Оптимизация, системный анализ и исследование операций

© 2025 г. О.П. КУЗНЕЦОВ, д-р техн. наук (olpkuz@yandex.ru)
(Институт проблем управления им. В.А. Трапезникова РАН, Москва)

РАСПРЕДЕЛЕННЫЙ АЛГОРИТМ НУМЕРАЦИИ ВЕРШИН ГРАФА, СОВМЕЩЕННЫЙ С ПОСТРОЕНИЕМ ДЕРЕВА ОБХОДА В ШИРИНУ

Предлагается новый распределенный алгоритм нумерации вершин корневого неориентированного графа, который в процессе нумерации строит остовное дерево, являющееся при этом деревом обхода в ширину. Приведены оценки сложности алгоритма.

Ключевые слова: неориентированный граф, распределенный алгоритм, нумерация вершин, остовное дерево, обход в ширину.

DOI: 10.31857/S0005231025110064

1. Введение

Задачи о распределенных вычислениях на графах имеют давнюю историю. Первой такой задачей является задача о синхронизации цепи стрелков, предложенная Дж. Майхиллом в 1957 г. и решенная В. Левенштейном [1] и Ф. Муром [2]. В ней рассматривается цепь одинаковых автоматов, которые после активизации одного из них должны за минимальное время перейти в одно и то же состояние. Начиная с 1980-х гг. стало появляться большое количество распределенных алгоритмов решения различных графовых задач, среди которых отметим задачи нахождения минимального остовного дерева (задача MST) [3–7] и дерева поиска в ширину (задача BFS) [8–10]. Обзоры методов решения этих и многих других задач содержатся в [11–13].

Общая схема распределенных алгоритмов заключается в следующем. В вершинах графа располагаются процессоры, которые обмениваются сообщениями в синхронном или асинхронном режиме. Возможны два варианта начала процесса вычисления: 1) процессоры во всех вершинах начинают работать одновременно; 2) работа алгоритма начинается с активизации одной вершины, называемой корнем графа; остальные вершины активизируются после получения сообщений. Практически все эти алгоритмы предполагают, что вершины либо пронумерованы (см. обзоры [11–13]), либо имеют какие-то уникальные идентификаторы [10]. В [11] описан распределенный алгоритм нумерации вершин, основанный на известном алгоритме Тэрри обхода ребер графа, изложенном в [14].

В настоящей статье предлагается распределенный алгоритм нумерации вершин неориентированного графа, который в процессе нумерации строит остовное дерево, являющееся при этом деревом обхода в ширину (BFS-деревом) и, соответственно, деревом кратчайших путей из корня.

2. Описание алгоритма

Дан связный неориентированный граф с n вершинами и m ребрами, в котором выделена начальная вершина (корень), которую обозначим через v_0. Такие графы будем называть корневыми; в дальнейшем по умолчанию будут рассматриваться именно такие графы. В каждой вершине находится процессор, который может выполнять один из локальных алгоритмов, в зависимости от состояния вершины. Все вершины могут находиться в одном из четырех состояний, которые будем обозначать цветами: белым, серым, черным и красным. Каждому состоянию (цвету) соответствует свой локальный алгоритм. В дальнейшем будем отождествлять процессор вершины с самой вершиной, т.е. говорить о действиях вершины, подразумевая действия ее процессора.

Каждая вершина имеет упорядоченный список инцидентных ей ребер. Ребра имеют признак, который принимает одно из четырех значений: входящее (верхнее), черное (нижнее), пунктирное (хорда) и красное (об этом ниже).

Вершины могут обмениваться сообщениями двух типов.

Тип 1 имеет вид (i, c), где i – номер, т.е. $i \in \{1, \ldots, n-1\}$, c – цвет, т.е. $c \in \{\text{черный, красный}\}$. Для краткости сообщения этого типа будем называть «черный (красный) i» или «черный (красный) номер». По умолчанию считаем, что номер черный, а сообщение имеет тип 1: выражение «отправить i» будет означать «отправить сообщение (i черный)».

Тип 2 имеет вид (i, j), где i, j – черные номера; сообщение этого типа возникает, когда при попытке присвоить вершине номер $i+1$ она сообщает, что уже имеет номер j.

В обоих типах сообщений i – это последний присвоенный номер. Его будем называть текущим i или текущим номером.

Вершина белая, если она не пронумерована; ожидает сообщения, чтобы получить свой номер; после этого она становится либо серой (если у нее есть, кроме входящего ребра, другие инцидентные ей ребра), либо красной (если таких ребер у нее нет, т.е. она висячая);

– серая, если она пронумерована, но у нее есть непронумерованные соседи; ожидает сообщения типа 1, чтобы начать нумерацию соседних вершин, или сообщения типа 2, чтобы пометить ребро, по которому это сообщение пришло, как хорду;

– черная, если пронумерована и она, и все ее соседи; при получении текущего номера она передает его на следующий уровень;

– красная, если достижимые из нее вершины следующих уровней либо отсутствуют, либо пронумерованы. В обоих случаях об этом сообщается наверх: по входящему ребру отправляется красное i. Подробнее условия перехода в красный цвет будут описаны ниже.

Идея раскраски вершин в белый, серый и черный цвета заимствована из [15], но модифицирована в связи с распределенностью вычислений: каждому цвету соответствует свой алгоритм; кроме того, добавлен красный цвет, который используется, чтобы сообщить об окончании нумерации на некоторой ветке.

