам приписываются предикаты из множества F. Таким образом нагруженную многополюсную ориентированную сеть называем ИГ над базовым множеством F, где $F = \{f_j, j \in J\}$, J — множество индексов.

Определим понятие преобразования ИГ. Для этого введем несколько множеств. Пусть $M \subseteq \mathbb{N}$, тогда рассмотрим следующее множество функций преобразования индексов $\mathcal{C}$,

$$\mathcal{C} = \{c_m : J^{d_{1,m}} \times Y^{d_{2,m}} \to J^{a_m} \times Y^{1-a_m},$$
$$m \in M;\ d_{1,m}, d_{2,m} \in \mathbb{N};\ a_m \in \{0,1\}\}.$$

Введем два счетных множества переменных $\mathcal{P}_J, \mathcal{P}_L$:

- $\mathcal{P}_J = \{p_J^j\}$, $j \in \mathbb{N}$, p_J^j принимают значения из J;
- $\mathcal{P}_L = \{p_L^l\}$, $l \in \mathbb{N}$, p_L^l принимают значения из Y.

Рассмотрим произвольный ИГ U. Обозначим $F(U)$ — множество индексов предикатов, $Y(U)$ — множество записей, входящих в ИГ U.

Заменим нагрузку всех входящих в U вершин и ребер по следующему правилу.

- Каждый предикат с индексом j, $j \in F(U)$ заменим на некоторую переменную из $\mathcal{P}_J$. Причем эта замена задается инъективной функцией Ω_F, $\Omega_F : F(U) \to \mathcal{P}_J$. Это означает, что предикаты с различными индексами j заменим на различные переменные из $\mathcal{P}_J$, а с одинаковыми индексами j заменим на одинаковые переменные из $\mathcal{P}_J$.
- Каждую запись y, приписанную листовой вершине, $y \in Y$, заменим на переменную из $\mathcal{P}_L$. Причем эта замена задается инъективной функцией Ω_Y, $\Omega_Y : Y(U) \to \mathcal{P}_L$.

Рассмотрим множество $\mathcal{P}(U)$, $\mathcal{P}(U) = \Omega_F(F(U)) \cup \Omega_Y(Y(U))$, то есть $\mathcal{P}(U)$ есть множество переменных, которые получились в результате замены нагрузки всех вершин и ребер из U. Рассмотрим функцию Ω,

$$\Omega : F(U) \cup Y(U) \to \mathcal{P}(U),$$

где $\Omega = \Omega_F$ на множестве $F(U)$ ($\Omega \mid_{F(U)} = \Omega_F$), $\Omega \mid_{Y(U)} = \Omega_Y$. Из определения Ω следует, что Ω — биективная функция, поэтому существует функция Ω^{-1}. Обозначим $I = \Omega^{-1}$ и будем называть эту обратную функцию *интерпретацией*, возникшей в результате замены индексов на переменные.

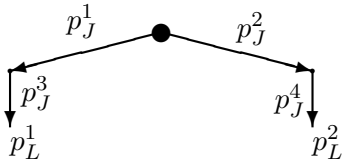

Рис. 1. Пример простого шаблона.

После этого сопоставления (замены нагрузки вершин и ребер на переменные в ИГ U) получим нагруженный граф. Выделим в нем множество вершин $V_{\mathcal{T}}$, которые назовем вершинами прикрепления, и занумеруем их числами от 1 до $|V_{\mathcal{T}}|$. Полученный граф назовем *простым шаблоном* $\mathcal{T}$. При этом будем писать $\mathcal{T} = \mathcal{T}(U, I, V_{\mathcal{T}})$, когда хотим подчеркнуть, что $\mathcal{T}$ получен из ИГ U и I, где I — интерпретация, возникшая в результате замены индексов на переменные,

$V_{\mathcal{T}}$ — упорядоченное множество вершин прикрепления. Пример простого шаблона изображен на рисунке 1 (жирным кружком обозначена единственная вершина прикрепления).

Рассмотрим ИГ $U^{'}$, выделим в нем множество вершин $V_{U'}$, которые назовем вершинами прикрепления, и занумеруем их числами от 1 до $|V_{U'}|$ (вершины прикрепления образуют упорядоченное множество). Будем говорить, что ИГ $U^{'}$ и простой шаблон $\mathcal{T}$ *согласованы*, если выполнены следующие условия:

- $U^{'}$ и $\mathcal{T}$ совпадают как графы, и если в ИГ $U^{'}$ встречаются одинаковые индексы предикатов и записи, то в соответствующих местах шаблона $\mathcal{T}$ находятся одинаковые переменные из $\mathcal{P}_J$ и $\mathcal{P}_L$;

- i-ая вершина прикрепления ИГ $U^{'}$ совпадает с i-ой вершиной прикрепления простого шаблона $\mathcal{T}$, $i = 1, \ldots, |V_{U'}|$ и $|V_{U'}| = |V_{\mathcal{T}}|$.

То есть $\mathcal{T} = \mathcal{T}(U^{'}, I, V_{\mathcal{T}})$ для некоторой интерпретации I, и будем писать $\mathcal{T} = \mathcal{T}(U^{'}, I, V_{U'})$, когда хотим подчеркнуть, что ИГ $U^{'}$ и простой шаблон $\mathcal{T}$ согласованы.

Рассмотрим простой шаблон $\mathcal{T}_1$ и заменим в нем каждую переменную, на формулу над множество функций $\mathcal{C}$ и каким-либо множеством переменных $M \subseteq \mathcal{P}_J \cup \mathcal{P}_L$. В результате, полученный граф назовем *шаблоном* $\mathcal{T}_2$. При этом будем писать $\mathcal{T}_2 = \mathcal{T}_2(\mathcal{T}_1, M, V_{\mathcal{T}_1})$, когда хотим подчеркнуть, что $\mathcal{T}_2$ получен из простого шаблона $\mathcal{T}_1$, множества переменных M и упорядоченного множеств вершин прикрепления $V_{\mathcal{T}_1}$.

На рисунках запись $c_i(M)$ будет означать формулу над множество функций $\mathcal{C}$ и множеством переменных из M.

Пример шаблона приведен на рисунке 2 (жирным кружком обозначена единственная вершина прикрепления).

Рассмотрим простой шаблон $\mathcal{T}_1 = \mathcal{T}_1(U, I, V_{\mathcal{T}_1})$. Обозначим множество входящих в него переменных $M(\mathcal{T}_1)$. Пусть $p_L^0 \in \mathcal{P}_L \setminus M(\mathcal{T}_1)$.

Преобразованием R назовем тройку

$$R = (\mathcal{T}_1(U, I, V_{\mathcal{T}_1}), \mathcal{T}_2(\mathcal{T}, M(\mathcal{T}_1) \cup \{p_L^0\}, V_{\mathcal{T}}), \xi),$$

где $\mathcal{T}_1$ — простой шаблон, $\mathcal{T}_2$ — шаблон (полученный из простого шаблона $\mathcal{T}$) в формулах которого встречаются только переменные из

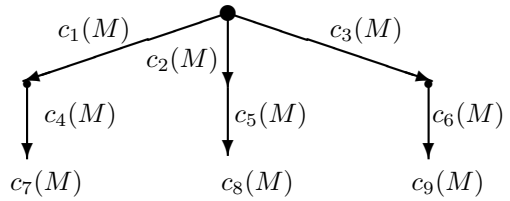

Рис. 2. Пример шаблона $\mathcal{T}_2$.

