Общий подход к решению задач на графах коллективом автоматов[1]

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

Аннотация. Предложен общий подход к решению задач на неориентированном упорядоченном корневом связном графе коллективом автоматов, расположенных в вершинах графа и обменивающихся сообщениями по рёбрам графа. Автоматы считаются полуроботами, т.е. размер их памяти может расти вместе с ростом числа *n* вершин и числа *m* рёбер графа, но описание графа может не помещаться в памяти автомата. В разделе 2 классифицируются модели коллектива автоматов на графе в зависимости от размера памяти автомата, времени срабатывания автомата и ёмкости ребра (числа сообщений, одновременно перемещающихся по ребру). Выбрана модель максимального распараллеливания, в которой время срабатывания автомата считается нулевым, а ёмкость ребра неограниченной. Это позволяет получать нижние оценки сложности алгоритмов решения задач. Раздел 3 определяет правила оценки алгоритмов. В разделе 4 описаны базовые процедуры обработки сообщений и проводится классификация используемых сообщений в зависимости от маршрутов, проходимых сообщениями, и способа размножения или слияния сообщений. На основе этих процедур предлагается строить алгоритмы решения задач на графах, что демонстрируется в следующих разделах статьи. В разделах 5-9 предложены алгоритм построения остова графа, алгоритм универсального «стопора», определяющего конец работы любого алгоритма по отсутствию в графе сообщений, используемых этим алгоритмом, алгоритм построения дерева кратчайших путей, алгоритм нумерации вершин графа, и алгоритм сбора полуроботами информации о графе в неограниченной памяти автомата корня. Предложенные алгоритмы имеют линейную сложность от *n* и *m*, и используют число сообщений, линейно зависящее от *n* и *m*. В разделе 10 рассматривается задача поиска максимального пути в нумерованном графе, которая относится к классу *NP*. За счёт неограниченного (экспоненциального) числа сообщений алгоритм решения этой задачи имеет линейную сложность. В заключении подводятся итоги и намечаются направления дальнейших исследований: решение других задач на графах и расширение подхода на ориентированные графы, а также недетерминированные и динамические графы.

Ключевые слова: неориентированные графы; задачи на графах; асинхронные распределённые системы; распределенные алгоритмы.

[1] Работа поддержана РФФИ (грант 17-07-00682 А)

Burdonov I.B., Kossatchev A.S. A general approach to solving problems on graphs by collective automata. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 2, 2017, pp. 27-76.

DOI: 10.15514/ISPRAS-2017-29(2)-2

Для цитирования: Бурдонов И.Б., Косачев А.С. Общий подход к решению задач на графах коллективом автоматов. Труды ИСП РАН, vol. 29, issue 2, 2017, pp.27-76. DOI: 10.15514/ISPRAS-2017-29(2)-2

1. Введение

Необходимость в решении различных задач на графах возникает во многих приложениях: в информатике и программировании, в коммуникационных и транспортных системах, схемотехнике, экономике, логистике, физике, химии и др. Примером таких задач могут служить задачи поиска максимального пути в графе, построение минимального остовного дерева во взвешенном графе, построение максимального независимого множества вершин, определение всех мостов в неориентированном графе, проверка наличия эйлерового пути или цикла, проверка наличия гамильтонового пути или цикла и построение их, проверка планарности графа и раскраска планарного графа, построение наибольшего ациклического подграфа ориентированного графа и др.

В мировой литературе, как в теоретическом, так и в практическом планах описано много различных алгоритмов решения актуальных задач на графах с оценками времени работы в зависимости от числа вершин, рёбер и других параметров графа. Однако в большинстве своём эти алгоритмы имеют большую вычислительную сложность, поэтому в последние годы растёт интерес к параллельным и распределённым компьютерным алгоритмам. Такие алгоритмы позволяют существенно уменьшить время решения задач и требуют меньше памяти в каждом из используемых компьютеров. Известные и апробированные алгоритмы часто плохо распараллеливаются или возможность такого распараллеливания требует дополнительных исследований. В первом случае необходим поиск новых алгоритмов, а во втором случае – адаптация имеющихся алгоритмов.

Моделью распределённых алгоритмов для решения задач на графах служит коллектив автоматов, которые как-то «привязаны» к графу и взаимодействуют между собой с помощью передачи сообщений. В зависимости от способа «привязки» к графу различают две альтернативные базовые метамодели. В обеих метамоделях автоматы находятся в вершинах графа, а различаются эти метамодели тем, как используются рёбра графа. В первой метамодели автоматы мобильны и могут перемещаться по рёбрам графа из одной вершины в другую, а для взаимодействия автоматов используется локальная память вершин, и/или обмен сообщениями по некоторой дополнительной сети связи, независимой от графа. Во второй метамодели автоматы неподвижно находятся в вершинах, а роль каналов передачи сообщений играют рёбра графа, по которым эти сообщения передаются. Если граф ориентированный, то автоматы (в первой метамодели) или сообщения (во второй метамодели) перемещаются по рёбрам только в направлении их ориентации; в неориентированном графе рёбра могут проходиться в обоих направлениях.

Для того чтобы из вершины по инцидентному ей ребру (выходящему ребру в ориентированном графе) мог перемещаться автомат или сообщение, автомат в вершине должен это ребро указать. Для этого предполагается, что в неориентированном графе все рёбра, инцидентные вершине, а в ориентированном графе все рёбра, выходящие из вершины, перенумерованы. При перемещении по ребру указывается его номер. В ориентированном графе ребро имеет только один номер – в той вершине, из которой оно выходит, а в неориентированном графе ребро имеет два номера – по одному в каждой инцидентной ему вершине. Такой граф с нумерацией рёбер называется *упорядоченным*. В дальнейшем предполагается, что граф упорядочен.

Мы будем считать, что в графе выделена одна вершина, эту вершину будем называть *корнем* графа, а граф – *корневым*. С корня и начинается решение той или иной задачи на графе: автомат, находящийся в корне, получает извне графа (по «фиктивному» ребру, входящему в корень) сообщение, инициирующее решение задачи. После окончания решения задачи автомат корня также вовне (по «фиктивному» ребру, выходящему из корня) посылает сообщение с «ответом». Для того чтобы коллектив автоматов мог решать задачу на всём графе, такой граф, очевидно, должен быть связным.

Ранее мы изучали задачу обхода графа коллективом автоматов, когда должно быть пройдено каждое ребро графа: каким-нибудь автоматом в первой метамодели ([1],[2],[3],[4],[5]) или каким-нибудь сообщением во второй метамодели ([6],[7],[8],[9],[10],[11]). Для второй метамодели также решалась задача вычисления функции от мультимножества значений, записанных в вершинах графа ([7],[8],[10],[11]). Данная статья открывает серию статей, посвящённых решению коллективом автоматов других задач на графах, которые не сводятся к обходу графа, хотя могут использовать алгоритмы обхода. Мы будем исходить из второй метамодели: автоматы неподвижно «сидят» в вершинах графа, а сообщения пересылаются по рёбрам графа.

Распределённая система предполагается асинхронной. Это означает, что время перемещения сообщения по ребру может быть разным как для разных рёбер, так и в разные моменты времени для одного и того же ребра. Будем считать, чтоб время перемещения сообщения по ребру ограничено сверху.

Методы решения задач на графах коллективом автоматов существенно зависят от 1) типа графа, 2) типа автоматов, 3) типа решаемой задачи.

Граф может быть неориентированным, ориентированным и смешанным. В неориентированном графе предполагается использовать возможность посылки ответного сообщения по тому же ребру, по которому было получено прямое сообщение. Это позволяет существенно уменьшить время решения задачи. Этот же способ можно использовать в смешанном графе для его неориентированных рёбер. В ориентированном графе для ответного сообщения приходится искать обратный путь из конца ребра в его начало. Для этого могут использоваться прямой (ориентированный от корня) и обратный (ориентированный к корню) остовы графа ([6],[7],[8],[10],[11]).

Граф может быть детерминированным или недетерминированным. В недетерминированном графе несколько рёбер, выходящих из одной вершины, могут иметь один номер; множество таких рёбер называется *дельта-ребром*. При посылке сообщения выбор ребра из дельта-ребра с данным номером выполняется недетерминированным образом. Для того чтобы гарантировать возможность решения задачи на недетерминированном графе приходится налагать те или иные ограничения на недетерминизм графа (для первой метамодели см. [3],[5]). Примером такого ограничения может служить справедливый недетерминизм, гарантирующий передачу сообщения по данному ребру из данного дельта-ребра после конечного (не обязательно ограниченного) числа попыток.

Другой подход к обходу недетерминированного графа – это изменение самой цели обхода: вместо того, чтобы проходить по всем рёбрам, требуется пройти по всем дельта-рёбрам, то есть хотя бы по одному ребру в каждом дельта-ребре [12]. Это имеет практический смысл при тестировании программной реализации модели системы: в каждом состоянии системы мы пробуем подать на неё каждое тестовое воздействие хотя бы по одному разу. Если граф ориентирован, то для обхода всех его дельта-рёбер требуется дельта-связность графа (в ориентированном графе сильно-дельта-связность [12]). В частности, достаточно наличия в графе детерминированного остова (в ориентированном графе два остова: ориентированный от корня и ориентированный к корню).

Также граф может быть динамически меняющимся: его ребро может исчезать, появляться или менять один из своих концов. Для решения задач на динамическом графе приходится использовать какие-то ограничения на «скорость» изменения рёбер, чтобы гарантировать доставку сообщений в нужные вершины. Например, некоторые рёбра «долгоживущие», т.е. такое ребро не меняется в течение времени, которое достаточно для передачи по нему хотя бы одного сообщения ([9],[10],[11]).

Кроме этого, существенной характеристикой графа является наличие или отсутствие нумерации его вершин. Граф *нумерованный*, если все его вершины помечены различными идентификаторами, в частности, пронумерованы целыми числами 0, 1, 2, ... Нумерация графа даёт возможность различать вершины графа по их номерам, что существенно влияет на алгоритмы решения задач на графе. Во многих случаях для решения задачи полезно сначала выполнить нумерацию графа с помощью подходящего алгоритма, а уже потом решать задачу на нумерованном графе.

Специфической характеристикой асинхронной распределённой системы является *ёмкость ребра* – максимальное число сообщений, одновременно передаваемых по ребру, или их суммарный размер.

Автоматы характеризуются размером памяти и временем срабатывания. По размеру памяти автоматы делятся на *роботы*, *полароботы* и *неограниченные автоматы* в зависимости от размера их памяти, который определяется числом состояний, входных и выходных символов. Роботы – это конечные автоматы,

память которых ограничена и не зависит от размера графа. Память неограниченного автомата зависит от размера графа и достаточно велика, чтобы в ней могло размещаться описание всего графа. Память полуробота тоже растёт вместе с ростом размера графа, однако описание графа может не помещаться в память одного автомата, хотя обычно предполагается, что это описание помещается в суммарную память коллектива автоматов.

В [13] доказано, что для неориентированных упорядоченных корневых графов «граница» между неограниченным автоматом и полуроботом – это память размером $\Theta(mlogn)$ для нумерованных графов и $\Theta((m\text{-}n\text{+}1)logn\text{+}n)$ для ненумерованных графов, где m – число рёбер, а n – число вершин графа. Поскольку $(m\text{-}n\text{+}1)logn\text{+}n < mlogn$ при $n \geq 3$, для того чтобы автомат был полуроботом на классах нумерованных и ненумерованных графов, достаточно, чтобы память автомата имела размер $o((m\text{-}n\text{+}1)logn\text{+}n)$. В частности, $O(nlognm) = o((m\text{-}n\text{+}1)logn\text{+}n)$ для фиксированного (но произвольного) $n \geq 1$ и $m \to \infty$. Заметим, что полуробот на классе графов может быть неограниченным автоматом на подклассе. Тривиальный пример – автомат с памятью $\Theta(nlognm)$ на подклассе деревьев, где $m = n\text{-}1$ и поэтому $(m\text{-}n\text{+}1)logn\text{+}n = n = o(nlognm)$.

Для ориентированных графов нужны дополнительные исследования, чтобы определить «границу» между неограниченными автоматами и полуроботами.

Время срабатывания автомата – это время, за которое автомат принимает одно или несколько сообщений, изменяет своё состояние и посылает одно или несколько сообщений. В асинхронных системах обычно не учитывают время срабатывания автомата, измеряя сложность алгоритма временем перемещения сообщений. Важно также, ограничено или нет число сообщений, которые автомат принимает и/или посылает за одно срабатывание.

Понятно, что как сама возможность решения задачи на графе, так и оценка требуемых для этого ресурсов (время, память и число сообщений) зависят от типа автоматов.

Наконец, методы решения задач на графах, естественно, зависят от самих этих задач. Предполагается, что удастся провести классификацию задач в зависимости от того, какие сообщения между автоматами требуются для решения задачи. Для одних задач достаточно, чтобы из корня графа сообщение дошло до каждой вершины, а ответное сообщение вернулось в корень. Для других задач может потребоваться передача сообщений от каждой вершины в каждую вершину и обратно. Могут быть и такие задачи, которые требуют многократной передачи сообщений между вершинами графа. Для некоторых задач нужны дополнительные ограничения на графы или дополнительные возможности для автоматов и системы обмена сообщениями.

В анонсируемой этим введением серии статей мы предполагаем не только предложить алгоритмы решения тех или иных конкретных задач на графах коллективом автоматов, но и провести классификацию и систематизацию

методов решения этих задач с целью выработки общей методологии и базового набора распределённых алгоритмов решения типовых подзадач.

В данной статье мы ограничиваемся случаем неориентированных детерминированных статических графов.

В разделе 2 формально определяется вторая метамодель коллектива автоматов на графе, и обсуждаются различные модели в рамках этой метамодели в зависимости от размера памяти автомата, времени срабатывания и ёмкости рёбер. Выбирается модель с максимальным параллелизмом для полуроботов, которая используется в дальнейшем. Раздел 3 посвящён правилам оценки предлагаемых далее алгоритмов как функции от числа вершин и рёбер графа. В разделе 4 предложены базовые процедуры обработки сообщений, и дана классификация используемых сообщений в зависимости от маршрутов, проходимых сообщениями, и способа размножения или слияния сообщений. Вводимые здесь классы сообщений используются в предлагаемых далее распределённых алгоритмах. В разделе 5 описаны алгоритмы построения остова графа. В разделе 6 предложен алгоритм универсального «стопора», определяющего конец работы любого алгоритма по отсутствию в графе сообщений, используемых этим алгоритмом, а также процедура очистки графа от сообщений указанных типов. В разделе 7 представлен алгоритм построения дерева кратчайших путей. В разделе 8 описаны алгоритмы нумерации графа. В разделе 9 предлагается алгоритм сбора информации о графе в памяти неограниченного автомата корня, автоматы остальных вершин – полуроботы. Раздел 10 посвящён задаче поиска максимального пути в нумерованном графе, относящейся к классу *NP*. В заключении подводятся итоги и намечаются направления дальнейших исследований.

2. Классификация моделей и выбор модели

Как сказано во введении, в данной статье используется вторая метамодель асинхронной распределённой системы: автоматы «привязаны» к вершинам графа, а сообщения посылаются по рёбрам графа. Сообщение понимается как набор *параметров* сообщения, а состояние автомата – как набор значений *переменных*. Допуская вольность речи, мы будем часто для краткости говорить «вершина» вместо «автомат вершины».

Граф неориентированный, статический, упорядоченный, детерминированный, корневой и связный. Обозначения: n – число вершин графа, m – число рёбер графа, D – длина максимального пути в графе, D_0 – длина максимального пути от корня, d – диаметр графа, т.е. максимальное расстояние между вершинами, где расстояние между вершинами – это длина кратчайшего пути между вершинами, d_0 – эксцентриситет корня, т.е. максимальное расстояние от корня до вершины, Δ – максимальная степень вершины.

Очевидно, $n \leq m\text{+}1$, $d_0 \leq d \leq 2d_0$, $D_0 \leq D \leq 2D_0$, $d_0 \leq D_0$, $d \leq D \leq n\text{-}1$ и $\Delta \leq 2m$. Различие между n, d и D демонстрируют граф-звезда $S_{n\text{-}1}$, в котором $d = D = 2$

при $n > 2$, и полный граф K_n, в котором $d = 1$ и $D = n\text{-}1$. При $d = 0$ граф состоит из одной вершины с петлями, поэтому $n = 1$ и $D = 0$. При $d = 1$ любые две различные вершины связаны ребром, поэтому $D = n\text{-}1$. Для любых d, D и n, удовлетворяющих условию $2 \le d \le D \le n\text{-}1$, существует граф с параметрами d, D и n (см. Рис. 1). В графе без кратных рёбер и петель $m \le n(n\text{-}1)/2$ и $\Delta \le n\text{-}1$.

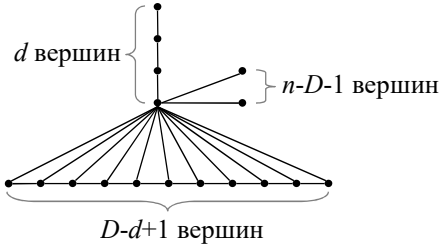


Рис. 1. Граф с n вершинами, диаметром $d \ge 1$ и длиной D максимального пути.
Fig. 1. A graph with n vertices, diameter $d \ge 1$, and length D of the maximal path.

Ребро графа будет пониматься как дуплексный канал передачи сообщений. При этом сообщения, перемещаемые по ребру в одном направлении, не генерируются самим ребром, не искажаются, не пропадают и не обгоняют друг друга. Иными словами, ребру *ab* соответствуют две очереди сообщений: посланных по этому ребру вершиной *a*, но ещё не принятых с него вершиной *b*, и, наоборот, посланных вершиной *b*, но ещё не принятых вершиной *a*.

Будем исходить из следующих предположений: 1) время перемещения сообщения по ребру ограничено сверху 1 тактом, 2) время срабатывания (автомата) вершины ограничено сверху 1 тактом, 3) за одно срабатывание вершина принимает не более одного сообщения и посылает не более одного сообщения. Третье предположение нуждается в пояснении. Сообщение может быть принято, если оно «дошло» по ребру до вершины, т.е. истекло время перемещения этого сообщения по ребру. Однако, поскольку сообщения, перемещаемые по ребру из вершины *a* в вершину *b*, образуют очередь, вершина *b* может принять только сообщение из головы этой очереди. Если есть несколько рёбер, по которым вершина может принять сообщения, то выбирается одно из них недетерминированным образом. Вместе с принимаемым сообщением вершина получает номер ребра, по которому это сообщение принимается. Если нет сообщений, которые вершина могла бы принять, она получает фиктивное пустое сообщение ε с нулевым номером ребра (рёбра в вершине нумеруются, начиная с 1). Если вершина не посылает сообщение, то указывается пустое сообщение ε с нулевым номером ребра.

Предполагается, что вершина «знает» (или может «узнать») свою степень, которая в дальнейшем хранится в переменной вершины. В некоторых алгоритмах граф предполагается *нумерованным*: его вершинам приписаны уникальные идентификаторы, в частности числа от 0 до n-1. Если граф

нумерованный, то предполагается, что вершина «знает» (или может «узнать») свой номер. Если граф ненумерованный, его можно пронумеровать, используя алгоритм нумерации из раздела 8. В любом случае будем предполагать, что для нумерованного графа номер вершины хранится в переменной вершины.

