ISSN 0005-1179 (print), ISSN 1608-3032 (online), Automation and Remote Control, 2025, Vol. 86, No. 11, pp. 1074–1087.

A Distributed Graph Vertex Numbering Algorithm Combined with Breadth-First Search Tree Construction

O. P. Kuznetsov

Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Moscow, Russia
e-mail: olpkuz@yandex.ru

Received February 11, 2025
Revised May 29, 2025
Accepted September 12, 2025

Abstract—A new distributed algorithm for numbering the vertices of a rooted undirected graph is proposed. During the numbering process, it constructs a spanning tree that is also a breadth-first search tree. The complexity of this algorithm is estimated.

Keywords: undirected graph, distributed algorithm, vertex numbering, spanning tree, breadth-first search

DOI: 10.7868/S1608303225110061

1. INTRODUCTION

Problems of distributed computations on graphs have a long history. The first problem of this class was the firing squad synchronization problem, proposed by J. Myhill in 1957 and solved by V. Levenshtein [1] and F. Moore [2]. It considers a chain of identical automata that, after activating one of them, must pass to the same state in minimal time. Starting from the 1980s, many distributed algorithms for solving various graph problems emerged. Among them, note the problems of finding a minimum spanning tree (MST) [3–7] and a breadth-first search (BFS) tree [8–10]. Methods for solving these and many other problems were reviewed in [11–13].

The general scheme of distributed algorithms is as follows. Processors located at the vertices of a graph exchange messages in a synchronous or asynchronous mode. The computation process can be initiated in two possible ways:

1) Processors in all vertices start working simultaneously.

2) The algorithm starts with activating one vertex, called the graph root; the remaining vertices are activated after receiving messages.

Almost all these algorithms assume that the vertices are either numbered (see the surveys [11–13]) or have some unique identifiers [10]. A distributed vertex numbering algorithm based on the well-known Tarry's algorithm for traversing graph edges [14] was described in [11].

This paper proposes a distributed algorithm for numbering the vertices of an undirected graph; during the numbering process, the algorithm constructs a spanning tree that is also a BFS tree and, accordingly, a tree of shortest paths from the root.

2. ALGORITHM DESCRIPTION

Consider a connected undirected graph with n vertices, m edges, and a specified initial vertex (root), denoted by v_0. Exactly such graphs, called rooted, will be studied below. Each vertex contains a processor that can execute one of the local algorithms, depending on the state of this vertex. All vertices can be in one of four states, indicated by colors: white, gray, black, and red.

A particular local algorithm corresponds to each state (color). From this point onwards, the vertex processor will be identified with the vertex itself: speaking of the actions of a vertex, we mean the actions of its processor.

Each vertex has an ordered list of incident edges. Edges have an attribute that takes one of the four values: incoming (upper), black (lower), dotted (chord), and red (see the details below).

Vertices can exchange messages of two types.

Type 1 has the form (i, c), where i is a number, i.e., $i \in \{1, \ldots, n-1\}$, and c is a color, i.e., $c \in \{\text{black, red}\}$. For brevity, messages of this type will be specified as "black (red) i" or "black (red) number". By default, we assume that the number is black, and the message is of type 1: the expression "send i" means "send message (i, black)".

Type 2 has the form (i, j), where i and j are black numbers; this type of messages arises when a vertex, responding to an attempt to assign number $i + 1$ to it, reports its number j.

In both types of messages, i is the last number assigned. It will be called the current i or current number.

A vertex is white if:

— It is not numbered.

— It waits for messages to receive its number.

After that, a vertex becomes either gray (when, in addition to the incoming edge, it has other incident edges) or red (when it has no such edges, i.e., it is pendant).

A vertex is gray if:

— It is numbered but has unnumbered neighbors.

— It waits for a type 1 message to start numbering the adjacent vertices or for a type 2 message to label the edge via which this message has arrived as a chord.

A vertex is black if it is numbered, and all its neighbors are also numbered. Upon receiving the current number, a black vertex transmits it to the next level.

A vertex is red if the vertices reachable from it at the next levels are either absent or numbered. In both cases, this is reported upward: red i is sent via the incoming edge. The conditions of transition to the red color will be described in detail below.

The idea of coloring vertices white, gray, and black is borrowed from [15], albeit with the following modifications due to the distributed nature of computations: a particular algorithm corresponds to each color; moreover, the red color is added to report the end of the numbering process on a certain branch.

The vertex numbering algorithm proposed here is based on breadth-first search in a graph [15] and is sequential: only one vertex is active at any time instant. Sending a message essentially means transferring control: receiving a message activates the vertex (starts its algorithm, depending on the current color of the vertex), whereas sending a message terminates activity. Due to the sequential nature of the algorithm, during each activity period, a vertex can send only one message to one address.

As will be shown below, the algorithm constructs a spanning tree consisting of incoming edges; each vertex of this tree is at a minimum distance from the root. Therefore, the algorithm can be described using two concepts, namely, level and branch. A branch of a vertex is a subtree with this vertex as the root; the ith level is the set of vertices at distance i from the root of the original graph. Breadth-first traversal means that the vertices of subsequent levels are not numbered until the vertices of the current level are all numbered; thus, if $i < j$, the number of any vertex at the ith level will be less than the number of any vertex at the jth level.

