Логарифмическая по сложности параллельная обработка автоматами произвольных потоков запросов в динамической базе данных

А. А. Плетнев

В данной работе рассматривается решение динамической задачи поиска идентичных объектов для произвольного потока запросов с логарифмической сложностью. 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 многоавтоматного динамического информационного графа [1].

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

1. Введение

Существует множество различных способов решения задачи поиска идентичных объектов. В основном все эти решения основываются на деревьях поиска. В частности, эту задачу решают с помощью сбалансированных деревьев поиска, таких как сбалансированное бинарное дерево, красно-черное дерево, B-дерево и его разновидности ($B+, B^*$). Так же отметим, 2-3 и 2-3-4 деревья, как частные случаи B-дерева. Особенность этих структур заключается в том, что скорость поиска, вставки и удаления записей в них логарифмически зависит от объема базы данных.

В данной работе будет использоваться 2-3-4 дерево. Поэтому напомним, что представляют из себя B-деревья. B-деревья — это

сбалансированные деревья поиска, являющиеся естественным обобщением бинарных деревьев поиска. Сбалансированные деревья и B-деревья достаточно подробно рассмотрены в книгах Кормена (Cormen) [2], Кнута (Knuth) [3], Ахо (Aho) , Хопкрофта (Hopcroft), Ульмана (Ulman) [4] и Седжвика (Sedgewick) [5]. Подробный обзор B-деревьев дан Комером (Comer) [6]. Гибас (Guibas) и Седжвик [7] рассмотрели взаимосвязи между различными видами сбалансированных деревьев, включая красно-черные деревья и 2-3-4 деревья.

С развитием компьютеров появилась возможность использовать несколько процессов одновременно, что позволяет убыстрять алгоритмы. Это означает, что появилась возможность работать (искать, преобразовывать) над общей структурой данных нескольким запросам одновременно, не дожидаясь завершения каждого в отдельности. Применительно к сбалансированным деревьям поиска, в частности, к красно-черным и 2-3 деревьям было проведено много исследований [8], [9]. Цель этих исследований была в построении дерева, за наименьшее время, для заданного отсортированного массива данных.

Естественным ограничением на параллельные вычисления является доступ к общей памяти. В зависимости от этого получаются принципиально разные результаты. Здесь будем рассматривать ОЧЭЗ (одновременное чтение, эксклюзивная запись) структуры данных. В данной системе разрешается одновременное чтение из общей памяти, при этом одновременная запись разными процессами в общую память запрещена. Подробнее можно прочитать в [10], [11].

В данной работе будет рассмотрена постановка задачи, совпадающая с постановкой задачи, описанной в [1]. Постановка задачи занимает большой объем, поэтому здесь опишем только ее смысл, а точную формулировку можно прочитать в [1].

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. 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 идентичных объектов, с логарифмической сложностью, для любого потока запросов. Идея построения искомой структуры данных основывается на информационно-графовом (ИГ) подходе [12], с использованием конечного автомата [13].

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

2. 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 все 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

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

Функция V 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 $H(i) = (\text{П}, x)$, 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

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

$$f_a^{\text{П}}(x) = f_a^{\text{В}}(x) = f_a^{\text{У}}(x) = \begin{cases} 1, & \text{если } x = a; \\ 0, & \text{иначе} \end{cases}; f^h(x) \equiv 0;$$

$$F = \left\{ f_a(x), f_a^h(x), f_{=,a}(x), f_a^{\text{П}}, f_a^{\text{В}}, f_a^{\text{У}} : a \in \mathbb{R} \right\} \cup \{f^h\}.$$

Так же, будем рассматривать базовое множества преобразований $\mathcal{R}$, которое будет строго определено ниже, в доказательстве теоремы 1.

В качестве базового множества для МДИГ будем рассматривать множество

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

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. Поэтому автомат, во время функционирования, видит значение этих функций на запросе, который он обслуживает, а не сами функции. Так же помимо значений функций на запросе, автомат видит структуру подграфа ИГ. Подграф, который видит автомат, ограничен фиксированным параметром R, который означает радиус окрестности с центром в вершине в которой находится автомат. 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 не сможет увидеть окрестность даже 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.

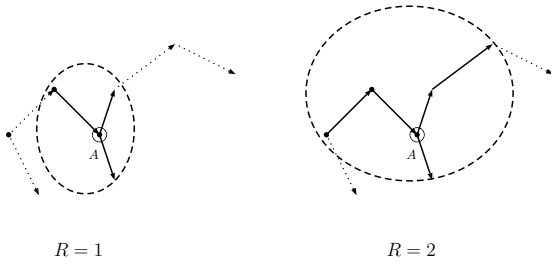

Рис. 1. Пример радиуса видимости автомата.

Будет говорить, что МДИГ типа (N, R), если ИГ степень ветвления не больше, чем N, а радиус видимости автомата не больше, чем R. Более подробно как именно автомат видит окрестность, можно прочитать в [1].

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

Теорема 1. *Существует МДИГ $\mathcal{D} = (\mathcal{A}, U)$ типа (8,4) над базовым множеством $\mathcal{F}$, определяемым соотношением (1), который решает ДЗПИО для любого потока запросов $H, H \in \mathcal{H}$. При этом для любого натурального i количество тактов, необходимое на выдачу ответа на поисковый запрос $H(i)$, не превосходит*

$$\left]\frac{\log_2(\max\{|V(i,H)|, 2\})}{3}\right[.$$

3. Доказательство теоремы 1

Структура доказательства будет следующей. Сначала будет описан алгоритм для 2-3-4 дерева в классической форме и на языке информационного графа. Преобразования, которые будут применять автоматы, будут связаны с алгоритмами вставки и удаления в 2-3-4 дереве. Поэтому после описания алгоритма 2-3-4 дерева, опишем преобразования, которые будут применять автоматы. Далее покажем, что эти преобразования не вызывают конфликтов, обусловленных эксклюзивной записью, и в заключении оценим сложность полученного алгоритма.

Опишем сначала алгоритм 2-3-4 дерева. Структура данных 2-3-4 дерево является частным случаем B-деревьев. В [14] был приведен ДИГ, который поддерживал базу данных, решая ДЗПИО, для одиночных запросов. Алгоритм ДИГ основывался на структуре данных 2-3 дерево. Для потоковых запросов необходимо, чтобы вставка и удаление могли осуществляться за один нисходящий проход от корня к листьям. Поэтому здесь используется 2-3-4 дерево.

B-дерево можно рассматривать как иерархический индекс. Корень B-дерева является индексом первого уровня. Каждый нелистовой узел на B-дереве имеет форму $(p_0, k_1, p_1, k_2, p_2, \ldots, k_n, p_n)$, где p_i является указателем на i-го сына, $0 < i \leqslant n$, а k_i — ключ, $1 < i \leqslant n$. Ключи в узле упорядочены, так что $k_1 < k_2 < \cdots < k_n$. Все ключи в поддереве, на который указывает p_0, меньше или равно k_1. В случае $1 < i < n$ все ключи в поддереве, на который указывает p_i, принадлежат полуинтервалу $(k_i, k_{i+1}]$. Все ключи в поддереве, на который указывает p_n, больше k_n. В случае 2-3-4 дерева, $n = 3$. Пример 2-3-4 дерева изображен на рисунке 2. На этом рисунке узлы изображены сокращенно, например, в корне должны быть еще пустые ячейки k_2, p_2, k_3, p_3.

Поиск записи r с ключом x осуществляется по следующему алгоритму. Нужно проследить путь от корня к листу, который содержит r, если такая запись существует в базе данных. Каждый шаг вычисляется положение x в узле ключей. Если $x < k_0$ ($k_i \leqslant x < k_{i+1}$ или $x \geqslant k_n$), то на следующий шаг будет рассмотрен узел, на который указывает p_0 (p_i или p_n). Шаг, на котором x ищется в листе, — заключительный. B-дерево, все записи, которого хранятся в листовых

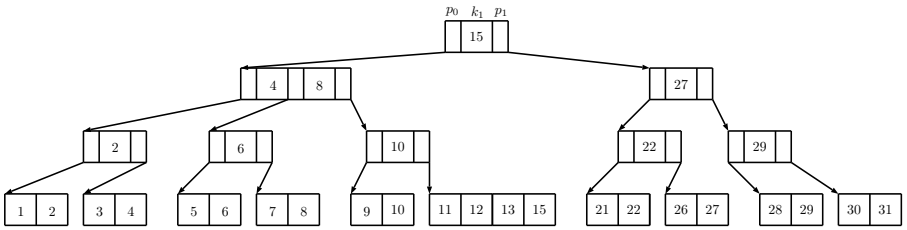

Рис. 2. Пример 2-3-4 дерева.

вершинах, так же называют $B+$ дерево. Так же, в этой работе будем считать, что множество ключей совпадает с множеством записей.

Существуют различные алгоритмы для вставки и удаления из B-дерева. Здесь, важны алгоритмы, которые позволяют вставить и удалить из дерева за один нисходящий проход. Такой алгоритм для вставки существует [2], но для удаления автору работы не удалось найти явную ссылку на такой алгоритм. На самом деле, его можно получить из уже существующих идей. Рассмотрим алгоритм, описанный в [2]. Этот алгоритм в некоторых случаях требует замены в узле ключа, для которого необходимо повторное прохождение от листа к этому узлу. Проблема заключалась в том, что было рассмотрено B-дерево, которое хранило записи в узловых вершинах. Если рассмотреть дерево, которое описано выше, так называемое $B+$ дерево (все данных хранятся в листьях), то можно сделать удаление за один проход от корня к листу. Удалось заметить, что, используя алгоритм удаления, описанный в [2], на $B+$ дереве, не обязательно совершать замену удаленного ключа в узле. Естественно, итоговое 2-3-4 дерево будет отличаться от 2-3-4 дерева, полученного в результате известного алгоритма. Но 2-3-4 дерево останется деревом, которое будет корректно работать. Тем самым, можно осуществить удаление из $B+$ дерева за один нисходящий проход, что является ключевым моментом в его распараллеливании, с помощью автоматов.

Напомним известный алгоритм вставки в 2-3-4 дереве, за один нисходящий проход, и, предлагаемый в данной работе, алгоритм удаления из 2-3-4 дерева, за один нисходящий проход.

Начнем со вставки. Алгоритм вставки в 2-3-4 дерево следующий. Он повторяет алгоритм поиска за некоторыми отличиями. Первое отличие заключается в том, что этот алгоритм требует, чтобы каж-

дый узел который он посещает, не был полностью заполнен, то есть имел бы не более трех сыновей. Если узел полностью заполнен (имеет ровно 4 сына), то алгоритм делает перестройку. Допустим, вставка находится в корне, и он имеет ровно 4 сына. В этом случае создается новый корень, а узел разбивается на два новых. Пример такого преобразования изображен на рисунке 3.

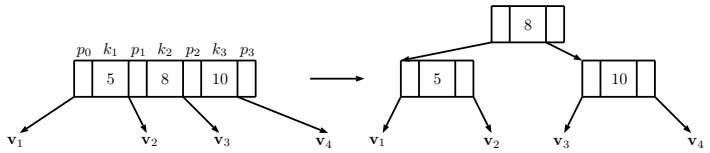

Рис. 3. Пример вставки в 2-3-4 дереве. Случай корня.

