Параллельные вычисления на динамически меняющемся графе

Игорь Бурдонов <igor@ispras.ru>
Александр Косачев <kos@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Рассматривается задача параллельного вычисления значения функции от мультимножества значений, записанных в вершинах ориентированного сильно-связного графа. Вычисление выполняется автоматами, находящимися в вершинах графа. Автомат имеет локальную информацию о графе: он «знает» только о дугах, выходящих из вершины, в которой он находится, но «не знает», куда (в какие вершины) эти дуги ведут. Автоматы обмениваются сообщениями, передаваемыми по дугам графа, которые играют роль каналов передачи сообщений. Вычисление инициируется сообщением, приходящим извне в автомат выделенной начальной вершины графа. Этот же автомат в конце работы посылает вовне вычисленное значение функции. Для решения этой задачи предлагаются два алгоритма. Первый алгоритм выполняет исследование графа, целью которого является разметка графа с помощью изменения состояний автоматов в вершинах. Такая разметка используется вторым алгоритмом, который и производит вычисление значения той или иной функции. Это вычисление основано на алгоритме пульсации: сначала от автомата начальной вершины по всему графу распространяются *сообщения-вопросы*, которые должны достигнуть каждой вершины, а затем от каждой вершины «в обратную сторону» к начальной вершине двигаются *сообщения-ответы*. Алгоритм пульсации, по сути, вычисляет агрегатные функции, для которых значение функции от объединения мультимножеств вычисляется по значениям функции от этих мультимножеств. Однако показано, что любая функция $F(x)$ имеет агрегатное расширение, то есть может быть вычислена как $H(G(x))$, где G агрегатная функция. Заметим, что разметка графа не зависит от той функции, которая будет вычисляться. Это означает, что разметка графа выполняется один раз, после чего может многократно использоваться для вычисления различных функций. Поскольку автоматы в вершинах графа работают параллельно, как разметка графа, так и вычисление функции выполняются параллельно. Это первая особенность работы. Вторая особенность – вычисления выполняются на динамически меняющемся графе: его дуги могут исчезать, появляться или менять свои конечные вершины. На изменения графа налагаются такие минимальные ограничения, которые позволяют решать эту задачу за ограниченное время. Приводится оценка времени работы обоих предлагаемых алгоритмов.

Ключевые слова: ориентированные графы, исследование графа, взаимодействующие автоматы, параллельная работа, агрегатные функции, динамически меняющиеся графы.

DOI: 10.15514/ISPRAS-2015-27(2)-12

Для цитирования: Бурдонов Игорь, Косачев Александр. Параллельные вычисления на динамически меняющемся графе. Труды ИСП РАН, том 27, вып. 2, 2015 г., стр. 189-220. DOI: 10.15514/ISPRAS-2015-27(2)-12.

1. Введение

Данная работа находится на пересечении двух направлений исследования, результаты которых опубликованы в статьях [1] и [2].

Первое направление [1] – это параллельные вычисления на ориентированном графе. Такие вычисления являются, по сути, исследованием графа, «нагруженного» дополнительными значениями в вершинах графа.

Исследование ориентированных графов – корневая задача во многих приложениях. Достаточно указать исследование сетей связи, в том числе сети интернета и GRID, и тестирование программных и аппаратных систем, моделируемых графами переходов. Исследование графа, как правило, базируется на его обходе, а это уже старая классическая задача обхода лабиринта. Эта задача нетривиальна, если граф ориентирован, то есть в лабиринте «улицы с односторонним движением».

Обход ориентированного сильно-связного графа требует времени порядка nm, где n – число вершин графа, а m – число дуг. Такое время обхода достигается многими хорошо известными алгоритмами: обход в глубину, обход в ширину, «жадный» алгоритм и др. [3,4,5].

В 1966 г. М.О. Рабин поставил задачу обхода ориентированного графа конечным автоматом [6]. Автомат на графе аналогичен машине Тьюринга: ячейке ленты соответствует вершина графа, а движение влево или вправо по ленте заменяется переходом по одной из дуг, выходящих из текущей вершины графа. На сегодняшний день наиболее быстрый алгоритм предложен в [7], он имеет оценку $nm+n^2 loglogn$. При повторном обходе, когда автомат может использовать пометки, оставленные им же после первого обхода, оценка уменьшается до $nm+n^2 l(n)$, где $l(n)$ — число логарифмирований, при котором достигается соотношение $1 \le log(log...(n)...) < 2$ [8]. Отличие от нижней оценки nm объясняется тем, что автомату бывает нужно «вернуться» в начало только что пройденной дуги.

За последние годы размер реально используемых систем и сетей и, следовательно, размер исследуемых графов непрерывно растёт. Проблемы возникают тогда, когда исследование графа одним автоматом (компьютером) либо требует недопустимо большого времени, либо граф не помещается в памяти одного компьютера, либо и то, и другое. Поэтому возникает задача

параллельного и распределённого исследования графов. Эта задача формализуется как задача исследования графа коллективом автоматов.

В [9] и [10] предложены алгоритмы работы такого коллектива автоматов. При этом предполагается, что автоматы не могут ничего писать в вершины графа или читать из них, но могут обмениваться между собой сообщениями с помощью сети связи, ортогональной графу, а также генерировать новые автоматы. Наилучшая полученная оценка $m+nD$, где D – диаметр графа, т.е. длина максимального пути (маршрута без самопересечений) в графе.

В [1] мы рассматривали классическую задачу исследования графа автоматами, обмен информацией между которыми происходит только через память вершин графа. Это эквивалентно исследованию графа с помощью сообщений, которыми обмениваются между собой автоматы, неподвижно «сидящие» в вершинах графа, а дуги графа играют роль каналов передачи сообщений. Автомат, находящийся в вершине, посылает сообщение по одной из дуг, выходящих из этой вершины, и через какое-то время такое сообщение принимается автоматом в конце дуги. Оценка времени работы алгоритма зависит от числа сообщений, которые могут одновременно передаваться по дуге. Такое число называется ёмкостью дуги и обозначается k. Предполагается, что исследуемый граф сильно связен, а время передачи сообщения по дуге ограничено сверху 1 тактом.

Как алгоритмы исследования графа, так и оценка времени их работы, существенно зависят от того, имеют ли автоматы в вершинах графа какую-то информацию о графе, или каждый автомат находится в начальном состоянии и «ничего не знает» о графе. В [1] мы предложили два алгоритма: алгоритм разметки и алгоритм пульсации. Алгоритм разметки выполняет первичный обхода графа с помощью пересылаемых сообщений, когда в начальный момент времени все автоматы находятся в начальном состоянии. После завершения обхода автоматы остаются в некоторых конечных состояниях, вообще говоря, отличающихся от начального, что, по сути, задаёт разметку графа. Такая разметка позволяет быстрее выполнять параллельное вычисление требуемой функции с помощью алгоритма пульсации. Разметка графа выполняется за время порядка $n/k+D$, а вычисление функции – за время порядка D.

В алгоритме пульсации сначала от автомата выделенной начальной вершины (корня) по всему графу распространяются сообщения-вопросы, которые должны достигнуть каждой вершины. А затем от каждой вершины «в обратную сторону» к корню двигаются сообщения-ответы. С помощью алгоритма пульсации можно параллельно вычислять любую функцию от мультимножества значений, записанных в памяти автоматов по всем вершинам графа (мы будем говорить «записанных в вершинах»). Вот лишь несколько примеров таких функций:

1. Максимум чисел, записанных в вершинах графа.

2. В более общем виде вместо максимума можно использовать любую коммутативную и ассоциативную операцию над числами: минимум, сложение, произведение и т.д.
3. Частные случаи: число вершин в графе, если в каждой вершине записать «1», и число дуг в графе, если в каждой вершине записать число выходящих дуг.
4. Дизъюнкция логических значений, записанных в вершинах графа.
5. В более общем виде вместо дизъюнкции можно использовать любую коммутативную и ассоциативную операцию над логическими значениями: конъюнкцию, эквивалентность и т.д.
6. В ещё более общем виде вместо чисел или логических значений можно использовать любые значения и любые коммутативные и ассоциативные операции над ними.
7. Среднее арифметическое, среднее геометрическое или среднее квадратичное от чисел, записанных в вершинах графа.

Второе направление [2] – это исследование динамически меняющегося графа. Исследование проводится также с помощью автоматов, находящихся в вершинах графа и обменивающихся между собой сообщениями, передаваемыми по дугам графа в направлении их ориентации. Ёмкость дуги предполагается равной 1. Особенность в том, что дуги графа могут динамически изменяться: появляться, исчезать или менять свою конечную вершину. При исчезновении дуги передаваемое по ней сообщение теряется, а при смене конца дуги сообщение попадает в новую конечную вершину.

Понятно, что если время существования дуги до её исчезновения или смены конца слишком мало, сообщение, передаваемое по ней, потеряется или не будет передано в «нужную» вершину. Поэтому хотя бы некоторые дуги должны быть «долгоживущими», чтобы за время существования такой дуги по ней могло пройти хотя бы одно сообщение. Для того чтобы можно было исследовать весь граф, достаточно, чтобы в каждый момент времени такие долгоживущие дуги образовывали сильно связный суграф.

В [2] целью исследования графа назван сбор полной информации о графе в корне графа. Такая информация представляет собой набор описаний всех дуг графа. Однако, поскольку граф динамически меняется, мы не можем гарантировать, что описания текущего состояния всех его дуг отражены в корне: сообщения о последних изменениях дуг могут просто не дойти до корня. Поэтому требуется только, чтобы через время, ограниченное сверху величиной T_0, после изменения дуги корень графа «узнал» об этом или более позднем изменении дуги. Если после данного изменения дуга больше не меняется, по крайней мере, в течение времени T_0, то в корне будет правильное описание этой дуги. Если в какой-то момент времени все изменения в графе прекращаются, то через время, ограниченное сверху величиной T_1, в корне

окажется полное описание состояния графа после прекращения изменений. В [2] предложен алгоритм исследования графа с оценками $T_0=\boldsymbol{O}(n)$ и $T_1=\boldsymbol{O}(D)$, где D – диаметр графа после прекращения изменений.