Ёмкостью ребра будем называть максимальное число сообщений, которые могут одновременно находиться на ребре (посланы, но не приняты) в одном направлении. Если ребро *ab* имеет ограниченную ёмкость *k* и в направлении от вершины *a* к вершине *b* по нему перемещается *k* сообщений, то вершина *a* не может посылать сообщение по ребру *ab* до тех пор, пока вершина *b* не примет с ребра хотя бы одно сообщение. Для того чтобы вершина смогла «узнать», можно или нельзя посылать по ребру сообщение, при ограниченной ёмкости ребра нужен сигнал «*освобождение ребра*» ϕ с параметром «номер ребра». Если ёмкость ребра неограниченная, сообщение по ребру всегда можно посылать, и такого сигнала не требуется.

Мы исходим из того, что вершина, принимая сообщение, размещает его в своей памяти, а для посылки сообщения вершина сначала формирует его в своей памяти. Тем самым, нужно учитывать не только суммарный размер переменных вершины, который мы будем называть *размером автомата* вершины, но и его сумму с максимальным размером сообщения, которую мы будем называть *полным размером автомата*.

Как указано во введении, в зависимости от их полного размера автоматы делятся на неограниченные автоматы, роботы и полуроботы. Рассмотрим различные модели в рамках данной метамодели в зависимости от полного размера автомата, времени срабатывания и ёмкости ребра (см. табл. 1).

Если время срабатывания больше нуля, то это может приводить к тому, что на ребрах, инцидентных вершине, скапливаются сообщения, которые уже завершили перемещение по этим рёбрам, но вершина не успевает их принять. Поэтому нужно учитывать такие задержки, и оценка времени работы становится достаточно сложной.

Если ёмкость ребра ограничена числом *k*, то это может приводить к тому, что вершина, инцидентная ребру, не может посылать сообщения по нему, если ребро полностью занято (на нём уже находится *k* сообщений), пока ребро не освободится. Поэтому при оценке времени работы нужно учитывать такие задержки в пересылке сообщений, и оценка становится достаточно сложной (в качестве примера, можно посмотреть оценку времени работы алгоритма в [8]).

Табл. 1. Классификация моделей
Tab. 1. Classification of models

Модель	Полный размер автомата	Время срабатывания	Ёмкость ребра
1	неограниченный автомат	> 0	не ограничена
2	неограниченный автомат	= 0	ограничена
3	неограниченный автомат	> 0	ограничена
4	неограниченный автомат	= 0	не ограничена

5	робот	> 0	не ограничена
6	робот	= 0	ограничена
7	робот	> 0	ограничена
8	робот	= 0	не ограничена
9	полуробот	> 0	не ограничена
10	полуробот	= 0	ограничена
11	полуробот	> 0	ограничена
12	полуробот	= 0	не ограничена

Формально, автомат в вершине – это набор (Mes,S,Tr,s_0), где Mes – множество всех непустых сообщений, которые автомат может принимать или посылать, S – множество состояний, Tr – множество переходов, s_0 – начальное состояние. Переход имеет вид s—(i,x),(j,y)→$s`$, $s$—($i$,$x$),(0,ε)→$s`$, s—(i,ϕ),(j,y)→$s`$, $s$—($i$,ϕ),(0,ε)→$s`$, s—(0,ε),(j,y)→$s`$, или $s$—(0,ε),(0,ε)→$s`$, где $s∈S$ – состояние, в котором выполняется переход, ε – пустое сообщение, ϕ – сигнал «освобождение ребра», i – номер ребра, по которому принимается непустое сообщение или сигнал ϕ, $x∈Mes$ – принимаемое сообщение, j – номер ребра, по которому посылается непустое сообщение, $y∈Mes$ – посылаемое сообщение, $s`∈S$ – состояние, в которое автомат переходит.

Если полный размер автомата ограничен (робот или полуробот) и ёмкость ребра тоже ограниченная, то возможны состояния, где вершина не принимает сообщений, поскольку не может послать сообщения. Вершина ожидает только сигналов освобождения рёбер, т.е. в таких состояниях определены только переходы вида s—(i,ϕ),(i,y)→$s`$ или $s$—($i$,ϕ),(0,ε)→$s`$. В остальных состояниях вершина v принимает любое сообщение, т.е. её автомат «всюду определён по приёму сообщений»: в каждом таком состоянии определены переходы по всем возможным парам (i,x), (i,ϕ), (0,ε), где i=1..Δ_v, $x∈Mes$ и Δ_v – степень вершины v.

2.1. Неограниченный автомат

Случай неограниченного автомата (модели 1-4) не очень интересен: решение любой задачи можно свести к следующей процедуре. Сначала информация о графе собирается в корне графа, а затем автомат корня решает задачу. В разделе 9 мы опишем алгоритм выполнения процедуры сбора информации.

2.2. Робот

В литературе имеется ряд работ по обходу графа роботом. Для неориентированного графа известен алгоритм Тэрри, основанный на обходе в глубину (DFS), с оценкой времени работы $2m$, и алгоритм, основанный на обходе в ширину (BFS), с оценкой $m+\boldsymbol{O}(n^2)$ ([14]). Для ориентированного графа наилучшая полученная оценка равна $\boldsymbol{O}(nm+n^2 log log n)$ или, выражая через D_0, $\boldsymbol{O}(D_0 m+D_0 n log log n)$ ([14],[15]). При повторном обходе графа, когда автоматы могут использовать результаты первого обхода, оценка

$\boldsymbol{O}(nm+n^2 log^* n)$ или $\boldsymbol{O}(D_0 m+D_0 n log^* n)$, где log^* – итерированный логарифм (число логарифмирований, после которых результат меньше или равен 1) [16]. Если имеются два робота, передвигающихся по графу и обменивающихся между собой сообщениями по дополнительной сети связи (первая метамодель), то время обхода уменьшается до минимального по порядку $\boldsymbol{O}(nm)$ или, выражая через D_0, $\boldsymbol{O}(D_0 m)$ ([14]).

Исследование графов роботами имеет целый ряд специфических особенностей, вызываемых ограниченной памятью робота. В настоящей статье мы не исследуем решение задач коллективом роботов.

2.3. Полуробот

Как и в общем случае, для полуроботов ненулевое время срабатывания и/или ограниченная ёмкость ребра приводят к задержкам в пересылке сообщений, что усложняет оценку времени работы алгоритмов. Но в случае полуроботов (как и роботов) ограниченная ёмкость ребра может также приводить к *тупику* (deadlock). Дело в том, что, если ребро полностью заполнено, и вершина не может по нему послать сообщение, то, поскольку полный размер автомата ограничен (хотя и зависит от размера графа для полуробота), может оказаться, что вершина приостановит приём сообщений. На цикле в графе может возникнуть тупик, когда каждая вершина цикла «хочет» послать сообщение по «выходящему» ребру цикла, но не может этого сделать, и поэтому не принимает сообщение по «входящему» ребру цикла. Алгоритмы решения задач на графе должны учитывать возможность тупика и не допускать тупика, что, конечно, налагает дополнительные требования к построению алгоритмов.

2.4. Выбор модели

Для серии статей, открываемой данной статьёй, мы выбираем модель 12: ёмкость ребра неограниченная, автомат – полуробот с нулевым временем срабатывания. Время перемещения сообщения по ребру ограничено сверху 1 тактом. Такая модель даёт возможность максимального распараллеливания, поскольку не возникает задержек и тупиков из-за ограниченной ёмкости ребра или ненулевого времени срабатывания. Тем самым, будет учитываться только время перемещения сообщений по рёбрам. Оценки сложности алгоритмов решения задач в такой модели являются нижними границами сложности аналогичных алгоритмов в других моделях с ограниченными автоматами (модели 5-11). Алгоритм в модели 12 либо будет работать медленнее в моделях 5-11, либо не будет работать из-за тупиков или размера памяти в моделях 5-8 для роботов. Однако любой алгоритм в моделях 5-11 будет работать и не большее время в модели 12.

Отметим, что при нулевом времени срабатывания не имеет смысла принимать или посылать больше одного сообщения за одно срабатывание, поскольку это

не уменьшает времени работы, но приводит к увеличению полного размера автомата за счёт хранения в его памяти сразу нескольких сообщений.

Наша модель отличается от хорошо известной модели распределённых вычислений LOCAL [17] асинхронностью и наличием корня. В модели LOCAL все автоматы в вершинах работают синхронно и начинают работать одновременно, поэтому сложность алгоритма в этой модели – это число срабатываний. В нашей модели работа начинается с корня, другая вершина включается в работу при получении ею первого сообщения. Кроме того, при оценке сложности алгоритма мы считаем концом работы алгоритма не момент времени, когда задача решена, а момент времени, когда об этом «узнал» корень и сообщил вовне. Асинхронность моделируется не разными временами срабатываний автоматов, а разными и, вообще говоря, переменными временами перемещения сообщений по рёбрам, хотя и ограниченными сверху 1 тактом. Это приводит к тому, что по ребру может передаваться не одно сообщение, а последовательность сообщений, посланных с одного конца ребра и ещё не принятых другим концом.

Для нашей модели, очевидно, получается нижняя оценка $2d_0+1$ сложности любого алгоритма, не имеющего предварительных знаний о графе (вершина знает только свою степень и, для нумерованного графа, свой номер) и решающего глобальную задачу на графе. Задача глобальна, если она не может быть решена без исследования каждого ребра графа. Такое исследование требует, очевидно, времени d_0+1 в наихудшем случае, когда по каждому ребру сообщение двигается ровно 1 такт. Для того чтобы сообщить корню о завершении работы, требуется ещё не менее d_0 тактов в наихудшем случае.

Следует, однако, учитывать, что неограниченная ёмкость ребра и нулевое время срабатывания в некоторых алгоритмах может приводить к неограниченному росту числа сообщений, одновременно циркулирующих в графе и, более того, одновременно перемещающихся по одному ребру. Иными словами, решение задачи на графе может использовать неограниченные ресурсы памяти за счёт сообщений. В этом смысле наша модель сближается с недетерминированной машиной Тьюринга, которая неограниченно (экспоненциально) копируется по дереву вычислений. Поэтому оценки сложности предлагаемых нами алгоритмов решения некоторых задач могут оказаться полиномиальными, хотя эти задачи относятся к классу *NP*. Примером может служить задача о поиске максимального пути в графе (*Longest Path Problem* [18]), которой посвящен раздел 10 данной статьи.

3. Оценка алгоритмов

Обозначения в оценках алгоритмов: M – максимальный размер сообщения, A – размер автомата (сумма размеров переменных), $F = M+A$ – полный размер автомата, T – время работы алгоритма, N – максимальное число сообщений, одновременно передаваемых по одному ребру в одном направлении, и E – максимальный суммарный размер таких сообщений. Очевидно, $E \le NM$.

Для оценки размера памяти от двух переменных n и m будем считать, что n любое, но фиксированное, достаточно большое число, а $m\to\infty$. Будем писать: $f(n,m) = \boldsymbol{o}_{m\to\infty}(g(n,m))$, если $\exists n_0\ \forall n \ge n_0\ \forall C > 0\ \exists m_0\ \forall m \ge m_0\ f(n,m) \le Cg(n,m)$, что в нашем случае эквивалентно $\exists n_0\ \forall n \ge n_0\ lim_{m\to\infty}(f(n,m)/g(n,m)) = 0$.

Тем не менее, там, где это возможно, будем применять оценки памяти, когда независимо $n\to\infty$ и $m\to\infty$ при условии связности графа $m \ge n$-1. Будем писать: $f(n,m) = \boldsymbol{o}_{n,m\to\infty}(g(n,m))$, если $\forall C > 0\ \exists n_0\ \forall n \ge n_0\ \forall m \ge n\text{-}1\ f(n,m) \le Cg(n,m)$.

Согласно [13], если $F = \boldsymbol{o}_{m\to\infty}(m\log n)$, то автомат полуробот на классе нумерованных графов, а если $F = \boldsymbol{o}_{m\to\infty}(n+(m\text{-}n+1)\log n)$, то автомат полуробот на классе ненумерованных графов. Класс ненумерованных графов можно рассматривать как подкласс класса нумерованных графов, а полуробот на подклассе является полуроботом на классе. Далее для краткости будем обозначать: $\boldsymbol{o}_{n,m\to\infty} = \boldsymbol{o}_{n,m\to\infty}(n+(m\text{-}n+1)\log n)$, $\boldsymbol{o}_{m\to\infty} = \boldsymbol{o}_{m\to\infty}(n+(m\text{-}n+1)\log n)$.

Конечно, на подклассе графов, где $m = \boldsymbol{O}(n)$, в частности, для деревьев $m = n$-1, уже не верно, что $\boldsymbol{O}(n\log nm) = \boldsymbol{o}_{m\to\infty}$, поскольку m не может бесконечно расти (для деревьев m не меняется) при фиксированном n. На таких подклассах могут существовать алгоритмы с меньшей памятью автоматов. Однако нас интересуют алгоритмы, работающие на классе всех графов, и они могут иметь «лишнюю» память на подклассах.

Для корректного использования некоторых наших оценок достаточно, чтобы на рассматриваемом классе графов число рёбер m могло расти быстрее, чем число вершин n, т.е. на некоторой последовательности графов $m/n\to\infty$. Обозначая $h(n) = m/n$, имеем $m = nh(n)$. Если $lim_{n\to\infty}h(n) = \infty$ влечёт $lim_{n\to\infty}(f(n,m)/g(n,m)) = 0$, то будем писать $f(n,m) = \boldsymbol{o}_{m/n\to\infty}(g(n,m))$. Пример: если $m = n\log n$, то $m = \boldsymbol{o}_{m/n\to\infty}(n^2)$. Обозначим $\boldsymbol{o}_{m/n\to\infty} = \boldsymbol{o}_{m/n\to\infty}(n+(m\text{-}n+1)\log n)$. Автоматы с памятью $\boldsymbol{o}_{m/n\to\infty}$ являются полуроботами на классе ненумерованных графов.

На подклассе графов без кратных рёбер и петель с n вершинами оценки зависят только от n, а m ограничено сверху по порядку n^2. Согласно [13] на этом подклассе автомат полуробот, если $F = \boldsymbol{o}(n^2\log n)$. Для краткости обозначим $\boldsymbol{o}_{n\to\infty} = \boldsymbol{o}(n^2\log n)$. Если F имеет эту оценку на подклассе графов без кратных рёбер и петель, будем указывать её в квадратных скобках $[\boldsymbol{o}_{n\to\infty}]$.

Докажем следующие основные отношения:

1) $\log m = \boldsymbol{o}_{n,m\to\infty}$, 2) $n\log nm = \boldsymbol{o}_{m\to\infty}$ и $n^2\log m = \boldsymbol{o}_{m\to\infty}$,
3) $m = \boldsymbol{o}_{m/n\to\infty}$, 4) $n\log n = \boldsymbol{o}_{n\to\infty}$ и $n^2 = \boldsymbol{o}_{n\to\infty}$.

1) Пусть $C > 0$. Тогда для $n \ge 1/C+3$ имеем $n \ge e$, что для $k \ge 0$ влечёт $n^k \ge e^k = (e^{kC})^{1/C}$ и, поскольку $e^x \ge 1+x$, имеем $n^k \ge (e^{kC})^{1/C} \ge (1+\kappa C)^{1/C} \ge (1+k/(n\text{-}3))^{1/C} \ge (1+k/(n\text{-}1))^{1/C} = ((n\text{-}1+k)/(n\text{-}1))^{1/C}$. Отсюда $k\log n \ge (1/C)\log(n\text{-}1+k)\text{-}(1/C)\log(n\text{-}1)$, что влечёт $n+k\log n\text{-}(1/C)\log(n\text{-}1+k) \ge n\text{-}(1/C)\log(n\text{-}1)$. Поскольку в связном графе $m\text{-}n+1 \ge 0$, выберем $k = m\text{-}n+1$. Поэтому $n+(m\text{-}n+1)\log n\text{-}(1/C)\log m \ge$

$\geq n\text{-}(1/C)log(n\text{-}1)$. Поскольку $lim_{n\to\infty}(n/((1/C)log(n\text{-}1))) = \infty$, найдётся такое $n_0 \geq 1/C+3$, что для каждого $n \geq n_0$ будет $n\text{-}(1/C)log(n\text{-}1) \geq 0$. Тем самым, для $n \geq n_0$ будет $n+(m\text{-}n+1)logn\text{-}(1/C)logm \geq 0$, что влечёт $logm \leq C(n+(m\text{-}n+1)logn)$. Тем самым, $logm = \boldsymbol{o}_{n,m\to\infty}$.

2) $nlognm = \boldsymbol{o}_{m\to\infty}$ доказано в [13]. При любом, но фиксированном $n \geq 2$, имеем $lim_{m\to\infty}((n^2logm)/(n+(m\text{-}n+1)logn)) = 0$. Следовательно, $n^2logm = \boldsymbol{o}_{m\to\infty}$.

3) Пусть $m = nh(n)$ и $lim_{n\to\infty}h(n) = \infty$. Тогда $lim_{n\to\infty}(m/(n+(m\text{-}n+1)logn)) =$
$= lim_{n\to\infty}(m/(n+mlogn\text{-}nlogn+logn)) = lim_{n\to\infty}(h(n)/(1+h(n)logn\text{-}logn+(logn)/n)) =$
$= lim_{n\to\infty}(h(n)/(1+h(n)logn\text{-}logn+0)) = lim_{n\to\infty}(h(n)/(1+h(n)logn\text{-}logn)) =$
$= lim_{n\to\infty}((1/logn)/(1/(h(n)logn)+1\text{-}1/h(n))) = 0/(0+1\text{-}0) = 0$. Поэтому $m = \boldsymbol{o}_{m/n\to\infty}$.

4) $lim_{n\to\infty}((nlogn)/(n^2logn)) = lim_{n\to\infty}(1/n) = 0$. Поэтому $nlogn = \boldsymbol{o}_{n\to\infty}$.

$lim_{n\to\infty}((n^2)/(n^2logn)) = lim_{n\to\infty}(1/logn) = 0$. Поэтому $n^2 = \boldsymbol{o}_{n\to\infty}$.

В дальнейшем при оценках мы будем также учитывать, что $d_0 \leq d \leq D \leq n\text{-}1$, $d_0 \leq D_0 \leq D$, $\Delta \leq 2m$, а для графа без кратных рёбер и петель $\Delta \leq n\text{-}1$.

Когда алгоритм $\mathcal{A}$ использует результат работы другого алгоритма $\mathcal{B}$, то иногда даётся оценка только алгоритма $\mathcal{A}$ без учёта ресурсов, используемых алгоритмом $\mathcal{B}$. Если нужна суммарная оценка, то величины M, A, F, T и E суммируются, а для величины N берётся максимум.

4. Классификация сообщений

В предлагаемых алгоритмах будут использоваться некоторые базовые процедуры обработки сообщений в вершинах. Каждой процедуре соответствует класс сообщения, где под классом сообщения понимается не функциональное предназначение сообщения, а способ его передачи по графу: маршрут, который проходит сообщение, и способ «размножения» или «слияния» сообщений. Те параметры сообщения и переменные вершины, которые существенны для обработки сообщений данного класса, будем называть, соответственно, базовыми параметрами и базовыми переменными. Временем работы алгоритма базовой процедуры будет считаться время распространения сообщения от его создания до уничтожения.

Для любого класса базовым параметром является тип сообщения. Сообщение данного типа, попадая в вершину, далее либо 1) уничтожается, после чего, возможно, из вершины посылается одно или несколько сообщений уже другого типа, либо 2) пересылается вершиной дальше по одному или нескольким ребрам, быть может, с модификацией содержимого сообщения, но не его типа. Если принимается одно сообщение, а потом посылаются сообщения того же типа по нескольким ребрам, то это будем называть *размножением* сообщения. Если принимается несколько сообщений одного типа, быть может за несколько срабатываний, а потом посылается сообщение того же типа по одному ребру, то это будем называть *слиянием* сообщений.

