И. Бурдонов, А. Косачев.
Распределённые алгоритмы на корневых неориентированных графах.
Труды Института системного программирования РАН. Том ???, выпуск ???, 2017, стр. ???- ???.
ISSN 2220-6426 (Online), ISSN 2079-8156 (Print).
??? стр.

Распределённые алгоритмы на корневых неориентированных графах

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

Аннотация. Рассматриваются распределённые алгоритмы решения задач на неориентированных графах. В разделе 2 определяется используемая модель, особенностью которой является наличие корня, с которого начинается и в котором заканчивается работа алгоритма. Описываются синхронная и асинхронная разновидности модели. В разделе 3 предлагаются алгоритмы решения любых задач, основанные на сборе информации о всём графе в корне или в каждой вершине, а также, если необходимо, разметке графа (его вершин и/или рёбер). Акцент сделан на времени работы алгоритма, а при минимальном времени – на экономии памяти в вершинах и суммарном объёме пересылаемых сообщений. В остальной части статьи рассматриваются оптимизации для конкретных задач: построение максимального независимого множества (MIS – Maximal Independent Set), поиск множества всех мостов в графе (FSB – Finding Set of Bridges), построение минимального остовного дерева во взвешенном графе (MST – Minimum Spanning Tree). В разделе 4 предлагается модификация общих алгоритмов для этих задач, уменьшающая оценки размера памяти вершин и сообщений. Раздел 5 содержит нижние оценки сложности решения этих задач. В разделе 6 для синхронной модели уменьшается время работы алгоритмов с разметкой графа до нижней границы для задач с однозначным решением, зависящим только от простых циклов графа, в частности, FSB, MST и задачи поиска гальмильтонова цикла. В разделе 7 рассматриваются оптимальные по времени алгоритмы для FSB и MST для обеих моделей: синхронной и асинхронной. Заключение подводит итоги и намечает направления дальнейших исследований.

Ключевые слова: корневой неориентированный граф; распределённые алгоритмы; задачи на графах; максимальное независимое множество; минимальное остовное дерево; поиск мостов.

И. Бурдонов, А. Косачев.
Распределённые алгоритмы на корневых неориентированных графах.
Труды Института системного программирования РАН. Том ???, выпуск ???, 2017, стр. ???- ???.
ISSN 2220-6426 (Online), ISSN 2079-8156 (Print).
??? стр.

1. Введение

Исследование алгоритмов решения задач на графах имеет давнюю историю. Первоначально такие алгоритмы были чисто последовательными, но со второй половины XX века всё больше внимания стало уделяться методам распараллеливания, позволяющим уменьшить время решения. Общий принцип параллельных графовых алгоритмов заключается в том, что граф подразделяется на меньшие локальные области, в которых задачи решаются параллельно, выполняя идентичный алгоритм. Вначале использовались модели типа *PRAM* (*Parallel Random-Access Machine*) [1], в которой процессоры имеют разделяемую общую память, через которую взаимодействуют друг с другом. Это модель, скорее, параллельной, чем распределённой обработки. Впоследствии исследования распространились на случай распределенных алгоритмов на компьютерных кластерах, когда общей памяти нет, но процессоры взаимодействуют друг с другом с помощью обмена сообщениями, используя заданную сеть связи. В конечном итоге такую сеть стали считать совпадающей с графом, на котором решается графовая задача. В вершинах графа находятся вычислительные единицы, которые назывались по-разному: процессами, процессорами, автоматами. Они не имеют общей памяти, но могут обмениваться между собой сообщениями, передаваемыми по рёбрам графа. В ориентированном графе сообщения передаются по рёбрам в направлении их ориентации, а в неориентированном графе – в обоих направлениях. Можно сказать, что такая графовая сеть исследует саму себя, решая те или иные задачи на этом графе, а «локальные области», на которые разбивается граф, сужаются до одиночных вершин, автоматы которых работают параллельно, выполняя идентичный алгоритм.

В качестве характерного примера проследим краткую историю решения задачи о максимальном независимом множестве. Подмножество вершин

неориентированного графа называется *независимым*, если никакие две вершины из подмножества не соединены ребром. *Максимальным независимым множеством*, сокращённо *MIS* (*Maximal Independent Set*), называется независимое множество, являющееся максимальным элементом в семействе всех независимых множеств по отношению вложенности. Отметим, что в русскоязычной литературе часто используется не вполне удачная терминология, когда такое множество называется наибольшим, а максимальным называется наибольшее множество, в котором наибольшее число вершин (соответствует англ. термину *MaxIS – Maximum Independent Set*).

Проблема построения MIS – одна из основных проблем в теории графов и теории распределённых алгоритмов на графах. Многие другие проблемы могут быть сведены к проблеме MIS, например, раскраска графа и поиск максимального паросочетания. Другие проблемы, хотя и не сводятся к проблеме MIS, но также тесно связаны с ней, поскольку алгоритмы их решения используют алгоритм построения MIS как подпрограмму. Примером могут служить задача о вершинном покрытии или поиск максимальной клики.

Следует отметить, что проблемы построения MaxIS и перечисления всех MIS NP-трудные. В противоположность этому проблема MIS легко решается тривиальным последовательным алгоритмом: сначала линейно упорядочиваем множество вершин графа, затем строим множество, добавляя к нему вершину, если она не смежна ни с какой вершиной множества. Результатом будет MIS, которое называется *лексикографическим MIS* (*LFMIS*), поскольку при построении используется заданный порядок вершин.

Задача поиска LFMIS P-полна, поэтому сначала думали, что эта задача трудна для распараллеливания. Однако было показано, что детерминированное параллельное решение можно получить с помощью NC^1 редукции из решения

проблемы максимальной упаковки множеств или проблемы максимального паросочетания, либо же редукцией из решения проблемы 2-*SAT* (задача выполнимости булевых формул в 2-конъюнктивной нормальной форме) [2][3]. Напомним, что через NC^i обозначается класс сложности проблем, разрешимых за время $\boldsymbol{O}(log^i n)$ на параллельном компьютере с полиномиальным числом процессоров, где n – число вершин графа. В 1984 Karp и др. показали, что для модели PRAM детерминированное параллельное решение проблемы MIS принадлежит классу сложности NC^4 [4]. Их алгоритм находит MIS за время $\boldsymbol{O}(log^4 n)$, используя $\boldsymbol{O}((n/logn)^3)$ процессоров, где n – число вершин графа. В той же статье предложено рандомизированное параллельное решение с временем выполнения $\boldsymbol{O}(log^4 n)$ и числом процессоров $\boldsymbol{O}(n^2)$. Через 2 года после этого Luby и Alon и др. независимо улучшили этот результат, перенеся проблему MIS в область NC^2 с временем выполнения $\boldsymbol{O}(log^2 n)$ и число процессоров $\boldsymbol{O}(mn^2)$, где m – количество рёбер в графе [5][6]. Они предложили рандомизированный алгоритм, который использует $\boldsymbol{O}(m)$ процессоров, но может быть дерандомизирован с дополнительными $\boldsymbol{O}(n^2)$ процессорами для каждого из первоначальных m процессоров.

Работа Luby и Alon и др. инициировала исследования по распределённым алгоритмам [7][8][9]. В [7] Peleg предложил модель распределённых алгоритмов LOCAL, в которой нет общей памяти процессоров и которая является графово-ориентированной: процессоры находятся в вершинах графа взаимодействуют друг с другом только обменом сообщениями, передаваемым по рёбрам графа. Тем самым проблема MIS решается для графа, который является графом связи процессоров в модели LOCAL. Предлагавшиеся алгоритмы имели размер сообщения с нижней границей $\boldsymbol{O}(logn)$ битов, и требовали знания дополнительных характеристик графа. Например, должен

быть известен размер графа, или для данной вершины можно было узнать максимальную степень соседних вершин. В 2016 г. Levi и др. сумел и уменьшить размер сообщения до $\boldsymbol{O}(1)$, который минимален, и устранили необходимость любого дополнительного знания о графе [10].

Одним из последних достижений в этой области стала работа Ghaffari, доложенная на симпозиуме SODA и опубликованная в 2016 г. [11], но выложенная в интернет на сайте Корнеллского университета в 2015 г. [12]. В рамках модели LOCAL его рандомизированный алгоритм с вероятностью не менее $1-1/n$ строит MIS за время $\boldsymbol{O}(log\Delta)+2^{\boldsymbol{O}(\sqrt{(loglogn)})}$, где Δ – максимальная степень вершины. Это лучше, чем предыдущий результат $\boldsymbol{O}(log^2\Delta)+2^{\boldsymbol{O}(\sqrt{(loglogn)})}$ в [13], и уже близок к нижней оценке $\Omega(min\{log\Delta, \sqrt{(logn)}\})$, доказанной в [14].

LOCAL является синхронной моделью: автоматы во всех вершинах срабатывают одновременно, принимая все посланные им сообщения и посылая новые сообщения; это называется *раундом*. При этом время перемещения сообщения по ребру фиксировано (1 такт), а временем срабатывания автомата, т.е. временем вычислений в вершинах пренебрегают. Время работы алгоритма считается в числе раундов (или тактов). В асинхронных моделях сообщение может двигаться по ребру произвольное ограниченное сверху (1 тактом) время, поэтому время работы алгоритма считается в тактах и в «наихудшем» случае. Для некоторых алгоритмов это существенно: существуют синхронные алгоритмы, которые в асинхронной модели либо не работают, либо имеют другие временные оценки. В данной статье мы рассматриваем оба типа модели, хотя некоторые предлагаемые алгоритмы применимы только в синхронном варианте.

Но более важной особенностью как системы LOCAL, так и некоторых асинхронных моделей является предположение об одновременном начале

работы автоматов в вершинах. В противоположность этому существуют модели, в которых работа начинается с одной вершины, называемой *корнем* графа. Остальные вершины «спят», а «просыпаются» только при получении сообщений, посылаемых в конечном счёте от корня. В данной статье мы рассматриваем как раз такие *корневые* модели. В этих моделях существенно то, что время работы алгоритма учитывает время начального распространения сообщений от корня до остальных вершин (*broadcast*). Это время в синхронной модели равно, а в асинхронной не превышает эксцентриситета корня d_0 (максимального расстояния от корня до других вершин). Более того, мы будем считать, что работа алгоритма не только начинается, но и заканчивается в корне. Это значит, что, используя завершающее распространение сообщений от всех вершин к корню (*convergecast*), корень должен «узнать» о завершении решения задачи в каждой вершине, что требует такого же времени d_0. Тем самым, время работы алгоритма для задач, решение которых зависит от всего графа (в частности, проблемы MIS), имеет нижнюю границу $2d_0 = \boldsymbol{O}(n)$. Это делает неактуальным для корневых моделей многие алгоритмы, разработанные для модели LOCAL.

