Минимально возможный по степени ветвления информационный граф с радиусом видимости один, обрабатывающий произвольный поток запросов к динамической базе данных

А. А. Плетнев

Рассматривается динамическая задача поиска идентичных объектов (ДЗПИО). В данной работе представлен конечный МДИГ с радиусом видимости один и степени ветвления два, обрабатывающий произвольный поток запросов. Это минимально возможный по степени ветвления МДИГ с радиусом видимости один, решающий поставленную задачу.

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

Введение

В современном мире нас окружает огромное количество информации, поэтому нахождение среди нее необходимых данных является актуальной задачей. Математический интерес к данной проблеме заключается в описании модели, с помощью которой можно получать нужные алгоритмы поиска искомой информации. К решению этой задачи подходят с различных сторон. Существенный вклад в моделирование баз данных внес Гасанов Эльяр Эльдарович [2]. Он предложил информационно-графовый

подход. В настоящее время этот подход используют многие ученые. Одни из последних работ по этой тематике можно найти в [3, 4, 5, 6].

В данной работе будет рассмотрена постановка задачи, совпадающая с описанной в [7]. Она занимает большой объем, поэтому здесь опишем только ее смысл, а точную формулировку можно прочитать в [7].

Рассматривается поток запросов. 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. Также 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. Ограничения на разные процессы состоит в том, что они не могут изменять одновременно один участок памяти, но при этом могут считывать такие участки. Такие структуры данных называют ОЧЭЗ (одновременное чтение эксклюзивная запись) структурами. В иностранной литературе они описываются как CREW структуры [8].

Будем считать, что у нас есть неограниченное количество доступных процессов. Каждый процесс обрабатывает один запрос, который может быть одним из трех типов: поиск, вставка или удаление. В работах [7, 9, 10] были предложены ОЧЭЗ структуры данных, решающие ДЗПИО. Идея построения искомой структуры данных основывается на информационно-графовом(ИГ) подходе [2] с использованием конечного автомата [11]. Такая структура данных названа МДИГ, и она зависит от нескольких параметров. В данной работе будет приведен МДИГ с радиусом видимости один и с минимальной возможной степенью ветвления — два .

Минимально возможный по степени ветвления информационный граф с радиусом видимости один, обрабатывающий произвольный поток запросов к динамической базе данных

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

1. Постановка задачи и результаты

Пусть H — поток запросов, $H : \mathbb{N} \to X$, где $X, X = \{\text{П, В, У}, \Lambda\} \times \mathbb{N}$ — множество запросов. Буква в запросе означает его действие: поиск (П), вставка (В), удаление (У) или пустой запрос (Λ).

Множество всех потоков запросов обозначим через $\mathcal{H}$. Рассмотрим функцию множества записей $V : \{\mathbb{N} \cup \{0\}\} \times \mathcal{H} \to 2^X$, которая удовлетворяет следующим условиям:

$$V(i, H) = \begin{cases} \emptyset, & \text{если } i = 0; \\ V(i-1, H), & \text{если } i > 0 \text{ и } H(i) \text{ запрос на поиск или пустой запрос;} \\ V(i-1, H) \cup \{x\}, & \text{если } i > 0 \text{ и } H(i) = (\text{В}, x); \\ V(i-1, H) \setminus \{x\}, & \text{если } i > 0 \text{ и } H(i) = (\text{У}, x). \end{cases}$$

Функция V сопоставляет 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).

Пусть дана функция $L : \mathbb{N} \to \mathbb{N}$. Будем говорить, что МДИГ решает ДЗПИО (*логическую ДЗПИО*) со сложностью L, если для любого потока H и любого такта i выполняются следующие условия

- если $H(i) = (\text{П}, x)$, то результатом функционирования МДИГ должен быть $\{x\}$ (да), если $x \in V(i, H)$ и пустое множество (нет) в противном случае. При этом количество тактов, необходимое на выдачу ответа, должно не превосходить $L(|V(i, H)|)$;
- если $H(i) = (\text{В}, x)$, то для любого запроса на поиск (П, z), поступившего в такт $i + 1$, результат функционирова-

ния МДИГ должен быть $\{z\}$(да), если $z \in V(i+1,H) = V(i,H) \cup \{x\}$, и пустое множество(нет) 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 $\{z\}$(да), если $z \in V(i+1,H) = V(i,H) \setminus \{x\}$, и пустое множество(нет) в противном случае.

Очевидно, что если МДИГ решает ДЗПИО с выдачей ответа, то он решает ее в постановке да или нет. Также не сложно заметить, что, если МДИГ не решает задачу в постановке да или нет, то он не решает ее с выдачей ответа.

Напомним, что означает базовое множество для МДИГ. Множество предикатов F – это множество функций, которые являются нагрузками ребер ИГ. МДИГ базируется на конечном автомате, поэтому сами функции, приписанные ребрам, автомат увидеть не может. Автомат во время функционирования видит значение этих функций на запросе, который он обслуживает, а не сами функции. Также, помимо значений функций на запросе, автомат видит структуру подграфа ИГ. Подграф, который видит автомат, ограничен фиксированным параметром R, означающим радиус окрестности с центром в вершине, в которой автомат находится. Другими словами, если автомат находится в вершине v, то он видит все вершины, которые находятся на расстояния не большем, чем R ребер от нее. Пример радиуса видимости автомата изображен на рисунке 1. Естественно, сам ИГ не может иметь бесконечное ветвление, так как конечный автомат даже не сможет увидеть окрестность радиуса 1 в этом случае. Поэтому МДИГ подразумевает, что ИГ имеет ограничение на ветвление (максимальное количество ребер инцидентных вершине), которые зависит от параметра N. На рисунке 1, N равняется 3 и достигается в вершине, в которой стоит автомат. Множество преобразований $\mathcal{R}$ означает, что автомат в каждый такт своего функционирования может применить преобразование из этого множества. Цель преобразований – это перемещение по ИГ и его локальная модификация для поддержания базы данных в актуальном состоянии.

Минимально возможный по степени ветвления информационный граф с радиусом видимости один, обрабатывающий произвольный поток запросов к динамической базе данных

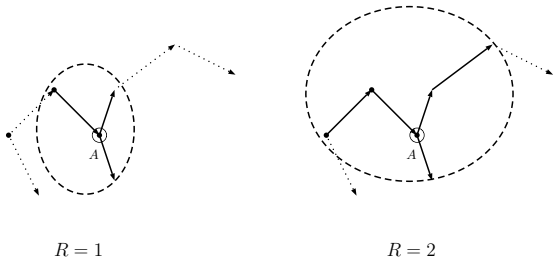


Рис. 1: Пример радиуса видимости автомата.

Будем говорить, что МДИГ типа (N, R), если ИГ степень ветвления не больше, чем N, а радиус видимости автомата не больше, чем R. Более подробно о том, как именно автомат видит окрестность, можно прочитать в [7].

Введем понятие *конечный* МДИГ. Будем говорить, что МДИГ конечный, если для произвольного потока запросов H и произвольного такта времени i существует такая константа C,$C < \infty$, что для потока запросов $H^{'}, H^{'}(1) = H(1), \ldots H^{'}(i) = H(i), H^{'}(i+1) = \Lambda, \ldots, H^{'}(i+C) = \Lambda$, автомат обслуживающий запрос $H(i)$ завершит свое функционирование.

Неформально говоря, МДИГ будем называть конечным, если автомат для любого потока запроса и любого такта, функционирует конечное время.

В ИГ каждой вершине присвоен один из трех типов: "к"(корень), "в"(внутренняя) и "л"(листовая). В данной работе разрешим использовать другие типы вершин в ИГ. А именно, введем три новых типа вершин ИГ: "у", "$к_{\leftarrow}$" и "$к_{\rightarrow}$". Эти типы будут нужны для доказательства теоремы 1. Сразу сделаем замечание, что можно было бы и не вводить новые типы вершин. Для этого вместо типа "у" нужно подставить тип "к", вместо типа "$к_{\leftarrow}$" тип "в", а вместо типа "$к_{\rightarrow}$" тип "л". В этом случае доказательство теоремы 1 сильно бы усложнилось дополнительными объяснениями. Поэтому будем считать, что у нас есть три новых типа вершин, которые могут встречаться в ИГ.