Далее по индукции можно считать, что родитель узла всегда не более трех сыновей. Рассмотрим теперь случай, когда вставка находится во внутреннем узле, и он имеет 4 потомка. В этом случае делается преобразование, изображенное на рисунке 4. Оно разбивает узел на два узла. Это можно сделать, так как отец рассматриваемого узла не полностью заполнен, следовательно, ему можно добавить еще одного сына.

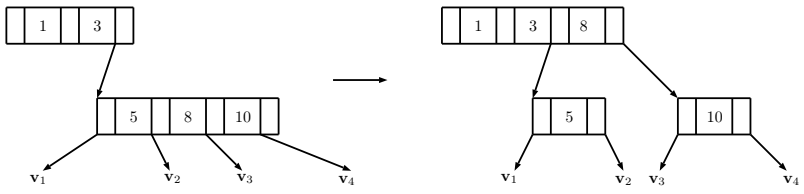

Рис. 4. Пример вставки в 2-3-4 дереве. Случай внутреннего узла.

Второе отличие алгоритма вставки от алгоритма поиска заключается в том, когда вставка оказывается в листе и не находит запись. Если запись уже есть, то алгоритм завершается. Если же записи нет и лист не заполнен, то вставляется запись так, чтобы сохранить порядок ключей. Если же лист полностью заполнен, то делается перестройка, аналогичная перестройке в узле, описанной выше, и после этого запись вставляется в нужное место. Пример этого случая изоб-

ражен на рисунке 5. Заметим, что этот алгоритм позволяет вставить новую запись в 2-3-4 дерево за один нисходящий проход.

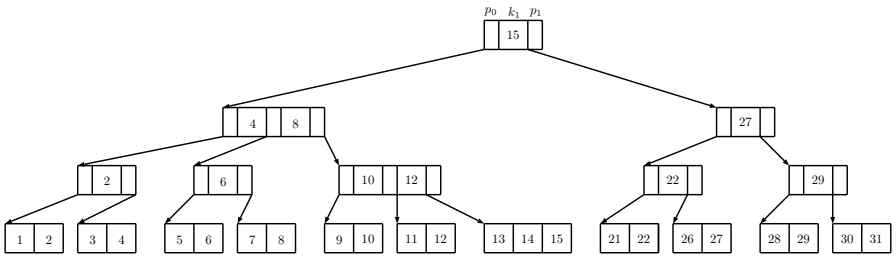

Рис. 5. Пример вставки в 2-3-4 дереве. Случай листа. Добавлена запись 14 к примеру 2-3-4 дерева, изображенного на рисунке 2.

Теперь опишем алгоритм удаления. Он, как и алгоритм вставки, повторяет алгоритм поиска за некоторым исключением. Первое отличие заключается в том, что этот алгоритм требует, чтобы в узле, в котором он находится, было не меньше двух ключей, то есть не меньше трех сыновей. Рассмотрим случай корня. Если у корня есть двое сыновей, каждый из которых содержит только по одному ключу (двое детей), то применяется преобразование, изображенное на рисунке 6. Оно удаляет корень, уменьшая при этом высоту дерева, и создает новый корень, объединяя ключи старого корня и его детей и записывая их в новый корень. Если же у одного из детей есть три сына, то алгоритм может применить преобразование переброски детей, изображенное на рисунке 7, если он должен перейти в узел, содержащий только один ключ. Если же на следующий шаг удаление переходит, в вершину, у которой не меньше трех сыновей, то преобразование не применяется.

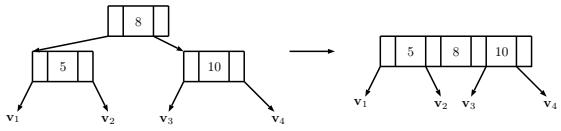

Рис. 6. Пример удаления в 2-3-4 дереве. Случай корня.

Далее по индукции можно считать, что родитель узла содержит не менее трех сыновей. Если у рассматриваемого узла есть брат, ко-

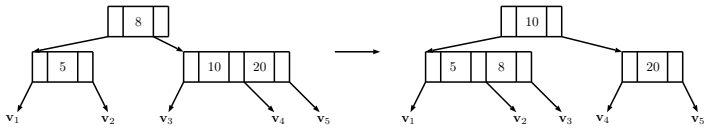

Рис. 7. Пример удаления в 2-3-4 дереве. Случай переброски детей. Это необходимо, если следующий шаг автомата на удаление оказывается в узле с одним ключом.

торый содержит не менее двух ключей (трех сыновей), то применяется преобразование переброски детей. Возможно, придется применить его не один раз, если братья не соседние. Разберем случай, когда все братья содержат ровно по два сына. Тогда применяется преобразование, изображенное на рисунке 8. Это преобразование объединяет двух братьев в один узел. Это можно сделать в силу того, что у отца есть не менее трех сыновей.

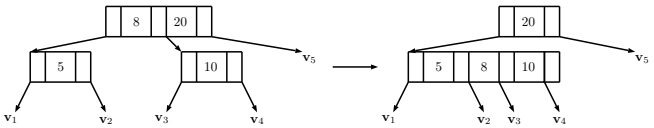

Рис. 8. Пример удаления в 2-3-4 дереве. Случай объединения братьев. Алгоритм переходит в вершину с ключом 5.

Второе отличие алгоритма на удаление от алгоритма на поиск заключается в последнем шаге алгоритма. Если алгоритм на удаление не находит запись, то он завершается. Если находит, то он удаляет ее, при необходимости применяя перестройки, описанные выше, чтобы сохранить структуру 2-3-4 дерева. Удаление записи 15 из дерева, изображенного на рисунке 2, представлено на рисунке 9. На этом рисунке изображено отличие от стандартного алгоритма удаления в 2-3-4 дереве за один нисходящий проход, описанное в [2]. Предлагается не заменять ключ 15, находящийся в корне на 13, а оставить его. Ключ 15 в корне не будет означать максимальный элемент в левом поддереве. Но это не мешает поиску правильно находить нужный запрос, так как в листьях идет проверка на равенство. Это наблюдение позволяет сделать алгоритм удаления за один нисходящий проход без исключений.

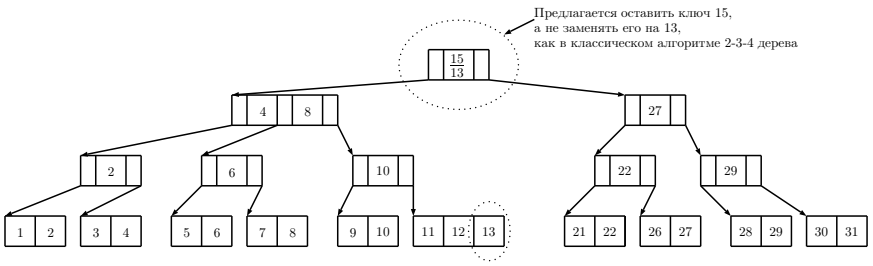

Рис. 9. Пример удаления в 2-3-4 дереве. Случай листа. Удаление записи 15 из примера, изображенного на рисунке 2. Отличие от стандартного алгоритма.

Для распараллеливания этого алгоритма на случай произвольного потока запроса переведем структуру 2-3-4 дерево на язык информационного графа. Проведем следующую аналогию между классическим 2-3-4 деревом и информационным графом. Узлу будет соответствовать предикатная вершина. Ключам в узле будут соответствовать предикатные ребра первого типа f_k. Пример соответствия между классическим узлом в 2-3-4 дереве и его аналогом на языке информационного графа, изображен на рисунке 10. Заметим, что самый правый предикат (f_{30}) означает, что максимальным элемент в правом поддереве 30. В классическом 2-3-4 дереве не хранится максимальный элемент. Можно было бы заменить f_{30} на предикат тождественно равный 1, но так удобнее объяснять преобразования, которые будут применять автоматы.

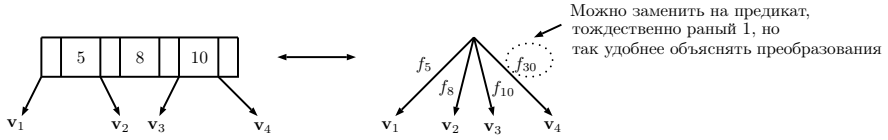

Рис. 10. Соответствие между узлом 2-3-4 дерева и его аналогом в информационном графе.

Листьям 2-3-4 дерева, будут соответствовать пара предикатного ребра первого типа (f_k) и предикатного ребра второго типа ($f_{=,k}$). Напомним, что автомат знает не только значение предиката на запросе, но и его тип. Это возможно так как количество типов преди-

катов конечно. В данной теореме это количество равно 7. Это сделано для удобства описания преобразований с помощью автоматов. Пример соответствия между классическим листом в 2-3-4 дереве и его аналогом на языке информационного графа, изображен на рисунке 11. Обратим внимание, что предикаты первого уровня для листа не обязательны, но это упростит преобразование для автомата, так как он сможет найти правильное место для вставки новой записи, используя эти предикаты. Под правильным понимается место, которое не нарушает порядок ключей в листе (от меньшего к большему).

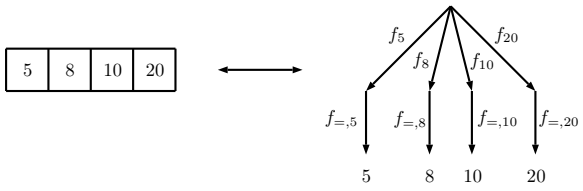

Рис. 11. Соответствие между листом 2-3-4 дерева и его аналогом в информационном графе.

Информационный граф соответствующий примеру 2-3-4 дерева, изображенного на рисунке 2, представлен на рисунке 12. На нем не изображены предикаты второго типа, но их можно однозначно восстановить по записям, к которым они ведут.

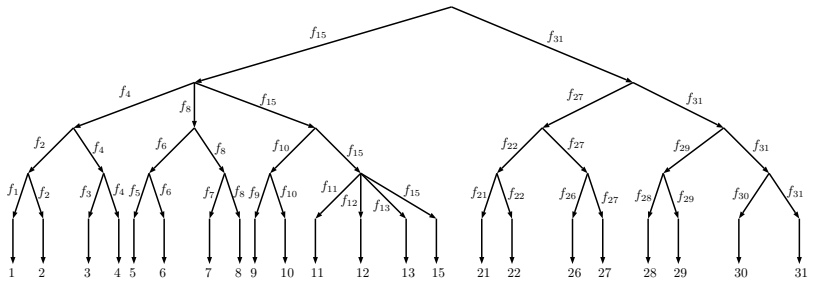

Рис. 12. Информационный граф, соответствующий 2-3-4 дереву, изображенному на рисунке 2.

Прежде чем перейти к описанию преобразований, которые будут использовать автоматы для перестроения ИГ, опишем их смысл. Один шаг алгоритма на поиск заключается в прохождении одного

уровня дерева. При этом он переходит по самому левому ребру, код которого равен 1. Если таких ребер нет, то самым левым считаем самое правое ребро. Например для ИГ, изображенного на рисунке 12, автомат на поиск 15 пошел по левому ребру, а на поиск 30 пошел бы по правому ребру. Алгоритм преобразований основывается на классических перестройках 2-3-4 дерева, которые были описаны выше.