Данная статья – вторая в серии статей, начатой нами в [15]. Там был предложен общий подход к распределенному решению задач на графах, основанный на систематизации способов распространения сообщений по графу и построения типовых конструкций, в частности, остова графа.

В разделе 2 описывается общая корневая модель распределённых алгоритмов для неориентированных графов в синхронном и асинхронном вариантах, отмечаются её существенные отличия от других моделей, прежде всего, модели LOCAL. В разделе 3 рассматриваются алгоритмы решения любых задач на неориентированных нумерованных графах, основанные на сборе

информации о всём графе в корне (*broadcast+convergecast*) или в каждой вершине (*broadcast+all-to-all-broadcast+convergecast*). Акцент сделан на времени работы алгоритма, а при минимальном времени – на экономии памяти автоматов в вершинах и суммарном объёме пересылаемых сообщений как по одному ребру, так и по всем рёбрам, как одновременно, так и за всё время работы алгоритма. Эти алгоритмы различаются, прежде всего, типом модели: синхронная или асинхронная. Кроме того, задача решается без разметки графа или с разметкой графа. В первом случае корень формирует ответ в виде сообщения вовне графа. Например, для MIS – как множество номеров вершин из MIS. Во втором случае решение задачи – это разметка графа, при которой помечаются некоторые вершины и/или рёбрам графа, а корень сообщает вовне только о том, что задача решена и разметка выполнена. Например, для MIS помечаются вершины, входящие в MIS.

Общие алгоритмы, пригодные для решения любых задач на графах, конечно, могут оказаться не оптимальными по времени и/или памяти при решении той или иной конкретной задачи. В остальной части статьи рассматриваются различные оптимизации на примере трёх важных графовых задач. Одна из них – это проблема MIS. Вторая – задача поиска множества всех мостов в графе – FSB (*Finding Set of Bridges*). Напомним, что мост – это ребро графа, удаление которого увеличивает число компонентов связности графа. Третья – задача построения минимального остовного дерева во взвешенном графе – MST (*Minimum Spanning Tree*). Напомним, что MST – это остовное дерево графа, имеющее минимальный возможный вес, где под весом дерева понимается сумма весов входящих в него рёбер.

В разделе 4 предлагается модификация общих алгоритмов для этих трёх задач, уменьшающая оценки размера памяти вершин и сообщений. Раздел 5

содержит нижние оценки сложности алгоритмического решения задач MIS, FSB и MST. В разделе 6 показано, что в синхронной модели можно уменьшить время работы алгоритмов с разметкой графа до нижней границы для задач с однозначным решением, зависящим только от простых циклов графа. К таким задачам относятся, в частности, задачи FSB, MST и задача поиска гальмильтонова цикла. В разделе 7 в обеих моделях (синхронной и асинхронной) рассмотрены алгоритмы для задач FSB и MST, в которых удаётся совместить начальное (*broadcast*) и завершающее (*convergecast*) распространение сообщений с решением задачи и разметкой графа за минимально возможное время. Заключение подводит итоги и намечает направления дальнейших исследований.

2. Две модели распределённых алгоритмов

Используемая нами модель похожа на модель LOCAL: автоматы находятся в вершинах графа и обмениваются между собой сообщениями, которые пересылаются по рёбрам графа. Для краткости вместо «автомат в вершине» мы будем часто писать просто «вершина». Граф предполагается неориентированным, связным, без кратных рёбер и петель, с выделенной вершиной – корнем графа.

Ребро понимается как дуплексный канал передачи сообщений. Для того, чтобы вершина могла указать ребро, по которому она посылает сообщение, все инцидентные вершине рёбра считаются пронумерованными от 1 до степени вершины. Такой граф называется *упорядоченным* [15][16]. При посылке сообщения по ребру вершина указывает номер ребра. Получая сообщение, вершина «узнаёт» номер ребра, по которому сообщение получено.

Основное отличие от системы LOCAL в том, что алгоритм начинает работать не одновременно во всех вершинах графа, а только с корня. Другие вершины

подключаются к решению задачи после того, как получат сообщения, передаваемые по путям от корня до этих вершин. Соответственно, задача считается решённой только после того, как об этом «узнал» корень. Будем считать, что корень связан с «окружением» фиктивным ребром для получения извне сообщения, инициирующего решение задачи, и для посылки вовне сообщения о завершении решения задачи. Такую модель будем называть *корневой* (rooted).

Мы будем рассматривать две разновидности корневой модели: синхронную – *RSM* (Rooted Synchronous Model) и асинхронную – *RAM* (Rooted Asynchronous Model). В синхронной модели, так же как в системе LOCAL, сообщение перемещается по ребру фиксированное время – 1 такт. В асинхронной модели сообщение передаётся по ребру, вообще говоря, произвольное время, ограниченное сверху 1 тактом. В *RAM* время передачи сообщения по ребру может быть различным для разных рёбер, для разных направлений на одном ребре, для разных сообщений на одном ребре в одном направлении и, вообще говоря, меняться со временем.

В обеих моделях за одно срабатывание автомата в вершине (*раунд*) принимаются все дошедшие до вершины сообщения и посылается одно или несколько сообщений по одному или нескольким рёбрам. Предполагается, что сообщения не теряются, не генерируются рёбрами, и сообщения, посланные по одному ребру и ещё не принятые другим его концом, не обгоняют друг друга, т.е. выстраиваются в очередь на ребре. Как и в системе LOCAL мы пренебрегаем временем срабатывания автоматов в вершинах графа. Поэтому посылка по одному ребру нескольких сообщений в одном раунде эквивалентна посылке одного «склеенного» сообщения.

Труды Института системного программирования РАН. Том ???, выпуск ???, 2017, стр. ???- ???.
ISSN 2220-6426 (Online), ISSN 2079-8156 (Print).
??? стр.

В *RSM* как и в LOCAL между раундами проходит ровно один такт, и время работы алгоритма можно считать как число раундов (*time complexity* в [9]). В *RAM* время между раундами может быть произвольным в интервале (0,1], поэтому нет смысла считать время работа алгоритма в раундах, оно оценивается в тактах в «наихудшем случае» в зависимости от времени передачи по рёбрам тех или иных сообщений в те или иные моменты времени.

Будем считать, что сообщение представляет собой набор параметров, а память вершины (автомата в вершине) – набор переменных, сохраняемых между раундами, т.е. без учёта памяти для приёма сообщений и формирования сообщений перед посылкой (эта память ограничена размером сообщения). Имена параметров будем писать *курсивом строчными буквами*, а имена переменных – ***жирным курсивом строчными буквами***.

Мы будем давать следующие оценки алгоритмов: T – время работы алгоритма, A – размер памяти вершины как сумма размеров переменных без учёта памяти для принимаемых и посылаемых сообщений, M_1 – максимальный суммарный размер сообщений, находящихся на одном ребре одновременно, M_{all} – максимальный суммарный размер сообщений, находящихся на всех рёбрах одновременно, S_1 – максимальный суммарный размер сообщений, проходивших через одно ребро за всё время работы алгоритма, S_{all} – максимальный суммарный размер сообщений, проходивших через все рёбра за всё время работы алгоритма (если сообщение проходит k рёбер, оно считается k раз).

Эти оценки будут даваться как функции от следующих параметров графа: n – число вершин, Δ – максимальная степень вершины, D – длина максимального пути, 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. Очевидно, $\Delta \le n\text{-}1$, $d_0 \le d \le 2d_0$, $D_0 \le D \le 2D_0$, $d_0 \le D_0$, $d \le D \le n\text{-}1$ [15] и, если $d_0 > 1$ и $\Delta > 2$, то $d_0+\Delta\text{-}1 \le n \le ((\Delta\text{-}1)^{d_0+1}\text{-}1)/(\Delta\text{-}2)$.

В предлагаемых ниже алгоритмах граф, как правило, будет предполагаться *нумерованным* [15][16]: его вершинам присвоены номера от 1 до n. Номер вершины хранится в её переменной с самого начала.

3. Общий случай

В этом разделе мы рассмотрим общие алгоритмы для решения любой задачи на графе в моделях $\mathcal{RSM}$ и $\mathcal{RAM}$. Мы будем различать решения без разметки графа и решения с разметкой графа. Если задача решается без разметки графа, то решение содержится в завершающем сообщении, которое корень посылает вовне в конце работы алгоритма. Если задача решается с разметкой графа, то решение – это глобальная разметка графа: вершинам и рёбрам присваиваются некоторые метки, а завершающее сообщение, посылаемое корнем вовне, только сообщает о том, что разметка выполнена. При этом каждая вершина выполняет локальную разметку: инициализирует метку вершины, задаваемую специальной переменной в вершине, и метки инцидентных вершине рёбер, задаваемые в другой переменной, содержащей отображение номера ребра в этой вершине на значение метки ребра. Поскольку ребро инцидентно двум вершинам (концам ребра), оно получает две метки, по одной в каждом из своих концов (обычно эти метки одинаковые).

Мы предлагаем алгоритмы с возможно меньшим временем работы T, а при данном T – с возможно меньшими размерами памяти вершины и сообщений.

3.1. Три типа алгоритмов и пять типов сообщений

Мы предложим три типа общих алгоритмов:

1. *Сбор в корне*. Из корня рассылается во все вершины сообщение, инициирующее сбор информации о графе. В корне графа собирается информация о всём графе, после этого в корне решается задача и посылается вовне сообщение с решением. Граф не размечается. Например, для MIS завершающее сообщение содержит множество номеров вершин из MIS, а для MST или FSB – множество рёбер из MST или множество мостов, соответственно.

2. *Сбор в корне и разметка из корня*. Выполняется сбор информации о графе в корне, решается задача, а затем выполняется глобальная разметка графа в соответствии с найденным решением. Для разметки из корня рассылается во все вершины информация о разметке, по которой каждая вершина выполняет свою локальную разметку, после чего извещает корень о завершении локальной разметки в этой вершине. Глобальная разметка завершена, когда корень получит извещения от всех вершин. Корень посылает вовне завершающее сообщение о том, что задача решена и граф размечен. Например, для MIS помечаются вершины из MIS, а для MST или FSB – рёбра из MST или мосты, соответственно.

3. *Сбор и разметка во всех вершинах*. Из корня рассылается во все вершины сообщение, инициирующее сбор информации о графе. Информация о графе собирается не только в корне, а в каждой вершине графа. Собрав всю информацию о графе, каждая вершина самостоятельно решает задачу и выполняет свою локальную разметку, после чего извещает корень о завершении локальной разметки в этой вершине. Важное требование: алгоритм должен гарантировать, что все вершины находят одно и то же

решение задачи. Получив извещения от всех вершин, корень посылает вовне завершающее сообщение о том, что задача решена и граф размечен.