множества $M(\mathcal{T}_1) \cup p_L^0$, $V_\mathcal{T}$ упорядоченное множество вершин прикрепления простого шаблона $\mathcal{T}$, ξ — биективная функция сопоставления вершин прикрепления ($\xi : V_{\mathcal{T}_1} \to V_\mathcal{T}$). Левой частью преобразования R будем называть простой шаблон $\mathcal{T}_1$.

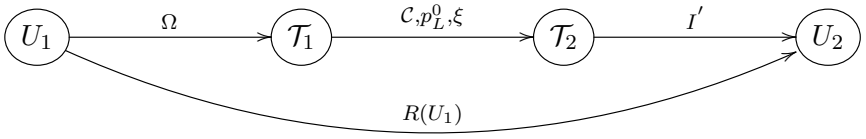

Рис. 3. Преобразование ИГ U_1 в ИГ U_2.

Пусть ИГ U_1 и простой шаблон $\mathcal{T}_1$ согласованы, то есть $\mathcal{T}_1 = \mathcal{T}_1(U_1, I, V_{U_1})$ для некоторой интерпретации I, и пусть $I^{'}$ — интерпретация множества переменных $M(\mathcal{T}_1) \cup p_L^0$, причем $I^{'} = I$ на множестве переменных $M(\mathcal{T}_1)$. Тогда применением преобразования $R = (\mathcal{T}_1(U, I, V_{U_1}), \mathcal{T}_2(\mathcal{T}, M(\mathcal{T}_1) \cup p_L^0, V_\mathcal{T}), \xi)$ к ИГ U_1 назовем ИГ U_2, получающийся из шаблона $\mathcal{T}_2(\mathcal{T}, M(\mathcal{T}_1) \cup p_L^0, V_\mathcal{T})$ подстановкой вместо каждой формулы значения данной формулы в интерпретации $I^{'}$. Заметим, что упорядоченное множество вершин прикрепления V_{U_2} в ИГ U_2 однозначно соответствует упорядоченному множеству вершин прикрепления V_{U_1} в ИГ U_1, так как ξ — биективная функция, и простой шаблон $\mathcal{T}_1$ согласован с ИГ U_1.

Результат применения преобразования R к ИГ U_1 будем обозначать $R(U_1) = U_2$. Преобразование ИГ U_1 в ИГ U_2 изображено на рисунке 3.

Следующие понятия будут введены для определения *кода информационного графа на запросе*. Код информационного графа на запросе будет использоваться для описания передвижения автомата по вершинам ИГ.

Рассмотрим ИГ U, $N(U)$ — множество входящих в него вершин. Пусть $\beta, \beta^{'} \in N(U)$, тогда путем $\pi(\beta, \beta^{'})$ назовем последовательность вершин и ребер ИГ U, которая начинается в вершине β, заканчивается в вершине $\beta^{'}$, и в этой последовательности два любых соседних элемента инциденты. Длиной пути $l(\pi(\beta, \beta^{'}))$ назовем количество ребер, участвующих в нем. Пусть $\Pi(\beta, \beta^{'})$ — множество всех путей из β в $\beta^{'}$. Тогда расстоянием между вершинами $\beta, \beta^{'} \in N(U)$ назовем минимальную из всех путей длину $l(\pi(\beta, \beta^{'})), \pi(\beta, \beta^{'}) \in \Pi(\beta, \beta^{'})$, и обозначим $(\beta, \beta^{'}) = \min\{l(\pi(\beta, \beta^{'})) : \pi(\beta, \beta^{'}) \in \Pi(\beta, \beta^{'})\}$. Эксцентриситетом вершины $\beta \in N(U)$ назовем число $\varepsilon(\beta) = \max\limits_{\beta^{'} \in N(U)} (\beta, \beta^{'})$. *Радиусом* ИГ U назовем число $r(U) = \min\limits_{\beta \in N(U)} \varepsilon(\beta)$.

Через $\mathcal{G}(N, R)$, $N, R \in \mathbb{N}$ обозначим класс ИГ радиуса не больше R таких, что количество ребер инцидентных любой вершине графа не превосходит N.

Рассмотрим ИГ U, $E(U)$ — множество входящих в него ребер. Ребро инцидентное вершинам $\beta_1, \beta_2 \in N(U)$ будем обозначать $e(\beta_1, \beta_2)$.

Рассмотрим ИГ U_2 и его подграф (подмножество ребер и инцидентным им вершин) U_1 (пишем $U_1 \subseteq U_2$). Множество $\Sigma(U_1, U_2) = \{\beta : \beta \in N(U_1)$ и $\exists \beta_1 \in N(U_2 \backslash U_1), \exists\ \beta_2 \in N(U_1),\ e(\beta_1, \beta) \in E(U_2),\ e(\beta_2, \beta) \in E(U_1)\}$ назовем множеством *граничных вершин информационных графов* U_1 *и* U_2.

Пусть $U_1, U_2 \in \mathcal{G}(N, \infty)$ и $U_1 \subseteq U_2$ ($N < \infty$). Назовем *кодом вершины на запросе* $x \in X$ *относительно пары* (U_1, U_2) пару (k_1, k_2), где $k_1 = 0$, если вершина корень ; $k_1 = 1$, если вершина листовая и $k_1 = 2$ в остальных случаях; $k_2 \in \{0, 1\}$, $k_2 = 1$, если вершина принадлежит множеству граничных вершин $\Sigma(U_1, U_2)$ и $k_2 = 0$, если не принадлежит.

Пусть количество различных типов предикатов конечно и равно K_T. Сопоставим им в биективном соответствие натуральные числа от 1 до K_T, тогда *кодом ребра на запросе* $x \in X$ *относительно пары* (U_1, U_2) назовем пару (k_1^r, k_2^r), где $k_r^1 \in \{1, \ldots, K_T\}$ это тип предиката соответствующее этому ребру; $k_2^r \in \{0, 1\}$ значение предиката на запросе x .

Рассмотрим ИГ U, в нем вершины и ребра имеют нагрузку. Граф $K(U)$, полученный из ИГ U удалением нагрузок вершин и ребер, назовем *каркасом* ИГ U.

Кодом ИГ U_1 относительно ИГ U_2 на запросе, назовем нагруженный каркас $K(U_1)$, где нагрузка каждой вершины $K(U_1)$ это код данной вершин на данном запросе относительно пары (U_1, U_2), а на нагрузка каждого ребра $K(U_1)$ это код данного ребра на данном запросе относительно пары (U_1, U_2). Через $\mathcal{K}(N, R)$ обозначим множество кодов ИГ U_1 относительно ИГ U_2, где U_1 пробегает класс ИГ $\mathcal{G}(N, R)$, U_2 пробегает класс ИГ $\mathcal{G}(N, R+1)$ и $U_1 \subseteq U_2$. Понятно, что $|\mathcal{K}(N, R)| < \infty$.

