DOI: 10.15514/ISPRAS-2019-31(4)-13

Самотрансформация деревьев с ограниченной степенью вершин с целью минимизации или максимизации индекса Винера

И.Б. Бурдонов, ORCID: 0000-0001-9539-7853 <igor@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Рассматривается распределённая сеть, граф связи которой является неориентированным деревом. Предполагается, что сеть может сама менять свою топологию. Для этого предлагается предельно локальная атомарная трансформация – добавление ребра, соединяющего разные концы двух смежных ребер, и одновременное удаление одного из этих ребер. Такая трансформация выполняется по «команде» от вершины дерева, а именно, общей вершины двух смежных ребер. Показывается, что из любого дерева можно получить любое другое дерево с помощью только атомарных трансформаций. Если рассматриваются деревья с ограничением d ($d \geq 3$) на степени вершин, то трансформация не нарушает этого ограничения. В качестве примера цели такой трансформации рассматриваются задачи максимизации и минимизации индекса Винера дерева с ограниченной степенью вершин без изменения множества его вершин. Предлагаются соответствующие распределённые алгоритмы и доказываются линейные оценки их сложности.

Ключевые слова: распределенная сеть; самотрансформация графа; индекс Винера

Для цитирования: Бурдонов И.Б. Самотрансформация деревьев с ограниченной степенью вершин с целью минимизации или максимизации индекса Винера. Труды ИСП РАН, том 31, вып. 4, 2019 г., стр. 189-210. DOI: 10.15514/ISPRAS-2019-31(4)-13

Благодарности. Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект 17-07-00682-а.

Self-transformation of trees with a bounded degree of vertices to minimize or maximize the Wiener index

I.B. Burdonov, ORCID: 0000-0001-9539-7853 <igor@ispras.ru>
Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. We consider a distributed network whose communication graph is a non-oriented tree. It is assumed that the network itself can change its topology. For this, an extremely local atomic transformation is proposed - the addition of an edge connecting the different ends of two adjacent edges, and the simultaneous removal of one of these edges. This transformation is performed by "command" from the vertex of the tree, namely, the common vertex of two adjacent edges. It is shown that any other tree can be obtained from any tree using only atomic transformations. If trees with degree bounded d (d>2) are considered, then the transformation does not violate this restriction. As an example of the goal of such a transformation, the tasks of maximizing and minimizing the Wiener index of a tree with a bounded degree of vertices without changing the set of its vertices are considered. Corresponding distributed algorithms are proposed and linear estimates of their complexity are proved.

Keywords: distributed network; transformation of graphs; Wiener index

For citation: Burdonov I.B. Self-transformation of trees with a bounded degree of vertices to minimize or maximize the Wiener index. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 4, 2019. pp. 189-210 (in Russian). DOI: 10.15514/ISPRAS-2019-31(4)-13

Acknowledgements. This work was supported by the Russian Foundation for Basic Research, project 17-07-00682-a.

1. Введение

В этой статье предлагаются два алгоритма самотрансформации неориентированного дерева, не нарушающие ограничения d ($d \geq 3$) на степени вершин, с целью минимизации или максимизации индекса Винера. Индекс Винера определяется как сумма расстояний между всеми парами вершин неориентированного графа. Он был предложен американским химиком Гарри Винером в 1947 г. [0], является наиболее старым из известных топологических индексов молекулярных графов, и используется во многих приложениях, особенно в математической и компьютерной химии. Мы далее предлагаем краткий обзор результатов в рассматриваемой области.

Максимальный индекс Винера, очевидно, имеет линейное дерево, содержащее две листовые вершины и путь между ними (или содержащее только одну изолированную вершину). Вид дерева с числом вершин n и ограничением d ($d \geq 3$) на степени вершин исследовался в работе [[2]], результаты которой (определение 2.1 и теорема 2.2., стр. 129) использованы в данной статье, и, специально для бинарных деревьев, в работе [[3]].

Самотрансформация дерева понимается как самотрансформация распределенной сети, топология которой и есть это дерево, динамически меняющееся в процесс трансформации. Это является частным случаем динамического графа, который формально определяется как последовательность «классических» (стационарных) графов, переход между которыми осуществляется теми или иными операциями. Изучением таких графов занимается зарождающаяся динамическая теория графов [[4], [5]], которая является теоретической основой для конструирования алгоритмов взаимодействия подвижных абонентов (вершин графа) и изучения самоорганизующихся сетей [[6]] различного физического и технического происхождения, в том числе, социальных сетей, нейронных сетей [[7]] и роевого интеллекта [[8], [9]].

К самоорганизующимся сетям относят Mesh (ячеистые), ad-hoc сети, беспроводные сенсорные сети (WSN) и др. [[6], [10], [11]]. Особенностью этих сетей является отсутствие разделения узлов на коммутаторы и хосты, а также отсутствие контроллеров, имеющих доступ ко всем или части коммутаторов и настраивающих, в частности, их таблицы маршрутизации. При исследовании таких сетей основное внимание уделяется вопросам маршрутизации, пропускной способности, помехоустойчивости, безопасности, распределения нагрузки и сетевых ресурсов, и т.п. С точки зрения самой сети изменение ее топологии является внешним фактором, который надо учитывать, но которым сама сеть не управляет или управляет лишь частично [[12], [13]]. Например, вершины графа понимаются как подвижные агенты в трехмерном пространстве, а наличие или отсутствие ребра $\{a, b\}$ определяется расстоянием в этом пространстве между агентами a и b. Когда агенты сближаются ближе некоторого порогового значения, ребро добавляется, а когда удаляются на большее расстояние, ребро удаляется. Задача заключается в корректировке информации, управляющей работой сети, в особенности маршрутизацией, при появлении или удалении вершин и ребер.

С другой стороны, в литературе много работ, посвященных как раз целенаправленной трансформации графа и, в частности, деревьев с целью оптимизации по тем или иным критериям. Одним из таких критериев как раз и является минимум или максимум индекса Винера. Такие трансформации носят глобальный или локальный характер. Примером глобальной трансформации может служить конструкция Мычельского (Mycielskian) [[14]],

когда к графу из n вершин и m ребер добавляется сразу $(n + 1)$ новых вершин и $(n + 2m)$ новых ребер.

В данной статье предлагается, напротив, предельно локальная трансформация, затрагивающая минимум вершин и ребер, максимально близких друг другу. Конкретно предлагается атомарная трансформация вида $a \to c \to b$, при которой конец c ребра $\{a, c\}$ «двигается» вдоль ребра $\{c, b\}$ к другому его концу b. Это сводится к одновременному удалению ребра $\{a, c\}$ и добавлению ребра $\{a, b\}$, а условием является наличие ребра $\{b, c\}$. Если граф является деревом, то при такой трансформации не появляются циклы, и сохраняется множество вершин.

Ранее специально для деревьев рассматривалась «элементарная» трансформация, которая также не добавляла циклов и не меняла множество вершин: добавление нового ребра $\{a, b\}$ и удаление в образовавшемся цикле $a = b_1, \ldots b_k = b$, a одного из старых ребер $\{b_i, b_{i+1}\}$. В [[15]] показано, что любое дерево может быть получено из любого другого дерева с тем же множеством вершин с помощью конечного числа таких трансформаций (теорема 4.3, стр. 245). Эта трансформация, однако, не достаточно локальна, поскольку в ней участвуют вершины a и b и ребро $\{b_i, b_{i+1}\}$, которые могут сколь угодно далеко отстоять друг от друга. Тем не менее, эта трансформация сводится к двум цепочкам наших атомарных трансформаций $b_{i+1} \to b_i \to b_{i-1}$, $b_{i+1} \to b_{i-1} \to b_{i-2}$, …, $b_{i+1} \to b_2 \to a$ и $a \to b_{i+1} \to b_{i+2}$, $a \to b_{i+2} \to b_{i+3}$, …, $a \to b_{k-1} \to b$.

Трансформации наиболее близкие к нашей атомарной трансформации рассматривались в [[16], [17]]. В [[16]] трансформация, названная «the edge-growth transformation», применяется относительно (не висячего) ребра $\{a, b\}$ и заключается в замене всех ребер $\{b, c_1\}$, $\{b, c_2\}$, …, $\{b, c_k\}$, где $c_i \neq a$, на ребра $\{a, c_1\}$, $\{a, c_2\}$, …, $\{a, c_k\}$, после чего ребро $\{a, b\}$ становится висячим. Это эквивалентно множеству наших атомарных трансформаций вида $c_1 \to b \to a$, $c_2 \to b \to a$, …, $c_k \to b \to a$. Заметим, что эти атомарные трансформации могут выполняться одновременно, поскольку они «не мешают» друг другу: не удаляют ребро $\{a, b\}$, наличие которого является условием выполнения каждой из этих трансформаций. Трансформация из [[17]] названа «diameter-growing transformation relative to the pendent edge» и совпадает с нашей атомарной трансформацией $a \to c \to b$. Однако она применяется для специального случая, когда ребро $\{c, b\}$ висячее, а ребро $\{a, c\}$ лежит на самом длинном пути (longest path), что приводит к увеличению длины пути и, соответственно, индекса Винера.

Ключевым аспектом предлагаемых в данной статье алгоритмов является то, что это распределенные и параллельные алгоритмы, выполняемые совместно вершинами дерева. Дерево трансформирует само себя по «командам» от его вершин (точнее, от вычислительных единиц, соотносимых с вершинами). Как раз для этого нужна предельная локальность атомарной трансформации $a \to c \to b$: «команду» на ее выполнение подает вершина c, «знающая» только об инцидентных ей ребрах и указывающая два из них $\{a, c\}$ и $\{c, b\}$ как параметры «команды». Сообщения, которыми вершины могут обмениваться, пересылая их по ребрам графа, как это обычно и происходит в распределенной сети, нужны для координации совместных действий вершин с целью достижения цели. В данной статье такой целью является минимизация или максимизация индекса Винера.

Структура статьи следующая. После введения разд. 2 содержит описание используемой модели распределенной сети. В разд. 3 определяется атомарная трансформация и показывается ее достаточность для любых преобразований деревьев. В разд. 4 определяются основные понятия и доказываются основные утверждения, которые связаны с индексом Винера и на которые опираются предлагаемые алгоритмы. Сами алгоритмы описываются в разд. 5, там же доказываются их корректность и линейные оценки сложности. В заключении намечаются направления дальнейших исследований.

2. Модель

Рассматривается распределённая сеть с неориентированным графом связей $G = (V, E)$ без кратных рёбер и петель, где V – множество вершин, $E \subseteq \{ \{a, b\} \mid a \in V \,\&\, b \in V \}$ – множество рёбер. В таком графе путь длины k из вершины a в вершину b однозначно задаётся последовательностью вершин $a = a_1, \ldots, a_{k+1} = b$.