Например, рассмотрим автомат на вставку. Один шаг автомата на вставку аналогичен шагу автомата на поиск, если автомат переходит в вершину, у которой не более двух братьев. Если же нет, то автомат применят одно из преобразований 2-3-4 дерева, описанных выше, для его перестройки, а после переходит в эту вершину. Аналогично поступает автомат на удаление. Но если автоматы за один такт будут делать только один шаг алгоритма, то не удается избежать конфликтов, связанных с эксклюзивностью записи. Два автомата могут перестроить одну и ту же вершину или преобразовать одно и тоже ребро и так далее. Поэтому, чтобы избежать конфликтов, автомат за один такт функционирования будет делать, по возможности, 3 шага алгоритма. Отдельно разбираются случаи вблизи листьев, когда автомат должен завершить функционирование.

Не трудно увидеть, что если автомат каждый такт перемещается ровно на 3 уровня вниз, то расстояние между различными автоматами не меньше 3 ребер. Но при удалении корня может возникнуть конфигурация, в которой расстояние между автоматами станет 2 ребра. Разбор этих ситуаций будет произведен ниже. Заметим только, что корень не может удалиться два такта подряд. Действительно, после удаления корня и перемещения автомата из корня будет выходить минимум 3 ребра. Поэтому корень не может быть удален два такта подряд.

В случае удаления корня могут возникнуть конфликты вблизи листьев. Чтобы их избежать, автоматом необходимо знать дополнительную информацию. А именно, автомат должен знать, есть ли в его окрестности другие автоматы, какие типы запросов они выполняют (поиск, вставка, удаление) и сравнить, совпадает ли обслуживаемый запрос с запросом других автоматов. Для получения этой информации, предназначены три специальных типа предикатов. Номера их типов: 4 ($f_a^{\text{П}}$), 5 ($f_a^{\text{В}}$) и 6 ($f_a^{\text{У}}$). В каждой вершине графа добавим две петли. Если в вершине нет ни одного автомата, то петлям будет присвоены предикаты 7-го типа f^h. Автоматы при переходе в новую

вершин, будут изменять значение одной из петель, нагрузка которой есть f^h, на предикат одного из типов 4, 5 или 6, аргументом которого будет запрос, который обслуживает автомат. Если автомат был автомат на поиск (вставку или удаление), то тип предиката будет 4 (5 или 6). При уходе из вершины, автомат будет изменять предикат на петле, на f^h. Пример изменения нагрузок петель изображен на рисунке 13.

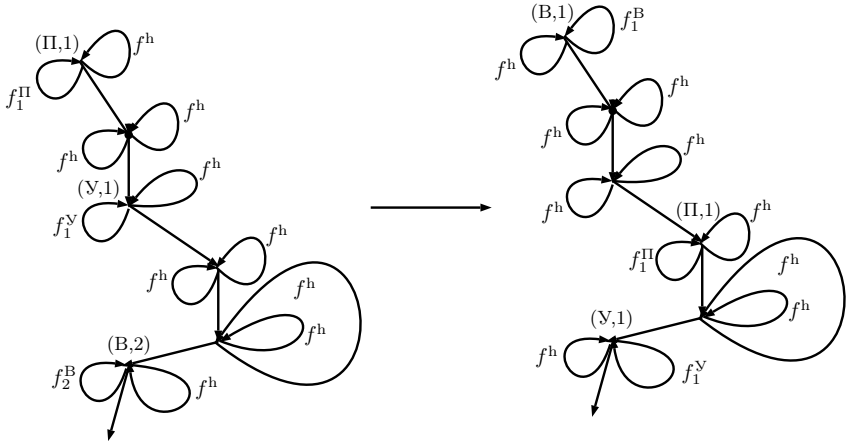

Рис. 13. Пример изменения нагрузок петель.

Заметим, что при изменении нагрузок петель конфликтов не возникает, так как у каждой вершины есть обязательно одна петля с нагрузкой f^h. Тем самым, автоматы не конфликтуют, изменяя нагрузки петель. Все предыдущие преобразования, на самом деле, содержат в каждой вершине по 2 петли. Это не было обозначено на рисунках, чтобы не загромождать их. Далее будем считать, что автомат знает информацию об наличии других автоматов в области своей видимости, и совпадают ли у них запросы или нет.

Зная эту информацию, некоторые автоматы могут закончить свое функционирование на первом такте. Будем говорить, что один автомат видит *впереди* другой автомат, если тот находится ближе к листьям. Это может произойти, только если другой автомат начал свое функционирование ровно на 1 такт раньше. Аналогично скажем, что автомат видит другой автомат *сзади*, если тот поступил на 1 такт позже, и находиться в его области видимости.

Если автомат на поиск x видит впереди автомат на удаление x, то он может завершить свое функционирование с сигналом $e = 2$(запись не найдена). Так же, если автомат на преобразование видит впереди другой автомат с тем же заданием, то он может завершить функционирование с соответствующим сигналом. Если это автомат на вставку, то он завершает функционирование с сигналом $e = 3$ (запись уже есть), а если это автомат на удаление, то с сигналом $e = 2$ (запись не найдена). В случае, когда автомат на удаление(вставку) x видит сзади другой автомат на вставку(удаление) x, то он завершает свое функционирование.

Кроме этого автомату требуется завершить функционирование, если расстояние до записи не больше 4. Так же введены вспомогательные ребра f_a^h, которые нужны, чтобы связывать ИГ до и после преобразования. Это может понадобиться, например, в следующем ситуации. Два автомата идут друг за другом на расстоянии 3 ребра. Автомат, который начал функционирование позже, переходит в вершину, которую автомат впереди разбил на две. Следовательно, автомат, который был сзади должен иметь ребра ко всем сыновьям разбитой вершины.

Итак, начнем описывать преобразования, используемые автоматами. Формальных преобразований будет большое количество, поэтому мы опишем только смысл данных преобразований, приводя примеры из них.

Рассмотрим первый блок преобразований. Преобразование, изображенное на рисунке 14, соответствует случаю пустой базы данных.

формальное преобразование пример преобразования ИГ

Рис. 14. Вставка для случая пустой базы данных.

На рисунке 14 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), $r_J^1(p_L^0) = p_J^1$, $r_J^2(p_L^0) = p_J^2$, $r_L(p_L^0) = p_L^0$, где $I(p_J^1) = f_y$, $I(p_J^2) = f_{=,y}$.

Так же буква «к», находящаяся рядом с вершиной, обозначает корень, а 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 ИГ.

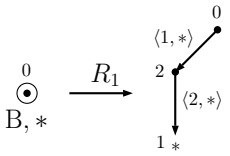

Рис. 15. Преобразование R_1 на вставку, с точки зрения автомата.

На рисунке 15, 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 два предикатных ребра. Одному ребру присвоен предикат первого типа $\langle 1, * \rangle$, который будем интерпретировать как f_*, второму ребру присвоен предикат второго типа $\langle 2, * \rangle$, 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 так же тип предикатов первого типа. 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.

В случае пустой базы данных для автоматов на поиск и удаление применяется преобразование R_2, изображенное на рисунке 16. В этом

случае автомат, обслуживающий запрос, завершает функционирование с сигналом $e = 2$ (запись не найдена).

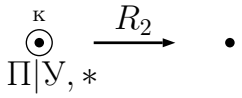

Рис. 16. Преобразование R_2.

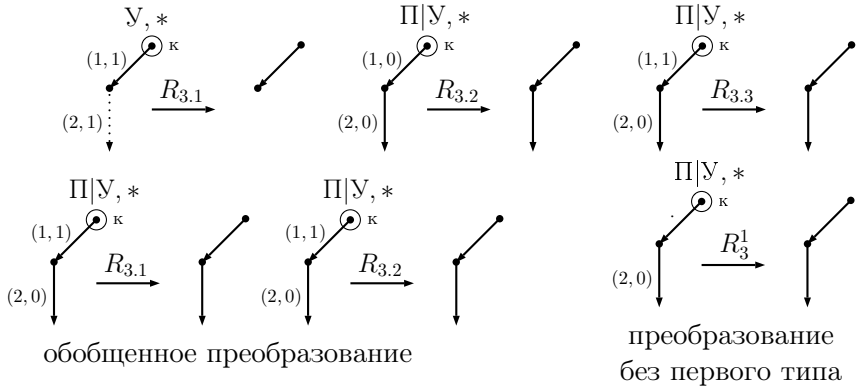

Рис. 17. Группа преобразований $R_3 = \{R_{3.1}, R_{3.2}, R_{3.3}\}$.

Если база данных состоит из одной записи, то применяются преобразования R_3, R_4, изображенные на рисунках 17 и 18. На этих рисунках ребрам приписаны код ребер на запросе, а именно первая координата соответствует типу предиката, а вторая — его значению на запросе. Например, ребро с кодом $(1, 0)$ означает, что в ИГ этому ребру соответствует предикат первого типа и его значение на запросе, который обслуживает автомат, равен 0. Преобразование $R_{3.1} \in R_3$ завершается с сигналом $e = 1$ (запись найдена). Преобразования $R_{3.2}, R_{3.3}$ завершаются с сигналом $e = 2$ (запись не найдена). В дальнейшем для ребер, которым соответствует предикат первого типа, будем применять сокращенную запись. А именно не будем писать тип предиката, а так же если его значение не важно 0 или 1, то будем писать точку на этом ребре. Пример сокращенных преобразований изображен на рисунке 17.

В преобразовании $R_{4.1} \in R_4$ есть ребро с кодом $(2,1)$ — это означает, что вставляемая запись уже существует и ее не нужно вставлять.

В преобразованиях $R_{4.1}, R_{4.2}$ вставляемая запись не совпадает с уже имеющейся, поэтому к базе данных добавляется новая запись соответствующая запросу. При этом в зависимости от значения предиката первого типа запись вставляется либо справа, либо слева от уже имеющейся.

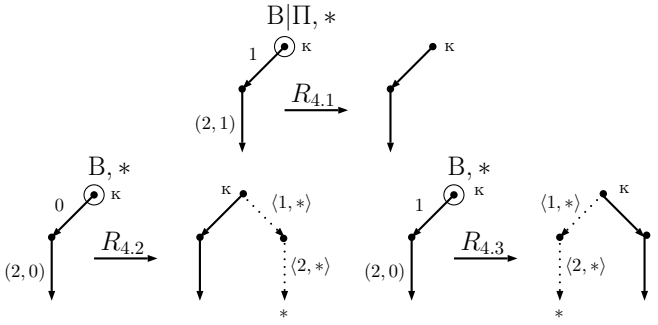

Рис. 18. Группа преобразований $R_4 = \{R_{4.1}, R_{4.2}, R_{4.3}\}$.

Для случая, когда в базе данных больше одной записи, рассмотрим по отдельности все преобразования на вставку, на удаление и на поиск.