Пусть

$$T = \{а, н\} \times \{-2, -1, 0, 1, 2\} \times \{П_{\leftarrow}, П_{\rightarrow}, В_{\leftarrow}, В_{\rightarrow}, У_{\leftarrow}, У_{\rightarrow}, н\},$$

$F(T) = \{f^t_{l,r,a}(x) : t \in T, l, r, a, x \in \mathbb{N}\}$, где

$f^t_{l,r,a}(x) = 1$, если $x = a$ и 0 иначе.

Теорема 1. *Существует множество преобразований $\mathcal{R}$ и существует конечный МДИГ $\mathcal{D}_П = (\mathcal{A}, U)$ типа (2,1) над базовым множеством $\langle F(T), \mathcal{R}\rangle$, решающим ДЗПИО со сложностью $L, L(n) =]n/2[+5$.*

Доказательство. Множество преобразований $\mathcal{R}$ будет представлено явно в процессе доказательства. В данной теореме не будем представлять иллюстрацию к каждому преобразования, так как здесь более понятно и удобнее описать алгоритм автомата. К некоторым преобразованиям приведем иллюстрации, а к остальным по аналогии можно так же их предъявить.

Идея доказательства будет следующей. МДИГ типа (2,1), поэтому база данных будет представлять из себя список, причем корень ИГ будет находиться посередине. Следовательно, запросу на поиск потребуется в худшем случае $]n/2[$ тактов на выдачу ответа, где n — это мощность базы данных. В теореме сложность оценивается как $]n/2[+5$. Ниже объясним, откуда могут взяться еще 5 тактов.

Чтобы поддерживать корень посередине базы данных, будет использоваться идея из теоремы [1, Теорема 3] с некоторыми дополнениями. А именно, в первый такт своего функционирования автомат меняет предикат или предикаты, выходящие из корня, спрашивая есть ли искомая запись в базе данных или нет. Во второй такт автомат смотрит значение t_3, полученного предиката. Например, если это автомат на поиск, то на месте t_3 он может увидеть "$П_{\leftarrow}$", "$П_{\rightarrow}$" или "н". "$П_{\leftarrow}$" ("$П_{\rightarrow}$") означает, что искомая запись существует и она находится слева(справа) от корня. Если автомат видит в качестве t_3 символ "н", это означает, что искомой записи нет.

Основная идея алгоритма

Итак, опишем основную идею алгоритма. Автомат в первый такт своего функционирования выясняет, есть ли обслуживаемый им запрос в базе данных или нет. Кроме этого, он смотрит

нарушался ли баланс. Под балансом понимаем соотношение количества записей слева и справа от корня. Баланс будет находиться в параметре t_2 типа предиката.

Алгоритм различается у автоматов на поиск, вставку и удаление, но их объединяет общая идея, что в первый такт своего функционирования, автомат должен узнать есть искомая запись в базе данных или нет. Если есть, то он должен знать слева или справа от корня она находится. Чтобы узнать есть ли искомая запись, автомат меняет ребра или ребро, выходящее из корня, и на следующий такт считывает информацию с этих ребер или ребра. Автомат не всегда может поменять оба ребра, выходящих из корня, так как одно из них может менять автомат, который начал функционирование раньше. Ниже будет доказано, что ситуации, когда автомат не сможет поменять ни одного ребра не возникнет.

Все ребрам будут соответствовать только предикаты. Поэтому, будем говорить тип ребра, подразумевая тип предиката ему соответствующего.

Для того, чтобы корректно определить есть ли обслуживаемая запись в базе данных или нет, введем понятие *актуального* ребра. *Актуальным* будем называть ребро, которое выходит из корня и параметр t_1 его типа равен "а". Ниже будет показано, что может возникнуть ситуация, когда такого ребра нет, при этом тип корня будет обязательно равен "$к_{\leftarrow}$" ("$к_{\rightarrow}$"). В этом случае актуальным будем называть левое (правое) ребро.

Сделаем небольшое замечание, что случай, когда оба ребра имеют t_1 равным "н", возникает из-за пустых запросов в совокупности с запросами на вставки. Если пустых запросов нет, то такой ситуации вообще не возникнет.

Автоматы на вставку могут поддерживать баланс, вставляя новую запись в нужную сторону. Автоматы на поиск, как и пустые запросы, не изменяют баланс. Единственное, что может нарушить баланс, это автомат на удаление. Поэтому если автомат на удаление видит, что баланс нарушен, то есть он равен 2 (-2), то он "перебрасывает" запись справа(слева) от корня, несмотря на то, будет он удалять запись или нет. После этого

преобразования баланс станет равен 0. Следовательно, если автомату на удаление нужно будет удалить запись, баланс станет равным 1 или -1, а если не нужно, то останется 0. В любом случае, он будет принадлежать множеству {-1,0,1}.

К дополнительным исключениям так же можно отнести тот факт, когда автомат на удаление удаляет запись, в следующий такт в эту вершину не должен переходить ни один автомат, чтобы избежать конфликтов. Это так же вносит коррективы в функционирование автоматов. Например, автомат на поиск должен будет один такт постоять на месте, если в вершину, в которую он собирается перейти, будет удаляться (тип вершины равен "в"). Будет доказано, что на корректность работы это не повлияет, и этот автомат на поиск все равно найдет искомую запись, при этом он позволит автомату на удаление удалить свою, не беспокоясь о том, что в удаляемую вершину кто-то может перейти.

Таким образом, описан основной смысл всех преобразований. Далее в доказательстве будет большой разбор случаев, а именно, как и что должен делать автомат, чтобы избежать конфликта с другими автоматами в различных ситуациях.

Формальное описание преобразований

Сначала рассмотрим случай, когда ИГ состоит из одной вершины (корень), или из корня может выходить одно ребро в вершину с типом "у". Естественно, что запросы на поиск и удаление видят, что ИГ состоит из одной вершины или в нем есть одно ребро в "пустую" вершину, поэтому сразу завершают свое функционирование. Одно из преобразований, изображено на рисунке 2. Знак "|" на рисунке означает "или". То есть, это общее преобразование для автомата на поиск и удаление. Второе преобразование, когда в пустой базе данных есть одно ребро, ведущее в вершину с типом "у", будет разъяснено ниже.

В следующих преобразованиях нам понадобится функция P, где $P : \mathbb{N} \to \mathbb{N}$, которая на входе получает натуральное число n, а на выходе выдает n-е простое число. То есть $P(1) = 2, P(2) = 3, P(3) = 5, P(4) = 7, \ldots$.

Минимально возможный по степени ветвления информационный граф с радиусом видимости один, обрабатывающий произвольный поток запросов к динамической базе данных

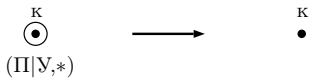


Рис. 2: Преобразование автоматов на удаление и поиск. Случай пустой базы данных.

Рассмотрим теперь поведение автомата на вставку в этом случае. Автомат вставляет запись слева от корня. Эти преобразования изображены на рисунке 3.

Разберем детально это преобразование. Автомат создает ребро и вершину, которой приписывает запись, которую он обслуживает. Ребру присваивает предикат $f^{(\text{а},-1,\text{В}_{\leftarrow})}_{P(*),1,*}$. У этого предиката есть верхний индекс $(\text{а},-1,\text{н})$ и нижний индекс $(P(*),1,*)$. Первый аргумент нижнего индекса означает произведение простых чисел, соответствующих записям слева от корня. Функция P — эта та же функция, что использовалась в теореме [1, Теорема 3]. Например, если $*$ равнялась бы 5, то нижний индекс был бы равен $(P(5),1,5) = (11,1,5)$, так как пятое простое число это 11 (2,3,5,7,11). Первый аргумент верхнего индекса означает, что числа $P(*)$ и 1 актуальны. Второй параметр верхнего индекса -1 означает баланс, то есть, что слева от корня записей на одну больше, чем справа. Третий параметр означает, что была вставка влево.

