ПРОГРАММИРОВАНИЕ, 2015, № 1, с. 3–20

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

УДК 681.3.06

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ НА ГРАФЕ

© 2015 г. И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин

Институт системного программирования РАН

109004 Москва, ул. А. Солженицына, 25

E-mail: igor@ispras.ru, kos@ispras.ru, kuliamin@ispras.ru

Поступила в редакцию 07.09.2014

Рассматривается задача параллельного вычисления значения функции от мультимножества значений, записанных в вершинах ориентированного графа. Вычисление выполняется автоматами, находящимися в вершинах графа и обменивающихся между собой сообщениями, передаваемыми по дугам графа (в направлении их ориентации). Предполагается, что ёмкость дуги, то есть число одновременно передаваемых по ней сообщений, ограничена. 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, основанное на его обходе. Его цель – разметить граф с помощью изменения состояний автоматов в вершинах. Строятся прямой и обратный остовы графа. Прямой остов ориентирован от корня, которым является начальная вершина графа. Обратный остов ориентирован к тому же корню. Кроме того, в каждой вершине устанавливается значение счётчика входящих дуг обратного остова. 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 $h(f'(x))$, где f' — 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.

1. ВВЕДЕНИЕ

Исследование ориентированных графов является корневой задачей 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 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. [1, 2, 3].

В 1976 г. М.О. Рабин поставил задачу обхода ориентированного графа конечным автоматом [4]. 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 [5], 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 $1 \leq log(log \dots (n) \dots) < 2$ [6]. 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 [7] и [8] 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 данной работе мы рассматриваем 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 мы будем называть *ёмкостью* дуги и обозначать k.

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 данной работе мы предлагаем два алгоритма. Первый алгоритм выполняет первичный обход графа с помощью пересылаемых сообщений, когда в начальный момент времени все автоматы находятся в начальном состоянии. После завершения обхода автоматы оказываются в таких конечных состояниях, которые позволяют выполнять параллельное вычисление требуемой функции с помощью второго алгоритма. Обход графа выполняется за время порядка $n/k + D$, а вычисление функции – за время порядка D.

Второй алгоритм – это *алгоритм пульсации,* основанный на том, что сначала от автомата выделенной 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.* 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.

2. ОПИСАНИЕ АЛГОРИТМА ОБХОДА

Там, где это не приведёт к коллизиям, мы для краткости будем говорить "вершина", имея в виду "автомат вершины" или "память автомата вершины". Посылая сообщение, автомат в вершине должен указывать дугу, выходящую из вершины, по которой сообщение посылается. Мы будем считать, что дуги, выходящие из вершины, перенумерованы, начиная с 1, и автомат указывает номер дуги.

Пусть в графе число дуг равно m, число вершин равно n, длина максимального пути (маршрута без самопересечений) равна D, число дуг, выходящих из вершины, не превышает s_0, а число дуг, входящих в вершину, не превышает s_1. Очевидно, $m \le ns_0$. Мы предлагаем алгоритм обхода графа со следующими характеристиками:

- память автомата вершины $\boldsymbol{O}(nDlogs_0)$,
- размер сообщения $\boldsymbol{O}(Dlogs_0)$,
- ёмкость дуги k,
- время работы алгоритма $\boldsymbol{O}(n/k + D)$.

Замечание 2.1: Очевидно, что посылка по дуге одновременно $k' \le k$ сообщений размером z бит эквивалентна посылке по дуге одного сообщения размером $k'z$ бит, или, в общем случае, посылке k_1 сообщений размером k_2z, где $k_1k_2 = k$. Тем самым оценка времени работы алгоритма при размере сообщения z бит и ёмкости дуги k сообщений совпадает с оценкой времени работы алгоритма при размере сообщения k_1z бит и ёмкости дуги k_2 сообщений, где $k_1k_2 = k$. Поэтому наш алгоритм после тривиальной модификации имеет следующие характеристики:

- память автомата вершины $\boldsymbol{O}(nDlogs_0)$,
- размер сообщения $\boldsymbol{O}(k_2Dlogs_0)$,
- ёмкость дуги k_1,
- время работы алгоритма $\boldsymbol{O}(n/k_1k_2 + D)$.

Для оценки времени работы алгоритма мы будем предполагать, что временем срабатывания автомата в вершине можно пренебречь, а время передачи сообщения по дуге ограничено сверху 1 тактом.

Одни сообщения посылаются из вершины сразу по всем выходящим дугам, другие – только по части выходящих дуг, а третьи – только по одной выходящей дуге. Тем не менее для простоты мы будем считать, что сообщение ожидает в вершине освобождения сразу всех выходящих дуг. Таким образом, для автомата вершины не нужен сигнал об освобождении данной выходящей дуги, а достаточно общего сигнала об освобождении всех выходящих дуг. Также будем считать, что автомат в общем случае посылает по дуге сразу несколько сообщений, но не более k – ёмкости дуги.

В процессе работы алгоритма строятся остовы графа: *прямой остов,* ориентированный от корня, и *обратный остов,* ориентированный к корню. Дуги, выходящие из вершины: *прямые дуги* – дуги прямого остова, *хорды* – остальные выходящие дуги, *обратная дуга* – дуга обратного остова (может быть как прямой дугой, так и хордой). *Прямой путь* – путь из прямых дуг от корня. *Обратный путь* – путь из обратных дуг до корня. *Вектором маршрута* будем называть список номеров дуг маршрута в графе. В алгоритме будут использоваться только вектора путей длины не более D и контуров (циклов, в которых вершина не встречается дважды) длины не более $D + 1$. *Вектор вершины* – это вектор прямого пути до этой вершины. Корень имеет пустой вектор ε. Размер вектора пути или контура $\boldsymbol{O}(Dlogs_0)$. Размер сообщения в описываемом ниже алгоритме равен $\boldsymbol{O}(1)$ векторов, т.е. $\boldsymbol{O}(Dlogs_0)$ бит.

Наше описание алгоритма мы разобьём на четыре части. Первая часть строит обратный остов, вторая часть – это "стопор", определяющий конец построения обратного остова, третья часть предназначена для разметки дуг, выходящих из вершины, на прямые и хорды, четвёртая часть устанавливает в вершинах счётчики входящих обратных дуг. Такая разметка графа (через память автоматов вершин) используется в дальнейшем алгоритмом пульсации.

2.1. *Первая часть алгоритма*

Используются четыре типа сообщения: ***Старт***, ***Поиск корня***, ***Прямое*** и ***Обратное***. Сообщение может быть либо создано в вершине, либо придти в вершину по входящей дуге. В обоих случаях мы будем для краткости говорить "вершина получает сообщение".

Сообщение ***Старт*** создаётся корнем в начале работы (при получении извне сообщения ***Старт***, которое инициирует работу алгоритма) и направляется всем вершинам. Его цель – сообщить каждой вершине её вектор и инициировать рассылку сообщений ***Поиск корня***. Каждая вершина (начиная с корня), получив сообщение ***Старт*** первый раз, рассылает его по всем выходящим дугам, а повторные сообщения ***Старт*** игнорирует. Сообщение содержит *вектор маршрута*, который оно проходит. Вершина (в том числе корень), посылая сообщение по выходящей дуге с номером i, добавляет i в конец *вектора маршрута*. Получая сообщение первый раз, вершина запоминает вектор маршрута (очевидно, это будет путь) из сообщения как свой *вектор вершины*; вектор корня пустой.

Сообщение ***Поиск корня*** создаётся вершиной, когда она получает первый раз сообщение ***Старт***. Эту вершину будем называть инициатором сообщения ***Поиск корня***. Цель сообщения ***Поиск корня*** – пройти некоторый путь от инициатора до корня и сообщить корню вектор этого пути. Корень в ответ посылает сообщение ***Прямое***, которое предназначено инициатору. Оба эти сообщения содержат как параметр *вектор инициатора*. В каждой вершине хранится *список векторов инициаторов* из тех сообщений ***Поиск корня*** или ***Прямое***, которые появлялись в этой вершине. Вначале список пустой. Как только вершина первый раз получает сообщение ***Поиск корня*** или ***Прямое*** с данным *вектором инициатора*, она помещает его в список. Собственный *вектор вершины* можно хранить как первый элемент списка; этот элемент будет пустым, если вершина ещё не имеет собственного вектора, т.е. не получала сообщения ***Старт***.