Рассмотрим сначала все преобразования на вставку. Основная идея для вставки схематично изображена на рисунке 19. При этом заметим, что преобразование добавляет вспомогательные ребра третьего типа (f_a^h). Рассмотрим вспомогательное ребро и вершину из которой оно выходит. Оно соединяет ее с ребенком брата этой вершины. Нагрузка этого ребра является копией нагрузки ребра, которое идет от брата рассматриваемой вершины, до этого ребенка. Эти ребра нужны, чтобы не нарушить алгоритм поиска автомата, который идет вслед за рассматриваемым. Сразу заметим, что если автомат видит по движению вспомогательные ребра, то он обязательно их стирает в следующий такт. Так как вспомогательное ребро является копией ребра ИГ, то количество таких ребер не превосходит количество ребер в ИГ, то есть его объем. Кроме того, если автомат видит вспомогательное ребро перед собой, то он обязательно его стирает.

Следовательно, в любой такт времени количество вспомогательных ребер не превосходит объема ИГ.

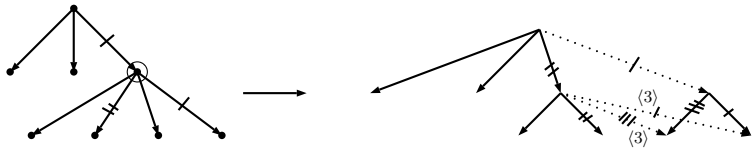

Рис. 19. Основная идея перестройки графа автоматом на вставку.

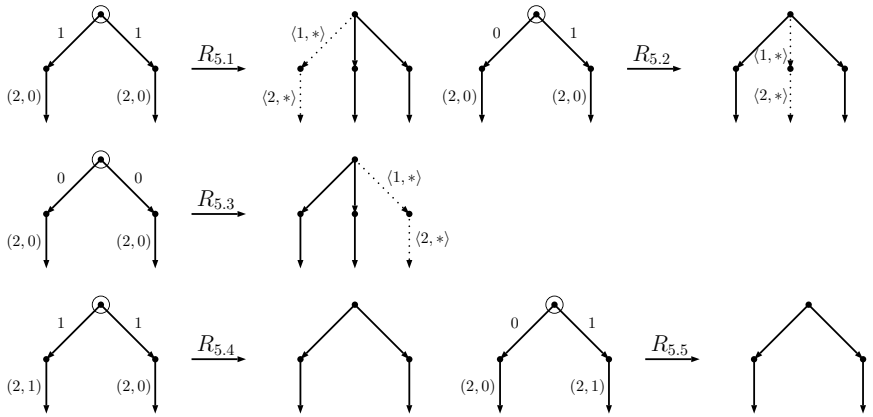

Рис. 20. Группа преобразований R_5.

Итак, сначала опишем преобразования на вставку, которые применяет автомат, находясь на расстоянии в два ребра от записей. Группа преобразований R_5, $R_5 = \{R_{5.1}, R_{5.2}, R_{5.3}, R_{5.4}, R_{5.5}\}$, соответствует случаю вставки третьей записи и изображена на рисунке 20. Видно, что эта группа преобразований схожа с уже разобранной группой R_4. Преобразования $R_{5.4}$ и $R_{5.5}$ — это случаи, когда вставляемая запись уже есть.

Группа преобразований R_6 аналогична группе преобразований R_5. В этой группе вставка добавляет четвертую запись, если ее не было. Пример преобразования из этой группы изображен на рис. 21.

Следующая группа преобразований R_7, образует новый корень (увеличивает высоту дерева). Условно можно сказать, что эта группа преобразований разбивает пятерку на сумму двойки и тройки.

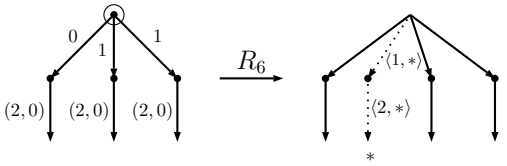

Рис. 21. Пример преобразования из группы R_6.

Преобразования из этой группы отличаются только тем, в какое по порядку место вставляется новая запись. Пример преобразования их этой группы изображен на рисунке 22.

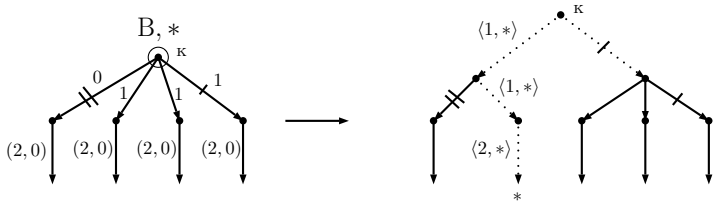

Рис. 22. Группа преобразований R_7.

Для дальнейшего описания преобразований на вставку введем несколько пояснений. Один шаг алгоритма на вставку заключается в прохождении одного уровня дерева. При этом он переходит по самому левому ребру, которому соответствует значение 1. Если таких ребер нет, то самым левым считаем самое правое ребро. Так же за один такт функционирования автомат делает три шага по дереву, если это возможно, или завершает функционирование с необходимым действием. Заметим, что автомат во время шага может перестроить ИГ при необходимости, применяя преобразование условно изображенное на рисунке 19. Основная цель этих преобразований состоит в том, чтобы автомат, оказавшийся в вершине, в которую нужно вставить запись (предпоследний уровень дерева), смог ее вставить и завершить функционирование.

Пример одного такта (трех шагов) автомата изображен на рисунках 23 и 24. На рисунке 23 изображен пример перехода автомат на 3 уровня вниз без применения перестроек. Автомат на вставку не применяет перестроек, так как он оказывается в вершине, у которой не

более двух братьев. Заметим, что при этом, если автомат вставляет запись так, что она больше максимального элемента в поддереве, то автомат меняет нагрузку самого правового ребра, заменяя ее на предикат $\langle 1, * \rangle$. Он делает это при необходимости на предикате, который в первый шаг стоит на уровень выше, и не делает этого преобразования на третьем шаге. Это необходимо, чтобы избежать конфликтов между автоматами.

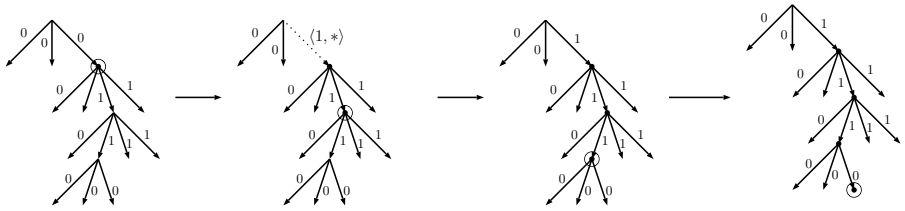

Рис. 23. Примера одного такта автомата на вставку, без применения перестроек.

На рисунке 24 изображен пример одного такта функционирования автомата на вставку с применением перестроек. Заметим, что после преобразования добавляются вспомогательные ребра третьего типа. Смысл данных ребер был описан выше, заметим только, что вспомогательные ребра находятся только на уровне, откуда автомат начал применять свое преобразование. На других уровнях вспомогательные ребра не обязательны, поэтому автомат их не добавляет. Действительно, минимальное расстояние от автомата на вставку, до следующего за ним автомата, не меньше трех ребер. Поэтому нужно добавить вспомогательные ребра, только на уровень, на котором находится рассматриваемый автомат, так как другие уровни не будут влиять на автоматы, которые начали свое функционирование после него.

Может возникнуть такая ситуация, что автомат на вставку окажется в вершине, у которой три брата. Это может произойти из-за того, что он применил преобразование без перестроек, в то время как автомат впереди мог применить перестройки на том уровне, в который перейдет рассматриваемый автомат. Для этого случая автомат, при необходимости, применяет преобразование, изображенное на рисунке 25. Рассмотрим вершину, в которой находится автомат. Необ-

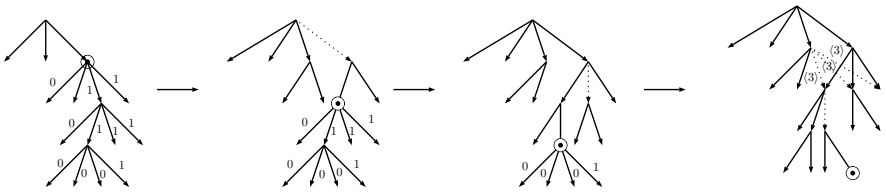

Рис. 24. Пример одного такта автомата на вставку, с применением перестроек.

ходимость в таком преобразовании возникает, если количество детей, у рассматриваемой вершины, равно 4. Заметим, что обязательно найдется брат у рассматриваемой вершины, у которого не более трех детей. Это не трудно заметить. Действительно, автомат мог оказаться в вершине, у которой три брата, только в том случае, если другой автомат, который идет впереди, применил преобразования перестройки. Видно, что после преобразования перестройки у родителя рассматриваемой вершины будут дети, у которых не более трех детей. На самом деле, у родителя рассматриваемой вершины, будут двое детей, у которых в сумме не более 5 детей.

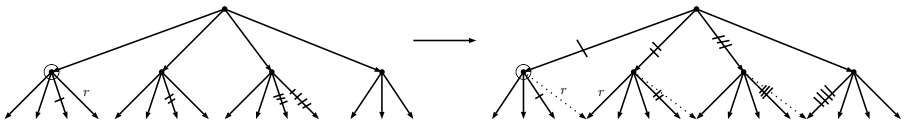

Рис. 25. Пример преобразования в случае трех братьев.

Разберем граничные случаи, когда автомат находиться на расстоянии 2,3 или 4 от записей. Автомат на вставку в этих случаях вставляет запись, если ее нет в базе данных, и завершает функционирования с нужным сигналом. Пример преобразования на вставку для расстояния 4 изображен на рисунке 26. По условию теоремы радиус видимости автомата 4, поэтому автомат сможет применить эти преобразования. Для оставшихся случаев — аналогично.

Заметим, что при таком алгоритме не может возникнуть ситуация, когда автомат оказывается на расстоянии 1 от записей, так как за один такт до этого расстояние до записей было равно 4, и он должен был завершить функционирование. Это свойство будет исполь-

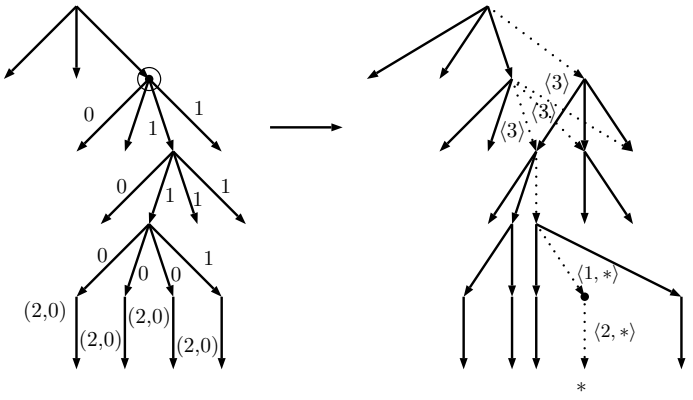

Рис. 26. Один такт автомата на вставку, когда расстояние до записей равно 4. Вставка записи и завершение функционирования.