В данной работе мы попытались совместить идею параллельного вычисления на графе из [1] и идею исследования динамически меняющегося графа из [2]. В разделе 2 кратко излагается теория агрегатных функций и агрегатных расширений функций. Раздел 3 содержит постановку задачи, в том числе, все предположения и ограничения на устройство графа, работу автоматов и передачу сообщений. При этих предположениях и ограничениях в разделе 4 описывается общая идея алгоритма. Формальное описание алгоритма приведено в разделе 5, а доказательство правильности работы алгоритма, а также оценка времени работы алгоритма, размеров памяти автомата и сообщений приведены в разделе 6.

2. Агрегатные функции и агрегатные расширения функций

Алгоритм пульсации, по сути, вычисляет агрегатные функции, для которых значение функции от объединения мультимножеств вычисляется по значениям функции от этих мультимножеств. В [1] мы дали формальное определение агрегатной функции, доказали критерий агрегатности и показали, что любая функция F имеет агрегатное расширение, то есть может быть вычислена как композиция функций HG, где G агрегатная функция. Также показали, что существует и единственное минимальное агрегатное расширение, вычисляющее минимум информации, по которой еще можно восстановить функцию F. Эта теория агрегатных функций является модификацией теории индуктивных функций в [11]. Здесь мы повторим определения и утверждения из [1], опустив доказательства.

Далее рассматриваются функции на конечных мультимножествах из элементов базового множества X. Множество всех конечных мультимножеств из элементов X обозначается, как и выше, через $X^{\#}$. Под операциями объединения, пересечения, дополнения и пр. далее подразумеваются операции на мультимножествах, т.е., учитывающие кратности элементов. Через $\boldsymbol{N}$ обозначим множество натуральных чисел.

Агрегатная функция $G:X^{\#}\to B$ – это такая функция, что

$\exists E:B\times B\to B\ \ \forall a,b\in X^{\#}\ \ G(a\cup b) = E(G(a),G(b))$.

Замечание 2.1: Для агрегатной функции G выполнено следующее: $\forall r\in\boldsymbol{N}\ \exists E_r : B^r\to B\ \ \forall a_1,\ldots,a_r\in X^{\#}\ G(\cup\{a_i \mid 1\le i\le r\}) = E_r(G(a_1),\ldots,G(a_r))$.

Утверждение 2.1: Функция $G:X^{\#}\to B$ является агрегатной тогда и только тогда, когда $\forall a,b\in X^{\#}\ \forall x\in X\ G(a) = G(b) \Rightarrow G(a\cup\{x\}) = G(b\cup\{x\})$.

Замечание 2.2: Из утверждения 2.1 следует, что $G : X^{\#}\to B$ является агрегатной тогда и только тогда, когда $\exists G` : B\times X\to B\ \ \forall a\in X^{\#}\ x\in X\ G(a\cup\{x\}) = G`(G(a), x)$. Действительно, достаточно определить $G`(G(a), x) = E(G(a), G(\{x\}))$.

Агрегатным расширением функции $F:X^{\#}\to A$ назовём агрегатную функцию $G:X^{\#}\to B$, такую, что $\exists H : B\to A\ \ \forall a\in X^{\#}\ F(a) = H(G(a))$.

Агрегатное расширение $G : X^{\#}\to B$ функции $F : X^{\#}\to A$ назовём *минимальным*, если $G(X^{\#}) = B$ и $\forall G` : X^{\#}\to C$ являющего агрегатным расширением $F$ имеет место $\exists i : C\to B\ G = iG`$.

Замечание 2.3: Агрегатное расширение G функции F представляет собой агрегатную функцию, по которой можно вычислить функцию F. При этом возможны такие расширения, которые на практике не помогают в этом, например, можно взять в качестве G тождественную функцию на $X^{\#}$, а в качестве H – саму функцию F. Чтобы избежать такого, используется минимальное агрегатное расширение; интуитивно, это агрегатная функция, вычисляющая минимум информации, по которой еще можно восстановить F.

Утверждение 2.2: Минимальное агрегатное расширение любой функции $F : X^{\#}\to A$ существует и единственно с точностью до взаимно однозначных отображений.

Замечание 2.4: Примеры минимального агрегатного расширения (π_i – проекция кортежа на i-ый компонент).

- Для функции вычисления среднего арифметического $F(a_1,\ldots,a_n) = (a_1+\ldots+a_n)/n$ минимальным агрегатным расширением является функция $G(a_1,\ldots,a_n) = (a_1+\ldots+a_n, n)$; $F = \pi_1 G / \pi_2 G$.
- Для функции вычисления среднего геометрического $F(a_1,\ldots,a_n) = \sqrt[n]{(a_1\cdot\ldots\cdot a_n)}$ минимальным агрегатным расширением является функция $G(a_1,\ldots,a_n) = (a_1\cdot\ldots\cdot a_n, n)$; $F = \sqrt[\pi_2 G]{\pi_1 G}$.
- Для функции вычисления среднего квадратичного $F(a_1,\ldots,a_n) = \sqrt{((a_1^2+\ldots+a_n^2)/n)}$ минимальным агрегатным расширением является функция $G(a_1,\ldots,a_n) = (a_1^2+\ldots+a_n^2, n)$; $F = \sqrt{(\pi_1 G / \pi_2 G)}$.

Итак, каждую функцию $F:X^{\#}\to A$ можно представить в виде $F(a) = H(G(a))$ и $\forall b,c\in X^{\#}\ \ G(b\cup c) = E(G(b),G(c))$. *Разбиением* мультимножества $b\in X^{\#}$ называется набор $b_1,\ldots,b_r$ его подмножеств, объединение которых совпадает с b: $b=b_1\cup\ldots\cup b_r$. Будем говорить, что задано *вложенное разбиение* мультимножества $b\in X^{\#}$, если задано его разбиение $b_1,\ldots,b_r$ и для каждого не синглетонного (содержащего более одного элемента) мультимножества b_i также задано вложенное разбиение. Тогда, если задано вложенное разбиение мультимножества $a\in X^{\#}$, то вычисление функции $F(a)$ можно выполнить следующим образом. Сначала вычисляются значения $G(x)$ для каждого $x\in a$ (без учёта кратности). Далее, учитывая замечание 2.1, для каждого элемента

$b=b_1\cup..\cup b_r$ вложенного разбиения вычисляется значение $G(b) = E_r(G(b_1),...,G(b_r))$. При этом сама функция E_r может вычисляться итеративно с помощью функции E: для $r>2$ имеем $E_r(G(b_1),...,G(b_r)) = E(E_{r-1}(G(b_1),...,G(b_{r-1})),G(b_r))$. После того, как будет получено значение $G(a)$, вычисляется искомый результат $F(a) = H(G(a))$.

3. Постановка задачи

Постановка задачи совмещает предположения и ограничения, сформулированные в [1] и [2], с тремя отличиями, которые мы специально оговорим.

Рассматривается ориентированный граф с n вершинами и m дугами. Множество вершин обозначим через V, одна из вершин $v_0\in V$ выделена и называется *корнем*. Пусть также задан некоторый алфавит X, множество всех конечных мультимножеств из элементов X обозначим через $X^\#$. Будем считать, что задана функция $q : V\to X$. Мультимножество, образуемое значениями $q(v)$ во всех вершинах графа, обозначим через $q(V)\in X^\#$. Для любой функции $F:X^\#\to A$ требуется вычислить значение $F(q(V))$. Как показано в разделе 2 функцию можно представить в виде $F=HG$, где $G : X^\#\to B$ – минимальное агрегатное расширение, и $H : B\to A$. При этом существует такая функция $E : B\times B\to B$, что $\forall a,b\in X^\# \; G(a\cup b) = E(G(a),G(b))$.

Вычисления выполняются автоматами, находящимися в вершинах графа и обменивающихся между собой сообщениями по дугам графа. Иногда для краткости мы будем вместо «автомат вершины» писать просто «вершина», если это не приводит к недоразумениям. Автомат в вершине может послать сообщение по любой из выходящих дуг и принять сообщение по любой из входящих дуг. Автомат начинает работать, получая первое сообщение по входящей дуге. Корень получает первое сообщение «*Старт*» извне графа, это сообщение инициирует работу алгоритма разметки; в ответном сообщении «*Готов к работе*» корень извещает о завершении разметки и готовности к вычислению функции.

Вычисление функции выполняется в алгоритме пульсации. Оно также инициируется извне с помощью сообщения «*Вопрос*», направляемого в корень и содержащего описание функции F: H,G,E. Когда корень вычисляет значение функции $F(q(V))$, он посылает вовне сообщение «*Ответ*», содержащее вычисленное значение и означающее готовность принять следующий вопрос. Мы будем предполагать, что сообщения извне приходят только в корень, а не в какую-нибудь другую вершину.

Автомат в вершине v может узнать значение $q(v)$ с помощью примитива «*Дай значение*». Каждая вершина имеет уникальный (среди вершин) *идентификатор вершины*. Автомат может узнать идентификатор своей вершины с помощью примитива «*Дай идентификатор*».

Все дуги, выходящие из вершины, перенумерованы, начиная с номера 1 до наибольшего номера, не превосходящего некоторого числа s. Очевидно, $s\le m$. Дуга однозначно идентифицируется парой (идентификатор начала дуги, номер дуги), которую мы будем называть *идентификатор дуги*. Число дуг m – это число различных идентификаторов дуг в графе.

Автомат, посылая сообщение по дуге, указывает её номер. В отличие от [1] и так же как в [2], мы будем считать, что ёмкость дуги $k=1$. Это первое отличие. Оно объясняется тем, что граф может динамически меняться, и гарантируется передача по дуге только одного сообщения, после чего дуга может измениться, и даже это гарантируется не для всех дуг, как описано ниже. Поэтому в наихудшем случае, для которого и делается оценка времени работы алгоритма, случаи $k>1$ и $k=1$ не отличаются. Иными словами, даже если $k>1$, автомат не будет посылать по дуге сообщение до тех пор, пока предыдущее сообщение продолжает передаваться по дуге. Будем говорить, что дуга занята, если по ней передаётся сообщение. Иначе дуга свободна. Сообщение посылается только по свободной дуге. В самом начале все дуги свободны. Когда сообщение передано по дуге, начало дуги извещается об этом сигналом *освобождения* дуги (*сигнал О*) с параметром номер дуги.

Граф может динамически меняться, причём вершины не меняются, а дуги могут изменяться следующим образом:

- Дуга может появиться, о чём начало дуги извещается сигналом *появления* дуги (*сигнал П*) с параметром номер дуги.
- Дуга может исчезнуть. Если по дуге передавалось сообщение, то оно пропадает, и в этом случае начало дуги извещается сигналом *исчезновения* дуги (*сигнал И*) с параметром номер дуги. Заметим, что если по дуге никакого сообщения не передавалось, то сигнала И не будет; однако если вершина попытается послать сообщение по исчезнувшей дуге, сообщение не будет передано, а вершина получит сигнал И.
- Дуга может поменять свой конец; в этом случае никаких сигналов не предусмотрено.