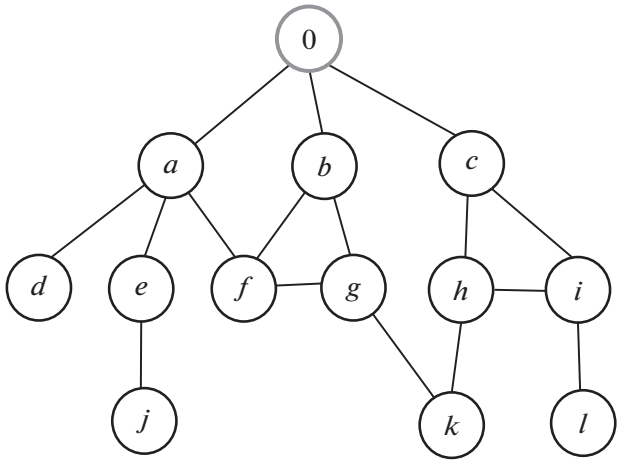

Рис. 1.

Предлагаемый алгоритм нумерации вершин основан на обходе графа в ширину [15] и является последовательным: в каждый момент активна только одна вершина. Передача сообщения фактически является передачей управления: получение сообщения активизирует вершину (запускает ее алгоритм, зависящий от текущего цвета вершины), отправление сообщения прекращает активность. В силу последовательного характера алгоритма в каждый период активности вершина может передать только одно сообщение по одному адресу.

Как будет показано ниже, предлагаемый алгоритм строит остовное дерево, состоящее из входящих ребер; каждая вершина этого дерева находится на минимальном расстоянии от корня. Это позволяет использовать при описании алгоритма два понятия: уровень и ветка. Ветка вершины – это поддерево, корнем которого она является; i-й уровень – это множество вершин, находящихся на расстоянии i от корня исходного графа. Обход в ширину означает, что пока не пронумерованы вершины очередного уровня, вершины следующих уровней не нумеруются; тем самым, если $i < j$, то номер любой вершины i-го уровня будет меньше номера любой вершины j-го уровня.

Напомним, что хорда – это ребро, не содержащееся в остовном дереве.

В качестве примера будем использовать граф на рис. 1 на различных этапах работы алгоритма. Буквы, которыми помечены вершины, введены для удобства описания и чтения. Буквы вершин неизвестны ни ее соседям, ни корню и потому не являются их идентификаторами в обычном смысле; в работе алгоритма они не используются.

Цвета вершин в черно-белой графике будем изображать следующим образом. Белые (ненумерованные) вершины – это вершины, помеченные буквами. Серые вершины имеют обычный (тонкий) контур. Черные вершины заштрихованы. Красные вершины и ребра имеют более толстый контур. Корень изображается наверху, поэтому выражение «отправить наверх» означает «отправить в сторону корня».

Схема общего алгоритма нумерации вершин:

Начальное состояние:

Все вершины, кроме корня, белые; у корня цвета нет; все ребра черные. Множество черных ребер (МЧР) упорядочено, поэтому имеет смысл говорить о первом ребре МЧР.

Начало:

корень присваивает себе номер 0;

формирует очередь из МЧР и посылает 0 по первому ребру очереди.

В дальнейшем он выполняет локальный алгоритм корня, который заключается в управлении нумерацией уровней. Общий процесс происходит следующим образом.

В первом цикле корень, отправляя 0 по первому ребру очереди, инициирует нумерацию вершин 1-го уровня. В конце этого цикла очередь оказывается пустой; это означает, что первый уровень пронумерован и его вершины стали серыми (висячие вершины первого уровня становятся красными). k-й цикл завершается тем, что вершины k-го уровня стали серыми или красными, все вершины предыдущих уровней – черные или красные, и сообщение с текущим номером приходит в вершину 0 по последнему ребру очереди.

В $(k+1)$-м цикле корень, получив этот текущий номер, заново формирует очередь из МЧР и отправляет сообщение с черным номером по первому ребру очереди. Это сообщение, пройдя $k-1$ черных вершин, приходит в серую вершину v k-го уровня, которая, получив это сообщение, нумерует своих соседей, связанных с v черными ребрами (входящими в МЧР); по окончании нумерации отправляет последний номер наверх по входящему ребру (u, v) и становится черной. Если соседняя вершина w уже пронумерована, то ребро (v, w) помечается как хорда. Если после этого МЧР вершины v оказывается пустым, она становится красной и отправляет красный номер наверх в свою черную вершину u, которая делает ребро (u, v) красным. После того, как все нижние ребра черной вершины стали красными, она сама становится красной и отправляет красный номер наверх. Алгоритм заканчивается тогда, когда все ребра, инцидентные корню, стали красными.

Выражение «отправить сообщение» по умолчанию означает конец алгоритма и ожидание очередного сообщения.

Теперь опишем конкретные локальные алгоритмы: алгоритм корня и алгоритмы, соответствующие различным состояниям (цветам) вершин.

Локальный алгоритм корня

1а) Присвоить себе номер 0.

1б) запустить алгоритм нумерации соседей с текущим i;

1в) сформировать очередь из МЧР;

1г) послать i по первому ребру очереди; ожидать сообщения.

2. Если по текущему ребру очереди пришло черное i (очередной уровень этой ветки пронумерован), то

2а) удалить это ребро из очереди;

2б) если оставшаяся очередь непуста, послать i по первому ребру очереди; иначе (очередной уровень графа пронумерован)

2в) сформировать очередь из черных ребер;

2г) послать i по первому ребру очереди.

3. Иначе (по текущему ребру очереди пришло красное число i; это означает, что на этой ветке ненумерованных вершин нет)

3а) сделать это ребро красным и удалить из очереди;

3б) если очередь непуста, послать i по первому ребру;

3в) если очередь пуста и МЧР непусто, то перейти к 2в;

3г) если МЧР пусто (все ребра красные), **конец общего алгоритма.**

На первом уровне хорд нет, поэтому корень получает только сообщения типа 1.

Локальный алгоритм белой вершины

Белая вершина не имеет номера. Все ее ребра черные.

При получении сообщения типа 1 с числом i:

1) присвоить себе номер $i + 1$;

2) пометить ребро, по которому пришло i, как входящее;

3) если МЧР непусто, то

3а) отправить черное $i + 1$ наверх по входящему ребру,

3б) стать серой; конец алгоритма;

1. иначе

4а) отправить красное $i + 1$ наверх по входящему ребру;

4б) стать красной; конец алгоритма.