Информация о графе, собираемая в корне (алгоритмы типа **1** и **2**) или в каждой вершине (алгоритмы типа **3**), в общем случае описывает все вершины и рёбра графа. Некоторые задачи требуют взвешенных графов, в которых вершинам и/или рёбрам приписаны веса. Для определённости будем считать, что вес – целое число от 1 до максимального значения w. В дальнейшем оценки алгоритмов приводятся для случая невзвешенных графов, для взвешенных графов размер описания вершин и/или рёбер увеличивается не более чем в $\log w$ раз. Также в дальнейшем считается, что метка вершины и/или ребра принимает два значения: «помечено», «не помечено». В более общем случае размер информации о разметке увеличивается не более чем в $\log u$ раз, где u – число различных значений метки.

Алгоритмы этих трёх типов используют сообщения пяти типов:

1. ***Старт*** – сообщение от корня в каждую вершину (*Broadcast*), инициирующее сбор информации о графе.
2. ***ИнфоКорню*** – сообщение от каждой вершины в корень (*Convergecast*), содержащее информацию о соседях вершины.
3. ***Разметка*** – сообщение от корня в каждую вершину (*Broadcast*), содержащее информацию о разметке: какие рёбра и вершины нужно пометить.
4. ***ИнфоВсем*** – сообщение от каждой вершины в каждую вершину (*All-to-All Broadcast*), содержащее информацию о соседях вершины.
5. ***Финиш*** – сообщение от каждой вершины в корень (*Convergecast*) об окончании локальной разметки в этой вершине.

Труды Института системного программирования РАН. Том ???, выпуск ???, 2017, стр. ???- ???.
ISSN 2220-6426 (Online), ISSN 2079-8156 (Print).
??? стр.

Сообщения, используемые разными типами алгоритмов:

1. *Сбор в корне*: ***Старт*** + ***ИнфоКорню***.
2. *Сбор в корне и разметка из корня*: ***Старт*** + ***ИнфоКорню*** + ***Разметка*** + ***Финиш***.
3. *Сбор и разметка во всех вершинах*: ***Старт*** + ***ИнфоВсем*** + ***Финиш***.

3.2. Пять классов сообщений по способу их передачи

В [15] определены 9 способов передачи сообщений. В данной статье нам достаточно 5 из них, мы определим их в этом подразделе. Оценки этих способов приведены в Таб. 1, где m означает максимальный размер сообщения. Как функции от m и n эти оценки достигаются на графе на Рис. 1.

Таб. 1.Оценки способов передачи сообщений.

Tabl. 1. Estimations of the methods of transmitting messages.

	P		***O***	***C***	***П***	***$M_к$***		***$M_в$***	
MODEL	*RSM*	*RAM*	*RSM*	*RSM*	*RSM*	*RSM*	*RAM*	*RSM*	*RAM*
A	$\boldsymbol{O}(\log\Delta) = \boldsymbol{O}(\log n)$				$\boldsymbol{O}(\Delta) = \boldsymbol{O}(n)$	$\boldsymbol{O}(n)$			
M_1 and S_1	$\boldsymbol{O}(m)$		$\boldsymbol{O}(nm)$	$\boldsymbol{O}(m)$		$\boldsymbol{O}(nm)$			
M_{all}	$\boldsymbol{O}(n\Delta m)$			$\boldsymbol{O}(nm)$		$\boldsymbol{O}(n^2\Delta m) = \boldsymbol{O}(n^3 m)$			
S_{all}	$= \boldsymbol{O}(n^2 m)$		$\boldsymbol{O}(nd_0 m) = \boldsymbol{O}(n^2 m)$						
T	$= d_0$	$\le d_0$	$= d_0$			$\le d_0$		$= d \le 2d_0$	$\le d \le 2d_0$
T*	$\le d_0+1$								

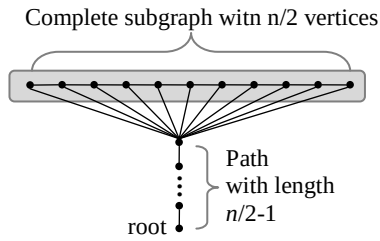

Рис. 1. Графы, где достигаются верхние оценки M_1, S_1, M_{all}, S_{all}.

Fig. 1. Graphs where the upper bounds M_1, S_1, M_{all}, S_{all} are reached.

1. Класс ***P*** – рассылка без повторения из корня. Вначале сообщение создаётся корнем и посылается по всем инцидентным корню рёбрам. Когда некорневая вершина получает сообщение первый раз по некоторому ребру, она пересылает его по всем инцидентным вершине рёбрам. Поскольку в одном раунде вершина может получить сообщения сразу по нескольким рёбрам, порядок их обработки вершиной, в том числе, выбор «первого» из них, недетерминированы. Повторно получаемые вершиной сообщения дальше не пересылаются. Для различения первого и повторных сообщений используется булевская переменная ***было***; вначале ***было*** = ***false***, а при получении первого сообщения ***было*** := ***true***. Для определения номеров рёбер, по которым нужно посылать сообщение, используется переменная ***степень*** вершины размером ***O***(*log*Δ).

Рёбра, по которым сообщение впервые попадает в вершины, образуют остов графа. Если для ребра *ab* этого остова вершина *a* расположена по остову ближе к корню, чем вершина *b*, то есть вершина *b* получила первое сообщение от вершины *a*, это ребро будем называть *прямым* в вершине *a* и *обратным* в вершине *b*. Каждой некорневой вершине инцидентно ровно одно обратное ребро, и каждой нелистовой вершине (включая корень) инцидентно хотя бы одно прямое ребро. Остальные рёбра графа будем называть *хордами* остова.

Класс ***P*** реализует *broadcast* от корня до всех вершин с минимальным временем T, которое в $\mathcal{RSM}$ равно, а в $\mathcal{RAM}$ не превосходит d_0. Для того чтобы сообщение прошло по всем рёбрам, нужно ещё не более 1 такта, общее время $T^* \le T+1$. По каждому ребру в каждом направлении пройдёт ровно одно сообщение.

В описываемых ниже алгоритмах используются дополнительные опции класса ***P***, которые мы будем вводить по ходу изложения.

2. Класс ***O*** – пересылка по обратным рёбрам до корня. Сообщение этого класса проходит путь по обратным рёбрам от вершины, где оно создано, до корня. Для этого в каждой некорневой вершине сохраняется ***номер обратного ребра*** размером $\boldsymbol{O}(log\Delta)$. Эту переменную инициализирует опция *O* класса ***P*** как номер ребра, по которому вершина первый раз получает сообщение класса ***P***.

Остов, создаваемый обратными рёбрами, в $\mathcal{RSM}$ является деревом кратчайших путей и имеет высоту d_0. Однако в $\mathcal{RAM}$ это не обязательно так, там этот остов может иметь высоту до D_0. Поэтому класс ***O*** нам пригодится только для алгоритмов в $\mathcal{RSM}$, где он реализует *convergecast* с минимальным временем $T = d_0$. Для оценок в Таб. 1 предполагается, что каждая некорневая вершина создаёт сообщение, т.е. всего будет n-1 сообщений, а в графе на Рис. 1 все вершины полного подграфа могут создать сообщения одновременно. Поэтому по некоторым рёбрам остова может пройти, в том числе одновременно, $\boldsymbol{O}(n)$ сообщений, и каждое из таких сообщений может пройти путь длиной до d_0.

3. Класс ***C*** – сбор по остову. Это способ передачи сообщений от всех вершин к корню по обратным рёбрам, но в отличие от класса ***O***, с проходом по каждому обратному ребру ровно одного сообщения: сообщение посылается из вершины по обратному ребру только тогда, когда получены сообщения по всем прямым рёбрам. Для этого в каждой некорневой вершине имеется ***номер обратного ребра*** и ***число прямых рёбер*** оба размером $\boldsymbol{O}(log\Delta)$. Первую переменную инициализирует опция *O* класса ***P*** как номер ребра, по которому вершина первый раз получает сообщение класса ***P***. Вторую переменную инициализирует опция *C* класса ***P***. Для этого в сообщении класса ***P*** есть булевский параметр *признак остова*, который равен ***true*** тогда, когда сообщение посылается по обратному ребру. Вершина подсчитывает число полученных сообщений класса ***P*** с *признак остова* = ***true*** в переменной ***число***

прямых рёбер. (Вместо опции *C* класса ***P*** может использоваться опция *П*, устанавливающая шкалу прямых рёбер, которую потом можно превратить в ***число прямых рёбер***. См. ниже класс ***П*** и алгоритмы типа **2** в 3.3.2.)

Остов из обратных рёбер в *RSM* является деревом кратчайших путей и имеет высоту d_0. Однако в *RAM* это не обязательно так, там этот остов может иметь высоту до D_0. Поэтому класс ***C*** нам пригодится только для алгоритмов в *RSM*, где он реализует *convergecast* с минимальным временем $T = d_0$.

4. Класс ***П*** – рассылка по прямым рёбрам от корня. Сообщение этого класса рассылается, начиная с корня, по прямым рёбрам до листовых вершин остова. Для этого в каждой нелистовой вершине (включая корень) имеется битовая ***шкала прямых рёбер*** размером ***O***(Δ), содержащая «1» в *i*-ом разряде, если ребро с номером *i* прямое. Эту переменную инициализирует опция *П* класса ***P***. Сообщение класса ***P***, как и в случае опции *C*, имеет булевский параметр *признак остова*, который равен ***true*** тогда, когда сообщение посылается по обратному ребру. Получая по ребру сообщение класса ***P*** с *признак остова* = ***true***, вершина отмечает это ребро как прямое в шкале прямых рёбер.

Остов, создаваемый прямыми рёбрами, в *RSM* является деревом кратчайших путей и поэтому имеет высоту d_0. Однако в *RAM* это не обязательно так, там этот остов может иметь высоту до D_0. Поэтому класс ***П*** нам пригодится только для алгоритмов в *RSM*, где он реализует *broadcast* с минимальным временем $T = d_0$. По каждому ребру остова пройдёт ровно одно сообщение и только в направлении «прямое» (от корня).