Пусть $N, R \in \mathbb{N}$, $\mathcal{P}$ — некоторое конечное множество преобразований, шаблоны которых порождены ИГ из $\mathcal{G}(N, R)$, и $P_0 \in \mathcal{P}$, где P_0 — тождественное преобразование такое, что $P_0(U) = U$ для произвольного ИГ U из $\mathcal{G}(N, R)$.

Пусть $N(U_1)$ множество вершин ИГ U_1. Рассмотрим множество вершин $\mathfrak{B} \subseteq N(U_1)$ и обозначим через $U(\mathfrak{B}, U_1)$ — ИГ, полученный из ИГ U_1 как подграф на вершинах $\mathfrak{B}$. Под подграфом на вершинах $\mathfrak{B}$ здесь понимается множество вершин $\mathfrak{B}$ и множество всех инцидентных им ребер.

Пусть $\mathcal{A}$ — конечный автомат (подробно об определении конечного автомата можно прочитать в [8]). Будем считать, что автомат $\mathcal{A}$ перемещается по вершинам ИГ $U_2 \in \mathcal{G}(N, \infty)$ и в каждый момент времени обозревает окрестность U_1 текущей вершины радиуса R, то есть будем считать, что входным символом автомата $\mathcal{A}$ является код обозреваемой окрестности относительно U_2. Понятно, что эти коды будут принадлежать $\mathcal{K}(N, R)$ и значит входным алфавитом автомата $\mathcal{A}$ является конечное множество $\mathcal{K}(N, R)$. Результатом этого автомата $\mathcal{A}$ на обозреваемой окрестности текущей вершины будет некоторое преобразование этой окрестности и перемещение в некоторую вершину преобразованной окрестности. Тем самым выходным символом автомата $\mathcal{A}$ будет пятерка $b = (\mathfrak{B}, \pi, R, \beta, e)$, где

- $\mathfrak{B} \subseteq N(U_1)$ определяет множество вершин обозреваемой окрестности к которому будет применено преобразование;
- π — функция нумерации граничных вершин $U(\mathfrak{B}, U_1)$ и U_2 (содержательно это множество граничных вершин $U(\mathfrak{B}, U_1)$ и U_1 объединенное со множеством вершин в коде которых $k_2 = 1$ и одновременно принадлежащих $U(\mathfrak{B}, U_1)$). Пронумерованные граничные вершины назовем вершинами прикрепления;
- R — преобразование из конечного множества $\mathcal{P}$ применяемое к ИГ $U(\mathfrak{B}, U_1)$, причем ИГ $U(\mathfrak{B}, U_1)$ с заданной функцией π

нумерацией вершин прикрепления согласован с левой частью преобразования R;

- $\beta \in N(R(U(\mathfrak{B}, U_1))) \cup \{U_1 \setminus U(\mathfrak{B}, U_1)\}$ — следующая текущая вершина автомата $\mathcal{A}$;
- $e \in \{0, 1, 2, 3\}$ отвечает за продолжение и выдачу ответа автоматом $\mathcal{A}$. Автомат еще не нашел ответ и продолжает функционирование ($e = 0$), либо завершает функционирование и возвращает информацию о том, что искомая запись найдена ($e = 1$), либо завершает функционирование с информацией, что искомой записи нет ($e = 2$), либо завершает функционирование с информацией, что запись уже есть, и он не может ее вставить ($e = 3$).

Множество всех таких выходных символов b образует выходной алфавит B автомата $\mathcal{A}$.

Пусть U — ИГ над базовым множеством F из класса $\mathcal{G}(N, \infty)$, тогда пару $(\mathcal{A}, U)$ назовем динамическим информационным графом (ДИГ) типа (N, R) над $\mathcal{F}{=}\langle F, \mathcal{P}\rangle$ и обозначим $\mathcal{D} = (\mathcal{A}, U)$.

Обобщим понятие запроса к базе данных на случай вставки, удаления и пустого действия (запрос не требует действий для него). Для этого рассмотрим алфавит $A = \{\text{П}, \text{В}, \text{У}, \Lambda\}$. Обобщенным запросом назовем пару $(a, x), (a, x) \in \tilde{X}$, где $\tilde{X} = A \times X$. В данной работе рассматривается задача поиска идентичного объекта, поэтому множество запросов и множество записей одинаково и равно X. При этом второй аргумент обобщенного запроса (буква из алфавита A) будет означать необходимое действие, которое нужно осуществить для запроса: П — поиск, В — вставка, У — удаление, Λ — ничего не делать. Далее везде под словом запрос будем понимать обобщенный запрос.

Потоком запросов H, $H : \mathbb{N} \to \tilde{X}$, назовем последовательность запросов, поступающих к базе данных через равные промежутки времени — такты. Другими словами поток запросов H означает, что к базе данных в i-ый такт времени будет подан запрос $H(i)$. Обозначим множество всех потоков через $\mathcal{H}$.

Рассмотрим функцию W, $W : \mathcal{H} \to A^\infty$ (через A^∞ обозначено множество всех сверхслов в алфавите A) сопоставляющую потоку запросов H сверхслово $W(H)$ из алфавита A, причем i-ая буква сверхслова $W(H)$ (обозначим ее через $W(H)(i)$) такова, что $H(i) = (W(H)(i), x_i)$.

Для обработки потоков запросов введем понятие *многоавтоматный динамический информационный граф* (МДИГ) и обозначим его $\mathcal{D}_M = (\mathcal{A}, U)$. Пусть дано бесконечное множество копий автомата $\mathcal{A}$. Будем считать, что несколько копий автомата $\mathcal{A}$ могут одновременно обслуживать несколько запросов. Например, пусть имеется поток запросов на поиск $H = (\Pi, x_1), (\Pi, x_2), (\Pi, x_3), \ldots$, тогда при наличии 3 и более копий автомата $\mathcal{A}$ МДИГ сможет параллельно обслужить три запроса x_1, x_2, x_3 на поиск, не дожидаясь завершения каждого ДИГ в отдельности. Если в потоке запросов H встречаются запросы на изменение базы данных (вставка и удаление), то при параллельной обработки этих запросов возможны конфликты преобразований (ИГ общий для всех автоматов).

Определим *функционирование МДИГ* $\mathcal{D}_M, \mathcal{D}_M = (\mathcal{A}, U)$, *типа* (N, R) *над* $\mathcal{F}, \mathcal{F} = \langle F, \mathcal{P} \rangle$, *для потока запросов* H *из* $\mathcal{H}$. Считаем, что время применения любого преобразования R из $\mathcal{P}$, ровно 1 такт. Тем самым автомат $\mathcal{A}$ за один такт делает ровно одно преобразование над ИГ U. Заметим, что мы всегда можем выбрать единицу измерения времени (такт), что это будет выполнено (например как максимум из времен всех преобразований). Так же мы предполагаем, что такт запроса совпадает с тактом работы автомата.