Из-за того, что дуги могут исчезать и появляться, номера дуг, выходящих из одной вершины и имеющихся (появившихся и не исчезнувших) в данный момент времени, могут идти не подряд, т.е. не образовывать отрезок натурального ряда, но в любом случае они располагаются на отрезке от 1 до s.

Итак, если не считать идентификатора вершины v и значения $q(v)$, которые автомат получает с помощью соответствующих примитивов, входными символами автомата, находящегося в вершине, являются сигналы от дуг, выходящих из вершины, и сообщения, получаемые по дугам, входящим в вершину. Срабатывание автомата – это обработка одного такого входного символа. Мы будем предполагать, что входные символы, как правило,

поступают в автомат в порядке их возникновения. Это означает, что существует очередь входных символов автомата.

Сигнал О вырабатывается не тогда, когда сообщение ставится в очередь входных символов, а тогда, когда оно выбирается из этой очереди автоматом конца дуги. Поэтому только после того, как сообщение будет принято автоматом конца дуги, по этой дуге будет послано следующее сообщение, а ещё позже оно будет поставлено в очередь в конце дуги. Следовательно, в очереди входных символов может быть не более одного сообщения для каждой входящей дуги.

Если сигнал от некоторой выходящей дуги вырабатывается в тот момент времени, когда предыдущий сигнал от этой дуги ещё находится в очереди входных символов, то новый сигнал не ставится в конец очереди, а замещает собой старый сигнал. Можно отметить, что таким новым сигналом может быть только сигнал П.

Исключением является случай, когда старый сигнал – это сигнал О. В отличие от [2] сигнал О не замещается сигналом П: в очереди остаётся сигнал О, а сигнал П игнорируется. Это второе отличие. В [2] замещение сигнала О сигналом П информировало автомат начала дуги о том, что дуга изменилась: исчезла и снова появилась. Это было важно, поскольку ставилась цель мониторинга динамического графа, т.е. нужно было отслеживать все изменения его дуг. В данной работе у нас нет такой цели, зато нужно, чтобы автомат в начале дуги мог узнать, дошло сообщение, посланное по дуге, до её конца или пропало. Это используется в алгоритме разметки для того, чтобы определить момент завершения сбора информации о всех вершинах графа (подробнее смотри ниже в подразделе 4.3). Если как в [2] сигнал О замещается сигналом П, автомат в начале дуги не различает следующие два случая. 1) Сообщение дошло до конца дуги, сгенерирован сигнал О; до его обработки дуга исчезает (сигнала И нет, так как на дуге нет сообщения), а затем появляется: сигнал П замещает сигнал О. 2) Сообщение пропадает, так как дуга исчезает. Генерируется сигнал И. До его обработки дуга снова появляется, и сигнал П замещает собой сигнал И. В обоих случаях автомат начала дуги получит сигнал П, однако в 1-ом случае сообщение дошло до конца дуги, а во 2-ом случае пропало. Итак, в [2] замещение сигналов происходит по схеме «П+П→П, И+П→П, О+П→П», а в данной работе – по схеме «П+П→П, И+П→П, О+П→О».

Заметим, что если бы нам нужно было отслеживать все изменения дуг как в [2] и гарантированно определять прохождение сообщения по дуге, как в данной работе, то нужно было бы сохранить оба сигнала: О и П. Сигнал О при появлении сигнала П нужно было бы замещать новым сигналом «*освобождение+появление*», который информировал бы начало дуги как о том, что сообщение дошло до конца дуги, так и о том, что после этого дуга изменилась: исчезла и снова появилась.

Из-за замещения сигналов от одной выходящей дуги в очереди входных символов может быть не более одного сигнала для каждой выходящей дуги. Тем самым, длина очереди входных символов автомата не превосходит степени вершины, то есть не больше $2m$ ($2m$ достигается для $n=1$, когда все дуги – петли, поскольку петля считается два раза: как входящая – для сообщений и как выходящая – для сигналов).

Обозначим через T_{ab} максимальное время, за которое информация, имеющаяся в некоторой вершине a, доходит до вершины b, и $T = max\{T_{ab} \mid a,b \in V\}$. Для того чтобы время T было конечным, нужно, чтобы сообщения могли распространяться по графу, доходя от каждой вершины a до каждой вершины b. Для этого нужно, чтобы время x пересылки сообщения по дуге и время y срабатывания автомата были конечными. Кроме того, *время z существования* дуги, т.е. время от появления дуги или изменения её конца до исчезновения дуги или изменения её конца, должно быть достаточно велико, чтобы по дуге успевало пройти хотя бы одно сообщение. Такими «долгоживущими» дугами могут быть не все дуги: достаточно, чтобы в каждый момент времени «долгоживущие» дуги порождали сильно связный суграф (подграф, содержащий все вершины графа).

Для ограниченности времени T необходимо, чтобы x и y были ограничены сверху: $x \le X$ и $y \le Y$, а z – снизу: $z \ge Z$. Выразим эту нижнюю границу Z через X и Y. Сообщение можно посылать по дуге сразу после того, как она появилась или освободилась (сигналы П, О). Однако в этот момент времени сигнал П или О ещё только ставится во входную очередь символов автомата начала дуги, длина которой может достигать $2m$. Поэтому сообщение посылается по дуге не сразу, а через время, которое может достигать $2mY$. Следовательно, сообщение дойдёт до конца дуги и будет поставлено во входную очередь символов автомата конца дуги через время, которое может достигать $2mY+X$, если в течение этого времени дуга не менялась. Поэтому будем предполагать, что нижняя граница времени существования «долгоживущих» дуг $Z \ge 2mY+X$.

Для оценки времени работы алгоритма мы будем для простоты предполагать, что временем срабатывания автомата можно пренебречь, то есть $Y=0$, время пересылки по дуге не превосходит 1 такта, то есть $X=1$, а время существования «долгоживущей» дуги может быть минимально, то есть $Z=2mY+X=1$.

Также как в [2] предполагается, что в начальный момент времени в каждой вершине имеется автомат в начальном состоянии, и для каждой дуги, имеющейся в графе в начальный момент времени, выработан сигнал П. Третье отличие от [2] – понятие начальной дуги и предположение о начальных дугах. Начальной дугой будем называть такую дугу $a \to b$, которая существует в начальный момент времени и которая не меняется до тех пор, пока по ней не пройдёт первое сообщение от a до b. Будем предполагать, что каждая вершина

достижима из корня по начальным дугам. Начальные дуги нужны для того, чтобы в алгоритме разметки определять момент завершения сбора информации о всех вершинах графа. Подробнее об этом будет рассказано ниже при описании алгоритма разметки.

4. Идея алгоритма

4.1. Распространение информации по динамическому графу

Предположение о долгоживущих дугах гарантирует возможность доставки информации из любой вершины в любую вершину, но только при условии, что автоматы в вершинах соблюдают определённую дисциплину передачи сообщений.

1) Каждая вершина, получив сообщение с данной информацией, должна сохранить её в своей памяти и далее помещать во все сообщения, которые постоянно нужно посылать по всем выходящим дугам, когда появляется такая возможность (по сигналам О, П).

Казалось бы, информацию можно не посылать по дуге повторно, однако пример на рис. 1 показывает, что в этом случае нет гарантии доставки информации. Здесь затемнёнными кружками показаны вершины, в которые поступила новая информация, двойной линией показаны дуги, по которым информация уже была отправлена и дошла до конца дуги. При переходе от п.4 к п.5 дуги 1 и 2 меняют свои концы, но по ним информация повторно не посылается. Из-за этого информация не достигает вершины *c*.

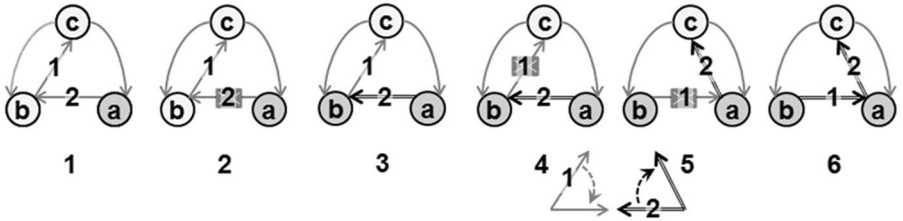

Рис. 1. Пример нераспространения информации по графу.

2. Из 1-го требования непосредственно следует, что если из вершины нужно передать в какие-то другие вершины несколько порций информации, то они должны посылаться не последовательными сообщениями, а помещаться в одно сообщение.

Эти требования достаточны для гарантированной доставки информации из каждой вершины в каждую вершину, что строго показано в [2] в несколько изменённой форме (Лемма 5). Здесь мы приводим доказательство аналогичного утверждения в нужной нам форме.

Лемма 1. Пусть в некоторый момент времени в некоторой вершине имеется новая информация. Тогда не более чем через *3(n-1)* тактов эта информация будет доставлена в каждую вершину.

Доказательство. Пусть в некоторый момент времени $A \neq \varnothing$ – это множество вершин, в которых уже есть нужная информация. В каждый момент времени сураф, порождённый «долгоживущими» дугами, сильно связен, время существования «долгоживущей» дуги не меньше 1 такта, время пересылки по дуге не больше 1 такта, а временем срабатывания автомата мы пренебрегаем. Кроме того, в каждой вершине при появлении или освобождении дуги по ней сразу посылается сообщение.

Докажем *утверждение о расширении* множества A: с момента времени t_1, когда образуется множество $A \neq \varnothing$ и $A \neq V$, не более чем через 3 такта множество A будет расширено.

Очевидно, что множество A может только расширяться (не может сужаться). В любой момент времени, в том числе в момент времени t_1+2, должна существовать «долгоживущая» дуга p, ведущая из вершины $a \in A$ «наружу», т.е. в вершину $b \notin A$. В момент времени t_1+2 по дуге p передаётся сообщение S. Это сообщение отправлено из a не ранее момента времени $(t_1+2)-1=t_1+1$, поэтому S содержит нужную информацию. Возможны два случая.