Recall that a chord is an edge not contained in a spanning tree.

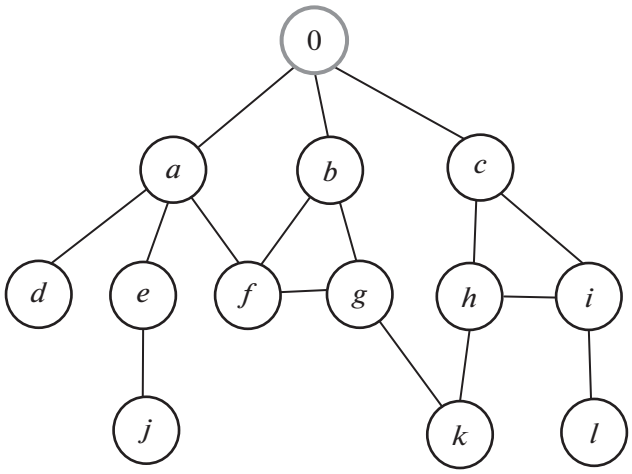

Fig. 1.

As an example, we will take the graph in Fig. 1 at various stages of the algorithm. The letters labeling the vertices are introduced for convenient description and reading. The letters of the vertices are unknown to their neighbors and to the root as well; therefore, they are not the identifiers of vertices in the conventional sense and are not used in the algorithm's operation.

In black-and-white graphics, vertex colors will be depicted as follows. White (unnumbered) vertices are those labeled by letters. Gray vertices have a regular (thin) contour. Black vertices are shaded. Red vertices and edges have a thicker contour. The root is depicted at the top, so the expression "send upward" means "send toward the root."

The scheme of the general vertex numbering algorithm

Initial state:

All vertices, except the root, are white; the root has no color; all edges are black. The set of black edges (SBE) is ordered, so it makes sense to speak of the first edge of the SBE.

Start:

The root assigns number 0 to itself.

The root forms a queue from the SBE and sends 0 via the first edge of the queue.

Subsequently, it executes the local algorithm of the root, which is to manage the numbering process of levels. The general process runs as follows.

In the first cycle, the root initiates the numbering process of level 1 vertices by sending 0 via the first edge of the queue. At the end of this cycle, the queue becomes empty; this means that the first level is numbered, and its vertices have become gray (and the pendant vertices of the first level have become red). The kth cycle ends with the following results: all vertices of the kth level have become gray or red, all vertices of the previous levels have become black or red, and a message with the current number has arrived at vertex 0 via the last edge of the queue.

In the $(k+1)$th cycle, upon receiving this current number, the root re-forms the queue from the SBE and sends a message with the black number via the first edge of the queue. Upon passing through $k-1$ black vertices, this message arrives at a gray vertex v of the kth level; upon receiving this message, the latter vertex numbers its neighbors connected to v by black edges (included in the SBE); upon finishing the numbering process, vertex v sends the last number upward via the incoming edge (u, v) and becomes black. If a neighbor vertex w has been already numbered, edge (v, w) is labeled as a chord. If the SBE of vertex v becomes empty, this vertex becomes red and sends a red number upward to its black vertex u, which makes edge (u, v) red. As soon as all

lower edges of a black vertex have become red, this vertex becomes red itself and sends the red number upward. The algorithm ends when all edges incident to the root become red.

By default, the expression "send message" means the end of the algorithm and waiting for the next message.

Now we describe particular local algorithms: that of the root and those corresponding to different vertex states (colors).

The local algorithm of the root

1a) Assign number 0 to itself.

1b) Start the algorithm for numbering neighbors with current i.

1c) Form a queue from the SBE.

1d) Send i via the first edge of the queue; wait for a message.

2) If black i has been received via the current edge of the queue (the next level of this branch is numbered), then:

2a) remove this edge from the queue;

2b) if the remaining queue is non-empty, send i via the first edge of the queue;
else (the next level of the graph is numbered):

2c) form a queue from the black edges;

2d) send i via the first edge of the queue.

3) Else (a red number i has been received via the current edge of the queue; this means the absence of unnumbered vertices on this branch):

3a) make this edge red and remove it from the queue;

3b) if the queue is non-empty, send i via the first edge;

3c) if the queue is empty and the SBE is non-empty, go to Step 2c);

3d) if the SBE is empty (all edges are red), **end of the general algorithm.**

At the first level, there are no chords, so the root receives only type 1 messages.

The local algorithm of a white vertex

A white vertex has no number. All its edges are black.

Upon receiving a type 1 message with number i :

1) Assign number $i + 1$ to itself.

2) Label the edge via which i has arrived as incoming.

3) If the SBE is non-empty, then:

3a) send black $i + 1$ upward via the incoming edge;

3b) become gray; end of the algorithm.

4) Else:

4a) send red $i + 1$ upward via the incoming edge;

4b) become red; end of the algorithm.

Upon receiving its number and becoming gray, a white vertex "does not know" the status of the numbering process of its level (completed or not); therefore, it does not number its neighbors at the next level, but only reports its number upward via the incoming edge to its "upper" vertex so that the latter continues numbering neighbors. But a white vertex may be pendant, i.e., have only one neighbor connected to it by an incoming edge. In this case (see Step 4) of the algorithm), it skips the gray and black stages and immediately becomes red.