Сообщение ***Поиск корня*** рассылается "веером" таким образом, чтобы оно прошло по каждой дуге ровно один раз. Для этого *вектор инициатора* в сообщении сравнивается с векторами из *списка векторов инициаторов*. Если сообщение первое, оно рассылается по всем выходящим дугам, а *вектор инициатора* из сообщения добавляется в *список векторов инициаторов*. Повторные сообщения игнорируются. Кроме *вектора инициатора*, сообщение ***Поиск корня*** содержит *вектор маршрута*, который оно проходит. Когда корень первый раз получает сообщение ***Поиск корня*** с данным *вектором инициатора*, он создаёт сообщение ***Прямое*** с теми же *вектором инициатора* и *вектором маршрута*. Заметим, что здесь *вектор инициатора* – это вектор прямого пути от корня до инициатора, а *вектор маршрута* – это вектор некоторого пути от инициатора до корня.

Сообщение ***Прямое*** создаётся корнем, когда он первый раз получает сообщение ***Поиск корня*** с данным *вектором инициатора*. Оно направляется по прямому пути до инициатора. Его цель – сообщить инициатору *вектор пути* от инициатора до корня. Когда вершина получает сообщение ***Прямое*** с данным *вектором инициатора* x, она сравнивает его с собственным *вектором вершины* y. Поскольку сообщение ***Прямое*** движется по прямому пути до инициатора, $y \le x$. Если $y < x$, то для некоторого номера дуги i имеет место $y \cdot i \le x$, и сообщение без изменения посылается по выходящей дуге i. Если $y = x$, т.е. вершина является инициатором для этого сообщения, то вершина создаёт сообщение ***Обратное***, которое содержит тот же *вектор пути* от инициатора до корня. Повторим, что как только вершина первый раз получает сообщение ***Прямое*** с данным *вектором инициатора*, она помещает его в список для того, чтобы последующие сообщения ***Поиск корня*** с тем же инициатором игнорировались.

Сообщение ***Обратное*** создаётся вершиной, когда она получает адресованное ей сообщение ***Прямое***. Его цель – разметить обратный путь от этой вершины до корня, задаваемый *вектором пути* из сообщения ***Прямое***. Когда вершина (не корень) получает (создаёт или принимает по входящей дуге) сообщение ***Обратное*** с *вектором пути* вида $i \cdot x$, дуга с номером i отмечается как обратная, и по ней посылается сообщение ***Обратное*** с *вектором пути* x. Корень игнорирует сообщение ***Обратное*** (которое, очевидно, приходит в корень с пустым *вектором пути*).

Сообщения ***Обратное*** "суммируются" в вершине в ожидании освобождения выходящих дуг. Более точно: если в вершине есть сообщение ***Обратное***, ожидающее освобождения дуг, то вновь поступающие сообщения этого типа игнорируются.

Утверждение 2.1: Первая часть алгоритма строит обратный остов сильно связного графа.

Доказательство:

По описанию алгоритма в каждой вершине, кроме корня, появляется не более одной обратной дуги. Каждый раз, когда некоторая выходящая из вершины дуга объявляется обратной, по ней посылается сообщение, которое гарантированно доходит до корня, быть может, по дороге "суммируясь" с ждущими сообщениями ***Обратное***. Поэтому если в вершине появилась обратная дуга, то появится и путь по обратным дугам из этой вершины до корня.

Число всех пересылок сообщений по дугам конечно. Действительно, сообщение ***Старт*** по каждой дуге проходит ровно один раз, поэтому число пересылок по дугам таких сообщений равно числу дуг m. Сообщение ***Поиск корня*** с данным вектором инициатора проходит по каждой дуге тоже ровно один раз, а число различных инициаторов равно числу вершин, отличных от корня, т.е. $n-1$, поэтому число пересылок по дугам сообщений ***Поиск корня*** не превосходит $(n-1)m$. Сообщения ***Прямое*** и ***Обратное*** создаются по одному для каждого инициатора, и каждое такое сообщение проходит путь длины не более D, поэтому суммарное число пересылок по дугам таких сообщений не более $2(n-1)D$. Таким образом, общее число всех пересылок сообщений по дугам конечно. А тогда, поскольку по каждой дуге сообщение пересылается за конечное время (не более такта), через конечное время на графе не будет сообщений этих четырёх типов. Покажем, что в этот момент времени обратный остов построен.

Для этого достаточно показать, что в каждой вершине, кроме корня, появилась обратная дуга. Допустим, это не так: найдётся вершина a, отличная от корня, в которой не появилась обратная дуга. Поскольку граф сильно связный, вершина a достижима из корня. Следовательно, вершина a получала сообщение ***Старт***, после чего из вершины a "веером" рассылалось сообщение ***Поиск корня***. Поскольку граф сильно связный, это сообщение дойдёт до корня, после чего корень пошлёт по прямому пути в вершину a сообщение ***Прямое***. Когда это сообщение придёт в вершину a, в ней появится обратная дуга, если она не появится раньше. Мы пришли к противоречию, следовательно, наше допущение было не верно.

Утверждение доказано.

2.2. Вторая часть алгоритма

Задача этой части – определить конец построения обратного остова. Идея основана на подсчёте в корне числа дуг графа таким образом, что для каждой дуги $a \to b$ корень сначала получает сообщение от начала дуги a, в котором для дуги $a \to b$ имеется "+1", а заведомо позже – сообщение от конца дуги b, в котором для дуги $a \to b$ имеется "−1". Кроме того, это последнее сообщение от конца дуги b приходит в корень заведомо позже сообщения, посылаемого из b и содержащего "+1" для каждой дуги, выходящей из b. Для этого модифицируется сообщение ***Поиск корня*** и используются два дополнительных сообщения: ***Финиш*** и ***Минус***.

При создании сообщения ***Поиск корня*** инициатор добавляет в сообщение ещё один параметр: *число дуг*, выходящих из инициатора. Будем говорить, что для каждой дуги, выходящей из инициатора, в сообщении ***Поиск корня*** имеется "+1". Корень ведёт *счётчик дуг*, который вначале устанавливается равным числу дуг, выходящих из корня. При получении (не игнорируемого) сообщения ***Поиск корня*** корень прибавляет *число дуг* из сообщения к этому *счётчику дуг*.

Сообщение ***Финиш*** создаётся корнем в начале работы (при получении извне сообщения Старт) и посылается по всем дугам, выходящим из корня. Любая другая вершина (не корень) создаёт сообщение ***Финиш*** при получении сообщения ***Прямое*** и посылает сообщение ***Финиш*** по всем выходящим дугам. Цель сообщения ***Финиш*** – инициировать в конце дуги, по которой оно посылается, создание сообщения ***Минус***.

До тех пор, пока в вершине, отличной от корня, нет обратной дуги, она ведёт *счётчик числа входящих дуг* как числа полученных ею сообщений ***Финиш***. При появлении обратной дуги

вершина посылает по обратному пути сообщение ***Минус*** с параметром *число дуг,* равным текущему значению *счётчика числа входящих дуг*, если оно больше нуля. Будем говорить, что для каждой из этих дуг в сообщении ***Минус*** имеется "-1". После появления в вершине обратной дуги каждое получение сообщения ***Финиш*** инициирует посылку из вершины по обратному пути сообщения ***Минус*** с параметром *число дуг,* равным 1. Будем говорить, что для этой дуги в сообщении ***Минус*** имеется "-1". Сообщение ***Минус*** передаётся без изменения вершинами по обратному пути до корня. Корень, получив сообщение ***Минус***, вычитает из своего *счётчика дуг* параметр *число дуг* из сообщения ***Минус***.

Сообщения ***Минус*** суммируются в вершине в ожидании освобождения дуг. Если в вершине уже есть сообщение ***Минус***, ожидающее освобождения дуг, и приходит новое сообщение ***Минус***, то сохраняется одно сообщение ***Минус***, в котором параметр *число дуг* равен сумме соответствующих параметров обоих этих сообщений.

Утверждение 2.2: Через конечное время *счётчик дуг* в корне обнулится. В этот момент времени обратный остов уже построен.