зовано ниже для доказательства, что преобразования на вставку не вызывают конфликтов.

Итак, основные преобразования на вставку описаны. Дополнительные преобразования на вставку опишем ниже. Эти дополнительные преобразования будут введены для избежания конфликтов с преобразованиями на удаления.

Опишем теперь основные преобразования на удаления. Алгоритм автомата на удаление будет схож с алгоритмом автомата на вставку. За один такт автомат на удаление делает по возможности 3 шага, спускаясь на 3 уровня вниз. Автомат на удаление делает перестройки, чтобы выполнялось свойство братьев вершины, в которой он будет находиться. А именно, чтобы количество братьев было не меньше двух. Основная идея преобразования на удаление изображена на рисунке 27. На нем видно, что автомат, переходя на один уровень вниз, увеличивает количество братьев, объединяя сыновей соседних вершин. Если у брата рассматриваемой вершины есть три сына и более, то автомат перекидывает одного из них, делая его сыном вершины, в которой он стоит. Если же у брата вершины, в которой стоит автомат, тоже только два сына, то автомат объединяет этих двух братьев, делая из них новую вершину, у которой четыре ребенка. При этом оставляет вершину брата и добавляет вспомогательные ребра

третьего типа. Это необходимо делать, чтобы избежать некорректной работы автоматов, которые начали функционирование позже.

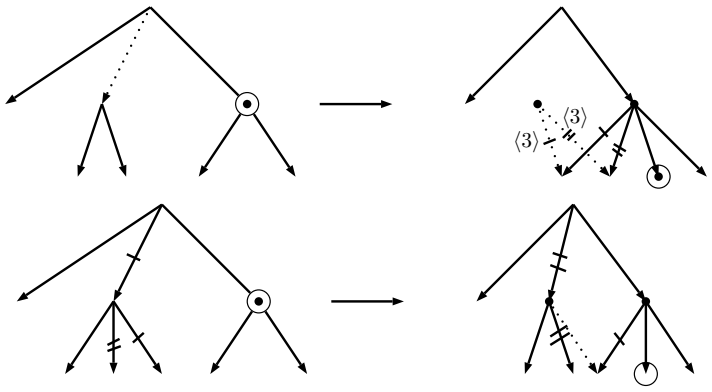

Рис. 27. Основная перестройка графа, автоматом на удаление.

Аналогично алгоритму работы автомата на вставку, автомат на удаление не применяет перестройки графа, если он после трех шагов переходит в вершину, у которой не менее двух братьев.

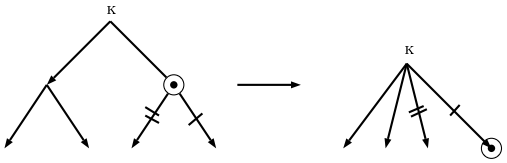

Рис. 28. Преобразование на удаление корня.

Возможна ситуация, когда автомат на удаление оказывается в вершине, у которой всего один брат. Если автомат находиться в корне, у которого два сына, то он применяет преобразование на удаление корня. В этом случае высота дерева уменьшается на один. Пример преобразования на удаление корня изображен на рисунке 28. Естественно, что вершину, являющуюся корнем, автомат не удаляет, так как туда может придти новый автомат. Это преобразование удаляет сыновей корня, а всех детей сыновей корня делает детьми корня.

Разберем другой возможный случай, когда автомат оказывается во внутренней вершине, у которой только один брат. Это могло

произойти только тогда, когда другой автомат на удаление применил преобразование графа на этой вершине. Действительно, если бы никто не применил преобразования на этой вершине, то она бы не изменилась, и автомат перешел бы в вершину, у которой не менее двух братьев, или он сам бы перестроил ее, если у нее был бы только один брат.

Допустим, что автомат оказался в вершине, у которой только один брат. В этом случае сумма детей у рассматриваемой вершины и ее брата не меньше 5. Это следует из того, что такт назад в этой вершине была перестройка другим автоматом на удаление, который не мог сделать сумму детей меньше, чем $2 + 3 = 5$. Следовательно, автомат может перераспределить детей так, чтобы в вершине, в которую он собирается перейти, было не менее трех детей. Тем самым граф уже на втором шаге автомата будет удовлетворять предположению, что автомат на удаление находится в вершине, у которой не меньше двух братьев. Пример преобразования перераспределения детей можно увидеть на нижнем преобразовании из рисунка 27, если считать, что из вершины выходит не 3 ребра, а только 2.

Как и в случае со вставкой, автомат на удаление, находясь на расстоянии 2, 3 или 4 ребра от записей, в следующий такт завершает функционировании с нужным сигналом. Так же автомат на удаление не может оказаться на расстоянии один от записей. Это свойство доказывается аналогично случаю вставки. Пример преобразования на удаление, для расстояния 4, изображен на рисунке 29. На этом рисунке нарисованы не все коды ребер. Так же, на нем условно обозначены вспомогательные ребра через $\langle 3 \rangle$. Смысл этих ребер был описан ранее. Напомним, что они служат для корректной работы других автоматов. В данном случае, эти ребра нужно добавлять на 2 яруса, так как удаление могло удалить корень, и следовательно, следующий за ним автомат мог попасть на один из этих уровней. Для оставшихся случаев — аналогично.

Итак, осталось рассмотреть случаи, когда автоматы выполняют разные задания и видят друг друга.

Рассмотрим всевозможные пары автоматов, которые видят друг друга. Заметим сначала, что стирания вспомогательных ребер, нагрузка которых есть предикаты третьего типа (f_a^h), не влияют на конфликтность. Поэтому автоматы на поиск не влияют на конфликтность преобразований. В дальнейшем не будем обращать внимания

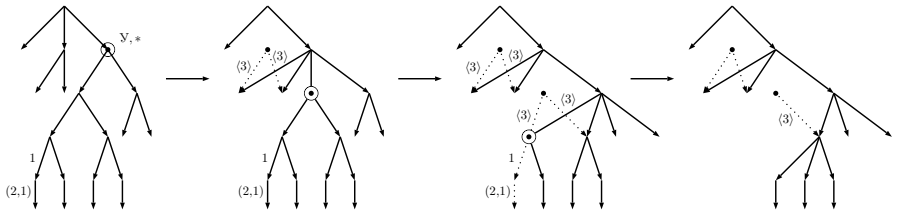

Рис. 29. Один такт автомата на удаление, когда расстояние до записей равно 4. Удаление записи и завершение функционирования.

на вспомогательные ребра, так как их удаление не будет влиять на конфликтность.

Прежде чем приступим к разбору случаев, оценим возможные расстояния между автоматами. Заметим, что автомат может изменить высоту дерева только в первый такт своего функционирования. Причем автомат на вставку может увеличивает высоту дерева ровно на 1, а автомат на удаление может уменьшить высоту дерева ровно на 1. Начиная со второго такта, все автоматы перемещаются либо на 3 уровня вниз, либо завершают функционирование, поэтому либо один из автоматов завершает функционирование, либо расстояние между ними сохраняется и равно либо 2 ребра, либо 3 ребра. Расстояние в 2 ребра может быть только между автоматом на удаление, который удалил корень, и следующим за ним автоматом. Если был удален корень, то расстояние между автоматом на удаление A_1 и следующим автоматом A_2 будет 2 ребра, которое не будет меняться. При этом, расстояние между автоматом A_2 и следующего за ним автоматом будет обязательно не меньше 3 ребер. Это было доказано выше, когда разбирался случай, что корень не может быть удален два такта подряд. Расстояние между любыми автоматами на вставку всегда не менее трех.

Итак, разберем первый случай. Два автомата на вставку видят друг друга. Как было показано выше, расстояние между ними не меньше трех ребер. Будем считать, что между ними ровно 3 ребра. Пусть это автоматы A_1 и A_2, причем автомат A_1 начал свое функционирование на такт раньше.

Рассмотрим первый случай. Если автомат A_2 не идет в вершину, в которой стоит автомат A_1 или к ее братьям, то конфликта не воз-

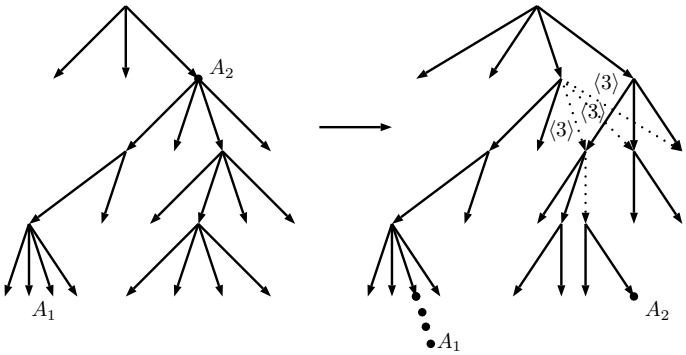

Рис. 30. Первый пример не конфликтности двух автоматов на вставку.

никнет. Докажем это. Автомат A_1 преобразует только те вершины, которые находятся на его уровне и, может быть, преобразует еще и родительскую вершину. Автомат A_2 не преобразует эти вершины, так как идет в другие. Поэтому в этой ситуации конфликтов не возникнет. Пример этого случая изображен на рисунке 30.

Пусть автомат A_2 идет в вершину, где стоит автомат A_1, или идет к ее братьям. Пусть при этом автомат A_1 находится в вершине, у которой не больше двух братьев. Тогда автомат A_2 не применяет преобразования на перестроение и не меняет значение предикатов на третьем шаге. Тем самым конфликтов не возникает. Пусть автомат A_1 находится в вершине, у которой ровно 3 брата. Как было разобрано выше, автомат A_1 применит преобразование для случая трех братьев, изображенное на рисунке 25. Это преобразование затрагивает только вершину, в которой стоит A_1, и ее братьев, и, возможно, изменяет значение предикатов у ребер исходящих из родителя. Таким образом, автомат A_2 применит преобразования с перестройкой, которое не будет конфликтовать с преобразованием A_1, так как на последнем третьем шаге, автомат A_2 только перераспределяет ребра, но не меняет значения на них. Пример этого случая изображен на рисунке 31.

Аналогичные случаи, когда автомат A_1 находиться рядом с записями, на расстоянии не более, чем 4 ребра, уже разобранным. Заметим, только что автомат не может находиться на расстоянии 1 от

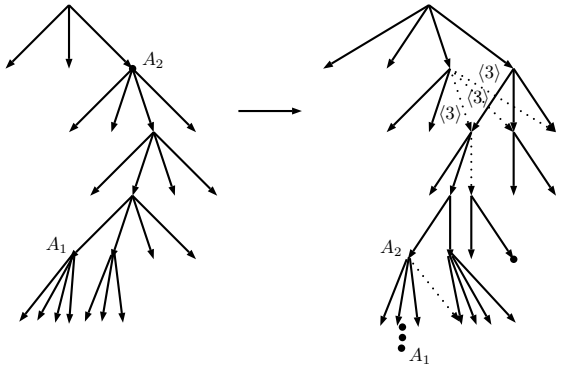

Рис. 31. Второй пример не конфликтности двух автоматов на вставку.

записей. Это было показано выше. Для большей ясности, что конфликтов не возникнет, приведен пример на рисунке 32.