Сообщение данного типа однозначно определяется маршрутом, который оно прошло: для любого маршрута одновременно существует не более одного сообщения этого типа, которое прошло этот маршрут. При размножении получаются сообщения одного типа, маршруты которых имеют одинаковый префикс, заканчивающийся в вершине, где произошло размножение. При слиянии нескольких сообщений одного типа маршрут каждого сообщения, кроме одного, заканчивается в вершине, где произошло слияние.

Таким образом, класс сообщения определяется способом размножения или слияния сообщений и маршрутом, которое сообщение проходит до уничтожения; такой маршрут будем называть маршрутом класса. Мы определим девять базовых классов сообщений, иллюстрируемых Рис. 2.

Формально *маршрутом* называется чередующаяся последовательность вершин и ребер, начинающаяся и заканчивающаяся вершиной, в которой каждое i-ое ребро инцидентно предыдущей i-ой и последующей i+1-ой вершинам. Номер i-ой вершины маршрута будем обозначать v_i, номер i-ого ребра в i-ой вершине будем называть *прямым номером* и обозначать f_i, а его номер в i+1-ой вершине будем называть *обратным номером* и обозначать r_i.

Сообщение может накапливать в себе пройденный им маршрут в виде последовательности $P = \langle v_1,f_1,r_1,v_2,f_2,r_2,\ldots v_k,f_k,r_k,v_{k+1},f_{k+1}\rangle$ номеров пройденных вершин и прямых и обратных номеров рёбер, а сообщение передаётся из вершины v_{k+1} по ребру f_{k+1}. Когда такое сообщение создаётся вершиной v_1 и посылается по ребру f_1, в сообщении параметр $P = \langle v_1,f_1\rangle$. Когда вершина v_i по ребру r_{i-1} получает сообщение с параметром $P = \langle v_1,f_1,r_1,v_2,f_2,r_2,\ldots v_{i-1},f_{i-1}\rangle$ и посылает сообщение далее по ребру f_i, в посылаемом сообщении параметр $P := P{\cdot}\langle r_{i-1},v_i,f_i\rangle$. Если какие-то номера не нужно накапливать: номера вершин, прямые номера рёбер или обратные номера рёбер, – то в конец P не добавляется соответствующий номер, и последовательность P не содержит эти номера. Будем называть вариант накопления *x-накоплением*, а накопленную последовательность P – *x-вектором маршрута*, где x определяется регулярным выражением $x = v\,?\,f\,?\,r\,?$ (знак «?» означает число повторений предыдущего символа 0 или 1 раз). По умолчанию накопления нет (x пусто).

В данной статье *хордой подграфа* будем называть ребро, оба конца которого принадлежат подграфу, но само ребро не принадлежит подграфу. *Путём* называется маршрут без самопересечения, т.е. такой маршрут длины k, что $\forall i,j=1..k+1$ $(i\neq j \Rightarrow v_i\neq v_j)$. Иногда такой маршрут называют простым путём. *Терминальной вершиной* будем называть вершину степени 1. *Путём с хордой* назовем маршрут, который состоит из пути, продолженного хордой этого пути, т.е. имеет вид $P{\cdot}\langle e,v\rangle$, где P – путь, e – хорда пути P, инцидентная конечной вершине пути и вершине v. *Путём с ребром* назовем маршрут, являющийся путём или путём с хордой. Очевидно, длина пути не превосходит D, а если начало маршрута – корень, то D_0. Максимальная длина пути с хордой на 1 больше. Для корневого дерева *листом* или *листовой вершиной* дерева называют терминальную вершину дерева, отличную от корня, а

внутренней вершиной дерева – его вершину, не являющуюся листом или корнем дерева. Корнем остова графа, будем считать корень графа.

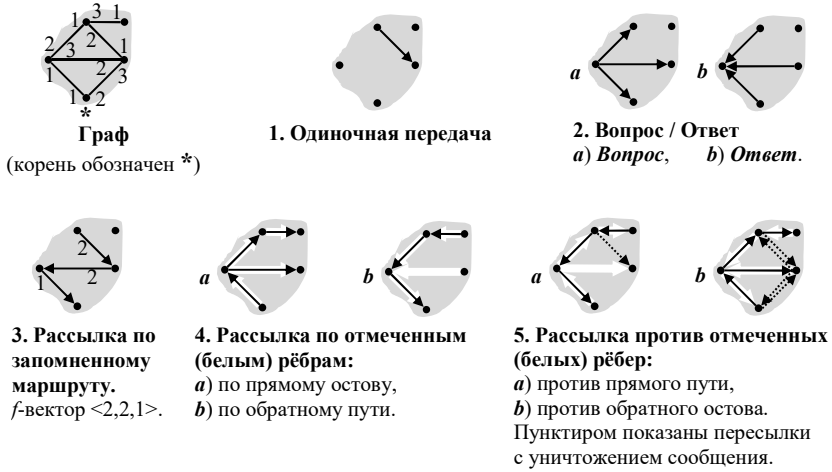


Рис. 2. Базовые классы сообщений
Fig. 2. Basic message classes

Размеченным графом назовём такой граф, в некоторых вершинах которого отмечены некоторые инцидентные им рёбра со следующими ограничениями: 1) каждое ребро отмечено не более чем в одной из инцидентных ему вершин,

и отмеченное ребро считается условно ориентированным от этой вершины; 2) условно ориентированные отмеченные ребра образуют ациклический граф. Очевидно, длина пути по отмеченным рёбрам не превосходит D, а если один из концов пути корень, то D_0. Будем говорить, что отмеченное ребро *выходит из вершины*, если оно отмечено в этой вершине, и *входит в вершину*, если оно инцидентно этой вершине, но не отмечено в ней.

Нам понадобятся два важных частных случая размеченного графа, когда отмеченное множество рёбер порождает лес деревьев, и в каждом дереве выделен его корень. Если все рёбра леса отмечены так, что каждое дерево условно ориентировано от корня (к корню) дерева, то такой размеченный граф будем называть *прямым* (*обратным*) *лесом*. Если лес состоит из одного дерева, являющегося остовом графа, то прямой (обратный) лес будем называть *прямым* (*обратным*) *остовом*. Рёбра прямого (обратного) леса будем называть *прямыми* (*обратными*) *рёбрами*, а путь из прямых (обратных) рёбер будем называть *прямым* (*обратным*) *путём*.

При описании классов сообщений *специальный критерий* понимается как некоторое условие, основанное на значениях переменных вершины.

В дальнейшем мы будем описывать алгоритмы, составленные как из «кирпичиков» из нескольких алгоритмов, в том числе, из базовых процедур, быть может, с какими-то их модификациями. Там, где будет возникать коллизия совпадающих имён переменных или типов сообщений из разных комбинируемых алгоритмов или при разных модификациях одного и того же алгоритма, будем предполагать систематическое переименование этих имён и типов, не оговаривая этого специально.

4.1. Одиночная передача

Сообщение этого класса двигается по одному ребру, после чего уничтожается.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$. Базовые переменные: *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$.

Вершина может создать сразу несколько сообщений этого типа и одновременно послать их по нескольким инцидентным рёбрам: по всем или по тем, которые удовлетворяют тому или иному специальному критерию. Также сообщение этого типа может одновременно создаваться в нескольких вершинах. В любом случае после создания всех таких сообщений по каждому ребру в каждом направлении проходит не более одного сообщения.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. $T \le 1$. $N \le 1$. $E = \boldsymbol{O}(1)$.

4.2. Вопрос/Ответ

В этой процедуре используются два типа сообщений: ***Вопрос*** и ***Ответ***.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$. Базовые переменные: *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$, *СчётчикОтветов* – изменяется от *Степень* до 0, имеет размер $\boldsymbol{O}(log\Delta)$.

Вершина посылает ***Вопрос*** по каждому инцидентному ей ребру, после чего ожидает получения сообщений ***Ответ*** по каждому такому ребру. Предполагается, что по каждому ребру должен придти ровно один ***Ответ***. Сначала *СчётчикОтветов* = *Степень*, а потом счётчик уменьшается на 1 при получении сообщения ***Ответ***. Когда счётчик становится равным 0, он снова инициализируется *СчётчикРёбер* := *Степень* для дальнейшего использования. Получив ***Вопрос*** по некоторому ребру, вершина посылает по этому же ребру в обратном направлении ***Ответ***. По каждому ребру в каждом направлении проходит не более одного ***Вопроса*** и не более одного ***Ответа***; причём ***Ответ*** посылается по ребру только после получения по этому ребру ***Вопроса***.

Возможны модификации этого алгоритма, когда ***Вопрос*** посылается только по тем рёбрам, которые удовлетворяют тому или иному специальному критерию, и вместо *Степень* используется число таких рёбер.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2$. $N \le 2$. $E = \boldsymbol{O}(1)$.

4.3. Пересылка по запомненному маршруту

Сообщение содержит в себе маршрут, по которому оно должно пройти, в виде *f*-вектора маршрута. Сообщение не размножается, а уничтожается тогда, когда проходит запомненный в нём маршрут.

Базовые параметры: 1) *Тип* – тип сообщения размером $\boldsymbol{O}(1)$, 2) *Маршрут* – *f*-вектор $P = \langle f_1, f_2, ... f_k \rangle$ ещё не пройденной части маршрута размером $\boldsymbol{O}(klog\Delta)$. Базовых переменных нет.

Пока *f*-вектор имеет вид $P = \langle f_1, f_2, ... f_k \rangle$, где $k > 0$, из сообщения удаляется номер ребра f_1, т.е. $P := \langle f_2, ... f_k \rangle$, и сообщение посылается по ребру f_1. Если $k = 0$, то сообщение уничтожается.

Оценка. Тогда $M = \boldsymbol{O}(klog\Delta)$. $A = \boldsymbol{O}(1)$. $F = \boldsymbol{O}(klog\Delta)$. Для $k = \boldsymbol{O}(D)$ имеем $F = \boldsymbol{O}(Dlog\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le k$. Сообщение не размножается, поэтому $N \le 1$. $E = \boldsymbol{O}(klog\Delta)$.

4.4. Рассылка по отмеченным рёбрам

Граф размеченный. Маршрут этого класса является путём, состоящим из отмеченных рёбер. Ребро проходится по его условной ориентации: из вершины, в которой ребро отмечено, в вершину, в которой оно не отмечено.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$. Базовых переменных нет, не считая отметок рёбер.

Рассылка сообщений начинается с одной или нескольких вершин-инициаторов. Сообщение двигается по отмеченным рёбрам. Обычно при размножении создается столько сообщений, сколько из вершины выходит отмеченных рёбер и сообщения посылаются по этим рёбрам. Возможно также, что сообщение посылается не по всем таким рёбрам, а только по тем, которые удовлетворяют тому или иному специальному критерию. Обычно сообщение

уничтожается в стоке ациклического подграфа отмеченных рёбер, т.е. в такой вершине, из которой не выходят отмеченные рёбра. Возможно также, что сообщение уничтожается раньше по тому или иному специальному критерию.

Нам понадобятся два важных частных случаях: рассылка по прямому лесу и пересылка по обратному пути. Если подграф отмеченных рёбер – это прямой лес, то рассылка обычно начинается в корне дерева этого леса; в каждой внутренней вершине степени больше 2 происходит размножение сообщения. Число сообщений равно числу листьев деревьев прямого леса. Если же подграф отмеченных рёбер – это обратный лес, то размножения не происходит, поскольку из каждой вершины выходит не более одного обратного ребра. Сообщение двигается по обратным рёбрам до корня дерева.

Оценка. $M = \boldsymbol{O}(1)$. Не считая отметок рёбер, $A = \boldsymbol{O}(1)$ и $F = \boldsymbol{O}(1) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le D$ ($T \le D_0$ при рассылке от корня или если корень единственный сток подграфа отмеченных рёбер). Для прямого леса, когда инициаторы – это корни деревьев, $N \le 1$ и $E = \boldsymbol{O}(1)$, для обратного леса, когда инициаторы – это листья деревьев, $N \le n\text{-}1$ и $E = \boldsymbol{O}(n)$.

4.5. Рассылка против отмеченных рёбер

Граф размеченный. Маршрут этого класса состоит из отмеченных рёбер, но проходимых против их условной ориентации: из вершины, в которой ребро не отмечено, в вершину, в которой оно отмечено. Точнее, сообщение пересылается и по рёбрам, не отмеченным в их другом конце, но после такой пересылки сообщение сразу уничтожается. Не считая таких пересылок, сообщение двигается по ациклическому графу отмеченных рёбер в направлении, обратном условной ориентации этих рёбер.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$. Базовые переменные (не считая отметок рёбер): *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$.

Рассылка сообщений начинается с одной или нескольких вершин-инициаторов. Инициатор рассылает сообщение по всем инцидентным ему рёбрам, не отмеченным в нём. Вершина, получив сообщение по некоторому ребру, проверяет, отмечено ли в ней это ребро. Если ребро не отмечено, сообщение уничтожается. Если ребро отмечено, сообщение пересылается дальше по всем инцидентным вершине неотмеченным рёбрам. Возможно также, что сообщения посылаются только по тем рёбрам, которые удовлетворяют тому или иному специальному критерию. Сообщение уничтожается после того, как оно, пройдя по ребру, попало в вершину, где это ребро не отмечено. В частности, всегда уничтожается сообщение, посланное из истока ациклического подграфа отмеченных рёбер. Возможно также, что сообщение уничтожается раньше по тому или иному специальному критерию. Например, если вершина «знает», что она является истоком, то она может не посылать дальше сообщение, а сразу уничтожать его. Отмеченное ребро будет пройдено один раз в одном направлении – против его условной ориентации.

Не отмеченное ребро будет пройдено по одному разу в каждом направлении (кроме рёбер, инцидентных истокам, если истоки «знают», что они истоки).

Наиболее важные частные случаи размеченного графа – прямой и обратный лес. В случае обратного леса инициатором является обычно корень дерева. Сообщение двигается по рёбрам дерева от корня до листьев с размножением. Если вершина «знает», что она лист дерева, сообщение сразу уничтожается. В случае прямого леса инициатор – это, как правило, лист дерева. Сообщение двигается по рёбрам дерева от листа до корня без размножения. Если вершина «знает», что она корень дерева, она сразу уничтожает сообщение.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. $T \le D+1$ или $T \le D$, если истоки «знают», что они истоки. При рассылке от корня или к корню, соответственно, $T \le D_0+1$ или $T \le D_0$. Для обратного леса, когда инициаторы – это корни деревьев, $N \le 1$ и $E = \boldsymbol{O}(1)$, для прямого леса, когда инициаторы – это листья деревьев, $N \le n-1$ и $E = \boldsymbol{O}(n)$.

4.6. Сбор по обратному дереву

Граф размеченный, подграф отмеченных рёбер является обратным лесом.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$. Базовые переменные: *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$, *ОбратноеРебро* – номер выходящего обратного ребра (в корнях деревьев равен 0) размером $\boldsymbol{O}(log\Delta)$, *ЧислоРёбер* – число входящих обратных рёбер размером $\boldsymbol{O}(log\Delta)$, *СчётчикРёбер* – изменяется от *ЧислоРёбер* до 0, имеет размер $\boldsymbol{O}(log\Delta)$.

Это единственная процедура обработки со слиянием сообщений: сообщение данного типа посылается из вершины по выходящему обратному ребру только тогда, когда получены сообщения этого типа по всем входящим обратным рёбрам. Возможно и дополнительное условие посылки сообщения. Сначала *СчётчикРёбер* = *ЧислоРёбер*, потом счётчик уменьшается на 1 при получении сообщения по входящему обратному ребру. Когда (не в корне) *СчётчикРёбер* становится равным 0, посылается сообщение по выходящему обратному ребру, а *СчётчикРёбер* := *ЧислоРёбер* для дальнейшего использования.

Сбор по обратному дереву обычно начинается в листьях обратных деревьев, по каждому обратному ребру проходит ровно одно сообщение данного типа.

Возможна модификация этого алгоритма, когда ожидаются сообщения (не обязательно этого типа) не по множеству входящих обратных рёбер, а по некоторому его надмножеству, не содержащему выходящего обратного ребра. Например, по всем инцидентным рёбрам, кроме выходящего обратного ребра. В этом случае переменная *ЧислоРёбер* инициализируется не числом входящих обратных рёбер, а числом рёбер в нужном надмножестве, например, *Степень*-1. Сообщение проходит одно ребро и далее путь по дереву. По каждому ребру леса проходит не более одного сообщения, а по другому ребру – не более одного сообщения в каждом направлении.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. $T \le D$ и $T \le D_0$, если корень дерева – это корень графа. Для модифицированной процедуры, соответственно, $T \le D+1$ и $T \le D_0+1$. В любом случае $N \le 1$. $E = \boldsymbol{O}(1)$.

4.7. Рассылка до самопересечения

Граф нумерованный, номер вершины меняется от 0 до n-1 и имеет размер $\boldsymbol{O}(logn)$. Маршрут класса является путём с хордой или путём, заканчивающимся в терминальной вершине графа. Применяется v-накопление.

Базовые параметры: 1) *Тип* – тип сообщения размером $\boldsymbol{O}(1)$, 2) *Маршрут* – v-вектор пути $P = \langle v_1,v_2,...v_k,v_{k+1}\rangle$ размером $\boldsymbol{O}((k+1)logn)$. Базовые переменные: *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$, *Номер* – номер вершины размером $\boldsymbol{O}(logn)$.

Когда сообщение создаётся в вершине v_1, оно рассылается по всем инцидентным ей рёбрам и содержит v-вектор $P = \langle v_1\rangle$. Пусть сообщение с v-вектором $P = \langle v_1,v_2,...v_k,v_{k+1}\rangle$, принимается в вершине v по ребру r. Если это «новая» вершина, т.е. $\forall i$=1..k+1 $v \ne v_i$, сообщение далее посылается по каждому ребру, инцидентному вершине v, кроме ребра r. В сообщении указывается v-вектор $P := P{\cdot}\langle v\rangle$. Если степень вершины больше 2, происходит размножение сообщения. Сообщение уничтожается, когда оно проходит хорду пути, т.е. образуется цикл: $\exists i$=1..k+1 $v = v_i$, или когда вершина v терминальная.

Рассылка до самопересечения начинается в одной вершине. По каждому маршруту с началом в этой вершине, являющемуся путём с хордой или путём, заканчивающимся в терминальной вершине, проходит ровно одно сообщение.

Возможны модификации алгоритма, когда сообщение посылается только по тем рёбрам, которые удовлетворяют тому или иному специальному критерию.

Оценка. $M = \boldsymbol{O}(Dlogn)$. $A = \boldsymbol{O}(log\Delta+logn)$. $F = \boldsymbol{O}(Dlogn+log\Delta) = \boldsymbol{o}_{m\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. $T \le d+1$, а при рассылке от корня $T \le d_0+1$. Число N сообщений на ребре $a\to b$ не больше числа путей, начинающихся в инициаторе и заканчивающихся ребром $a\to b$. На классе всех графов N и E экспоненциально зависят от n.

4.8. Рассылка без повторения

Маршрут этого класса является путём с ребром.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$. Базовые переменные: *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$, *Было* – булевский признак, равный ***true***, если в вершине было сообщение этого типа.

Когда вершина получает сообщение первый раз (*Было* = ***false***), оно пересылается по каждому ребру, инцидентному вершине, кроме того, по которому оно пришло в вершину, и *Было* := ***true***. Повторные (*Было* = ***true***) принимаемые сообщения игнорируются. Если степень вершины больше 2, сообщение размножается. Сообщение уничтожается, если оно повторное (*Было* = ***true***), или когда сообщение попадает в терминальную вершину графа.