Граф предполагается *упорядоченным*: рёбрам, инцидентным каждой вершине $v \in V$, присвоены различные ненулевые номера, номер ребра 0 зарезервирован для служебных целей. Тем самым, каждое ребро $\{a, b\}$ имеет два номера: в вершине a и в вершине b, которые мы будем обозначать как $e(a, b)$ и $e(b, a)$, соответственно.

В каждой вершине графа находится вычислительная единица (автомат), которая может посылать сообщения по инцидентным вершине рёбрам и принимать сообщения по этим рёбрам, посланные с других их концов. Для краткости там, где это не приводит к двусмысленности, мы будем вместо «вычислительная единица, находящаяся в вершине», говорить просто «вершина». Память вершины будет рассматриваться как набор переменных. Переменные, значения которых сохраняются между приёмом сообщений, будем называть *постоянными* переменными; остальные переменные – *временные*. Постоянную переменную p в вершине x будем обозначать $p(x)$, а временную переменную q – просто q. Предполагается, что с самого начала в каждой вершине a инициализирована постоянная переменная $E(a) = \{ e(a, b) \mid \{a, b\} \in E \}$ – множество номеров $e(a, b)$ рёбер $\{a, b\}$, инцидентных вершине a. В дальнейшем при трансформации графа вершина a сама корректирует переменную $E(a)$.

Сообщение будет указываться его типом и набором параметров: $Tun(параметр_1, \ldots, параметр_k)$. Когда вершина a посылает сообщение по ребру $\{a, b\}$, она указывает номер $e(a, b)$ этого ребра. Когда вершина b принимает сообщение по ребру $\{a, b\}$, ей становится известным номер $e(b, a)$ этого ребра. Ребро $\{a, b\}$ с указанием направления пересылки сообщения из a в b будем обозначать $a \to b$.

Предполагается, что сообщения генерируются только вершинами (не рёбрами), и сообщения, передаваемые по ребру, не теряются и не обгонят друг друга.

Для оценки времени работы предлагаемых далее алгоритмов будем считать, что каждое сообщение по каждому ребру графа перемещается недетерминированное время, которое не больше 1 такта. Иными словами, мы будем оценивать длины пройденных сообщениями путей в «наихудшем» случае.

В данной статье рассматривается модель, в которой граф связей G является деревом с заданным ограничением d ($d \geq 3$) на степени его вершин: для каждой вершины v её степень не превышает d. Случай $d < 3$ тривиален, поскольку для каждого числа вершин n и ограничения d ($d < 3$) на степени вершин существует не более одного дерева с точностью до изоморфизма: это линейное дерево.

3. Трансформация дерева

Мы рассматриваем динамические деревья, которые трансформируется не случайно, а по «командам» от их вершин. Целью такой трансформации является достижение некоторого оптимального вида дерева. Мы будем рассматривать только такие трансформации, которые не меняют множество вершин дерева, оставляют его деревом и не превышают заданного ограничения d на степени вершин дерева.

Атомарной трансформацией назовём замену ребра $\{a, c\}$ на ребро $\{a, b\}$ при условии, что в дереве есть ребро $\{c, b\}$. Эту трансформацию будем обозначать $a \to c \to b$. Будем предполагать, что сообщения, передаваемые по изменяемому ребру в момент атомарной трансформации, не теряются, но если сообщение направлялось в вершину c, то получит его вершина b.

Утверждение 1. Атомарная трансформация не меняет множество вершин дерева и оставляет его деревом.

Доказательство. Условием атомарной трансформации $a \to c \to b$ является наличие в графе рёбер $\{a, c\}$ и $\{c, b\}$. Если бы после трансформации появился цикл, он проходил бы по добавляемому ребру $\{a, b\}$, т.е. имел бы вид $a, b, d_1, \ldots, d_k, a$. Но тогда до трансформации был бы цикл $a, c, b, d_1, \ldots, d_k, a$, чего быть не может. Следовательно, в результате трансформации циклы не появляются. Поскольку граф до трансформации был связным, в нём для любых двух вершин v и w существовал путь из v в w. Если этот путь проходил через удаляемое ребро $\{a, c\}$, т.е. имел вид $v, \ldots, a, c, \ldots, w$, то после трансформации будет существовать путь $v, \ldots, a, b, c, \ldots, w$, т.е. граф останется связным.□

Развилкой называется вершина степени 3 или больше. *Линейным деревом* называется дерево без развилок. Очевидно, дерево линейное тогда и только тогда, когда в нём ровно два листа (вершин степени 1), или оно содержит только одну изолированную вершину.

Утверждение 2. Любое дерево можно преобразовать в линейное дерево цепочкой атомарных трансформаций без нарушения ограничения d на степень вершин.

Доказательство. Пусть дерево не линейное. Выберем произвольный лист c_1 и будет двигаться от него в дереве по пути $c_1, c_2, \ldots, c_k$ до ближайшей развилки c_k. В дереве должны быть, по крайней мере, два ребра $\{a, c_k\}$ и $\{b, c_k\}$, где $a \ne c_{k-1}$ и $b \ne c_{k-1}$. Выполним цепочку атомарных трансформаций $a \to c_k \to c_{k-1}$, $a \to c_{k-1} \to c_{k-2}$, …, $a \to c_2 \to c_1$. По утверждению 1 каждая из трансформаций $a \to c_i \to c_{i-1}$ не меняет множество вершин дерева и оставляет его деревом. Такая трансформация уменьшает на 1 степень вершины c_i и увеличивает на 1 степень следующей вершины c_{i-1}, которая до этого была равна 2, если $i < 2$, или 1, если $i = 2$. Тем самым, каждая из этих трансформаций не нарушает ограничения d на степень вершин. Кроме того, каждая из этих трансформаций, кроме последней, не меняет число листьев дерева, а последняя трансформация $a \to c_2 \to c_1$ делает лист c_1 внутренней (не листовой) вершиной. Тем самым, эта цепочка атомарных трансформаций уменьшает число листьев дерева на 1. Будем выполнять эту процедуру выбора листа и выполнения цепочки атомарных трансформаций до тех пор, пока дерево не станет линейным при числе листьев 2.□

Утверждение 3. Любое дерево можно получить из линейного дерева с тем же множеством вершин цепочкой атомарных трансформаций без нарушения ограничения d на степень вершин.

Доказательство. Достаточно заметить, что атомарная трансформация обратима: после трансформации $T = a \to c \to b$, которая выполняется при наличии ребра $\{c, b\}$, можно сделать обратную трансформацию $T' = a \to b \to c$. Если в дереве до трансформации $a \to c \to b$ степени вершин не превышали d, то, очевидно, обратная трансформация не нарушает ограничения d на степени вершин. Соответственно, после выполнения цепочки атомарных трансформаций $T_1, T_2, \ldots T_{k-1}, T_k$ можно выполнить обратную цепочку $T_k', T_{k-1}', \ldots T_2', T_1'$. А тогда, поскольку любое дерево G можно трансформировать в линейное дерево по утверждению 2, то можно выполнить и обратную трансформацию линейного дерева в дерево G.□

Из утверждений 2 и 3 непосредственно следует, что для любых двух деревьев с ограничением d ($d \ge 3$) на степени вершин одно можно получить из другого с помощью цепочки атомарных трансформаций, причем по утверждению 1 в процессе трансформации ограничение d на степени вершин не будет нарушено.

Будем считать, что атомарная трансформация $a \to c \to b$ выполняется тогда, когда вершина c подаёт команду ***Изменить*** ($e(c, a)$, $e(c, b)$, P_c), где P_c – дополнительные параметры, формируемые вершиной c и зависящие от того алгоритма, при выполнении которого делается эта трансформация. Выполнение этой команды заключается в следующем (рис. 1): 1) ребро $\{a, c\}$ меняет свою конечную вершину c на вершину b, 2) ребро $\{a, b\}$ получает в вершине a тот же номер, которое в этой вершине имело удаляемое ребро $\{a, c\}$, т.е. $e(a, c)$, а в вершине b – некоторый «свободный» номер $e(b, a)$, который не был номером какого-либо ребра, инцидентного вершине b в момент подачи команды ***Изменить***, 3) для того чтобы вершина b

«узнала» о появлении у неё нового ребра $\{a, b\}$, по ребру $a \to b$ автоматически посылается сообщение *Изменение*(P_c). Мы будем считать, что время атомарной трансформации, включая пересылку сообщения *Изменение*, не превышает 1 такта.

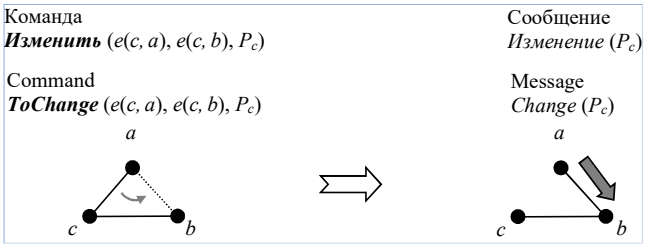

Рис. 1. Трансформация $a \to c \to b$: замена ребра $\{a, c\}$ на ребро $\{a, b\}$
Fig. 1. Transformation $a \to c \to b$: replacement of an edge $\{a, c\}$ by an edge $\{a, b\}$

После подачи команды ***Изменить*** ($e(c, a)$, $e(c, b)$, P_c) вершина c сама удаляет ребро $e(c, a)$ из множества $E(c) := E(c) \setminus \{e(c, a)\}$. Получив сообщение *Изменение*(P_c) по ребру $e(b, a)$, вершина b сама добавляет номер ребра во множество $E(b) := E(b) \cup \{e(b, a)\}$. Заметим, что если в момент подачи команды ***Изменить*** по ребру $a \to c$ пересылалось некоторое сообщение, то его получит не вершина c, а вершина b, причём раньше сообщения *Изменение*. Вершина b может распознать этот случай при получении сообщения по ребру с номером $e(b, a) \notin E(b)$.

4. Индекс Винера

Одной из важных числовых характеристик графа является *индекс Винера*, определяемый как сумма всех попарных расстояний между вершинами.

Максимальный индекс Винера, очевидно, имеет линейное дерево. Индекс Винера линейного дерева с n вершинами равен $(n - 1)n(n + 1)/6$ (последовательность A000292 [[18]]).

Без ограничения на степени вершин минимальный индекс Винера имеет граф-звезда: связный граф, все рёбра которого исходят из одной вершины. Индекс Винера звезды с n вершинами равен $(n - 1)^2$ (последовательность A000290 [[18]]). Но нас будут интересовать деревья, степень вершин которых ограничена сверху некоторым заданным числом $d \ge 3$. Вид дерева с минимальным индексом Винера на классе деревьев с n вершинами и ограничением d на степень вершин определён в [[2]]. Мы будем называть его *хорошим* деревом, термин взят из [[3]], где он применяется для бинарных ($d = 3$) хороших деревьев. Определим хорошее дерево своими словами.