После каждого такта работы МДИГ — ИГ U может изменяться, поэтому будем рассматривать $U = U(t)$ $(t \in \mathbb{N} \cup \{0\})$. В начальный момент функционирования $U = U(0)$.

Рассмотрим запрос $H(i), H(i) = (a_i, x_i)$, который поступил для обработки к МДИГ $\mathcal{D}_M, \mathcal{D}_M = (\mathcal{A}, U)$, $U = U(i)$ в i-ый такт, $i \geqslant 1$. МДИГ действует в зависимости от типа запроса a_i. Если $a_i = \Lambda$ (ничего не делать), то МДИГ ничего не делает для этого запроса и $U(i+1) = U(i)$, иначе МДИГ функционирует как ДИГ для данного запроса.

Функционирование ДИГ $\mathcal{D}$ на запросе определяется следующим образом. В начальный момент текущей вершиной объявляется корень ИГ $U = U(i)$. Рассматривается ИГ U_1 как подграф U, с центром в текущей вершине и радиуса R. На вход автомата $\mathcal{A}$ подается код ИГ U_1 относительно ИГ U на запросе. Пусть $b = (\mathfrak{B}, \pi, R, \beta, e) \in B$ выходная буква автомата $\mathcal{A}$, тогда ИГ U меняется по следующему правилу. В ИГ U ИГ $U(\mathfrak{B}, U_1)$ заменяется на ИГ U', полученной в результате применения преобразования R к ИГ $U(\mathfrak{B}, U_1)$ $(R(U(\mathfrak{B}, U_1)) = U')$ и «прикрепляется» к ИГ U посредством функции π и функции сопо-

ставления вершин прикрепления ξ из преобразования R. После этого текущее положение автомата $\mathcal{A}$ меняется на β и в зависимости от значения последней компоненты выходной буквы b ДИГ $\mathcal{D}_M$ либо продолжает функционирование ($e = 0$), либо завершает функционирование и возвращает информацию о том, что искомая запись найдена (вставлена или удалена)($e = 1$), либо завершает функционирование с информацией, что записи нет ($e = 2$), либо завершает функционирование с информацией, что запись уже есть, и он не может ее вставить ($e = 3$).

Теперь определим понятие *сложности* МДИГ. Для любого i из $\mathbb{N}$ через $T(\mathcal{D}_M, H(i))$ обозначим количество тактов необходимое для обработки запроса $H(i)$ динамическим информационным графом $\mathcal{D}_M$.

3. Постановка задачи и результаты

Пусть H из $\mathcal{H}$ — поток запросов, $H : \mathbb{N} \to \tilde{X}$, где $\tilde{X} = A \times X$, — множество обобщенных запросов. Рассмотрим функцию множества записей $V : \{\mathbb{N} \cup \{0\}\} \times \mathcal{H} \to 2^X$, которая удовлетворяет следующим условиям

$$V(i,H) = \begin{cases} \varnothing, & \text{если } i = 0; \\ V(i-1,H), & \text{если } i > 0 \text{ и } W(H)(i) \in \{\Lambda, \Pi\}; \\ V(i-1,H) \cup \{x\}, & \text{если } i > 0 \text{ и } H(i) = (\text{В}, x); \\ V(i-1,H) \setminus \{x\}, & \text{если } i > 0 \text{ и } H(i) = (\text{У}, x). \end{cases}$$

Функция V составляет каждому такту i и потоку запросов H — множество записей, которые образуют базу данных после завершения функционирования МДИГ для всех запросов до такта i включительно, если обработка любого запроса происходила бы мгновенно (за 1 такт).

Рассмотрим поток запросов H из $\mathcal{H}$ и произвольный такт $i, i \in \mathbb{N}$. Будем говорить, что МДИГ решает динамическую задачу поиску идентичного объекта (ДЗПИО), если выполняются следующие условия

- если $H(i) = (\Pi, x)$, то результатом функционирования МДИГ должен быть $\{x\}$, если $x \in V(i,H)$ и пустое множество в противном случае;

- если $H(i) = (\text{В}, x)$, то для любого запроса на поиск (П, z), поступившего в такт $i+1$, результат функционирования МДИГ должен быть $\{z\}$, если $z \in V(i+1, H) = V(i, H) \cup \{x\}$, и пустое множество в противном случае;
- если $H(i) = (\text{У}, x)$, то для любого запроса на поиск (П,z), поступившего в такт $i+1$, результат функционирования МДИГ должен быть $\{z\}$, если $z \in V(i+1, H) = V(i, H) \setminus \{x\}$, и пустое множество в противном случае.

Пусть

$$f_a(x) = \begin{cases} 1, & \text{если } x = a; \\ 0, & \text{иначе} \end{cases}.$$

$$F = \{f_a(x) : a \in \mathbb{R}\}.$$

Введем базовое множества преобразований $\mathcal{R}$. Базовое множество $\mathcal{R}$ состоит из 12 преобразований, изображенных на рисунках 5, 7 и 8–12. Преобразования будут описаны в доказательстве теоремы 1.

В качестве базового множества для МДИГ будем рассматривать множество

$$\mathcal{F} = \langle F, \mathcal{R} \rangle. \tag{1}$$

Через $]x[$ будем обозначать целую часть сверху от x (наименьшее целое число, которое не меньше x). Справедлива следующая теорема.

Теорема 1. *Существует МДИГ $\mathcal{D}_M = (\mathcal{A}, U)$ типа (2,2) над базовым множеством $\mathcal{F}$, определяемым соотношением (1), который решает ДЗПИО для любого потока запросов H из $\mathcal{H}$, причем для любого натурального i и для любого H из $\mathcal{H}$ выполняется*

$$T(\mathcal{D}_M, H(i)) \leqslant]|V(i, H)|/2[, \text{ если } W(H)(i) = \text{П}.$$

Теорема 2. *Для любого натурального R, не существует МДИГ типа $(1, R)$, решающего ДЗПИО для любого потока запросов.*

В качестве следствия этих теорем, приведем таблицу, изображенную на рисунке 4. Строкам этой таблицы будут соответствовать значения N, а столбцам — значения R. На пересечении i строки и j столбца будет стоять плюс(минус), если существует(не существует) МДИГ типа (i, j) решающий ДЗПИО для любого потока запросов.

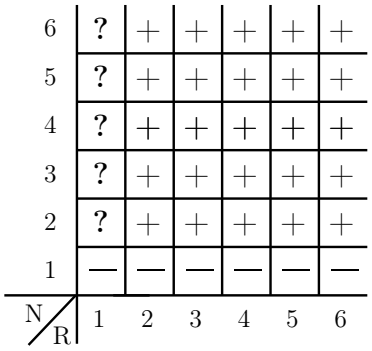

Рис. 4. Таблица существования МДИГ, решающего ДЗПИО для любого потока запросов. Плюс, минус и вопрос соответственно означают, что МДИГ типа (N, R) существует, не существует, не известно.