Рассылка без повторения начинается в одной вершине, обычно в корне графа, из которой сообщения рассылаются по всем инцидентным рёбрам. В связном графе для каждой вершины b будет ровно одно инцидентное ей ребро ab, по которому сообщение данного типа первый раз приходит в вершину b. Если это ребро условно ориентировать как $a \to b$, то множество так ориентированных рёбер образует прямой остов графа (ориентированный от корня). Если ребро условно ориентируется как $b \to a$, то множество так ориентированных рёбер образует обратный остов графа (ориентированный к корню). По каждому ребру остова пройдёт ровно одно сообщение в направлении $a \to b$. По каждой хорде остова пройдут ровно два сообщения, по одному в каждом направлении.

Возможны модификации алгоритма, когда сообщение посылается только по тем рёбрам, которые удовлетворяют тому или иному специальному критерию.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. Если сообщение рассылается по всем рёбрам, то $T \le d+1$ ($T \le d_0+1$ при рассылке от корня), а если не по всем, то $T \le D+1$ ($T \le D_0+1$ при рассылке от корня). $N \le 1$. $E = \boldsymbol{O}(1)$.

4.9. Множественная рассылка

Граф нумерованный, номер вершины меняется от 0 до n-1 и имеет размер $\boldsymbol{O}(logn)$. Множественная рассылка – это рассылка без повторения, которая параллельно ведётся, начиная с нескольких начальных вершин, которые мы назовём *инициаторами*. Для того чтобы различать сообщения от разных инициаторов, вершина хранит множество номеров инициаторов, сообщения от которых были в вершине. Обычно цель рассылки – доставить информацию в некоторые *конечные* вершины (не инициаторы), в частности, в корень, в которых сообщение не посылается дальше, а уничтожается. Время доставки информации в конечные вершины, если они есть, будем считать временем T, а общее время существования сообщений обозначим T*. Если после доставки информации в конечные вершины нужно, не дожидаясь времени T*, удалить оставшиеся в графе сообщения, можно воспользоваться универсальной процедурой очистки, описываемой ниже в подразделе 6.3.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$, *Инициатор* – номер инициатора размером $\boldsymbol{O}(logn)$. Базовые переменные: *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$, *Номер* – номер вершины размером $\boldsymbol{O}(logn)$, *Инициаторы* – множество номеров инициаторов размером $\boldsymbol{O}(nlogn)$.

Если вершина не конечная, то первое полученное ею сообщение от данного инициатора она пересылает по каждому инцидентному вершине ребру, кроме того, по которому оно получено, а номер инициатора добавляет в переменную *Инициаторы*. Повторные сообщения от того же инициатора, а в терминальной и конечной вершине любые сообщения, уничтожаются. По ребру в одном направлении пройдёт не более одного сообщения от каждого инициатора.

Возможны модификации алгоритма, когда сообщение посылается только по тем рёбрам, которые удовлетворяют тому или иному специальному критерию.

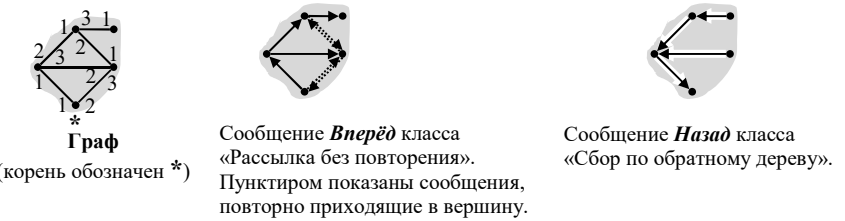
Оценка. $M = \boldsymbol{O}(logn)$. $A = \boldsymbol{O}(nlogn+log\Delta)$. $F = \boldsymbol{O}(nlogn+log\Delta) = \boldsymbol{o}_{m\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. Пусть k число конечных вершин. Если $k > 0$, то $T \le d$, а если сообщение посылается не по всем рёбрам, то $T \le D$. Если только корень конечная вершина, то $T \le d_0$ и $T \le D_0$, соответственно. T* $\le d$+1, а если сообщение посылается не по всем рёбрам, то T* $\le D$+1. $N \le n$-k. $E = \boldsymbol{O}((n\text{-}k)logn)$.

Модификация для векторов: Множественная рассылка для случая, когда номер вершины – это f-вектор пути от корня до вершины по прямому остову, рассматривается ниже в подразделе 5.5. Здесь мы приведём только оценку.

Оценка. $M = \boldsymbol{O}(D_0log\Delta)$. $A = \boldsymbol{O}(nlog\Delta)$, $F = \boldsymbol{O}(nlog\Delta) = \boldsymbol{o}_{m\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. Оценка времени T и числа сообщений N такие же. $E = \boldsymbol{O}((n\text{-}k)D_0log\Delta)$.

5. Построение остова графа

В этом разделе описывается процедура построения остова графа (см. Рис. 3), которая в дальнейшем будет использоваться либо как предварительный этап, либо как прототип алгоритмов решения задач на графе.



Граф
(корень обозначен *)

Сообщение ***Вперёд*** класса «Рассылка без повторения». Пунктиром показаны сообщения, повторно приходящие в вершину.

Сообщение ***Назад*** класса «Сбор по обратному дереву».

Рис. 3. Построение остова графа.
Fig. 3. Construction of the spanning tree.

5.1. Построение обратного остова

Используется сообщение ***Вперёд*** класса «Рассылка без повторения». Вершина имеет дополнительную переменную *ОбратноеРебро* размером $\boldsymbol{O}(log\Delta)$, предназначенную для хранения номера выходящего обратного ребра.

В начале процедуры корень создаёт сообщение ***Вперёд*** и посылает его по всем инцидентным рёбрам. Когда некорневая вершина первый раз (переменная *Было* = ***false***) получает сообщение ***Вперёд***, переменная *ОбратноеРебро* инициализируется номером ребра, по которому получено сообщение ***Вперёд***. Остальное регулируется правилами обработки сообщения класса «Рассылка без повторения». Остов будет построен через время d_0, но ещё не более 1 такта сообщения могут двигаться по графу (по хордам остова). Для определения конца построения остова, применяется процедура из следующего подраздела.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. $T \le d_0+1$. $N \le 1$. $E = \boldsymbol{O}(1)$.

5.2. Определение конца построения обратного остова

Если нужно не только построить обратный остов, но и определить момент завершения построения, то дополнительно используется сообщение ***Назад*** класса «Сбор по обратному дереву», где *ЧислоРёбер* понимается как число инцидентных вершине рёбер, кроме выходящего обратного ребра.

Переменные *ЧислоРёбер* и *СчётчикРёбер* в корне инициализируются степенью корня в начале процедуры построения обратного остова, а в некорневой вершине – степенью вершины минус 1, когда вершина первый раз (переменная *Было* = ***false***) получит сообщение ***Вперёд***. Некорневая вершина ожидает получения по каждому инцидентному ей ребру сообщения ***Вперёд*** или ***Назад***, что требует времени не более d_0+1, после чего посылает по выходящему обратному ребру сообщение ***Назад***. Сообщение ***Назад*** проходит путь к корню, но из-за различного и переменного времени прохождения сообщения по рёбрам графа этот путь может быть не кратчайшим. Поэтому сообщение ***Назад*** может пройти путь длиной не d_0, а D_0.

Процедура заканчивается, когда корень получит сообщение ***Вперёд*** или ***Назад*** по всем инцидентным корню рёбрам. По каждому ребру остова пройдёт сообщение ***Вперёд*** в прямом направлении и сообщение ***Назад*** в обратном направлении, а по каждой хорде остова в каждом направлении пройдёт по одному сообщению ***Вперёд***.

Оценка суммарно с построением обратного остова. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0+1+D_0$. $N \le 1$. $E = \boldsymbol{O}(1)$.

5.3. Установка счётчиков входящих обратных рёбер

Одновременно с построением обратного остова и определением конца построения можно инициализировать в каждой вершине переменную *ЧислоОбратныхРёбер* размером $\boldsymbol{O}(log\Delta)$. Для этого, во-первых, инициализируется переменная *ЧислоОбратныхРёбер* := 0 в корне в начале работы, а в некорневой вершине при получении первого (переменная *Было* = ***false***) сообщения ***Вперёд***. Во-вторых, при получении сообщения ***Назад***, которое приходит по входящему обратному ребру, вершина увеличивает *ЧислоОбратныхРёбер* на 1. При посылке сообщения ***Назад*** по выходящему обратному ребру вершина присваивает *СчётчикРёбер* := *ЧислоРёбер* := *ЧислоОбратныхРёбер*. Корень делает это в конце работы. Переменные *СчётчикРёбер* и *ЧислоРёбер* могут впоследствии использоваться в базовой процедуре «Сбор по обратному дереву», когда *ЧислоРёбер* понимается как число входящих обратных рёбер.

Оценка суммарно с построением обратного остова и определением конца построения. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0+1+D_0$. $N \le 1$. $E = \boldsymbol{O}(1)$.

5.4. Отметка прямых рёбер

Одновременно с построением обратного остова с определением конца построения можно построить и прямой остов, отметив рёбра остова в других их концах. Для этого в каждой вершине нужна переменная *ШкалаРёбер* – битовая шкала размером $\boldsymbol{O}(\Delta)$, содержащая по одному биту на каждое инцидентное ребро. Шкала инициализируется нулями в корне в начале работы, а в некорневой вершине при получении первого (переменная *Было* = ***false***) сообщения ***Вперёд***. При получении по ребру i сообщения ***Назад*** в i-ый бит шкалы записывается 1.

Оценка суммарно с построением остова. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta+\Delta)$. $F = \boldsymbol{O}(log\Delta+\Delta) = \boldsymbol{o}_{m/n\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0+1+D_0$. $N \le 1$. $E = \boldsymbol{O}(1)$.

5.5. Быстрое построение остова

Можно уменьшить оценку времени при построении остова до оптимальной, если использовать сообщение ***Назад*** класса «Множественная рассылка», когда только корень конечная вершина. Для этого класса граф должен быть нумерованным, но мы будем нумеровать вершины одновременно с построением остова. Мы применим способ нумерации вершин, аналогичный тому, который использовался в [6],[8] для ориентированного графа. Для этого в качестве номера вершины возьмём f-вектор прямого пути от корня до этой вершины, который назовём *вектором вершины*. Он имеет размер $\boldsymbol{O}(D_0log\Delta)$. Сообщение ***Вперёд*** посылается с f-накоплением, и вершина получает свой вектор из первого полученного ею сообщения ***Вперёд***. Как и раньше, при получении первого сообщения ***Вперёд*** по ребру i вершина запоминает i как номер выходящего обратного ребра.

Добавляется ещё одно сообщение ***Остов*** класса «Одиночная передача» без дополнительных параметров, которое при передаче из вершины a по ребру $a\to b$ сообщает вершине b, что это ребро является ребром остова. Сообщение посылается в ответ на первое сообщение ***Вперёд***, полученное вершиной a. При получении сообщения ***Остов*** (а не ***Назад*** как раньше) по ребру j вершина увеличивает *ЧислоОбратныхРёбер* на 1 и, если нужно отметить прямые рёбра, то в j-ый бит переменной *ШкалаРёбер* записывает 1.

Сообщение ***Назад*** содержит вектор создавшего его инициатора и число входящих обратных рёбер инициатора как дополнительный параметр. Сообщение создаётся в вершине, когда по каждому инцидентному ей ребру будет получено сообщение ***Вперёд*** или ***Остов***, т.е. когда полностью определится *ЧислоОбратныхРёбер* и, если нужно, *ШкалаРёбер*.

Поскольку корень конечная вершина, он не пересылает дальше сообщение ***Назад***, но запоминает вектор инициатора p и число q_p входящих в него обратных рёбер. Заметим, что входящее обратное ребро совпадает с прямым выходящим ребром с точностью до условной ориентации. Поэтому q_p равно числу выходящих прямых рёбер. Остов построен, когда множество P

запомненных векторов инициаторов содержит вектора всех вершин, что определяется условием «*векторной замкнутости*»: 1) префикс-замкнутость: $\forall p \in P$ любой префикс p принадлежит P, 2) постфикс-замкнутость: вектор вершины продолжается в P каждым прямым ребром, выходящим из неё: $\forall p \in P$ $\boldsymbol{card}(\{ f \mid p \cdot \langle f \rangle \in P \}) = q_p$ (здесь $\boldsymbol{card}(X)$ мощность множества X).

Пусть вершина a получит первое сообщение ***Вперёд*** через время r. Очевидно, $r \le d_0$. Затем, если $r = d_0$, то не более чем через 1 такт вершина a получит по каждому инцидентному ей ребру сообщение ***Вперёд***, и сообщение ***Назад*** будет создано через время не более r+1 = d_0+1. Если $r \le d_0$-1, то не более чем через 2 такта вершина a получит по каждому инцидентному ей ребру сообщение ***Вперёд*** или ***Остов***, и сообщение ***Назад*** будет создано через время не более r+2 ≤ d_0+1. Сообщение ***Назад*** двигается до корня не более d_0 тактов. Поэтому $T \le 2d_0+1$.

Размер вектора вершины равен $\boldsymbol{O}(D_0 log\Delta)$, поэтому переменная *Инициаторы* имеет размер $\boldsymbol{O}(nD_0 log\Delta) = \boldsymbol{o}_{m\to\infty}$. Однако на классе графов без кратных рёбер и петель, где $\Delta \le n$-1, этот размер может достигать по порядку размера описания графа $n^2 logn$. Например, если в графе есть путь от корня длиной n-1 по рёбрам с максимальными номерами n-1, то может получиться так, что k-ая вершина пути получит свой вектор размером порядка (k-1)log(n-1) как вектор префикса пути длиной k-1, а сумма размеров векторов по всем инициаторам будет порядка $n^2 logn$.

Размер памяти автомата можно уменьшить, если в переменной *Инициаторы* хранить не список векторов, а прямое дерево [13], в котором пометить инициаторы. Это дерево будем называть *деревом инициаторов*. Оно задаётся как список описаний вершин в порядке обхода дерева в ширину. Описание вершины – это список выходящих из вершины прямых номеров рёбер дерева. Прямой номер ребра, имеющий двоичное представление $x_1,x_2,...x_r$ хранится в виде последовательности $0,x_1,0,x_2,...\ 0,x_r,1$ размером $\boldsymbol{O}(log\Delta)$. Конец описания вершины задаётся кодом 100, конец списка описаний вершин задаётся кодом 110 (в [13] это коды 10 и 11, соответственно). Если вершина – это запомненный инициатор, то код 100 заменяется кодом 101. Тогда переменная *Инициаторы* имеет размер $\boldsymbol{O}(n+nlog\Delta) = \boldsymbol{O}(nlog\Delta)$. В корне после кода 101 дополнительно помещается число входящих обратных рёбер инициатора размером $\boldsymbol{O}(log\Delta)$; суммарно по всем инициаторам добавляется $\boldsymbol{O}(nlog\Delta)$ бит.

Оценка. $M = \boldsymbol{O}(D_0 log\Delta)$. Если прямые рёбра не отмечаются, то $A = \boldsymbol{O}(nlog\Delta)$, $F = \boldsymbol{O}(nlog\Delta) = \boldsymbol{o}_{m\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$], а если отмечаются (5.4), то $A = \boldsymbol{O}(nlog\Delta+\Delta)$, $F = \boldsymbol{O}(nlog\Delta+\Delta) = \boldsymbol{o}_{m/n\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. $T \le 2d_0+1$. По ребру в одном направлении одновременно перемещаются не более одного сообщения ***Вперёд*** или ***Остов*** и не более одного сообщения ***Назад*** от каждого инициатора (числом n-1), поэтому $N \le n$. $E = \boldsymbol{O}(nD_0 log\Delta)$.

6. Стопор и очистка

В этом разделе мы предложим универсальную процедуру обнаружения завершения работы любого алгоритма, использующего сообщения конечного числа типов, которую мы назвали *стопором*, а также процедуру *очистки* графа от сообщений указанных типов.

6.1. Стопор

Любой алгоритм $\mathcal{A}$, использующий сообщения конечного числа типов, завершает свою работу не позже того момента, когда в графе исчезнут все сообщения используемых им типов. Предполагается, что работа алгоритма $\mathcal{A}$ инициируется корнем, т.е. сообщения указанных типов генерирует только корень, а любая некорневая вершина посылает сообщение одного из указанных типов только в таком срабатывании, в котором она получает сообщение одного из указанных типов. Если нужно отслеживать наличие или отсутствие в графе сообщений любого из типов $t_1,t_2,...t_k$, то множество этих типов $t = \{t_1,t_2,...t_k\}$ понимается как обобщённый тип, и считается, что сообщение имеет тип t, если оно имеет тип $t_i \in t$.

Стопор используется корнем в цикле: корень начинает процедуру стопора и, когда она заканчивается и оказывается, что в графе ещё есть сообщение указанного типа, начинается новая процедура стопора, и так далее до тех пор, пока не окажется, что сообщений указанного типа нет.

Стопор строится как модификация процедуры (не быстрого) построения обратного остова с определением конца построения (раздел 5). В сообщение ***Вперёд*** добавлен один параметр: тип учитываемого сообщения *Утип* размером $\boldsymbol{O}(1)$. Задача сообщений ***Вперёд*** – известить все вершины о начале процедуры стопора. Сообщение ***Назад*** содержит дополнительный булевский параметр *Найдено*, который сообщает о том, обнаружено или нет сообщение типа *Утип*. Задача сообщений ***Назад*** – доставить в корень дизъюнкцию их параметров *Найдено*.

Вершина находится в одном из двух режимов работы: *Рабочий* и *Учёт*. Режим работы хранится в переменной *Режим* размера $\boldsymbol{O}(1)$. Также в вершине имеется дополнительная булевская переменная *Найдено*. Корень переходит из режима *Рабочий* в режим *Учёт*, когда он начинает процедуру стопора. Можно считать, что это происходит сразу после начала работы алгоритма $\mathcal{A}$, когда корень генерирует первое сообщение типа *Утип*, или при завершении очередной процедуры стопора, если в графе ещё остались сообщения типа *Утип* и должна начаться следующая процедура стопора. В этот момент корень инициализирует переменную *Найдено* := ***false***, создаёт сообщение ***Вперёд*** и рассылает его по всем инцидентным корню рёбрам.

Некорневая вершина в режиме *Рабочий*, получив сообщение ***Вперёд***, переходит в режим *Учёт* и присваивает своей переменной *Найдено* := ***false***. Когда вершина в режиме *Учёт* посылает сообщение типа *Утип*, она

присваивает своей переменной *Найдено* := ***true***. Когда вершина получает сообщение ***Назад*** с параметром *Найдено* = ***true***, она присваивает своей переменной *Найдено* := ***true***. Когда вершина посылает сообщение ***Назад***, параметр *Найдено* делается равным значению переменной *Найдено*, и вершина переходит в режим *Рабочий*. Когда процедура завершается, корень переходит в режим *Рабочий*. Если при этом в корне *Найдено* = ***true***, то корень начинает новую процедуру стопора, снова переходя в режим *Учёт*.

Утверждение 1: Если в начале процедуры стопора в графе нет сообщений типа *Ytun*, и во время процедуры корень не генерирует таких сообщений, то при окончании процедуры стопора в корне *Найдено* = ***false***.

Доказательство: Действительно, при этих условиях во время процедуры стопора в графе не могут возникнуть сообщения типа *Ytun*, поскольку некорневые вершины не генерируют такие сообщения, а только пересылают их. Поэтому в каждой вершине переменная *Найдено* = ***false*** в течение всей процедуры стопора. А тогда в конце процедуры стопора в корне *Найдено* = ***false***, что и требовалось доказать.

Утверждение 2: Если при окончании процедуры стопора в корне *Найдено* = ***false***, то в графе нет сообщений типа *Ytun*.