1) Сообщение S дойдёт до вершины b. Тогда это произойдёт не позднее момента времени $(t_1+2)+1=t_1+3$. Следовательно, в этом случае не более чем через 3 такта после момента времени t_1 множество A будет расширено.
2) Сообщение S не дойдёт до вершины b. Это означает, что дуга p изменится до того, как по ней до вершины b дойдёт сообщение S. А тогда, поскольку p – «долгоживущая» дуга, по ней должно дойти до вершины b предыдущее сообщение $S`$. И это должно произойти до момента времени $t_1+2$. Поскольку сообщение $S$ отправлено из $a$ не ранее момента времени $t_1+1$, сообщение $S`$ принято в b не ранее этого же момента времени. Следовательно, сообщение $S`$ отправлено из $a$ не ранее момента времени $(t_1+1)-1=t_1$, поэтому $S`$ содержит нужную информацию. Следовательно, в этом случае не более чем через 2 такта после момента времени t_1 множество A будет расширено.

При каждом расширении множества A в него добавляется хотя бы одна вершина. Поскольку с самого начала множество A содержит хотя бы одну вершину, а общее число вершин равно n, получается, что число расширений множества A не более $n-1$. Таким образом, требуется не более чем $3(n-1)$ тактов, чтобы в каждой вершине оказалась нужная информация.

Лемма доказана.

4.2. Оптимальная разметка графа

В [1] разметка графа состояла из двух остовных (содержащих все вершины графа) деревьев: *прямое дерево*, ориентированное от корня, и *обратное дерево*, ориентированное к корню. В каждой вершине (кроме листовых вершин прямого дерева) отмечены выходящие прямые дуги, т.е. дуги прямого дерева. Также в каждой вершине, кроме корня, отмечена одна выходящая обратная дуга, т.е. дуга обратного дерева. Кроме того, в каждой вершине устанавливался счётчик числа входящих обратных дуг. В алгоритме пульсации сообщение-вопрос, содержащее описание функции, распространялось от корня по прямому дереву. Обратное дерево задавало вложенное разбиение мультимножества $q(V)$, т.е. было предназначено для сбора ответов.

Листовая вершина обратного дерева, получив вопрос, сразу же вычисляет свой ответ и посылает его по выходящей обратной дуге. Внутренняя (не листовая) вершина обратного дерева сначала собирает сообщения-ответы по всем входящим в неё обратным дугам, после чего вычисляет свой ответ и посылает его по выходящей обратной дуге. Корень посылает ответ вовне графа.

Такой алгоритм позволял распространять по графу вопрос за время $\boldsymbol{O}(h_0)$, где h_0 – высота прямого дерева, после чего вычислять функцию, собирая все ответы, за время $\boldsymbol{O}(h)$, где h – высота обратного дерева. Однако если граф динамически меняется, то возникает проблема: изменение графа может потребовать изменения прямого или обратного дерева, если меняется (исчезает или меняет свой конец) дуга дерева. В общем случае такое изменение может потребовать полной перестройки деревьев, т.е. фактически понадобится строить эти деревья заново. Кроме того, не гарантируется доставка сообщений по дугам построенного дерева: эти дуги могут исчезать или менять свой конец до того, как по ним будет передано нужное сообщение.

В то же время предположение о долгоживущих дугах гарантирует возможность доставки информации из любой вершины в любую вершину. Для этого, как показано в подразделе 4.1 и в [2], каждая вершина, получив сообщение с этой информацией, должна сохранить её в своей памяти и далее помещать во все сообщения, которые постоянно нужно посылать по всем выходящим дугам, когда появляется такая возможность (по сигналам О, П).

Тогда вместо дуги $a \to b$ дерева можно использовать просто пару вершин (a,b), которую назовём виртуальной дугой. Информация, которую нужно переправить по виртуальной дуге, помещается в сообщение, которое отправляется из вершины a и распространяется «веером» по всем дугам, а когда сообщение с этой информацией попадает в вершину b, вершина получает требуемую информацию. Правда, время перемещения сообщения по дуге равно $\boldsymbol{O}(1)$, а по виртуальной дуге – $\boldsymbol{O}(n)$. Соответственно, вместо дерева

используется виртуальное дерево как набор виртуальных дуг. Вместо оценок $\boldsymbol{O}(h_0)$ и $\boldsymbol{O}(h)$ получатся оценки $\boldsymbol{O}(nh_0)$ и $\boldsymbol{O}(nh)$.

Сообщение-вопрос содержит одну и ту же информацию, предназначенную всем вершинам: функции G и E (функция H нужна только в корне для окончательного вычисления значения функции F). Такое сообщение может распространиться из корня по всему графу без использования прямого дерева за время $\boldsymbol{O}(n)$, что лучше, чем $\boldsymbol{O}(nh_0)$. Поэтому виртуальное прямое дерево не требуется.

Однако ответы, вычисляемые разными вершинами, разные. Кроме того, как показано в подразделе 4.1, все ответы, которые должны быть отправлены из вершины (ответ, вычисленный в ней, или проходящий «транзитом») должны отправляться в одном сообщении, и в этом же сообщении следует отправлять вопрос. Только в этом случае гарантируется доставка всей требуемой информации адресатам.

В то же время использование виртуального обратного дерева позволяет передавать в одном сообщении не все ответы. Дело в том, что если вершина a уже получила ответ от вершины b (или вычислила его сама при $b=a$) и получает сообщение с ответом от вершины c, расположенной на той же ветви обратного дерева, что вершина b, но *ниже* (ближе к корню) вершины b, то ответ от вершины b (вычисленное ею значение функции G) уже не нужен, поскольку он уже использован при вычислении ответа от c. Аналогично, если вершина a получает сообщение с ответом от вершины c, расположенной на той же ветви обратного дерева, что вершина b, но *выше* (дальше от корня) вершины b, то ответ от вершины c (вычисленное ею значение функции G) уже не нужен, поскольку он уже использован при вычислении ответа от b. Это означает, что в одном сообщении достаточно иметь не более одного ответа от вершин на одной ветви обратного дерева. Если есть ответ хотя бы от одной вершины ветви дерева, то достаточно ответа от самой нижней (ближней к корню) вершины.

Таким образом, наличие виртуального обратного дерева позволяет уменьшить размер сообщения. Без обратного дерева можно было бы собирать в корне просто мультимножество значений функции q, получая от каждой вершины v_i, где $i=1..n$, значение $q_i=q(v_i)$, и все вычисления производить в корне $F=H(E_n(G(q_1),...,G(q_n)))$. Однако значение $q(v_i)$ может быть слишком большим, например, это может быть база данных большого размера. Можно было бы также собирать в корне мультимножество значений композиции функций Gq, получая от каждой вершины v_i, где $i=1..n$, значение $G_i=G(q_i)$, и аналогично все вычисления производить в корне $F=H(E_n(G_1,...,G_n))$. Однако в этом случае в одном сообщении могут оказаться значения G_i для всех вершин v_i, кроме корня, т.е. до $n-1$ значения функции G.

Дерево определяется двумя параметрами: высотой h и шириной (числом листовых вершин = число ветвей от листа до корня, причём корень листом не

считается) w. Высота h влияет на время работы алгоритма пульсации, а ширина w – на размер сообщения. Возникает задача: для данного числа вершин n определить минимальную высоту дерева h при заданной ширине $w \le n-1$ и описать деревья с такой минимальной высотой. Очевидно, что $n \le hw+1$. Отсюда $h \ge (n-1)/w$. Следовательно, минимальная высота $h =](n-1)/w[$ (ближайшее сверху целое число). Это достигается для деревьев вида, изображённого на рис. 2, который можно назвать «сбалансированный веник».

Заметим, что если разрешить задавать ширину $w>n-1$, то реальная ширина дерева всё равно будет не больше $n-1$. В частности, если $n=1$, т.е. граф содержит одну вершину, то дерево будет иметь фиксированную ширину $w=0$ и высоту $h=0$.

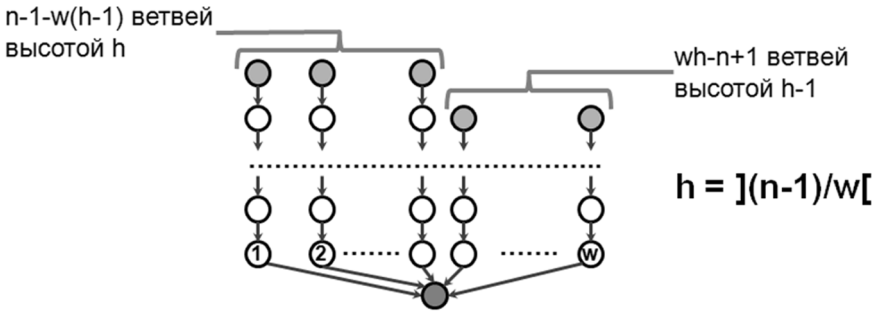

Рис. 2. «Сбалансированный веник»

Для «сбалансированного веника» внутренняя вершина (не лист и не корень) имеет только одну входящую дугу, поэтому для неё не требуется счётчика входящих дуг. Такой счётчик нужен только для корня, его значение равно w. Некорневой вершине достаточно «знать», является она листовой (счётчик равен 0) или внутренней (счётчик равен 1) вершиной дерева.

В [1] разметка графа при $k=1$ занимала время $\boldsymbol{O}(n)$. При этом можно было использовать относительно короткие сообщения, поскольку необходимую информацию можно было доставлять в вершины не одним, а несколькими сообщениями. Правда, при этом строились не оптимальные прямое и обратное деревья, а те, которые «получались» при данном алгоритме разметки и данном графе. Как показано в подразделе 4.1, в динамическом графе мы вынуждены передавать всю необходимую информацию в одном сообщении, размер которого, тем самым, существенно больше. За счёт этого мы можем за то же по порядку время $\boldsymbol{O}(n)$ строить оптимальное обратное виртуальное дерево. Как показано выше, такое дерево имеет вид «сбалансированного веника», а прямое виртуальное дерево излишне.

Обратное виртуальное дерево вида «сбалансированного веника» (далее просто дерево) можно описать с помощью двухуровневых индексов вершин. Индекс

состоит из номера ветви дерева от 1 до w и из номера вершины на ветви от 1 до h или $h-1$, считая, что корень имеет номер 0 на любой ветви. Таким образом, корень имеет w индексов вида $(i,0)$, а другая вершина имеет только один индекс. Дерево описывается как множество описаний некорневых вершин. Описание некорневой вершины содержит тип вершины «лист» или «внутренняя», идентификатор вершины и её индекс. Виртуальная дуга $a \to b$ существует тогда и только тогда, когда a имеет индекс (i,j), $j>0$ и b имеет индекс $(i,j-1)$.