Некоторый элементы таблицы будут заполнены знаками вопроса, это означает, что пока не известно для этих N и R существует ли МДИГ это типа, решающий ДЗПИО для любого потока запросов или не существует.

4. Доказательство теорем 1 и 2

Докажем теорему 1. Сначала опишем базовое множество преобразований

$$\mathcal{R} = \{R_1, R_2, \ldots, R_{12}\},$$

которые будет применять автомат $\mathcal{A}$ в зависимости от входной окрестности.

Все преобразования можно разделить на три типа: преобразования на вставку, удаление и перемещение. Цель МДИГ создать динамический список, поэтому алгоритм вставки нового элемента в список будет следующим. Автомат должен пройти по всему списку и если не обнаружит вставляемой записи — вставить запись в конец списка. Поэтому преобразований на вставку будет три, и они изображены на рисунках 5 и 7. Преобразование, изображенное на рисунке 5, соответствует случаю пустой базы данных. Преобразования, изображенные на рисунке 7, соответствует случаю добавлению новой записи в конец списка.

формальное преобразование пример преобразования ИГ

Рис. 5. Вставка для случая пустой базы данных.

На рисунке 5 слева направо изображены соответственно простой шаблон $\mathcal{T}_1$, шаблон $\mathcal{T}_2$, ИГ U_1 (ИГ к которому применяем преобразование), и последним на рисунке изображен пример ИГ U_2. Здесь $M(\mathcal{T}_1) = \varnothing$, $p_L^0 \in \mathcal{P}_L \setminus M(\mathcal{T}_1)$, $I(p_L^0) = y$ (запрос на вставку), $r_J(p_L^0) = p_J$, $r_L(p_L^0) = p_L^0$, где $I(p_J) = f_y$.

Так же буква «к» находящаяся рядом с вершиной обозначает корень, вершина обведенная кругом означает центр рассматриваемой окрестности (текущее положение автомата $\mathcal{A}$), а так же новое текущее положение автомата $\mathcal{A}$, которое будет соответствовать компоненте β выходного символа $b \in B$ автомата $\mathcal{A}$. При этом если в правой части преобразования нет вершины, обведенной кругом, то это означает, что автомат завершает функционирование. Например, автомат на поиск завершает функционирование, когда он находит запись, или не находит, но при этом находиться в конце списка.

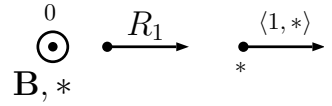

Рис. 6. Преобразование R_1 на вставку, с точки зрения автомата.

На рисунке 6, изображено преобразование R_1 с точки зрения автомата. Автомат на вход получает код ИГ на запросе $*$. Вершина, которой соответствует код 0, является корнем. Автомат дает инструкцию ИГ, что данную окрестность нужно заменить на окрестность, соответствующей правой части преобразования. В этой окрестности есть одно предикатное ребро. Этому ребру присвоен предикат первого типа $\langle 1, * \rangle$, который будет интерпретировать как f_*.

В дальнейшем будем описывать все преобразования с точки зрения автомата. Так же будем использовать сокращенную запись преобразования. В преобразованиях не будем указывать код вершин, а

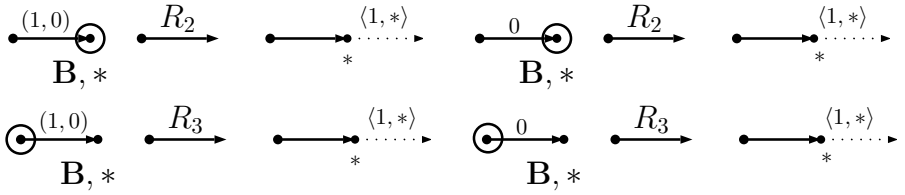

Рис. 7. Преобразование R_2 и R_3 на вставку, с точки зрения автомата.

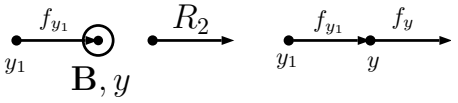

Рис. 8. Преобразование R_2 на вставку, с точки зрения ИГ.

так же тип предикатов. При этом в преобразованиях будем выделять ребра жирным, если автомат не меняет это ребро. Ребро будем изображать пунктиром, если автомат удаляет или вставляет это ребро. Это поможет понять, какие ребра автомат преобразовывает, а какие ребра нет. Пример сокращенного преобразования изображен на рисунке 7.

Преобразования R_2 и R_3 добавляют запрос в конец списка. Они различаются только положением автомата. Пример изменения ИГ для преобразования R_2, изображен на рисунке 8.

Для большей ясности смысла введенных преобразований на вставку, приведем пример ИГ, который получен после нескольких преобразований на вставку. ИГ после последовательности запросов (В,y_1),(В,y_2),(В,y_3), (В,y_4),(В,y_5), изображен на рисунке 9.

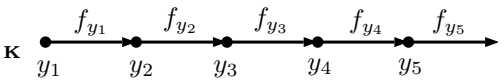

Рис. 9. Пример ИГ после нескольких вставок.

Преобразования на удаление изображены на рисунках 10–11. Преобразование R_4 применяется, если удаляется последняя запись (самая дальняя от корня) в списке. Преобразование R_5 удаляет сосед-

нюю справа запись, при этом вершина $\mathbf{v}_3$ может быть не последней в списке. В этом преобразование удаляется ребро ($\mathbf{v}_1$, $\mathbf{v}_2$), а ребро ($\mathbf{v}_2$, $\mathbf{v}_3$) меняет начало с $\mathbf{v}_2$ на $\mathbf{v}_1$. При этом черточкой на рисунке показано совпадение нагрузок ребер, то есть ребру ($\mathbf{v}_2$, $\mathbf{v}_3$) будет присвоен предикат, который соответствовал ребру ($\mathbf{v}_1$, $\mathbf{v}_2$) до преобразования.

Преобразование R_6 удаляет ребро, инцидентное вершине $\mathbf{v}_1$, при этом саму вершину не удаляет. Это преобразование удаляет вершину $\mathbf{v}_2$, а в качестве начала инцидентных ей ребер устанавливает вершину $\mathbf{v}_1$. Черточками показано соответствие нагрузок ребер, одинаковому количеству черточек соответствуют одинаковые предикаты, то есть нагрузка ребра $(\mathbf{v}_2, \mathbf{v}_3)$ совпадает с нагрузкой ребра $(\mathbf{v}_1, \mathbf{v}_3)$. Так же на рисунке 11 через «звездочки» со штрихами, обозначены записи, соответствующие вершинам. Преобразование R_6 меняет запись, находящуюся в вершине $\mathbf{v}_1$ на запись, находившуюся в вершине $\mathbf{v}_2$.

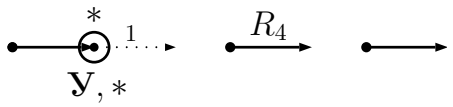

Рис. 10. Удаление последней записи.