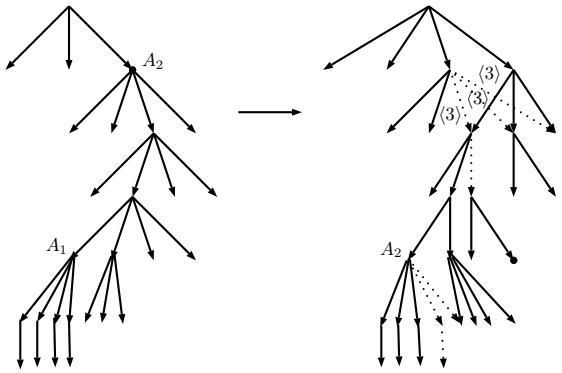

Рис. 32. Третий пример неконфликтности двух автоматов на вставку. Автомат A_1 вставил запись и завершил функционирование.

Итак, мы показали , что автоматы на вставку не вызывают конфликтов. Теперь разберем неконфликтность двух удалений подряд. Первый случай, расстояние между автоматами на удаление 3 ребра.

Если расстояние между автоматами на удаление равно 3, то неконфликтность двух преобразований на удаление доказывается схоже с доказательством неконфликтности преобразований на встав-

ку. Действительно, пусть даны два автомата на удаление A_1 и A_2, и автомат A_1 начал свое функционирование на 1 такт раньше A_2.

Для простоты объяснения, будем считать, что автомат $A_1(A_2)$ находится в вершине $v_1(v_2)$ на первом шаге своего такта.

Первый случай, когда автомат A_2 переходит в вершину, которая не совпадает с v_1 и не является ее братом. В этом случае автоматы не конфликтуют. Пример, иллюстрирующий этот случай, изображен на рисунке 33. На нем изображены два овала. Овал, который находиться ниже уровнем, чем вершина v_1, в которой находиться A_1, включает вершины и ребра, которые возможно будет преобразовывать A_1 на пером шаге функционирования. Другой овал показывает, какие вершины и ребра будет затрагивать автомат A_2 на третьем шаге своего такта функционирования. Видно, что эти преобразования не конфликтуют.

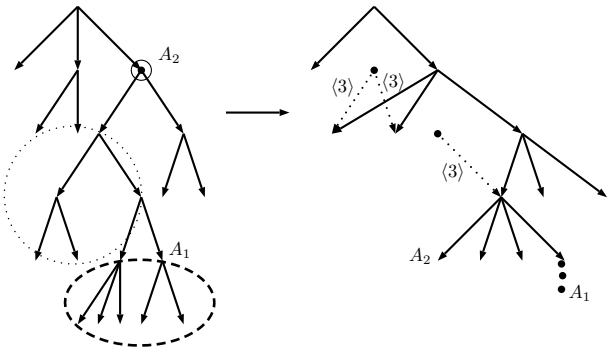

Рис. 33. Первый пример неконфликтности двух автоматов на удаление.

Второй случай, когда автомат A_2 переходит в вершину v_1, в которой находится A_1, или к ее братьям. При этом либо вершина v_1, либо существует ее брат, который имеет не менее трех потомков. Доказательство этого случая также схоже с уже разобранным случаям для вставок. Если автомат A_1 находится в вершине, у которой как минимум 2 брата, то автомат A_2 не применяет преобразований, и, следовательно, конфликтов не возникает. Если же автомат A_1 находится в вершине, у которой ровно один брат, то он обязательно применит преобразование перераспределения детей, изображенное на рисунке 27. Чтобы не загромождать доказательство рисунками, пример, иллю-

стрирующий этот случай, можно увидеть на рисунке 33. Достаточно только заменить положение автомата A_2, на положение, относящееся к этому случаю.

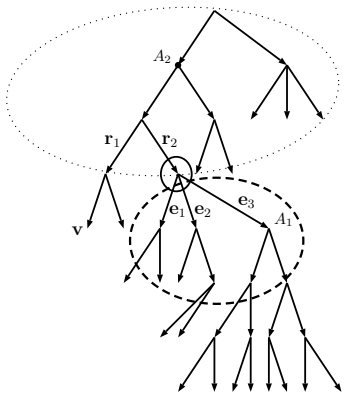

Рис. 34. Конфигурация при которой возможен конфликт, при использовании стандартного алгоритма. Автомат A_2 должен перейти в вершину $\mathbf{v}$.

Третий случай, если автомат A_1 находится в вершине v_1 у которой два(три) брата, но суммарное количество сыновей и племянников, у этой вершины, равно 4(6). В этом случае автоматы A_1 и A_2 уже не могут действовать по стандартному алгоритму, так как может возникнуть конфликт в вершине, которая является родителем v_1. Пример этого случая изображен на рисунке 34. Автомат A_2 переходит в вершину v, и он будет вынужден совместить ребра $\mathbf{r}_1$ и $\mathbf{r}_2$. Автомат A_1 совмещает ребра $\mathbf{e}_2$ и $\mathbf{e}_3$. Конфликт возникает в вершине, которая является родителем v_1(обведена овалом, граница которого непрерывна).

Прежде чем начать разбирать алгоритм действия автоматов в этой конфигурации, рассмотрим причину возникновения конфликта и разберем конфигурации, при которых автоматы будут действовать стандартно.

Если у вершины v два и более братьев или входная окрестность автомата A_1 такая же, как и на рисунке 35, то автоматы действуют стандартно. Случай, когда у вершины v два и более братьев очевиден, так как автомат A_2 не делает перестроек графа. Случай, изображен-

ный на рисунке 35, означает, что автомат A_2 совместит ребра $\mathbf{r}_1$ и $\mathbf{r}_2$, но это не вызовет конфликта, так как при этом не будет задействован родитель вершины v_1.

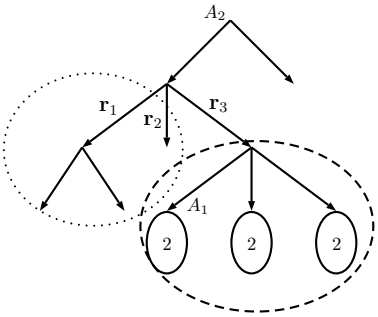

Рис. 35. Пример, когда автоматы на удаления A_1 и A_2 действуют стандартно. Число в овале означает количество ребер.

Итак, разберем алгоритм автоматов для случая, изображенного на рисунке 34. Чтобы избежать конфликтов, автомат A_1 поменяет первый шаг своего такта, а автомат A_2 поменяет третий шаг своего такта. Заметим, что другие шаги автоматов не могут создать конфликтов.

Рассмотрим конфигурацию, когда автомат A_2 готов сделать третий шаг, а автомат A_1 делает первый шаг. Пример изображен на рисунке 36. На этом рисунке изображено три преобразования. Первое преобразование, обозначенное стрелкой A_1, соответствует первому шагу автомата A_1. Он соединяет ребра $\mathbf{e}_1$ и $\mathbf{e}_2$, делая из них $\mathbf{e}_1^2$, а так же помечает вместо ребра e_2, делает вспомогательное ребро. Это ребро нужно оставить, чтобы не взывать конфликта, в этом и заключается отличие первого шага автомата A_1. Кроме того, автомат A_1 обязательной делает в первый шаг перестройку, даже если это не нужно было в стандартном алгоритме. Например, A_1 переходит в вершину, у которой не меньше двух братьев. В этом преобразовании, были использованы параметры m_1 и m_2. Они будут нужны для избежания конфликтов, когда расстояниями между автоматами будет равно двум, а не трем. Ниже будет пояснение.

Второе преобразование, обозначенное стрелкой A_2, соответствует третьему шагу автомата A_2. Он соединяет ребра $\mathbf{r}_1$ и $\mathbf{r}_2$, делая из

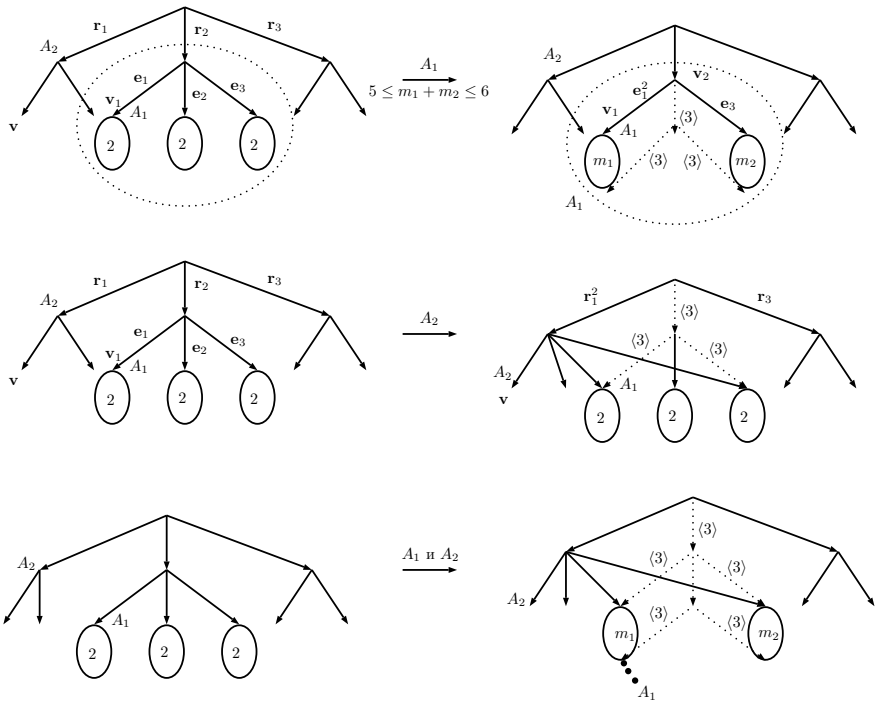

Рис. 36. Преобразование автомата A_1 и A_2, отличающиеся от стандартных, для избежания конфликта.

них $\mathbf{r}_1^2$. Важно, что он помечает ребра $\mathbf{e}_1$ и $\mathbf{e}_3$, как вспомогательные, чтобы следующий автомат их мог удалить. Тем самым, показано, что конфликтов не возникает и в специальных случаях.

Осталось рассмотреть случай неконфликтности двух преобразований на удаление вблизи записей. Этот случай является частным случаем уже разобранных выше, так как минимальное расстояние до записей равно 2. Это было доказано выше, когда разбирались алгоритм удаления вблизи записей. Пример, иллюстрирующий этот случай, изображен на рисунке 37.

Рассмотрим теперь случай, когда расстояние между двумя запросами на удаление равно двум. В такой конфигурации автомат A_2 после одного такта окажется на один уровень ниже, чем тот уровень, на котором находиться автомат A_1. Для избежания конфликтов в этих случаях, алгоритм перестройки автоматов будет немного отличаться

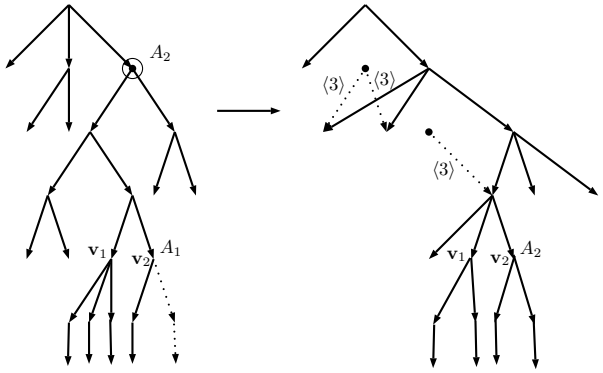

Рис. 37. Пример неконфликтности двух автоматов на удаление. Случай вблизи листьев. Автомат A_1 удалил запись и завершил функционирование.