Доказательство:

Рассмотрим произвольную дугу ab. Если a корень, то в самом начале *счётчик дуг* в корне увеличится на 1 для дуги ab. Затем корень пошлёт сообщение ***Финиш***, которое через конечное время дойдёт до вершины b. Если вершина a не корень, то она получит первое сообщение ***Старт*** через конечное время, после чего создаст сообщение ***Поиск корня***, в котором для дуги ab будет "$+1$". Через конечное время это сообщение дойдёт до корня, и *счётчик дуг* в корне увеличится на 1 для дуги ab. Затем от корня до вершины a дойдёт сообщение ***Прямое***. После этого по дуге ab будет послано сообщение ***Финиш***, которое дойдёт в вершину b через конечное время. Также через конечное время в вершине b появится обратная дуга. Следовательно, через конечное время произойдут оба события: получение вершиной b сообщения ***Финиш*** и появление у неё обратной дуги. В этот момент времени из вершины b по обратной дуге будет послано сообщение ***Минус*** с "-1" для дуги ab. Через конечное время это сообщение дойдёт до корня и корень из своего *счётчика дуг* вычтет 1 для дуги ab. Тем самым для каждой дуги корень сначала один раз прибавит к своему счётчику дуг 1, а потом один раз вычтет 1. Тем самым через конечное время, когда это произойдёт для всех дуг, *счётчик дуг* в корне обнулится.

Покажем, что в момент времени, когда счётчик дуг в корне обнуляется, обратный остов уже построен. Относительно каждой дуги корень сначала один раз получает "$+1$" в сообщении ***Поиск корня***, отправляемом из начала дуги, а потом один раз получает "-1" в сообщении ***Минус***, отправляемом из конца дуги. Поэтому в момент обнуления счётчика дуг относительно каждой дуги корень либо получил "$+1$" и "-1", либо не получил ни "$+1$", ни "-1". Допустим, утверждение не верно: найдётся вершина, в которой ещё не появилась обратная дуга. Поскольку граф сильно связный, найдётся путь от корня до этой вершины. Относительно последней дуги этого пути корень не получал "-1", следовательно, не получал "$+1$". Обозначим через bc первую дугу на этом пути, относительно которой корень не получал ни "$+1$", ни "-1". Поскольку корень сразу добавляет к счётчику дуг "$+1$" для каждой выходящей из корня дуги, вершина b – не корень. Тогда через ab обозначим предыдущую дугу на пути. Корень получил из вершины b сообщение ***Минус*** с "-1" относительно дуги ab. Но тогда корень до этого получил из вершины b сообщение ***Поиск корня*** с "$+1$" относительно дуги bc, что не верно. Мы пришли к противоречию, следовательно, наше допущение не верно, и обратный остов уже построен.

Утверждение доказано.

2.3. Третья часть алгоритма

Эта часть размечает в каждой вершине выходящие дуги как прямые дуги и хорды. Сначала все выходящие дуги будем считать хордами. Затем каждая дуга ab, по которой посылается сообщение ***Прямое***, отмечается как прямая дуга в вершине a.

Утверждение 2.3: Когда в корне обнулится *счётчик дуг*, все дуги правильно размечены как хорды или прямые.

Доказательство:

Поскольку сообщение ***Прямое*** направляется из корня по прямому пути в каждую вершину,

все прямые дуги рано или поздно будут отмечены. Покажем, что это произойдёт до обнуления *счётчика дуг* в корне. Рассмотрим произвольную прямую дугу ab. Поскольку граф сильно связен, имеется некоторая дуга bc. Обнуление *счётчика дуг* в корне произойдёт только после того, как из вершины c будет послано сообщение ***Минус*** с "-1" для дуги bc, и это сообщение дойдёт до корня. Но такое сообщение посылается только после того, как по дуге bc будет послано сообщение ***Финиш***, а это сообщение посылается только после того, как вершина b получит сообщение ***Прямое***, которое должно пройти по прямой дуге ab и отметить в вершине a дугу ab как прямую.

Утверждение доказано.

2.4. Четвёртая часть алгоритма

Эта часть устанавливает в вершинах значения *счётчиков входящих обратных дуг*. Вначале в каждой вершине *счётчик входящих обратных дуг* равен нулю. Добавим два новых сообщения ***Начало подсчёта*** и ***Конец подсчёта***. Сообщение ***Начало подсчёта*** начинает рассылаться из корня по прямым дугам после обнуления *счётчика дуг* в корне. Каждая вершина, кроме корня, получив сообщение ***Начало подсчёта***, во-первых, рассылает по всем выходящим прямым дугам такое же сообщение ***Начало подсчёта***, а, во-вторых, создаёт сообщение ***Конец подсчёта*** и посылает его по обратной дуге.

Сообщение ***Конец подсчёта*** содержит признак *первая обратная дуга*. Когда вершина создаёт сообщение ***Конец подсчёта***, она снабжает его этим признаком. Когда вершина принимает по входящей дуге сообщение ***Конец подсчёта*** с признаком *первая обратная дуга*, она прибавляет 1 к своему *счётчику входящих обратных дуг*, и, если это не корень, посылает сообщение ***Конец подсчёта*** дальше по обратной дуге уже без этого признака. Когда вершина принимает по входящей дуге сообщение ***Конец подсчёта*** без признака *первая обратная дуга*, она просто пересылает сообщение ***Конец подсчёта*** без изменения дальше по обратной дуге.

Сообщения ***Конец подсчёта*** суммируются в вершинах в ожидании отправки по обратной дуге. Для этого сообщение снабжается параметром *число суммируемых сообщений*. Когда в вершине ожидает отправки по обратной дуге сообщение ***Конец подсчёта*** и приходит новое сообщение ***Конец подсчёта***, их параметры *число суммируемых сообщений* складываются и с этим числом сохраняется одно сообщение ***Конец подсчёта***; при освобождении обратной дуги по ней посылается сообщение ***Конец подсчёта*** с накопленным *числом суммируемых сообщений*.

Когда *счётчик дуг* в корне обнуляется, в корне имеется *список векторов инициаторов*, который содержит вектора всех вершин, кроме корня. Корень подсчитывает длину списка и записывает её в *счётчик некорневых вершин*. Корень, получая сообщение ***Конец подсчёта***, вычитает параметр *число суммируемых сообщений* из *счётчика некорневых вершин*. Когда этот счётчик становится равным нулю, работа этой части и всего алгоритма закончены.

Утверждение 2.4: Когда в корне обнулится *счётчик некорневых вершин*, во всех вершинах будут правильные значения *счётчиков входящих обратных дуг*.

Доказательство:

Сообщение ***Начало подсчёта***, посылаемое по прямым дугам, дойдёт до каждой вершины один раз. Каждая вершина пошлёт один раз сообщение ***Конец подсчёта*** с признаком *первая обратная дуга*. Тем самым по каждой обратной дуге пройдёт ровно одно сообщение ***Конец подсчёта*** с признаком *первая обратная дуга*. Поэтому в каждой вершине будет установлено правильное значение *счётчика входящих обратных дуг* после того, как по всем обратным дугам будет получено сообщение ***Конец подсчёта*** с признаком *первая обратная дуга*.

Также для каждой обратной дуги до корня дойдёт сообщение ***Конец подсчёта*** с "1" для этой дуги в параметре *число суммируемых сообщений*. Число таких "1" равно числу обратных дуг, что равно *числу некорневых вершин*. Поэтому когда *счётчик некорневых вершин* станет равным нулю, на дугах графа не останется сообщений ***Начало подсчёта*** и ***Конец подсчёта***, а значения *счётчиков входящих обратных дуг* во всех вершинах будут правильные.

Утверждение доказано.

Итак, после завершения работы алгоритма бу-

дут построены прямой и обратный остовы графа: в каждой вершине будут отмечены прямые дуги и (если это не корень) одна обратная дуга. Кроме того, в каждой вершине будет установлено значение счётчика входящих обратных дуг.