Алгоритм разметки выполняется в два этапа. На первом этапе в корне графа собирается информация о всех вершинах. После этого корень строит в своей памяти описание дерева, которое на втором этапе рассылается во все вершины.

4.3. Разметка графа: сбор информации о вершинах

Разметка графа начинается с получения извне сообщения «Старт» с параметром w, которое должно приходить в корень по предположению из раздела 3. На этапе сбора информации о вершинах графа по графу циркулируют сообщения типа 1. Вершина графа и каждое сообщение содержат множество описаний дуг.

Описание дуги создаётся в её начале при появлении дуги (по сигналу П), но только для тех дуг, которые появляются до прихода в вершину первого сообщения. Предположение о начальных дугах в разделе 3 гарантирует, что будут созданы описания всех начальных дуг. При получении сообщения вершина переписывает к себе отсутствующие у неё описания дуг из сообщения, или корректирует имеющиеся у неё описания дуг. Посылая сообщение, вершина помещает в него всё накопленное в ней множество описаний дуг. Тем самым, сообщение аккумулирует в себе описания дуг из всех вершин, через которые оно проходит.

Описание дуги содержит: идентификатор дуги (идентификатор начала дуги и номер дуги в начале дуги) и статус дуги. Статус дуги принимает три значения: 1,2,3. Схема изменения статуса дуги $a \to b$ на рис. 3 иллюстрирует следующие правила:

Правило 1. *Нет описания дуги* $\to 1$ (сплошная линия на рис.3). Описания дуги $a \to b$ со статусом 1 создаётся в вершине a по сигналу П ещё до того, как в вершину a поступит первое сообщение. Предположение о существовании начальных дуг в начальный момент времени (раздел 3) гарантирует, что в вершине a будут созданы описания всех начальных дуг, выходящих из a (хотя не обязательно только начальных). Поэтому первое и, следовательно, каждое сообщение, посылаемое из вершины a, содержит описания этих дуг.

Правило 2. Изменение статуса дуги *1* → *2* (сплошная линия на рис.3) происходит в вершине *a*, когда вершина *a* по сигналу О узнаёт о том, что первое сообщение, посланное по дуге *a*→*b*, дошло до её конца, вершины *b*.

Правило 3. Изменение статуса дуги *1* → *3* (сплошная линия на рис.3) происходит в вершине *a*, когда вершина *a* по сигналу И или П узнаёт о том, что первое сообщение, посланное по дуге *a*→*b*, не дошло, или, по повторному сигналу П, узнаёт о том, что дуга повторно появилась (и, следовательно, исчезала) до посылки по ней первого сообщения. В обоих случаях дуга не является начальной.

Правило 4. Такое же изменение статуса дуги *1* → *3* (сплошная линия на рис.3) происходит в вершине *b*, когда вершина *b* получает по дуге *a*→*b* сообщение, в котором статус дуги *a*→*b* равен *1* (это будет первое сообщение, посылаемое по дуге *a*→*b*).

Правило 5. Изменение статуса дуги происходит также в любой вершине при получении сообщения, в котором эта дуга имеет статус больше, чем в вершине или в вершине отсутствует описание дуги (сплошная или пунктирная стрелка на рис.3).

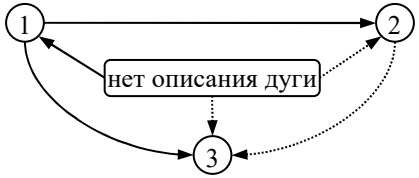

Рис. 3. Схема изменения статуса дуги

Корень получает информацию о вершине *a*, когда получает сообщение, содержащее описание некоторой дуги *a*→*b*, в котором качестве идентификатора начала дуги указан идентификатор вершины *a*. Если статус дуги в корне равен *3*, то это означает одно из двух. 1) Статус *3* установлен в конце дуги, вершине *b*, и сообщение из *b* дошло до корня. А тогда корень имеет информацию хотя бы об одной дуге, выходящей из *b*, тем самым корень знает о вершине *b*. 2) Статус *3* установлен в начале дуги, вершине *a*, следовательно, эта дуга не начальная.

Как сказано выше, предположение о существовании начальных дуг в начальный момент времени гарантирует, что корень, узнавая о вершине (т.е. хотя бы об одной выходящей из неё дуге), узнаёт о всех выходящих из неё дугах, которые были в начальный момент времени. Предположение о том, что любая вершина достижима из корня по начальным дугам, гарантирует, что до каждой вершины от корня по начальным дугам дойдёт сообщение, и это сообщение будет первым сообщением, передаваемым по этим дугам. Предположение о долгоживущих дугах гарантирует, что до корня дойдут все сообщения, посылаемые из вершин графа. Поэтому, когда в корне статус всех

дуг станет равным *3*, это будет означать, что корень узнал о всех вершинах графа. На этом первый этап заканчивается и начинается второй этап рассылки описаний виртуального обратного дерева.

4.4. Разметка графа: рассылка описания дерева

Когда информация о всех вершинах собрана в корне, корень строит виртуальное обратное дерево как множество описаний некорневых вершин (подраздел 4.2). Цель второго этапа разметки графа – доставить в каждую некорневую вершину её двухуровневый индекс и её тип «лист» или «внутренняя».

Для этого по графу распространяются сообщения типа 2, содержащие множества описаний вершин, из которых каждая вершина запоминает своё описание и удаляет его из множества.

Когда вершина *a* (не корень) первый раз получает сообщение типа *2*, она переписывает в свою память всё множество описаний вершин из сообщения. В этом множестве должно быть и описание вершины *a*, которое определяется по идентификатору вершины *a*. Из этого описания запоминается индекс и тип вершины *a*, а само описание удаляется из множества. После этого вершина игнорирует все принимаемые сообщения типа *1*, которые ещё могут оставаться на графе.

В дальнейшем, когда вершина *a* получает сообщение типа *2*, она строит пересечение множеств описаний вершин из сообщения и из своей памяти. Это пересечение сохраняется в памяти вершины *a*.

Каждый раз, когда по дуге, выходящей из *a*, можно послать сообщение (по сигналу О или П), вершина *a* помещает в это сообщение то подмножество описаний вершин, которое хранится в её памяти.

Конец рассылки описания дерева определяет корень, когда хранящееся в нём множество описаний вершин становится пустым.

Поскольку каждая вершина удаляет из множества описаний вершин своё описание, а при получении сообщения строится пересечение множеств, рано или поздно такое множество в некоторой вершине становится пустым. В этот момент времени все вершины получили предназначенную им информацию. Далее это пустое множество, двигаясь по графу, «опустошает» множества, хранящиеся в вершинах. Когда «опустошится» множество в корне, этап закончен, корень посылает вовне сообщение «*Готов к работе*».

4.5. Вычисление функции

Вычисление функции *F* начинается с получения корнем извне сообщения «Вопрос» с параметрами *H,G,E*. Функция *H* требуется только корню, а остальным вершинам вопрос доставляется как пара функций *(G,E)*.

Корень нумерует вопросы для того, чтобы при вычислении ответа на вопрос игнорировать все сообщения, относящиеся к вычислению ответов на предыдущие вопросы, поскольку они могут циркулировать по графу даже после окончания вычисления.

Виртуальное обратное дерево задаёт вложенное разбиение мультимножества $q(V)$, как описано в разделе 2 и подразделе 4.2.

Листовая вершина v с индексом (i,j), получив вопрос, сразу же вычисляет ответ $G(q(v))$ и должна передать его по виртуальной обратной дуге, т.е. в вершину с индексом $(i,j-1)$.

Внутренней вершине v дерева типа «сбалансированный веник» соответствует элемент $b=b_1 \cup b_2$ вложенного разбиения, где $b_1=q(D_v)$, где D_v – множество вершин, расположенных на дереве выше v, а $b_2=q(v)$. Вершина v сначала получает ответ $G(b_1)$ по входящей в неё виртуальной обратной дуге, то есть от вершины с индексом $(i,j+1)$. После этого вершина v вычисляет свой ответ $G(b) = E(G(b_1),G(b_2))$ и должна передать его по виртуальной выходящей обратной дуге, то есть вершине с индексом $(i,j-1)$.

Корню v_0 соответствует элемент $b=b_1 \cup ... \cup b_{w+1}$ вложенного разбиения, где w равно числу входящих в корень виртуальных обратных дуг, а $b_{w+1}=q(v_0)$. Корень сначала собирает ответы $G(b_1),...,G(b_w)$ по всем входящим в него виртуальным обратным дугам, то есть ответы от вершин с индексами $(1,1),...,(w,1)$. После этого корень вычисляет $G(b) = E_{w+1}(G(b_1),...,G(b_w),G(b_{w+1}))$. При этом сама функция E_{w+1} вычисляется итеративно с помощью функции E: для $j>1$ имеем $E_{j+1}(G_1,...,G_{j+1}) = E(E_j(G_1,...,G_j), G_{j+1})$. Дополнительно корень применяет функцию H, т.е. вычисляет $H(G(b))$, и посылает полученное значение вовне графа в сообщении «Ответ».

Как показано в подразделах 4.1 и 4.2, при вычислении используется сообщение, которое должно содержать всю распространяемую по графу информацию: вопрос (G,E), номер вопроса и все нужные ответы. Такое сообщение имеет тип *3*. При этом достаточно, чтобы сообщение содержало не более одного ответа от вершин одной ветви дерева. Если вершина получила два ответа от одной ветви i, т.е. от вершин с индексами (i,j_1) и (i,j_2), то выбирается ответ от вершины ближайшей к корню. Это вершина с минимальным номером на ветви, т.е. вершина с индексом $(i,\min\{j_1,j_2\})$. Сообщение типа *3* посылается по дугам графа при первой возможности (по сигналу О или П). В сообщении ответ (значение функции G) содержится вместе с индексом отправителя, т.е. вершины, вычислившей этот ответ. Эту пару (ответ, индекс отправителя) будем называть *индексированным ответом*.