Белая вершина, получив свой номер и став серой, «не знает», завершена ли нумерация ее уровня, и потому не нумерует своих соседей, которые находятся на следующем уровне, а лишь сообщает по входящему ребру свой номер «своей» верхней вершине, чтобы та продолжила нумерацию соседей. Но белая вершина может оказаться висячей, т.е. иметь единственного соседа, соединенного с ней входящим ребром. В этом случае (п. 4 алгоритма) она, минуя серый и черный этапы, сразу становится красной.

Локальный алгоритм серой вершины

1) если сообщение типа 1 пришло по входящему ребру, то

1а) сформировать из МЧР очередь;

1б) послать i по первому ребру очереди;

1в) если по этому ребру вернулось черное $i + 1$, то

1в1) удалить это ребро из очереди;

1в2) если очередь непуста, перейти к 1б) с $i + 1$;

иначе

1в3) отправить $i + 1$ по входящему ребру;

1в4) стать черной; конец локального алгоритма;

1г) если по этому ребру вернулось красное $i+1$, то

1г1) сделать это ребро красным;

1г2) удалить его из очереди;

1г3) если очередь непуста, перейти к 1б) с $i+1$;

иначе

1г4) если МЧР непусто, перейти к 1в3);

иначе (все ребра красные)

1г5) отправить наверх красное i;

1г6) стать красной;

2) иначе (сообщение (x, y) типа 1 пришло по черному ребру)

2а) сделать это ребро пунктирным (это ребро – хорда);

2б) отправить по этому ребру сообщение типа 2 (i, j), где j – номер этой вершины;

2в) если МЧР непусто, то конец алгоритма;

2г) иначе

2г1) отправить наверх красное i;

2г2) стать красной; конец алгоритма.

Пункт 2 соответствует ситуации, когда вершина уже пронумерована и является концом хорды. В результате шага 2б МЧР может стать пустым, и тогда (2г) вершина становится красной. Случай 2г нарушает последовательный характер алгоритма (см. ниже пример 4). Приход номера по черному ребру означает, что идет нумерация на другой ветке q, а обнуление МЧР вызывает параллельную передачу красного номера вверх по данной ветке p, которая может остановиться в любой вершине ветки. Когда красный номер достигнет корня (это произойдет, когда все вершины ветки p станут красными), то корень получит два текущих номера: от ветки p, пронумерованной раньше, и от ветки q, на которой прошла последняя нумерация. При этом: а) входящее ребро ветки p станет красным и будет удалено из МЧР корня; б) текущим останется номер, полученный от ветки q, поскольку он не может быть меньше номера, полученного от ветки p.

Пример 1 (рис. 2). Вершина 1 завершила нумерацию своих нижних соседей (при этом вершины 5 и 6 стали серыми, вершина 4 (висячая) стала красной); отправила последний номер 6 наверх и стала черной. Корень, получив номер 6, удаляет ребро (0, 1) из очереди и отправляет номер 6 серой вершине 2.

Пример 2 (рис. 3). Вершина 2 (серая) запускает нумерацию соседей и пытается пронумеровать серую вершину 6, уже пронумерованную вершиной 1. У вершины 6 срабатывает п. 2 серого алгоритма: она помечает ребро (2, 6) как пунктирное (хорда) и отправляет по этому ребру вершине 2 сообщение (6, 6) типа 2. Вершина 2 также помечает у себя ребро (2, 6) как

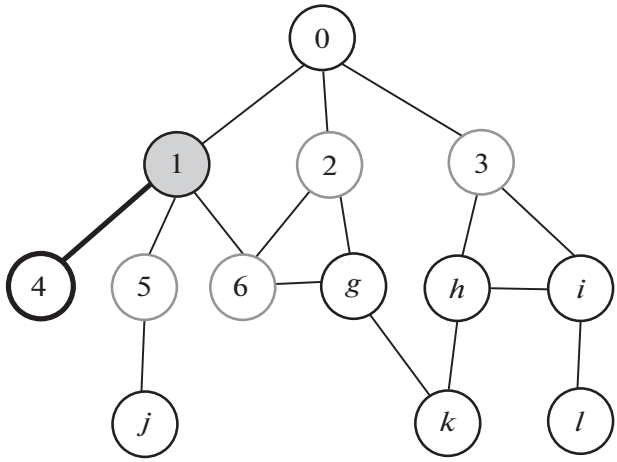

Рис. 2.

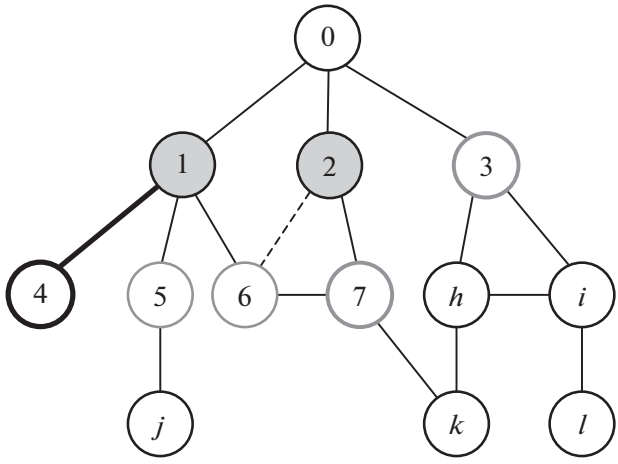

Рис. 3.

пунктирное и отправляет 6 в следующую вершину, которая получает номер 7 и отправляет его по входящему ребру вершине 2. Нумерация соседей вершины 2 на этом закончена (очередь пуста); она сообщает об этом наверх и становится черной.