5. Класс ***M*** – множественная рассылка. Множественная рассылка – это рассылка без повторения, которая параллельно ведётся, начиная с нескольких вершин, которые мы назовём *инициаторами*. Сообщение должно дойти либо из каждой вершины в каждую вершину (класс $\boldsymbol{M_в}$), либо из каждой вершины в

корень (класс $\boldsymbol{M_к}$). Для класса $\boldsymbol{M_в}$ инициаторы – это все вершины, а для класса $\boldsymbol{M_к}$ – все вершины, кроме корня. Для того чтобы различать сообщения от разных инициаторов, используется ***номер инициатора***, который хранится в каждой вершине как её номер и имеет размер $\boldsymbol{O}(\log n)$, т.е. граф предполагается нумерованным. *Номер инициатора* также является параметром сообщения. Кроме того, в вершине для каждого инициатора должна быть своя переменная ***было***, т.е. вершина хранит множество номеров инициаторов, сообщения от которых были в вершине, в виде ***шкалы инициаторов*** размером $\boldsymbol{O}(n)$.

Класс $\boldsymbol{M}$ реализует *all-to-all-broadcast* с минимальным временем $T = d$. По каждому ребру в каждом направлении пройдёт ровно одно сообщение от каждого инициатора. Для оценок предполагается, что в графе на Рис. 1 все вершины полного подграфа могут создать сообщения одновременно.

3.3. Алгоритмы

Оценки алгоритмов приведены в Таб. 2. Обозначения: $f = \min\{n,\Delta\log n\}$, P – размер описания графа, необходимого для решения задачи, R – размер информации о разметке. Оценки как функции от n даны для общего случая, когда $P = R = \boldsymbol{O}(nf)$, они достигаются на графе на Рис. 1.

Таб. 2.Оценки алгоритмов.

Tabl. 2. Estimations of the algorithms.

	1		**2**	**2a**	**2**	**2a**	**3**	
MODEL	*RSM*	*RAM*	*RSM*		*RAM*		*RSM*	*RAM*
Classes, ***P***(*options*)	$\boldsymbol{P}(O)+\boldsymbol{O}$	$\boldsymbol{P}+\boldsymbol{M_к}$	$\boldsymbol{P}(O\Pi)+\boldsymbol{O}$ $+\boldsymbol{\Pi}+\boldsymbol{C}$	$\boldsymbol{P}(O)+\boldsymbol{O}$ $+\boldsymbol{P}(OC)+\boldsymbol{C}$	$\boldsymbol{P}+\boldsymbol{M_к}$ $+\boldsymbol{P}(Б)+\boldsymbol{M_к}$	$\boldsymbol{P}+\boldsymbol{M_к}$ $+\boldsymbol{P}+\boldsymbol{M_к}$	$\boldsymbol{P}(OC)$ $+\boldsymbol{M_в}+\boldsymbol{C}$	$\boldsymbol{P}$ $+\boldsymbol{M_в}+\boldsymbol{M_к}$
A (not in root)	$\boldsymbol{O}(f)$ или $\boldsymbol{O}(\log n)$ *	$\boldsymbol{O}(n)$	$\boldsymbol{O}(\Delta\log n)$ $=\boldsymbol{O}(n\log n)$	$\boldsymbol{O}(f)$ или $\boldsymbol{O}(\log n)$ *	$\boldsymbol{O}(n+\Delta\log n)$ $=\boldsymbol{O}(n\log n)$	$\boldsymbol{O}(n)$	$\boldsymbol{O}(P+$ $+n+\Delta\log n)$ $=\boldsymbol{O}(n^2)$	
A (in root)	$\boldsymbol{O}(P+$ $+\log n)$ $=\boldsymbol{O}(n^2)$	$\boldsymbol{O}(P+$ $+n)$ $=\boldsymbol{O}(n^2)$	$\boldsymbol{O}(P+$ $+\Delta\log n)$ $=\boldsymbol{O}(n^2)$	$\boldsymbol{O}(P+$ $+\log n)$ $=\boldsymbol{O}(n^2)$	$\boldsymbol{O}(P+$ $+n+\Delta\log n)$ $=\boldsymbol{O}(n^2)$	$\boldsymbol{O}(P+$ $+n)$ $=\boldsymbol{O}(n^2)$		
M_1 and S_1	$\boldsymbol{O}(nf)=\boldsymbol{O}(n^2)$		$\boldsymbol{O}(R+nf)=\boldsymbol{O}(n^2)$				$\boldsymbol{O}(nf)=\boldsymbol{O}(n^2)$	
M_{all}	$\boldsymbol{O}(n\Delta\log n)$ $=\boldsymbol{O}(n^2\log n)$	$\boldsymbol{O}(\Delta n^2 f)$ $=\boldsymbol{O}(n^4)$	$\boldsymbol{O}(nR+$ $+n\Delta\log n)$ $=\boldsymbol{O}(n^3)$	$\boldsymbol{O}(n\Delta R+$ $+n\Delta\log n)$ $=\boldsymbol{O}(n^4)$	$\boldsymbol{O}(n\Delta R+$ $+\Delta n^2 f)$ $=\boldsymbol{O}(n^4)$		$\boldsymbol{O}(\Delta n^2 f)$ $=\boldsymbol{O}(n^4)$	
S_{all}	$\boldsymbol{O}(n\Delta\log n+$ $+d_0 nf)$ $=\boldsymbol{O}(n^3)$		$\boldsymbol{O}(n\Delta\log n+$ $+nR+d_0 nf)$ $=\boldsymbol{O}(n^3)$	$\boldsymbol{O}(n\Delta\log n+$ $+n\Delta R+d_0 nf)$ $=\boldsymbol{O}(n^4)$				
T	$\le 2d_0+1$		$\le 4d_0+1$	$\le 4d_0+2$	$\le 4d_0+1$	$\le 4d_0+2$	$\le 2d_0+d+1$ **	

* – при «экономной модификации», ** – очевидно, что $3d_0+1 \le 2d_0+d+1 \le 4d_0+1$.
Тёмно-серым фоном выделены оценки, в которые входят величины P или R.

3.3.1. Алгоритмы типа 1: Сбор в корне

Цель: *broadcast* с проходом по каждому ребру и *convergecast* за минимальное суммарное время $2d_0+1$. Используются сообщения ***Старт*** и ***ИнфоКорню***, которые имеют следующие классы и опции. В *RSM*: ***Старт*** – ***P*** с опцией *O* сохранения обратного ребра, ***ИнфоКорню*** – ***O***. В *RAM*: ***Старт*** – ***P*** без опций, ***ИнфоКорню*** – $\boldsymbol{M_к}$.

Граф предполагается нумерованным, каждая вершина хранит свой ***номер вершины*** размером $\boldsymbol{O}(\log n)$. Сообщение ***Старт*** имеет параметр: *номер вершины*, из которой оно посылается.

В каждой некорневой вершине имеется переменная ***множество соседей***, в которую помещается номер соседней вершины при получении сообщения

Старт. Это множество задаётся либо битовой шкалой вершин размером $\boldsymbol{O}(n)$, в которой i-ый разряд равен 1, если вершина с номером i является соседом вершины a, либо списком соседей размером $\boldsymbol{O}(\Delta log n)$, в котором j-ый элемент списка содержит номер вершины на другом конце ребра, имеющего номер j в a. Переменная ***множество соседей*** имеет размер $\boldsymbol{O}(f)$, где $f = min\{n,\Delta log n\}$.

Сообщение ***ИнфоКорню*** вершина создаёт тогда, когда получит сообщение ***Старт*** от всех соседей, что определяется сравнением мощности множества соседей и степени вершины. Это сообщение имеет параметры: *номер вершины*, которая его создала, и *множество соседей*, равное значению переменной ***множество соседей*** этой вершины.

Корень сохраняет информацию о графе в виде ***списка множеств соседей*** размером $P = \boldsymbol{O}(nf)$, в котором i-ый элемент списка – это множество соседей вершины с номером i; это множество пусто, если соседи вершины ещё не известны. Длина списка равна максимальному номеру известной корню вершины. Первоначально корню известен только его номер, а некорневая вершина становится известной корню, когда корень получает от неё сообщение ***Старт*** или ***ИнфоКорню***, или когда она описана как соседняя вершина в сообщении ***ИнфоКорню***. Когда корень r получает сообщение ***Старт*** от вершины j, он добавляет j в своё множество соседей в списке. Когда корень получает сообщение ***ИнфоКорню*** от вершины i, i-ое множество соседей из списка объединяется с множеством соседей из сообщения.

Список множеств соседей описывает весь граф, если корень получил сообщение ***Старт*** по каждому инцидентному ему ребру, и от каждой известной корню некорневой вершины, т.е. от некорневой вершины с номером i от 1 до длины списка, получено сообщение ***ИнфоКорню***, т.е. i-ое множество

соседей в списке не пусто. В этом случае корень решает задачу и посылает её решение вовне в завершающем сообщении.

Оценка времени $T = 2d_0+1$ достигается на любом графе, в котором есть ребро, соединяющее две вершины, каждая на расстоянии d_0 от корня. Такие вершины и рёбра будем называть *периферийными*. Если периферийных рёбер нет, то $T = 2d_0$. Максимальный размер сообщения: для ***Старт*** $m = \boldsymbol{O}(\log n)$, для ***ИнфоКорню*** $m = \boldsymbol{O}(f)$.

«Экономная модификация» алгоритма в *RSM*. В *RSM* можно уменьшить память вершины, если сообщение ***ИнфоКорню*** посылать «без накопления» информации о соседях, т.е. сразу при получении сообщения ***Старт***. Переменная ***множество соседей*** не нужна, а параметр *множество соседей* сообщения ***ИнфоКорню*** равен множеству номеров вершин, от которых в данном раунде вершина получила сообщение ***Старт***. Если вершина находится на расстоянии r от корня, то её соседи находятся от корня на расстоянии r-1, r или r+1. Поэтому сообщение ***ИнфоКорню*** создаётся вершиной не более чем на трёх раундах, и суммарный размер этих сообщений всё равно равен $\boldsymbol{O}(f)$. Поскольку число таких раундов колеблется от 1 до 3, последнее сообщение от данной вершины ***ИнфоКорню*** маркируется булевским параметром *последнее сообщение*. Для этого вершина подсчитывает число полученных ею сообщений ***Старт*** в переменной ***счётчик соседей*** размером $\boldsymbol{O}(\log\Delta)$: сообщение ***ИнфоКорню*** последнее, если при его создании счётчик соседей равен степени вершины. Корень дополнительно запоминает, было ли полученное сообщение ***ИнфоКорню*** последним от создавшей его вершины i, в переменной ***шкала последних*** размером $\boldsymbol{O}(n)$, i-ый разряд которой соответствует вершине i. При определении конца сбора информации корень учитывает, что от каждой

некорневой вершины должно быть получено последнее сообщение ***ИнфоКорню***. Таким образом, за счёт нескольких (но не больше трёх) сообщений ***ИнфоКорню*** от одной вершины мы экономим память вершины: вместо $A = \boldsymbol{O}(f)$ будет $A = \boldsymbol{O}(\log n)$.