3. ОЦЕНКА ПАМЯТИ И ВРЕМЕНИ РАБОТЫ АЛГОРИТМА ОБХОДА

3.1. Оценка размера памяти автомата вершины

Память автомата некорневой вершины содержит:

- 1 бит – признак того, что сообщение ***Старт*** ждёт освобождения выходящих дуг;
- для каждого инициатора в *списке векторов инициаторов* хранится
  - 1 вектор – *вектор инициатора*;
  - 1 бит – признак того, что сообщение ***Поиск корня*** ждёт освобождения выходящих дуг;
  - 1 бит – признак того, что сообщение ***Прямое*** ждёт освобождения выходящих дуг;
  - 1 вектор – параметр *число дуг* ждущего сообщения ***Поиск корня*** или параметр *вектор обратного пути* ждущего сообщения ***Прямое***; заметим, что для данного инициатора ждущим может только одно из сообщений ***Поиск корня*** и ***Прямое***, а число дуг – это вектор длины 1;
- 1 вектор – параметр *вектор обратного пути* ждущего сообщения ***Обратное***;
- 1 бит – признак ждущего сообщения ***Финиш***;
- $logs_1$ бит – параметр *число дуг* ждущего сообщения ***Минус***;
- 1 бит – признак ждущего сообщения ***Начало подсчёта***;
- $logn$ бит – параметр *число суммируемых сообщений* ждущего сообщения ***Конец подсчёта***, равен нулю, если нет ждущего сообщения;
- $logs_1$ бит – *счётчик входящих обратных дуг*.

Итого размер памяти автомата некорневой вершины: 1 бит $+n($1 вектор $+1$ бит $+1$ бит $+$ 1 вектор$)+1$ вектор $+1$ бит $+logs_1$ бит $+$ 1 бит $+logn$ бит $+logs_1$ бит
$=(2n+1)$ векторов $+(2n+logn+2logs_1+3)$ бит $\le (2n+1)$ векторов $+(2n+logn+2logm+3)$ бит $\le (2n+1)$векторов $+(2n+logn+2logns_0+3)$ бит $=(2n+1)\boldsymbol{O}(Dlogs_0)+2n+logn+2logns_0+3$, что равно $\boldsymbol{O}(nDlogs_0)$ для $D>0$.

Память автомата корня содержит:

- для каждого инициатора в *списке векторов инициаторов*;
  - 1 вектор – *вектор инициатора*;
  - 1 бит – признак того, что сообщение ***Прямое*** ждёт освобождения выходящих дуг;
  - 1 вектор – параметр *вектор обратного пути* ждущего сообщения ***Прямое***;
- 1 бит – признак ждущего сообщения ***Финиш***;
- 1 бит – признак ждущего сообщения ***Начало подсчёта***;
- $logm$ бит – *счётчик дуг*;
- $logn$ бит – *счётчик некорневых вершин*.

Итого размер памяти автомата корня:
$n($1 вектор $+1$ бит $+1$ вектор$)+1$ бит $+1$ бит $+$ $logm$ бит $+logn$ бит $=2n$ векторов $+(n+logm+$ $(n+logm+(n+logm+logn+2)$ бит $\le$
$2n$ векторов $+(n+logns_0+logn+2)$ бит $=$ $2n\boldsymbol{O}(Dlogs_0)+n+logns_0+logn+2$, что равно $\boldsymbol{O}(nDlogs_0)$ для $D>0$.

Итак, память автомата любой вершины равна $\boldsymbol{O}(nDlogs_0)$ для $D>0$.

3.2. Оценка времени работы алгоритма обхода

Мы всегда можем считать, что сообщения посылаются из вершины в порядке уменьшения их приоритетов: 1) **Старт**, 2) **Поиск корня**, 3) **Прямое**, 4) **Обратное**, 5) **Финиш**, 6) **Минус**, 7) **Начало подсчёта**, 8) **Конец подсчёта**.

Ниже мы докажем утверждение о том, что 1) если нет других сообщений, кроме

Поиск корня, то каждое такое сообщение достигает корня за время не более $T(n,k,D)$, 2) если нет других сообщений, кроме ***Прямое***, то каждое такое сообщение достигает своей цели за время не более $T(n,k,D)$. Здесь мы представим работу алгоритма как последовательность 8 этапов по 8 приоритетам сообщений и оценим время работы каждого этапа.

1. Сообщение ***Старт*** как самое приоритетное может ожидать в вершине освобождения выходящих дуг не более 1 такта. Ещё не более 1 такта сообщение может передаваться по дуге. Поскольку по каждой дуге сообщение ***Старт*** посылается ровно 1 раз, сообщения ***Старт*** достигнут всех вершин за время не более $2D$.

2. Рассмотрим движение сообщений ***Поиск корня*** после того, как все сообщения ***Старт*** достигнут всех вершин. Поскольку сообщений ***Старт*** уже нет, а остальные сообщения менее приоритетны, чем сообщение ***Поиск корня***, каждое такое сообщение ожидает в вершине освобождения выходящих дуг, занятых сообщениями других типов, не более 1 такта. Тем самым оценка $T(n,k,D)$ увеличивается не более, чем вдвое, то есть все сообщения ***Поиск корня*** достигнут корня за время не более $2T(n,k,D)$.

3. Рассмотрим движение сообщений ***Прямое*** после того, как все сообщения ***Поиск корня*** достигнут всех вершин. Поскольку более приоритетных сообщений уже нет, а остальные сообщения менее приоритетны, чем сообщение ***Прямое***, каждое такое сообщение ожидает в вершине освобождения выходящих дуг, занятых сообщениями других типов, не более 1 такта. Тем самым оценка $T(n,k,D)$ увеличивается не более, чем вдвое, то есть все сообщения ***Прямое*** достигнут своих целей за время не более $2T(n,k,D)$.

4. Рассмотрим движение сообщений ***Обратное*** после того, как все сообщения ***Прямое*** достигнут всех вершин. Эти сообщения движутся по обратным дугам и суммируются в вершинах в ожидании освобождения выходящих дуг. Каждое такое сообщение проходит путь длиной не более D. Поскольку более приоритетных сообщений уже нет, такое суммирующее сообщение ***Обратное*** ожидает в вершине освобождения выходящих дуг не более 1 такта. Тем самым все сообщения ***Обратное*** достигнут корня за время не более $2D$.

5. Рассмотрим движение сообщений ***Финиш*** после того, как все сообщения ***Обратное*** достигнут корня. С этого момента времени во всех вершинах, кроме корня, есть обратные дуги. Сообщения ***Финиш*** движутся по дугам, проходя по каждой дуге ровно 1 раз. Каждое такое сообщение проходит путь длиной не более D. Поскольку более приоритетных сообщений уже нет, а обратные дуги есть везде, каждое сообщение ***Финиш*** ожидает в вершине не более 1 такта. Тем самым все сообщения ***Финиш*** пройдут по всем дугам за время не более $2D$.

6. Рассмотрим движение сообщений ***Минус*** после того, как все сообщения ***Финиш*** пройдут по всем дугам. Эти сообщения движутся по обратным дугам до корня, суммируясь в каждой вершине. Каждое такое сообщение проходит путь длиной не более D, ожидая в вершине не более 1 такта. Тем самым все сообщения ***Минус*** дойдут до корня за время не более $2D$.
Итак, *счётчик дуг* в корне обнулится через время не более $2D + 2T(n,k,D) + 2T(n,k,D)+2D+2D+2D = 4T(n,k,D)+8D$. После этого происходит установка *счётчиков входящих обратных дуг.*

7. Сообщения ***Начало подсчёта*** посылаются по прямым дугам до каждой вершины. Поэтому каждое такое сообщение проходит путь длиной не более D. Поскольку более приоритетных сообщений уже нет, каждое сообщение ***Начало подсчёта*** ожидает в вершине освобождения выходящих дуг не более 1 такта. Тем самым все сообщения ***Начало подсчёта*** закончат своё движение за время не более $2D$.

8. Рассмотрим движение сообщений ***Ко-***

нец подсчёта после того, как все сообщения ***Начало подсчёта*** закончат своё движение. Эти сообщения движутся по обратным дугам до корня, суммируясь в каждой вершине. Каждое такое сообщение проходит путь длиной не более D, ожидая в вершине не более 1 такта. Тем самым все сообщения ***Конец подсчёта*** дойдут до корня за время не более $2D$.

Итак, весь алгоритм закончит свою работу за время не более $4T(n,k,D) + 8D + 2D + 2D = 4T(n,k,D) + 12D$.