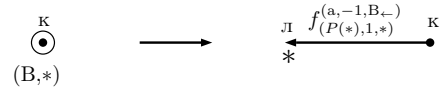


Рис. 3: Преобразование автомата на вставку. Случай пустой базы данных.

Разберем теперь случай, когда ИГ состоит из одного ребра, ведущего в вершину "в". Этот случай аналогичен уже разобранному выше, так как в базе данных нет записей, за исключением вставки. Вставка вставляет новую запись в противоположную сторону. Ниже будет разъяснено, как правильно должны удаляться граничные вершины ИГ, чтобы избежать конфликтов.

Случай, когда ИГ состоит из одного ребра ведущего в вершину ”л”, попадает под общий случай, когда из корня выходит два ребра, считая условно, что второе ребро существует, и его предикат равен $f^{н,0,н}_{1,1,1}$.

Теперь перейдем к общему случаю. Опишем алгоритма автомата на поиск.

Поиск, первый такт

В дальнейшем фразу ”меняет ребро” нужно понимать как ”меняет предикат, принадлежащий ребру”. Отдельно будем оговаривать ситуации, когда ребро будет удаляться или будут меняться инцидентные вершины.

Первый такт любого автомата заключается в определении искомой записи в базе данных. Для этого автомат преобразует одно или два ребра, выходящих из корня. Количество преобразуемых ребер зависит от входной окрестности. Основная цель — не создать конфликт. Автомат может понять, будет ли преобразовываться одно из ребер или нет другим автоматом. Если оно будет преобразовываться, то он меняет другое ребро, иначе меняет оба ребра.

Опишем сначала ситуации, когда автомат на поиск меняет два ребра. Тип корня должен быть ”к”, а типы вершины слева от корня и справа от корня должны принадлежать множеству {”л”, ”у”}.

Теперь опишем входные окрестности, при которых автомат на поиск меняет только левое или правое ребро. Если тип корня равен ”$к_{\leftarrow}$” (”$к_{\rightarrow}$”), то автомат меняет только правое (левое) ребро. Так же, если тип левой (правой) вершины равен ”в”, то он меняет только правое (левое) ребро.

Осталось описать, как именно меняется ребро, то есть каким образом меняется его предикат. Для этого автомат определяет актуальное ребро. Понятие актуального ребра было описано выше. От актуального ребра важно знать, какие записи есть слева, а какие справа. Идея преобразования ребра аналогична идее, используемой в [1, Теорема 3].

Минимально возможный по степени ветвления информационный граф с радиусом видимости один, обрабатывающий произвольный поток запросов к динамической базе данных

Преобразования соответствующие этим случаям изображены на рисунке 4. Функция ϕ^f_Π описывается таблицей (1). Столбцы 2—3 таблицы 2 содержат значения параметров актуального ребра, а в столбце 4 представлен новый предикат ребра. Разберем подробнее функцию ϕ^f_Π. Точкой обозначены параметры, которые не будут меняться. Параметр t_1 нового предиката в любом случае становится равен "а". Тип t_3 равен "н" если искомой записи нет в базе данных и равен "$\Pi_\leftarrow$" ("$\Pi_\rightarrow$") если запись находится в левой (правой) половине ИГ относительно корня.

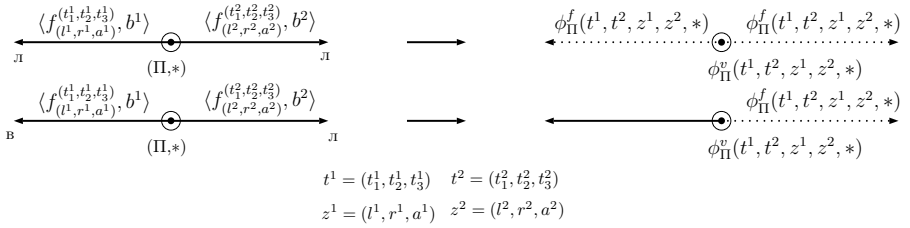


Рис. 4: Преобразование первого такта автомата на поиск в общем случае.

	$l \dot{:} P(*)$	$r \dot{:} P(*)$	ϕ^f_Π
1	нет	нет	$f^{(\text{а},\cdot,\text{н})}_{(\cdot,\cdot,\cdot)}$
2	да	-	$f^{(\text{а},\cdot,\Pi_\leftarrow)}_{(\cdot,\cdot,\cdot)}$
3	-	да	$f^{(\text{а},\cdot,\Pi_\rightarrow)}_{(\cdot,\cdot,\cdot)}$

(1)

Поиск, второй и следующие такты

Второй такт функционирования автомата на поиск заключается в определении, есть ли запись в базе данных или нет. Если записи нет, то автомат завершает свое функционирование. Он понимает, что записи нет, если значение t_3 ребра, которое он менял на предыдущем такте равно "н". Если же значение t_3 равно "$\Pi_\leftarrow$" ("$\Pi_\rightarrow$"), то он понимает, что запись есть, и она находится слева (справа) от корня.

В этих случаях автомат применяет следующий алгоритм. Он меняет свое внутренне состояние на двигаться влево (вправо). Если предикат слева (справа) принимает значение один, то автомат выдает в качестве ответа запись приписанной вершине слева (справа), за небольшим исключением, которое поясним ниже. Если значение предиката равно 0, то автомат смотрит на тип вершины слева (справа). Если он не равен "в", то он перемещается в эту вершину, иначе он остается на месте. Напомним, что тип вершины "в" в данном алгоритме означает, что эта вершина будет удаляться на следующий такт. Поэтому автомат на поиск не может перейти в эту вершины, чтобы избежать конфликта.

Исключение, которое упоминалось выше, состоит в следующем. Автомат на поиск может стоять в корне некоторое количество тактов из-за того, что вершина куда он должен двигаться имеет тип "в". Поэтому он следит за параметром актуального ребра. Если оно меняется и становится таким, что его параметр t_3 принял значение "$В_{\rightarrow}$" ("$В_{\leftarrow}$"), где "направление вставки" совпадает с направлением движения автомата на поиск, то автомат это запоминает. На следующий такт вставка вставит на место вершины с типом "в" новую запись, которая могла совпасть с той, которую ищет поиск. Если такое происходит, то поиск пропускает эту вершину и продолжает искать свою, несмотря на то, что предикат был равен одному. Ниже подобная ситуация будет объяснена для автомата на удаление.

Итак опишем теперь алгоритм, который применяет автомат на вставку.

Вставка, первый такт

Здесь будет много случаев в зависимости от типов вершин входной окрестности и баланса. Рассматриваемый случай будем писать курсивом.

Обе вершины слева и справа от корня имеют тип "л", корень имеет тип "к"

Минимально возможный по степени ветвления информационный граф с радиусом видимости один, обрабатывающий произвольный поток запросов к динамической базе данных

	t_2	$l \dot{:} P(*)$	$r \dot{:} P(*)$	ϕ^f_{B}	ϕ^v_{B}
1	-	да	-	$f^{(\text{a},\cdot,\text{н})}_{(l,r,\cdot)}$	к
2	-	-	да	$f^{(\text{a},\cdot,\text{н})}_{(l,r,\cdot)}$	к
3	$-2,-1,0$	нет	нет	$f^{(\text{a},t_2+1,\text{B}_{\rightarrow})}_{(l,r*P(*),\cdot)}$	$\text{к}_{\rightarrow}$
4	$1,2$	нет	нет	$f^{(\text{a},t_2-1,\text{B}_{\leftarrow})}_{(l*P(*),r,\cdot)}$	$\text{к}_{\leftarrow}$

(2)

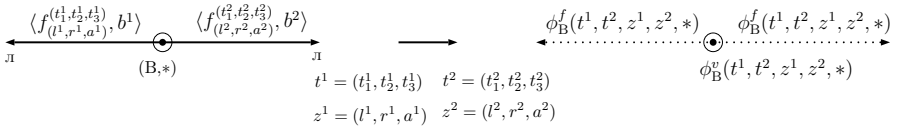


Рис. 5: Преобразование на вставку. Первый такт, общий случай.