Заметим, что в *RAM* такая модификация не имеет смысла. Дело в том, что в *RAM* на каждом раунде вершина может получать не более одного сообщения ***Старт***. Если создавать сообщение ***ИнфоКорню*** каждый раз при получении сообщения ***Старт***, то число таких сообщений ***ИнфоКорню*** может достигать степени вершины, а их суммарный размер достигать $\boldsymbol{O}(\Delta f)$, что увеличивает оценки M_1, M_{all}, S_1, S_{all} в $\Delta = \boldsymbol{O}(n)$ раз. Кроме того, в *RAM* сообщение ***ИнфоКорню*** посылается множественной рассылкой, а это предполагает либо одно сообщение от вершины, либо увеличение числа инициаторов в Δ раз, то есть, по сути, инициатором становится не вершина, а ребро, и размер шкалы инициаторов увеличивается с $\boldsymbol{O}(n)$ до $\boldsymbol{O}(\Delta n)$.

3.3.2. Алгоритмы типа 2: Сбор в корне и разметка из корня

Цель: сбор информации – *broadcast* с проходом по каждому ребру и *convergecast* за минимальное суммарное время $2d_0+1$, и разметка от корня – *broadcast* и *convergecast* за минимальное суммарное время $2d_0$. Используются сообщения ***Старт***, ***ИнфоКорню***, ***Разметка*** и ***Финиш***, которые имеют следующие классы и опции: В *RSM*: ***Старт*** – ***P*** с опцией *O* сохранения обратного ребра и опцией *П* создания шкалы прямых рёбер, ***ИнфоКорню*** – ***O***, ***Разметка*** – ***П***, ***Финиш*** – ***C***. В *RAM*: ***Старт*** – ***P*** без опций, ***ИнфоКорню*** – $\boldsymbol{M_к}$, ***Разметка*** – ***P*** с опцией *Б*, ***Финиш*** – $\boldsymbol{M_к}$. Опция *Б* состоит в том, что сообщение не посылается по обратному ребру.

Граф предполагается нумерованным, каждая вершина хранит свой номер размером $\boldsymbol{O}(\mathit{logn})$.

Сбор информации в корне выполняется аналогично тому, как это делается в алгоритмах типа **1**, за исключением следующего. Во-первых, для удобства последующей разметки при обработке сообщения ***Старт*** вершина сохраняет отображение номера ребра на номер соседа на другом конце этого ребра в переменной ***соответствие*** размером $\boldsymbol{O}(\Delta\mathit{logn})$. Из-за этого переменная ***множество соседей*** становится лишней, поскольку её значение восстанавливается по переменной ***соответствие***. Во-вторых, в $\mathcal{RSM}$ сообщение ***Старт*** имеет опцию *П* для формирования ***шкалы прямых рёбер***, которая затем используется для рассылки сообщения ***Разметка*** класса ***П***. В-третьих, поскольку в памяти вершины уже есть переменная ***соответствие*** размером $\boldsymbol{O}(\Delta\mathit{logn})$, не имеет смысла «экономная модификация» в $\mathcal{RSM}$ из 3.3.1. В-четвёртых, когда в $\mathcal{RSM}$ вершина получает сообщение ***Разметка*** и пересылает его дальше по прямым рёбрам, она на месте переменной ***шкала прямых рёбер*** создаёт переменную ***число прямых рёбер***, которая в дальнейшем используется при обработке сообщения ***Финиш*** класса ***С***.

После сбора информации решается задача, формируется и рассылается сообщение ***Разметка***. Сообщение ***Разметка*** содержит в качестве параметра *информацию о разметке* размером $R = \boldsymbol{O}(\mathit{nf})$. Информация о разметке вершин задаётся шкалой вершин, которые нужно пометить, размером $\boldsymbol{O}(n)$, а информация о разметке рёбер задаётся в виде списка множеств соседей размером $\boldsymbol{O}(\mathit{nf})$, в котором для каждой вершины i в её множестве соседей оставлены только те вершины j, для которых нужно пометить ребро ij. Получив первое сообщение ***Разметка***, вершина с номером i выполняет свою локальную разметку, определяемую i-ым разрядом шкалы вершин и i-ым

элементом списка множеств соседей. Для пометки ребра *ij* вершина должна узнать его номер в *i*, что делается по отображению номера ребра в номер соседа в переменной ***соответствие***.

В *RSM* сообщение ***Финиш*** не имеет параметров и относится к классу ***C***, поэтому по каждому прямому ребру в корень придёт ровно одно сообщение ***Финиш***. Когда корень получит все эти сообщения, глобальная разметка закончена. В *RAM* сообщение ***Финиш*** относится к классу M_κ и содержит номер инициатора, поэтому корень может использовать шкалу инициаторов: число «1» в ней равно числу вершин, от которых пришли сообщения ***Финиш***. Глобальная разметка закончена, когда это число становится равным числу некорневых вершин, т.е. на 1 меньше длины списка множеств соседей. После окончания глобальной разметки корень посылает вовне сообщение о завершении разметки и заканчивает работу.

Максимальный размер сообщения: для ***Старт*** $m = \mathbf{O}(\log n)$, для ***ИнфоКорню*** $m = \mathbf{O}(f)$, для ***Разметка*** $m = R = \mathbf{O}(nf)$, для ***Финиш*** в *RSM* $m = \mathbf{O}(1)$, а в *RAM* $m = \mathbf{O}(\log n)$.

3.3.3. Алгоритмы типа 2a: Уменьшение памяти вершины

Цель: модификация **2a** алгоритмов типа **2** для уменьшения памяти некорневой вершины за счёт удаления переменной ***соответствие*** размером $\mathbf{O}(\Delta \log n)$. Для этого достаточно сообщение ***Разметка*** посылать способом ***P*** без опции *Б* с дополнительным параметром *номер вершины*-отправителя размером $\mathbf{O}(\log n)$. Каждая вершина получит это сообщение по каждому инцидентному ей ребру. Получая сообщение ***Разметка*** от вершины *j* по ребру *k*, вершина *i*, во-первых, узнаёт соответствие *j* и *k*, а, во-вторых, по информации о разметке узнаёт, нужно ли помечать ребро *ij*. Тем самым, она может пометить ребро с номером *k* нужной меткой.

Сообщения имеют следующие классы и опции. В *RSM*: ***Старт*** – ***P*** с опцией *O* сохранения обратного ребра и опцией *C* подсчёта числа прямых рёбер, ***ИнфоКорню*** – ***O***, ***Разметка*** – ***P*** с опциями *O* и *C*, ***Финиш*** – ***C***. В *RAM*: ***Старт*** – ***P*** без опций, ***ИнфоКорню*** – $M_к$, ***Разметка*** – ***P*** без опций, ***Финиш*** – $M_к$.

Поскольку мы удаляем переменную ***соответствие***, снова нужна переменная ***множество соседей*** размером ***O***(*f*), кроме случая «экономной модификации» в *RSM* (3.3.1). Эта модификация снова имеет смысл: сообщение ***ИнфоКорень*** можно посылать «без накопления» информации о соседях, т.е. сразу при получении сообщения ***Старт***, используются переменные ***счётчик соседей*** и, в корне, ***шкала последних***, а также параметр *последнее сообщение* в сообщении ***ИнфоКорень***.

По сравнению с типом **2** время работы увеличивается не более чем на 1 такт (или остаётся тем же, если нет периферийных рёбер). Максимальные размеры сообщений *m* такие же как для типа **2**.

3.3.4. Алгоритмы типа 3: Сбор и разметка во всех вершинах

Цель: сбор информации в каждой вершине – *broadcast* с проходом по каждому ребру и *all-to-all-broadcast* за минимальное суммарное время d_0+1+d, и сообщение корню о завершении разметки – *convergecast* за минимальное время d_0. Используются сообщения ***Старт***, ***ИнфоВсем*** и ***Финиш***, которые имеют следующие классы и опции. В *RSM*: ***Старт*** – ***P*** с опцией *O* сохранения обратного ребра и *C* подсчёта числа прямых рёбер, ***ИнфоВсем*** – $M_в$, ***Финиш*** – ***C***. В *RAM*: ***Старт*** – ***P*** без опций, ***ИнфоВсем*** – $M_в$, ***Финиш*** – $M_к$.

Граф предполагается нумерованным, каждая вершина хранит свой номер размером ***O***(*logn*). Как и в алгоритмах типа **2** сообщение ***Старт*** содержит *номер вершины*, из которой оно посылается, а при обработке сообщения

Труды Института системного программирования РАН. Том ???, выпуск ???, 2017, стр. ???- ???.
ISSN 2220-6426 (Online), ISSN 2079-8156 (Print).
??? стр.

Старт вершина сохраняет отображение номера ребра на номер соседа на другом конце этого ребра в переменной ***соответствие*** размером $O(\Delta log n)$. Как и в алгоритмах типа **2** после того как вершина получит сообщение ***Старт*** от всех своих соседей, она формирует сообщение, но не ***ИнфоКорню***, а ***ИнфоВсем***, помещая в него *номер вершины* из переменной ***номер вершины*** и *множество соседей*, формируемое по переменной ***соответствие***. Кроме этого, вершина поддерживает ***список множеств соседей*** и определяет конец сбора информации о графе так же, как корень в алгоритмах типа **2** и **2a**. После того как информация о графе собрана, вершина решает задачу и выполняет свою локальную разметку. Напомним, что алгоритм решения задачи должен гарантировать, что в разных вершинах будет получаться одно и то же решение. После этого вершина посылает корню сообщение ***Финиш*** аналогично алгоритмам типа **2** и **2a**. Заметим, что в $\mathcal{RAM}$ это сообщение посылается множественной рассылкой ($M_к$) так же как сообщение ***ИнфоВсем*** ($M_в$), поэтому в каждой вершине нужно иметь две шкалы инициаторов – по одной для каждого из этих двух типов сообщений: ***Финиш*** и ***ИнфоВсем***.

Корень определяет конец работы, когда, во-первых, сам получит информацию о всём графе, и, во-вторых, получит все нужные сообщения ***Финиш***, что определяется так же как в алгоритмах типа **2** и **2a**.

Заметим, что «экономная модификация» в $\mathcal{RSM}$ для сбора информации не имеет смысла, поскольку она уменьшает память некорневой вершины с $O(f)$ до $O(log n)$, но в алгоритмах типа *3* все вершины имеют память размера $O(P+n+\Delta log n)$.

Максимальный размер сообщения: для ***Старт*** $m = O(log n)$, для ***ИнфоВсем*** $m = O(f)$, для ***Финиш*** в $\mathcal{RSM}$ $m = O(1)$, а в $\mathcal{RAM}$ $m = O(log n)$.