Локальный алгоритм черной вершины

1) если по входящему ребру пришло черное число i, то
   1а) сформировать очередь из МЧР;
   1б) послать i по первому ребру очереди.
2) если снизу пришло черное число i, то
   2а) удалить это ребро из очереди;
   2б) если очередь непуста, перейти к 1б) с i;
   иначе отправить i по входящему ребру; конец локального алгоритма;

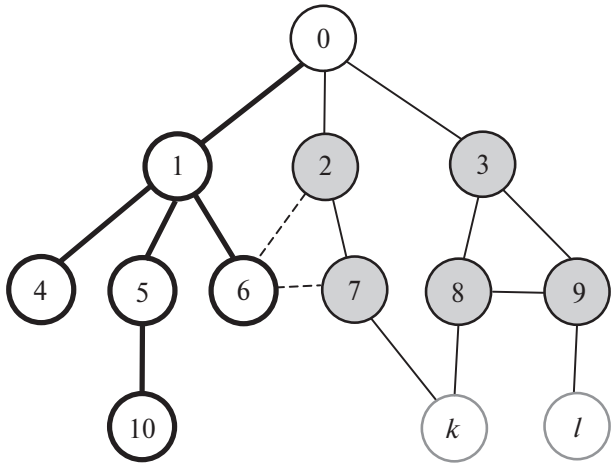

Рис. 4.

3) если снизу пришло красное число i, то

3а) сделать красным это ребро, удалить из очереди;

3б) если очередь непуста, послать черное i по первому ребру;

3в) если очередь пуста, то

если все ребра красные, стать красной и послать наверх красное i; конец локального алгоритма;

иначе послать наверх черное i; конец локального алгоритма.

Пример 3. На рис. 4 представлен этап работы общего алгоритма, на котором завершена нумерация вершин 3-го уровня ветки 2. Рассмотрим, как происходила эта нумерация, приведшая граф к состоянию, изображенному на этом рисунке. Корень послал текущий номер 9 в вершину 1. Вершина 1 и ребро (1, 4) стали красными еще раньше, при нумерации 2-го уровня (см. пример 1), и потому ребро (1, 4) уже не входит в МЧР вершины 1. Вершина 5 пронумеровала вершину 10, которая обнаружила, что она – красная, отправила красный номер 10 вершине 5, которая тоже стала красной и отправила красный номер 10 вершине 1, которая делает ребро (1, 5) красным. Далее вершина 1 отправляет номер 10 в вершину 6, которая пытается пронумеровать вершину 7, получает от нее сообщение типа 1, помечает ребро (6, 7) как хорду, становится красной (п. 2г серого алгоритма) и отправляет красный номер 10 в вершину 1. Вершина 1 делает ребро (1, 6) красным, обнаруживает, что черных ребер у нее нет, становится красной и посылает красный номер 10 в корень. Ребро (0,1) становится красным, т.е. удаляется из МЧР корня, и потому ветка 1 в дальнейших нумерациях не участвует.

Отметим следующее. Уровни концов хорды могут либо совпадать (такие хорды будем называть горизонтальными; пример – хорда (6, 7)), либо отличаться не более чем на 1 (вертикальные хорды; пример – хорда (2, 6)). Действительно, пусть на ребре (x, y) вершина x пронумерована и находится на k-м уровне, а y – еще нет. Очевидно, что (x, y) в этот момент входит в

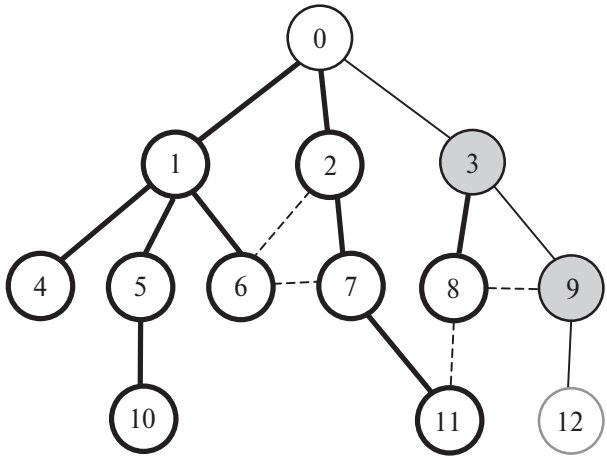

Рис. 5.

МЧР x. Поэтому при нумерации $(k+1)$-го уровня вершина x либо пронумерует y, либо обнаружит, что y уже пронумерована; в последнем случае (x, y) будет помечена как хорда. В любом случае, как видно на примере ребра (6, 7), горизонтальная хорда k-го уровня будет обнаружена как хорда только при нумерации $(k+1)$-го уровня. Как увидим ниже, это может повлиять на общее время нумерации.

Пример 4 (продолжение примера 3). Получив номер 10 от вершины 1, корень начинает нумерацию ветки 2, которая заканчивается приходом в корень черного номера 11 и не дает новых красных вершин, потому что в алгоритме белой вершины 11 срабатывает п. 3 (МЧР еще содержит ребро (8, 11)), и она становится серой, а не красной. После этого начнется нумерация 3-го уровня ветки 3, в ходе которой вершина 8 обнаруживает хорды (8, 9) и (8, 11). При попытке вершины 8 пронумеровать уже пронумерованную (т.е. серую) вершину 11 в этой вершине срабатывает п. 2 серого алгоритма, она становится красной и отправляет наверх свой красный номер. Таким образом, какое-то время будут происходить **два параллельных процесса**: по ветке 2 передается в корень красный номер 11, а по ветке 3 продолжается нумерация. В общем случае возможны два варианта: а) процессы идут по разным веткам; б) процессы идут по разным подветкам одной ветки. В первом случае они сойдутся в корне; во втором случае они придут в некоторую вершину одной ветки, которая либо поменяет свой цвет (если ее МЧР станет пустым), либо нет.

На рис. 5 представлен момент окончания нумерации. Вершина 12 только что получила свой номер. В следующий момент она станет красной; и через вершины 9 и 3 красный номер 12 придет в корень, на чем процесс нумерации завершится.