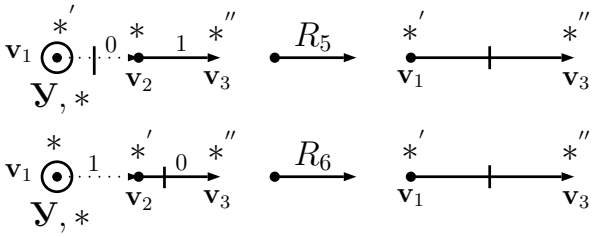

Рис. 11. Удаление в общем случае.

Преобразования $R_7, \ldots, R_{12}$ — это преобразования на перемещение по ИГ. Эти преобразования изображены на рисунке 12. R_7 — это перемещение автомата по ИГ в случае, если запись еще не найдена. Заметим, что автомат по возможности передвигается сразу на два ребра. R_8 и R_9 применяются автоматами для запроса на поиск и вставку, если запись найдена. Для запросов на удаление в этом случае применяется преобразование R_4, R_6 или R_5. В преобразованиях

R_8, R_9 автомат на поиск завершает функционирование с сигналом $e = 1$ (запись найдена), а автомат на вставку завершает свое функционирование $e = 3$ (запись уже есть, ее нельзя вставить). R_{10} и R_{11} применяется для автоматов на поиск и удаление, если запись не найдена, при этом автомат завершает функционирование с сигналом $e = 2$ (запись не найдена). Для автоматов на вставку применяются преобразования R_3 и R_2 соответственно. Преобразование R_{12} применяется в случае пустой базы данных для запросов типа поиск и удаление. В этом случае автомат, обслуживающий запрос, завершает функционирование с сигналом $e = 2$.

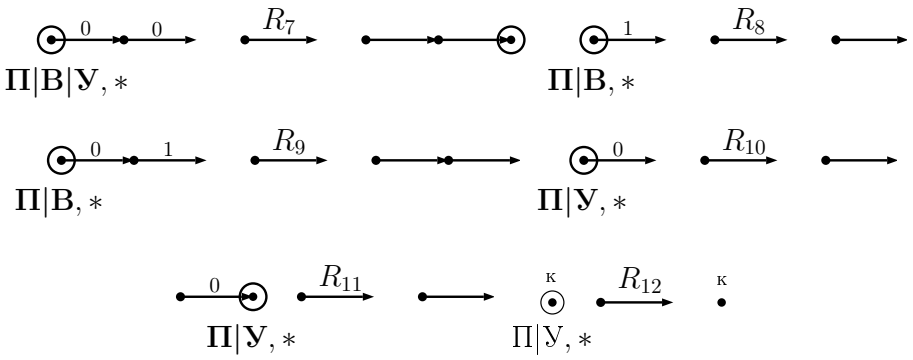

Рис. 12. Перемещение по ИГ.

Алгоритм работы автомата $\mathcal{A}$ зависит от типа запроса, который нужно обработать. Поэтому для каждого типа запроса отдельно опишем работу $\mathcal{A}$. Заметим, что все преобразования $R_1, \ldots, R_{12}$ однозначно применяются в зависимости от входной окрестности и типа запроса. Алгоритм автомата будет состоять в последовательном применении этих преобразований, пока в правой части преобразования не будет присутствовать кружок возле вершины. Это будет означать, что автомат завершил свою работу.

Когда база данных пустая, ИГ U состоит из одной вершины (корень). Если поступил запрос на поиск или удаление, то применяется преобразование R_{12} и после этого функционирование завершается с сигналом $e = 2$ (запись не найдена). Если поступил запрос на вставку, то применяется преобразование R_1 и после этого функционирование завершается с сигналом $e = 1$ (запись вставлена).

Для общего случая, алгоритм автомата $\mathcal{A}$ состоит в последовательном прохождении ребер, начиная с корня, при помощи преобразований $R_7, \ldots, R_{11}$.

Автомат на поиск применят преобразование в зависимости от входной окрестности. Если окрестность соответствует левой части преобразования R_7, то автомат применяет преобразование R_7. Он перемещается по списку, просмотрев два его элемента. Заметим, что автомат находит искомую запись если применяет преобразование R_8 или R_9, и не находит, если применяет преобразование R_{10} или R_{11}.

Алгоритм работы автомата на запросах типа вставка или удаление, повторяет алгоритм автомата на запросе типа поиск. Отличие алгоритма на вставку заключается в том, что вместо преобразований R_{10} или R_{11}, автомат применяет преобразование R_3 или R_2 соответственно. Отличие алгоритма на удаление заключается в том, что вместо преобразований R_8 или R_9, автомат применяет преобразование R_4, R_6 или R_5.

Докажем несколько вспомогательных лемм, для доказательства, что приведенный МДИГ $\mathcal{D}_M = (\mathcal{A}, U)$ удовлетворяет условиям теоремы 1.

Лемма 1. *Для любого потока запросов H из $\mathcal{H}$, МДИГ $\mathcal{D}_M$ функционирует без конфликтов.*

Доказательство. Запросы типа поиск, пустые запросы, а так же преобразования на перемещение по ИГ $R_7, \ldots, R_{12}$ не влияют на корректность функционирования МДИГ, так как не преобразуют окрестности. Поэтому, чтобы доказать корректность функционирования МДИГ, достаточно показать, что запросы на вставку и удаления не образуют конфликтов.

Докажем, что если нет запросов на удаление, то конфликтов не возникает. Рассмотрим два автомата с запросом на вставку. Каждый такт расстояние между ними не меньше двух ребер. Докажем это утверждение. Действительно рассмотрим первый автомат на вставку, который применил преобразование R_7. Это единственный случай, когда автомат продолжит функционирование. Если в следующий такт поступит еще один запрос на вставку, то автомат его обслуживающий, будет находиться в корне. Расстояние между эти автоматами будет ровно два ребра. Это расстояние не сможет уменьшиться, так как автоматы на вставку имеют одинаковый алгоритм. При этом если

один из автоматов завершает свое функционирование, то расстояния между автоматами не будет. Следовательно утверждение доказано. Аналогично доказывается, что расстояние между любыми автоматами на поиск и вставку не меньше двух. в дальнейшем будем говорить, что выполняется свойство удаленности, если расстояние между любыми двумя работающими автоматами не меньше двух ребер.

Как следствие мы получаем, что одновременно не могут произойти два преобразования на вставку R_2 или R_3. А это означает, что преобразования на вставку не вызывают конфликтов. Пример передвижения нескольких автоматов на вставку изображен на рисунке 13.

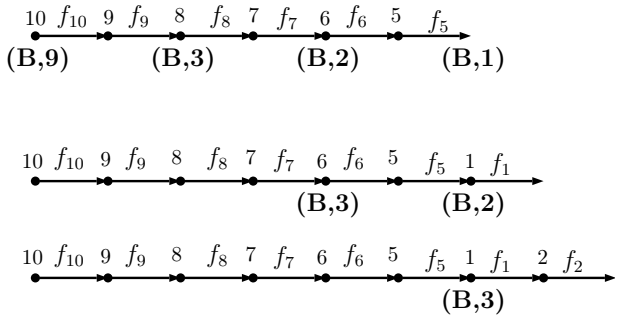

Рис. 13. Пример нескольких вставок.