3.3. Формальная модель движения сообщений ***Поиск корня*** *и* ***Прямое***

Величина $T(n,k,D)$ – это ограничение сверху на время движения всех сообщений ***Поиск корня*** в предположении, что других сообщений нет, а также время движения всех сообщений ***Прямое*** в предположении, что других сообщений нет.

Сначала рассмотрим движение *последнего сообщения* ***Поиск корня***, т.е. сообщения, которое последним приходит в корень. Это сообщение проходит путь от вершины-инициатора до корня длиной $d \le D$. Другие сообщения ***Поиск корня*** могут приходить в вершины этого пути "сбоку", т.е. по входящим дугам, не принадлежащим пути. В этом случае мы будем говорить, что сообщение *рождается* в вершине. Поскольку в вершине повторные сообщения с тем же инициатором игнорируются, будем говорить, что они *умирают* в вершине.

Теперь рассмотрим движение *последнего* сообщения ***Прямое***, т.е. сообщения, которое последним приходит от корня к своему инициатору. Сообщение ***Прямое*** движется от корня к вершине-инициатору по прямому пути длиной $d \le D$. Поскольку сообщения ***Прямое*** не рассылаются "веером" и двигаются только по прямым дугам, они, в отличие от сообщений ***Поиск корня***, не приходят в вершины пути "сбоку" и в вершинах не бывает повторных сообщений с тем же инициатором. Очевидно, это частный случай движения сообщений ***Поиск корня***.

Формализуем описание движения сообщений ***Поиск корня.***

Имеется путь длины $d \le D$. Вершины и дуги будем называть *позициями*. Перенумеруем позиции следующим образом: начальная вершина пути получает номер 0, номер дуги на 1 больше номера её начала, номер конца дуги на 1 больше номера дуги. Тем самым вершины имеют чётные номера $0, 2, \ldots, 2d$, а дуги имеют нечётные номера $1, \ldots, 2d-1$. Если $i < i'$, то будем говорить, что позиция i находится *левее* позиции i', а позиция i' находится *правее* позиции i. В позициях могут находиться сообщения, которые в формальной модели будем называть *точками*.

Для точек возможны следующие события: 1) рождение точки в вершине (приход сообщения "сбоку"), 2) смерть точки в вершине (сообщение игнорируется), 3) перемещение точки с начала дуги на дугу (сообщение посылается по дуге из её начала), 4) перемещение точки с дуги в конец дуги (сообщение принимается с дуги в её конец). Будем считать, что все эти действия происходят мгновенно.

Событием *появления* точки в позиции будем называть событие рождения этой точки в этой позиции (если позиция вершина) или событие перемещения этой точки в эту позицию из предыдущей позиции. Соответственно, событием *исчезновения* точки из позиции будем называть событие смерти этой точки в этой позиции (если позиция вершина) или событие перемещения этой точки из этой позиции в следующую позицию.

Из описания алгоритма следуют следующие правила движения точек:

1. Точка находится на дуге не более одного такта.

2. Точки перемещаются на дугу тогда, когда на дуге не остаётся точек, причём на дугу сразу перемещается минимум из числа точек в начале дуги и "ёмкости" дуги k.

3. Если точка умирает в вершине, то это происходит в тот самый момент, когда она появляется в вершине (рождается или перемещается с входящей дуги).

4. Число точек, умерших в вершине, не превышает числа точек, рождённых в вершине (это следует из того, что умирающая точка –

это повторное сообщение с тем же инициатором).

5. В начальной вершине 0 на такте 0 есть точка, которая не умрёт. Будем называть её *нашей* точкой (это последнее сообщение).

6. Общее число точек, появляющихся во всех позициях, конечно (конечно число сообщений ***Поиск корня***, что доказано выше в доказательстве Утверждения 2.1).

7. Число точек, которые появляются, но не умирают в конечной позиции $2d$ до момента появления в этой позиции нашей точки включительно, не превышает n (на самом деле, вместе с нашей точкой число таких точек не превышает числа некорневых вершин $n-1$).

Если точка находится в некоторой вершине в начальный момент времени 0, то будем считать, что точка рождается в этой вершине в момент времени 0. Если точка находится на некоторой дуге в начальный момент времени 0, то будем считать, что точка рождается в начале этой дуги в момент времени 0 и мгновенно перемещается на пустую дугу вместе с несколькими точками.

Будем говорить, что задано *поведение точек*, если для каждой точки заданы место (позиция) и время её рождения и смерти, а также задержка этой точки на каждой дуге таким образом, что соблюдаются указанные выше правила движения точек.

Обозначим через $U_{i,j}$ число точек, появившихся и не умерших в позиции i на такте $i+j$. Обозначим через $T = T(n,k,D)$ номер такта, на котором наша точка достигает конечной позиции $2d$.

Лемма 3.1: Пусть задано поведение точек, удовлетворяющее правилам 1–4 (может не удовлетворять правилам 5–7). Пусть при этом поведении для каждого j такого, что $i+j \le T$, имеет место $U_{0,j} \ge kj$. Тогда для каждой позиции i и каждого j такого, что $i+j \le T$, имеет место $U_{i,j} \ge kj$.

Доказательство:

Будем вести доказательство индукцией по i и j.

База индукции по i для $j \le T$: $U_{0,j} \ge kj$, что задано условием леммы.

База индукции по j для $i \le T$: $U_{i,0} \ge k \cdot 0 = 0$, что очевидно.

Шаг индукции: пусть утверждение леммы верно как для i и $j+1$, так и для $i+1$ и j, где $i < 2d$, т.е. $U_{i,j}+1 \ge k(j+1)$ и $U_{i+1,j} \ge kj$. Нужно доказать, что если $i+j+2 \le T$, то утверждение леммы верно для $i+1$ и $j+1$, т.е. $U_{i+1,j+1} \ge k(j+1)$. Рассмотрим два возможных случая.

i чётно, т.е. позиция i является дугой.

Введём обозначения:

A – число точек, переместившихся с дуги i в вершину $i+1$ и не умерших там за все такты от 0 до $i+j+1$;

A' – число точек, переместившихся с дуги i в вершину $i+1$ и умерших там за все такты от 0 до $i+j+1$;

B – число точек, рождённых в вершине $i+1$ и не умерших там за все такты от 0 до $i+j+1$;

B' – число точек, рождённых в вершине $i+1$ и умерших там за все такты от 0 до $i+j+1$;

A_Δ – число точек, переместившихся с дуги i в вершину $i+1$ и не умерших там в течение одного такта $i+j+2$;

A'_Δ – число точек, переместившихся с дуги i в вершину $i+1$ и умерших там в течение одного такта $i+j+2$;

B_Δ – число точек, рождённых в вершине $i+1$ и не умерших там в течение одного такта $i+j+2$;

B'_Δ – число точек, рождённых в вершине $i+1$ и умерших там в течение одного такта $i+j+2$;

a – число точек на дуге i в конце такта $i+j+1$, которые не умрут в вершине $i+1$;

a' – число точек на дуге i в конце такта $i+j+1$, которые умрут в вершине $i+1$.

Учитывая, что по правилу 3 точка умирает в вершине в момент появления там или вообще не умирает в этой вершине, по правилу 4 имеем $A' + A'_\Delta + B' + B'_\Delta \leq B + B_\Delta + B' + B'_\Delta$.

Отсюда $A' + A'_\Delta \leq B + B_\Delta$.

По определению $U_{i+1,j} = A + B$.

Точка, появляющаяся на дуге, не умирает на дуге и находится либо на дуге, либо среди точек (числом $A + A'$), перемещавшихся в конец дуги. Поэтому $U_{i,j+1} = a + a' + A + A'$.

Точки, приходящие в вершину $i+1$ за такт и не умирающие в ней, слагаются из двух частей: точки (числом A_Δ), переместившиеся с дуги i и не умершие в вершине $i+1$, и точки (числом B_Δ), рождённые в вершине $i+1$ и не умершие в ней. Поэтому $U_{i+1,j+1} = U_{i+1,j} + A_\Delta + B_\Delta$.

По правилу 1 за 1 такт с дуги i в вершину $i+1$ перемещаются все точки (числом $a + a'$), находившиеся на дуге в начале такта, но может переместиться и больше точек (числом $A_\Delta + A'_\Delta$), если их задержки на дуге достаточно малы и дуга за такт освобождается несколько раз. Поэтому $A_\Delta + A'_\Delta \geq a + a'$.