от уже введенных преобразований. При этом данные алгоритмы будут применяться исключительно в этих случаях, так как иначе перестроенный граф не будет 2-3-4 деревом. Пусть автомат A_1 находиться в вершине v_1, а автомат A_2 находиться в вершине v_2. Дополнительные преобразования будут влиять только на уровни между вершиной v_2 и вершиной v_1, на других уровнях автоматы будут применять уже введенные ранее преобразования. Как уже было доказано выше, двух подряд удалений корня не может быть. Рассмотрим последовательность автоматов A_0, A_1, A_2, A_3. Если расстояние между A_1 и A_2 равно двум, то расстояние между A_0 и A_1, A_2 и A_3 не меньше трех. Поэтому, если мы покажем, что между автоматами A_1 и A_2 конфликтов не возникает, то конфликтов не возникнет для любого количества подряд идущих автоматов на удаление. Докажем это.

Опишем отдельно преобразования для автомата A_1 и A_2. После этого покажем, что конфликтов не возникнет, а перестроенный граф останется 2-3-4 деревом. Сначала опишем преобразования автомата A_1.

Пусть автомат A_1 находиться в вершине v_1. Рассмотрим первый случай, когда у v_1 только один брат, или два брата, но либо вершина v_1 имеет трое потомков, либо среди братьев v_1 есть вершина, у которой трое детей.

Преобразование на первом шаге автомата A_1 для этого подслучая изображено на рисунке 38. Оно отличается от стандартного тем, что автомат делает суммарное количество внуков не меньше 5, и суммарное количество внуков брата не меньше 5. При этом автомат A_2 не будет делать преобразований, если он идет в вершину, которая является сыном или племянником вершины v_1. При этом автомат A_2 окажется в хорошей для себя конфигурации. Под хорошей конфигурацией понимается, что если автомат A_2 оказывается в вершине, у которой только 1 брат, то количество детей и племянников этой вершины должно быть не меньше 5. Это возможно из-за того, что $m_1 + m_2 \geqslant 5$ и $m_3 + m_4 \geqslant 5$. Если же автомат A_2 идет в другие вершины, то конфликтов опять не возникает, так как автомат A_1 не преобразует родителя вершины v_1.

Нужно показать, что обязательно найдутся такие m_1, m_2, m_3, m_4, что $m_1 + m_2 \geqslant 5$, $m_3 + m_4 \geqslant 5$, $m_1 + \cdots + m_4 = n_1 + \cdots + n_5$ или $m_1 + \cdots + m_4 = n_1 + \cdots + n_5 - 1$. Поясним, откуда может взяться минус один.

Алгоритм автомата на удаление устроен таким образом, что на одном шаге он может совместить только одну пару ребер в одно ребро. Следовательно, автомат A_1 может уменьшить сумму на один. Таким образом, если $n_1 + \cdots + n_5 \geqslant 11$, то для своих целей автомат A_1 может уменьшить это количество только на 1. Поэтому после того как автомат A_1 перестроит граф для себя, он сможет перестроить граф и для A_2, так как $m_1 + \cdots + m_4 \geqslant 10$. Отметим, что для этого преобразования необходимо, что радиус видимости 4. Иначе автомат A_1 не смог бы увидеть внуков брата.

Докажем, что $n_1 + \cdots + n_5 \geqslant 11$. Для доказательства этого факта нужно посмотреть на конфигурацию за 1 такт до этого. Автомат A_1 оказался в вершине, у которой всего 1 брат, потому что за такт до этого в этой вершине был другой автомат A_0, и он применил перестройку в этой вершине. Для более понятного объяснения представлено вспомогательное изображение на рисунке 39.

На этом рисунке изображена одна из возможных конфигураций. В частности, видно, что автомат A_0 объединил вершины k_3 и k_4, после чего мог сократить их количество на 1. Сократить количество детей у вершины более чем на 1 автомат на удаление не может, так как использует определенные преобразования, описанные выше, которые этого не делают. Из примера видно, что в момент, когда автомат A_1

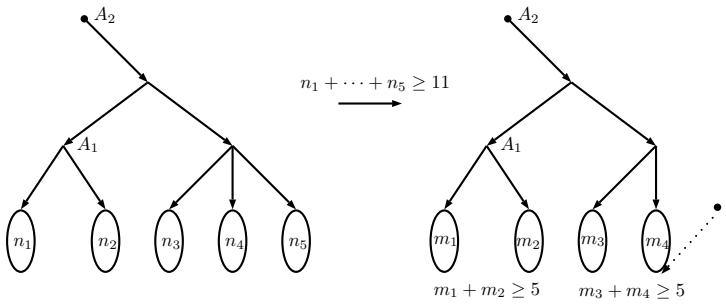

Рис. 38. Дополнительная перестройка автоматом A_1, в случае расстояния 2 до другого автомата на удаление.

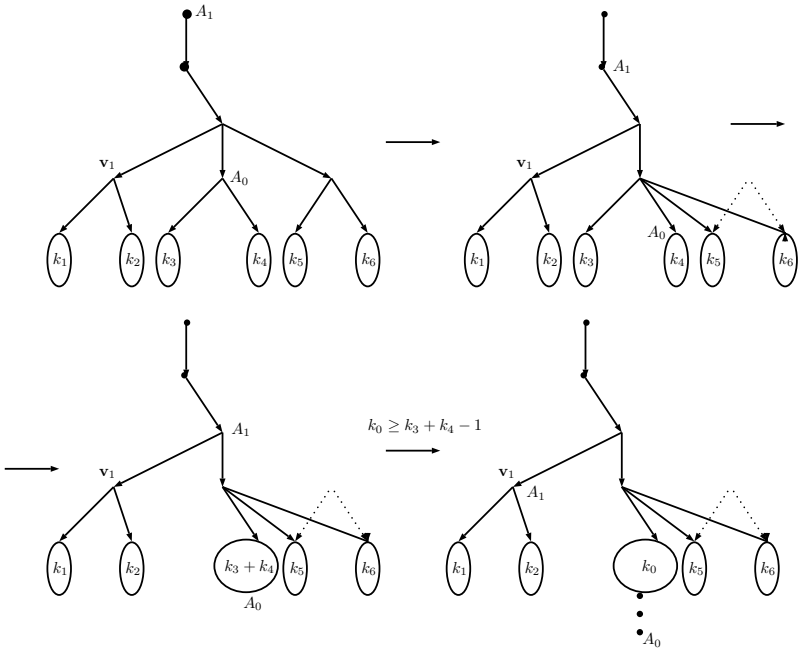

Рис. 39. Вспомогательное изображение. За такт до того, как автомат A_1 оказался в вершине v_1.

попадет в вершину v_1, суммарного количество внуков и детей племянников было $k_1 + k_2 + k_0 + k_5 + k_6 \geqslant k_1 + \cdots + k_6 - 1 = n_1 + \cdots + n_5 \geqslant 12 - 1 = 11$. Утверждение доказано.

Итак, мы разобрали преобразования автоматов A_1 и A_2 для случая, когда у вершины v_1 только 1 брат, или есть сын или племянник, у которого трое детей. Остается разобрать случай, когда у вершины v_1 более одного брата, но у всех детей и племянником по двое сыновей.

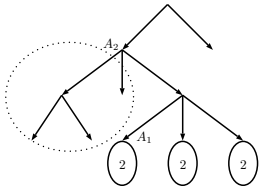

Рис. 40.

Если входная окрестность автомата A_1 такая же, как и на рисунке 40, то автомат A_1 делает стандартные преобразования и в конце, при необходимости, делает дополнительные преобразования, как для случая, когда у вершины v_1 был один брат. Алгоритм A_2 остается таким же, как был описан выше. А именно, если он переходит в вершину, которая является сыном или племянником вершины v_1, то автомат A_2 не делает преобразований. Иначе делает стандартные преобразования, так как конфликтов при этом не возникнет. На рисунке 40, пунктирным овалом выделены вершины, которые при преобразованиях будет задействовать A_2. Видно, что при этом конфликтов с A_1 не возникнет.

Итак, осталось рассмотреть последний вариант, когда отец вершины v_1 является средним братом или правым, но когда количество детей у вершины v_2 равно двум. Идея избежания конфликтов совпадает с алгоритмом, изображенным на рисунке 36. Как упоминалось ранее, этот алгоритм содержит переменные m_1 и m_2, $m_1 + m_2 \geqslant 5$. Автомат A_1 на первом шаге делает $m_1 = m_2 = 3$, но на втором шаге может склеить пару ребер, и сделать например $m_1 = 2$. Автомат A_2 может не делать преобразования, если идет в вершину, которая является сыном или племянником v_1. Сделаем пояснение. Пусть автомат A_2 оказывается в овале m_1 и $m_1 \geqslant 3$, тогда эта конфигурация хорошая для A_2, так как количество братьев не меньше двух. Если же m_1 равно двум, то это означает, что автомат A_1 склеил два ребра на втором шаге своего функционирования. Следовательно, количе-

ство детей у вершины, в которой будет находиться A_2, не меньше 5, поэтому конфигурация и в этом случае хорошая для A_2.

Осталось рассмотреть граничные случаи. Единственный случай, который еще не разобран, когда автомат A_1 находиться на расстоянии 2 от записей, при этом автомат A_2 находиться на расстоянии 4 от записей. Преобразования на удаление устроено так, что автомат A_2 должен применить свое преобразование и завершить функционирование. Поэтому автомат A_1, чтобы не вызвать конфликт, помечает свою запись как удаленную, меняя предикат на ней на вспомогательный (f^h). При этом он не завершает функционировании. На следующий такт автомат A_1 смотрит, есть ли на расстоянии 4 от записей новый автомат. Если другой автомат есть, то автомат A_1 завершает функционировании, в силу того, что другой автомат, которого он видит, сотрет вспомогательные ребра. Если другого автомата нет, то A_1 удаляет запись. Заметим, что автоматы A_1 и A_2 обслуживают разные запросы. Это было разобрано выше. Поэтому они не смогут одновременно поменять информацию на одном и том же ребре.