Пусть имеется один запрос на удаление. Покажем, что при этом так же не возникнет конфликтов. Запрос на удаление повторяет алгоритм вставки до того момента как найдет искомую запись. Следовательно, если автомат на удаление не найдет искомую запись, то конфликтов не возникнет. При этом будет сохранено свойство удаленности. Пусть существует запись, которую он должен удалить. Преобразование R_4 удаляет только ребро и запись и не удаляет вершину в которой находиться. Поэтому это не вызовет конфликта с автоматом, который в следующий такт должен был перейти в эту вершину.

Осталось разобрать два внутренних случая. Это преобразование R_5 или R_6. Рассмотрим первый случай, когда автомат применил преобразование R_5. Это преобразование удаляет элемент списка. При этом в эту вершину не мог прийти ни один автомат. Докажем это.

Действительно, до момента удаления алгоритм удаления совпадал с алгоритмом на вставку (поиск). Как уже было доказано выше, расстояние между любыми автоматами на вставку и поиск не меньше

двух ребер. Следовательно, ближайшие автоматы к рассматриваемому, находятся не меньше чем на расстоянии в 2 ребра. Поэтому в удаляемою вершину в следующий такт не придет ни один автомат. Следовательно, это преобразование не вызывает конфликт. Так же это преобразование не нарушает свойство удаленности. При этом после данного преобразования расстояние между запросами, пришедших до и после рассматриваемого, обязательно будет не менее трех ребер. Пример передвижения нескольких автоматов на вставку и одного удаления R_5 изображен на рисунке 14.

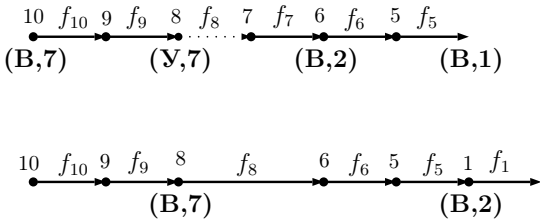

Рис. 14. Пример нескольких вставок и одного удаления R_5.

Рассмотрим теперь второй вариант, когда автомат применил преобразование R_6. Как и в предыдущем случае, автоматы находятся на расстоянии не менее, чем два ребра друг от друга. Поэтому в следующий такт в этой вершине мог оказаться автомат, который перешел в нее только по преобразованию R_7. Кроме того, в вершине $\mathbf{v}_2$ не может находиться автомат, а следовательно и применяться преобразование. Преобразование R_6 удаляет ребро, инцидентное вершине $\mathbf{v}_1$, при этом саму вершину не удаляет. Так же преобразование удаляет вершину $\mathbf{v}_2$, а инцидентное ей ребро, присваивает вершине $\mathbf{v}_1$. Следовательно, это преобразование не вызывает конфликтного изменения одного и того же участка памяти. Кроме того, это преобразование, как и преобразование R_5, сохраняет свойство удаленности. Причем аналогично R_5 расстояние между запросами до и после, рассматриваемого, будет не меньше трех ребер. Пример передвижения нескольких автоматов на вставку и одного удаления R_6 изображен на рисунке 15.

Итак, мы доказали, что одно удаление не вызывает конфликтов. Теперь рассмотрим случай, когда одновременно обрабатываются два запроса на удаление. Аналогично случаю с одним удалением, алгоритм удаления совпадают с алгоритмом на поиск и вставку, пока не будет найдена искомая запись. Если хотя бы один из автоматов на

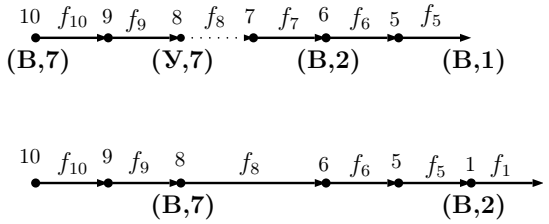

Рис. 15. Пример нескольких вставок и одного удаления R_6.

удаление не найдет искомую запись, то этот случай равносилен разобранному выше случаю с одним удалением.

Рассмотрим все сценарии функционирования МДИГ, в случае двух запросов на удаление. Будем считать, что имеется два запроса на удаление $У_1$ и $У_2$, и запрос $У_1$ поступил раньше запроса $У_2$. Тогда, как уже было доказано выше, между этими запросами будет расстояние не менее двух ребер. Если запрос на удаление $У_2$ завершает свое функционирование раньше, чем запрос $У_1$, то этот случай равносилен случаю с одним удалением. Если запрос $У_1$ применил преобразование, а $У_2$ в этот такт не удаляет запись, то его можно считать запросом на поиск. Этот случай был разобран выше. Остался последний вариант, когда оба запроса на удаление одновременно удаляют записи. Если между ними был запрос на поиск или вставку, то это случай был разобран выше. Если же нет, то минимальное возможно расстояние между этими запросами два ребра.

Если $У_1$ применил преобразование R_4, то оно не вызовет конфликта. Осталось рассмотреть четыре варианта. Понятно, что достаточно рассмотреть случай минимального возможного расстояния между автоматами в два ребра.

Первый случай — оба автомата применили преобразование R_5. Преобразование R_5 удаляет вершину справа и меняет информацию о ребре для вершины, в которой находиться автомат. При этом данное преобразование не меняет информацию во второй справа вершине в которой находится другой автомат. Следовательно, конфликтов не возникает. Пример для этого случая изображен на рисунке 16.

Второй случай — $У_2$ применил преобразование R_5, а $У_1$ применил преобразование R_6. Преобразование R_5 не меняет информацию, в которой находится другой автомат. При этом автомат $У_1$ меняет

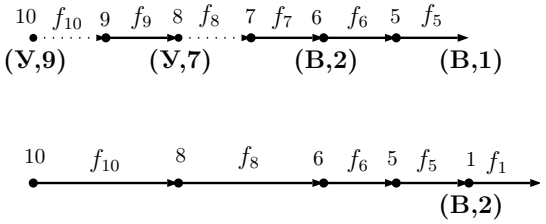

Рис. 16. Пример двух удалений. Случай R_5, R_5.

информацию только в вершине, в которой находиться. Следовательно, конфликтов не возникает. Пример для этого случая изображен на рисунке 17.

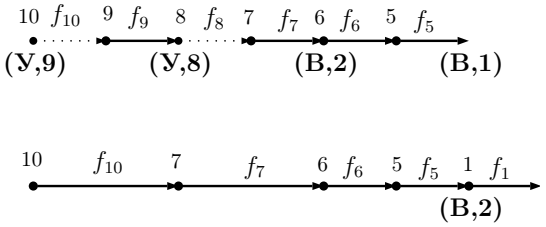

Рис. 17. Пример двух удалений. Случай R_5, R_6.

Третий случай — $У_2$ применил преобразование R_6, а $У_1$ применил преобразование R_5. В этом случае так же не возникает конфликтов. Преобразования меняют информацию в вершине в которой находятся, при этом изменяемые ребра не пересекаются. Пример для этого случая изображен на рисунке 18.