Корневым деревом называется дерево, в котором выделена одна вершина, называемая *корнем*. *Высотой вершины* в корневом дереве называют расстояние от вершины до корня. *Высотой корневого дерева* называют максимум из высот его вершин. *Ветвью вершины* v в корневом дереве G называется подграф $G(v)$, порождённый множеством вершин, связанных с корнем путём, проходящим через v. Для ребра $\{a, b\}$ вершину a называют *отцом* вершины b, а вершину b – *сыном* вершины a, если путь из корня в вершину b проходит через вершину a. У каждой вершины, кроме корня, есть ровно один отец. ([19][1])

Если корневое дерево с корнем r упорядочено, то множество сынов каждой вершины v линейно упорядочено: $w_1, \ldots, w_k$. Будем говорить, что вершина w_i расположена *левее* вершины w_j, а вершина w_j расположена *правее* вершины w_i, если $i < j$. Эти линейные порядки сынов каждой вершины индуцируют линейный порядок вершин одной высоты h: вершина v расположена *левее* вершины v', если после общего префикса путей, ведущих из корня в эти

[1] В последнее время в англоязычной литературе употребляется асексуальная пара терминов *parent-child (родитель-ребенок)*.

вершины r, v_1, …, v_i, v_{i+1}…, $v_h = v$ и r, v_1, …, v_i, v'_{i+1}…, $v'_h = v'$, т.е. после вершины v_i, следующая вершина v_{i+1} на пути в вершину v расположена левее следующей вершины v'_{i+1} на пути в вершину v'. Соответственно, вершина v' расположена *правее* вершины v. Для любой вершины v на её ветви $G(v)$ любая вершина w имеет то же множество сынов, что и в дереве G. Поэтому, если дерево G упорядочено, то будем считать, что ветвь $G(v)$ также упорядочена: для каждой вершины на ветви $G(v)$ задан тот же линейный порядок её сынов, что в дереве G. Очевидно, что для любой вершины v на высоте h линейный порядок множества вершин ветви $G(v)$, находящихся в этой ветви на высоте k, является отрезком линейного порядка множества вершин дерева G на суммарной высоте $h + k$. Пусть задан путь из корня в лист v высотой h. Будем говорить, что вершина w на высоте $k \le h$ расположена *левее* (*правее*) этого пути, если она расположена левее (правее) вершины, лежащей на этом пути и имеющей ту же высоту k.

Корневое дерево высотой h с n вершинами и ограничением d ($d \ge 3$) на степени вершин будем называть *почти хорошим*, если дерево может быть так упорядочено, что 1) степень корня равна $min\{d - 1, n - 1\}$, 2) для $h \ge 3$ все вершины на высоте 1 .. h - 2 имеют степень d, 3) для $h \ge 2$ на высоте h - 1 самая правая внутренняя вершина u имеет степень не больше d, вершины левее вершины u имеют степень d, а вершины правее вершины u имеют степень 1 (листья). Эту вершину u будем называть *разделяющей* вершиной в дереве. Пример почти хорошего дерева для $d = 3$, $h = 4$, $n = 26$ на рис. 2 справа.

Определение *хорошего* дерева отличается от определения почти хорошего дерева только первым условием: степень корня равна $min\{d, n - 1\}$. Пример хорошего дерева для $d = 3$, $h = 4$, $n = 27$ на рис. 2 слева.

Почти хорошее дерево высотой h будем называть *правильным почти хорошим деревом*, если 1) корень имеет степень 0 или d - 1, и 2) для $h \ge 2$ все вершины на высоте h - 1 имеют степень d. Очевидно, что все листья находятся на высоте h. Число вершин такого дерева обозначим $N(d, h) = 1 + (d - 1) + (d - 1)^2 + … + (d - 1)^h = ((d - 1)^{h+1} - 1) / (d - 2)$. Пример правильного почти хорошего дерева для $d = 3$, $h = 3$, $n = 15$ на рис. 2 справа, если удалить «серые» вершины на высоте 4.

Хорошее дерево высотой h будем называть *правильным хорошим деревом*, если 1) корень имеет степень 0 или d, и 2) для $h \ge 2$ все вершины на высоте h - 1 имеют степень d. Очевидно, что все листья находятся на высоте h. Число вершин такого дерева обозначим $M(d, h)$. Очевидно, $M(d, 0) = 1$ и $M(d, h) = 1 + d + d(d - 1) + … + d(d - 1)^{h-1} = 1 + dN(d, h - 1)$ для $h \ge 1$. Пример правильного хорошего дерева для $d = 3$, $h = 3$, $n = 22$ на рис. 2 слева, если удалить «серые» вершины на высоте 4.

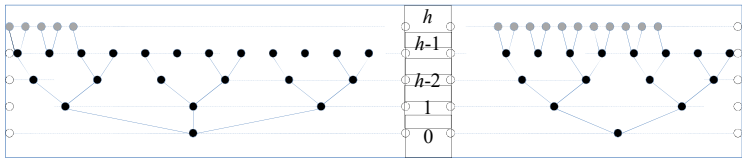

Рис. 2. Хорошее дерево (слева) и почти хорошее дерево (справа)
Fig. 2. Good tree (left) and almost good tree (right).

Утверждение 4. (Теорема 2.2. в [[2]]) Дерево с n вершинами и максимальной степенью вершин не больше d ($d \ge 3$) имеет минимальный индекс Винера тогда и только тогда, когда это хорошее дерево.

Утверждение 5. Для каждого $n \ge 1$ существует и единственное с точностью до изоморфизма, сохраняющего корень, (почти) хорошее дерево с числом вершин n.

Доказательство. Сначала докажем индукцией по n существование (почти) хорошего дерева с n вершинами. Для $n = 1$ утверждение очевидно. Пусть утверждение доказано для n и докажем его для $n + 1$. Пусть G (почти) хорошее дерево с n вершинами высотой h. Если это дерево

правильное, выберем самую левую вершину на высоте h, и соединим её ребром с новым листом. Очевидно, получится (почти) хорошее дерево с числом вершин $n + 1$ и высотой $h + 1$. Если дерево G не является правильным, то на высоте h - 1 можно выбрать самую левую вершину u степени меньше d. Соединим её ребром с новым листом. Очевидно, получится (почти) хорошее дерево с числом вершин $n + 1$ и высотой h.

Теперь покажем единственность с точностью до изоморфизма (почти) хорошего дерева с числом вершин n. Для вершины v обозначим через h_v высоту вершины v, а через i_v ее порядковый номер в линейном порядке вершин той же высоты. Изоморфизм определяется следующим соответствием вершин: вершины v и v' двух деревьев с n вершинами соответствуют друг другу, если $h_v = h_{v'}$ и $i_v = i_{v'}$. Если v корень, $h_v = 0$ и $i_v = 1$. Для завершения доказательства достаточно показать, что существование ребра $\{v, w\}$, где вершина v отец вершины w, однозначно определяется тем, является ли вершина v корнем хорошего дерева, и парами чисел (h_v, i_v) и (h_w, i_w). Действительно, если ребро $\{v, w\}$ существует, то $h_w = h_v + 1$. Из определения (почти) хорошего дерева следует, что все вершины на высоте h_v, расположенные левее вершины v, имеют d - 1 сынов, поэтому $i_w \ge (d - 1)(h_v - 1) + 1$. Если v корень хорошего дерева, то у вершины v не больше d сынов, поэтому $i_w \le (d - 1)h_v + 1$. В противном случае у вершины v не больше d - 1 сынов, поэтому $i_w \le (d - 1)h_v$. Если вершины v и w существуют, то эти соотношения пар чисел (h_v, i_v) и (h_w, i_w) однозначно определяют, существует ли ребро $\{v, w\}$, где вершина v отец вершины w.□

Если дерево G имеет высоту h, то его максимальное поддерево с тем же корнем, имеющее высоту h - 1, обозначим $G^\wedge$.

Утверждение 6. Пусть G (почти) хорошее дерево G с n вершинами высотой $h > 0$. Тогда 1) дерево $G^\wedge$ является правильным (почти) хорошим деревом высотой h - 1 2); 2) число вершин дерева $G^\wedge$ равно $N(d, h - 1)$, если G почти хорошее дерево, и равно $M(d, h - 1)$, если G хорошее дерево; 3) число вершин дерева G на высоте h равно $n - N(d, h - 1)$, если G почти хорошее дерево, и равно $n - M(d, h - 1)$, если G хорошее дерево.

Доказательство. 1-я часть утверждения следует из определений (правильного) (почти) хорошего дерева. 2-я часть утверждения следует из 1-й части утверждения и определения величин $N(d, h - 1)$ и $M(d, h - 1)$ как числа вершин правильного почти хорошего дерева и правильного хорошего дерева высотой $h - 1$. 3-я часть утверждения следует из 1-й и 2-й частей утверждения и обозначения числа вершин дерева G через n.□

Утверждение 7. В (почти) хорошем дереве G ветвь $G(v)$ соседней с корнем вершины v является почти хорошим деревом.

Доказательство. Пусть h высота дерева G, n' число вершин ветви $G(v)$, а $h' \le h - 1$ ее высота. Вершина v имеет в $G(v)$ степень на 1 меньше её степени в G, а степени остальных вершин $G(v)$ одинаковы в $G(v)$ и в G. Докажем для $G(v)$ выполнение условий в определении почти хорошего дерева.

- Условие 1. Если $h = 1$, вершина v имеет в G степень 1 и, следовательно, имеет в $G(v)$ степень 0, а $n' = 1$. Имеем: $0 = min\{d - 1, n' - 1\}$. Если $h = 2$, то все сыны вершины v листья, поэтому ее степень равна $n' - 1$. По условию 3 для G вершина v имеет в G степень от 1 до d и, следовательно, имеет в $G(v)$ степень от 0 до d - 1. Имеем: $n' - 1 = min\{d - 1, n' - 1\}$. Если $h \ge 3$, то по условию 2 для G вершина v имеет в G степень d и, следовательно, имеет в $G(v)$ степень d - 1, а $n' \ge d$. Имеем: $d - 1 = min\{d - 1, n' - 1\}$.
- Условие 2. Пусть $h' \ge 3$ и вершина w принадлежит $G(v)$ и находится в $G(v)$ на высоте от 1 до $h' - 2$. Тогда $h \ge 4$ и в G вершина w находится на высоте от 2 до $h' - 1 \le h - 2$. По условию 2 для G вершина w в G имеет степень d, следовательно, в $G(v)$ она тоже имеет степень d.
- Условие 3. Пусть $h' \ge 2$ и вершина w принадлежит $G(v)$ и находится в $G(v)$ на высоте $h' - 1$. Тогда в G вершина w находится на высоте h'. Поскольку по условиям 2 и 3 для G

все листья в G находятся на высоте h или h - 1, в $G(v)$ все листья находятся на высоте h - 1 или h - 2. Отсюда $h' = h$ - 1 или $h' = h$ - 2.