5. Описание алгоритма

5.1. Память автомата и сообщения

1. Определения:
   - идентификатор дуги: (идентификатор начала дуги, номер дуги);
   - описание дуги: (идентификатор дуги, статус дуги);
   - индекс вершины: (номер ветви дерева, номер вершины на ветви);
   - описание вершины: (тип вершины: «лист» или «внутренняя», идентификатор вершины, индекс вершины);
   - индексированный ответ: (ответ как значение функции G, индекс вершины-отправителя).
2. Управляющие состояния автомата: *начальное*, *0*, *1*, *2*, *3*.
3. Память автомата (дополнительно к управляющему состоянию).
   3.1. Память автомата в начальном состоянии.
   Несущественно.
   3.2. Память автомата в состоянии *0*.
   - идентификатор вершины,
   - множество описаний дуг.

   3.3. Память автомата в состоянии *1*.
   - идентификатор вершины,
   - тип вершины: корень или не корень,
   - w (только для корня),
   - множество описаний дуг.

   3.4. Память автомата в состоянии *2*.
   - идентификатор вершины,
   - тип вершины: корень, лист, внутренняя,
   - w (только для корня),
   - индекс вершины (не для корня),
   - множество описаний вершин.

   3.5. Память автомата в состоянии *3*.
   - тип вершины: корень, лист, внутренняя,
   - w (только для корня),
   - H (только для корня),
   - индекс вершины (не для корня),
   - номер вопроса,
   - (G,E),
   - множество индексированных ответов.

4. Формат «внешних» сообщений (дополнительно к типу сообщения).
   4.1. Извне сообщение «Старт»: *w*.
   4.2. Вовне сообщение «Готов к работе»: без параметров.
   4.3. Извне сообщение «Вопрос»: описание функций *H,G,E*.
   4.4. Вовне сообщение «Ответ»: значение *H(G(q(V)))*.
   4.5. Вовне сообщение «Ошибка протокола»: без параметров.
5. Формат «внутренних» сообщений (дополнительно к типу сообщения).
   5.1. Формат сообщения типа *1*.
   - идентификатор дуги, по которой сообщение передаётся,
   - множество описаний дуг.

   5.2. Формат сообщения типа *2*.
   - множество описаний вершин.

   5.3. Формат сообщения типа *3*.
   - номер вопроса,
   - *(G,E)*,
   - множество индексированных ответов.

5.2. Работа автомата

1. Начальное состояние.

   Автомат вершины начинает работать в начальный момент времени с *начального состояния*. Автомат переходит в состояние *0*, определяет идентификатор вершины, в которой он находится, с помощью примитива «Дай идентификатор» и запоминает его в своей памяти. Далее автомат формирует в своей памяти пустое множество описаний дуг.
2. Состояние *0*.
   2.1. Сигнал П, параметры: номер дуги.

   Если описания дуги с этим номером ещё нет, то создаётся описание дуги (по правилу 1 из 4.3): идентификатор вершины, номер дуги, статус = *1*. В противном случае (по правилу 3 из 4.3) статус дуги меняется на *3*.

   2.2. Сигналы О и И невозможны в состоянии *0*.
   2.3. Сообщение «Старт».

   По предположению, это сообщение приходит извне в корень. В этот момент в корне могут быть описания только выходящих дуг со статусом *1* или *3*. Автомат переходит в состояние *1*, устанавливает тип вершины = корень, запоминает *w* и посылает по каждой (выходящей) дуге со статусом *1* сообщение типа *1* с множеством описаний дуг из корня.

   Альтернативное поведение в вырожденном случае: если в корне до получения сообщения «Старт» не появилось ни одной выходящей дуги. Отсюда следует, по предположению о начальных дугах в разделе 3, что нет начальных дуг и, следовательно, нет других вершин. Корень переходит в состояние 2, устанавливает тип вершины = корень, устанавливает *w=0*, формирует пустое множество описаний вершин и посылает вовне ответное сообщение «Готов к работе».

   2.4. Сообщение «Вопрос».

   По предположению, это сообщение извне может придти только в корень. При нормальной работе оно должно приходить только после того, как корень ответит на сообщение «Старт». Поэтому в состоянии *0* корень посылает ответное сообщение «Ошибка протокола».

   2.5. Сообщение типа *1*.

   Вершина в состоянии *0* может получить это сообщение только, если это не корневая вершина. Это сообщение приходит по входящей дуге из другой вершины. В этот момент времени в вершине могут быть описания только выходящих дуг со статусом *1* или *3*, а в принятом сообщении не могут быть описания этих дуг, поскольку автомат ещё не посылал ни одного сообщения по выходящим дугам. Автомат переходит в состояние *1*, устанавливает тип вершины = не корень и (по правилу 5 из 4.3) добавляет все описания дуг из сообщения во множество описаний дуг в вершине. Если статус дуги, по которой пришло сообщение, равен *1*, то (по правилу 4 из 4.3) он меняется на *3*. После этого автомат посылает по каждой выходящей дуге (дуге, идентификатор начала которой равен идентификатору текущей вершины) сообщение типа *1* с множеством описаний дуг из вершины.

   2.6. Сообщения типа *2* и *3* невозможны в состоянии *0*.
3. Состояние *1*.
   3.1. Сигнал О, параметры: номер дуги.

   Если описание дуги есть в вершине и статус дуги равен *1*, то он меняется на *2* по правилу 2 из 4.3. В любом случае по дуге посылается сообщение типа *1* с множеством описаний дуг из вершины.

   3.2. Сигнал П, параметры: номер дуги.

   Если описание дуги есть в вершине и статус дуги равен *1*, то он меняется на *3* по правилу 3 из 4.3. В любом случае по дуге посылается сообщение типа *1* с множеством описаний дуг из вершины.

3.3. Сигнал И, параметры: номер дуги.

Если описание дуги есть в вершине и статус дуги равен *1*, то он меняется на *3* по правилу 3 из 4.3.

3.4. Сообщение «Старт» или «Вопрос».

По предположению, такое сообщение извне может придти только в корень. При нормальной работе сообщение «Старт» не должно приходить повторно, а сообщение «Вопрос» должно приходить только после того, как корень ответит на сообщение «Старт». Поэтому в состоянии *1* корень посылает ответное сообщение «Ошибка протокола».

3.5. Сообщение типа *1*.

Автомат просматривает множество описаний дуг в сообщении. По правилу 5 из 4.3, если описания дуги не было в вершине, оно добавляется в вершину, а в противном случае статус дуги в вершине корректируется. Если статус дуги, по которой пришло сообщение, равен *1*, то он меняется на *3* (по правилу 4 из 4.3).

Автомат корня дополнительно проверяет статусы всех дуг, описания которых хранятся в его памяти. Если статусы всех дуг равны *3*, то автомат переходит в состояние 2, корректирует ширину дерева $w:=min\{w,n-1\}$ и строит виртуальное дерево, как описано в подразделе 4.2, т.е. создаёт множество описаний некорневых вершин. Если это множество пусто (для случая $n=1$ при наличии петель, появившихся до получения сообщения «Старт»), корень посылает вовне сообщение «Готов к работе».

3.6. Сообщение типа *2*.

В состоянии *1* автомат вершины может получить сообщение типа *2* только в том случае, если эта вершина – не корень. Автомат переходит в состояние *2*, переписывает в свою память всё множество описаний вершин из сообщения. В этом множестве должно быть и описание вершины *a*, которое определяется по идентификатору вершины *a*. Из этого описания запоминается индекс и тип вершины *a*, а само описание удаляется из множества.

3.7. Сообщение типа *3* невозможно в состоянии *1*.

4. Состояние *2*.

4.1. Сигнал О или П, параметры: номер дуги.

По дуге посылается сообщение типа *2* с множеством описаний вершин из вершины.

4.2. Сигнал И, параметры: номер дуги.

Ничего не делается.

4.3. Сообщение «Старт».

Аналогично 3.4. корень посылает ответное сообщение «Ошибка протокола».

4.4. Сообщение «Вопрос».

По предположению, такое сообщение извне может придти только в корень. При нормальной работе в состоянии *2* сообщение «Вопрос» должно приходить только после того, как корень ответит на предыдущее сообщение извне «Старт». Корень проверяет, пусто ли множество описаний вершин в его памяти. Если не пусто, то это означает, что ответ на сообщение «Старт» ещё не отправлен вовне, поэтому корень посылает ответное сообщение «Ошибка протокола». Если пусто, корень переходит в состояние *3*, запоминает из сообщения функции H, G и E, устанавливает номер вопроса = *1*, формирует пустое множество индексированных ответов. Если $w=0$, т.е. корень v_0 – единственная вершина, корень вычисляет значение $H(G(q(v_0)))$ и посылает вовне сообщение «Ответ» с этим значением.

4.5. Сообщение типа *1*.

В состоянии *2* это сообщение игнорируется: автомат ничего не делает.

4.6. Сообщение типа *2*.

Автомат строит пересечение множеств описаний вершин из сообщения и из своей памяти. Это пересечение сохраняется в памяти автомата. Если это пересечение пусто, а текущая вершина – корень, то корень посылает вовне сообщение «Готов к работе».

4.7. Сообщение типа *3*.

В состоянии 2 такое сообщение может получить только некорневая вершина v с индексом (i,j). Автомат переходит в состояние 3, переписывает в свою память из сообщения вопрос (G,E), номер вопроса (который должен быть равен *1*) и множество индексированных ответов. Если это листовая вершина, то автомат вычисляет значение $G(q(v))$ и записывает его вместе со своим индексом (i,j) как индексом отправителя во множество индексированных ответов. Заметим, что это будет первый индексированный ответ от вершин ветви i. Если это внутренняя вершина, то автомат проверяет, нет ли во множестве индексированных ответов в сообщении ответа $G(b_1)$ от вершины с индексом $(i,j+1)$. Если есть, то автомат вычисляет значение $E(G(b_1), G(q(v)))$ и записывает его вместе с индексом (i,j) как индексом отправителя во множество индексированных ответов вместо ответа $G(b_1)$ от вершины с индексом $(i,j+1)$.

5. Состояние *3*.

5.1. Сигнал О или П, параметры: номер дуги.

По дуге посылается сообщение типа *3* с множеством индексированных ответов из вершины.

5.2. Сигнал И, параметры: номер дуги.

Ничего не делается.

5.3. Сообщение «Старт».

По предположению, такое сообщение извне может придти только в корень. При нормальной работе сообщение «Старт» не должно приходить повторно. Поэтому в состоянии *3* корень посылает ответное сообщение «Ошибка протокола».

5.4. Сообщение «Вопрос».