Преобразование, описывающее первый такт автомата на вставку, когда слева и справа от корня вершины имеют тип "л", изображено на рисунке 5. Опишем функцию ϕ^f_{B}, которую использует автомат. Обратим внимание, что после преобразования слева и справа от корня будут одинаковые предикаты.

Функции ϕ^f_{B} и ϕ^v_{B} описываются таблицей (2). Столбцы 2—4 таблицы 2 содержат значения параметров актуального ребра, а в столбцах 5—6 представлен новый предикат ребра и новый тип корня.

Строчки 1—2 таблицы, изображают изменение предиката, когда автомат вставляет уже существующую запись. Строчки 3—4 таблицы соответствуют случаям, когда автомат вставляет новую запись. Функция ϕ^v_{B} меняет тип вершины. При этом не важно какой тип был у корня. Если автомат на вставку собирается вставить запись влево (вправо), то функция ϕ^v_{B} делает тип вершины "$\text{к}_{\leftarrow}$" ("$\text{к}_{\rightarrow}$"). Это будет использоваться, чтобы избежать конфликтов между вставкой и удалением. Заметим, что типы "$\text{к}_{\leftarrow}$" ("$\text{к}_{\rightarrow}$") введены для лучшего понимания. На самом деле, им соответствуют типы в(л), а автомат знает, что, если корень имеет тип в (л), то его нужно воспринимать как тип "$\text{к}_{\leftarrow}$" ("$\text{к}_{\rightarrow}$"). Далее везде будем писать типы "$\text{к}_{\leftarrow}$" ("$\text{к}_{\rightarrow}$") вместо в (л).

Разберем подробнее функцию $\phi^f_{\text{В}}$. Прежде всего отметим, что точкой обозначены параметры, которые не будут меняться. Параметр t_1 нового предиката в любом случае становится равен "а". Начнем со строчек 1—2. В них представлены случаи, когда вставляемая запись уже существует. Параметр t_3 нового предиката становится равным "н", что и будет означать, что вставлять запись не нужно.

Разберем теперь строчки 3—4 таблицы 2. Это случаи, когда вставляется запись, которой в базе данных нет. В этом случае функция $\phi^f_{\text{В}}$ говорит, куда вставке нужно будет вставить запись $*$ на следующий такт. Выбор осуществляется, исходя из баланса t_2. Баланс — это разница между количеством записей справа от корня и количеством записей слева от него. Например, баланс -1 означает, что слева от корня на одну запись больше, чем справа. Следовательно, в этом случае вставке лучше вставить запись вправо от корня, так как после этого преобразования баланс станет равен 0. Ниже будет доказано, что баланс будет всегда принадлежать множеству {-2, -1,0,1, 2}.

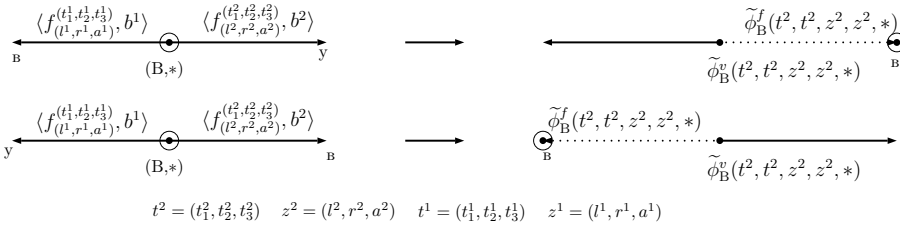
Вершина слева (справа) от корня имеет тип "в", а другая тип "л" или корень имеет тип "к$_{\leftarrow}$" ("к$_{\rightarrow}$")

Аналогично разобранному выше алгоритму поиска, в этом случае можно поменять только одно ребро. Опишем преобразование, когда вершина справа имеет тип "в" или тип корня "к$_{\leftarrow}$". Случай, когда вершина слева имеет тип "в" или тип корня "к$_{\rightarrow}$", будет симметричен предыдущему. По сути это то же преобразование, что и в первом случае, только автомат на вставку меняет ребро ведущие в вершину с типом "л", а другое ребро не изменяет. Но есть небольшое исключение. Оно заключается в том, что, если можно вставить в сторону вершины с типом "в" так, чтобы баланс принадлежал множеству {-1,0,1}, то автомат будет вставлять в эту сторону. Пример этого преобразования изображен на рисунке 6. Функции $\widetilde{\phi}$ отличатся от уже рассмотренных выше функций ϕ, описанным выше свойством, что если нет разницы куда вставлять, с точки зрения баланса, то лучше вставлять запись в сторону вершины "в".

Минимально возможный по степени ветвления информационный граф с радиусом видимости один, обрабатывающий произвольный поток запросов к динамической базе данных

Вершина слева или справа от корня имеет тип "в", другая тип "у"

Этот случай аналогичен предыдущему, за исключением того, что автомат меняет предикат, ведущий в вершину с типом "у".



$t^2 = (t_1^2, t_2^2, t_3^2) \quad z^2 = (l^2, r^2, a^2) \quad t^1 = (t_1^1, t_2^1, t_3^1) \quad z^1 = (l^1, r^1, a^1)$

Рис. 6: Преобразования на вставку. Первый такт, меняется одно ребро.

Вершина слева (справа) от корня имеет тип "л", вершина справа (слева) тип "у"

Этот случай аналогичен уже рассмотренному выше, когда одна из вершин имеет тип "в", а другая тип "л", но есть небольшое исключение. Разница заключается в том, что автомат меняет оба ребра, выходящих из корня на актуальные.

Напомним, что вставка всегда вставляет так, чтобы баланс после ее преобразования принадлежал множеству {-1,0,1}.

Вставка, второй такт

Автомат на вставку смотрит тип корня. Если тип корня равен "к", то автомат на вставку завершает функционирование.

Рассмотрим, оставшиеся случаи. Если тип корня не равен "к", то он обязательно равен "к$_{\leftarrow}$" или "к$_{\rightarrow}$". Разберем первый из них, второй разбирается аналогично. В этом случае возможно два варианта.

Вставка влево и вершина слева от корня имеет тип "л"

В этом случае автомат применяет преобразование, изображенное на рисунке 7. Ребро, нагрузка которого не меняется,

обозначено одной палочкой. Ребро, изображенное сплошным отрезком, не изменяется вообще.

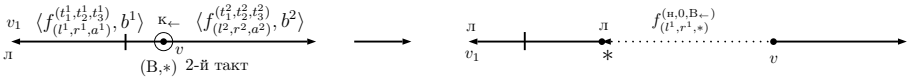


Рис. 7: Преобразование на вставку второй такт. Вставляется в сторону вершины с типом "л".

Вставка влево и вершина слева от корня имеет тип "в" или "у"

В этом случае автомат применяет преобразование, изображенное на рисунке 8. Автомат вставляет новую запись вместо вершины, которую нужно было удалить. Конфликтов не возникнет, так как автомат на удаление по типу корня поймет, что вставка произойдет в этот такт и не будет удалять свою вершину.