The local algorithm of a gray vertex

1) If a message (x, y) of type 1 has been received via the incoming edge, then:

1a) Form a queue from the SBE.

1b) Send i via the first edge of the queue.

1c) If black $i+1$ is returned via this edge, then:

1c1) remove this edge from the queue;

1c2) if the queue is non-empty, go to Step 1b) with $i+1$;
else:

1c3) send $i+1$ via the incoming edge;

1c4) become black; end of the local algorithm.

1d) If red $i+1$ is returned via this edge, then:

1d1) make this edge red;

1d2) remove it from the queue;

1d3) if the queue is non-empty, go to Step 1b) with $i+1$;
else

1d4) if the SBE is non-empty, go to Step 1c3);
else (all edges are red):

1d5) send red i upward;

1d6) become red.

2) Else (a message (x, y) of type 1 has been received via a black edge):

2a) make this edge dashed (this edge is a chord);

2b) send via this edge a type 2 message (i, j), where j is the number of this vertex;

2c) if the SBE is non-empty, end of the algorithm;

2d) else:

2d1) send red i upward;

2d2) become red; end of the algorithm.

Step 2) corresponds to the situation when the vertex is already numbered and is the end of a chord. As a result of Step 2b), the SBE may become empty, and then 2d) the vertex becomes red. Case 2d) violates the sequential nature of the algorithm (see Example 4 below). The arrival of a number via a black edge means that the numbering process runs on another branch q, and the emptying of the SBE causes the parallel transmission of a red number upward via this branch p, which may stop at any vertex of the branch. When the red number reaches the root (this will happen when all vertices of branch p become red), the root receives two current numbers: from branch p, numbered earlier, and from branch q, where the last numbering has occurred. In this case: (a) the incoming edge of branch p becomes red and is removed from the root's SBE; (b) the current number remains the one received from branch q since it cannot be less than the number received from branch p.

Example 1 (Fig. 2). Vertex 1 finished numbering its lower neighbors (vertices 5 and 6 became gray and vertex 4 (pendant) red), sent the last number 6 upward, and became black. Upon receiving number 6, the root removes edge (0, 1) from the queue and sends number 6 to gray vertex 2.

Example 2 (Fig. 3). Vertex 2 (gray) starts numbering neighbors and attempts to number gray vertex 6, which has already been numbered by vertex 1. Step 2) of the gray algorithm is triggered for vertex 6: it labels edge (2, 6) as dashed (chord) and sends a type 2 message (6, 6) via this edge to vertex 2. Vertex 2 also labels edge (2, 6) as dashed and sends 6 to the next vertex, which receives number 7 and sends it via the incoming edge to vertex 2. The numbering process of neighbors of vertex 2 is now complete (the queue is empty); it reports this upward and becomes black.

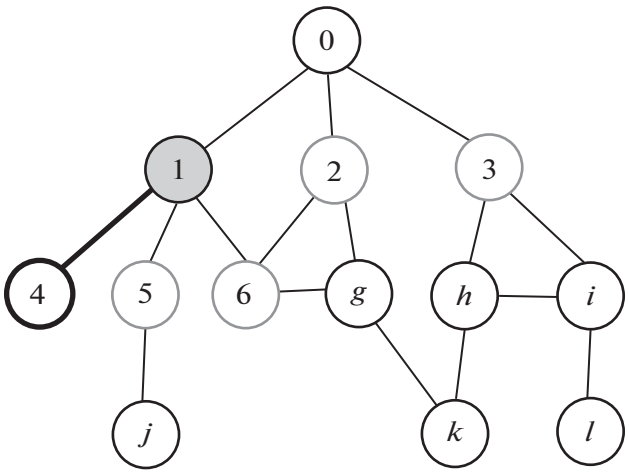

Fig. 2.

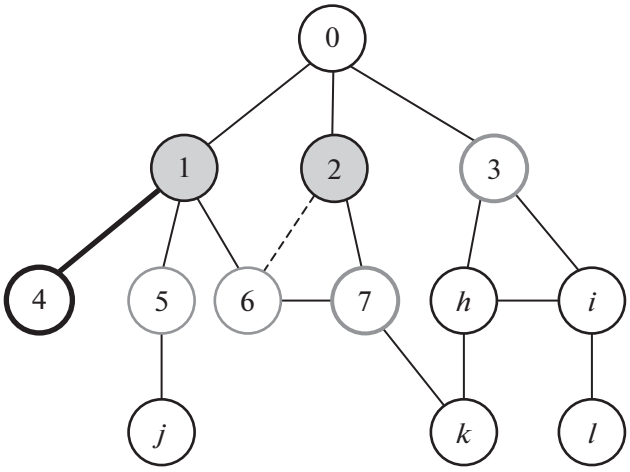

Fig. 3.