Если $h' = h$ - 1, то $h \ge 3$ и по условию 3 для G в G на высоте h - 1 есть разделяющая вершина u. Линейный порядок $w_1, \ldots, w_m$ вершин ветви $G(v)$ на высоте h' - 1 в $G(v)$ является отрезком линейного порядка вершин G на высоте $h' = h$ - 1 в G. В $G(v)$ есть внутренняя вершина, лежащая в $G(v)$ на высоте h' - 1 и, следовательно, лежащая в G на высоте $h' = h$ - 1. Поэтому в G вершина u лежит не левее вершины w_1. Если u одна из вершин $w_1, \ldots, w_m$, выберем $u' = u$. Если u в G лежит правее w_m, выберем $u' = w_m$. В любом из этих двух случаев в $G(v)$ на высоте h' - 1 вершина u' является разделяющей. Если $h' = h$ - 2, то $h \ge 4$ и по условию 2 для G в G на высоте h - 2 все вершины имеют степень d. Следовательно, в $G(v)$ все вершины на высоте h - 3 = h' - 1 имеют степень d. В $G(v)$ самая правая вершина на высоте h' - 1 является разделяющей.□

Пусть (почти) хорошее корневое дерево G имеет n вершин. Упорядочим соседей корня $v_1, v_2, \ldots$ в порядке невозрастания числа вершин в их ветвях и обозначим эти числа:

- для почти хорошего дерева G: $N(d, n, 1) \ge \ldots \ge N(d, n, min\{d - 1, n - 1\})$,
- для хорошего дерева G: $M(d, n, 1) \ge \ldots \ge M(d, n, min\{d, n - 1\})$.

Следующее утверждение определяет эти числа.

Утверждение 8. Пусть $d \ge 3$ и G (почти) хорошее корневое дерево с n вершинами. Пусть $L(d, i) = N(d, i)$, если G почти хорошее дерево, и $L(d, i) = M(d, i)$, если G хорошее дерево. Пусть $n = L(d, h - 1) + m(d - 1) + r < L(d, h)$, где $0 \le r < d$ - 1. Пусть $m = p(d - 1)^{h-2} + q$, где $0 \le q < (d - 1)^{h-2}$. Тогда для p самых левых соседей корня их ветви имеют по $N(d, h - 1)$ вершин, для следующего справа соседа корня его ветвь имеет $N(d, h - 2) + q(d - 1) + r$ вершин, а для остальных соседей корня их ветви имеют по $N(d, h - 2)$ вершин.

Доказательство. Пусть $n = L(d, h - 1)$. Тогда $p = q = r = 0$ и по утверждению 5 дерево G правильное (почти) хорошее дерево высотой h - 1. По утверждению 7 ветвь каждого соседа корня – это почти хорошее дерево. Поскольку в правильном дереве высотой h - 1 все листья находятся на высоте h - 1, в ветви соседа корня все листья находятся на высоте h - 2. Следовательно, ветвь соседа корня правильное почти хорошее дерево высотой h - 2. Поэтому ветви всех соседей корня имеют по $N(d, h - 2)$ вершин.

Пусть $L(d, h - 1) < n < N(d, h)$. Тогда для $n = L(d, h - 1) + m(d - 1) + r$, где $0 \le r < d$ - 1, дерево G имеет высоту h, а все листья находятся на высоте h - 1 или h. На высоте h - 1 имеется m вершин степени d, одна вершина имеет степень $r + 1$, а остальные вершины – листья.

Пусть $m = p(d - 1)^{h-2} + q$, где $0 \le q < (d - 1)^{h-2}$. Правильное почти хорошее дерево высотой h - 2 на этой высоте имеет $(d - 1)^{h-2}$ листьев. Поэтому для p самых левых соседей корня дерева G ветвь каждого из этих соседей имеет листья на одной высоте h в дереве G, т.е. на высоте h - 1 в ветви соседа. Следовательно, это правильные почти хорошие деревья высотой h - 1, поэтому они имеют по $N(d, h - 1)$ вершин.

Для следующего справа соседа w корня его ветвь $G(w)$ – это почти хорошее дерево высотой h - 1. По утверждению 6 максимальное поддерево $G(w)^\wedge$ с тем же корнем и высотой h - 2 является правильным почти хорошим деревом высотой h - 2 и содержит $N(d, h - 2)$ вершин. Ветвь $G(w)$ содержит дополнительно $q(d - 1) + r$ листьев на высоте h - 1 и ведущих в них ребер. Поэтому общее число вершин в ветви $G(w)$ равно $N(d, h - 2) + q(d - 1) + r$.

Если $p + 1$ меньше степени корня, то для остальных соседей корня ветвь каждого из них имеет листья на одной высоте h - 1 в дереве G, т.е. на высоте h - 2 в ветви соседа. Следовательно, это правильные почти хорошие деревья высотой h - 2, поэтому они имеют по $N(d, h - 2)$ вершин.□

5. Алгоритмы

Линейное дерево имеет максимальный индекс Винера на классе деревьев с тем же числом вершин. По утверждению 2 любое дерево можно превратить в линейное дерево с помощью цепочки атомарных трансформаций, не нарушающих ограничения d ($d \ge 3$) на степени вершин.

По утверждению 4 хорошее дерево имеет минимальный индекс Винера на классе деревьев с тем же числом вершин и тем же ограничением на степени вершин. Утверждение 3 говорит о существовании обратной цепочки атомарных трансформаций линейного дерева, не нарушающих ограничение на степени вершин, в любое наперёд заданное дерево с тем же числом вершин и тем же ограничением на степени вершин. Тем самым, любое дерево можно превратить в хорошее дерево с помощью цепочки атомарных трансформаций, не нарушающих ограничение на степени вершин.

Эти утверждения, однако, не конструктивные: они говорят только о существовании нужной цепочки атомарных трансформаций. Нужны алгоритмы таких преобразований, по которым могли бы работать вычислительные единицы в вершинах распределённой сети.

Мы определим два алгоритма:

- *Алгоритм* $\mathcal{A}$ трансформации произвольного дерева в линейное дерево.
- *Алгоритм* $\mathcal{B}$ трансформации линейного дерева в хорошее дерево.

5.1 Алгоритм $\mathcal{A}$ – трансформация в линейное дерево

Пусть задано произвольное корневое дерево G. Если для вершины v ветвь ее сына w является линейным деревом $w = w_1, w_2, \ldots, w_k$, то путь $v, w_1, w_2, \ldots, w_k$ будем называть *линейкой*, ведущей из v. Если вершина v изолированная (и, следовательно, единственная в дереве), путь нулевой длины из вершины v тоже будем считать линейкой, ведущей из v. *Звездообразным* деревом называется корневое дерево, в котором только корень может иметь степень больше 2. Звездообразное дерево состоит из линеек, ведущих из корня.

Работу алгоритма $\mathcal{A}$ на дереве G с корнем *root* обозначим $\mathcal{A}(G, root)$. Для удобства описания алгоритма будем считать, что у корня дерева имеется фиктивное ребро с номером 0, ведущее к фиктивному отцу. Алгоритм $\mathcal{A}(G, root)$ начинает работать в корне с получения от его (фиктивного) отца сообщения *Старт*(). Алгоритм завершается посылкой из корня его (фиктивному) отцу сообщения *Линия*(n), где n – число вершин дерева.

Алгоритм выполняется рекурсивно, уровень рекурсии равен высоте вершины. На уровне рекурсии h алгоритм выполняется на каждой ветви $G(v)$, начиная с вершины v, имеющей в G высоту h. Алгоритм состоит из трёх этапов.

- Этап 1 (рис. 3). Вершина v получает сообщение *Старт* от своего отца, который фиктивен для $h = 0$ и принадлежит дереву для $h > 0$. Затем вершина v рассылает сообщение *Старт* всем своим сынам, после чего начинается этап 2.
- Этап 2 (рис. 4). Вершина v ожидает от своих сынов получения сообщений *Линия*. Когда сообщения *Линия* будут получены от всех сынов, начинается этап 3 алгоритма. В этот момент времени ветвь $G(v)$ является звездообразным деревом. Постоянная переменная *nson*(v) равна числу ожидаемых сообщений *Линия*. Она инициализируется при получении вершиной v сообщения *Старт* числом сынов вершины v. При получении вершиной v сообщения *Линия* переменная *nson*(v) уменьшается на 1.

  Этап 2 завершается, когда переменная *nson*(v) достигает нуля. Также на этапе 2 вершина v подсчитывает число вершин ветви. Для этого используется постоянная переменная *sson*(v). Она инициализируется единицей при получении вершиной v сообщения *Старт* () и увеличивается на число l вершин ветви $G(w)$ при получении от сына w сообщения *Линия* (l).

- Этап 3 (рис. 5). Пусть v^1_1, …, v^k_1 сыны вершины v. Для $i = 1 .. k$ имеется i-ая линейка $v = v^i_0$, v^i_1, …, $v^i_{x(i)}$ длиной $x(i)$. Вершина v запускает k - 1 параллельных цепочек атомарных трансформаций так, что для $i = 1 .. k$ - 1 у первого ребра i + 1-ой линейки $\{v, v^{i+1}_1\}$ один его конец v^{i+1}_1 фиксируется, а другой конец двигается по i-ой линейке от вершины v до листа $v^i_{x(i)}$. Для этого вершина v выполняет сразу k - 1 атомарных трансформаций $v^{i+1}_1 \to v \to v^i_1$, подавая команду ***Изменить*** ($e(v, v^{i+1}_1)$, $e(v, v^i_1)$, $f(i)$), в последовательности $i = k$ - 1, ..., 1. О параметре $f(i)$ мы скажем ниже.

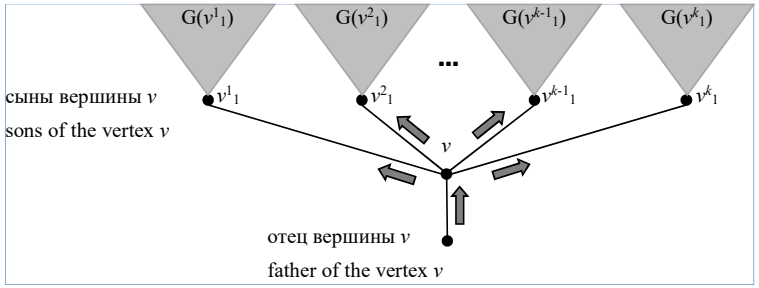

Рис. 3. Этап 1: Сообщения Старт и ветви дерева
Fig.3. Stage 1: Messages "Start" and tree branches

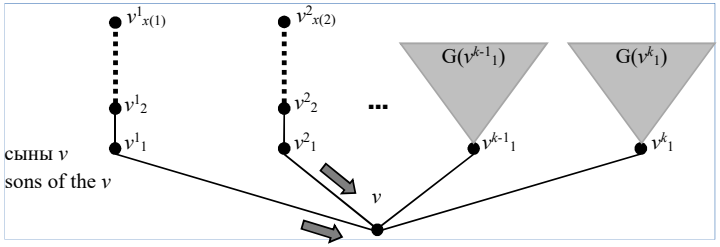

Рис. 4. Этап 2: Сообщения Линия и линейки
Fig. 4. Stage 2: Line and Messages "Line".