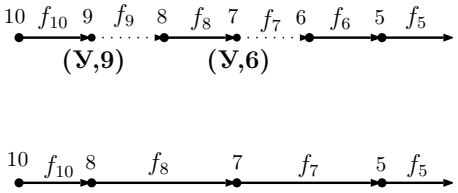

Рис. 18. Пример двух удалений. Случай R_6, R_5.

В последнем, четвертом случае, когда оба автомата применили преобразование R_6, также не возникает конфликтов. Пример для это-

го случая изображен на рисунке 19 (преобразования (У,5) и (У,3)). Следовательно, мы доказали, что в случае двух удалений конфликтов не возникает.

Рассмотрим общий случай, когда удалений может быть больше двух. Рассмотрим произвольный запрос на поиск или вставку. Автоматы обслуживающие эти запросы не конфликтуют с удалениями. Действительно, если перед автоматом на вставку или поиск идет запрос на удаление, то конфликтов не возникает в силу разобранного выше случая с одним удалением. Таким образом, если мы докажем, что три и более подряд удалений не конфликтуют, то мы докажем лемму.

Рассмотрим автомат на удаление, поступивший в такт i. Если в такт $i+1$ или в такт $i-1$ не поступали запросы на удаление, то этот случай был разобран выше, так как при этом не возникнет больше двух удалений подряд.

Теперь рассмотрим вариант, когда и в $i+1$ и в i такт поступили запросы на удаление. Конфликтов между запросами, поступивших в такты $i-1$ и i, не возникает, так как этот случай был разобран выше. Аналогично для пары запросов, поступивших в такты $i+1$ и i. Очевидно, что между запросами, поступившими в такты $i-1$ и $i+1$ так же нет конфликтов. Следовательно, среди трех удалений не возникает конфликтов. Аналогично доказывается, что для любого числа, идущих подряд удалений, не возникает конфликтов. Пример для этого случая изображен на рисунке 19.

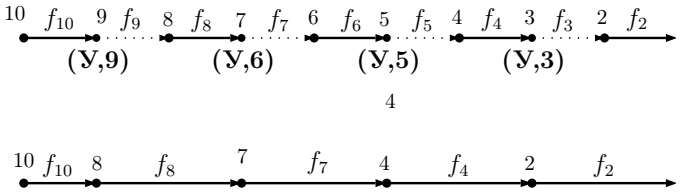

Рис. 19. Пример трех и более удалений.

Лемма доказана.

Лемма 2. *Для любого потока запросов $H, H \in \mathcal{H}$, МДИГ $\mathcal{D}_M$ функционирует корректно, кроме того то для любого натурального i выполняется*

$$T(\mathcal{D}_M, H(i)) \leqslant]|V(i,H)|/2[, \text{ если } W(H)(i) = \Pi.$$

Доказательство. Нужно показать, что для любого запроса на поиск $H(i), H(i) = (\Pi, x)$, МДИГ выдает в ответ $\{x\}$, если $x \in V(i, H)$ и пустое множество в противном случае.

Из леммы 1, в частности, следует, что любые преобразования сохраняют свойство удаленности для любого потока запросов H из $\mathcal{H}$. Другими словами, автоматы идут друг за другом в порядке появления запроса. Рассмотрим произвольный запрос на поиск, поступивший в i такт.

Рассмотрим преобразование на вставку (удаление), поступившее в такт $j, 1 \leqslant j \leqslant i-1$ и которое еще не завершило свое функционирование. Так как МДИГ сохраняет свойство удаленности, то в стадии поиска между ними будет расстояние $i - j \geqslant 2$ ребра, следовательно автомат на вставку (удаление) успеет вставить (удалить) запись прежде, чем автомат на поиск сможет сократить расстояние.

Поэтому МДИГ работает корректно, так как любое преобразование завершиться не менее чем за такт до того, как запрос на поиск их достигнет. Следовательно, корректность доказана.

Из корректности МДИГ $\mathcal{D}_M$, а так же его сохранения свойства удаленности следует, что автомат на поиск затратит не больше $]|V(i, H)|/2[$ тактов. Действительно, свойство удаленности означает, что любой запрос на преобразование завершиться таким образом, что другие автоматы будут идти уже по обновленному ИГ, если они поступили на такт позже. При этом за один такт автомат проходит сразу два элемента списка или находит искомый. Таким образом, получаем оценку на сложность поиска

$$T(\mathcal{D}_M, H(i)) \leqslant]|V(i, H)|/2[.$$

Лемма доказана.

Утверждение теоремы 1 является следствием лемм 1 и 2.

Докажем теперь теорему 2. Смысл теоремы 2 можно переформулировать следующим образом. Если в ИГ любой вершине инцидентно не более 1 ребра, то не существует автомата, который смог бы обслужить любой поток запросов.

Ограничение на количество инцидентных ребер означает, что ИГ может представлять из себя объединение разных компонент связности, каждая из которых состоит либо из 1 вершины, либо из двух вершин и ребра их соединяющего. Других компонентов связности быть

не может, так как из вершины не может выходить более одного ребра. Пример графа изображен на рисунке 20.

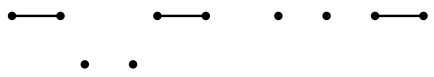

Рис. 20. Пример графа с $N = 1$.

Заметим, что автомат видит только вершины, между которыми есть путь длины не больше R, а это в свою очередь означает, что вершины находятся в одной компоненте связности. Следовательно, автомат не сможет переместиться из одной компоненты связности в другую, во время функционирования.

В одной вершине ИГ не может находиться более одной записи, следовательно, в одной компоненте связности не может находиться более двух записей. Поэтому рассмотрим следующий поток запросов: $(В, 1), (В, 2), (В, 3), (П, 1), (П, 2), (П, 3)$. Очевидно, что автомат на поиск не сможет найти одну из трех записей, поэтому он не будет работать корректно на этом потоке запросов. Теорема доказана.

Список литературы

[1] Плетнев А. А. Моделирование динамических баз данных // Интеллектуальные системы. — 2013. Т. 17, вып. 1-4. — С. 75–79.

[2] Плетнев А. А. Информационно-графовая модель динамических баз данных и ее применение // Интеллектуальные системы. — 2014. Т. 18, вып. 1. — С. 111–140.

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

[4] Snir M. On parallel searching // SIAM J. Computing. — 1985. 14. — P. 688–708.

[5] Gafni E., Naor J., Ragde P. On Separating the EREW and CREW PRAM Models // Theoretical Computer Science. — 1989. 68. — P. 343–346.

[6] Park H., Park K. Parallel algorithms for red-black trees // Theoretical Computer Science. — 2001. 262. — P. 415–435.

[7] Wang B.-F., Chen G.-H. Cost-optimal parallel alghoritms for constructing 2-3 trees // Journal Of Parallel And Distributed Computing. — 1991. 11. — P. 257–261.

[8] Кудрявцев В. Б., Алешин С. В., Подколзин А. С. Введение в теорию автоматов. — М.: Наука, 1985.