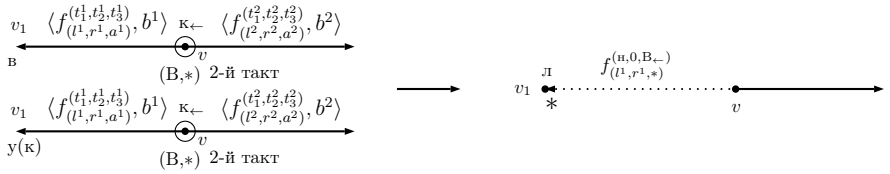


Рис. 8: Преобразование на вставку второй такт. Вставляется в сторону вершины с типом "в" или "у".

Опишем теперь преобразования, которые применяет автомат на удаление.

Удаление, первый такт

Сделаем небольшое замечание, связанное с пустыми запросами. Первый такт автомата на удаление, как и автомат на вставку, меняет тип корня, если нужно будет изменить базу данных и не нужно делать переброску вершины. Чтобы различить, кто именно поменял тип корня - автомат на вставку или

автомат на удаление - нужно посмотреть на параметр t_3 актуального ребра. Если его тип "У$_{\leftarrow}$" или "У$_{\rightarrow}$", то этот тип сделал автомат на удаление, иначе — автомат на вставку. Такая нагроможденная конструкция нужна лишь для того, чтобы обеспечить актуальность в случае пустых запросов, которые могли придти после автомата на удаление. В дальнейшем, если будем писать тип корня "к$_{\leftarrow}$" ("к$_{\rightarrow}$"), то это будет значить, что его создал автомат на вставку. Так же будем считать, что автомат на удаление меняет тип корня только на "к", поэтому в таблице 3 нет функции $\phi^v_{\text{У}}$. Это упростит доказательство, не ограничивая его в общности.

Как и другие автоматы, автомат на удаление меняет либо одно ребро, выходящее из корня, либо два. Одно ребро автомат меняет, если баланс принадлежит множеству {-1,0,1}, и выполнено одно из следующих условий: одна из соседних с корнем вершин имеет тип "в"; тип корня равен "к$_{\leftarrow}$" или "к$_{\rightarrow}$". В других случаях автомат меняет два ребра, выходящих из корня. Преобразование, соответствующее этому случаю, изображено на рисунке 9.

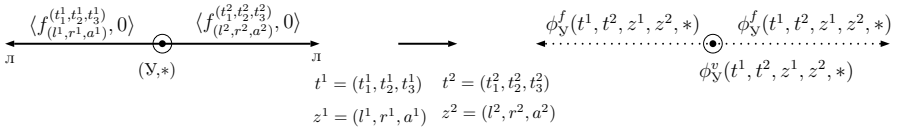


Рис. 9: Преобразование на удаление без переброски.

	t_2	$l \dot{:} P(*)$	$r \dot{:} P(*)$	$\phi^f_{\text{У}}$
1	-	нет	нет	$f^{(\text{а},\cdot,\text{н})}_{(l,r,\cdot)}$
2	-1,0,1	да	-	$f^{(\text{а},t_2+1,\text{У}_{\leftarrow})}_{(l/P(*),r,\cdot)}$
3	-1,0,1	-	да	$f^{(\text{а},t_2-1,\text{У}_{\rightarrow})}_{(l,r/P(*),\cdot)}$

(3)

Первая строчка таблицы 3 соответствует случаю, когда удаляемой записи нет в базе данных. Вторая и третья строчки соответствуют случаям, когда удаляемая запись есть в базе данных.

При балансе, равном 2 (-2), автомат на удаление ”перебрасывает” вершину справа (слева) от корня вправо (влево), меняет тип переброшенной вершины на ”в”, переходит в нее для последующего удаления во второй такт. Переброска осуществляется алгоритмом вставки, то есть считаем, что вставляется запись равная перебрасываемой вершине. Например, если перебрасываем вершину слева направо и справа от корня, вершина имеет тип ”у” или ”в”, то перебрасываемая вершина присваивается этой вершине. Иначе, создается новая вершина справа от корня, которой и присваивается перебрасываемое значение.

Тип корня меняется на ”к". Если значение обоих предикатов равно нулю, то типы вершин соседних с корнем не меняется. Отдельно опишем, как меняется тип вершин соседних с корнем, если значение предиката на запросе равен одному, то есть, удаляемая запись находится рядом с корнем. Если тип вершины в которую ведет это ребро имеет тип ”в”, то автомат не меняет ее. Если ее тип равен ”л”, то он меняет ее тип на ”у”, если удаляемая запись есть в базе данных, то есть функция, которая принимает тип вершины, зависит от того, есть удаляемая запись в базе данных или нет. Также автомат на удаление меняет вершины с типом ”у” на тип ”в”, если такая есть, и переходит в нее для последующего удаления.

Удаление, второй и последующие такты

Сначала разберем второй такт автомата на удаление. Как и в случае с поиском, автомат на удаление смотрит, есть ли запись, которую он удаляет, в базе данных или нет. Если есть, то в какой половине списка она находится (слева или справа от корня).

Если автомат находится в состоянии дополнительного удаления вершины, то он удаляет ее, согласуясь с правилами удаления вершины. После того, как он ее удалит или по алгоритму, он должен будет завершить функционирование, и автомат сразу переходит к удалению собственной вершины. Алгоритм удаления вершины, в которой находится автомат, будет описан ниже.

Минимально возможный по степени ветвления информационный граф с радиусом видимости один, обрабатывающий произвольный поток запросов к динамической базе данных

Опишем общий случай, когда автомату не нужно делать дополнительные удаления. Если удаляемой записи в базе данных нет, автомат на удаление завершает свое функционирование. Если запись есть, то возможны варианты.

Напомним, что из-за пустых запросов пришлось усложнить алгоритм автомата на удаление. А именно тем фактом, что они меняют тип корня, точно так же, как это делают автоматы на вставку. Автомат на удаление должен точно знать, этот тип сделал автомат на удаление или на вставку, когда он должен удалить вершину соседнюю с корнем. Это нужно, чтобы избежать конфликтов.

Если автомат на удаление должен удалить вершину рядом с корнем, то он должен будет понять, кто сделал тип "$У_{\leftarrow}$" или "$У_{\rightarrow}$" в корне. Если это сделал он, значит он сможет удалить вершину, соседнюю с корнем, иначе это сделал автомат на вставку, и в этом случае он не будет ее удалять. Автомат на удаление определяет этот факт, смотря на ребро, ведущее из корня в вершину, которое он должен удалить. Если ребро было актуальным, то его параметр t_3 должен был измениться на "$В_{\leftarrow}$" или "$В_{\rightarrow}$". Если же это ребро было не актуальным, а затем стало актуальным, с типом "$В_{\leftarrow}$" или "$В_{\rightarrow}$", то это опять признак того, что пришел автомат на вставку. В других случаях этот тип сделал автомат на удаление.

Перейдем к вариантам, когда автомат на удаление должен найти вершину. Пусть это вершина, которая находится слева от корня и соседняя с ним. Случай, когда эта правая вершина разбирается аналогично. В этом случае автомат меняет тип вершины слева от корня на "в", переходит в нее и меняет свое состояние на "удаление вершины". Возможен интересный случай, когда эта вершина на следующий такт уже будет находиться не рядом с корнем. Это могло произойти, если другой автомат на удаление применил преобразование переброски.

Если автомат продолжает функционирование, то он смотрит тип корня. Если он равен "$к_{\leftarrow}$", то он завершает свое функционирование, так как автомат на вставку вставит на место этой вершины новую запись. Так же если автомат видит, что баланс

стал равен -2, то он также завершает функционирование, так как следующий за ним автомат на преобразование базы данных уберет эту вершину. В других ситуациях, автомат на удаление смотрит тип вершины слева. Если он равен ”в”, то он остается в этой вершине, и на следующий такт повторяет тот же алгоритм, что и в предыдущий такт. Случай удаления вершины не вблизи корня отличается только тем, что не смотрится его тип и баланс.