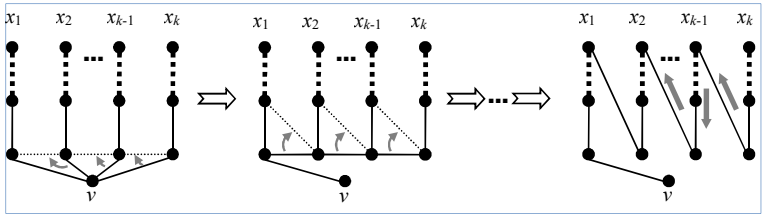

Рис. 5. Этап 3: Сообщения Финиш и трансформации
Fig. 5. Messages "Finish" and Transformations.

В результате для $i = 1 .. k$ - 1 ребро $\{v, v^{i+1}_1\}$ заменится ребром $\{v^i_1, v^{i+1}_1\}$ и по этому ребру в вершину v^i_1 придёт сообщение *Изменение* ($f(i)$). Получив это сообщение, вершина v^i_1 выполнит следующую атомарную трансформацию $v^{i+1}_1 \to v^i_1 \to v^i_2$, и так далее. Для $i = 1 .. k$ - 1 вдоль i-й линейки будет выполнена цепочка $v^{i+1}_1 \to v \to v^i_1$, $v^{i+1}_1 \to v^i_1 \to v^i_2$, $v^{i+1}_1 \to v^i_2 \to v^i_3$, …, $v^{i+1}_1 \to v^i_{x(i)-1} \to v^i_{x(i)}$ атомарных трансформаций. Отметим, что все эти цепочки атомарных трансформаций выполняются параллельно вдоль (k - 1) линеек. Цепочка атомарных трансформаций вдоль i-й линейки заканчивается в ее

листе, и в результате произойдёт конкатенация i-й и (i + 1)-й линеек. Когда это случится для всех $i = 1 .. k$ - 1 ветвь $G(v)$ станет линейным деревом, т.е. в графе G появится линейка, ведущая из отца вершины v через вершину v.

Для того, чтобы вершина v «узнала» о завершении конкатенации всех линеек, используются сообщение *Финиш* (), булевский параметр f и постоянная переменная $f(w)$ во всех вершинах w из ветви $G(v)$. Первое сообщение *Финиш* посылается, когда заканчивается конкатенация k-й и (k - 1)-й линеек. Для того чтобы отличить эту конкатенацию от всех остальных, используется параметр f команды ***Изменить*** и сообщения *Изменение*, который равен ***true*** для атомарных трансформаций при конкатенации k-й и (k - 1)-й линеек и равен ***false*** для атомарных трансформаций при остальных конкатенациях. Сообщение *Финиш* пересылается от сына к отцу, двигаясь по (k - 1)-й линейке через вершины $v^{k-1}_{x(k-1)}$, …, v^{k-1}_1 и далее, если к этому моменту времени уже завершена конкатенация (k - 1)-й и (k - 2)-й линеек, (k - 2)-й и (k - 3)-й линеек и т.д.

Если же конкатенации i-й и (i + 1)-й линеек ещё не завершена, сообщение *Финиш* может быть послано в вершину v^i_j из вершины v^{i+1}_1, где $j < x(i)$, тогда, когда ребро $\{v^i_j, v^{i+1}_1\}$ появилось в результате некоторой атомарной трансформации и, следовательно, по ребру послано сообщение *Изменение*, но это сообщение ещё не дошло до вершины v^i_j. Когда вершина v^i_j получит сообщение *Изменение*, она начнёт трансформацию $v^i_j \to v^{i+1}_1 \to v^i_{j+1}$, меняющую ребро $\{v^i_j, v^{i+1}_1\}$ на ребро $\{v^i_{j+1}, v^{i+1}_1\}$. По ребру $\{v^i_{j+1}, v^{i+1}_1\}$ будет послано сообщение *Изменение*, но оно придёт в вершину v^i_{j+1} после сообщения *Финиш*, поскольку сообщения, посланные по ребру, не обгоняют друг друга. Тем самым, вершина v^i_{j+1} получит сообщение *Финиш* по ребру, номера которого ещё нет в множестве $E(v^i_{j+1})$. В этом случае вершина v^i_{j+1} ожидает последующего сообщения *Изменение* для того, чтобы начать следующую атомарную трансформацию, если $j + 1 < x(i)$, или послать сообщение *Финиш* дальше уже по i-й линейке, если $j + 1 = x(i)$. Иными словами, вершина v^i_{j+1} должна «помнить», получила ли она сообщение *Финиш*. Для этого используется постоянная переменная $f(v^i_{j+1})$. Если вершина v^i_{j+1} выполняет следующую атомарную трансформацию, то она использует параметр $f = f(v^i_{j+1})$ в команде ***Изменить*** для того, чтобы следующая вершина v^i_{j+2}, получив сообщение *Изменение* с этим параметром, тоже «узнала» о приходе сообщения *Финиш*. В целом параметр f = ***true*** в команде ***Изменить*** и сообщении *Изменение* показывает, что, когда сообщение пересылается из вершины b в вершину a, имеется линейка a, b, …

Когда все линейки склеются в одну и вершина v получит сообщение *Финиш*, она пошлёт своему отцу сообщение *Линия* ($sson(v)$), тем самым завершая работу алгоритма на ветви $G(v)$.

Определим алгоритм $\mathcal{A}$ (G, *root*) формально.

- Постоянные переменные вершины v в алгоритме $\mathcal{A}$ (G, *root*):
  - $E(v)$ – множество номеров рёбер, инцидентных вершине v;
  - $fat(v)$ – номер в вершине v ребра, ведущего из вершины v к её отцу;
  - $line(v)$ – признак (***Bool***) того, что вершина v послала сообщение *Линия*;
  - $nson(v)$ – число ожидаемых вершиной v сообщений *Линия* от ее сынов;
  - $sson(v)$ – сумма длин линеек от вершины v в полученных сообщениях *Линия*;
  - $f(v)$ – признак (***Bool***) получения *Финиш* раньше *Изменение*.
- Обозначения: $element(S)$ – функция выбора произвольного элемента из непустого множества S.
- Сообщения в алгоритме $\mathcal{A}$(G, *root*): *Старт*(); *Линия*(l), где l – длина линейки; *Изменение*(f), где f – признак (***Bool***) того, что сообщение передаётся в вершину v по ребру, с которого начинается линейка из вершины v; *Финиш*().
- Предусловие алгоритма $\mathcal{A}$ (G, *root*): В дереве G в каждой вершине v переменная $E(v)$ инициализирована корректно, т.е. множеством номеров рёбер, инцидентных вершине v.

Одна из вершин – корень *root*, в корень по фиктивному ребру с номером 0 приходит сообщение *Старт*().

- Постусловие алгоритма $\mathcal{A}$ (*G*, *root*): Дерево *G* линейное, множество вершин сохранено, вершина *root* – лист или единственная изолированная вершина. В каждой вершине *v* значение переменной *E*(*v*) корректно. Из корня *root* по фиктивному ребру с номером 0 послано сообщение *Линия*(*n*), где *n* число вершин дерева.

Алгоритм $\mathcal{A}$ (*G*, *root*). Вершина *v*.

```
1  while не конец алгоритма {
2  wait () { /* приём сообщения */
3     if приём Старт() по ребру i { /* этап 1 */
4        fat(v) := i; line(v) := false;  /* инициализация */
5        nson(v) := | E(v) \ { fat(v) } |; sson(v) := 1; f(v) := 0;
6        посылка Старт() по каждому ребру j ∈ E(v) \ { fat(v) };
7        if nson(v) = 0 { /* v – лист или изолированная вершина */
8           посылка Линия(sson(v)) по ребру fat(v); line(v) := true;
9           if fat(v) = 0 { конец алгоритма; } /*v – изолированная вершина*/
10    } }
11    if приём Линия(l) по ребру i { /* этап 2 */
12       nson(v) := nson(v) - 1; sson(v) := sson(v) + l; /* новая линейка
         из v */
13       if E(v) \ { fat(v) } = { i } { /* у вершины v один сын */
14          посылка Линия(sson(v)) по ребру fat(v); line(v) := true;
15          if fat(v) = 0 { конец алгоритма; }
16       }
17       else { /* v – развилка или корень степени 2 */
18          if nson(v) = 0 { /*начало этапа 3: G(v) звездообразное дерево */
19             j := element( E(v) \ { fat(v), i } ); /* выбор ребра */
20             /* трансформация a→v→b, где e(v, a) = i, e(v, b) = j */
21             команда Изменить(i, j, true);
22             E(v) = E(v) \ { i }; /* ребро удалено */
23             while | E(v) \ { fat(v) } | > 1 { /* цикл трансформаций */
24                m := j;
25                j := element( E(v) \ { fat(v), m } ); /* выбор ребра */
26                /* трансформация a→v→b, где e(v, a) = m, e(v, b) = j */
27                команда Изменить(m, j, false);
28                E(v) = E(v) \ { m }; /* ребро удалено */
29    } } } }
30    if приём Изменение(f) по ребру i { /* этап 3: трансформация a→c→v */
31       f(v) := f(v) ∨ f; /* f(v) = true, если имеется линейка v, a, … */
32       if | E(v) | = 2 { /*цепочка трансформаций по линейке не закончена*/
33          j := element(E(v) \ { fat(v) }; /* выбор следующего ребра */
34          /* трансформация a→v→b, где e(v, a) = i, e(v, b) = j */
35          команда Изменить(i, j, f(v)); f(v) := false;
36       }
37       else { /*| E(v) | = 1, цепочка трансформаций по линейке закончена*/
38          E(v) = E(v) ∪ { i }; /* ребро добавлено */
39          if f(v) { /* произошла конкатенация двух линеек */
40             посылка Финиш() по ребру fat(v); f(v) := false;
41    } } }
42    if приём Финиш() по ребру i { /* этап 3 */
43       if i ∉ E(v) { f(v) := true; } /*Финиш пришёл в v раньше Изменение*/
44       else {
```



```
45          if  line(v) { /* вершина v посылала Линия */
46             посылка Финиш() по ребру fat(v);
47          }
48          else { /* вершина v не посылала Линия */
49             посылка Линия(sson(v)) по ребру fat(v); line(v) := true;
50             if fat(v) = 0 { конец алгоритма; }
51 }} }  }  }
```

□

Утверждение 9. Алгоритм $\mathcal{A}$ на классе корневых деревьев с *n* вершинами не нарушает ограничения *d* (*d* ≥ 3) на степень вершины и заканчивает работу с выполнением постусловия алгоритма за время $t(n) \le 2n - 2$.