По предположению, такое сообщение извне может придти только в корень. При нормальной работе сообщение «Вопрос» должно приходить только после того, как корень ответит на предыдущее сообщение извне «Вопрос». Корень проверяет, что во множестве индексированных ответов индексы отправителей пробегают всё множество индексов ближайших к корню вершин: $\{(i,1)|i\in[1..w]\}$.

Если это не так, то это означает, что корень ещё не ответил вовне на предыдущий «Вопрос». Поэтому корень посылает ответное сообщение «Ошибка протокола».

В противном случае корень начинает работу над поступившим вопросом. Запоминает из сообщения функции H, G и E, увеличивает номер вопроса на *1*, формирует пустое множество индексированных ответов. Если $w=0$, т.е. корень v_0 – единственная вершина, корень вычисляет значение $H(G(q(v_0)))$ и посылает вовне сообщение «Ответ» с этим значением.

5.5. Сообщение типа *1*, типа *2*.

В состоянии *3* такое сообщение игнорируется: автомат ничего не делает.

5.6. Сообщение типа *3*.

Пусть текущая вершина v имеет индекс (i,j). Автомат сравнивает номер вопроса a в сообщении и номер вопроса b в своей памяти.

5.6.1. Если $a<b$, то это означает, что ответ на вопрос с номером a уже отправлен корнем вовне, а принятое сообщение – «остаточное» от вычисления ответа на более ранний вопрос сообщение игнорируется: автомат ничего не делает.

5.6.2. Если $a>b$, то это означает, что ответ на вопрос с номером b уже отправлен корнем вовне, а сообщение содержит новый вопрос. Заметим, что в этом случае $a=b+1$, т.е. сообщение содержит следующий вопрос, а текущая вершина – не корень. Автомат приступает к работе над новым вопросом: переписывает в свою

память из сообщения вопрос (G,E), номер вопроса и множество индексированных ответов.

Если это листовая вершина, то автомат вычисляет значение $G(q(v))$ и записывает его вместе со своим индексом (i,j) как индексом отправителя во множество индексированных ответов. Заметим, что это будет первый индексированный ответ (на вопрос с номером a) от вершин ветви i.

Если это внутренняя вершина, то автомат проверяет, нет ли во множестве индексированных ответов в сообщении ответа $G(b_1)$ от вершины с индексом $(i,j+1)$. Если есть, то автомат вычисляет значение $E(G(b_1), G(q(v)))$ и записывает его вместе с индексом (i,j) как индексом отправителя во множество индексированных ответов вместо ответа $G(b_1)$ от вершины с индексом $(i,j+1)$.

5.6.3. Если $a=b$, то это означает, что продолжается работа над текущим вопросом с номером $a=b$. Автомат сравнивает индексированные ответы из сообщения и из своей памяти. Если в сообщении есть индексированный ответ $(G_1,(i_1,j_1))$, а в вершине нет ответа от вершины на той же ветви i_1, то индексированный ответ $(G_1,(i_1,j_1))$ добавляется в множество индексированных ответов в вершине. Если в вершине есть индексированный ответ $(G_2,(i_1,j_2))$, то автомат сравнивает расположение на ветви отправителей ответов из сообщения и из вершины. Если $j_1 \geq j_2$, то ответ из сообщения игнорируется. Если $j_1 < j_2$, то ответ $(G_1,(i_1,j_1))$ из сообщения замещает собой ответ $(G_2,(i_1,j_2))$ в вершине.

Если текущая вершина листовая, то автомат больше ничего не делает.

Если текущая вершина внутренняя, то автомат проверяет, нет ли во множестве индексированных ответов в вершине ответа $G(b_1)$ от вершины с индексом $(i,j+1)$. Если есть, то автомат вычисляет значение $E(G(b_1), G(q(v)))$ и записывает его вместе с индексом (i,j) как индексом отправителя в множество индексированных ответов вместо ответа $G(b_1)$ от вершины с индексом $(i,j+1)$.

Если текущая вершина корень, то автомат проверяет, что во множестве индексированных ответов индексы отправителей пробегают всё множество индексов ближайших к корню вершин: $\{(i,1)|i\in[1..w]\}$. Если это не так, то автомат больше ничего не делает. В противном случае автомат вычисляет результирующее значение функции $H(E_{w+1}(G_1,...,G_w,G_{w+1}))$, где G_i ответ от вершины с индексом $(i,1)$ для $i=1..w$, $G_{w+1}=G(q(v_0))$. При этом функция E_{w+1} вычисляется итеративно с помощью функции E: для $j>1$ имеем $E_{j+1}(G_1,...,G_{j+1}) = E(E_j(G_1,...,G_j), G_{j+1})$.

Вычисленное значение функции автомат отправляет вовне в сообщении «Ответ».

6. Утверждения и оценки

6.1. Размеры памяти автомата и сообщения

Будем считать, что идентификатор вершины занимает x бит памяти, номер дуги ограничен сверху числом $s \le m$, описание каждой из функций H,G или E занимает не более y бит памяти, значение функции занимает не более z бит памяти, номер вопроса не превышает N. Тогда следующие величины имеют ограничения по размеру: идентификатор дуги – $x+logs$, описание дуги – $x+logs+2$, номер ветви дерева – $logw$, номер вершины на ветви – $logh=log\lceil(n-1)/w\rceil$, индекс – $logw+logh=logn$, описание вершины – $1+x+logn$, индексированный ответ – $z+logn$, управляющее состояние автомата – 3, множество описаний дуг – $m(x+logs+2)$, тип вершины – 2, множество описаний вершин – $n(1+x+logn)$, множество индексированных ответов – $w(z+logn)$, тип сообщения – 3.

Отсюда следуют следующие ограничения на размер памяти автомата в различных состояниях (учитывая, что $max\{logw,logn\}=logn$):

начальное состояние – $3 = \boldsymbol{O}(1)$,

состояние 0 – $3+x+m(x+logs+2) = \boldsymbol{O}(mx+mlogs)$,

состояние 1 – $3+x+3+logw+m(x+logs+2) = \boldsymbol{O}(logw+mx+mlogs)$,

состояние 2 – $3+x+3+logn+n(1+x+logn) = \boldsymbol{O}(nx+nlogn)$,

состояние 3 – $3+3+logn+3y+logN+w(z+logn) = \boldsymbol{O}(y+logN+wz+wlogn)$.

Также получаются следующие ограничения на размер сообщения (учитывая, что $max\{logw,logn\}=logn$):

Старт – $3+logw = \boldsymbol{O}(logw)$,

Готов к работе – $3 = \boldsymbol{O}(1)$,

Вопрос – $3+3y = \boldsymbol{O}(y)$,

Ответ – $3+z = \boldsymbol{O}(z)$,

Ошибка протокола – $3 = \boldsymbol{O}(1)$,

Тип 1 – $3+x+logs+m(x+logs+2) = \boldsymbol{O}(mx+mlogs)$,

Тип 2 – $3+n(1+x+logn) = \boldsymbol{O}(nx+nlogn)$,

Тип 3 – $3+logN+2y+w(z+logn) = \boldsymbol{O}(logN+y+wz+wlogn)$.

6.2. Время работы алгоритма

Лемма 2. После получения корнем извне сообщения «Старт» за время $\boldsymbol{O}(n)$ каждая вершина получит сообщение.

Доказательство. Для корня утверждение очевидно. Пусть $n>1$. По предположению о начальных дугах в разделе 3, начальная дуга существует в

начальный момент времени и не меняется до тех пор, пока по ней не пройдёт первое сообщение, и каждая вершина достижима из корня по начальным дугам. Поскольку временем срабатывания автомата мы пренебрегаем, автомат корня сформирует сообщение типа 1 через 0 тактов после получения сообщения «Старт». Поскольку длина любого пути в графе, в том числе пути по начальным дугам, не превосходит $n-1$, от корня до каждой некорневой вершины дойдёт сообщение типа 1 за время не более $n-1$.

Лемма доказана.

Регистрируемой дугой назовём такую дугу $a \to b$, которая появляется первый раз до получения вершиной a первого сообщения.

Лемма 3. В любой момент времени, когда вершина находится в состоянии 0 или 1, множество описаний дуг в этой вершине содержит описания только регистрируемых дуг. Если в этом множестве есть описание дуги $a \to b$, то в нём есть описание всех регистрируемых дуг, выходящих из a.

Доказательство. Описание дуги $a \to b$ создаётся только до получения вершиной a первого сообщения, поэтому создаются описания только регистрируемых дуг, и только такие описания могут оказаться в сообщениях и в вершинах. Поскольку все регистрируемые дуги, выходящие из вершины a, появляются до получения вершиной a первого сообщения, описания всех этих дуг совместно оказываются в каком-либо сообщении или в какой-нибудь вершине.

Лемма доказана.

Лемма 4. Через время $\boldsymbol{O}(n)$ после получения вершиной a первого сообщения все регистрируемые дуги, выходящие из этой вершины, будут в корне иметь описание со статусом 3.

Доказательство. Рассмотрим регистрируемую дугу $a \to b$. При получении вершиной a первого сообщения по дуге $a \to b$ посылается сообщение, содержащее описание этой дуги со статусом 1. Поскольку время передачи сообщения по дуге не превосходит 1 такт, не позже чем через 1 такт это сообщение либо будет доставлено в вершину b, либо пропадёт. В первом случае в вершине b дуга $a \to b$ получит статус 3, после чего по лемме 1 через время не более $3(n-1)$ в корне эта дуга тоже будет иметь статус 3. Во втором случае в вершине a дуга $a \to b$ получит статус 3, после чего по лемме 1 через время не более $3(n-1)$ в корне эта дуга тоже будет иметь статус 3. Таким образом, после получения вершиной a первого сообщения через время не более чем $1+3(n-1)=\boldsymbol{O}(n)$ в корне все регистрируемые дуги, выходящие из этой вершины, будут иметь статус 3.

Лемма доказана.

Лемма 5. Если начальная дуга $a \to b$ имеет в корне статус 3, то корень «знает» о конце дуги, вершине b, т.е. в корне есть множество описаний всех регистрируемых дуг, выходящих из b, и это множество не пусто.