The local algorithm of a black vertex

1) If a black number i is received via the incoming edge, then:
   1a) form a queue from the SBE;
   1b) send i via the first edge of the queue.
2) If a black number i is received downward, then:
   2a) remove this edge from the queue;
   2b) if the queue is non-empty, go to Step 1b) with i.
   Else send i via the incoming edge; end of the local algorithm.
3) If a red number i is received downward, then:
   3a) make this edge red and remove it from the queue;
   3b) if the queue is non-empty, send black i via the first edge;
   3c) if the queue is empty, then:
   if all edges are red, become red and send red i upward; end of the local algorithm.
   Else send black i upward; end of the local algorithm.

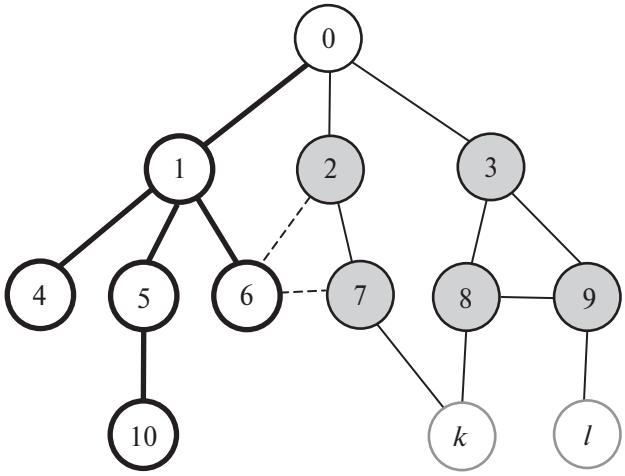

Fig. 4.

Example 3. Figure 4 shows a stage of the general algorithm where the numbering process of level 3 vertices of branch 2 is completed. Consider this process, which leads to the state demonstrated in the figure. The root sent the current number 9 to vertex 1. Vertex 1 and edge (1, 4) became red earlier, during the numbering process of level 2 (see Example 1), so edge (1, 4) is no longer in the SBE of vertex 1. Vertex 5 numbered vertex 10; the latter detected itself to be red and sent red number 10 to vertex 5, which also became red and sent red number 10 to vertex 1, making edge (1, 5) red. Then vertex 1 sends number 10 to vertex 6, which attempts to number vertex 7, receives a type 2 message from it, labels edge (6, 7) as a chord, becomes red (see Step 2d) of the gray algorithm), and sends red number 10 to vertex 1. Vertex 1 makes edge (1, 6) red, detects itself to have no black edges, becomes red, and sends red number 10 to the root. Edge (0,1) becomes red, i.e., is removed from the root's SBE, so branch 1 does not participate in further numbering cycles.

Note the following. The levels of the chord ends may either coincide (such chords will be called horizontal; see chord (6, 7) as an example) or differ at most by 1 (vertical chords; see chord (2, 6) as an example). Indeed, let vertex x on an edge (x, y) be numbered and be at level k, and let y be not numbered yet. Obviously, at this time instant, (x, y) is in the SBE of x. Therefore, during the numbering process of the $(k+1)$th level, vertex x either numbers y or detects that y has already been numbered; in the latter case, (x, y) is labeled as a chord. In any case, as illustrated by the example of edge (6, 7), a horizontal chord of level k is detected as a chord only during the numbering process of the $(k+1)$th level. As we will see below, this can affect the total numbering time.

Example 4 (continuation of Example 3). Upon receiving number 10 from vertex 1, the root starts numbering branch 2, which ends with the arrival of black number 11 at the root, yielding no new red vertices: Step 3) is triggered in the algorithm of white vertex 11 (the SBE still contains edge (8, 11)), and it becomes not red but gray. After this, the numbering process of level 3 of branch 3 begins, during which vertex 8 detects chords (8, 9) and (8, 11). When vertex 8 attempts to number the already numbered (i.e., gray) vertex 11, Step 2) of the gray algorithm is triggered for this vertex, it becomes red and sends its red number upward. Thus, for some time, **two parallel processes** will run: along branch 2, red number 11 is transmitted to the root, and along branch 3, numbering continues. In general, two options are possible: a) the processes run along different branches; b) the processes run along different sub-branches of one branch. In the first case, they converge at the root; in the second, they arrive at some vertex of one branch, which may either change its color (if its SBE becomes empty) or not.

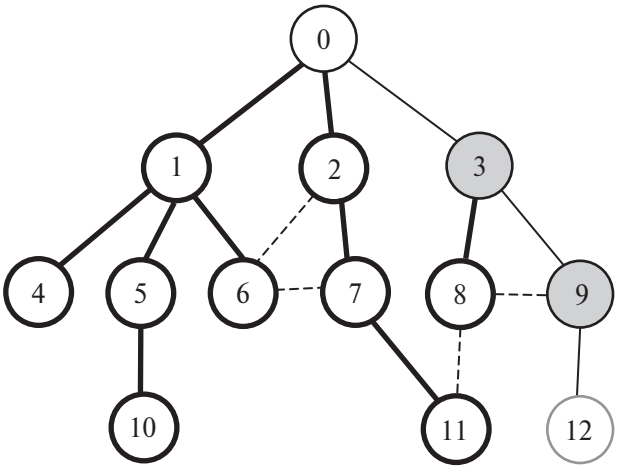

Fig. 5.