Если вершину, которую должен удалить автомат, находится не рядом с корнем, то автомат переходит в состояние поиска этой вершины. Алгоритм поиска совпадает с уже описанном выше для автомата на поиск. Отличие состоит в том, что, когда автомат на удаление находит вершину, которую нужно удалить, он меняет ее тип на ”в” и переходит в нее. Далее смотрит, не будет ли удаляться вершина слева (справа) от него, если он двигался влево (вправо). Он понимает это так же по типу вершины. Если тип вершины слева (справа) равен ”в”, то значит она будет удаляться иначе нет. Если вершина будет удаляться, то автомат на удаление ждет такт, иначе он удаляет эту вершину. Начиная со следующего такта автомат повторяет алгоритм предыдущего такта. Ниже будет доказано, что максимальное количество подряд идущих удалений — три, поэтому автомат максимально может стоять два такта.

Отметим, что когда автомат удаляет вершину, вместе с ней он удаляет и ребро, которое ведет из этой вершины.

Отдельно стоит описать поведение автомата на вставку, когда, находясь в корне, автомат не может двигаться в сторону вершины, которую он должен удалить из-за вершины с типом ”в”. Если такая ситуация возникает, то автомат следит за типом корня. Допустим, автомат на вставку должен был двигаться влево от корня. Тогда, если он видит, что во время простоя в вершине, тип корня стал равен ”$к_{\leftarrow}$”, а вершина слева все еще имеет тип ”в”, то он переходит в специальное состояние, готовясь к следующему такту. Если предикат слева от корня будет равен 1 на запросе, то это не та запись, которую он должен удалить. Он должен ее воспринимать как обычную запись и

двигаться дальше к удалению собственной. В других случаях автомат на удаление действует стандартным образом.

Нужно сделать небольшое замечание в алгоритме удаления листовой вершины графа (самой левой или правой вершины графа), которое не меняет его смысл, но позволяет избежать конфликтов. При удалении самой левой или самой правой вершины графа, автомат удаляет только вершину и не удаляет ребро в нее ведущее. Автоматы же понимают, что если они видят ребро, которое не ведет в вершину, то значит они находятся в последней вершине графа. Такое необычное поведение нужно, чтобы избежать конфликтов изменения предпоследней вершины. Так как автомат на удаление может изменить ее тип на "в" в тот же такт, что и удаление последнего ребра в графе. Это не влияет на общее описание алгоритма, поэтому будем считать, что автомат может сразу удалить последнее ребро, подразумевая под этим алгоритм описанный выше.

Доказательство, что полученный МДИГ, удовлетворяет условию теоремы

Итак, осталось доказать, что приведенный выше алгоритм автоматов работает корректно и с заявленной сложностью. Для этого докажем несколько вспомогательных лемм.

Лемма 1. *Актуальное ребро определяется в каждый такт функционирования МДИГ — корректно.*

Доказательство. Эта лемма утверждает, что автомат может корректно определить актуальное ребро. Другими словами, если из корня выходит два ребра с типом t_1 равным "а", то значения t_2, l, r у них совпадают, кроме того эти значения соответствуют ИГ в данный так.

В случае пустой базы данных возможен случай, когда ребер нет, или есть одно ребро, ведущее в вершину с типом "у". В обоих случаях автомат не будет узнавать актуальное ребро, он понимает, что в данный так ИГ пуст и будет выполнять, описанный выше алгоритм в этих случаях. В частности, автоматы на поиск и удаление завершают свое функционирование, а автомат

на вставку вставляет свою запись вместо вершины с типом "у" и меняет предикат, ведущей к ней на актуальный.

Если база данных состоит из одной записи, то она могла возникнуть двумя способами. В первом — из пустой базы данных, добавлением новой записи, а во втором — удалением из двухэлементной базы данных одного элемента. Первый вариант уже разобран выше, в этом случае в ИГ всегда есть ровно одно актуальное ребро.

Рассмотрим теперь второй случай. Он совпадает и с общим случаем, поэтому объединим их в один.

Сначала разберем вариант, когда пустых запросов нет, то есть в каждый такт времени к базе данных поступает либо запрос на поиск, либо на вставку, либо на удаление. Заметим, что после первого такта автомата всегда существует как минимум одно актуальное ребро, поэтому если пустых запросов нет, то актуальное ребро будет. Кроме этого, если автоматы делают преобразования во второй и последующие такты, то предикаты, которые они создают, имеют тип t_1 равный "н", то есть не актуальный. Поэтому не возникнет ситуации, когда после первого такта автомата возникнет два актуальных ребра с разными параметрами.

Осталось рассмотреть вариант, когда есть пустые запросы. Здесь так же возможны подслучаи.

За автоматом на поиск поступил один или несколько пустых запросов. Автомат на поиск в первый такт своего функционирования создал одно или два актуальных ребра.

Допустим он создал одно актуальное ребро, а другое ребро не изменял. Такие ситуации описаны выше в алгоритме. Они возникают по нескольким причинам. Первая из них, если тип корня равен "к$_{\leftarrow}$" или "к$_{\rightarrow}$". Эта означает, что перед автоматом на поиск первый не пустой запрос был на вставку. Если тип корня равен "к$_{\leftarrow}$", то вершина справа от корня обязательно имеет тип "л" или "у". В этом можно убедиться, рассмотрев все преобразования на вставку. Таким образом, ребро справа от корня и вершина справа от него изменяться не будут до следующего запроса. Действительно, автоматы на поиск и удаление,

которые поступили раньше, уже не будут менять эту вершину и ребро. Следовательно, неважно, какое количество пустых запросов поступило к базе данных, все равно это актуальное ребро останется до следующего запроса.

Второй вариант, когда одна из вершин слева или справа от корня имела тип "в". Заметим, что в этом случае другая вершина обязательно имеет тип "л" или "у" в силу алгоритма автоматов. В этой ситуации опять же автомат на поиск сделает актуальным ребро ведущее в вершину с типом "л" или "у". Даже если до этого был автомат на удаление вершины с типом "л", соседней с корнем, то этот автомат на удаление не изменит ребра, в нее ведущее, а только поменяет тип вершины с "л" на "у". Следовательно, это ребро останется актуальным до следующего не пустого запроса.

Рассмотрим теперь случай, когда автомат на поиск создал сразу два актуальных ребра. Это означает, что тип вершин слева и справа от корня был "л" или "у", а так же тип корня "к". Аналогично разобранному выше случаю с одним актуальным ребром, на последующие такты может поменяться только тип вершин с "л" на "у", а оба ребра останутся актуальными до следующего не пустого запроса.

Разберем случай, когда за автоматом на вставку поступили один или несколько пустых запросов. В отличии от поиска автомат на вставку во второй такт своего функционирования может изменить ребро выходящее из корня на не актуальное. Поэтому, если до этого это было единственное актуальное ребро, и он его изменил на не актуальное, то оба ребра, выходящих из корня, будут иметь тип t_1 равный "н", то есть не актуальным. Но тип корня не поменялся и остался равным "к$_{\leftarrow}$" или "к$_{\rightarrow}$". "Стрелка" и будет индикатором актуальности ребра в этом случае, поэтому это вошло в определение актуального ребра. Если тип корня "к$_{\leftarrow}$", то актуальным считаем левое ребро иначе правое. Изменяться эти ребра больше не будут до следующего не пустого запроса.