4. Экономия памяти для задач FSB, MST и MIS

В алгоритмах для общего случая размер памяти вершины имеет порядок $O(P) = O(nf) = O(n^2)$ в корне или во всех вершинах за счёт хранения списка множеств соседей. Такой же по порядку размер имеет информация о разметке. Для некоторых задач можно уменьшить эти размеры, используя вместо списка множеств соседей другую структуру меньшего размера.

4.1. Определение конца сбора информации о графе и конца работы

Список множества соседей используется также для определения конца сбора информации о графе. Можно предложить альтернативный способ: используются две битовые шкалы вершин, каждая размером $O(n)$: ***шкала всех вершин***, содержащая корень и вершины, от которых получены сообщения ***ИнфоКорню*** в алгоритмах типа **1** и **2** или вершины, от которых получены сообщения ***ИнфоВсем*** в алгоритмах типа **3**, и ***шкала всех соседей*** этих вершин. Конец сбора информации определяется по совпадению этих шкал. В алгоритмах типа **2** и **3** с этого момента времени корень знает число вершин (оно равно числу «1» в этих шкалах), что используется в *RAM* для определения конца приёма сообщений ***Финиш*** от всех некорневых вершин.

4.2. FSB

Мост – это ребро, не входящее в цикл, поэтому для определения мостов можно использовать следующий способ. Вместо списка множеств соседей хранится лес деревьев, вначале пустой. При получении вершиной *a* сообщения ***ИнфоКорню*** или ***ИнфоВсем***, в котором указана соседняя вершина *b*, проверяется ребро *ab*. Если при добавлении этого ребра к лесу цикл не образуется, ребро добавляется к лесу. В противном случае все рёбра этого

Труды Института системного программирования РАН. Том ???, выпуск ???, 2017, стр. ???- ???.
ISSN 2220-6426 (Online), ISSN 2079-8156 (Print).
??? стр.

цикла помечаются как «не мосты», а ребро *ab* к лесу не добавляется. В конце работы все рёбра леса, не помеченные как «не мосты», – это все мосты графа.

Для хранения остова достаточно памяти размером $\boldsymbol{O}(n\log n) = \boldsymbol{o}(nf)$ при $\Delta\to\infty$ (и, поскольку $\Delta \le n\text{-}1$, также $n\to\infty$). Информация о разметке содержит описание мостов как части рёбер остова, также размером $\boldsymbol{O}(n\log n) = \boldsymbol{o}(nf)$. Получаются варианты алгоритмов с оценками из Таб. 2 для $R = P = \boldsymbol{O}(n\log n)$.

4.3. MST

Для построения MST можно использовать известный критерий Тарьяна минимальности остовного дерева: остовное дерево минимально тогда и только тогда, когда любое ребро не из дерева является максимальным по весу на цикле, который образуется при его добавлении в дерево. Как и для FSB вместо списка множеств соседей хранится лес деревьев, вначале пустой. При получении от вершины *a* сообщения ***ИнфоКорню*** или ***ИнфоВсем***, в котором указана соседняя вершина *b*, проверяется ребро *ab*. Если при добавлении этого ребра к лесу цикл не образуется, ребро добавляется к лесу. В противном случае проверяется, какое ребро в этом цикле максимально по весу. Если это ребро $a\text{`}b\text{`} \ne ab$, оно удаляется из леса, а ребро *ab* добавляется к лесу.

Задача MST формулируется для взвешенного графа, поэтому список множеств соседей с указанием весов рёбер до них имеет размер $\boldsymbol{O}(nf_w)$, где $f_w = \min\{n\log w, \Delta\log nw\}$. Для хранения взвешенного остова достаточно памяти размером $\boldsymbol{O}(n\log nw) = \boldsymbol{o}(nf_w)$ при $\Delta\to\infty$ (и, поскольку $\Delta \le n\text{-}1 \le w$, также $n\to\infty$ и $w\to\infty$). Информация о разметке содержит описание рёбер остова размером $\boldsymbol{O}(n\log n) = \boldsymbol{o}(nf)$. Получаются варианты алгоритмов общего типа с оценками из Таб. 2 для $P = \boldsymbol{O}(n\log nw)$ и $R = \boldsymbol{O}(n\log n)$, и заменой *f* на f_w.

4.4. MIS

Для построения MIS можно использовать хорошо известную стандартную процедуру. Вместо списка множеств соседей в корне используется ***шкала вершин независимого множества***, представляющая независимое множество вершин *IS*. Вначале шкала содержит только корень. При получении корнем сообщения ***ИнфоКорню*** или ***ИнфоВсем*** от вершины *x* по множеству соседей в этом сообщении проверяется, смежна ли вершина *x* с какой либо вершиной из *IS* или нет. Если *x* не смежна ни с какой вершиной из *IS*, то *x* добавляется к множеству *IS*, т.е. в ***шкалу вершин независимого множества***. Однако применение такой процедуры имеет два ограничения.

Во-первых, в *RSM* в алгоритмах типа **1** и **2a** нельзя использовать «экономную модификацию», так как нельзя определить, смежна или не смежна вершина *x* с какой либо вершиной из *IS*, до получения всех сообщений ***ИнфоКорню*** от *x*.

Во-вторых, для алгоритмов типа **3** эта процедура не применима. Дело в том, что MIS в отличие от MST и FSB определяется в графе, вообще говоря, неоднозначно. Процедура основана на линейном упорядочении всех вершин в соответствии с порядком получения от них сообщений ***ИнфоКорню*** или ***ИнфоВсем***. Однако такой порядок, вообще говоря, разный для разных вершин, и поэтому они могут строить разные MIS'ы. Для того, чтобы вершины строили один и тот же MIS, в вершинах должен использоваться один и тот же линейный порядок вершин, порождающий, так называемый, лексикографический MIS (LFMIS). Однако для этого, скорее всего, потребуется больше памяти, скорее всего, порядка размера графа $O(nf)$.

Если вершина собрала описание всего графа, она может определить все возможные MIS`ы. Если каждый MIS упорядочить, т.е. преобразовать в последовательность, например, по возрастанию номеров вершин, то можно

Труды Института системного программирования РАН. Том ???, выпуск ???, 2017, стр. ???- ???.
ISSN 2220-6426 (Online), ISSN 2079-8156 (Print).
??? стр.

выбрать MIS являющийся наибольшим в лексикографическом порядке. Заметим, что такой MIS является наибольшим по числу вершин в нём. Тем самым, решается более сложная задача поиска MaxIS (maximim independent set). Эта задача может решаться в алгоритмах всех типов **1**, **2**, **2a**, **3**.

Для хранения *IS* в виде шкалы вершин достаточно памяти размером $\boldsymbol{O}(n) = \boldsymbol{o}(nf)$ при $n \to \infty$. Информация о разметке содержит MIS также в виде шкалы вершин размером $\boldsymbol{O}(n) = \boldsymbol{o}(nf)$. Получаются варианты алгоритмов с оценками из Таб. 2 для $R = P = \boldsymbol{O}(n)$.

5. Нижние оценки сложности для FSB, MST и MIS

Если решение задачи зависит от периферийных рёбер, то, очевидно, $T \le 2d_0+1$. Задачи FSB, MST и MIS зависят от периферийных рёбер, что показывается примерами, в которых два графа отличаются друг от друга только периферийными рёбрами, а задачи имеют на этих графах разные решения.

FSB: 1) дерево, в котором есть, по крайней мере, две периферийные вершины, и 2) такое же дерево с добавлением периферийного ребра. В случае 1 все рёбра дерева являются мостами, а во втором случае все рёбра цикла, образуемого этим периферийным ребром и деревом, мостами не являются.

MST: 1) дерево, в котором есть, по крайней мере, две периферийные вершины, и 2) такое же дерево с добавлением периферийного ребра, имеющего вес меньше, чем какое-нибудь ребро дерева из цикла, образуемого этим периферийным ребром и деревом. В случае 1 все рёбра дерева принадлежат MST, а во втором случае периферийное ребро принадлежит MST, а некоторое ребро дерева не принадлежит MST.

MIS: пример на Рис. 2. Одна из вершин *a* или *b* должна не принадлежать MIS, так как они соединены ребром. Пусть, для определённости, это будет вершина

a. Посмотрим, что можно сказать о вершинах, обозначенных знаком «?». Если они не соединены ребром, то обе должны принадлежать MIS, а иначе только одна из них. Но это ребро периферийное.

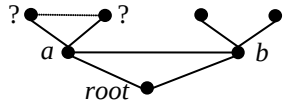

Рис. 2. Зависимость MIS от периферийных рёбер.

Fig. 2. Dependence of MIS on peripheral edges.

6. Алгоритмы типа 4 в RSM для задач с однозначным решением, зависящим только от простых циклов

Будем говорить, что задача на графе имеет однозначное решение, зависящее только от простых циклов графа, если локальная разметка в вершине зависит только от простых циклов, в которые входит эта вершина. К таким задачам относятся, например, FSB, MST и задача о гамильтоновом цикле.

Для FSB ребро, инцидентное вершине, является мостом тогда и только тогда, когда оно не входит ни в какой простой цикл, проходящий через вершину. Для MST ребро, инцидентное вершине, принадлежит MST тогда и только тогда, когда в каждом простом цикле, проходящем через вершину, это ребро не имеет наибольшего веса.

Гамильтонов цикл существует не во всяком графе, но в некоторых графах (например, в полном графе при $n > 3$) может быть несколько разных гамильтоновых циклов. В нумерованном графе всегда есть возможность однозначно выбрать один из гамильтоновых циклов. Для этого достаточно каждый гамильтонов цикл представить в виде последовательности номеров вершин, начиная с вершины с наименьшим номером 1 и выбирая направление обхода в сторону той из двух вершин, смежной по циклу с вершиной с

номером 1, которая имеет наименьший номер. После этого однозначно выбирается гамильтонов цикл как наименьший в лексикографическом порядке таких последовательностей.

Для подобного рода задач можно предложить в $\mathcal{RSM}$ алгоритмы типа **4** как модификацию алгоритмов типа **3** с временем $T = 2d_0+1$ или $T = 2d_0$, если нет периферийных рёбер. Идея алгоритмов основана на том факте, что расстояние между вершинами в одном цикле ограничено сверху не $d \le 2d_0$, а примерно в два раза меньшей величиной. Покажем формально, что каждая вершина узнает о всех простых циклах, в которые она входит, не позже, чем корень узнает весь граф. Как показано в п.3.3.1, корень узнаёт весь граф через время $2d_0+1$ или $2d_0$, если нет периферийных рёбер.