Figure 5 shows the time instant when the numbering process is complete. Vertex 12 has just received its number. At the next time instant, it will become red; and through vertices 9 and 3, red number 12 will arrive at the root, which ends the numbering process.

Example 5. Now suppose that a horizontal chord (11, 12) is added to the graph in Fig. 5. As noted at the end of Example 3, a horizontal chord of level k will be detected as a chord only during the numbering process of the $k+1$th level. Therefore, the process will run not as described in Example 4: vertices 11 and 12 do not become red; after completing the numbering of level 3, black number 12 arrives at the root, and the root starts numbering level 4, unaware of its non-existence. Vertices 11 and 12 become red only in the fourth cycle, and the corresponding messages are sent in parallel along branches (11, 7, 2, 0) and (12, 9, 3, 0).

3. MAIN RESULTS. ALGORITHM COMPLEXITY ESTIMATION

Theorem 1. *The incoming edges of an original graph G, labeled by the algorithm, form a tree that is (a) spanning and (b) a tree of shortest paths from the root, i.e., a BFS tree.*

Consider the graph G^* formed by the incoming edges. Each vertex has only one incoming edge. This follows from the fact that an edge is labeled as incoming only when the vertex is white, which then becomes gray. Moreover, if an edge (u, v) is incoming for v, then the number of u is less than the number of v: vertex v has received its number via the edge (u, v), and vertex u has already been numbered by that time. Therefore, on any path from v via incoming edges, the vertex numbers decrease. Hence, (1) a path of incoming edges from any vertex v cannot be a loop, and (2) it can end only at the root, because only the root has no incoming edge. Thus, the graph G^* is connected (any two vertices are either on the same path to the root or connected by two paths leading to the root) and contains no loops, thereby representing a tree. Since each vertex has an incoming edge, the tree G^* contains all vertices of the original graph, i.e., it is spanning.

Now we prove that for any vertex v, its path to the root v_0 in the graph G^* is the shortest one in the graph G. The proof is by induction on the cycles of the numbering process.

The 1st cycle is to number the root's neighbors. Obviously, after its completion, all neighbors of the root become gray, they are numbered, the edges connecting them to the root are incoming, and their distance to the root is 1. Suppose that they form level 1. Clearly, there are no other vertices with unit distance to the root.

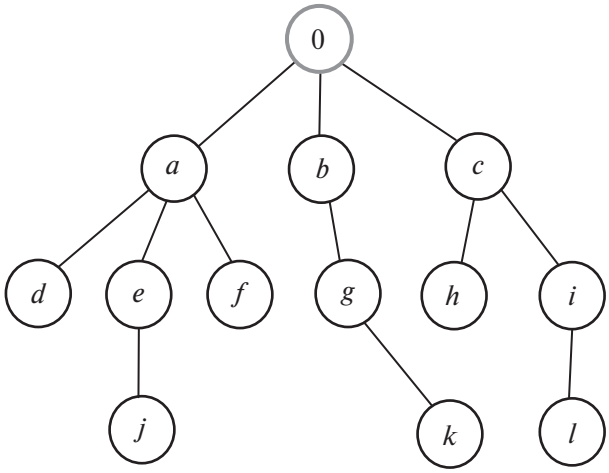

Fig. 6.

Next, assume that the kth numbering cycle has been completed, resulting in new gray vertices whose distance to the root is k, and there are no other vertices with distance k to the root. These vertices form level k. Then all vertices with distance $k+1$ to the root are neighbors of level k vertices; consequently, the former vertices will be numbered by the latter vertices in the $k+1$th cycle, becoming gray and forming level $k+1$. The edges via which they have been numbered will become incoming for them and enter the tree G^*. Thus, for any k, a vertex with distance k to the root will enter level k of the tree G^*, proving that the graph G^* composed of the incoming edges of the original graph G is a tree of shortest paths to the root, i.e., a BFS tree.

Thus, the algorithm produces the following results:

(1) All vertices of the graph are numbered, and the total number of vertices is found.

(2) A spanning tree of the graph is constructed from the incoming edges; the chords of the graph are found, thereby determining its cyclomatic number.

(3) This spanning tree is a BFS tree.

Thus, the algorithm simultaneously solves two problems: distributed vertex numbering and distributed construction of a BFS tree. Therefore, this algorithm will be called the NBFS algorithm.

Note that the local parallelism, arising sometimes during the operation of the NBFS algorithm (see Example 4), does not require synchronization: the final state of the vertex, where two parallel messages arrive, is independent of the order of their arrival.

The spanning tree constructed is not unique and depends on the order of edges in the queues formed by the local algorithms.

For the current example, the BFS tree is shown in Fig. 6. If in the second cycle the queue at the root had the form bac, vertex f would receive number 4, and the edge (b, f) would not be a chord.