Пример 5. Предположим теперь, что в граф на рис. 5 добавлена горизонтальная хорда (11, 12). Как отмечено в конце примера 3, горизонтальная хорда k-го уровня будет обнаружена как хорда только при нумерации $k+1$-го уровня. Поэтому процесс пойдет не так, как описано в примере 4: вер-

шины 11 и 12 не станут красными, после завершения нумерации 3-го уровня в корень придет черный номер 12 и корень запустит нумерацию 4-го уровня, не зная, что его нет. И только на 4-м цикле вершины 11 и 12 станут красными, о чем пойдут параллельно сообщения по веткам (11, 7, 2, 0) и (12, 9, 3, 0).

3. Основные результаты. Оценка сложности алгоритма

Теорема 1. Входящие ребра исходного графа G, помеченные в результате работы алгоритма, образуют дерево, которое является а) остовным, б) деревом кратчайших путей от корня, т.е. деревом обхода в ширину (BFS-деревом).

Рассмотрим граф G^*, образованный входящими ребрами. У каждой вершины только одно входящее ребро. Это следует из того, что ребро помечается как входящее, только когда вершина белая, которая после этого становится серой. Кроме того, если ребро (u, v) – входящее для v, то номер u меньше номера v, потому что v получила свой номер по ребру (u, v), и вершина u при этом уже была пронумерована. Поэтому на любом пути из v по входящим ребрам номера вершин будут убывать. Отсюда следует, что 1) путь из входящих ребер из любой вершины v не может быть циклом; 2) он может закончиться только в корне, потому что только в корне нет входящего ребра. Таким образом, граф G^* является связным (любые две вершины либо находятся на одном пути к корню, либо связаны двумя путями, ведущими к корню), не содержит циклов и, следовательно, является деревом. Поскольку каждая вершина имеет входящее ребро, дерево G^* содержит все вершины исходного графа, т.е. является остовным.

Докажем теперь, что для любой вершины v ее путь к корню v_0 в графе G^* – кратчайший в графе G. Доказательство проведем индукцией по циклам нумерации.

1-й цикл нумерации – это нумерация корнем своих соседей. Очевидно, что после его окончания все соседи корня стали серыми, они пронумерованы, ребра, соединяющие их с корнем, – входящие, и их расстояние до корня равно 1. Будем считать, что они образуют уровень 1. Ясно, что других вершин с единичным расстоянием до корня нет.

Пусть теперь завершился k-й цикл нумерации, в результате чего появились новые серые вершины, расстояние которых до корня равно k, и других вершин с расстоянием k до корня нет. Эти вершины образуют уровень k. Тогда все вершины, имеющие расстояние $k + 1$ до корня, являются соседями вершин уровня k и, следовательно, в $k + 1$-м цикле будут ими пронумерованы, станут серыми и образуют $k + 1$-й уровень. При этом ребра, по которым им был доставлен номер, станут для них входящими и войдут в дерево G^*. Таким образом, для любого k вершина с расстоянием k до корня войдет в k-й уровень дерева G^*, что и доказывает, что граф G^*, образованный входящими ребрами исходного графа G, является деревом кратчайших путей до корня, т.е. деревом обхода в ширину (BFS-деревом).

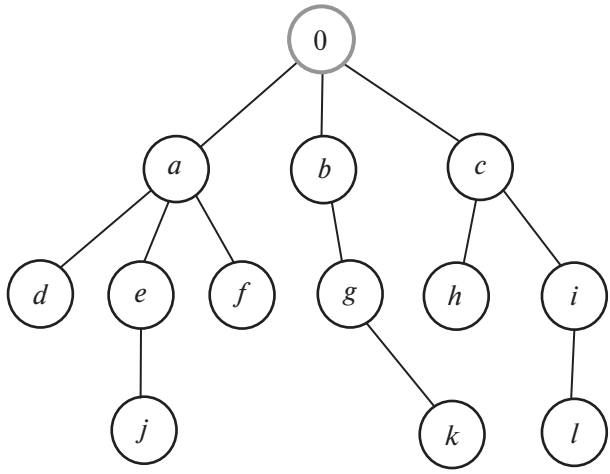

Рис. 6.

Итак, в результате работы алгоритма:

1. Пронумерованы все вершины графа и найдено общее число вершин.
2. Построено остовное дерево графа, состоящее из входящих ребер, найдены хорды графа и тем самым определено цикломатическое число графа.
3. Построенное остовное дерево является деревом обхода в ширину.

Таким образом, предложенный алгоритм решает одновременно две задачи: распределенную нумерацию вершин и распределенное построение дерева обхода в ширину. Поэтому этот алгоритм будем называть NBFS-алгоритмом.

Отметим, что локальный параллелизм, иногда возникающий при работе NBFS-алгоритма (см. пример 4), не требует синхронизации: конечное состояние вершины, в которую придут два параллельных сигнала, не зависит от того, в каком порядке они придут.

Построенное остовное дерево не является единственным и зависит от порядка ребер в очередях, формируемых локальными алгоритмами.

Дерево обхода в ширину для нашего примера приведено на рис. 6. Если на втором цикле очередь в корне имела бы вид bac, то вершина f получила бы номер 4, а ребро (b, f) не было бы хордой.

Сложность. Под сложностью NBFS-алгоритма в дальнейшем будем понимать его коммуникационную сложность, т.е. число сообщений, порождаемых NBFS-алгоритмом в ходе его работы. Введем следующие обозначения:

$C_{NBFS}(n)$ – сложность NBFS-алгоритма для корневых графов с n вершинами;

$C_{NBFS}(G)$ – сложность NBFS-алгоритма для конкретного графа G;

$e(G)$ – эксцентриситет корня графа G, т.е. максимальное из расстояний от v_0 до остальных вершин;

$\gamma(G)$ – цикломатическое число (и соответственно число хорд) графа G,

G_{BFS} – остовное дерево графа G. В дальнейшем будем писать просто e и γ.