Доказательство. Пусть дерево *G* имеет *n* вершин. Доказательство будем вести индукцией по числу *n*. Напомним, что время вычислений в вершине не учитывается, и время выполнения алгоритма слагается из времени последовательных пересылок сообщений, включая сообщение *Изменение* при атомарной трансформации.

Для *n* = 1 дерево *G* состоит из одной изолированной вершины *root*. Время работы алгоритма без учёта пересылок сообщений *Старт* и *Линия* по фиктивному ребру с номером 0, очевидно, равно нулю: $t(1) = 0 = 2*1 - 2$. Постусловие алгоритма выполнено: дерево *G* линейное, *root* – изолированная вершина, $E(v) = \varnothing$, по фиктивному ребру с номером 0 послано сообщение *Линия*(1).

Пусть утверждение верно для любого числа вершин не больше *n*, где $n \ge 1$, и докажем его для числа вершин $n + 1$. Пусть у корня *k* сынов и числа вершин в ветвях сынов равны $x_1, \ldots, x_k$. Имеем $x_1 + \ldots + x_k = n$. Сначала покажем, что алгоритм работает время не более $2(n + 1) - 2 = 2n$.

Этап 1 рассылок сообщений *Старт* занимает время $t_1(n + 1) \le 1$, поскольку сообщения рассылаются одновременно.

На этапе 2 алгоритм выполняется параллельно на ветвях всех сынов корня. Поэтому этап 2 занимает время $t_2(n + 1)$, слагаемое из максимума от времён выполнения алгоритма на ветвях сынов и времени пересылки от сына корню сообщения *Линия*. Пусть $t(x_1) \le \ldots \le t(x_k)$. Имеем $t_2(n + 1) = max\{ t(x_1), \ldots, t(x_k) \} + 1 \le t(x_k) + 1$. Поскольку $x_k \le n$, по предположению шага индукции $t(x_k) \le 2x_k - 2$. Поэтому $t_2(n + 1) \le 2x_k - 2 + 1 = 2x_k - 1$.

По предположению шага индукции каждую из ветвей сынов корня алгоритм трансформирует в линейное дерево, которое в дереве *G* будет линейкой. Этап 3 трансформации звездообразного дерева в линейное занимает время $t_3(n + 1)$, слагаемое из времени выполнения всех атомарных трансформаций и времени пересылки сообщения *Финиш* до корня. Атомарные трансформации вдоль разных линеек выполняются параллельно. Поэтому время выполнения трансформаций равно максимуму из времён выполнения трансформаций вдоль одной линейки, а это последнее время не превышает длины линейки. Сообщение *Финиш* проходит по всем линейкам, кроме последней *k*-й линейки, появившейся позже других. Тем самым, $t_3(n + 1) \le max\{x_1, \ldots, x_{k-1}\} + (x_1 + \ldots + x_{k-1})$. Пусть $max\{x_1, \ldots, x_{k-1}\} = x_m$. Тогда $t_3(n + 1) \le x_m + (n - x_k)$.

Суммарно имеем:

$t(n + 1) = t_1(n + 1) + t_2(n + 1) + t_3(n + 1) \le\le 1 + 2x_k - 1 + x_m + (n - x_k) = n + (x_m + x_k) \le n + (x_1 + \ldots + x_k) = 2n$,

что и требовалось доказать.

Докажем выполнение постусловия работы алгоритма. По предположению шага индукции на этапе 2 каждая из ветвей сынов корня трансформируется в линейное дерево. После этого в результате трансформаций на этапе 3 звездообразное дерево трансформируется в линейное дерево. Переменная *sson*(*v*) инициализируется единицей на этапе 1. По предположению шага индукции на этапе 2 от каждого своего сына *w* вершина *v* получит сообщение *Линия*(*l*), где *l*

– число вершин ветви $G(w)$. Сумма этих чисел, очевидно, на 1 меньше числа вершин ветви $G(v)$. На этапе 2 все эти величины суммируются в переменной $sson(v)$, которая в конце этапа 2 будет равна числу вершин ветви $G(v)$. Для корня $root$ число $sson(root)$ равно числу вершин дерева, именно это значение будет в конце работы алгоритма указано как параметр сообщения *Линия*, которое корень пошлёт по фиктивному ребру с номером 0.

Этот алгоритм не нарушает ограничения $d \ge 3$ на степень вершин, поскольку 1) каждая атомарная трансформация $a \to c \to b$ происходит вдоль линейки …, c, b, … в сторону листа, т.е. вершина b имеет степень 2 или 1, а трансформация увеличивает её степень на 1, 2) на каждом уровне рекурсии не происходят никакие две трансформации $a \to c \to b$ и $a' \to c' \to b$ с общей конечной вершиной b. Более того, на этапе 3 на каждой ветви $G(v)$ первые трансформации в цепочках атомарных трансформаций по линейкам сразу уменьшают степень вершины v до 2, если $v \ne root$, или 1, если $v = root$. После этого степень каждой вершины ветви не больше 3 до конца трансформаций на этой ветви, когда она становится не больше 2. □

Оценка времени $2n - 2$ достигается на дереве, которое уже является линейным, а его корень – один из листьев. Для этого случая эта оценка не улучшаема, если алгоритм должен закончиться в корне. Это объясняется тем, что для того, чтобы выяснить, что дерево уже линейное, сообщение должно дойти из корня до другого листа и вернуться обратно, то есть пройти путь длиной $2n - 2$.

5.2 Алгоритм $\mathcal{B}$ – трансформация линейного дерева в хорошее дерево

Работу алгоритма $\mathcal{B}$ на линейном дереве G с корнем $root$ обозначим $\mathcal{B}(G, root)$. Для своей работы алгоритму $\mathcal{B}(G, root)$ нужно «знать» число n вершин дерева и ограничение d ($d \ge 3$) на степени вершин. Выполнение алгоритма $\mathcal{B}(G, root)$ запускается сообщением *Начало* (d, n), которое приходит в корень $root$ по фиктивному ребру с номером 0 в корне. Алгоритм завершается посылкой из корня по (фиктивному) ребру с номером 0 сообщения *Конец* ().

Алгоритм выполняется рекурсивно, уровень рекурсии равен высоте вершины в итоговом хорошем дереве, которое должно быть построено в конце работы алгоритма. Предполагается, что на уровне рекурсии h построена часть хорошего дерева на высотах от 0 до h. Вначале $h = 0$ и построенная часть хорошего дерева состоит только из корня. На уровне h алгоритм выполняется на каждой ветви $G(v)$, где вершина v имеет в G высоту h. Алгоритм состоит из двух этапов.

(*Почти*) *хорошим звездообразным* деревом с числом вершин k будем называть звездообразное дерево, у которого степень корня и числа вершин на ветвях соседей корня такие же, как у (почти) хорошего дерева с числом вершин k.

- Этап 1. В начале этого этапа на уровне рекурсии h ветвь $G(v)$, где вершина v имеет в G высоту h, является линейкой ведущей из v. Задача этого этапа – построить звездообразное дерево с корнем в вершине v. Это дерево должно быть хорошим звездообразным деревом, если $v = root$ ($h = 0$), или почти хорошим звездообразным деревом, если $v \ne root$ ($h > 0$). Напомним, что по утверждению 7 в (почти) хорошем дереве ветвь соседа корня является почти хорошим деревом, а по утверждению 5 (почти) хорошее дерево однозначно определяется с точностью до изоморфизма, сохраняющего корень, числом его вершин. Тем самым, звездообразное дерево, в которое нужно превратить линейку ветви $G(v)$, однозначно с точностью до изоморфизма, сохраняющего корень, определяется числом вершин в этой ветви и признаком $v = root$ или $v \ne root$, соответствующим хорошему и почти хорошему деревьям. Число вершин ветви $G(v)$ вычисляется по утверждению 8 и является параметром сообщения *Начало*, с получения которого начинается этап 1 на ветви $G(v)$.
- Этап 2. На этом этапе на уровне рекурсии h ветвь $G(v)$, где вершина v имеет в G высоту h, является хорошим (если $v = root$) или почти хорошим (если $v \ne root$) звездообразным деревом, а из каждого сына w вершины v через ребро, отличное от ребра, ведущего к вершине v, ведет линейка. Вершина v посылает каждому своему сыну w сообщение *Начало* (d, l), где l – число вершин на ветви $G(w)$, инициируя работу алгоритма на следующем уровне рекурсии $h + 1$. Очевидно, это можно делать, не дожидаясь, когда этап 1 полностью завершён, то есть полностью построено звездообразное дерево с корнем в вершине v, а как только построена линейка нужной длины, ведущая из вершины w через ребро, отличное от ребра, ведущего к вершине v.

Вершина v ожидает от своих сынов сообщений *Конец* (). Постоянная переменная $nson(v)$ равна числу ожидаемых сообщений *Конец*. Она инициализируется при получении вершиной v сообщения *Начало* и равна числу сынов вершины v в звездообразном дереве, которое должно быть построено. Если l – число вершин звездообразного дерева, то $nson(v) = min\{d, l - 1\}$, если $v = root$, или $nson(v) = min\{d - 1, l - 1\}$, если $v \ne root$. При получении вершиной v сообщения *Конец* переменная $nson(v)$ уменьшается на 1. Этап 2 завершается, когда переменная $nson(v)$ достигает нуля. Вершина v посылает своему отцу сообщение *Конец* (). Если $v = root$, алгоритм заканчивается.

Оптимизация: заметим, что если число вершин ветви меньше 3, то ветвь уже является хорошим и почти хорошим деревом, и запуск алгоритма на ветви не требуется.

Рассмотрим подробнее построение звездообразного дерева на этапе 1. Мы будем использовать понятие *текущей вершины* и две операции: *перемещение* и *трансформация*. В начале этапа 1 на уровне рекурсии h на ветви $G(v)$, где вершина v имеет в G высоту h, текущей вершиной становится вершина v при получении ею сообщения *Начало*. Если $v = root$, строится хорошее звездообразное дерево, иначе строится почти хорошее звездообразное дерево.

Две операции: Параметр t означает число трансформаций, которые осталось сделать для построения линейки звездообразного дерева.

Перемещение $c \dashrightarrow b$. Предусловие: c текущая вершина, есть ребро $\{c, b\}$. Вершина c посылает в вершину b сообщение *Перемещение*(t). После получения вершиной b этого сообщения текущей вершиной становится вершина b.

Трансформация $a \to c \to b$. Предусловие: c текущая вершина, есть рёбра $\{a, c\}$ и $\{c, b\}$. Величина t уменьшается на 1: $t := t - 1$. Вершина c подаёт команду ***Изменить***$(e(c, a), e(c, b), t)$. После получения вершиной b сообщения *Изменение*(t) текущей вершиной становится вершина b.

Сначала опишем построение хорошего звездообразного дерева на ветви $G(v)$ как последовательность этих операций, см. рис. 6. На этом рисунке серыми стрелками показаны текущие вершины, белыми кружками – вершина v, а чёрными кружками – остальные вершины.