Complexity. Henceforth, by the complexity of the NBFS algorithm we mean its communication complexity, i.e., the number of messages generated during the operation of this algorithm. Let us introduce the following notation:

$C_{NBFS}(n)$ is the complexity of the NBFS algorithm for rooted graphs with n vertices;

$C_{NBFS}(G)$ is the complexity of the NBFS algorithm for a particular graph G;

$e(G)$ is the eccentricity of the root of the graph G, i.e., the maximum of the distances from v_0 to the other vertices;

$\gamma(G)$ is the cyclomatic number (and, accordingly, the number of chords) of the graph G;

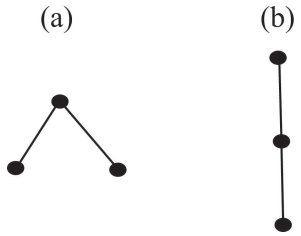

Fig. 7.

G_{BFS} is the spanning tree of the graph G. From this point onwards, we will simply write e and γ.

Any numbering algorithm for the vertices of a rooted graph must sequentially traverse all vertices starting from the root. A distributed algorithm must traverse all edges: in the absence of initial information about global properties of the graph, there is no guarantee of visiting all vertices until visiting all edges. The well-known Tarry's algorithm [14] for traversing all edges is based on splitting each edge into two "half-edges," resulting in an Euler graph where each half-edge can be traversed once; this is equivalent to traversing the original edge twice in different directions, with the traversal necessarily ending at the root. Therefore, for the distributed Tarry algorithm [11], the communication complexity equals the number of visits to each edge, i.e., $2m$. Obviously, due to the sequential nature of any numbering algorithm, the estimate $\mathbf{O}(m)$ cannot be lowered.

Note that there are e levels in G and G_{BFS}; hence, they have e numbering cycles as well. Obviously, among all rooted undirected graphs with n vertices, the chain Ch_n has the greatest eccentricity, i.e., a connected graph with n vertices where two end vertices have degree 1, one representing the root, and all other vertices have degree 2. In this case,

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

Let us estimate the complexity of the chain Ch_n.

Lemma 1.

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

Due to (1), the chain Ch_n has $n-1$ levels; therefore, its numbering requires $n-1$ cycles. The kth cycle ($k = 1, \ldots, n-1$) involves $k+1$ vertices (including the root). The messages they generate can be represented as a chain of length $k+1$ of the form $12\ldots21$, where the ith element of the chain ($i \leqslant k+1$) corresponds to the number of messages generated by the level $(i-1)$ vertex. Indeed, the root and the last vertex of the chain generate one message each, and the remaining vertices generate two each: one to number the next vertex, and the other to report the current number towards the root. The total number of messages in one such chain is $2k$; accordingly, the total number of messages generated by the numbering process of a chain of length n is the sum of messages over all $n-1$ cycles, i.e., twice the sum of the corresponding arithmetic progression: $2\sum_{k=1}^{n-1} k = (n-1)n$.

Since the kth numbering cycle involves all black edges of the first k levels in message exchange, obviously, eccentricity significantly affects the value of C_{NBFS}. A more precise assertion is as follows.

Lemma 2. *Among all rooted trees with n vertices, the tree with the greatest eccentricity—the chain Ch_n—has the highest NBFS complexity.*

We prove this result by induction.

The minimum n for which this statement makes sense is 3. For $n = 3$, two trees are possible (Fig. 7).

Tree 7a has one level and therefore requires one numbering cycle, which corresponds to the chain 211 (two messages from the root and one message from each of the other vertices), i.e., 4 messages. Tree 7b is a chain with two levels and two numbering cycles: the first cycle corresponds to chain 11 and the second cycle to chain 121; in total, $2 + 4 = 6$ messages. Thus, the value of C_{NBFS} is greater for tree 7b than for tree 7a, so the lemma is valid for $n = 3$.

Assume now that the desired assertion is true for all trees with a number of vertices not exceeding k, i.e., the complexity $C_{NBFS}(Ch_k) = (k-1)k$ is maximum for all trees with k vertices. Based on this inductive hypothesis, we will establish the lemma's validity for $k+1$: the chain's complexity $C_{NBFS}(Ch_{k+1}) = (k+1)k$ is maximum for all trees with $k+1$ vertices.

Any tree with $k+1$ vertices can be obtained from some tree G_k with k vertices by attaching an edge (x, y) so that vertex x is identified with some vertex of G_k and vertex y becomes pendant in the new tree G_{k+1}. Two options are possible here.

(a) G_k is a chain. Attaching an edge (x, y) to the chain's end yields the chain Ch_{k+1} with complexity $(k+1)k$. If an edge (x, y) is attached to any other level i vertex of G_k $(i < k)$, the level of the resulting tree G_{k+1} will not change; hence, the $(k+1)$th numbering cycle will be absent, and during the numbering of the $(i+1)$th level, two new messages will appear, from vertex y and back; and y will become red and will not generate further messages. Thus, $C_{NBFS}(G_{k+1}) = C_{NBFS}(Ch_k)+2 < C_{NBFS}(Ch_{k+1})$ and, consequently, the chain Ch_{k+1} has the maximum complexity in the class of all trees with $k+1$ vertices obtained by attaching an edge to the chain Ch_k.