Любой алгоритм нумерации вершин корневого графа должен совершить последовательный обход всех вершин начиная с корня. Распределенный алгоритм при этом должен обойти все ребра, поскольку при отсутствии исходной информации о глобальных свойствах графа нет гарантии, что посещены все вершины, пока не посещены все ребра. Известный алгоритм Тэрри [14] обхода всех ребер основан на том, что если расщепить каждое ребро на два «полуребра», то получим эйлеров граф, в котором возможен обход каждого полуребра по одному разу; это равносильно тому, что исходное ребро можно пройти два раза в разных направлениях, причем обход обязательно оканчивается в корне. Поэтому для распределенного алгоритма Тэрри [11] коммуникационная сложность равна числу посещений каждого ребра, т.е. $2m$. Очевидно, что в силу последовательного характера любого алгоритма нумерации оценку $\mathbf{O}(m)$ понизить нельзя.

Заметим, что число уровней G и G_{BFS} равно e; следовательно, число циклов нумерации также равно e. Очевидно, что из всех корневых неориентированных графов с n вершинами наибольший эксцентриситет имеет цепь Ch_n, т.е. связный граф с n вершинами, в котором имеются две концевые вершины со степенями 1, причем одна из этих вершин является корнем; все остальные вершины имеют степень 2. При этом

$$e(Ch_n) = n - 1. \tag{1}$$

Оценим сложность цепи Ch_n.

Лемма 1.

$$C_{NBFS}(Ch_n) = (n-1)n. \tag{2}$$

Цепь Ch_n согласно (1) имеет $n-1$ уровень; поэтому для ее нумерации потребуется $n-1$ цикл. В k-м цикле ($k = 1, \ldots, n-1$) участвуют $k+1$ вершин (включая корень). При этом сообщения, которые они порождают, можно представить в виде цепочки длины $k+1$ вида $12\ldots21$, где i-й элемент цепочки ($i \leqslant k+1$) соответствует числу сообщений, порождаемых вершиной $(i-1)$-го уровня. Действительно, корень и последняя вершина цепочки порождают по одному сообщению, а остальные вершины – по два (одно нумерует следующую вершину, другое сообщает текущий номер в сторону корня). Общее число сообщений в одной такой цепочке равно $2k$; соответственно общее число сообщений, порождаемых процессом нумерации цепи длины n, равно сумме сообщений по всем $n-1$ циклам, т.е. удвоенной сумме арифметической прогрессии $2\sum_{k=1}^{n-1} k = (n-1)n$.

Поскольку k-й цикл нумерации вовлекает в обмен сообщениями все черные ребра первых k уровней, ясно, что величина эксцентриситета существенно влияет на величину C_{NBFS}. Более точное утверждение выглядит так.

Лемма 2. Среди всех корневых деревьев с n вершинами наибольшей NBFS-сложностью обладает дерево с наибольшим эксцентриситетом, т.е. цепь Ch_n.

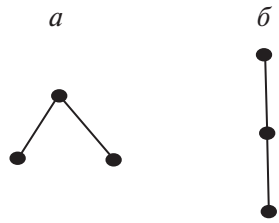

Рис. 7.

Это утверждение докажем по индукции.

Минимальное n, для которого это утверждение имеет смысл, равно 3. При $n = 3$ возможны два дерева, изображенные на рис. 7.

Дерево 7,*а* имеет один уровень и потому требует одного цикла нумерации, которому соответствует цепочка 211 (два сообщения от корня и по одному сообщению от остальных вершин), т.е. 4 сообщения. Дерево 7,*б* – это цепь, имеющая два уровня и два цикла нумерации: первому циклу соответствует цепочка 11, второму циклу – цепочка 121; итого $2 + 4 = 6$ сообщений. Таким образом, C_{NBFS} дерева 7,*б* больше, чем C_{NBFS} дерева 7,*а*, и, следовательно, для $n = 3$ лемма верна.

Предположим теперь, что утверждение леммы верно для всех деревьев с числом вершин, не превосходящим k, т.е. сложность $C_{NBFS}(Ch_k) = (k-1)k$ максимальна для всех деревьев с k вершинами. Докажем, что оно верно для $k+1$: сложность цепи $C_{NBFS}(Ch_{k+1}) = (k+1)k$ максимальна для всех деревьев с $k+1$ вершинами.

Любое дерево с $k+1$ вершинами можно получить из некоторого дерева G_k с k вершинами путем присоединения к G_k ребра (x, y) таким образом, что вершина x отождествляется с некоторой вершиной G_k, а вершина y становится висячей вершиной нового дерева G_{k+1}. Возможны два варианта.

а) G_k – цепь. Если ребро (x, y) присоединяется к концу цепи, то получим цепь Ch_{k+1} со сложностью $(k+1)k$. Если же ребро (x, y) присоединяется к любой другой вершине G_k уровня $i < k$, то уровень полученного дерева G_{k+1} не изменится, поэтому $(k+1)$-го цикла нумерации не будет, а при нумерации $(i+1)$-го уровня появятся два новых сообщения – от вершины y и обратно, причем при этом y станет красной и в дальнейшем сообщений генерировать не будет. Таким образом, $C_{NBFS}(G_{k+1}) = C_{NBFS}(Ch_k) + 2 < C_{NBFS}(Ch_{k+1})$ и, следовательно, в классе всех деревьев с $k+1$ вершинами, полученных присоединением ребра к цепи Ch_k максимальную сложность имеет цепь Ch_{k+1}.