Тогда $U_{i+1,j+1} = U_{i+1,j} + A_\Delta + B_\Delta = A + B + A_\Delta + B_\Delta = A + A_\Delta + B + B_\Delta \geq A + A_\Delta + A' + A'_\Delta = A + A' + A_\Delta + A'_\Delta \geq A + A' + a + a' = U_{i,j+1} \geq k(j+1)$.

Итак, $U_{i+1,j+1} \geq k(j+1)$, что и требовалось доказать.

i нечётно, т.е. позиция i является вершиной.

Обозначим через x – число точек, находящихся в вершине i в конце такта $i + j + 1$.

Точки, которые приходили в вершину i и не умерли там до такта $i + j + 1$ включительно, либо находятся в конце этого такта в вершине (их число равно x), либо перемещались на дугу $i+1$ и не умирали там, поскольку точки на дуге не умирают.

Поэтому: $U_{i,j+1} = x + U_{i+1,j}$.

Рассмотрим случай $x \leq k$.

По правилу 1 за один такт дуга $i+1$ освободится хотя бы один раз и, поскольку $x \leq k$, по правилу 2 все точки, находившиеся в вершине в начале такта, переместятся на дугу. Заметим, что на дугу может переместиться больше точек, если они появляются в вершине до первого освобождения дуги или появляются позже, но в течение такта, а задержки на дуге достаточно малы и дуга за такт освобождается несколько раз.

Поэтому $U_{i+1,j+1} \geq U_{i+1,j} + x = U_{i,j+1} \geq k(j+1)$.

Итак, $U_{i+1,j+1} \geq k(j+1)$, что и требовалось доказать.

Рассмотрим случай $x > k$.

По правилу 1 за один такт дуга $i+1$ освободится хотя бы один раз и, поскольку $x > k$, по правилу 2 на дугу переместятся k точек. Заметим, что, как и в предыдущем случае, на дугу может переместиться и больше точек. Поэтому $U_{i+1,j+1} \geq U_{i+1,j\,+\,k} \geq kj + k = k(j+1)$.

Итак, $U_{i+1,j+1} \geq k(j+1)$, что и требовалось доказать.

Шаг индукции доказан.

Лемма доказана.

Утверждение 3.1: Наша точка достигает конечной позиции $2d$ на такте $T \leq n/k + 2d + 1$.

Доказательство:

Прежде всего, заметим, что наша точка достигает конечной позиции $2d$ за конечное время. Действительно, поскольку наша точка не умирает по правилу 5, для этого достаточно, чтобы наша точка находилась в каждой позиции конечное время. Задержка нашей точки на дуге конеч-

на по правилу 1 (не более 1 такта). Конечность задержки нашей точки в вершине определяется конечностью задержек точек на выходящей дуге (правило 1), обязательностью перемещения точек на дугу при освобождении дуги (правило 2) и конечностью числа точек (правило 6).

Точки, которые всегда находятся левее нашей точки, очевидно, не влияют на движение нашей точки, поэтому их можно игнорировать: будем считать, что таких точек нет. Если точка рождается левее нашей точки, но "догоняет" нашу точку в некоторой вершине в некоторый момент времени, то это эквивалентно тому, что в этот момент времени эта точка рождается в этой вершине. Учитывая вышесказанное, в дальнейшем будем считать, что точки не рождаются левее нашей точки. Тем более, точки не рождаются левее самой левой точки в текущем расположении точек.

Рассмотрим *расширенное* поведение точек, которое отличается тем, что в позиции 0 на каждом такте $t \leq T$ рождается дополнительное число точек так, чтобы выполнялось условие леммы 3.1 $U_{0,t} \geq kt$ для $t \leq T$. Будем считать, что каждая из этих дополнительных точек 1) рождается в начальной позиции 0 и не умирает (ни в какой вершине), 2) имеет максимальную задержку на дугах в 1 такт, 3) перемещается из вершины на выходящую из неё дугу только после исходных (не дополнительных) точек.

Расширенное поведение точек удовлетворяет правилам 1–6. Пояснения требует только выполнение правила 2 в связи с тем, что дополнительные точки перемещаются из вершины на выходящую из неё дугу только после исходных точек. Здесь нет противоречия, поскольку исходные точки не рождаются левее самой левой исходной точки. Поэтому в тот момент времени, когда согласно правилу 2 дополнительные точки должны перемещаться из вершины на выходящую дугу, в этой вершине либо уже нет исходных точек, либо их число меньше k и они перемещаются вместе с несколькими первыми дополнительными точками. Тем самым в дальнейшем самая левая исходная точка оказывается правее данной вершины и, следовательно, в этой вершине уже не появятся исходные точки.

По определению дополнительные точки не обгоняют исходные точки, поэтому дополнительные точки не влияют на движение исходных точек. Тем самым в расширенном поведении точек наша точка, по-прежнему, достигает конечной позиции $2d$ на такте T и до этого такта включительно число исходных точек, которые появлялись, но не умирали в конечной позиции $2d$, по правилу 7 не превышает n. Вместе с нашей точкой в конечную позицию с входящей дуги могут переместиться также несколько дополнительных точек, но их число не превышает $k-1$. Поэтому общее число точек (как исходных, так и дополнительных), которые появлялись, но не умирали в конечной позиции $2d$ до такта T включительно $U_{2d,T-2d} \leq n + k - 1$.

Поскольку расширенное поведение точек удовлетворяет условиям леммы 3.1, для $T \geq 2d$ имеем $U_{2d,T-2d} \geq k(T-2d)$.
Следовательно, $k(T-2d) \leq n + k - 1$.
Отсюда $T \leq n/k + 1 - 1/k + 2d \leq n/k + 2d + 1$.
Если же $T < 2d$, то тем более $T \leq n/k + 2d + 1$.

Утверждение доказано.

Итак, $T(n,k,D) \leq n/k + 2d + 1$. Тем самым, время работы алгоритма не превосходит $4T(n,k,D) + 12D = 4n/k + 8D + 4 + 12D = 4n/k + 20D + 4 = \boldsymbol{O}(n/k + D)$. В более общем случае, время равно $\boldsymbol{O}(n/k_1k_2 + D)$, где k_1 — ёмкость дуги, а k_2 — размер сообщения в векторах.

4. АГРЕГАТНЫЕ ФУНКЦИИ И 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 этом разделе мы дадим формальное определение агрегатной функции, докажем критерий агрегатности и покажем, что любая функция $f(x)$ имеет агрегатное расширение, то есть может быть вычислена как $h(g(x))$, где g — агрегатная функция. Также покажем, что существует и единственное минимальное агрегатное расширение, вычисляющее минимум информации, по которой еще можно восстановить функцию f. Теория агрегатных функций, излагаемая в этом разделе, является модификацией теории индуктивных функций в [9].

Далее рассматриваются функции на конечных мультимножествах из элементов некоторого ба-

зового множества X. Множество всех конечных мультимножеств из элементов X обозначим через X^-. Для $a \in X^-$ через $\#a$ обозначим мощность мультимножества a. Под операциями объединения, пересечения, дополнения и пр. далее подразумеваются операции на мультимножествах, т.е. учитывающие кратности элементов. Через N обозначим множество натуральных чисел.

Агрегатная функция $g : X^- \to A$ – это такая функция, что $\exists e : A \times A \to A \forall a, b \in X^- \; g(a \cup b) = e(g(a), g(b))$.

Замечание 4.1: Легко показать, что для агрегатной функции g выполнено следующее: $\forall r \in \boldsymbol{N} \exists e_r \; : \; A^r \to A \forall a_1, \ldots, a_r \in X^- g(\cup\{a_i \mid 1 \le i \le r\}) = e_r(g(a_1), \ldots, g(a_r))$.

Замечание 4.2: Множество агрегатных функций не пусто: например, такими функциями являются сумма всех членов мультимножества и их минимум, в обоих случаях функциями e также служат суммы и минимумы.

Лемма 4.1 (критерий агрегатности): Функция $g : X^- \to A$ является агрегатной тогда и только тогда, когда $\forall a, b \in X^- \forall x \in X g(a) = g(b) \Rightarrow g(a \cup \{x\}) = g(b \cup \{x\})$.