Доказательство. Поскольку начальная дуга $a \to b$ не меняется до тех пор, пока по ней не пройдёт первое сообщение, она получает статус *3* в вершине *b*, когда это сообщение дойдёт до вершины *b*. Одновременно по лемме 3 вершина *b* добавит во множество описаний все регистрируемые дуги, выходящие из неё. В этот же момент времени в вершине *a* дуга получает статус *2*, следовательно, в вершине *a* эта дуга не получит статус *3* по сигналам И или П. Следовательно, любое множество описаний дуг (в вершине или в сообщении), содержащее описание этой дуги со статусом *3*, содержит и описание всех регистрируемых дуг, выходящих из вершины *b*. Покажем, что из вершины *b* выходит хотя бы одна регистрируемая дуга. Действительно, по предположению о долгоживущих дугах (в каждый момент времени они образуют сильно связный сураф), хотя бы одна долгоживущая дуга должна существовать в момент получения вершиной *b* первого сообщения. А тогда эта долгоживущая дуга регистрируемая.

Лемма доказана.

Теорема 1. После получения корнем извне сообщения «Старт» за время $\boldsymbol{O}(n)$ корень перейдёт в состояние *2*, причём во множестве описаний вершин в корне будут все вершины графа.

Доказательство. В вырожденном случае, когда из корня не выходит ни одна регистрируемая дуга, корень переходит в состояние *2* в тот же момент времени, когда он получает сообщение «Старт». В этом случае по предположению о начальных дугах $n=0$.

Пусть $n>1$. По лемме 2 за время $\boldsymbol{O}(n)$ все некорневые вершины получат первое сообщение. По описанию алгоритма это может быть только сообщение типа *1*. По лемме 4 после этого через время $\boldsymbol{O}(n)$ в корне будет описание всех регистрируемых дуг со статусом *3*. А в этом случае автомат корня переходит в состояние 2. Теперь покажем, что во множестве описаний вершин в корне будут все вершины графа. Для этого достаточно, чтобы при переходе из состояния 1 в состояние 2 в корне каждая вершина встречалась как начало описания некоторой дуги. По предположению о начальных дугах в каждую вершину *b* существует путь от корня по начальным дугам. Поскольку все начальные дуги регистрируемые, в корне будет описание некоторой начальной дуги $a \to b$. А тогда по лемме 5 корень «знает» о вершине *b*. Итак, за время $\boldsymbol{O}(n)$ корень перейдёт в состояние *2*, причём во множестве описаний вершин в корне будут все вершины графа.

Теорема доказана.

Теорема 2. Через время $\boldsymbol{O}(n)$ после перехода корня в состояние *2* корень отправит вовне сообщение «Готов к работе».

Доказательство. Когда корень переходит в состояние 2, по теореме 1 корень «знает» о всех вершинах. Он формирует описание виртуального обратного дерева как множество описаний вершин и инициирует рассылку сообщения типа *2*. По лемме 1 все вершины получат это сообщение первый раз через

время $\boldsymbol{O}(n)$. Каждая вершина, получая сообщение типа *2* первый раз, удаляет из него своё описание. Кроме того, при получении вершиной каждого следующего сообщения типа 2 строится пересечение множеств описаний вершин в сообщении и в вершине. Поэтому по лемме 1 через время $\boldsymbol{O}(n)$ в корне получится пустое множество описаний вершин. Тогда корень отправит вовне сообщение «Готов к работе». Общее время $\boldsymbol{O}(n)$.

Теорема доказана.

Теорема 3. Через время $\boldsymbol{O}(n^2/w)$ после получения корнем сообщения «Вопрос» (без нарушения протокола) корень посылает вовне сообщение «Ответ», в котором находится правильно вычисленное значение указанной в вопросе функции.

Доказательство. Правильность вычисления функции непосредственно следует из её представления через минимальное агрегатное расширение (раздел 2), вложенное разбиение мультимножества $q(V)$ на виртуальном обратном дереве (подраздел 4.2) и реализацию этого дерева в алгоритме.

По лемме 1 за время $\boldsymbol{O}(n)$ каждая вершина, в том числе каждая листовая вершина, получит сообщение типа 3 с текущим вопросом. Листовая вершина сразу вычисляет требуемый ответ. Далее каждая вершина с индексом (i,j) вычисляет требуемый ответ после получения по виртуальной дуге сообщения, в котором содержится ответ от предыдущей на ветви вершины с индексом $(i,j+1)$. По лемме 1 время передачи сообщения по виртуальной дуге равно $\boldsymbol{O}(n)$. Следовательно, корень получит ответы от всех вершин с индексами $(1,1)..(w,1)$ за время $\boldsymbol{O}(hn)$. Корень вычисляет свой ответ и посылает его вовне в сообщении «Ответ». Таким образом, общее время вычисления равно $\boldsymbol{O}(hn)=\boldsymbol{O}(n^2/w)$, поскольку $h=](n-1)/w[$.

Теорема доказана.

Список литературы

[1]. И. Бурдонов, А. Косачев, В. Кулямин. Параллельные вычисления на графе. Программирование, 2015, №1, с. 3-20.

[2]. И. Бурдонов, А. Косачев. Мониторинг динамически меняющегося графа. Труды Института системного программирования РАН Том 27. Выпуск 1. 2015 г. Стр. 69-96. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). DOI: 10.15514/ISPRAS-2015-27(1)-5

[3]. Steven S. Skiena. The Algorithm Design Manual. Springer-Verlag, New York, 1997.

[4]. И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин. Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай. Программирование, 2003 г., №5, с. 59-69.

[5]. И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин. Неизбыточные алгоритмы обхода ориентированных графов. Недетерминированный случай. Программирование, 2004 г., №1, с. 2-17.

[6]. M.O. Rabin. Maze Threading Automata. An unpublished lecture presented at MIT and UC. Berkeley, 1967.

[7]. И.Б. Бурдонов. Обход неизвестного ориентированного графа конечным роботом. Программирование, 2004 г., № 4, с. 11-34.
[8]. И.Б. Бурдонов. Проблема отката по дереву при обходе неизвестного ориентированного графа конечным роботом. Программирование, 2004 г., № 6, с. 6-29.
[9]. Бурдонов И.Б., Косачев А.С. Обход неизвестного графа коллективом автоматов. Труды ИСП РАН. Том 26-2, 2014 г., стр. 43-86.
[10]. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Исследование графа набором автоматов. Программирование, 2015, №6, (в печати).
[11]. Кушнеренко А.Г., Лебедев Г.В. "Программирование для математиков", Наука, Главная редакция физико-математической литературы, Москва, 1988.

Parallel Calculations on Dynamic Graph

Igor Burdonov <igor@ispras.ru>
Alexander Kossatchev <kos@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. The problem of parallel computation of the value of a function of multiset of values recorded at the vertices of a directed strongly connected graph is considered. Computation is performed by automata that are located at the graph vertices. The automaton has local information on the graph: it "knows" only about arcs outgoing from the vertex it resides in, but it "does not know" where (to which vertices) those arcs go. The automata exchange messages with each other that are transmitted along the graph arcs that play role of message transfer channels. Computation is initiated by a message coming from outside to the automaton located at the initial vertex of the graph. At the end of work, this automaton sends outside the calculated function value. Two algorithms are proposed to solve this problem. The first algorithm carries out an analysis of the graph. Its purpose is to mark the graph by a change of the states of the automata at the vertices. Such marking is used by the second algorithm, which calculates the function value. This calculation is based on a pulsation algorithm: first, request messages are distributed from the automaton of the initial vertex over the graph, which should reach each vertex, and then response messages are sent from each vertex back to the initial vertex. In fact, the pulsation algorithm calculates aggregate functions for which the value of a function of a union of multisets is calculated by the values of the function of these multisets. However, it is shown that any function F(x) has an aggregate extension; that is, an aggregate function can be calculated as H(G(x)), where G is an aggregate function. Note that the marking of a graph does not depend on a function that will be calculated. This means that the marking of a graph is carried out once; after that, it can be reused for calculating various functions. Since the automata located in different vertices of the graph work in parallel, both graph marking and function calculation are performed in parallel. It is the first feature of this work. The second feature is that calculations are performed on a dynamically changing graph: its arcs can disappear, reappear or change their

target vertices. The constraints put on the graph changes are as minimal as it allows solving this problem in limited time. Estimate of working time is given for both algorithms.

Keywords: directed graphs; graph exploration, communicating automata, parallel processing, aggregate functions, dynamic graphs.

DOI: 10.15514/ISPRAS-2015-27(2)-12

For citation: Burdonov Igor, Kossatchev Alexander. Parallel Calculations on Dynamic Graph. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 2, 2015, pp. 189-220 (in Russian). DOI: 10.15514/ISPRAS-2015-27(2)-12.

References

[1]. I. B. Burdonov, A. S. Kossatchev, V. V. Kuliamin. Parallel computations on graphs. Programming and computer Software, 41(1): 1-13, 2015.
[2]. I. B. Burdonov, A. S. Kossatchev. Monitoring of dynamically changed graph. Proceedings of the Institute for System Programming Volume 27 (Issue 1). 2015. pp. 69-96. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). DOI: 10.15514/ISPRAS-2015-27(1)-5 (in Russian).
[3]. Steven S. Skiena. The Algorithm Design Manual. Springer-Verlag, New York, 1997.
[4]. I. B. Burdonov, A. S. Kossatchev, V. V. Kuliamin. Irredundant Algorithms for Traversing Directed Graphs: The Deterministic Case. Programming and Computer Software, 29(5):245-258, 2003.
[5]. I. B. Burdonov, A. S. Kossatchev, V. V. Kuliamin. Irredundant Algorithms for Traversing Directed Graphs: The Nondeterministic Case. Programming and Computer Software, 30(1):2-17, 2004.
[6]. M.O. Rabin. Maze Threading Automata. An unpublished lecture presented at MIT and UC. Berkeley, 1967.
[7]. I. B. Burdonov. Traversal of an unknown directed graph by a finite automaton. Programming and Computer Software, 30(4): 11-34, 2004.
[8]. I. B. Burdonov. Backtracking on a tree in traversal of an unknown directed graph by a finite automaton. Programming and Computer Software, 30(6): 6-29, 2004.
[9]. I. B. Burdonov, A. S. Kossatchev. Obkhod neizvestnogo grafa kollektivom avtomatov [Unknown graph traversing by learning by automata group] Trudy ISP RAN [The proceeding of ISP RAS], Vol. 26-2, 2014, pp. 43-86. (in Russian)
[10]. I. B. Burdonov, A. S. Kossatchev, V. V. Kuliamin. Graph Learning by automata group. Programming and computer Software, 41(6), 2015 On printing.
[11]. Kushnerenko, A.G., and Lebedev, G.V., Programmirovanie dlya matematikov (Programming for Mathematicians), Moscow: Nauka, 1988