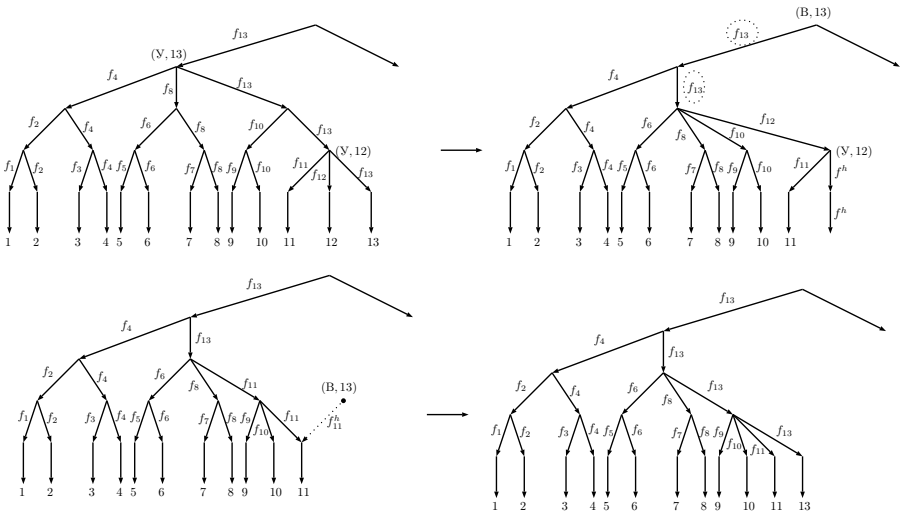

Рис. 41. Пример граничного случая.

Пример, иллюстрирующий этот случай, изображен на рисунке 41. Заметим, что автомат на удаление 13 (A_2) делает перестройку, так как видит, что автомат на удаление 12 (A_1) или какой-то другой ав-

томат должен будет удалить 12 на следующий такт. Таким образом, конфликтов не возникает. Также обратим внимание, что после удаления 13 выделенные предикаты на ребрах f_{13} не заменяются на f_{12}. Заметим, что при этом корректность работы не нарушится. Благодаря этому наблюдению автомат на удаление может удалить запись в один проход. Для наглядности в пример добавлен автомат на вставку 13. Расстояние до записей у него 5, поэтому автомат на удаление 12 сам удалит вспомогательные ребра. Заметим, что автомат на вставку не применяет перестройку графа на третьем шаге. Это будет разобрано ниже, когда будут исключаться конфликты между автоматами на вставку и автоматами на удаление. Если бы расстояние до записей у автомата на вставку 13 было бы 4, то автомат на удаление 12 завершил бы функционирование, а автомат на вставку 13 удалил бы сам вспомогательные ребра.

Итак, доказано, что автоматы на удаление не вызывают конфликтов. Осталось доказать, что автоматы на вставку и автоматы на удаление не конфликтуют.

Рассмотрим автомат $A_{\text{В}}$ на вставку, а автомат $A_{\text{У}}$ на удаление. Причем автомат $A_{\text{В}}$ начал функционирование раньше. Расстояние между ними не меньше трех ребер. Пусть автомат $A_{\text{В}}$ стоит в вершине v_1. Если у этой вершины братьев не меньше 2, то автомат $A_{\text{В}}$ не уменьшит это количество, поэтому автомату $A_{\text{У}}$ не нужно перестраивать граф, если он идет в вершину v_1 или к ее братьям. Если же $A_{\text{У}}$ идет в другие вершины, то он может делать необходимые перестроения, и это не вызовет конфликта. Если же у v_1 только один брат, то автомат $A_{\text{У}}$ должен посмотреть на количество детей у вершины v_1. Если их количество не превосходит 3, то он может перестраивать граф. Если же их количество равно 4, то $A_{\text{У}}$ не нужно перестраивать граф, если он идет в вершину v_1 или к ее брату, так как автомат на вставку применит перестройку, и у вершину v_1 станет 2 брата. При этом автомат на вставку видя, что за ним идет автомат на удаление должен применить перестройку, даже в том случае, если ему это было не нужно. На рисунке 42(а) изображен пример конфигурации, когда действие алгоритма на удаление отличается от стандартного из-за автомата на вставку.

Рассмотрим теперь случай, когда автомат $A_{\text{У}}$ начал функционирование раньше, причем расстояние между ним и автоматом $A_{\text{В}}$ не меньше трех ребер. Доказывается симметрично предыдущему слу-

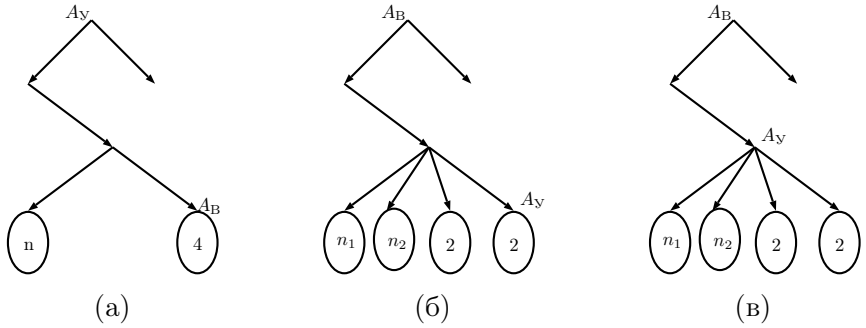

Рис. 42. Пример, когда действие алгоритма на удаление и вставку отличается от стандартных из-за их близости.

чаю. Единственное исключение изображено на рисунке 42(б). Автомат на вставку видит, что автомат на удаление применит перестроение, и не делает перестроение, которое мог бы сделать.

Рассмотрим оставшийся случай, когда расстояние между автоматом на удаление $A_У$ и автоматом на вставку $A_В$ равно двум. Это может произойти только тогда, когда автомат на удаление $A_У$ начал функционирование раньше. Этот случай похож на предыдущий. Так же конфликтов можно избежать. Пример исключения изображен на рисунке 42(в).

Единственный случай, который еще не разобран, когда автомат $A_У$ находиться на расстоянии 2 от записей, а автомат на вставку находиться на расстоянии 4 от записей. В этом случае, как и в разобранном ранее с двумя автоматами на удаление, автомат $A_У$ не удаляет свою запись, а меняет предикаты на вспомогательные. После этого он на следующий такт смотрит, есть ли в области видимости другой автомат. Если нет, то сам удаляет запись, иначе завершает функционирование. Пример этого случая изображен на рисунке 43. Заметим, что вставка делает перестройку, и это не вызывает конфликт с удалением, так как удаление меняет нагрузки ребер, которые вставка не изменяет. Так же автомат на удаление оказывается в вершине, в поддереве которого нет предикатов f^h. Но он все равно может удалить эти вспомогательные ребра, так как радиус видимости у него 4 ребра.

В качестве следствия вышеперечисленного справедливо следующее утверждение.

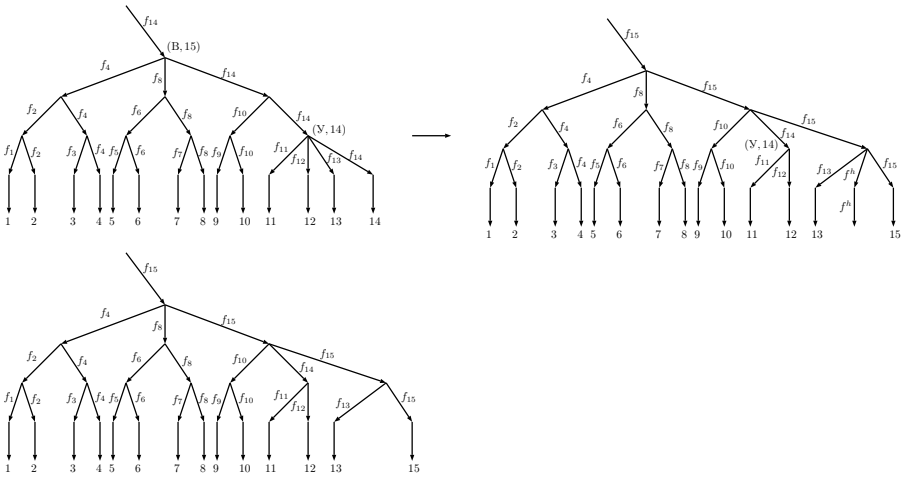

Рис. 43. Граничный случай, удаление и вставка.

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

Докажем еще одно утверждение о корректности работы МДИГ и о его сложности.

Лемма 2. *Для любого потока запросов $H, H \in \mathcal{H}$ МДИГ $\mathcal{D}$ функционирует корректно, кроме того, для любого натурального i количество тактов, необходимого на выдачу ответа на поисковый запрос $H(i)$, не превосходит*

$$\left]\frac{\log_2(\max\{|V(i,H)|,2\})}{3}\right[.$$

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 еще не завершившее 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 меньше, чем за такт до того, как запрос на поиск их достигнет. Преобразования на изменение ИГ, устроены так, что после каждого такта полученный ИГ является 2-3-4 деревом, не учитывая вспомогательные ребра. Причем, если нагрузка ребра есть предикат f_a, то это означает, что в этом поддереве находятся записи, которые не превосходят a. Кроме того, к каждой записи y ведет ребро с нагрузкой $f_{=,y}$. Следовательно, поиск найдет искомую запись, если она есть в базе данных, и не найдет, если ее нет. Корректность доказана.

Оценим теперь сложность МДИГ $\mathcal{D}$. Заметим, что автомат может изменить высоту ИГ только один раз, причем ровно на 1 и только в первый такт своего функционирования. Это следует из алгоритма автомата. Все преобразования не изменяют высоту ИГ, кроме преобразований в корне. Преобразование на вставку может увеличить на 1 высоту ИГ, а преобразование на удаление может уменьшить на 1 высоту ИГ.

Рассмотрим произвольный поток запросов H. Пусть запрос на поиск пришел в такт i. База данных в этот такт есть $V(i,H)$. Следовательно, высота ИГ не превосходит $]\log_2(\max\{|V(i,H)|,2\})[$. Автомат на поиск каждый такт проходит по 3 уровня вниз или завершает функционирование. Следовательно, он затратит на поиск не больше, чем $\left]\frac{\log_2(\max\{|V(i,H)|,2\})}{3}\right[$ тактов, так как подграфы ИГ не изменят свою высоту. Лемма доказана.

Утверждение теоремы является следствием лемм 1 и 2.

Список литературы

[1] Плетнев А. А. Динамическая база данных, допускающая параллельную обработку произвольных потоков запросов // Интеллек-

туальные системы. Теория и приложения — 2015. Т. 19, вып. 1. — С. 117–142.

[2] Cormen T. H., Leiserson Ch. E., Rivest R. L., Stein C. Introduction to Algorithms (3rd ed.). — MIT Press and McGraw-Hill, 2009 [1990].

[3] Knuth D. E. Sorting and Searching. Vol. 3 of The Art of Computer Programming. — Addison-Wesley, 1973.

[4] Aho A. V., Hopcroft J. E., Ulman J. E. The Design and Analysis of Computer Algorithms. — Addison-Wesley, 1974.

[5] Sedgewick R. Implementing Quicksort Programs // Communications of the ACM. — 1978. 21 (10). — P. 847–857.

[6] Comer D. The Ubiquitous B-tree // ACM Computing Surveys. — 1979. 11 (2). — P. 121–137.

[7] Guibas L. J., Sedgewick R. A Dichromatic Framework for Balanced Trees // Proceedings of the 19th Annual Symposium on Foundations of Computer Science. — IEEE Computer Society, 1978. — P. 8–21.

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

[9] 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.

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

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

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

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

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