Осталось рассмотреть случаи, когда за автоматом на удаление поступают пустые запросы. Этот случай аналогичен разо-

бранному выше случаю со вставкой. В определении преобразования автомата на удаление в самом начале была оговорка, что автомат на удаление точно так же, как и автомат на вставку, меняет тип корня. Смена типа корня позволит понять, какое ребро актуальное, даже если его тип t_1 равен "н". Лемма доказана. □

Лемма 2. *Справедливы следующие два утверждения. МДИГ функционирует бесконфликтно. Если автомат на удаление применяет преобразование переброски, то он обязательно перебросит существующую запись из базы данных.*

Доказательство. Рассмотрим первый такт произвольного автомата. Покажем, что преобразование, которое он применит не будет конфликтовать с другими автоматами. Алгоритмы всех автоматов в первый такт уже были описаны. Так же было пояснено, почему автомат меняет в некоторых случаях только одно ребро, выходящее из корня, а не два. Автомат делает это как раз для того, чтобы избежать конфликтов. Осталось только показать, что автомат сможет поменять хотя бы одно ребро. Рассмотрим все случаи, когда он не смог бы это сделать, и докажем, что таких ситуаций не возникнет во время функционирования. Таких вариантов три: вершина слева и справа от корня имеет тип "в"; вершина слева от корня имеет тип "в" и корень имеет тип "к$_{\rightarrow}$"; вершина справа от корня имеет тип "в" и корень имеет тип "к$_{\leftarrow}$". Докажем, что ни один их этих случаем не возможен.

Рассмотрим первый случай, когда вершина слева и справа от корня имеет тип "в". Заметим, что автоматы на поиск и пустые запросы не меняют типы вершин. Также видно, что автоматы на вставку не создают новых вершин с типом "у" и "в", а наоборот могут поменять тип "у" на "л" или "в" на "л". Алгоритм удаления устроен так, что если одна из соседних с корнем вершин имеет тип "в", а другая тип "л", то автомат на удаление либо оставит ей тип "л", либо сделает ей тип "у". Осталось рассмотреть последний случай, когда в первый такт своего функционирования автомат на удаление меняет тип "у" на "в", другая

вершина имела тип "в", и она не исчезает на следующий такт. Для этого нужно более детально посмотреть, когда появляется тип "в". Она появляется минимум на следующий такт за удалением, которое сделало тип "у". Вершина с типом "в" могла не удалиться по причине того, что рядом с ней была еще одна вершина с типом "в". Докажем, что это невозможно, то есть вершина с типом "в" данной ситуации обязательно успела бы удалится. Ниже будет доказано, что соседних вершин с типом "в" может быть не больше трех, следовательно, любая вершина с типом "в" исчезает не более, чем за три такта. Под исчезает понимаем два варианта: вершина удаляется или на ее место вставляется другая запись. Вершина с типом "в" рядом с корнем появляется минимум за два такта. Поэтому на третий такт она обязательно удалится.

Рассмотрим второй (третий) случай, когда вершина слева (справа) от корня имеет тип "в", а корень имеет тип "$к_{\rightarrow}$" ("$к_{\leftarrow}$"). Оба варианта симметричны, поэтому разберем случай, когда вершина слева от корня имеет тип "в", а корень имеет тип "$к_{\rightarrow}$". Допустим такая ситуация возможна. Понятно, что одновременно за один такт такого не могло произойти. Тогда нужно разобрать два варианта. Сначала тип вершины слева от корня стал равен "в", а затем тип корня стал равен "$к_{\rightarrow}$", и, наоборот, сначала тип корня стал равен "$к_{\rightarrow}$", а затем вершина слева от корня стала иметь тип "в".

Пусть сначала тип корня стал равен "$к_{\rightarrow}$". Тогда, чтобы тип вершины слева от корня стал равен "в", должен придти запрос на удаление. В другом случае он не примет этот тип. Если поступит запрос на удаление, то он поменяет тип корня на "к".

Рассмотрим теперь случай, когда вершина слева от корня имеет тип "в". В этом случае автомат на вставку сможет только сделать тип корня равным "$к_{\rightarrow}$". Но как уже было разобрано выше вершина с типом "в" удаляется за один такт, поэтому в этом случае не получится конфигурации, когда вершина слева имеет тип "в", а корень имеет тип "$к_{\rightarrow}$".

Итак, разобраны все случаи, когда автомат не сможет поменять ребро в первый такт своего функционирования. Доказано,

что таких конфигураций не возникнет, поэтому любой автомат в первый такт своего времени не образует конфликтов с другими автоматами. Рассмотрим преобразования, начиная со второго такта его функционирования. Если это автомат на поиск, то он не образует конфликтов, так как только перемещается по ИГ, с учетом того, что в алгоритме поиска запрещается переходить в вершину, которая возможно будет удаляться. Автомат на вставку делает преобразование только во второй такт функционирования. При этом он сообщает автоматам, куда он собирается вставить запись. Поэтому автомат на вставку не образует конфликтов.

Осталось рассмотреть автомат на удаление. Автомат на удаление может изменить тип вершин. Но это не вызывает конфликтов. Этот случай уже подробно разобран в самом алгоритме удаления. Аналогично автомату на поиск, перемещение по графу автомата на удаление так же не конфликтует с другими автоматами. Преобразование, которое удаляет вершину, так же подробно разобрано в алгоритме удаления. Таки образом, доказано, что автоматы не конфликтуют.

Докажем теперь второе утверждение леммы 2. Для этого нужно доказать, что не возникнет конфигураций, когда автомат на удаление должен будет перебросить вершину типа ”в” или ”у” через корень. Не ограничивая общности рассуждения, нужно перебросить вершину слева от корня вправо. В этом случае баланс должен быть равен -2, чтобы случилась переброска слева направо.

Рассмотрим первый случай. Автомат должен перебросить вершину с типом ”у” при балансе -2.

Вершина с типом ”у” появляется только после первого такта. В этом случае баланс стал равен не меньше 0 , так как удаление в левой половине увеличивает баланс на единицу. Баланс не мог бы стать -1, так как до этого он был бы равен -2, а при таком балансе автомат применил бы переброску и вершина с типом ”у” не появилась бы. Если баланс стал равен 0 (1), то должно придти как минимум 2 (3) удаления, чтобы баланс стал равен -2. За два удаления вершина с типа ”у” уже точно поменяет на тип

”в”, а этот случай уже был разобран выше и доказано, что не бывает ситуаций, когда вершина с типом ”в” перебрасывается.

Осталось доказать факт, что не бывает трех подряд вершин на удаление. Этот факт следует из алгоритма переброски. В одну сторону графа может пойти не более трех подряд удалений, после чего обязательно произойдет переброска. Действительно, переброска происходит при балансе 2 или -2. Каждое удаление вправой (влевой) части уменьшает (увеличивает) баланс на 1. Следовательно, если все запросы на удаление поступают в одну сторону, то максимальное количество равняется 3, так как -2 = 1 - 3 и 2 = -1 + 3. Во время переброски, автомат удаляется от группы удалений на 2 ребра. Причем баланс становиться равен -1 (1), а это значит, что в эту же сторону можно послать только один запрос на удаление, после чего произойдет переброска, и опять будет удаление на 2 вершины. Следовательно, группы более чем из трех удалений образоваться не может. Лемма доказана. □

Лемма 3. *Баланс после любого запроса будет принадлежать множеству {-2,-1,0,1,2}. Причем после преобразования на вставку баланс принадлежит множеству {-1,0,1}.*

Доказательство. Данная лемма предполагает, что все преобразования корректны. Этот факт доказан в лемме 2.

Лемму будем доказывать индукцией по тактам для любого потока запроса. Если такт равен 0,1 то в базе данных не более одной записи и утверждение леммы очевидно.

Рассмотрим произвольный такт. Из алгоритма на поиск следует, что автоматы на поиск не меняют баланс, пустые запросы так же его не меняют, поэтому осталось рассмотреть случай, когда к базе данных поступил запрос на вставку или на удаление.

Нетрудно заметить, что если нет запросов на удаление, а есть только запросы на вставку, то баланс будет принадлежать множеству {-1,0,1}. Это свойство написано в алгоритме вставки.

Допустим, пришел первый запрос на удаление. В момент его поступления баланс принадлежал множеству {-1,0,1}, причем

тип вершин слева и справа от корня равнялся ”л”. После удаления возможно два варианта. Либо баланс остался во множестве {-1,0,1}, либо стал равен 2 или -2, что удовлетворяет предположению индукции.