(b) G_k is not a chain. Then $e(G_k) \leqslant k-1$ and, by the inductive hypothesis,

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

Let an edge (x, y) be attached to a pendant vertex v at level $i \leqslant k-1$. Any pendant vertex is the end of some chain attached at the other end to a vertex with a degree above 2, and its length does not exceed $k-1$. During the numbering process of the tree G_k, vertex v (and the entire chain) becomes red at the ith numbering cycle and does not participate in further cycles. In the new tree G_{k+1}, this chain is lengthened by one edge, and its former end v is replaced by vertex y at level $i+1 \leqslant k$. Hence, this chain will participate in the $(i+1)$th numbering cycle, where at most $2(i+1) \leqslant 2k$ messages will be added to the complexity $C_{NBFS}(G_k)$. Therefore, $C_{NBFS}(G_{k+1}) \leqslant C_{NBFS}(G_k) + 2k$, and, using (3), we have $C_{NBFS}(G_{k+1}) \leqslant (k-1)k + 2k = (k+1)k = C_{NBFS}(Ch_{k+1})$. Thus, in case (b) as well, the chain's complexity turns out to be maximum, which finally proves Lemma 2.

Consider now the NBFS complexity of an arbitrary rooted undirected graph G with n vertices and m edges. The cyclomatic number (equivalently, the number of chords) of such a graph is calculated by the well-known formula $\gamma = m - n + 1$.

Theorem 2.

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

The complexity $C_{NBFS}(G)$ of a particular graph G can be represented as the sum of two terms:

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

The first term is the complexity of the spanning tree of the graph G. It is determined by messages related to the numbering itself (assigning a number and reporting the current number back); they are transmitted only via incoming edges, i.e., through the future spanning tree. According to Lemmas 1 and 2, $C_{NBFS}(G_{BFS}) \leqslant n^2 - n$.

The second term is the number of messages generated when detecting chords. In general, detecting a chord (x, y) is associated with two messages: vertex x attempts to number vertex y;

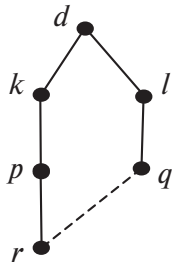

Fig. 8.

vertex y reports that it is already numbered. After this, the edge (x, y) is removed from the SBE of vertices x and y; no subsequent messages are transmitted via it. However, a more complex situation is possible, see Example 4. We study it in detail.

Figure 8 shows a fragment of some graph where vertices p, q are at the ith level and the $(i+1)$th numbering cycle is in progress. Along branch *dkpr* it has already been completed, vertex r has received its number and sent it back to vertex p; meanwhile, the edge (r, q) remains in the SBE of vertex r. After its activation, vertex q attempts to number vertex r and receives a message from this vertex that the latter has already been numbered. Then, both vertices remove the edge (r, q) from their SBE. If the resulting SBE of vertex r is empty (possible only if this vertex is pendant in the future spanning tree), then it becomes red and sends another message to vertex p. Upon receiving this message, vertex p may also become red, which will generate a message from it to vertex k, etc. Thus, simultaneously with the numbering process in branch *dlq*, an additional chain of messages will arise in branch *dkpr*, contributing to the term $C_\gamma(G)$. All these messages make some edges red, causing their removal from the SBE; therefore, only one such message can be transmitted via each edge. Hence, their total number does not exceed m (the total number of edges in the graph); as is well known, $m \leqslant \frac{n^2-n}{2}$.

Another option in this situation has been described in Example 5, where a horizontal chord connects two pendant vertices of the tree G_{BFS}. In this case, an additional numbering cycle arises. The messages of this cycle will be transmitted, at most twice, via all incoming edges of the graph. In this case, the chord's contribution to $C_\gamma(G)$ does not exceed $2m \leqslant n^2 - n$.

Thus, both terms are smaller than or equal to $n^2 - n$, which completes the proof of Theorem 2.

For different classes of graphs, the ratio between the two terms in (5) can vary significantly. In particular, in sparse graphs (graphs with a small number of chords), the main contribution to C_{NBFS} is made by the complexity of the spanning tree, which, in turn, is determined by the root's eccentricity. Therefore, even within the same graph, the complexity will change appreciably depending on the choice of the root; as naturally expected, it takes the minimum value when the root is the graph's center and the maximum value when the root is the end of one of the diameters. According to Lemmas 1 and 2, the upper bound is achieved when the graph is a chain with the root at one of its ends.

The other extreme case is the complete graph K_n, in which $e = 1$ for any choice of root and there are $m = \frac{n^2-n}{2}$ edges; of these, $n - 1$ are incoming edges, and the rest are chords: $\gamma = \frac{n^2-n}{2} - n + 1$. From $e = 1$ it follows that all chords are horizontal. Due to these formulas, the main contribution to $C_{NBFS}(K_n)$ is made by the value of $C_\gamma(G)$, and the upper bound is reached by the number of chords.

The operation of the NBFS algorithm on the graph K_n consists of two numbering cycles. In the first cycle, all vertices receive their numbers, but owing to the horizontality of all chords, no vertex becomes red; therefore, the root does not know that the numbering is complete and starts the second cycle, in which all chords are detected.

4. CONCLUSIONS

The numbering problem has universal significance for distributed algorithms. Almost all distributed algorithms on graphs assume that vertex numbering has already been performed. Moreover, with the uniform numbering for all vertices, the root can request from each vertex the lists of incident edges and thus reconstruct a complete description of the graph.