Доказательство: Допустим, это не так, и при окончании процедуры стопора в графе на некотором ребре $a \to b$ есть сообщение M_1 типа *Ytun*. Поскольку при окончании процедуры стопора в корне *Найдено* = ***false***, в каждой вершине также *Найдено* = ***false***. Сообщения типа *Ytun* не генерируются во время процедуры стопора. Следовательно, сообщение M_1 послано из вершины a до её перехода в режим *Учёт* и всё ещё находится на ребре, когда вершина b выходит из режима *Учёт*. Но вершина b выходит из режима *Учёт* только после получения по ребру $a \to b$ сообщения M_2 типа ***Вперёд*** или ***Назад***, а такое сообщение посылается только в режиме *Учёт*. Следовательно, M_2 посылается из a позже M_1, а принимается в b раньше, чего быть не может. Мы пришли к противоречию, и утверждение доказано.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0+D_0+1$. $N \le 1$. $E = \boldsymbol{O}(1)$.

Для определения конца работы алгоритма $\mathcal{A}$, использующего сообщения типа *Ytun*, процедура стопора запускается в цикле до тех пор, пока она не завершится с переменной в корне *Найдено* = ***false***. Поэтому в тот момент времени, когда в графе реально исчезают сообщения типа *Ytun*, уже может быть запущена процедура стопора, которая может дать *Найдено* = ***true***. Но зато следующая процедура стопора даст гарантированно *Найдено* = ***false***. Поэтому время может удвоиться: $T \le 2(d_0+D_0+1)$.

6.2. Быстрый стопор

Процедуру стопора можно реализовать как модификацию процедуры быстрого построения остова (подраздел 5.5). Вершина находится в режиме

Учёт не более 2 тактов между получением первого сообщения ***Вперёд*** и посылкой сообщения ***Назад***: за эти 2 такта по каждому инцидентному вершине ребру придёт сообщение ***Вперёд*** или ***Остов***. В доказательстве утверждения 2 сообщение M_2 – это сообщение типа ***Вперёд*** или ***Остов***.

После выполнения процедуры быстрого стопора в графе могут остаться сообщения ***Назад***, поскольку они рассылаются множественной рассылкой. Для того чтобы эти сообщения не перепутывались с сообщениями ***Назад*** следующей процедуры быстрого стопора, будем разделять эти процедуры на чётные и нечётные. Чётность процедуры является дополнительным параметром сообщений ***Вперёд*** и ***Назад***. Когда вершина первый раз получает сообщение ***Вперёд***, она запоминает чётность процедуры и далее игнорирует все принимаемые сообщения ***Назад*** предыдущей чётности. Когда вершина посылает сообщение ***Назад***, она помещает в него текущую чётность, а сама готова к приёму сообщения ***Вперёд*** следующей чётности, считая его первым сообщением ***Вперёд*** следующей процедуры. Важно отметить, что процедура быстрого стопора не заканчивается, пока по каждому ребру в каждом направлении не пройдёт сообщение ***Вперёд*** или ***Остов***. Эти сообщения «очищают» рёбра от сообщений ***Назад*** предыдущей чётности. Поэтому, когда закончится процедура с данной чётностью, в графе не останется сообщений ***Назад*** предыдущей чётности.

Оценка. $M = \boldsymbol{O}(D_0 log\Delta)$. $A = \boldsymbol{O}(nlog\Delta)$. $F = \boldsymbol{O}(nlog\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2d_0+1$. $N \le n$. $E = \boldsymbol{O}(nD_0 log\Delta)$.

6.3. Очистка и быстрая очистка

Процедуры стопора и быстрого стопора легко модифицировать в процедуры очистки и быстрой очистки, которые вместо того, чтобы определять, есть ли в графе сообщения указанного типа, удаляют все такие сообщения. Вершина после перехода в режим *Учёт* не посылает сообщений типа *Ytun*. Правда, если применяется процедура быстрой очистки, то она использует сообщения класса «Множественная рассылка», и эти её сообщения могут остаться в графе при завершении очистки, хотя сообщений типа *Ytun* уже не будет.

7. Дерево кратчайших путей

Дерево кратчайших путей – это такой остов графа, что расстояние от корня до любой вершины в остове и в графе совпадают. Высота этого дерева равна d_0, что позволяет впоследствии использовать его для рассылки от корня до каждой вершины (если отмечены прямые рёбра, т.е. дерево прямое) и обратно (если отмечены обратные рёбра, т.е. дерево обратное) за время не более $2d_0$.

7.1. Построение обратного дерева кратчайших путей

Для того чтобы построить обратное дерево кратчайших путей, модифицируем алгоритм построения остова, описанный в подразделе 5.1. Сообщение ***Вперёд***

будет накапливать длину пройденного им маршрута в дополнительном параметре *Длина* размером $\boldsymbol{O}(logD_0)$. Некорневая вершина сохраняет в своей дополнительной переменной *Расстояние* размером $\boldsymbol{O}(logD_0)$ минимум из параметров *Длина* сообщений ***Вперёд***, полученных этой вершиной, а в переменной *ОбратноеРебро* номер ребра, по которому получено сообщение ***Вперёд*** с минимальным значением параметра *Длина*.

Корень посылает сообщение ***Вперёд*** с параметром *Длина* = 1. Некорневая вершина, получив сообщение ***Вперёд*** первый раз, инициализирует *Расстояние* := *Длина*, *ОбратноеРебро* := *f*, где *f* – номер ребра, по которому получено сообщение. Сообщение ***Вперёд*** посылается дальше по всем инцидентным вершине рёбрам, кроме ребра *f*, с параметром *Длина*, увеличенным на 1. Повторное сообщение ***Вперёд*** игнорируется только в том случае, когда *Длина* ≥ *Расстояние*. Если же *Длина* < *Расстояние*, то выполняются присваивания *Расстояние* := *Длина* и *ОбратноеРебро* := *f*, где *f* – номер ребра, по которому получено сообщение. Сообщение ***Вперёд*** посылается дальше по всем инцидентным вершине рёбрам, кроме ребра *f*, с параметром *Длина*, увеличенным на 1.

Для определения конца построения используется универсальный стопор – процедура стопора из раздела 6 для типа сообщения ***Вперёд***.

Покажем, что через d_0 тактов в графе не останется сообщений ***Вперёд***, и будет построено дерево кратчайших путей. Действительно, пусть расстояние от корня до вершины равно *r*. Тогда не более чем через *r* тактов вершина получит сообщение, прошедшее от корня до этой вершины кратчайший путь. Тем самым, после *r* тактов вершина перестанет посылать сообщение ***Вперёд***. Поэтому сообщения ***Вперёд*** исчезнут из графа не позже чем через d_0 тактов. Очевидно, что если в каждой некорневой вершине выбрать в качестве обратного ребра любое инцидентное вершине ребро, лежащее на кратчайшем пути от корня до вершины, то получится дерево кратчайших путей.

Оценка без учёта стопора. По сравнению с алгоритмом 5.1 размер сообщения и памяти автомата увеличиваются на $\boldsymbol{O}(logD_0)$. $M = \boldsymbol{O}(logD_0)$. $A = \boldsymbol{O}(logD_0+log\Delta)$. $F = \boldsymbol{O}(logD_0+log\Delta) = \boldsymbol{o}_{m,n\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0$. Вершина может получить несколько подряд сообщений ***Вперёд*** со строго уменьшающимися значениями параметра *Длина* и переслать их дальше. Для пересылаемых дальше сообщений *Длина* ≤ D_0. Поэтому $N \le D_0$, $E = \boldsymbol{O}(D_0logD_0)$.

7.2. Установка счётчиков входящих обратных рёбер

В подразделе 5.3 счётчики входящих обратных рёбер устанавливались одновременно с построением обратного остова и определением конца построения. Здесь мы применим аналогичный алгоритм, но только после определения конца построения обратного дерева кратчайших путей. Вместо сообщения ***Вперёд*** класса «Рассылка без повторения» используется сообщение ***Вперёд*****1** класса «Рассылка против отмеченных рёбер», где

отмеченное ребро – это ребро обратного дерева кратчайших путей. Сообщение ***Вперёд*****1** отличается от сообщения ***Вперёд*** не только своим классом, т.е. способом рассылки, но также тем, что не происходит инициализации переменной *ОбратноеРебро*, поскольку она уже установлена при построении дерева кратчайших путей. Соответственно, сообщение ***Назад*** создаётся и посылается по выходящему обратному ребру тогда, когда вершина получит по каждому инцидентному ей ребру сообщение ***Вперёд*****1** (вместо ***Вперёд***) или сообщение ***Назад***.

Оценка без учёта построения дерева кратчайших путей. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2d_0+1$. $N \le 1$. $E = \boldsymbol{O}(1)$.

Можно совместить установку счётчиков с процедурой стопора для определения конца построения дерева кратчайших путей. Переименуем сообщения ***Вперёд*** в стопоре на ***Вперёд*****2**. Первое сообщение ***Вперёд*****2**, приходящее в вершину, инициирует сброс счётчика в ноль и посылку сообщения ***Остов*** по текущему выходящему обратному ребру. Получая сообщение ***Остов***, вершина увеличивает счётчик на 1. Если очередная процедура стопора не обнаруживает сообщений ***Вперёд***, то счётчики будут установлены правильно. Это даёт суммарную оценку верхней границы времени построения дерева кратчайших путей с установкой счётчиков не $d_0+2(d_0+D_0+1)+2d_0+1 = 5d_0+2D_0+3$, а $d_0+2(d_0+D_0+1) = 3d_0+2D_0+1$.

7.3. Отметка прямых рёбер

После определения конца построения обратного дерева кратчайших путей можно построить и прямой остов, т.е. отметить рёбра остова в других их концах. Это можно совместить с установкой счётчиков входящих обратных рёбер. Алгоритм аналогичен описанному в подразделе 5.4. Переменная *ШкалаРёбер* – инициализируется нулями при получении сообщения ***Вперёд*****1** по выходящему обратному ребру (в корне с самого начала). При получении по ребру *i* сообщения ***Назад*** в *i*-ый бит шкалы записывается 1.

Оценка без учёта построения дерева кратчайших путей. $M = \boldsymbol{O}(1)$, $A = \boldsymbol{O}(log\Delta+\Delta)$, $F = \boldsymbol{O}(log\Delta+\Delta) = \boldsymbol{o}_{m/n\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2d_0+1$. $N \le 1$. $E = \boldsymbol{O}(1)$.

Можно совместить отметку прямых рёбер с процедурой стопора для определения конца построения дерева кратчайших путей аналогично тому, как это делается в подразделе 7.2. Получая сообщение ***Остов*** по некоторому ребру, вершина отмечает это ребро в шкале рёбер.

8. Нумерация графа

В этом разделе мы предложим несколько алгоритмов нумерации графа. Вершина имеет дополнительную переменную *Номер*, которая инициализируется в процедуре нумерации графа. Вершины можно пронумеровать произвольными уникальными идентификаторами, например, векторами вершин. Однако отметим, что минимальная память, требуемая для

хранения идентификаторам вершины, получается тогда, когда вершины пронумерованы числами от 0 до n-1.

8.1. Нумерация векторами

Если на нумерацию графа налагается единственное ограничение: разные вершины имеют разные номера, – то в качестве номера вершины можно использовать вектора вершин. Процедура нумерация векторами является модификацией процедуры построения обратного остова с определением конца построения (подраздел 5.2). Сообщение ***Вперёд*** выполняется с f-накоплением и поэтому имеет размер не $\boldsymbol{O}(1)$ как в подразделе 5.2, а $\boldsymbol{O}(D_0 log\Delta)$. Когда такое сообщение первый раз (переменная *Было* = ***false***) попадает в вершину, f-вектор пути запоминается в переменной *Номер* вершины.

Оценка. $M = \boldsymbol{O}(D_0 log\Delta)$, $A = \boldsymbol{O}(D_0 log\Delta)$. $F = \boldsymbol{O}(D_0 log\Delta) = \boldsymbol{o}_{m\to\infty}$ $[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0+D_0+1$. $N \le 1$. $E = \boldsymbol{O}(D_0 log\Delta)$.

Модификация 1. Если предварительно построен обратный остов с установкой счётчиков входящих обратных рёбер и отметкой прямых рёбер, то можно предложить другой алгоритм нумерации векторами. Используются сообщения ***Вперёд*** класса «Рассылка по отмеченным рёбрам» с f-накоплением размером $\boldsymbol{O}(D_0 log\Delta)$, где отмеченными рёбрами считаются прямые рёбра остова, и сообщение уничтожается в листе остова, и сообщение ***Назад*** класса «Сбор по обратному дереву», где *ЧислоРёбер* понимается как число входящих обратных рёбер. Вершина, получая сообщение ***Вперёд***, запоминает f-вектор пути как свой вектор вершины. Листовая вершина остова, кроме этого, создаёт сообщение ***Назад*** без дополнительных параметров. Процедура заканчивается, когда в корне становится *СчётчикРёбер* = 0, после чего для дальнейшего использования *СчётчикРёбер* := *ЧислоРёбер*. Сообщение ***Вперёд*** проходит путь от корня до листа остова длиной не более D_0, после чего сообщение ***Назад*** проходит постфикс этого пути в обратном направлении.

Оценка без учёта ресурсов для предварительного построения остова. $M = \boldsymbol{O}(D_0 log\Delta)$. $A = \boldsymbol{O}(D_0 log\Delta)$. $F = \boldsymbol{O}(D_0 log\Delta) = \boldsymbol{o}_{m\to\infty}$ $[\boldsymbol{o}_{n\to\infty}]$. $T \le 2D_0$, а если остов – это дерево кратчайших путей, то $T \le 2d_0$. $N \le 1$. $E = \boldsymbol{O}(D_0 log\Delta)$.

Модификация 2: Алгоритм модификации 1 можно применить и тогда, когда предварительно построен обратный остов с установкой счётчиков входящих обратных рёбер, но без отметки прямых рёбер. Для этого достаточно использовать сообщение ***Вперёд*** с f-накоплением не класса «Рассылка по отмеченным рёбрам», а класса «Рассылка против отмеченных рёбер», где под отмеченными рёбрами понимаются рёбра обратного остова.

Поскольку установлены счётчики входящих обратных рёбер, вершина «знает», что является листом, если в ней счётчик равен нулю, и в этом случае не посылает сообщение ***Вперёд*** дальше. Поэтому сообщение ***Вперёд*** проходит путь от корня до листа остова длиной не более D_0. После этого сообщение ***Назад*** проходит постфикс этого пути в обратном направлении.

Оценка без учёта ресурсов для предварительного построения остова. $M = \boldsymbol{O}(D_0 log\Delta)$. $A = \boldsymbol{O}(D_0 log\Delta)$. $F = \boldsymbol{O}(D_0 log\Delta) = \boldsymbol{o}_{m\to\infty}$ $[\boldsymbol{o}_{n\to\infty}]$. $T \le 2D_0$, а если остов – это дерево кратчайших путей, то $T \le 2d_0$. $N \le 1$. $E = \boldsymbol{O}(D_0 log\Delta)$.

8.2. Быстрая нумерация векторами

Процедуру нумерации векторами можно построить как модификацию процедуры быстрого построения остова (подраздел 5.5).

Оценка. $M = \boldsymbol{O}(D_0 log\Delta)$. $A = \boldsymbol{O}(n log\Delta)$. $F = \boldsymbol{O}(n log\Delta) = \boldsymbol{o}_{m\to\infty}$ $[\boldsymbol{o}_{n\to\infty}]$. $T \le 2d_0+1$. $N \le n$. $E = \boldsymbol{O}(nD_0 log\Delta)$.

8.3. Нумерация по алгоритму Тэрри

Для того чтобы пронумеровать граф числами от 0 до n-1, можно воспользоваться хорошо известным алгоритмом Тэрри обхода неориентированного графа [14]. Сообщение обходит граф, проходя по каждому ребру ровно 2 раза. Более строго, здесь используются сообщения трёх типов: ***Вперёд*** класса «Рассылка без повторения» при пересылке по ещё не пройденным рёбрам, ***ОткатПоХорде*** класса «Одиночная передача» при возвращении по хорде остова, и ***Назад*** класса «Рассылка по отмеченным рёбрам», где отмеченное ребро – это выходящее обратное ребро, и сообщение уничтожается сразу после прохода этого ребра. См. Рис. 4.

Дополнительные переменные вершины: *МойНомер* размером $\boldsymbol{O}(logn)$, *Ребро* размером $\boldsymbol{O}(log\Delta)$, в которой хранится номер ребра, по которому последний раз из вершины посылалось сообщение ***Вперёд***, и *ШкалаРёбер* размером $\boldsymbol{O}(\Delta)$, в которой отмечаются пройденные рёбра: как при посылке, так и при получении по ребру сообщения ***Вперёд***.

Переменные *Ребро* и *ШкалаРёбер* инициализируются нулями: в корне – в начале работы, а в некорневой вершине – при получении первого (*Было* = ***false***) сообщения ***Вперёд***. Затем некорневая вершина дополнительно присваивает: *ОбратноеРебро* := f, где f – ребро, по которому получено сообщение ***Вперёд***, *ШкалаРёбер*(f) := 1. Если все рёбра, инцидентные некорневой вершине, пройдены, что проверяется по переменной *ШкалаРёбер*, то посылается сообщение ***Назад*** по выходящему обратному ребру. Если все рёбра пройдены в корне, обход закончен. Если не все рёбра, инцидентные вершине, пройдены, то посылается сообщение ***Вперёд*** (с использованием специального критерия) по одному ребру – следующему после *Ребро* непройденному ребру f. *Ребро* := f и *ШкалаРёбер*(f) := 1. Когда вершина получает сообщение ***Вперёд*** повторно (*Было* = ***true***), то это сообщение приходит по хорде. Вершина отмечает эту хорду в переменной *ШкалаРёбер* и в обратном направлении по хорде посылает сообщение ***ОткатПоХорде***. Когда вершина получает сообщение ***Назад*** или ***ОткатПоХорде***, она проверяет непройденность рёбер и, если нужно, посылает сообщение ***Назад*** или ***Вперёд*** как описано выше.

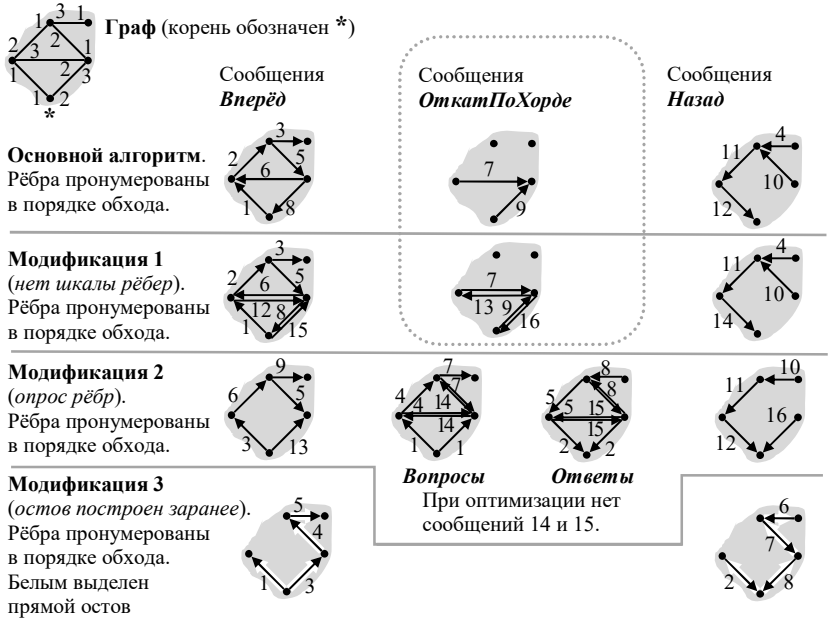


Рис. 4. Нумерация графа по алгоритму Тэрри.
Fig. 4. Numbering of the graph by the Terry algorithm.