Пусть поступил очередной запрос на удаление. Разберем случай, когда баланс принадлежит множеству {-1,0,1}. В этом случае после преобразования на удаление баланс будет принадлежать множеству {-2,-1,0,1,2}, что удовлетворяет предположению индукции. Если баланс был равен -2 или 2, то автомат на удаление в любом случае применяет преобразование переброски, которое делает баланс равным 0. Далее, если ему нужно будет удалить запись, то баланс станет равным либо 1, либо -1. Если запись удалять не нужно, то баланс станет равным 0. То есть предположение индукции выполняется и в этом случае. Лемма доказана. □

Лемма 4. *Автомат на поиск работает корректно. То есть автомат на поиск x, выдает ответ $\emptyset$ если искомой записи нет и $\{x\}$, если запись есть.*

Доказательство. Автомат на поиск после первого такта узнает есть искомая запись в базе данных или нет. Если ее нет, то выдает в качестве ответа $\emptyset$. Заметим, что из леммы 1 следует, что это правильный ответ. Если запись есть, то он начинает ее искать. В алгоритме поиска уже было детально описано, почему вставка не сможет помешать поиску найти свою запись. Если автомат на поиск оказывается в одной вершине с автоматом на удаление этой же вершины, то он все равно выдаст в ответ $\{x\}$, так как автомат на удаление в этот такт только присвоит этой вершине тип ”в” или ”у”. □

Оценка сложности поиска является нетрудным следствием вышеперечисленных лемм. Главное здесь, что баланс принадлежит множеству {-2,-1,0,1,2}. Если в базе данных n записей, то в одной половине при таком балансе не больше, чем $]n/2[+2$. Автомат на поиск мог затормозить свое передвижение не более, чем на три такта. Следовательно, если автомат на поиск

подождет три такта, то он не догонит ни одну другую группу удалений и не будет больше простаивать. Поэтому количество тактов, необходимое на выдачу ответа, не превосходит $]n/2[+5$. На рисунке 10 приведен пример функционирования МДИГ.

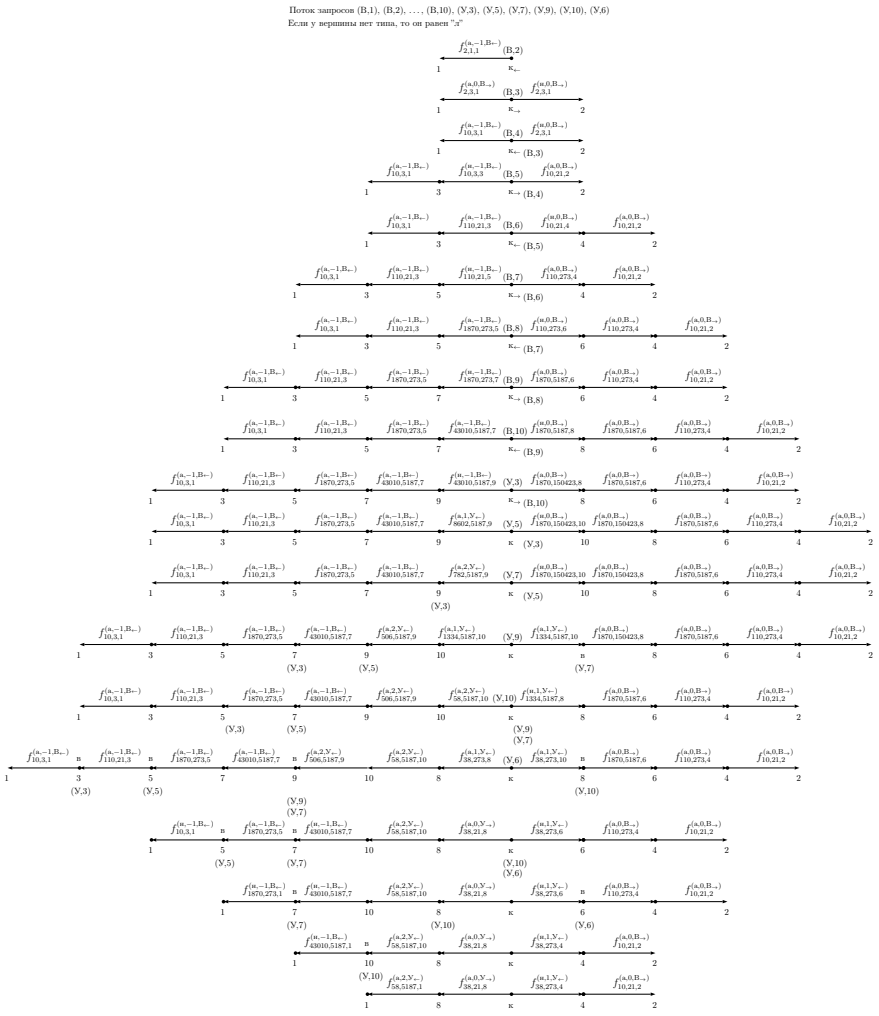


Рис. 10: Пример функционирования МДИГ.

Теорема доказана. □

Список литературы

[1] Плетнев А.А. Нижняя оценка на область видимости автомата, обрабатывающего произвольный поток запросов к динамической базе данных// Интеллектуальные системы. — 2016. Т. 19, Вып. 4.

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

[3] Гасанов Э.Э., Ефремов Д.В. Фоновый алгоритм решения двумерной задачи о доминировании // Интеллектуальные системы. — 2014. — Т. 18, вып. 3. — С. 133–158.

[4] Е. М. Перпер. Нижние оценки временной и объёмной сложности задачи поиска подслова // Дискретная математика, 2014, том 26:2, 58–70.

[5] Шуткин Ю.С. Моделирование схемных управляющих систем // Интеллектуальные системы. — 2014. — Т. 18, вып. 3. — С. 253–261.

[6] Перпер Е.М. Порядок сложности задачи поиска в множестве слов вхождений подслова // Интеллектуальные системы. — 2014. — Т. 19, вып. 1. — С. 99–116.

[7] Плетнев А.А. Динамическая база данных, допускающая параллельную обработку произвольных потоков запросов// Интеллектуальные системы. — 2015. Т. 19, Вып. 1. — С. 117–145.

[8] E. Gafni, J. Naor, P. Ragde On Separating the EREW and CREW PRAM Models, Theoretical Computer Science 68 (1989) 343-346.

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

[10] Плетнев А.А. Логарифмическая по сложности параллельная обработка автоматами произвольных потоков запросов в динамической базе данных// Интеллектуальные системы. — 2015. Т. 19, Вып. 1. — С. 171–213.

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

[12] Титова Е.Е. Конструирование движущихся изображений клеточными автоматами // Интеллектуальные системы. — 2014. — Т. 18, вып. 1. — С. 153–180.

[13] В.Б.Кудрявцев. Кафедра математической теории интеллектуальных систем (МаТИС) // Интеллектуальные системы. — 2014. — Т. 18, вып. 2. — С. 5–30.

[14] Летуновский А.А. Выразимость линейных автоматов относительно расширенной суперпозиции // Интеллектуальные системы. — 2015. — Т. 19, вып. 1. — С. 161–170.

[15] Гасанов Э.Э. Прогнозирование периодических сверхсобытий автоматами // Интеллектуальные системы. — 2015. — Т. 19, вып. 1. — С. 23–34.

[16] Бабин Д.Н. Автоматы с суперпозициями, пример нерасширяемости до предполного класса // Интеллектуальные системы. — 2015. — Т. 19, вып. 3. — С. 87 – 94.

[17] Гасанов Э.Э., Мастихина А.А. Прогнозирование общерегулярных сверхсобытий автоматами // Интеллектуальные системы. — 2015. — Т. 19, вып. 3. — С. 127–153.

[18] Часовских А.А. Критериальные системы в классах линейно-автоматных функций над конечными полями // Интеллектуальные системы. — 2015. — Т. 19, вып. 3. — С. 195 – 207.