Доказательство:

Необходимость.

$\forall a, b \in X^- \forall x \in X g(a \cup \{x\}) = e(g(a), g(\{x\}))$ и $g(b \cup \{x\}) = e(g(b), g(\{x\}))$.

Поэтому если $g(a) = g(b)$, то $g(a \cup \{x\}) = e(g(a), \; g(\{x\})) = e(g(b), \; g(\{x\})) = g(b \cup \{x\})$.

Достаточность.

Сначала индукцией по $\#c$ докажем $\forall a, b, c \in X^- g(a) = g(b) \Rightarrow g(a \cup c) = g(b \cup c)$.

Для $\#c = 0$, утверждение тривиально.

Для $\#c = 1 \exists x \in X = \{x\}$, и мы получаем заданное условие:

$\forall a, b \in X^- g(a) = g(b) \Rightarrow g(a \cup c) = g(a \cup \{x\}) = g(b \cup \{x\}) = g(b \cup c)$.

Пусть утверждение доказано для $\#c = n - 1$, докажем его для $\#c = n > 1$.

Имеем: $\forall a, b, c \in X^- \exists x \in cg(a) = g(b) \Rightarrow g(a \cup c\backslash\{x\}) = g(b \cup c\backslash\{x\}) \Rightarrow g(a \cup c) = g((a \cup c\backslash\{x\}) \cup \{x\}) = g((b \cup c\backslash\{x\}) \cup \{x\}) = g(b \cup c)$.

Теперь докажем: $\forall a_1, a_2, b_1, b_2 \in X^-(g(a_1) = g(b_1) \& g(a_2) = g(b_2)) \Rightarrow g(a_1 \cup a_2) = g(b_1 \cup b_2)$.

Имеем: $g(a_2) = g(b_2) \Rightarrow g(a_1 \cup a_2) = g(a_1 \cup b_2)$ и $g(a_1) = g(b_1) \Rightarrow g(a_1 \cup b_2) = g(b_1 \cup b_2)$.

Итак, $g(a_1 \cup a_2) = g(a_1 \cup b_2) = g(b_1 \cup b_2)$.

Определим теперь $e \; : \; g(X^-) \times g(X^-) \to A$ как $e(g(a), g(b)) = g(a \cup b)$. Поскольку по доказанному выбор конкретных a и b не имеет значения – важны только значения $g(a)$ и $g(b)$, эта функция определена корректно. Из ее определения следует, что g агрегатная.

Лемма доказана.

Замечание 4.3: Из критерия агрегатности следует, что $g \; : \; X^- \to A$ является агрегатной тогда и только тогда, когда $\exists g' \; : \; A \times X \to A \forall a \in X^- x \in X g(a \cup \{x\}) = g'(g(a), x)$. 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 4.4: 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 4.2 (единственность минимального агрегатного расширения): Минимальное агрегатное расширение функции $f : X^- \to A$ единственно с точностью до взаимно однозначных отображений.

Доказательство:

Если $g : X^- \to B$ и $g' : X^- \to C$ – минимальные агрегатные расширения функции f, то $\exists i : C \to B, i' : B \to C$ взаимно однозначные и такие, что ii' и $i'i$ тождественные отображения на B и C, $g = ig'$ и $g' = i'g$.

Утверждение доказано.

Утверждение 4.3 (существование минимального агрегатного расширения): $\forall f : X^- \to A$ существует минимальное агрегатное расширение.

Доказательство:

Определим отношение на X^-.

Для $a, b \in X^-$ определим $a \sim b \Leftrightarrow \forall c \in X^- f(a \cup c) = f(b \cup c)$.

Оно рефлексивно, симметрично и транзитивно, значит, является эквивалентностью.

Оно совместимо с функцией f, т.е., $\forall a, b \in X^- a \sim b \Rightarrow f(a) = f(b)$.

Для любого совместимого с f отношения эквивалентности можно определить фактор-функцию $h : X^-/\sim \to A$, такую, что $h(\pi^\sim(a)) = f(a)$, где $\pi^\sim$ – проекция на фактор-множество по отношению $\sim$. Функция $h(\pi^\sim(a))$ определяется как $f(a)$ для любого $a \in \pi^\sim(a)$. Поскольку отношение $\sim$ совместимо с f, это определение корректно – значения f равны для всех представителей одного класса эквивалентности.

Кроме того, $\forall a, b \in X^- \forall x \in X a \sim b \Rightarrow a \cup \{x\} \sim b \cup \{x\}$.

Действительно, если $\forall c \in X^- f(a \cup c) = f(b \cup c)$, то и $\forall c \in X^- f(a \cup \{x\} \cup c) = f(b \cup \{x\} \cup c)$.

Значит, для проекции $\pi^\sim$ выполнен критерий агрегатности: $\forall a, b \in X^- \forall x \in X (\pi^\sim(a) = \pi^\sim(b) \Leftrightarrow a \sim b) \Rightarrow (a \cup \{x\} \sim b \cup \{x\} \Leftrightarrow \pi^\sim(a \cup \{x\}) = \pi^\sim(b \cup \{x\}))$.

Т.е., $\pi^\sim : X^- \to X^-/\sim$ удовлетворяет определению агрегатного расширения функции f.

Теперь возьмем произвольную функцию $g' : X^- \to C$, являющуюся агрегатным расширением функции f.

По определению расширения
$\exists h' : C \to A \forall a \in X^- f(a) = h'(g'(a))$.

Покажем, что $\forall a, b \in X^- g'(a) = g'(b) \Rightarrow a \sim b$.

Поскольку g' – агрегатная функция, то $g'(a) = g'(b) \Rightarrow \forall c \in X^- g'(a \cup c) = g'(b \cup c)$.

Из $f = h'g'$ следует, что $g'(a) = g'(b) \Rightarrow f(a) = f(b)$.

Поэтому $g'(a) = g'(b) \Rightarrow \forall c \in X^- g'(a \cup c) = g'(b \cup c) \Rightarrow \forall c \in X^- f(a \cup c) = f(b \cup c) \Rightarrow a \sim b$.

Значит, $\forall a, b \in X^- g'(a) = g'(b) \Rightarrow a \sim b \Rightarrow \pi \sim (a) = \pi \sim (b)$. Таким образом, можно определить функцию $i : g'(X^-) \to X^-/\sim$ так, что $i(g'(a)) = \pi \sim (a)$. Определение корректно, поскольку, как только что показано, значение $\pi \sim (a)$ не зависит от выбора конкретного a, если сохраняется значение $g'(a)$.

Таким образом, $\pi \sim$ является агрегатным расширением f, вычисляется по любому агрегатному расширению f, кроме того, образом $\pi \sim$ является все множество $X^-/\sim$. Значит, $\pi \sim$ удовлетворяет определению минимального агрегатного расширения f.

Утверждение доказано.

Замечание 4.5: Примеры минимального агрегатного расширения (π_i – проекция кортежа на i-ый компонент). Легко показать, что:

- для функции вычисления среднего арифметического

$f(a_1, \ldots, a_n) = (a_1 + \cdots + a_n)/n$ минимальным агрегатным расширением является $g(a_1, \ldots, a_n) = (a_1 + \cdots + a_n,\ n)$; $f = \pi_1 g \ / \ \pi_2 g$;

- для функции вычисления среднего геометрического $f(a_1, \ldots, a_n) = \sqrt[n]{(a_1 \cdot \ \ldots \ \cdot a_n)}$ минимальным агрегатным расширением является $g(a_1, \ldots, a_n) = (a_1 \cdot \ \ldots \ \cdot a_n,\ n)$; $f = {}^{\pi_2 g}\sqrt{\pi_1 g}$;
- для функции вычисления среднего квадратичного $f(a_1, \ldots, a_n) = \sqrt{((a_1^2 + \cdots + a_n^2)/n)}$ минимальным агрегатным расширением является $g(a_1, \ldots, a_n) = (a_1^2 + \cdots + a_n^2,\ n)$; $f = \sqrt{(\pi_1 g \ / \ \pi_2 g)}$.