Для нумерации вершин достаточно, чтобы сообщение каждого из этих трёх типов содержало параметр *Номер* размера $\boldsymbol{O}(\log n)$, который должен быть присвоен очередной вершине. Нумерация начинается с того, что корень присваивает себе *МойНомер* := 0, и посылает сообщение ***Вперёд*** с параметром *Номер* = 1. Когда некорневая вершина получает сообщение ***Вперёд*** первый раз (*Было* = ***false***), она получает свой номер из сообщения ***Вперёд***: *МойНомер* := *Номер*, а *Номер* + 1 становится параметром посылаемого далее сообщения ***Вперёд*** или ***Назад***. В остальных случаях параметр *Номер* из принятого сообщения без изменения переписывается в посылаемое сообщение.

Оценка. $M = \boldsymbol{O}(\log n)$. $A = \boldsymbol{O}(\log n+\Delta)$. $F = \boldsymbol{O}(\log n+\Delta) = \boldsymbol{o}_{m/n\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2m$. $N \le 1$, поскольку алгоритм последовательный. $E = \boldsymbol{O}(\log n)$.

Модификация 1. Можно сократить размер автомата за счёт увеличения времени работы. Достаточно просто отказаться от шкалы рёбер: номер ребра, по которому посылается сообщение ***Вперёд***, – это следующий номер после *Ребро*, отличный от *ОбратноеРебро*. По хорде остова сообщения будут проходить теперь не 2, а 4 раза: из каждого конца хорды туда и обратно.

Оценка. $M = \boldsymbol{O}(\log n)$, $A = \boldsymbol{O}(\log n\Delta)$, $F = \boldsymbol{O}(\log n\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. Время увеличится на удвоенное число хорд остова: $T \le 4m-2n+2$. $N \le 1$. $E = \boldsymbol{O}(\log n)$.

Модификация 2. Можно, наоборот, уменьшить время работы алгоритма, сохраняя шкалу рёбер и размер автомата $\boldsymbol{O}(\log n+\Delta)$. Для этого корень вначале, а другая вершина при получении сообщения ***Вперёд*** выполняет опрос инцидентных рёбер, чтобы установить, какие из них являются хордами остова. Не считая опросов, выполняться будет обход остова, а не всего графа. В каждую некорневую вершину придёт ровно одно сообщение ***Вперёд***.

Переменная *ШкалаРёбер* хранит «0» для прямых выходящих рёбер и (до опроса) неопрошенных рёбер. С самого начала в корне *ШкалаРёбер* нулевая.

Опрос распараллеливается, используя сообщения ***Вопрос*** и ***Ответ*** класса «Вопрос/Ответ». ***Вопрос*** посылается по всем рёбрам, для которых *ШкалаРёбер* содержит «0». ***Ответ*** содержит булевский параметр *Хорда*, показывающий, является ли ребро хордой остова или нет. Пусть вершина получает ***Вопрос*** по ребру *x*. Если *Было* = ***false***, то вершина посылает ***Ответ*** с параметром *Хорда* = ***false*** и устанавливает *Было* := ***true***, *ОбратноеРебро* := *x*, *ШкалаРёбер* инициализируется нулями, а потом *ШкалаРёбер*(*x*) := 1. В противном случае вершина посылает ***Ответ*** с параметром *Хорда* = ***true***. Вершина, получающая по ребру *y* сообщение ***Ответ*** с параметром *Хорда* = ***true***, устанавливает *ШкалаРёбер*(*y*) := 1. Сообщение ***Вперёд*** посылается после получения сообщений ***Ответ*** на все посланные сообщения ***Вопрос***. Сообщение ***Вперёд*** теперь относится к классу «Рассылка по отмеченным рёбрам»: оно посылается только по выходящим прямым рёбрам, но, как и прежде, только по одному ребру – следующему после *Ребро*.

Параметр *Номер* имеется в каждом сообщении. При получении сообщения ***Вперёд*** *МойНомер* := *Номер*, а параметр *Номер* в посылаемом сообщении увеличивается на 1. В остальных случаях параметр *Номер* из принятого сообщения без изменения переписывается в посылаемое сообщение.

Оценка. $M = \boldsymbol{O}(\log n)$. $A = \boldsymbol{O}(\log n+\Delta)$. $F = \boldsymbol{O}(\log n+\Delta) = \boldsymbol{o}_{m/n\to\infty}[\boldsymbol{o}_{n\to\infty}]$. Обход остова занимает время не более $2(n-1)$, а в каждой из n вершин опрос рёбер занимает не более 2 тактов. $T \le 4n-2$. $N \le 1$. $E = \boldsymbol{O}(\log n)$.

Оптимизация. Когда вершина первый раз получает сообщение ***Вопрос*** по ребру *x*, она отмечает это ребро как хорду: *ШкалаРёбер*(*x*) := 1. Эта вершина не будет опрашивать это ребро. В наихудшем случае время T не уменьшится, а в общем случае опрос рёбер делают не все, а $k = 1..n$ вершин, и $T = 2n+2k-2$.

Модификация 3. Если предварительно построен обратный остов с установкой счётчиков входящих обратных рёбер и отметкой прямых рёбер, то нумерацию естественно делать обходом не графа, а остова. Используются сообщение ***Вперёд*** класса «Рассылка по отмеченным рёбрам», которое, как и в модификации 2, посылается по одному выходящему прямому ребру, и сообщение ***Назад*** класса «Рассылка по отмеченным рёбрам» при возвращении по выходящему обратному ребру. Оба сообщения имеют дополнительный параметр *Номер*, а вершина имеет дополнительную переменную *Ребро*.

Оценка без учёта предварительного построения остова. $M = \boldsymbol{O}(logn)$. $A = \boldsymbol{O}(logn+log\Delta)$. $F = \boldsymbol{O}(logn\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2(n\text{-}1)$. $N \le 1$. $E = \boldsymbol{O}(logn)$.

8.4. Нумерация через корень

Можно сократить время нумерации числами от 0 до n-1 за счёт распараллеливания так же, как в случае нумерации векторами. Для этого применяется модификация процедуры построения обратного остова с определением конца построения (см. Рис. 5).

Добавляются два новых сообщения: ***ДайНомер*** класса «Рассылка по отмеченным рёбрам» с r-накоплением размером $\boldsymbol{O}(D_0log\Delta)$, где отмеченными рёбрами считаются выходящие обратные рёбра, и сообщение уничтожается, когда доходит до корня, и ***БериНомер*** класса «Пересылка по запомненному маршруту» с длиной маршрута не более D_0 и дополнительным параметром *Номер* размером $\boldsymbol{O}(logn)$. В вершине имеются дополнительные переменные *МойНомер* размером $\boldsymbol{O}(logn)$ и булевский *ПризнакНумерации*, а в корне – переменная *ТекущийНомер* размером $\boldsymbol{O}(logn)$.

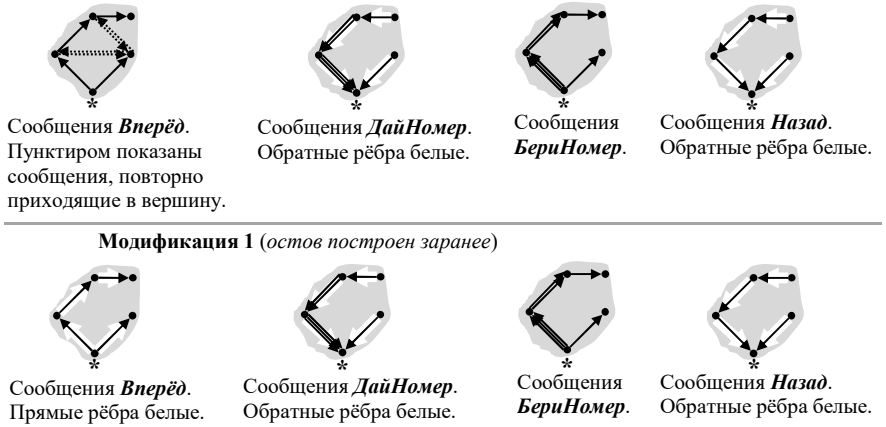


Рис. 5. Нумерация через корень.
Fig. 5. Numbering through the root.

Работа начинается с того, что корень присваивает себе номер 0, т.е. *МойНомер* := 0, инициализирует переменные *ПризнакНумерации* := ***true*** и *ТекущийНомер* := 0, создаёт сообщение ***Вперёд*** и посылает его по всем инцидентным корню рёбрам.

Когда вершина v первый раз (переменная *Было* = ***false***) получает сообщение ***Вперёд***, инициализируется переменная *ПризнакНумерации* := ***false*** и создаётся сообщение ***ДайНомер***. Это сообщение двигается по обратному пути до корня с накоплением r-вектора пути. Корень получает из принятого сообщения ***ДайНомер*** r-вектор $<r_1,...r_k>$ пройденного сообщением пути. Очевидно,

обратная последовательность $<r_k,...r_1>$ – это f-вектор пути от корня до вершины v. Получая сообщение ***ДайНомер***, корень увеличивает *ТекущийНомер* на 1 и создаёт сообщение ***БериНомер*** класса «пересылка по запомненному маршруту», в котором запомненный маршрут – это f-вектор $<r_k,...r_1>$ пути от корня до вершины v, и есть дополнительный параметр *Номер* размером $\boldsymbol{O}(logn)$, который устанавливается равным значению переменной *ТекущийНомер*. Когда сообщение ***БериНомер*** пройдёт запомненный в нём путь, конечная вершина этого пути, т.е. вершина v, запоминает значение параметра *Номер* в переменной *МойНомер*.

Также изменяется условие создания сообщения ***Назад***: дополнительно требуется, чтобы вершина была уже пронумерована. Это означает, что сообщение ***Назад*** создаётся либо 1) при получении сообщения ***Вперёд*** или ***Назад***, когда *СчётчикРёбер* становится равен нулю, а вершина уже пронумерована, т.е. *ПризнакНумерации* = ***true***, либо 2) в момент нумерации вершины (*ПризнакНумерации* := ***true***) при получении сообщения ***БериНомер***, предназначенного этой вершине (в сообщении f-вектор нулевой длины), если *СчётчикРёбер* = 0. В обоих случаях после создания сообщения ***Назад*** *СчётчикРёбер* := *ЧислоРёбер* для дальнейшего использования. Когда в корне переменная *СчётчикРёбер* станет равной нулю, нумерация закончена.

Сообщение ***ДайНомер*** создаётся, когда сообщение ***Вперёд*** проходит путь от корня до вершины v за время не более d_0, и само проходит этот путь в обратном направлении от вершины v до корня за время не более D_0, после чего сообщение ***БериНомер*** проходит этот путь в прямом направлении от корня до вершины v за время не более D_0, а после этого сообщение ***Назад*** двигается по тому же пути от вершины v в сторону корня за время не более D_0. Тем самым, время $T \le d_0+3D_0$ при условии $D_0 > 0$. Если $D_0 = 0$, то в связном графе только одна вершина, а все рёбра – петли. В этом случае требуется время для прохождения петель: $T = 1$.

Сообщения ***ДайНомер*** от всех вершин, кроме корня, т.е. от n-1 вершины, могут оказаться на одном ребре (а потом сообщения ***БериНомер*** для всех вершин, кроме корня, могут оказаться на одном ребре).

Оценка. $M = \boldsymbol{O}(D_0log\Delta+logn)$. $A = \boldsymbol{O}(log\Delta+logn)$. $F = \boldsymbol{O}(D_0log\Delta+logn) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le max\{d_0+3D_0,1\}$. $N \le max\{n\text{-}1,1\}$. $E = \boldsymbol{O}(n(D_0log\Delta+logn))$.

Модификация 1. Эта модификация применяется тогда, когда уже построен обратный остов с установкой счётчиков входящих обратных рёбер и отметкой прямых рёбер. Сообщение ***Вперёд*** теперь будет относиться к классу «Рассылка по отмеченным рёбрам», где отмеченные рёбра – это прямые выходящие рёбра. Сообщение ***Назад*** теперь будет относиться к классу «Сбор по обратному дереву», где *ЧислоРёбер* понимается как число входящих обратных рёбер.

В начале корень создаёт и рассылает по выходящим из корня прямым рёбрам сообщение ***Вперёд***. Сообщения ***ДайНомер*** и ***БериНомер*** такие же, как в описанном выше алгоритме. Некорневая вершина получит сообщение ***Вперёд***

один раз и создаст сообщение ***ДайНомер***. Дополнительное условие посылки сообщения ***Назад*** такое же: вершина должна быть уже пронумерованной.

Процедура заканчивается, когда в корне становится *СчётчикРёбер* = 0, после чего *СчётчикРёбер* := *ЧислоРёбер* для дальнейшего использования. Если $n = 1$, с самого начала в корне *СчётчикРёбер* = 0, поэтому сообщения не посылаются и процедура заканчивается после присвоения корню номера 0.

Оценка без учёта предварительного построения остова. $M = \boldsymbol{O}(D_0 log\Delta + logn)$. $A = \boldsymbol{O}(log\Delta + logn)$. $F = \boldsymbol{O}(D_0 log\Delta + logn) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 4D_0$, а если остов – это дерево кратчайших путей, то $T \le 4d_0$. $N \le n$-1. $E = \boldsymbol{O}(n(D_0 log\Delta + logn))$.

8.5. Быстрая нумерация вершин через корень

Можно уменьшить время нумерации через корень, меняя в оценке D_0 на d_0, если применять модификацию быстрого построения остова (подраздел 5.5). Сообщения ***ДайНомер*** и ***БериНомер*** относятся к классу «Множественная рассылка», как и сообщение ***Назад***. Сообщение ***ДайНомер*** создаётся, как и раньше, когда вершина первый раз получает сообщение ***Вперёд***, и имеет только базовые параметры своего класса. Сообщение ***БериНомер*** создаётся, как и раньше, когда корень получит сообщение ***ДайНомер***, и имеет дополнительный параметр *Номер*. Имеется дополнительное условие создания сообщения ***Назад***: требуется, чтобы вершина была уже пронумерована.

Оценка. $M = \boldsymbol{O}(D_0 log\Delta + logn)$. $A = \boldsymbol{O}(nlog\Delta + logn)$. $F = \boldsymbol{O}(nlog\Delta + logn) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 4d_0$+1. $N \le n$. $E = \boldsymbol{O}(n(D_0 log\Delta + logn))$.

8.6. Быстрая нумерация вершин не через корень

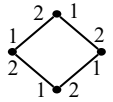
Можно ещё уменьшить время быстрой нумерации, если заметить следующий факт. В алгоритме быстрого построения остова (подраздел 5.5) сообщение ***Назад***, содержащее вектор инициатора и число входящих в него обратных рёбер, рассылается множественной рассылкой. Поэтому не только корень, но каждая вершина получает в переменной *Инициаторы* дерево инициаторов и может определить конец построения такого дерева. После этого вершина может сама вычислить свой номер, применяя один и тот же алгоритм обхода дерева в ширину. Сообщения ***ДайНомер*** и ***БериНомер*** не нужны.

Мы возьмём за основу алгоритм быстрого построения остова, но только поменяем название сообщения ***Назад*** на ***Конец***. Когда вершина определит конец построения дерева инициаторов по условию «векторной замкнутости», она вычисляет свой номер и посылает сообщение ***Назад*** класса «Множественная рассылка», сообщая как дополнительный параметр свой номер. Когда корень получит сообщения ***Назад*** от всех вершин, алгоритм заканчивается. Каждый инициатор посылает сообщение ***Конец*** через время не более d_0+1, до каждой вершины оно идёт не более d тактов. Затем сообщение ***Назад*** от каждой вершины доходит до корня за время не более d_0.

Оценка. $M = \boldsymbol{O}(D_0 log\Delta + logn)$. $A = \boldsymbol{O}(nlog\Delta + logn)$. $F = \boldsymbol{O}(nlog\Delta + logn) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0 + d + d_0 + 1 \le 4d_0 + 1$. $N \le n$. $E = \boldsymbol{O}(n(D_0 log\Delta + logn))$.

9. Сбор информации о графе в памяти автомата корня

В этом разделе мы предложим алгоритм сбора полной информации о графе в памяти неограниченного автомата корня. При этом автоматы остальных вершин будут полуроботами. Прежде всего, отметим, что если граф ненумерованный, то информация о графе может содержать только *fr*-вектора маршрутов. Однако множество таких *fr*-векторов определяет граф не однозначно. Например, неразличимы графы, каждый из которых состоит из простого цикла рёбер длины $k > 0$ с нумерацией рёбер по циклу (1,2),(1,2),...(1,2) (см. Рис. 6).



Множество *fr*-векторов задаётся регулярным выражением (12)* (здесь знак «*» означает конечное число повторений выражения в скобках 0 или более раз) и не зависит от числа вершин в цикле.

Рис. 6. Множество fr-векторов не однозначно определяет граф.
Fig. 6. The set of fr-vectors does not uniquely determine the graph.

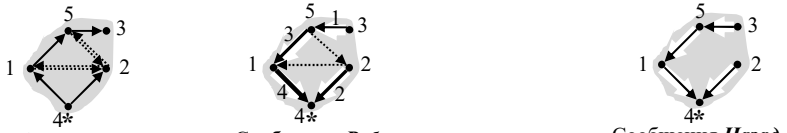
Будем предполагать, что в нумерованном графе номера вершин линейно упорядочены. Алгоритмы нумерации из раздела 8 это гарантируют как для чисел от 0 до n-1, так и для векторов. Алгоритм сбора – это модификация алгоритмов построения обратного остова с определением конца построения или быстрого построения остова. Можно также совместить сбор информации с нумерацией графа по любому из алгоритмов раздела 8.

Информация о графе собирается в виде множества описаний рёбер формата (v,f,r,v`), где $v$ и $v$` – номера вершин, инцидентных ребру, f – номер ребра в вершине v, r – номер ребра в вершине v`. Если вершины пронумерованы числами от 0 до n-1, то размер описания ребра $\boldsymbol{O}(logn\Delta)$, а если используются вектора вершин, то $\boldsymbol{O}(D_0 log\Delta)$. Описание ребра посылается как параметр сообщения ***Ребро***, которое должно дойти до корня. Вершина должна создать все нужные сообщения ***Ребро*** до того, как она создаст сообщение ***Назад***, и все эти сообщения должны идти по одним и тем же путям от вершины до корня. Тогда корень получит все сообщения ***Ребро*** до того, как получит все сообщения ***Назад***, по которым он определяет конец работы.

Заметим, что объединять в одном сообщении описания всех рёбер, инцидентных вершине, – плохая идея, так как это увеличивает порядок размера сообщения и, соответственно, полного размера автомата до $\Delta logn\Delta$ или (для векторов) до $\Delta D_0 log\Delta$. На классе всех графов с n вершинами и m рёбрами эти величины имеют порядок $mlognm$ или (для векторов) до $mnlogm$, т.е. автомат является неограниченным.

9.1. Сбор по остову нумерованного графа

Сбор информации о нумерованном графе можно реализовать как модификацию алгоритма построения обратного остова с определением конца построения (подраздел 5.2, см. также Рис. 7). Сообщение ***Ребро*** имеет класс «Рассылка по отмеченным рёбрам», где отмеченное ребро – это выходящее обратное ребро; сообщение двигается по обратному пути до корня.



Сообщения ***Вперёд***. Пунктиром показаны сообщения, повторно приходящие в вершину.

