Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 25/02/2026 11:07:11

Annales UMCS Informatica AI XII, 2 (2012) 27–37
DOI: 10.2478/v10065-012-0035-x

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

A self–stabilizing algorithm for finding weighted centroid in trees

Halina Bielak[1]*, Michał Pańczyk[2]†

[1] *Institute of Mathematics, Maria Curie-Sklodowska University, pl. M. Curie-Sklodowskiej 1, 20-031 Lublin, Poland.*
[2] *Institute of Computer Science, Maria Curie-Sklodowska University, pl. M. Curie-Sklodowskiej 1, 20-031 Lublin, Poland.*

Abstract – In this paper we present some modification of the Blair and Manne algorithm for finding the center of a tree network in the distributed, self-stabilizing environment. Their algorithm finds $\frac{n}{2}$-separator of a tree. Our algorithm finds *weighted centroid*, which is direct generalization of the former one for tree networks with positive weights on nodes. Time complexity of both algorithms is $O(n^2)$, where n is the number of nodes in the network.

1 Introduction

A notion of self-stabilizing algorithms on distributed systems was introduced by Dijkstra [**1**] in 1974. A survey in the topic can be found in the paper by Schneider [**2**], and more details in the book by Dolev [**3**]. The notions from the graph theory not defined in this paper can be found in the book by Harary [**4**].

A distributed self-stabilizing system consists of a set of processes (computing nodes) and communication links between them. Every node in the system runs the same algorithm and can change state of local variables. These variables determine *local state* of a node. Nodes can observe the state of variables on themselves and their neighbour nodes. The state of all the nodes in the system determines the *global state*.

Every self-stabilizing algorithm should have a class of global states called *legitimate state* defined, when the system is stable and no action can be done by the algorithm itself. Every other global state is called *illegitimate* and for the algorithm to be correct

*hbiel@hektor.umcs.lublin.pl
†mjpanczyk@gmail.com

UMCS

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 25/02/2026 11:07:11

there has to be some possibility to make a move if the state is illegitimate. The aim of the self-stabilizing algorithm is to bring the legitimate (desirable) state of the whole system after some alteration (from the outside of the system) of variables in the nodes or after the system has been started.

The tree leader (medians) notions were first studied by Zelinka in [**5**] in 1967. The self-stabilizing leader electing in unweighted trees was presented in [**6**] by Bruell, et al. and improved in [**7**] by Blair and Manne. Recently Chepoi et al. [**8**] have presented the self-stabilizing algorithm for the so-called partial rectangular grids. Their algorithm uses generalization of the algorithm from [**6**], which computes also the median of a tree, but in a different way from that in our paper.

In this paper we present an algorithm for electing a centroid node in weighted trees. Every node in the system has its weight. We are going to find the node whose removal would split the tree into connected components with the sum of weights not greater than the half of the weight of the whole tree. Of course, for any node there would be as many such components as the degree of the node is. In other words, we are looking for *weighted centroid* of a tree. Fig. 1 presents the weighted tree of order 7.

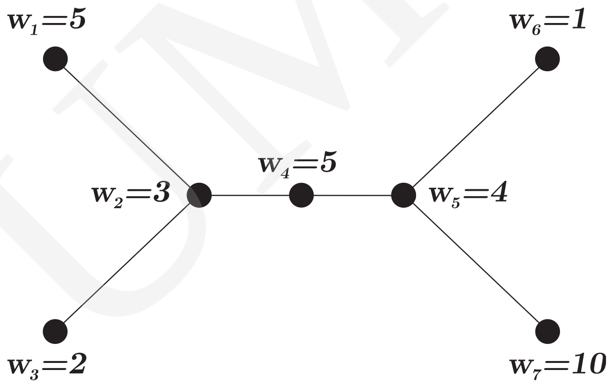

Fig. 1. An example of a weighted tree.

2 Computational Model

We consider a self-stabilizing system modelled by a finite, undirected graph $G = (V, E)$. Let the number of vertices be $n = |V|$ and the number of edges $e = |E|$. In this paper we consider trees, so the number e of the edges in the graph is $n - 1$. A set of all neighbours of the node i is $N(i)$.

We think of every process as a node in the tree, whereas edges are the connection links between them. There are some variables in each node. Names and types of the variables are set during design of an algorithm. Every node can also look up the state of the variables at its neighbours.

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 25/02/2026 11:07:11

Each node runs the same algorithm. The algorithm consists of a set of rules. A rule has the form

```
: label
  If guard
  then assignment instructions.
```

A *guard* is a logic predicate which can refer to variables in the node itself and its neighbours. We say that a rule is *active* if its guard is evaluated to be true. A node is *active* if it contains any active rule. If there is no active node in the graph, we say that the system is *stabilized*.