Пусть l число вершин на ветви $G(v)$. Вначале имеется линейка $v = v_1, v_2, \ldots, v_l$, текущая вершина v.

Обозначим:

$x = min\{d, l - 1\}$ – степень вершины v в хорошем звездообразном дереве;

$SM(d, l, 0) = 1$ – число вершин в дереве из одной вершины;

$SM(d, l, j) = 1 + M(d, l, 1) + M(d, l, 2) + \ldots + M(d, l, j)$, для $j = 1 \,..\, x$, – суммарное число вершин в хорошем звездообразном дереве на первых j линейках плюс единица, соответствующая корню дерева;

$v^j_i = v_{SM(d, l, j - 1) + i}$, для $j = 1 \,..\, x - 1$ и $i = 1 \,..\, M(d, l, j)$, – i-я вершина j-й линейки;

$v^x_i = v_{SM(d, l, x) - i + 1}$, для $i = 1 \,..\, M(d, l, x)$, – i-я вершина x-й линейки.

Строим линейки «справа налево», начиная от x-й и заканчивая 2-й.

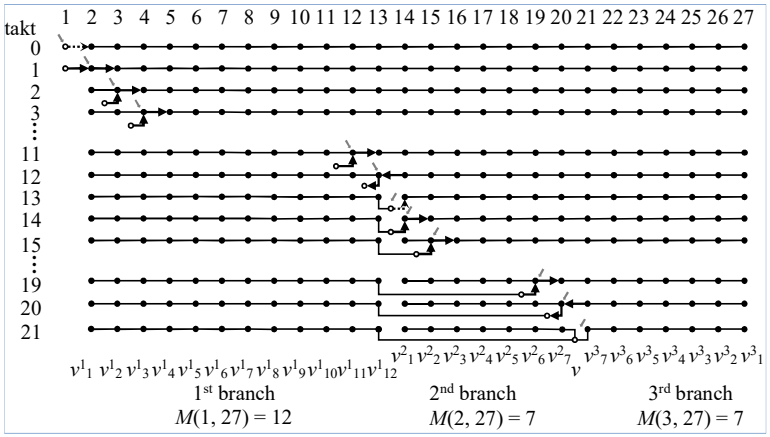

Рис. 6. Построение хорошего звездообразного дерева для n = 27 и d = 3
Fig. 6. Construction of a good starlike tree for n = 27 and d = 3

Построение j-й линейки хорошего звездообразного дерева для $j = x$ .. 2:

1. $t = M(d, l, j)$.
2. Одно перемещение $v \dashrightarrow v^j{}_1$.
3. Цепочка $M(d, l, j)$ - 1 трансформаций:
   $v \to v^j{}_1 \to v^j{}_2$, $v \to v^j{}_2 \to v^j{}_3$, …, $v \to v^j{}_{M(d,\,l,\,j)\,-\,1} \to v^j{}_{M(d,\,l,\,j)}$;
   $t = M(d, l, j)$ - 1, $t = M(d, l, j)$ - 2, …, $t = 1$.
4. Одна завершающая $M(d, l, j)$-я трансформация: $v^{j+1}{}_1 \to v^j{}_{M(d,\,l,\,j)} \to v$.
5. Если $M(d, l, j) > 2$, запуск алгоритма построения почти хорошего звездообразного дерева на j-й ветви: вершина v посылает вершине $v^j{}_{M(d,\,l,\,j)}$, сообщение *Начало* (d, $M(d, l, j)$).
   После построения 2-й линейки также построена и 1-я линейка. Если $M(d, l, 1) > 2$, запуск алгоритма построения почти хорошего звездообразного дерева на 1-й линейке: вершина v посылает вершине $v^1{}_{M(d,\,l,\,1)}$, сообщение *Начало* (d, $M(d, l, 1)$).

Построение почти хорошего звездообразного дерева с l вершинами отличается от алгоритма построения хорошего звездообразного дерева с l вершинами только тем, что число строящихся линеек x равно не $min\{d, l - 1\}$, а $min\{d - 1, l - 1\}$, и тем, что число вершин в j-й линейке равно не $M(d, l, j)$, а $N(d, l, j)$.

Определим алгоритм $\mathcal{B}(G, root)$ формально.

- Постоянные переменные вершины v в алгоритме $\mathcal{B}(G, root)$:
  - $E(v)$ – множество номеров рёбер, инцидентных вершине v;
  - $fat(v)$ – номер в вершине v ребра, ведущего из вершины v к её отцу;
  - $sson(v)$ – число вершин в ветви $G(v)$;
  - $d(v)$ – ограничение d на степени вершин;
  - $j(v)$ – номер строящейся линейки звездообразного дерева;
  - $first(v)$ – номер первого ребра строящейся линейки звездообразного дерева;
  - $nson(v)$ – число ожидаемых вершиной v сообщений *Конец*.
- Сообщения в алгоритме $\mathcal{B}(G, root)$: *Начало* (d, l), *Конец* (), *Перемещение* (t), *Изменение* (t), где d – ограничение на степень вершин, l – число вершин ветви, t число трансформаций, которые осталось сделать для построения линейки звездообразного дерева.

- Предусловие алгоритма $\mathcal{B}(G, root)$: Дерево G линейное с n вершинами и корнем $root$ в одном из двух листьев. В каждой вершине v переменная $E(v)$ инициализирована корректно, т.е. множеством номеров рёбер, инцидентных вершине v. В корень по фиктивному ребру с номером 0 приходит сообщение *Начало* (d, n).
- Постусловие алгоритма $\mathcal{B}(G, root)$: Дерево G хорошее с корнем в вершине $root$. В каждой вершине v значение переменной $E(v)$ корректно. Из корня по фиктивному ребру с номером 0 послано сообщение *Конец* ().

Алгоритм $\mathcal{B}$ (G, $root$). Вершина v.

```
1  while не конец алгоритма {
2  wait () { /* приём сообщения */
3    if приём Начало(d, l) по ребру i {
4      d(v) := d; sson(v) := l; fat(v) := i; /* инициализация */
5      if fat(v) = 0 { nson(v) := min{d, sson(v) - 1}; }
6      else { nson(v) := min{d - 1, sson(v) - 1}; }
7      j(v) := nson(v); /* начинаем с последней линейки меньшей длины */
8      if sson(v) ≥ 3 { /* число вершин ≥ 3, нужно делать трансформацию
       */
9        if fat(v) = 0 { t := M(d, sson(v), j(v)); } else { t := N(d, s
         son(v), j(v)); }
10       first(v) := element( E(v) \ { i } ); /* выбор ребра */
11       посылка Перемещение(t) по ребру first(v);
12     }
13     else { /* число вершин ≤ 2, не нужно делать трансформацию */
14       посылка Конец() по ребру fat(v);
15       if fat(v) = 0 { конец алгоритма; } /* v = root */
16   }        }
17   if приём Перемещение(t) по ребру i {
18     e := element( E(v) \ { i } ); /* выбор ребра */
19     if t > 1 { /* первая трансформация цепочки трансформаций */
20       /* трансформация a→v→b, где e(v, a) = i, e(v, b) = e */
21       команда Изменить(i, e, t);
22       E(v) := E(v) \ { i }; /* ребро удалено */
23     }
24     else { /* завершающая трансформация */
25       /* трансформация a→v→b, где e(v, a) = e, e(v, b) = i */
26       команда Изменить(e, i, t);
27       E(v) := E(v) \ { e }; /* ребро удалено */
28   }        }
29   if приём Изменение(t) по ребру i {
30     if t > 2 { /* продолжаем цепочку трансформаций */
31       e := element( E(v) \ { fat(v), i } ); /* выбор ребра */
32       /* трансформация a→v→b, где e(v, a) = i, e(v, b) = e */
33       команда Изменить(i, e, t - 1);
34     }
35     if t = 2 { завершающая трансформация */
36       e := element( E(v) \ { fat(v), i } ); /* выбор ребра */
37       /* трансформация a→v→b, где e(v, a) = e, e(v, b) = i */
38       команда Изменить(e, i, t - 1);
39     }
40     if t = 1 { все трансформации закончены */
41       E(v) := E(v) ∪ { i }; /* ребро добавлено */
42       /* запуск алгоритма на построенной линейке */
```

```
43          if fat(v) = 0 { t := M(d, sson(v), j(v)); } else { t := N(d, s
            son(v), j(v)); }
44          if t ≥ 3 { посылка Начало(d(v), t) по ребру first(v); }
45          else { nson(v) := nson(v) - 1; }
46          j(v) := j(v) - 1; /* переходим к следующей линейке */
47          if fat(v) = 0 { t := M(d, sson(v), j(v)); } else { t := N(d, s
            son(v), j(v)); }
48          if j(v) > 1 { /* начинаем строить следующую линейку */
49             first(v) := i; посылка Перемещение(t) по ребру first(v);
50          }
51          if j(v) = 1 { /* все линейки построены */
52             /* запуск алгоритма на последней линейке */
53             if t ≥ 3 { посылка Начало(d(v), t) по ребру i; }
54             else { nson(v) := nson(v) - 1; }
55    }          }          }
56    if приём Конец() по ребру i {
57       nson(v) := nson(v) - 1;
58       if nson(v) = 0 { /* ветвь построена */
59          посылка Конец() по ребру fat(v);
60          if fat(v) = 0 { конец алгоритма; } /* v = root */
61 }}    }          }□
```

Утверждение 10. Алгоритм $\mathcal{B}$ на классе линейных деревьев с n вершинами с корнем в одном из листов не нарушает ограничения d ($d \ge 3$) на степень вершины и заканчивает работу с выполнением постусловия алгоритма за время $t(n) \le 2n - 2$.

Доказательство. Мы будем доказывать, что для линейки $G(v)$, ведущей из вершины v и содержащей n вершин, алгоритм после получения вершиной v сообщения *Начало* по ребру $fat(v)$ строит хорошее дерево, если $fat(v) = 0$, или почти хорошее дерево, если $fat(v) \ne 0$. В обоих случаях не нарушается ограничение d на степени вершин, корнем построенного дерева является вершина v, в каждой вершине w значение переменной $E(w)$ корректно, в конце работы алгоритма из вершины v по ребру $fat(v)$ посылается сообщение *Конец* (), и время работы алгоритма не превышает $2n - 2$. Доказательство будем вести индукцией по числу n. Напомним, что время вычислений в вершине не учитывается, и время выполнения алгоритма слагается из времени последовательных пересылок сообщений, включая сообщение *Изменение* при атомарной трансформации.

Для $n = 1$ дерево $G(v)$ состоит из одной изолированной вершины v и является как почти хорошим, так и хорошим деревом. Алгоритм не меняет дерево, $E(v) = \varnothing$, из корня по ребру с номером $fat(v)$ послано сообщение *Конец*. Время работы алгоритма без учёта пересылок сообщений *Начало* и *Конец*, очевидно, равно нулю: $t(1) = 0 = 2*1 - 2$.