Сообщения ***Ребро***. Обратные рёбра – белые. Хорды – пунктирные линии. Описание ребра $a \to b$ посылает вершина a. Показано число сообщений, проходящих по ребру.

Сообщения ***Назад***.

Рис. 7. Сбор информации о графе в неограниченной памяти автомата корня.
Fig. 7. Collect information about the graph in unlimited memory of the root automaton.

Описание ребра добавляется также в сообщение ***Вперёд***. Когда это сообщение посылается из вершины v по ребру f, описание ребра в нём имеет формат $(v,f,0,0)$. Когда первое сообщение ***Вперёд*** приходит в некорневую вершину $v`$ по ребру $r$, создаётся сообщение ***Ребро*** с описанием ребра $(v,f,r,v`)$. Если сообщение ***Вперёд*** повторное, то сообщение ***Ребро*** создаётся в том случае, когда $v > v`$ в линейном порядке вершин. Сообщение ***Назад*** посылается из вершины после того, как она пошлёт все свои сообщения ***Ребро***. При получении корнем сообщения ***Вперёд*** можно считать, что корень создаёт сообщение ***Ребро*** и сам же его принимает без реальной пересылки по рёбрам.

В корень поступит одно описание каждого ребра. Так как все сообщения ***Ребро*** посылаются из вершины раньше, чем сообщение ***Назад***, и эти сообщения двигаются по одному и тому же обратному пути, в момент определения конца построения остова все сообщения ***Ребро*** уже достигли корня, и корень получил полную информацию о графе.

В «худшем» случае на одном ребре могут оказаться все сообщения ***Ребро*** и одно сообщение ***Назад***, суммарно m+1.

Оценка. Размер сообщения и памяти некорневой вершины увеличивается на размер описания ребра. $T \le d_0+D_0+1$. $N \le m+1$. Если номер вершины – это число от 0 до n-1, то $M = \boldsymbol{O}(log n\Delta)$, $A = \boldsymbol{O}(log n\Delta)$ и $F = \boldsymbol{O}(log n\Delta) = \boldsymbol{o}_{m\to\infty}$ $[\boldsymbol{o}_{n\to\infty}]$, $E = \boldsymbol{O}((m+1)log n\Delta)$. Если номер вершины – это её вектор, то $M = \boldsymbol{O}(D_0 log\Delta)$, $A = \boldsymbol{O}(D_0 log\Delta)$ и $F = \boldsymbol{O}(D_0 log\Delta) = \boldsymbol{o}_{m\to\infty}$ $[\boldsymbol{o}_{n\to\infty}]$, $E = \boldsymbol{O}((m+1)D_0 log\Delta)$.

9.2. Быстрый сбор по нумерованному графу

Сбор информации о нумерованном графе можно реализовать как модификацию алгоритма быстрого построения обратного остова (подраздел

5.5). Аналогично алгоритму из предыдущего подраздела описание ребра добавляется в сообщение ***Вперёд***. Когда это сообщение посылается из вершины v по ребру f, описание ребра в нём имеет формат $(v,f,0,0)$. Когда первое сообщение ***Вперёд*** приходит в некорневую вершину $v`$ по ребру $r$, создаётся сообщение ***Ребро*** с описанием ребра $(v,f,r,v`)$. Если сообщение ***Вперёд*** повторное, то сообщение ***Ребро*** создаётся в том случае, когда $v > v`$ в линейном порядке вершин. Сообщение ***Назад*** посылается из вершины после того, как она пошлёт все свои сообщения ***Ребро***. При получении корнем сообщения ***Вперёд*** можно считать, что корень создаёт сообщение ***Ребро*** и сам же его принимает без реальной пересылки по рёбрам.

Теперь сообщение ***Ребро***, как и сообщение ***Назад***, имеет класс «Множественная рассылка», когда только корень конечная вершина. На одном ребре могут оказаться одновременно сообщения ***Ребро*** для каждого ребра и сообщения ***Назад*** от каждой вершины, кроме корня, а также одно сообщение ***Вперёд*** или ***Остов***; всего не более $m+(n-1)+1 = m+n$ сообщений.

Но здесь есть проблема с инициаторами сообщений ***Ребро***. Как сказано выше, это сообщение содержит описание только одного ребра, поэтому вершина посылает множество таких сообщений ***Ребро*** с описаниями инцидентных вершине рёбер, и все они должны дойти до корня. Поэтому инициатором сообщения ***Ребро*** с описанием ребра $(v,f,r,v`)$ следует считать не вершину $v`$, создавшую это сообщение, а ребро, задаваемое парой $(r,v`)$. Эта проблема решается немного по-разному в зависимости от того, пронумерованы вершины числами от 0 до n-1 или векторами.

Если номер вершины – это вектор, то переменная *Инициаторы* (подраздел 5.5) хранит дерево инициаторов-вершин как список описаний вершин в порядке обхода дерева в ширину, где описание вершины – это список выходящих из вершины прямых номеров рёбер дерева. Для добавления инициаторов-рёбер, достаточно в описание вершины добавить (через разделитель) битовую шкалу рёбер, инцидентных вершине. Инициатор-ребро, задаваемый парой $(r,v`)$, отмечается «1» в $r$-м бите шкалы рёбер в описании вершины $v`$. Размер переменной *Инициаторы* увеличивается на $\boldsymbol{O}(n+m)$ бит.

Оценка (номер вершины – её вектор). $M = \boldsymbol{O}(D_0 log\Delta)$. Для некорневой вершины $A = \boldsymbol{O}(n+m+nlog\Delta)$ и $F = \boldsymbol{O}(n+m+nlog\Delta) = \boldsymbol{o}_{m/n\to\infty}$ $[\boldsymbol{o}_{n\to\infty}]$. $T \le 2d_0+1$. $N \le m+n$. $E = \boldsymbol{O}((m+n)D_0 log\Delta)$.

Если вершины пронумерованы числами от 0 до n-1, то нет смысла использовать алгоритм, основанный на векторах вершин. В переменной *Инициаторы* хранится список описаний вершин в порядке возрастания их номеров. Описание вершины содержит один бит для отметки вершины-инициатора и битовую шкалу рёбер, инцидентных вершине, для отметки рёбер-инициаторов. Размер переменной равен $\boldsymbol{O}(n+m)$ бит.

Сообщение ***Вперёд*** посылается без f-накопления. Также как описано выше, в сообщение ***Вперёд*** добавляется описание ребра $(v,f,0,0)$, которое используется для создания сообщения ***Ребро*** с описанием ребра $(v,f,r,v`)$. Сообщение ***Назад***

содержит номер (а не вектор) создавшего его инициатора и число входящих обратных рёбер инициатора как дополнительный параметр.

Кроме того, меняется процедура определения конца работы, выполняемая в корне. В подразделе 5.5 эта процедура использовала дерево инициаторов с указанием числа входящих обратных рёбер для каждого инициатора. Теперь для описания дерева инициаторов достаточно в описаниях рёбер отметить рёбра остова. Для этого в описание ребра добавляется булевский признак остова, который равен ***true***, если описание ребра создаётся при получении вершиной первого сообщения ***Вперёд***, и равен ***false***, если описание ребра создаётся при получении вершиной повторного сообщения ***Вперёд***. Число входящих обратных рёбер, как и в подразделе 5.5, равно числу полученных вершиной сообщений ***Остов*** и передаётся корню в сообщении ***Назад***.

Вместо условия «векторной замкнутости» у нас будет аналогичное условие «числовой замкнутости»: 1) для каждого ребра остова (v,f,r,$v`$) оба его конца $v$ и $v`$ должны быть вершинами-инициаторами, 2) для каждой вершины-инициатора $v`$ (а также для корня) число рёбер остова вида ($v$,$f$,$r$,$v`$) должно быть равно числу обратных рёбер, входящих в вершину $v`$.

Оценка (номер вершины – число от 0 до n-1). $M = \boldsymbol{O}(log n\Delta)$. Для некорневой вершины $A = \boldsymbol{O}(n+m+log n\Delta) = \boldsymbol{O}(m)$ и $F = \boldsymbol{O}(m) = \boldsymbol{o}_{m/n\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. $T \le 2d_0+1$. $N \le m+n$. $E = \boldsymbol{O}((m+n)log n\Delta)$.

9.3. Сбор по остову с одновременной нумерацией графа

Алгоритм сбора может быть построен как модификация любого не быстрого алгоритма нумерации графа (подразделы 8.1, 8.3, 8.4). Размер сообщения и памяти некорневой вершины увеличивается на размер описания ребра.

Идея алгоритма следующая. Получая номер, вершина опрашивает соседей с помощью сообщений класса «Вопрос/Ответ». ***Вопрос*** посылается из вершины v по ребру f и принимается вершиной $v`$ по ребру $r$, а ***Ответ*** посылается обратно по тому же ребру. Оба сообщения содержат описание ребра: ***Вопрос*** – в формате ($v$,$f$,0,0), ***Ответ*** – в формате ($v$,$f$,0,0), если вершина $v`$ ещё не имеет номера, или в формате (v,f,r,$v`$), если уже имеет. Сообщение ***Ребро*** с описанием ребра ($v$,$f$,$r$,$v`$) создаёт либо вершина $v`$ при получении сообщения ***Вопрос***, если вершина v` пронумерована и $v > v`$ в линейном порядке вершин, либо вершина $v$ при получении сообщения ***Ответ*** с параметром ($v$,$f$,$r$,$v`$), если $v < v`$. Сообщение ***Ребро*** имеет класс «Рассылка по отмеченным рёбрам», где отмеченное ребро – выходящее обратное ребро. Сообщение двигается по обратному пути до корня. Сообщение ***Назад*** создаётся после окончания опроса соседей и посылки всех нужных сообщений ***Ребро***.

Опрос соседей занимает не более 2-х тактов. Опрос начинается, когда вершина получает номер, происходит параллельно в разных вершинах и не тормозит передачу других сообщений, кроме, быть может, сообщения ***Назад***. Поэтому время T увеличивается не более чем на 2 такта. Для некоторых

алгоритмов нумерации эти «лишние» два такта можно удалить, если опрос соседей выполнять с помощью уже имеющихся в алгоритме сообщений.

В основном алгоритме нумерации векторами ***Вопрос*** и ***Ответ*** не нужны. Сообщение ***Ребро*** создаётся при получении либо первичного сообщения ***Вперёд***, либо повторного сообщения ***Вперёд*** от вершины с большим вектором. В модификации 1 предварительно построен обратный остов с отметкой прямых рёбер. Для ребра остова создаётся одно сообщение ***Ребро*** при получении первичного сообщения ***Вперёд***. Вершина опрашивает только хорды (соседей на концах хорд). В модификации 2 предварительно построен обратный остов без отметки прямых рёбер. Для рёбер остова создаётся сообщение ***Ребро*** при получении либо сообщения ***Вперёд*** по выходящему обратному ребру, либо по хорде от вершины с большим вектором.

В алгоритме Тэрри нужно в сообщении ***Вперёд***, посылаемом из вершины v по ребру f, дополнительно указать описание ребра в формате (v,f,0,0). В основном алгоритме сообщение ***Ребро*** создаётся при получении сообщения ***Вперёд***. В модификации 1 нет шкалы рёбер, поэтому сообщение ***Ребро*** создаётся при получении вершиной $v`$ сообщения ***Вперёд*** либо по ребру остова, либо по хорде при условии $v` < v$. В модификации 2 (без оптимизации) уже выполняется опрос соседей. Поэтому достаточно в сообщении ***Вопрос***, посылаемом из вершины v по ребру f и принимаемом вершиной $v`$ по ребру $r$, указать описание ребра в формате ($v$,$f$,0,0), а в сообщении ***Ответ*** с параметром *Хорда* = ***true*** указать описание ребра в формате ($v$,$f$,0,0), если вершина $v`$ ещё не имеет номера, или в формате (v,f,r,$v`$), если вершина $v`$ уже пронумерована. Сообщение ***Ребро*** создаётся при получении вершиной $v`$ по ребру $r$ либо сообщения ***Вперёд***, либо сообщения ***Вопрос***, если вершина $v`$ уже пронумерована и $v > v`$, либо при получении вершиной $v$ сообщения ***Ответ*** с параметрами *Хорда* = ***true***, $r \ne 0$ и $v` > v$. В модификации 3 предварительно построен остов с отметкой прямых рёбер и выполняется обход остова, а не графа. Для рёбер остова создаётся сообщение ***Ребро*** при получении сообщения ***Вперёд***. Хорды (соседи на концах хорд) опрашиваются.

В основном алгоритме нумерации через корень и его модификации 1 нужно делать опрос соседей при получении сообщения ***БериНомер***.

9.4. Быстрый сбор с одновременной нумерацией графа

Алгоритм сбора может быть построен как модификация любого быстрого алгоритма нумерации графа (подразделы 8.2, 8.5, 8.6). Размер сообщения и памяти некорневой вершины увеличивается на размер описания ребра. Как и в предыдущем подразделе используется опрос соседей, который увеличивает время T не более чем на 2 такта. Иногда от этих 2-х тактов можно избавиться.

При быстрой нумерации векторами ***Вопрос*** и ***Ответ*** не нужны. Сообщение ***Ребро*** создаётся при получении либо первичного сообщения ***Вперёд***, либо повторного сообщения ***Вперёд*** от вершины с большим вектором.

В алгоритме быстрой нумерации через корень нужно делать опрос соседей при получении сообщения ***БериНомер***.

В алгоритме быстрой нумерации не через корень опрашиваются только хорды (соседи на их концах). Каждая вершина сама определяет конец построения дерева инициаторов по условию его «векторной замкнутости» и вычисляет свой номер. Также она может вычислить номер n_v любой вершины v по её вектору p_v. Для прямого ребра $a \to b$ вершина b знает его номер r в вершине b (выходящее из b обратное ребро) и свой вектор $p_b = p_a \cdot \langle f \rangle$, где f – номер ребра в вершине a. Вершина b строит описание ребра $a \to b$ в формате (n_a,f,r,n_b) и создаёт сообщение ***Ребро***. Также вершина b по своему вектору p_b находит в дереве инициаторов своё описание, содержащее номера выходящих прямых рёбер. Тем самым, вершина b определяет номера хорд, которые опрашивает.

10. Поиск максимального пути

В этом разделе рассмотрим решение коллективом автоматов задачи поиска максимального пути в графе (*Longest Path Problem*). Как известно, эта задача относится к классу *NP*. Но в нашей модели, допускающей неограниченное число сообщений, она решается за линейное время. Граф предполагается нумерованным числами от 0 до n-1. Сначала рассмотрим задачу поиска и, если нужно, разметки максимального пути, начинающегося в корне графа, а потом общую задачу поиска и разметки максимального пути в графе.

10.1. Максимальный путь от корня

Идея алгоритма заключается в следующем. Рассмотрим все маршруты, начинающиеся в корне и имеющие вид $B \cdot \langle e \rangle$ или C, где B – путь до вершины v, e – хорда пути B, инцидентная вершине v, C – путь, заканчивающийся в терминальной вершине. Максимальный путь от корня – это максимальный среди путей B и C. Используются два типа сообщения: ***Прямо*** класса «Рассылка до самопересечения» с *vfr*-накоплением размером $\boldsymbol{O}(D_0 log n \Delta)$ и ***Обратно*** класса «Рассылка по отмеченным рёбрам», которыми считаются рёбра обратного остова (Рис. 8). Вершина содержит дополнительную булевскую переменную *Было*, которая равна ***false***, если эта вершина ещё не получала сообщения ***Прямо***; вначале *Было* = ***false***.

В начале работы корень присваивает *Было* := ***true***, создаёт сообщение ***Прямо*** и рассылает его по всем инцидентным рёбрам. Получая первое сообщение ***Прямо***, вершина отмечает ребро, по которому принято сообщение, как выходящее обратное ребро. Так строится обратный остов графа. Сообщение ***Обратно*** создаётся при уничтожении сообщения ***Прямо***, когда его маршрут самопересекается или заканчивается в терминальной вершине. В этот момент сообщение ***Прямо*** содержит *vfr*-вектор пути с ребром вида $P_k \cdot \langle f_k \rangle$, где $P_k = \langle v_1,f_1,r_1,v_2,f_2,r_2,...v_k \rangle$, v_1 корень, f_k прямой номер ребра, по которому вершина v_k посылала сообщение. Сообщение получает вершина v_{k+1} по ребру

r_k, причём либо $v_{k+1} = v_i$ для некоторого i=1..k, либо вершина v_{k+1} терминальная. Сообщение ***Обратно*** имеет дополнительный параметр $P = P_k$, если было самопересечение, или $P = P_k \cdot \langle f_k,r_k,v_{k+1} \rangle$, если маршрут закончился в терминальной вершине.

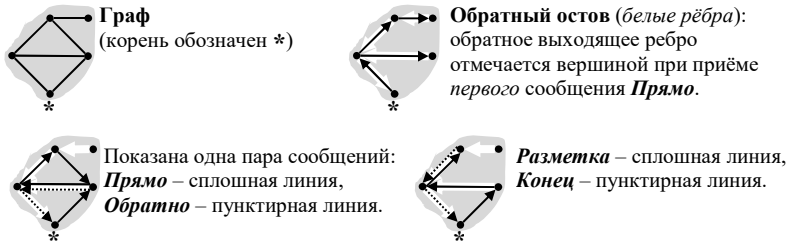


Рис. 8. Поиск максимального пути от корня и его разметка.
Fig. 8. Search for the maximum path from the root and its markup.

В корне хранится *vfr*-вектор P_{max}, вначале пустой. Когда корень получает сообщение ***Обратно*** с параметром P, он сравнивает длину P из сообщения с длиной P_{max}. Если P длиннее, то запоминается $P_{max} := P$.

Конец работы определяется с помощью универсальной процедуры стопора (раздел 6) для типов сообщений ***Прямо*** и ***Обратно***. Корень посылает вовне сообщение, содержащее как параметр $P_{max} = \langle v_1,f_1,r_1,v_2,f_2,r_2,...v_{k_{max}} \rangle$.

Сообщение ***Прямо*** проходит от корня путь с хордой, длиной не более D_0+1, или путь, заканчивающийся в терминальной вершине, длиной не более D_0. Затем сообщение ***Обратно*** проходит путь до корня длиной не более D_0.

Число сообщений на ребре экспоненциально, что и следовало ожидать, поскольку задача поиска максимального пути – класса *NP*.

Оценка без учёта стопора. $M = \boldsymbol{O}(D_0 log n \Delta)$. $A = \boldsymbol{O}(D_0 log n \Delta)$. $F = \boldsymbol{O}(D_0 log n \Delta) = \boldsymbol{o}_{m \to \infty}[\boldsymbol{o}_{n \to \infty}]$. $T \le 2D_0$+1. N и E экспоненциальны.

Модификация 1. Число сообщений ***Обратно*** можно сделать линейным (а не экспоненциальным), если в каждой вершине хранить максимальную длину k_{max} векторов P из проходивших через вершину сообщений ***Обратно***. Полученное вершиной сообщение ***Обратно***, содержашее вектор P длины k, посылается дальше только, если $k > k_{max}$. Поскольку k меняется в диапазоне от 0 до D, число сообщений ***Обратно*** на одном ребре не превосходит D+1.