A self-stabilizing system contains also a *scheduler*. Its task is to choose one process from the set of active processes and to trigger an active rule in it. We call such an action a *move*. In this article the scheduler is assumed to be distributed and adversarial, so the order of activating the nodes is nondeterministic. As a consequence, while describing pessimistic complexity of the algorithm (number of moves), we must take the worst case scenario of triggering actions in particular nodes.

3 Finding Centroid Node Algorithm

Blair and Manne [**7**] presented a fast algorithm for leader election in unweighted trees. Now we present an algorithm for finding centroid in weighted trees. The idea is quite similar to the original algorithm without weights. Let us have an unrooted tree. Every node i of the tree has got the assigned weight w_i which is a positive natural number. *A weighted centroid* is such a node that its removal splits the tree into connected components with the sum of weight of each one not greater than $\mathcal{W}/2$, where $\mathcal{W}$ denotes the entire tree weight.

Lemma 1. There is at least one weighted centroid node in a weighted tree.

PROOF. Let us assume that there is no weighted centroid node in a tree. This means that if we remove any node from the tree, there will be one connected component with the weight greater than $\mathcal{W}/2$.

Starting at an arbitrary node we can go to a neighbour which is a part of the component with the weight greater than $\mathcal{W}/2$. By repeating this step, at some point we will get back to the previous node we have been in. Further steps would loop infinitely between these two nodes. If we cut the edge between them, there will be two connected trees, both with the weight greater than $\mathcal{W}/2$. But this cannot be true since the weights should sum up to $\mathcal{W}$. □

As the existence of centroid nodes has been proven, now we state an upper bound on the number of such nodes.

Theorem 1. The number of centroid nodes in a tree with positive weights on the nodes is either one or two. Moreover, if there are two centroid nodes, they are adjacent.

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 25/02/2026 11:07:11

PROOF. Let us assume that there are two weighted centroid nodes i, j in a tree, such that they are not adjacent. There is at least one node between them from the subtree containing nodes from the set B as shown in Fig. 2. Let B contain neither i nor j. Also let A and C denote the sets of nodes in subtrees (other than the subtree containing B) rooted on the neighbours of nodes i and j respectively.

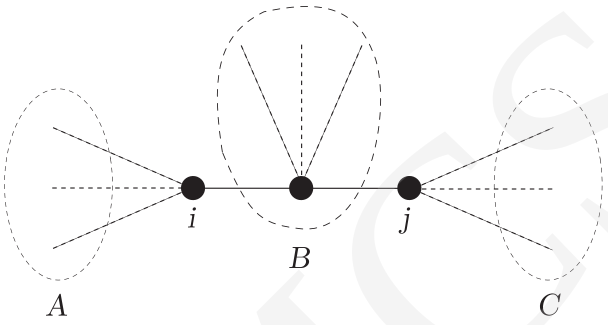

Fig. 2. Visualisation of the proof of Theorem 1.

Let $\mathcal{W}$ denote the weight of the entire tree, so we can write

$$w_A + w_i + w_B + w_j + w_C = \mathcal{W}, \tag{1}$$

where w_A, w_B, w_C, w_i, w_j mean the weights of subgraphs induced by the sets $A, B, C, \{i\}, \{j\}$, respectively. Since i is a weighted centroid we can write down:

$$w_B + w_j + w_C \leq \mathcal{W}/2.$$

The same applies for j, so

$$w_A + w_i + w_B \leq \mathcal{W}/2.$$

By adding the above two inequalities side by side we have

$$w_A + w_i + 2w_B + w_j + w_C \leq \mathcal{W},$$

which compared to (1) implies that $w_B = 0$. But this cannot be true since all weights in the tree are positive. This proves that two centroid nodes are always adjacent.

To end up the proof, it is sufficient to show that there cannot be more than two centroid nodes. Let us assume that there are at least 3 centroid nodes. Since every two of them must be adjacent, they must form a cycle with 3 nodes as a subgraph. But this is impossible since a tree is an acyclic connected graph (net). □

Now we are ready to present the searching weighted centroid node algorithm in the weighted trees. The algorithm consists of two phases. The first one determines weights of some components adjacent to every node of a tree. After stabilization of the first phase every node can also find out the weight of entire entire tree. The second phase of the algorithm selects the centroid node relying on the information given by the first phase. Both phases of algorithms run in parallel. However, while the first one is running, moves made by the second phase are meaningless — they only increase the number of moves made by the algorithm.

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 25/02/2026 11:07:11

3.1 Phase one — determining weights of components

In our algorithm each node contains in itself an array W_i of weights. After stabilization of the system, the value of array $W_i[j]$ for every neighbour j of the node i is the weight of connected component containing node i but with $\{i, j\}$ edge cut.

```
: R1
    If $\exists_{j\in N(i)}\, W_i[j] \neq w_i + \sum_{k\in N(i)-\{j\}} W_k[i]$
    then $W_i[j] := w_i + \sum_{k\in N(i)-\{j\}} W_k[i]$
```

Note that for any leaf i and its neighbour j, $W_i[j]$ will be set to the weight w_i of the node i.