Пусть утверждение верно для любого числа вершин не больше n, где $n \ge 1$, и докажем его для числа вершин $n + 1$. Сначала покажем, что алгоритм работает время не более $2(n + 1) - 2 = 2n$.

Пусть x – число соседей корня (почти) хорошего дерева с числом вершин n, а числа вершин ветвей соседей корня равны $l_1 \le \ldots \le l_x$. Очевидно, $1 \le l_1$ и $n = l_1 + \ldots + l_x$. Тогда $n \ge (x - 1) + l_x$, что влечёт $l_x \le n - x + 1$. Время $t_1(l_j)$ построения j-й линейки (почти) хорошего звездообразного дерева, содержащей l_j вершин, для $j = x \,..\, 2$, состоит из времени одного перемещения и l_j трансформаций. Поэтому $t_1(l_j) \le 1 + l_j$, а общее время этапа 1 равно $t_1(n + 1) \le (1 + l_1) + \ldots + (1 + l_{x-1}) = x - 1 + (n - l_x)$. На этапе 2 алгоритм трансформирует каждую линейку (почти) хорошего звездообразного дерева в почти хорошее дерево, начиная с пересылки сообщения *Начало* из вершины v в первую вершину линейки, соседней с корнем, и заканчивая пересылкой сообщения *Конец* в обратном направлении. На j-ую линейку тратится время $t_2(l_j) \le t(l_j) + 2$. Алгоритм выполняется на всех линейках параллельно, поэтому

общее время этапа 2 равно $t_2(n + 1) = max\{\, t_2(l_j) \mid j = 1 \,..\, x \,\} \le \le max\{\, t(l_j) + 2 \mid j = 1 \,..\, x \,\}$. Так как $l_j \le n$, по предположению шага индукции $t_2(n + 1) \le max\{\, 2l_j - 2 + 2 \mid j = 1 \,..\, x \,\} = 2l_x$. Тем самым, $t(n + 1) = = t_1(n + 1) + t_2(n + 1) \le x - 1 + (n - l_x) + 2l_x = x + n + l_x - 1$. Так как $l_x \le n - x + 1$, имеем $t(n + 1) \le x + n + (n - x + 1) - 1 = 2n$, что и требовалось доказать.

Докажем выполнение остальных условий. На этапе 1 строится хорошее звездообразное дерево, если $fat(v) = 0$, или почти хорошее звездообразное дерево, если $fat(v) \ne 0$. На этапе 2 по предположению шага индукции каждая из линеек этого дерева трансформируется в почти хорошее дерево. По определению (почти) хорошего звездообразного дерева, а также по утверждениями 7 и 5 на этапе 2 строится хорошее дерево, если $fat(v) = 0$, или почти хорошее дерево, если $fat(v) \ne 0$.

На этапе 1 ограничение d на степени вершин не нарушается, так как степень вершины v увеличивается ровно до того значения, которое она должна иметь в (почти) хорошем звездообразном дереве, а степени остальных вершин увеличиваются не более чем до 3. На этапе 2 ограничение d на степени вершин не нарушается по предположению шага индукции.

На этапе 1 корнем строящегося звездообразного дерева остаётся вершина v и она остаётся корнем строящегося на этапе 2 (почти) хорошего дерева. Значение переменных $E(w)$ поддерживается корректным в процессе всей работы алгоритма: при удалении (добавлении) ребра его номер удаляется из (добавляется в) $E(w)$. Алгоритм заканчивается посылкой сообщения *Конец*() из вершины v по ребру $fat(v)$.□

6. Заключение

В статье предложены два алгоритма самотрансформации дерева, лежащего в основе распределенной сети, с помощью локальных атомарных трансформаций, выполняемых по «командам» от узлов дерева. Это алгоритм трансформации любого дерева в линейное дерево и алгоритм трансформации линейного дерева в хорошее дерево, имеющее минимальный индекс Винера при том же числе вершин. Трансформации не меняют множество вершин дерева и не нарушают ограничение d ($d \ge 3$) на степени вершин.

Оба алгоритма имеют верхнюю оценку времени работы $2n - 2$, где n число вершин дерева. Однако, если для алгоритма $\mathcal{A}$ трансформации в линейное дерево эта оценка достижима, то для алгоритма $\mathcal{B}$ трансформации в хорошее дерево это не так. Это объясняется тем, что для алгоритма $\mathcal{A}$ оценка достигается на линейном дереве, а хорошее дерево алгоритм $\mathcal{A}$ трансформирует в линейное за меньшее время. Соответственно, и «обратный» алгоритм $\mathcal{B}$ трансформирует линейное дерево в хорошее меньше чем за $2n - 2$ тактов. Поэтому верхняя оценка для алгоритма $\mathcal{B}$ нуждается в уточнении (в сторону уменьшения).

Для минимизации индекса Винера дальнейшие исследования могут быть связаны с расширением класса графов, когда допускаются циклы. Предложенный подход самотрансформации графов, основанный на атомарной трансформации, может быть использован не только для максимизации или минимизации индекса Винера, но и для достижения других целей, когда используются другие критерии «оптимальности» графа. Кроме того, могут использоваться атомарные трансформации других типов, позволяющие удалять и добавлять вершины. Наконец, дополнительные проблемы возникают для ориентированных и смешанных графов, в которых сообщения можно пересылать по ориентированным ребрам только в направлении их ориентации.

Список литературы / References

[1]. Wiener H. Structural determination of paraffin boiling points. Journal of the American Chemical Society, vol. 69, no. 1, 1947, pp. 17–20.

[2]. Miranca Fischerman, Arne Hoffmann, Dieter Rautenbach, László Székely, Lutz Volkmann. Wiener index versus maximum degree in trees. Discrete Applied Mathematics, volume 122, issues 1–3, 2002, pp. 127–137.
[3]. L. A. Székely and Hua Wang. On Subtrees of Trees. Advances in Applied Mathematics, vol. 34, issue 1, 2005, pp. 138–155.
[4]. А.А. Кочкаров, Л.И. Сенникова, Р.А. Кочкаров. Некоторые особенности применения динамических графов для конструирования алгоритмов взаимодействия подвижных абонентов. Известия ЮФУ. Технические науки», № 1, 2015, стр. 207–214 / A.A. Kochkarov, L.I. Sennikova, R.A. Kochkarov. Some features of dynamic graphs applications for construction of mobile agents interaction algorithms. Izvestiya SFedU. Engineering sciences, № 1, 2015, pp. 207–214 (in Russian).
[5]. A.A. Kochkarov, R.A. Kochkarov, G.G. Malinetskii. Issues of dynamic graph theory. Computational Mathematics and Mathematical Physics, 2015, vol. 55, no. 9, pp. 1590–1596.
[6]. А.В. Проскочило, А.В. Воробьев, М.С. Зряхов, А.С. Кравчук. Анализ состояния и перспективы развития самоорганизующихся сетей. Научные ведомости Белгородского государственного университета, Экономика, Информатика, no. 36/1, вып. 19 (216), 2015, стр. 177-186 / A.V. Proskotchylo, A.V. Vorobyov, M.S. Zriakhov, A.S. Kravchuk. Analysis of state and development perspectives of self-organizing networks. Belgorod State University Scientific Bulletin. Economics, Information technologies, no. 19 (216), issue 36/1, 2015, pp. 177-186 (in Russian).
[7]. Zhi Chen, Shuai Li, and Wenjing Yue. SOFM Neural Network Based Hierarchical Topology Control for Wireless Sensor Networks. Journal of Sensors, vol. 2014, Article ID 121278, 6 p.
[8]. Chen Dongning, Zhang Ruixing, Yao Chengyu, and Zhao Zheyu. Dynamic topology multi force particle swarm optimization algorithm and its application. Chinese Journal of Mechanical Engineering, vol. 29, issue 1, 2016, pp. 124–135.
[9]. Simin Mo, Jian-Chao Zeng, Ying Tan. Particle Swarm Optimization Based on Self-organizing Topology Driven by Fitness. In Proc. of the International Conference on Computational Aspects of Social Networks, 2010, pp. 23–26.
[10]. Al-Sakib Khan Pathan (ed.) Security of self-organizing networks: MANET, WSN, WMN, VANET. CRC press, 2010, 638 p.
[11]. Boukerche A. (ed.) Algorithms and protocols for wireless, mobile Ad Hoc networks. John Wiley & Sons, 2008, 496 p.
[12]. Wen Chih-Yu and Hung-Kai Tang. Autonomous distributed self-organization for mobile wireless sensor networks. Sensors, vol. 9, issue 11, 2009, pp. 8961-8995.
[13]. Jaime Llorca, Stuart D. Milner, Christopher Davis. Molecular System Dynamics for Self-Organization in Heterogeneous Wireless Networks. EURASIP Journal on Wireless Communications and Networking, 2010, Article number: 548016.
[14]. Rangaswami Balakrishnan and S. Francis Raj. The Wiener number of powers of the Mycielskian. Discussiones Mathematicae Graph Theory, vol. 30, no. 3, 2010, p. 489–498.
[15]. Chen Wai-kai. Net Theory and its Applications: Flows in Networks. World Scientific Publishing, 2003, 672 p.
[16]. Hongzhuan Wang. On the Extremal Wiener Polarity Index of Hückel Graphs. Computational and Mathematical Methods in Medicine, vol. 2016, article ID 3873597. http://dx.doi.org/10.1155/2016/3873597.
[17]. Xiaoxin Xu, Yubin Gao, Yanbin Sang, and Yueliang Liang. On the Wiener Indices of Trees Ordering by Diameter-Growing Transformation Relative to the Pendent Edges. Mathematical Problems in Engineering, vol. 2019, article ID 8769428.
[18]. The On-Line Encyclopedia of Integer Sequences (OEIS). http://oeis.org/
[19]. Михаил Дехтярь. Основы дискретной математики. Лекция 10: Деревья. ИНТУИТ (национальный открытый университет). https://www.intuit.ru/studies/courses/1084/192/lecture/5017 / Mihail Dekhtyar'. The basics of discrete mathematics. Lecture 10: The trees. INTUIT (National Open University). Available at: https://www.intuit.ru/studies/courses/1084/192/lecture/5017 (in Russian).

Информация об авторе / Information about the author

Игорь Борисович БУРДОНОВ – доктор физико-математических наук, ведущий научный сотрудник. Научные интересы: формальные спецификации, генерация тестов, технология компиляции, системы реального времени, операционные системы, объектно-ориентированное программирование, сетевые протоколы, процессы разработки программного обеспечения.

Igor Borisovich BURDONOV – Doctor of Physical and Mathematical Sciences, Leading Researcher. Research interests: formal specifications, test generation, compilation technology, real-time systems, operating systems, object-oriented programming, network protocols, software development processes.