The efficiency of the NBFS algorithm can be judged by comparing it with Tarry's algorithm. For "dense" graphs, close in the number of edges to K_n, the complexity estimates of both algorithms are of the same order, and consequently, the NBFS algorithm is preferable since it simultaneously constructs a BFS tree. Conversely, for sparse graphs, close to trees, the NBFS algorithm should not be used: firstly, for such graphs, it is much inferior to Tarry's algorithm in terms of complexity (obviously, a chain can be numbered in one traversal); and secondly, the main advantage of the NBFS algorithm (the simultaneous construction of a BFS tree during numbering) does not work here since any tree is itself a BFS tree.

Among potential applications of the NBFS algorithm, in addition to communication networks as a usual application of distributed algorithms, we mention swarm robotics. One possible scenario is as follows: a group of robots starts a patrolling mission in an area, having a complete graph of connections within the group. By the end of the mission, some connections are disrupted, and the leader needs to reconstruct the graph of remaining connections. The percentage of disrupted connections is small, and the graph of remaining connections is close to complete; therefore, the NBFS algorithm will be effective.

Another algorithm that constructs a BFS tree and numbers the vertices was described in [16]. It first constructs a BFS tree and then performs vertex numbering on this tree using a tree traversal similar to Tarry's algorithm, with the significant difference that $m = n - 1$ for trees. Combined with the use of parallelism (in [16], vertices send messages to all their neighbors simultaneously), this gives a time complexity estimate of $O(n)$, which is better than the complexity of the NBFS algorithm. However, this improvement comes at the cost of introducing synchronization, which may either require additional hardware or simply be difficult to implement in real applications, e.g., the above swarm robotics scenario. Moreover, the numbering described in [16] appears irregular: the vertex number value says nothing about the vertex's proximity to the root (i.e., the numbers are merely unique identifiers). In contrast, the numbers in the NBFS algorithm possess the following property (see the beginning of Section 2): if $i < j$, any level i vertex will have a number smaller than any level j vertex. This property has been utilized in the proof of Theorem 1 and may be valuable in applications.

REFERENCES

1. Levenshtein, V.I., On a Method of Solving the Problem of Synchronizing a Chain of Automata in Minimal Time, *Probl. Inf. Transm.*, 1965, vol. 1, no. 4, pp. 14–25.

2. Moore, F.R. and Langdon, G.G., A Generalized Firing Squad Problem, *Information and Control*, 1968, vol. 12, pp. 212–220.

3. Gallager, R.G., Humblet, P.A., and Spira, P.M., A Distributed Algorithm for Minimum-Weight Spanning Trees, *ACM Transactions on Programming Languages and Systems*, 1983, vol. 5, no. 1, pp. 66–77.

4. Peleg, D. and 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, pp. 253–261.

5. Vyalyi, M.N. and Khuziev, I.M., Distributed Communication Complexity of Spanning Tree Construction, *Probl. Inf. Transm.*, 2015, vol. 51, no. 1, pp. 49–65.

6. Vyalyi, M.N. and Khuziev, I.M., Fast Protocols for Leader Election and Spanning Tree Construction in a Distributed Network, *Probl. Inf. Transm.*, 2017, vol. 53, no. 2, pp. 183–201.

7. Dinitz, M., Halldórsson, M., Izumi, T., and Newport, C., Distributed Minimum Degree Spanning Trees, *Proceedings 2019 ACM Symposium Principles Distributed Computing*, 2019, pp. 511–520.

8. Awerbuch, B. and Gallagher, R.G., Distributed BFS Algorithms, *Proc. 26 IEEE Symposium on Foundations Computer Science (FOCS)*, 1985, pp. 250–255.

9. Park, J., Tokura, N., Masuzawa, T., and Hagihara, K., An Efficient Distributed Algorithm for Constructing a Breadth-First Search Tree, *Systems and Computers in Japan*, 1989, vol. 20, pp. 15–30.

10. Makki, S.A.M., Efficient Distributed Breadth-First Algorithm, *Computer Communications*, 1996, vol. 19, no. 8, pp. 628–636.

11. Burdonov, I.B. and Kossatchev, A.S., A General Approach to Solving Problems on Graphs by Collective Automata, *Proceedings of the Institute for System Programming of the RAS*, 2017, vol. 29, no. 2, pp. 27–76.

12. Burdonov, I., Kossatchev, A., and Sortov, A., Distributed Algorithms on Rooted Undirected Graphs, *Proceedings of the Institute for System Programming of the RAS*, 2017, vol. 29, no. 5, pp. 283–310.

13. Ghaffari, M., Distributed Graph Algorithms, 2022.
https://people.csail.mit.edu/ghaffari/DA22/Notes/DGA.pdf

14. Ore, O., *Theory of Graphs*, Providence: American Mathematical Society, 1962.

15. Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C., *Introduction to Algorithms*, Cambridge: MIT Press, 2009.

16. Métivier, Y., Robson, J.M., and Zemmari, A., A Distributed Enumeration Algorithm and Applications to All Pairs Shortest Paths, Diameter., *Information and Computation*, 2016, vol. 247, pp. 141–151.

This paper was recommended for publication by P.Yu. Chebotarev, a member of the Editorial Board