б) G_k – не цепь. Тогда $e(G_k) \leqslant k-1$ и по предположению индукции

$$C_{NBFS}(G_k) \leqslant (k-1)k. \tag{3}$$

Пусть ребро (x, y) присоединяется к висячей вершине v с уровнем $i \leqslant k-1$. Любая висячая вершина является концом некоторой цепи, присоединенной другим концом к вершине со степенью > 2, причем ее длина не пре-

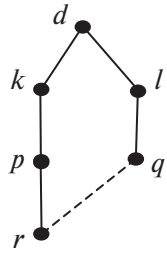

Рис. 8.

восходит $k-1$. При нумерации дерева G_k вершина v (и вся цепь) становится красной на i-м цикле нумерации и в дальнейших циклах не участвует. В новом дереве G_{k+1} эта цепь удлиняется на одно ребро и ее концом вместо v становится вершина y с уровнем $i+1 \leqslant k$. Поэтому эта цепь будет участвовать в $(i+1)$-м цикле нумерации, в котором к сложности $C_{NBFS}(G_k)$ добавится не более чем $2(i+1) \leqslant 2k$ сообщений. Поэтому $C_{NBFS}(G_{k+1}) \leqslant C_{NBFS}(G_k) + 2k$, откуда, используя (3), получим $C_{NBFS}(G_{k+1}) \leqslant (k-1)k + 2k = (k+1)k = C_{NBFS}(Ch_{k+1})$. Таким образом, и в случае б) сложность цепи оказывается максимальной, что и доказывает лемму.

Рассмотрим теперь NBFS-сложность произвольного корневого неориентированного графа G с n вершинами и m ребрами. Цикломатическое число (оно же – число хорд) такого графа вычисляется по известной формуле $\gamma = m - n + 1$.

Теорема 2.

$$C_{NBFS}(n) = \mathbf{O}(n^2 - n). \tag{4}$$

Сложность $C_{NBFS}(G)$ конкретного графа G можно представить в виде суммы двух слагаемых:

$$C_{NBFS}(G) = C_{NBFS}(G_{BFS}) + C_\gamma(G). \tag{5}$$

Первое слагаемое – это сложность остовного дерева графа G. Она определяется сообщениями, которые связаны с собственно нумерацией (присвоение номера и обратное сообщение о текущем номере) и проходят только по входящим ребрам, т.е. по будущему остовному дереву. Согласно леммам 1 и 2 $C_{NBFS}(G_{BFS}) \leqslant n^2 - n$.

Второе слагаемое – это число сообщений, порождаемых при обнаружении хорд. В общем случае с обнаружением хорды (x, y) связано два сообщения: вершина x пытается пронумеровать вершину y; вершина y сообщает, что она уже пронумерована. После этого ребро (x, y) удаляется из МЧР вершин x и y; последующие сообщения по нему не проходят. Однако возможна более сложная ситуация, которая описана в примере 4. Рассмотрим ее более подробно.

На рис. 8 представлен фрагмент некоторого графа, в котором вершины p, q находятся на i-м уровне и идет $(i+1)$-й цикл нумерации. По ветке $dkpr$ он

уже прошел, вершина r получила свой номер и отправила его обратно в вершину p; при этом в МЧР вершины r ребро (r, q) сохраняется. При активизации вершины q она пытается пронумеровать вершину r и получает от нее сообщение, что она уже пронумерована. После этого обе вершины удаляют ребро (r, q) из своих МЧР. Если при этом оказывается, что МЧР вершины r пусто (это возможно, только если в будущем остовном дереве эта вершина – висячая), то она становится красной и посылает еще одно сообщение в вершину p. Вершина p, получив это сообщение, тоже может стать красной, что породит сообщение от нее в вершину k, и т.д. Таким образом, одновременно с процессом нумерации в ветке *dlq* возникнет дополнительная цепочка сообщений в ветке *dkpr*, которая и внесет свой вклад в слагаемое $C_\gamma(G)$. Все эти сообщения делают некоторые ребра красными, что приводит к их удалению из МЧР; поэтому по каждому ребру может пройти только одно такое сообщение. Следовательно, их общее число не превосходит m (общего числа ребер графа), для которого, как известно, справедливо $m \leqslant \frac{n^2-n}{2}$.

Другой вариант этой ситуации описан в примере 5, где горизонтальная хорда соединяет две висячие вершины дерева G_{BFS}. В этом случае возникает дополнительный цикл нумерации. Сообщения этого цикла, как максимум, пройдут не более двух раз по всем входящим ребрам графа. В этом случае вклад хорды в $C_\gamma(G)$ не превысит $2m \leqslant n^2 - n$.

Итак, оба слагаемых не превосходят величину $n^2 - n$, что и доказывает теорему.

Для разных классов графов соотношение между двумя слагаемыми в (5) может сильно отличаться. В частности, в разреженных графах (графах с малым числом хорд) основной вклад в C_{NBFS} вносит сложность остовного дерева, которая, в свою очередь, определяется величиной эксцентриситета корня. Поэтому даже в пределах одного графа сложность будет заметно меняться в зависимости от выбора корня; естественно ожидать, что минимальной она будет, когда корень – центр графа, а максимальной, когда корень – конец одного из диаметров. Верхняя оценка, согласно леммам 1, 2, достигается, когда граф является цепью с корнем на одном из ее концов.

Другой крайний случай – полный граф K_n, у которого при любом выборе корня $e = 1$, число ребер $m = \frac{n^2-n}{2}$; из этих ребер $n - 1$ – входящие, остальные – хорды: $\gamma = \frac{n^2-n}{2} - n + 1$, причем из $e = 1$ следует, что все хорды горизонтальны. Из этих формул видно, что основной вклад в $C_{NBFS}(K_n)$ вносит величина $C_\gamma(G)$, причем достижение верхней оценки обеспечивается числом хорд.

Работа NBFS-алгоритма на графе K_n состоит из двух циклов нумерации. В первом цикле все вершины получают свои номера, но благодаря горизонтальности хорд ни одна вершина не становится красной; поэтому корень об окончании нумерации не знает и запускает второй цикл, в котором обнаруживаются все хорды.