Итак, каждую функцию $f : X^-$ 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: $b = b_1 \cup \cdots \cup b_r$. 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 4.1, для каждого элемента $b = b_1 \cup \cdots \cup b_r$ вложенного разбиения вычисляется значение $g(b) = e_r(g(b_1), \ldots, g(b_r))$. При этом сама функция e_r может вычисляться итеративно с помощью функции e: для $r > 2$ имеем $e_r(g(b_1), \ldots, g(b_r)) = e(e_r - 1(g(b_1), \ldots, g(b_{r-1})), g(b_r))$. После того, как будет получено значение $g(a)$, вычисляется искомый результат $f(a) = h(g(a))$.

5. ОПИСАНИЕ АЛГОРИТМА ПУЛЬСАЦИИ И ОЦЕНКА ВРЕМЕНИ ЕГО РАБОТЫ

Алгоритм пульсации предназначен для вычисления значения заданной функции f от мультимножества $a \in X^-$ значений, записанных в вершинах графа. Мы будем предполагать, что в каждой вершине i записано значение $x(i)$ с единичной кратностью. Алгоритм пульсации использует разметку графа, оставленную описанным выше алгоритмом обхода. Эта разметка включает прямой и обратный остовы графа: в каждой вершине отмечены прямые дуги и (если это не корень) одна обратная дуга. Кроме того, в каждой вершине установлено значение *счётчика входящих обратных дуг*.

5.1. *Описание алгоритма пульсации*

Алгоритм пульсации использует два типа сообщений: ***Вопрос*** и ***Ответ***. Сначала в автомат корня поступает извне сообщение ***Вопрос***, содержащее указание на три функции: h, e и g. После завершения алгоритма корень посылает вовне сообщение ***Ответ*** с параметром $f(a)$.

Сообщение ***Вопрос*** распространяется от корня ко всем вершинам по прямому остову. Получив это сообщение, корень запоминает параметры h, e и g и посылает сообщение ***Вопрос*** по каждой выходящей прямой дуге с параметрами e и g. Также каждая некорневая вершина, получив сообщение ***Вопрос***, запоминает параметры e и g и пересылает ***Вопрос*** далее по каждой выходящей прямой дуге.

Сообщение ***Ответ*** распространяется от всех вершин к корню по обратному остову. Обратный остов задаёт вложенное разбиение мультимножества a.

Листовая вершина i обратного остова (в этой вершине *счётчик входящих обратных дуг* равен нулю) при получении сообщения ***Вопрос*** вычисляет $g(x(i))$ и посылает его как параметр сообщения ***Ответ*** по обратной дуге.

Внутренней (не листовой) вершине i обратного остова соответствует элемент $b_i = b_{i1} \cup \cdots \cup b_{ir}$ вложенного разбиения, где $r - 1$ равно числу входящих в эту вершину обратных дуг. Задача этой вершины – вычислить значение $g(b_i)$, при этом для $j = 1 \ldots r - 1$ значение $g(b_{ij})$ будет получено по j-ой входящей обратной дуге, а $b_{ir} = g(x(i))$. Получив сообщение ***Вопрос***, автомат такой вершины вычисляет $g(x(i))$ и запоминает его как *промежуточный результат* $y(i)$, а *счётчик входящих обратных дуг* копирует в *счётчик ответов*. Далее при получении сообщения ***Ответ*** по входящей обратной дуге j с

параметром $g(b_{ij})$, изменяется *промежуточный результат* $y(i) := e(g(b_{ij}), y(i))$, а *счётчик ответов* уменьшается на 1.

Если вершина i не корень, то при обнулении *счётчика ответов* по выходящей обратной дуге посылается сообщение ***Ответ*** с параметром $y(i) = g(b_i)$. Если вершина i корень, то при обнулении *счётчика ответов* вычисляется окончательное значение $f(a) = h(y(i))$ и посылается вовне как параметр сообщения ***Ответ***.

5.2. *Оценка времени работы алгоритма пульсации*

Мы всегда можем считать, что сообщения посылаются из вершины в порядке уменьшения их приоритетов: 1) ***Вопрос***, 2) ***Ответ***.

Сообщения ***Вопрос*** распространяются по прямому остову, причём каждое сообщение проходит путь длиной не более D. Одновременно с сообщениями ***Вопрос*** могут пересылаться только сообщения ***Ответ***, распространяющиеся по обратным дугам, а некоторые дуги могут быть как прямыми, так и обратными. Поэтому сообщение ***Вопрос*** может ожидать в вершине освобождения дуги, занятой сообщением ***Ответ***. Однако, поскольку ***Вопрос*** приоритетнее ***Ответа***, время ожидания не превышает 1 такт. Тем самым, время распространения всех сообщений ***Вопрос*** не превышает $2D$ тактов.

Рассмотрим движение сообщений ***Ответ*** после того, как перестали распространяться все сообщения ***Вопрос***. Каждое сообщение ***Ответ*** двигается по обратным дугам до корня и, тем самым, проходит путь длиной не более D. Поскольку сообщения ***Ответ*** "суммируются" в каждой вершине i в виде *промежуточного результата* $y(i)$, а сообщений других типов нет, время распространения всех сообщений ***Ответ*** не превышает D тактов.

Итак, время работы алгоритма пульсации не превышает $3D = \boldsymbol{O}(D)$.

6. ЗАКЛЮЧЕНИЕ

Алгоритм пульсации не только вычисляет значение функции $f(a)$, где a – мультимножество значений, записанных в вершинах графа, но и меняет состояние автоматов вершин: в каждой вершине i остаётся в качестве *промежуточного результата* значение $g(b_i)$, где b_i – мультимножество значений, записанных в вершинах максимального поддерева обратного остова с корнем в вершине i. Понятно, что алгоритм пульсации можно использовать для установки автоматов вершин и в какие-то другие, например, одинаковые состояния.

Также можно поставить задачу вычисления функции не от мультимножества, а от последовательности значений, записанных в вершинах графа. Для этого нужно задать какой-то линейный порядок вершин. В то же время у нас уже есть нумерация дуг, выходящих из каждой вершины. Такая нумерация задаёт естественный частичный порядок вершин в виде прямого остова. Такой частичный порядок может использоваться как основа соответствующего линейного порядка. Но можно поставить задачу вычисления функции с учётом не линейного, а этого естественного частичного порядка. Последнее означает, что значение функции может зависеть от порядка значений в вершинах, которые сравнимы по частичному порядку, но не должно зависеть от порядка значений в вершинах, которые не сравнимы по частичному порядку.

Однако, во-первых, описанный выше алгоритм обхода строит не обязательно тот прямой остов, который определяется нумерацией дуг, а, во-вторых, алгоритм пульсации производит вычисления не по прямому, а по обратному остову. В этом и состоит проблема разработки алгоритма вычисления функции при заданном частичном или линейном порядке вершин графа. Решение этой проблемы может быть темой будущих исследований.

СПИСОК ЛИТЕРАТУРЫ

1. *Steven S. Skiena.* The Algorithm Design Manual. Springer-Verlag, New York, 1997.

2. *Бурдонов И.Б., Косачев А.С., Кулямин В.В.* Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай // Программирование. 2003. № 5. С. 59–69.

3. *Бурдонов И.Б., Косачев А.С., Кулямин В.В.* Неизбыточные алгоритмы обхода ориентированных графов. Недетерминированный случай // Программирование. 2004. № 1. С. 2–17.

4. *Rabin M.O.* Maze Threading Automata. An unpublished lecture presented at MIT and UC. Berkeley, 1967.

5. *Бурдонов И.Б.* Обход неизвестного ориентированного графа конечным роботом // Программирование, 2004. № 4. С. 11–34.

6. *Бурдонов И.Б.* Проблема отката по дереву при обходе неизвестного ориентированного графа конечным роботом // Программирование. 2004. № 6. С. 6–29.

7. *Бурдонов И.Б., Косачев А.С.* Обход неизвестного графа коллективом автоматов. Труды Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: все грани параллелизма”, 2013, изд. МГУ. С. 228–232.

8. *Бурдонов И.Б., Косачев А.С.* Обход неизвестного графа коллективом автоматов // Труды ИСП РАН. 2014. № 27. С. 43–86.

9. *Кушнеренко А.Г., Лебедев Г.В.* Программирование для математиков. Наука, Главная редакция физико-математической литературы, Москва, 1988.