Действительно, пусть ребро ab и вершина c входят в простой цикл (Рис. 3). Нам достаточно показать, что время t, через которое вершина c узнает о ребре ab, не больше $2d_0+1$ или $2d_0$, если нет периферийных рёбер. Обозначим: d_{ac} – расстояние по циклу от a до c, d_{bc} – расстояние по циклу от b до c, d_a – расстояние от a до корня, d_b – расстояние от b до корня. Тогда $t \le min\{d_a+1+d_{bc},\ d_b+1+d_{ac}\}$. Но $d_a \le d_0$ и $d_b \le d_0$. Поэтому $min\{d_a+1+d_{bc},\ d_b+1+d_{ac}\} \le d_0 + 1 + min\{d_{bc}, d_{ac}\}$. Так как цикл простой, его длина $d_{ac} + d_{bc} + 1 \le d+1 \le 2d_0+1$, поэтому $min\{d_{bc}, d_{ac}\} \le d_0$. Тем самым, $t \le 2d_0+1$. Но если периферийных рёбер нет, то либо $d_a < d_0$, либо $d_b < d_0$. Пусть для определённости $d_a < d_0$. Если $d_{bc} \le d_0$, то $d_a+1+d_{bc} < 2d_0 + 1$. Если $d_{bc} > d_0$, то $d_{ac} < d_0$ (так как $d_{ac} + d_{bc} \le 2d_0$), поэтому $d_b+1+d_{ac} < 2d_0 + 1$. Тем самым, если периферийных рёбер нет, то $t \le 2d_0$.

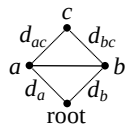

Рис. 3. Вершина c и ребро ab входят в один цикл.

Fig. 3. The vertex c and the edge ab enter into one cycle.

Алгоритмы типа **4** используют сообщения ***Старт*** класса ***P*** с опцией *O* и ***ИнфоВсем*** класса $M_в$, сообщение ***Финиш*** не используется. Поведение каждой вершины зависит от решаемой задачи.

Для FSB каждая вершина помечает все инцидентные ей рёбра как «мосты», а потом снимает эту пометку для ребра, который оказывается в некотором цикле. Для MST каждая вершина помечает все инцидентные ей рёбра как «принадлежащие MST», а потом снимает эту пометку для ребра, который оказывается ребром с наибольшим весом в некотором цикле. Для гамильтонова цикла каждая вершина на каждом раунде, узнав о части графа, выбирает в ней простой цикл, в который она входит, наибольшей длины и в соответствии с лексикографическим порядком соответствующей циклу последовательности номеров вершин. Если такой цикл есть, вершина помечает два инцидентных ей ребра, входящих в этот цикл, как принадлежащие гамильтонову циклу (если на предыдущем раунде были помечены какие-то другие два ребра, с них пометка снимается).

Когда корень узнаёт весь граф, ему не нужно дожидаться извещения от всех вершин о выполнении локальной разметки: он может сразу послать вовне завершающее сообщение о том, что задача решена. Для FSB или MST корень посылает вовне сообщение «мосты размечены» или «MST размечено», соответственно. При поиске гамильтонова цикла корень сначала проверяет существование гамильтонова цикла по собранной информации о графе. Если

его нет, корень посылает вовне сообщение «гамильтонова цикла нет», а если есть, корень посылает вовне сообщение «гамильтонов цикл есть и размечен».

Для алгоритмов типа **4** по сравнению с алгоритмами типа **3** в $\mathcal{RSM}$ меняется только время $T \le 2d_0+1$, а остальные оценки из Таб. 2 сохраняются.

Заметим, что в $\mathcal{RAM}$ аналогичный алгоритм не работает, так как из-за различных и переменных временах перемещения сообщений по рёбрам корень может узнать о всём графе до того, как все вершины узнали о простых циклах, в которые они входят, т.е. до того, как в них завершена локальная разметка.

7. Алгоритмы типа 5 – разметка во время сбора информации для FSB и MST

Для задач FSB и MST можно предложить алгоритмы типа **5** как модификацию алгоритмов типа **1** (без «экономной модификации»), которые размечают граф за одну пару *broadcast* + *convergecast*. Сообщение ***Старт*** имеет класс ***P*** с опцией *O* сохранения обратного ребра, а сообщение ***ИнфоКорню*** – в $\mathcal{RSM}$ класс ***C***, а в $\mathcal{RAM}$ – класс $\boldsymbol{M_K}$. Отличие от алгоритмов типа **1** в том, что там сообщение ***Старт*** в $\mathcal{RAM}$ имел класс ***P*** без опций. Как и для алгоритмов типа **1** сообщение ***Старт*** имеет параметр *номер вершины*, а сообщение ***ИнфоКорню*** – параметры *номер вершины* и *множество соседей*. Также эти сообщения имеют дополнительные параметры, описываемые ниже.

Конец работы корень определяет так, как описано выше в п.4.1, с помощью двух битовых шкал: ***шкалы всех вершин*** и ***шкала всех соседей***.

Идея алгоритма основана на использовании *векторов*, где *вектором маршрута* называется последовательность номеров рёбер маршрута, который проходится сообщением. Если ребро ab проходится в направлении $a \to b$, то в вектор помещается номер этого ребра в вершине a. Вектор пути от корня

имеет размер $\boldsymbol{O}(d_0 log\Delta)$ (в $\mathcal{RAM}$ $\boldsymbol{O}(D_0 log\Delta)$, но так как $D_0 \le 2d_0$, то $\boldsymbol{O}(D_0 log\Delta) = \boldsymbol{O}(d_0 log\Delta)$). Для взвешенного графа вектор, кроме номера ребра, содержит его вес, будем называть такой вектор с весами рёбер *взвешенным вектором*, он имеет размер $\boldsymbol{O}(d_0 log\Delta w)$. Сообщение ***Старт*** содержит дополнительный параметр: *вектор маршрута*, который сообщение проходит. *Вектор вершины* a – это вектор маршрута из первого полученного вершиной a сообщения ***Старт***, он хранится в переменной ***вектор вершины*** и далее обозначается как v_a, а его длина – как r_a. Заметим, что (взвешенный) вектор вершины – это последовательность номеров (и весов) прямых рёбер от корня до вершины.

Рассмотрим ситуацию, когда вершина a получает по ребру ab повторное сообщение ***Старт*** с вектором маршрута v. Если $v = v_a{\cdot}i{\cdot}j$, то ребро ab – прямое в a с номером i и обратное в b с номером j. В противном случае ребро ab – хорда. Каждая хорда ab образует цикл, состоящий из этой хорды и двух путей по остову от вершин a и b до *развилки* c, где развилка c – это такая вершина, что путь по остову от корня до c – это наибольший общий префикс путей по остову от корня до вершин a и b (Рис. 4). Вектор развилки v_c есть наибольший общий префикс векторов v_a и v_b.

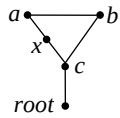

Рис. 4. Хорда ab и развилка c.

Fig. 4. Chord ab and fork c.

Поведение вершины a при обнаружении хорды ab с развилкой c зависит от решаемой задачи. Для FSB каждое ребро цикла нужно пометить как «не мост» в обоих его концах. При этом вершина a должна обеспечить пометку хорды ab в вершине a и каждого ребра на пути от a до c в обоих его концах. Для MST

Труды Института системного программирования РАН. Том ???, выпуск ???, 2017, стр. ???- ???.
ISSN 2220-6426 (Online), ISSN 2079-8156 (Print).
??? стр.

нужно пометить как «не принадлежащее MST» ребро цикла с наибольшим весом. Вершина *a* должна обеспечить пометку этого ребра при условии, что это хорда (помечается в вершине *a*) или ребро принадлежит пути от *a* до *c* (помечается в обоих его концах). Для обеих задач при обнаружении хорды *ab* вершиной *b*, вершина *b* обеспечивает выполнение аналогичной работы.

Вершина *a* помечает хорду *ab* (для MST если её нужно пометить) сразу, как только обнаруживает эту хорду. А *информацию о разметке остова*, т.е. о том, какие рёбра нужно пометить на пути от *a* до *c*, вершина *a* помещает в параметры создаваемого ею сообщения ***ИнфоКорню***. Однако это сообщение создаётся вершиной *a* после обнаружения не одной хорды *ab*, а всех хорд, инцидентных *a*, т.е. после получения вершиной *a* сообщения ***Старт*** по всем инцидентным вершине *a* рёбрам. Поэтому информация о разметке остова для всех хорд $ab_1,ab_2,...ab_k$ с развилками $c_1,c_2,...c_k$ объединяется. Заметим, что все эти развилки $c_1,c_2,...c_k$ располагаются на одном пути от корня до *a*. Пусть *c* – ближайшая к корню развилка среди развилок $c_1,c_2,...c_k$ т.е. имеющая вектор наименьшей длины $r = min\{r_{c_1},r_{c_2},...r_k\}$.

Для FSB результат объединения информации о разметке остова – это все рёбра на пути от *a* до *c*. Достаточно указать путь по остову от корня до *a*, т.е. вектор v_a, и число *r* как индекс по вектору v_a: пометить нужно все рёбра, соединяющие вершины с векторами $v_a[1..j]$ и $v_a[1..j+1]$, где $r_a > j \geq r$.

Для MST результатом объединения информации о разметке остова является множество рёбер на пути от *a* до *c*. Достаточно указать вектор v_a и множество индексов $\{r_1,r_2,...r_k\}$ по вектору v_a: пометить нужно ребро, соединяющие вершины с векторами $v_a[1..r_j]$ и $v_a[1..r_j+1]$, где $j = 1..k$. Множество $\{r_1,r_2,...r_k\}$ можно задавать битовой шкалой размером $\boldsymbol{O}(d_0)$.

Информация о разметке остова зависит также от модели.

В $\mathcal{RSM}$ сообщение ***ИнфоКорню*** имеет класс ***C*** и посылается по обратному ребру после того, как получены сообщения ***ИнфоКорню*** по всем прямым рёбрам. Поэтому в вершине происходит объединение информации о разметке остова из всех принимаемых сообщений ***ИнфоКорню***, результат такого объединения всё равно имеет вид (v_a,r) для FSB или $(v_a,\{r_1,r_2,...r_k\})$ для MST. Кроме того, сообщение ***ИнфоКорню*** двигается из вершины a к корню по остову, поэтому вектор v_a можно не указывать. Таким образом, в $\mathcal{RSM}$ у сообщения ***ИнфоКорню*** есть параметр *информация о разметке остова* вида r размером $\boldsymbol{O}(log d_0)$ для FSB или $\{r_1,r_2,...r_k\}$ размером $\boldsymbol{O}(d_0)$ для MST. Вершина x, получая по ребру i сообщение ***ИнфоКорню***, помечает как «не мост» обратное ребро, если для FSB $r < r_x$ или для MST $r_j+1 = r_x$ для некоторого j, и ребро i, если для FSB $r \le r_x$ или для MST $r_j = r_x$ для некоторого j.