4. Заключение

Задача нумерации имеет универсальное значение для распределенных алгоритмов. Почти все распределенные алгоритмы на графах предполагают, что нумерация вершин уже проведена. Кроме того, наличие единой для всех нумерации позволяет корню после ее проведения запросить от каждой вершины списки инцидентных ей ребер и тем самым восстановить у себя полное описание графа.

Об эффективности NBFS-алгоритма можно судить, сравнив его с алгоритмом Тэрри. Для «плотных» графов, близких по числу ребер к K_n, оценки сложности обоих алгоритмов по порядку совпадают, и, следовательно, NBFS-алгоритм оказывается предпочтительным, поскольку он одновременно строит BFS-дерево. Напротив, для разреженных графов, близких к деревьям, NBFS-алгоритм применять не стоит: во-первых, для таких графов он слишком сильно проигрывает алгоритму Тэрри по сложности (ясно, что цепь можно пронумеровать за один проход), а во-вторых, главное достоинство NBFS-алгоритма (одновременное с нумерацией построение BFS-дерева) здесь не работает, поскольку всякое дерево само по себе уже является BFS-деревом.

Среди возможных применений NBFS-алгоритма, помимо обычных для распределенных алгоритмов приложений к сетям коммуникаций, следует указать групповую робототехнику. Одна из возможных ситуаций: группа роботов отправляется на задание по патрулированию местности, имея полный граф связей между роботами группы. К концу задания некоторые связи нарушены, и лидеру необходимо восстановить граф оставшихся связей. Процент нарушенных связей невелик, граф оставшихся связей близок к полному, и потому NBFS-алгоритм будет эффективен.

Еще один алгоритм, который строит BFS-дерево и нумерует вершины, описан в [16]. Он сначала строит BFS-дерево и уже на этом дереве проводит нумерацию вершин, используя обход дерева, похожий на алгоритм Тэрри, с той существенной разницей, что для дерева, как известно, $m = n - 1$. Вместе с использованием параллелизма (в алгоритме [16] вершины отправляют сигналы одновременно всем своим соседям) это дает оценку временной сложности $O(n)$, что лучше сложности NBFS-алгоритма. Однако за это улучшение приходится платить введением синхронизации, которая в реальных приложениях может либо потребовать дополнительного оборудования, либо просто затруднительна в реализации – например, в приведенном выше примере с групповой робототехникой. Кроме того, нумерация, описанная в [16], выглядит нерегулярно (величина номера вершины ничего не говорит о близости вершины к корню, т.е. числа – это всего лишь уникальные идентификаторы), тогда как в NBFS-алгоритме номера обладают указанным в начале раздела 2 свойством: если $i < j$, то номер любой вершины i-го уровня будет меньше номера любой вершины j-го уровня. Это свойство использовано при доказательстве теоремы 1 и может оказаться полезным в приложениях.

СПИСОК ЛИТЕРАТУРЫ

1. *Левенштейн В.И.* Об одном методе решения задачи синхронизации цепи автоматов за минимальное время // Пробл. передачи информ. 1965. Т. 1. Вып. 4. С. 20–32.
2. *Moore F.R., Langdon G.G.* A generalized firing squad problem // Information and Control. 1968. V. 12. P. 212–220.
3. *Gallager R.G., Humblet P.A., Spira P.M.* A distributed algorithm for minimum-weight spanning trees // ACM Transactions on Programming Languages and Systems. 1983. V. 5. No. 1. P. 66–77.
4. *Peleg D., Rubinovich V.* A near-tight lower bound on the time complexity of distributed MST construction // Proc. 40 IEEE Symp. on Found. of Comp. Sci. (FOCS). 1999. P. 253–261.
5. *Вялый М.Н., Хузиев И.М.* Распределенная коммуникационная сложность построения остовного дерева // Пробл. передачи информ. 2015. Т. 51. № 1. С. 54–71.
6. *Вялый М.Н., Хузиев И.М.* Быстрые протоколы выбора лидера и построения остовного дерева в распределенной сети // Пробл. передачи информ. 2017. Т. 53. № 2. С. 91–111.
7. *Dinitz M., Halldórsson M., Izumi T., Newport C.* Distributed minimum degree spanning trees // Proceedings 2019 ACM Symposium Principles Distributed Computing, 2019. P. 511–520.
8. *Awerbuch B., Gallagher R.G.* Distributed BFS algorithms // Proc. 26 IEEE Symposium Foundations Computer Science (FOCS). 1985. P. 250–255.
9. *Park J., Tokura N., Masuzawa T., Hagihara K.* An efficient distributed algorithm for constructing a breadth-first search tree // Systems and Computers in Japan. 1989. V. 20. P. 15–30.
10. *Makki S.A.M.* Efficient distributed breadth-first algorithm // Computer Communications. 1996. V. 19. No. 8. P. 628–636.
11. *Бурдонов И., Косачев А.* Общий подход к решению задач на графах коллективом автоматов // Труды ИСП РАН. 2017. Т. 29. Вып. 2. С. 27–76.
12. *Бурдонов И., Косачев А., Сортов А.* Распределённые алгоритмы на корневых неориентированных графах // Труды ИСП РАН, 2017. Т. 29. Вып. 5. С. 283–310.
13. *Ghaffari M.* Distributed Graph Algorithms. 2022. https://people.csail.mit.edu/ghaffari/DA22/Notes/DGA.pdf
14. *Оре О.* Теория графов. М.: Наука. 1968.
15. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: Построение и анализ. М.: МЦНМО. 2001.
16. *Métivier Y., Robson J.M., Zemmari A.* A distributed enumeration algorithm and applications to all pairs shortest paths, diameter... // Information and Computation. 2016. V. 247. P. 141–151.

Статья представлена к публикации членом редколлегии П.Ю. Чеботаревым.

Поступила в редакцию 11.02.2025

После доработки 29.05.2025

Принята к публикации 12.09.2025