Модификация 2. Если нужно разметить найденный максимальный путь от корня, то после завершения поиска такого пути корень создаёт сообщение ***Разметка*** класса «Пересылка по запомненному маршруту», в котором параметр *Маршрут* равен $\langle f_1,f_2,...f_{k_{max}-1} \rangle$ и имеет длину не более D_0. Когда вершина v_j, где j=1..k_{max}-1, посылает сообщение ***Разметка*** по ребру f_j, она отмечает выходящее ребро пути в переменной $Fn := f_j$ размером $\boldsymbol{O}(log \Delta)$. Когда вершина v_j, где j=2..k_{max}, принимает сообщение ***Разметка*** по ребру r_{j-1},

она отмечает входящее ребро пути в переменной $Rn := r_{j-1}$ размером $\boldsymbol{O}(log\Delta)$. Когда сообщение ***Разметка*** дойдёт до конца запомненного пути, т.е. до вершины $v_{k_{max}}$, эта вершина создаёт сообщение ***Конец*** класса «Рассылка по отмеченным рёбрам», которыми считаются рёбра обратного остова. Разметка заканчивается, когда корень получает сообщение ***Конец***.

Сообщение ***Разметка*** проходит путь от корня длиной не более D_0, а сообщение ***Конец*** возвращается в корень по пути длиной не более D_0.

Оценка без учёта поиска максимального пути. $M = \boldsymbol{O}(D_0 log n\Delta)$. $A = \boldsymbol{O}(D_0 log n\Delta)$. $F = \boldsymbol{O}(D_0 log n\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2D_0$. $N \le 1$ и $E = \boldsymbol{O}(D_0 log n\Delta)$.

Модификация 3. Если в графе построен обратный остов, при получении первого сообщения ***Прямо*** не нужно отмечать выходящее обратное ребро. Выигрыш по времени будет, если остов – дерево кратчайших путей.

Оценка. $M = \boldsymbol{O}(D_0 log n\Delta)$. $A = \boldsymbol{O}(D_0 log n\Delta)$. $F = \boldsymbol{O}(D_0 log n\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. При поиске пути без учёта стопора $T \le D_0+d_0+1$. Для разметки $T \le D_0+d_0$.

Заметим, что такой же выигрыш по времени можно получить и в том случае, когда дерево кратчайших путей не построено, если сообщения ***Обратно*** и ***Конец*** имеют класс «Множественная рассылка». Однако в этом случае память автомата больше: $A = \boldsymbol{O}(n log n\Delta)$, $F = \boldsymbol{O}(n log n\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$.

10.2. Максимальный путь в графе

Идея поиска максимального пути в связном графе заключается в следующем. Рассмотрим все маршруты вида $A{\cdot}B{\cdot}\langle e\rangle$ или $A{\cdot}C$, где A – путь от корня до некоторой вершины v, B – путь от вершины v до вершины $v`$, $e$ – хорда пути $B$, инцидентная вершине $v`$, C – путь от вершины $v`$, заканчивающийся в терминальной вершине. Вершину v назовём *инициатором*. Максимальный путь в графе – это максимальный среди путей B и C для всех инициаторов.

Как и при поиске максимального пути от корня, используются два типа сообщения: ***Прямо*** класса «Рассылка до самопересечения» с *vfr*-накоплением и ***Обратно*** класса «Рассылка по отмеченным рёбрам», которыми считаются рёбра обратного остова (Рис. 9). Вершина содержит переменную *Было*.

Сообщение ***Прямо*** может создаваться не только в корне, но и в любой вершине-инициаторе v, поэтому *vfr*-вектор имеет вид $P_k{\cdot}\langle f_k\rangle$, где $P_k = \langle v_{j+1},f_{j+1},r_{j+1},v_{j+2},f_{j+2},r_{j+2},...v_k\rangle$ и $v_{j+1} = v$. Вершина v становится инициатором, когда получает первое (*Было* = ***false***) сообщение ***Прямо*** с *vfr*-вектором $P_j{\cdot}\langle f_j\rangle$, где $P_j = \langle v_1,f_1,r_1,v_2,f_2,r_2,...v_j\rangle$. Вершина v сначала рассылает дальше сообщение ***Прямо*** в соответствии с его классом, а потом создаёт новое сообщение ***Прямо***, которое рассылается по всем инцидентным рёбрам. Новое сообщение ***Прямо***, посылаемое по ребру f_{j+1}, содержит *vfr*-вектор $\langle v_{j+1}\rangle{\cdot}\langle f_{j+1}\rangle$.

Создание и обработка сообщения ***Обратно*** и определение конца работы такие же как в случае поиска максимального пути от корня.

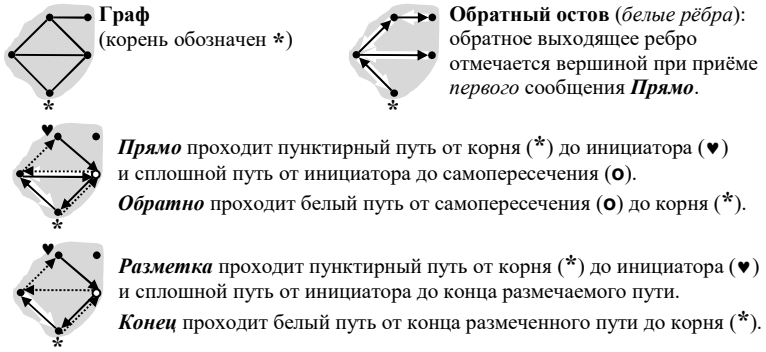


Рис. 9. Поиск максимального пути в графе и его разметка.
Fig. 9. Search for the maximum path in the graph and its markup.

Модификация 1 аналогична модификации 1 из подраздела 10.1. Число сообщений ***Обратно*** на ребре не больше $D+1$.

Модификация 2 для разметки найденного максимального пути. В сообщение ***Прямо*** добавляется ещё один параметр R, содержащий *f*-вектор пути от корня до инициатора. Когда инициатор – корень, R пусто. Сообщение ***Прямо*** с инициатором в некорневой вершине v создаётся этой вершиной при получении первого сообщения ***Прямо*** (его инициатор – корень), содержащего *vfr*-вектор $P_j{\cdot}\langle f_j\rangle$, где $P_j = \langle v_1,f_1,r_1,v_2,f_2,r_2,...v_j\rangle$. Тогда $R = \langle f_1,...f_j\rangle$. Сообщение ***Обратно*** также имеет дополнительный параметр R, переписываемый из сообщения ***Прямо*** при создании сообщения ***Обратно***.

Корень вместе с переменной $P_{max} = \langle v_{j+1},f_{j+1},r_{j+1},v_{j+2},f_{j+2},r_{j+2},...v_{k_{max}}\rangle$ хранит соответствующую ей переменную $R_{max} = \langle f_1,f_2,...f_j\rangle$, в которую из сообщения ***Обратно*** переписывается параметр R, когда параметр P переписывается в переменную P_{max}. Когда создаётся сообщение ***Разметка***, оно имеет два параметра: R_{max} и $P^f_{max} = \langle f_{j+1},f_{j+2},,...f_{k_{max}-1}\rangle$ как подпоследовательность P_{max}, являющаяся *f*-вектором найденного пути. Сначала сообщение ***Разметка*** двигается по пути с *f*-вектором R_{max} от корня до начала найденного пути, а потом – по этому пути с *f*-вектором P^f_{max}, размечая путь в переменных Rn и Fn.

Модификация 3 полностью аналогична модификации 3 в подразделе 10.1.

Оценка. По сравнению с оценками в подразделе 10.1 увеличиваются размеры сообщения M и автомата A за счёт того, что величина D_0 меняется на D: $M = A = F = \boldsymbol{O}(Dlog n\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. Если используется множественная рассылка, то память ещё больше: $A = F = \boldsymbol{O}(n log n\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$.

Время работы (без учёта стопора) слагается из времени X прохождения пути от корня до инициатора, времени Y прохождения максимального пути и времени Z прохождения пути до корня. При поиске пути $X \le d_0$, $Y \le D$ и $Z \le D_0$, а при разметке $X \le D_0$, $Y \le D$ и $Z \le D_0$. Если используется ранее построенное дерево кратчайших путей или множественная рассылка, то $Z \le d_0$.

При наличии дерева кратчайших путей можно при разметке сделать $X \le d_0$. Для этого надо, чтобы вершина становилась инициатором при получении не первого сообщения ***Прямо***, а того, которое прошло кратчайший путь по дереву от корня. Для этого сообщение ***Прямо*** имеет булевский признак *Кратчайший*, который равен ***true*** при создании сообщения в корне, а затем «сбрасывается» в ***false***, когда сообщение проходит не остовное ребро (что отмечено в его конце) или когда оно создаётся в некорневом инициаторе.

Если используется множественная рассылка для передачи сообщения ***Разметка*** от корня до инициатора, то при разметке также $X \le d_0$. В этом случае в сообщениях не нужен параметр R, описывающий путь от корня до инициатора. Инициатор сам «ловит» адресованное ему сообщение ***Разметка***, поскольку в нём есть *vfr*-вектор пути, начинающийся как раз в инициаторе.

В целом получаем: при поиске пути $T \le d_0+2D+1$, при разметке $T \le d_0+2D$. При использовании дерева кратчайших путей или множественной рассылки: при поиске пути $T \le 2d_0+D+1$, при разметке $T \le 2d_0+D$. При поиске пути N и E экспоненциальны, а при разметке $N \le 1$ и $E = \boldsymbol{O}(D_0 log n \Delta)$.

11. Заключение

В статье предложен общий подход к решению задач на неориентированном упорядоченном связном корневом графе коллективом автоматов, расположенных в вершинах графа и обменивающихся сообщениями по рёбрам графа. Автоматы – полуроботы, т.е. размер их памяти может расти вместе с ростом числа вершин n и числа рёбер m графа, но описание графа, вообще говоря, не помещается в память автомата. Выбрана модель максимального распараллеливания: время срабатывания автомата нулевое, а ёмкость ребра (число сообщений на нём) не ограничена. Это позволяет получать нижние оценки сложности алгоритмов решения задач.

Предложены базовые процедуры обработки сообщений, из которых строятся алгоритмы решения задач на графах. Это продемонстрировано на примерах построения остова графа, универсального «стопора», определяющего конец работы любого алгоритма, построения дерева кратчайших путей и нумерации графа. Также предложен алгоритм сбора в неограниченной памяти автомата корня полной информации о графе полуроботами некорневых вершин. Все эти алгоритмы имеют линейную сложность от n и m, и не более чем линейное число сообщений, одновременно передаваемых по ребру. Также рассмотрена задача поиска максимального пути в графе класса *NP*. За счёт неограниченного (экспоненциального) числа сообщений алгоритм решения этой задачи имеет линейную сложность.

Полученные результаты позволяют надеяться, что предложенная методика может успешно применяться для решения других задач на графах. Мы намереваемся в дальнейшем рассмотреть такие задачи, как построение минимального остовного дерева во взвешенном графе, построение

максимального независимого множества вершин, определение всех мостов в неориентированном графе и др. Также предполагается расширить предлагаемый подход на случай ориентированных графов, недетерминированных и динамических графов.

Список литературы

[1]. И. Бурдонов, А. Косачев. Обход неизвестного графа коллективом автоматов. Труды ИСП РАН, том 26, вып. 2, 2014, стр. 43-86. DOI: 10.15514/ISPRAS-2014-26(2)-2

[2]. И. Бурдонов, А. Косачев. Исследование графа взаимодействующими автоматами. «Вестник Томского государственного университета. Управление, вычислительная техника и информатика», №3, 2014, стр. 67-75.

[3]. И. Бурдонов, А. Косачев. Обход неизвестного графа коллективом автоматов. Недетерминированный случай. Труды ИСП, том 27, вып. 1, 2015, стр. 51-68. DOI: 10.15514/ISPRAS-2015-27(1)-4

[4]. И.Б. Бурдонов, А.С. Косачев., В.В. Кулямин. Исследование графа набором автоматов. "Программирование", 2015, №6, стр. 3-8.

[5]. И.Б. Бурдонов, А.С. Косачев. Исследование графов коллективом двигающихся автоматов. "Программная инженерия", 2016, том 7, № 12, стр. 559-567.

[6]. И.Б.Бурдонов, А.С. Косачев. Построение прямого и обратного остовов автоматами на графе. Труды ИСП РАН, том 26, вып. 6, 2014, стр. 57-62. DOI: 10.15514/ISPRAS-2014-26(6)-4

[7]. И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин. Параллельные вычисления автоматами на прямом и обратном остовах графа. Труды ИСП РАН, том 26, вып. 6, 2014, стр. 63-66. DOI: 10.15514/ISPRAS-2014-26(6)-5

[8]. И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин. Параллельные вычисления на графе. "Программирование", 2015, №1, стр. 3-20.

[9]. И. Бурдонов, А. Косачев. Мониторинг динамически меняющегося графа. Труды ИСП РАН, том 27, вып. 1, 2015, стр. 69-96. DOI: 10.15514/ISPRAS-2015-27(1)-5

[10]. И. Бурдонов, А. Косачев. Параллельные вычисления на динамически меняющемся графе. Труды ИСП РАН, том 27(2), 2015, стр. 189-220. DOI: 10.15514/ISPRAS-2015-27(2)-12

[11]. И.Б. Бурдонов, А.С. Косачев. Исследование ориентированного графа коллективом неподвижных автоматов. "Программная инженерия", 2017, том 8, № 1, стр. 16-25.

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

[13]. И. Бурдонов, А. Косачев. Размер памяти для хранения упорядоченного корневого графа. Труды ИСП РАН, том 29, вып. 2, 2017, стр. xx-xx. DOI: 10.15514/ISPRAS-2017-29(2)-1

[14]. И.Б. Бурдонов, А.С. Косачев. Исследование графа автоматом. "Программная инженерия", 2016, №11, стр. 498-508.

[15]. И.Б.Бурдонов. Обход неизвестного ориентированного графа конечным роботом. "Программирование", 2004, №4, стр.11-34.

[16]. И.Б.Бурдонов. Проблема отката по дереву при обходе неизвестного ориентированного графа конечным роботом. "Программирование", 2004, №6, стр.6-29.

[17]. David Peleg. Distributed computing — A Locality-sensitive approach. SIAM Monographs on Discrete Mathematics and Applications. 2000, 359 pp.
[18]. Garey, Michael R.; Johnson, David S. (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, ISBN 0-7167-1045-5.

A general approach to solving problems on graphs by collective automata

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

Abstract. We propose a general method to solve graph problems by a set of automata (computational agents) located in vertices of undirected ordered connected rooted graph and communicating by passing messages along graph edges. The automata are semi-robots, i.e., their internal memory size is sufficient to store values depending on the number of vertices and edges of the graph (n and m, correspondingly) but is too small to store the complete graph description. Section 2 presents classification of graph-based distributed computational models depending on internal memory size of vertex automaton, vertex automaton operation time, and edge capacity (the number of messages that are passing along an edge simultaneously). We choose for further use the model of maximum parallelism, having zero automaton operation time and unbounded edge capacity. It allows to obtain lower complexity bounds on distributed graph problems. Section 3 describes algorithm complexity assessment rules. Section 4 presents basic message processing procedures and message classification according to paths passed by them and methods of message processing by vertex automaton. We propose to construct algorithms solving graph problems on the base of the procedures considered, and present some examples in further sections. Sections 5-9 describe distributed algorithms for spanning tree construction, for task termination detection (based on detection of absence of messages used by the task), for shortest paths tree construction, for graph vertices enumeration, for collecting graph structure information in the root automaton memory (if it is unbounded). The algorithms proposed has linear time complexity in n and m, and use linear in n and m number of messages. Section 10 considers construction of maximum weight path in a weighted graph, which is the NP problem. We propose the algorithm that for the sake of using unbounded number of messages can solve this problem in linear time in n and m. The conclusion summarizes the paper and draws directions of further research: constructing algorithms for other graph problems and generalization of the approach to directed, non-deterministic and dynamic graphs.

Keywords: undirected graphs; graph problems; asynchronous distributed systems; distributed algorithms.

DOI: 10.15514/ISPRAS-2017-29(2)-2

For citation: Burdonov I.B., Kossatchev A.S. A general approach to solving problems on graphs by collective automata. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 2, 2017, pp. 27-76. DOI: 10.15514/ISPRAS-2017-29(2)-2

References

[1]. I. Burdonov, A. Kosachev. Graph learning by a set of automata. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 2, 2014, pp. 43-86 (in Russian). DOI: 10.15514/ISPRAS-2014-26(2)-2
[2]. I. Burdonov, A. Kosachev. Analysis of a Graph by a Set of Interacting Automata. «Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitel'naja tehnika i informatika» [Tomsk State University. Journal of Control and Computer Science], №3, 2014, pp. 67-75. (in Russian).
[3]. I. Burdonov, A. Kosachev. Graph learning by a set of automata. The nondeterministic case. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 1, 2015, pp. 51-68, (in Russian). DOI: 10.15514/ISPRAS-2015-27(1)-4
[4]. I.B. Bourdonov, A.S. Kossatchev, V.V. Kulyamin. Analysis of a Graph by a Set of Automata. Programming and Computer Software, Vol. 41, No. 6, 2015, pp. 307-310. DOI: 10.1134/S0361768815060031
[5]. I.B. Burdonov, A.S. Kosachev. Analysis of a Graph by a Set of Moving Automata. "Programmnaja inzhenerija" [Software Engineering], 2016, Vol. 7, № 12, pp. 559-567, (in Russian).
[6]. I.B.Burdonov, A.S. Kosachev. Building direct and back spanning trees by automata on a graph. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 6, 2014, pp. 57-62, (in Russian). DOI: 10.15514/ISPRAS-2014-26(6)-4
[7]. I.B.Burdonov, A.S.Kosachev, V.V.Kuljamin. Parallel calculations by automata on direct and back spanning trees of a graph. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 6, 2014, pp. 63-66 (in Russian). DOI: 10.15514/ISPRAS-2014-26(6)-5
[8]. I.B. Bourdonov, A.S. Kossatchev, V.V. Kulyamin. Parallel Computations on a Graph. Programming and Computer Software, Vol. 41, No. 1, 2015, pp. 1-13. DOI: 10.1134/S0361768815010028
[9]. I. Burdonov, A. Kosachev. Monitoring of dynamically changed graph. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 1, 2015, pp. 69-96 (in Russian). DOI: 10.15514/ISPRAS-2015-27(1)-5
[10]. I. Burdonov, A. Kosachev. 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
[11]. I.B. Burdonov, A.S. Kosachev. Analysis of an Oriented Graph by a Set of Motionless Automata. "Programmnaja inzhenerija" [Software Engineering], 2017, Vol. 8, № 1, pp. 16-25, (in Russian).
[12]. I. B. Bourdonov, A.S. Kossatchev, V V. Kuliamin. Irredundant Algorithms for Traversing Directed Graphs: The Nondeterministic Case. Programming and Computer Software, Vol. 30, No. 1, 2004, pp. 2-17. DOI: 10.1023/A:1025733107700
[13]. I. Burdonov, A. Kosachev. Size of the memory for storage of ordered rooted graph. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 2, 2017, pp. xx-xx (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-1
[14]. I.B. Burdonov, A.S. KosachevAnalysis of a Graph by an Automaton. "Programmnaja inzhenerija" [Software Engineering], 2016, №11, pp. 498-508, (in Russian).
[15]. I.B. Bourdonov. Traversal of an Unknown Directed Graph by a Finite Robot. Programming and Computer Software, Vol. 30, No. 4, 2004, pp. 188-203. DOI: 10.1023/B:PACS.0000036417.58183.64
[16]. I. B. Bourdonov. Backtracking Problem in the Traversal of an Unknown Directed Graph by a Finite Robot. Programming and Computer Software, Vol. 30, No. 4, 2004, pp. 305-322. DOI: 10.1023/B:PACS.0000049509.66710.3a
[17]. David Peleg. Distributed computing — A Locality-sensitive approach. SIAM Monographs on Discrete Mathematics and Applications. 2000, 359 pp.
[18]. Garey, Michael R.; Johnson, David S. (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, ISBN 0-7167-1045-5.