В $\mathcal{RAM}$ сообщение ***ИнфоКорню*** распространяется множественной рассылкой, поэтому вектор v_a нужно указывать. Таким образом, в $\mathcal{RAM}$ у сообщения ***ИнфоКорню*** есть параметр *информация о разметке остова* вида (v_a,r) размером $\boldsymbol{O}(d_0 log\Delta+log d_0) = \boldsymbol{O}(d_0 log\Delta)$ для FSB или $(v_a,\{r_1,r_2,...r_k\})$ размером $\boldsymbol{O}(d_0 log\Delta+d_0) = \boldsymbol{O}(d_0 log\Delta)$ для MST. Вершина x, получая по ребру i сообщение ***ИнфоКорню***, делает такие же пометки как в $\mathcal{RSM}$, но только при условии, что она лежит на пути от корня до a, т.е. $v_x \le v_a$.

Для алгоритмов типа **5** оценки приведены в Таб. 3.

Таб. 3.Оценки алгоритмов типа 5.

Tabl. 3. Estimations of algorithms of type 5.

	FSB	***MST***	***FSB***	***MST***
MODEL	*RSM*		*RAM*	
Classes, P(*options*)	***P***(*O*)+***C***		***P***(*O*)+$\boldsymbol{M_\kappa}$	
A	$\boldsymbol{O}(d_0 log\Delta)$	$\boldsymbol{O}(d_0 log\Delta w)$	$\boldsymbol{O}(nlogd_0)$	
M_1 and S_1	$= \boldsymbol{O}(nlogn)$	$= \boldsymbol{O}(nlogn)$	$= \boldsymbol{O}(nlogn)$	
M_{all}	$\boldsymbol{O}(n\Delta d_0 log\Delta)$	$\boldsymbol{O}(n\Delta d_0 log\Delta w)$	$\boldsymbol{O}(n^2\Delta logd_0)$	
S_{all}	$= \boldsymbol{O}(n^3 logn)$	$= \boldsymbol{O}(n^3 logn)$	$= \boldsymbol{O}(n^3 logn)$	
T	$\leq 2d_0+1$			

8. Заключение

В статье рассматривается модель распределённой системы с выделенным корнем, с которого начинается и в котором заканчивается работа алгоритмов. Предлагается 8 типов алгоритмов (по 4 для синхронного и асинхронного вариантов системы) для решения любой задачи на нумерованном неориентированном графе, основанные на сборе информации о всём графе в корне или в каждой вершине. Для трёх задач (MIS, FBS и MST) приводятся нижние оценки сложности, а алгоритмы оптимизируются по памяти. Кроме того, для задач, имеющих однозначное решение, зависящее только от простых циклов (в частности, FBS, MST, поиск гамильтонова цикла), предлагаются алгоритмы с минимальным временем работы в синхронном варианте модели. Для двух задач (FBS и MST) предложены алгоритмы, которые оптимальны по времени как в синхронном, так и в асинхронном вариантах модели.

Для синхронной некорневой модели, в частности, модели LOCAL существуют эффективные алгоритмы решения различных задач на графах, в том числе рандомизированные. Поэтому одним из возможных направлений дальнейших исследований предполагается изучение возможности симуляции этих алгоритмов в корневой модели, как синхронной, так и асинхронной.

Список литературы

[1]. Joseph JaJa. An Introduction to Parallel Algorithms, Addison-Wesley, 1992, ISBN 0-201-54856-9.

[2]. Stephen A.Cook. An overview of computational complexity. Communications of the ACM. 1983, June, Vol.26, No.6.

[3]. Luis Barba. LITERATURE REVIEW: Parallel algorithms for the maximal independent set problem in graphs. October 2012. web-site: http://cglab.ca/~lfbarba/parallel_algorithms/Literature_Review.pdf. (проверено в декабре 2016.)

[4]. R.M. Karp, A. Wigderson. A fast parallel algorithm for the maximal independent set problem. Proc. 16th Annual ACM Symposium on Theory of Computing. ACM, New York, 1984, pp. 266–272.

[5]. M. Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. SIAM Journal on Computing. 1986, Vol.15, No 4., pp. 1036-1053. DOI:10.1137/0215074.

[6]. Noga Alon, Laszlo Babai, Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. Journal of Algorithms. 1986, December, Vol. 7, Issue 4, pp. 567-583.

[7]. David Peleg. Distributed computing — A Locality-sensitive approach. SIAM Monographs on Discrete Mathematics and Applications. 2000, 359 pp.

[8]. N.A. Lynch. Distributed Algorithms. The Morgan Kaufmann Series in Data Management Systems.1996, 904 pp.

[9]. Thomas Moscibroda. Locality, Scheduling, and Selfishness: Algorithmic Foundations of Highly Decentralized Networks. PhD thesis, ETH Zurich, 2006.

[10]. Y. Métivier, J. M. Robson, N. Saheb-Djahromi, A. Zemmar. An optimal bit complexity randomized distributed MIS algorithm. Distributed Computing. April 2011, Vol. 23, Issue 5, pp. 331–340.

[11]. Mohsen Ghaffari. An Improved Distributed Algorithm for Maximal Independent Set. Proc. of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). 2016, pp.270-277.

[12]. Mohsen Ghaffari. An Improved Distributed Algorithm for Maximal Independent Set. Cornell University Library: https://arxiv.org/pdf/1506.05093v2.pdf (проверено в декабре 2016).

[13]. Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In Foundations of Computer Science (FOCS) 2012, IEEE, 2012, pp. 321–330. Also coRR abs/1202.1983v3.

[14]. Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What cannot be computed locally! In the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC). ACM, 2004, pp 300–309.

[15]. И. Бурдонов, А. Косачев. Общий подход к решению задач на графах коллективом автоматов. Труды Института системного программирования РАН, том 29, вып. 2, 2017, стр. 27-76.

---

[16]. И. Бурдонов, А. Косачев. Размер памяти для хранения упорядоченного корневого графа. Труды Института системного программирования РАН, том 29, вып. 2, 2017, стр. 7-26.

И. Бурдонов, А. Косачев.
Распределённые алгоритмы на корневых неориентированных графах.
Труды Института системного программирования РАН. Том ???, выпуск ???, 2017, стр. ???- ???.
ISSN 2220-6426 (Online), ISSN 2079-8156 (Print).
??? стр.

Distributed algorithms on rooted undirected graphs

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

Abstract. Distributed algorithms of solving problems on undirected graphs are considered. In section 2, a model is defined featuring a root as a starting and ending point of the algorithm execution. Synchronous and asynchronous versions of the model are described. In section 3, algorithms of solving any problems are suggested based on collecting information on the whole graph in the root or in any vertex, as well as, on the graph labeling (its vertices and/or edges), if required. Emphasis is made on the time of the algorithm execution or on saving memory in vertices and total size of transferred messages, if this time is minimal. The rest of the paper considers optimizations for particular problems: creation of Maximal Independent Set (MIS), Finding Set of Bridges (FSB), creation of Minimum Spanning Tree (MST) in a edge-weighted graph. In section 4, a modification of general algorithms for these problems is suggested decreasing the estimate of memory size of vertices and messages. Section 5 includes lower-bound estimates of solution complexity for these problems. In section 6, for synchronous model, the time of algorithms execution with graph labeling is decreased to the lower bound for problems with single-valued solution depending on only simple cycles of the graph, in particular, FSB, MST and the problem of Hamiltonian cycle search. In section 7, time-optimal algorithms for FSB and MST are considered for both synchronous and asynchronous models. Conclusion summarizes the results and outlines the directions for further research.

Keywords: rooted undirected graph; distributed algorithms; graph problems; maximal independent set; minimum spanning tree; finding bridges.

References

[1]. Joseph JaJa. An Introduction to Parallel Algorithms, Addison-Wesley, 1992, ISBN 0-201-54856-9.
[2]. Stephen A.Cook. An overview of computational complexity. Communications of the ACM. 1983, June, Vol.26, No.6.
[3]. Luis Barba. LITERATURE REVIEW: Parallel algorithms for the maximal independent set problem in graphs. October 2012. web-site:

http://cglab.ca/~lfbarba/parallel_algorithms/Literature_Review.pdf. (проверено в декабре 2016.)

[4]. R.M. Karp, A. Wigderson. A fast parallel algorithm for the maximal independent set problem. Proc. 16th Annual ACM Symposium on Theory of Computing. ACM, New York, 1984, pp. 266–272.

[5]. M. Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. SIAM Journal on Computing. 1986, Vol.15, No 4., pp. 1036-1053. DOI:10.1137/0215074.

[6]. Noga Alon, Laszlo Babai, Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. Journal of Algorithms. 1986, December, Vol. 7, Issue 4, pp. 567-583.

[7]. David Peleg. Distributed computing — A Locality-sensitive approach. SIAM Monographs on Discrete Mathematics and Applications. 2000, 359 pp.

[8]. N.A. Lynch. Distributed Algorithms. The Morgan Kaufmann Series in Data Management Systems.1996, 904 pp.

[9]. Thomas Moscibroda. Locality, Scheduling, and Selfishness: Algorithmic Foundations of Highly Decentralized Networks. PhD thesis, ETH Zurich, 2006.

[10]. Y. Métivier, J. M. Robson, N. Saheb-Djahromi, A. Zemmar. An optimal bit complexity randomized distributed MIS algorithm. Distributed Computing. April 2011, Vol. 23, Issue 5, pp. 331–340.

[11]. Mohsen Ghaffari. An Improved Distributed Algorithm for Maximal Independent Set. Proc. of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). 2016, pp.270-277.

[12]. Mohsen Ghaffari. An Improved Distributed Algorithm for Maximal Independent Set. Cornell University Library: https://arxiv.org/pdf/1506.05093v2.pdf (проверено в декабре 2016).

[13]. Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In Foundations of Computer Science (FOCS) 2012, IEEE, 2012, pp. 321–330. Also coRR abs/1202.1983v3.

[14]. Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What cannot be computed locally! In the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC). ACM, 2004, pp 300–309.

[15]. I. Burdonov, A. Kossatchev. 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 (in Russian).

[16]. 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. 7-26 (in Russian).

И. Бурдонов, А. Косачев.
Распределённые алгоритмы на корневых неориентированных графах.
Труды Института системного программирования РАН. Том ???, выпуск ???, 2017, стр. ???- ???.
ISSN 2220-6426 (Online), ISSN 2079-8156 (Print).
??? стр.

Распределённые алгоритмы на корневых неориентированных графах