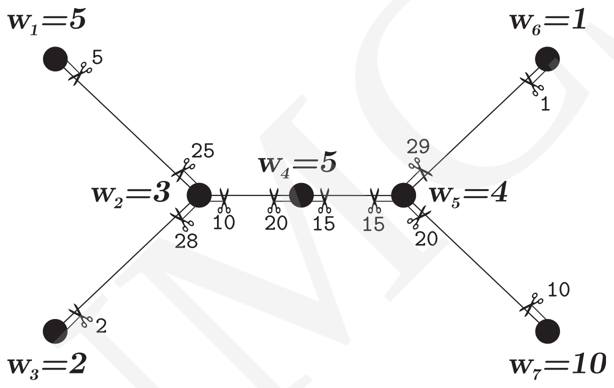

Fig. 3. The tree from Fig. 1 with $W_i[j]$ calculated for every node i and its neighbour j — after stabilization of R1 algorithm.

UMCS

Below we state a lemma about stabilization of R1 algorithm. It is analogous to that from [**7**], but in this case for the weighted tree network algorithm. The idea of the proof is also the same, but in this paper we give an extension to the weighted tree network algorithm.

Lemma 2. Algorithm R1 stabilizes.

Proof. Let us have any sequence of executions $E_1, E_2, \ldots, E_k$ of the rule R1 on a sequence of the consecutive nodes $n_1, n_2, \ldots, n_k, n_{k+1}$ (E_i is an execution of R1 rule on n_i node) such that E_i, $1 < i \leq k$, is the first update of $W_{n_i}[n_{i+1}]$ that makes use of the value $W_{n_{i-1}}[n_i]$. For example, the first such dependency is the use of $W_{n_1}[n_2]$ for updating $W_{n_2}[n_3]$.

Now we show that $n_i \neq n_j$ for $i \neq j$. According to the rule R1, it is impossible to have $n_i = n_{i+1}$. To show that also $n_i \neq n_{i+2}$ for every i, let us consider that propagation of R1 moves along a path in the tree (see Fig. 4). For any node i the value of $W_i[j]$ depends only on the values $W_h[i]$ in the neighbours $h \neq j$. On the other hand, the value $W_i[j]$ influences the value $W_j[k]$ in a node j for its any neighbour $k \neq i$. If we take $n_i = h$ and $n_{i+2} = j$, then the equation $n_i = n_{i+2}$ would imply that there is

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 25/02/2026 11:07:11

a cycle in the graph. But since it is a tree it is not possible. The above argument can be repeated for any $c \geq 2$ in rejecting the equation $n_i = n_{i+c}$. Thus we have $k < n$, where n is the number of nodes of the tree.

There exists exactly one path between any two nodes in a tree. Thus the set of all maximal paths of executing the rule R1 contains n^2 elements when the orientation of such paths (the order from left to right, and from right to left) is distinguished. The only way the number of paths is unbounded is if the same path sequence appears more than once in the set but with different input values. To show that it is impossible, note that the first move of every maximal sequence of moves must depend only on the system initial state. So if E and E' are two maximal paths of execution of the rule R1 such that $n_i = n_i'$, then the first move for both sequences must be the same. Now it follows by induction that E and E' must consist of the same moves since the i-th move is uniquely dependent on the previous $(i-1)$-th move. □

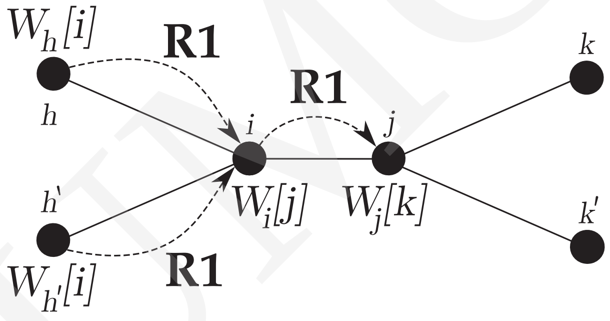

Fig. 4. Propagation of R1 moves along a path in a tree.

Lemma 3. When algorithm R1 has stabilized $W_i[j]$ is correctly computed for every node i and its neighbour j.

PROOF. The proof is by induction on the number of nodes in a tree. As the base case let us take a leaf. There are two possibilities: either $W_i[j]$ in a leaf i is correct or it is not correct at the beginning of running of the algorithm. If it is not correct, the rule R1 is active and it will be triggered finally. After this move, the rule R1 becomes inactive. If the $W_i[j]$ is correct in a leaf, it will never be changed or become incorrect.

Now our induction hypothesis is that for a node i its every neighbour k except for a node j, i.e. $k \neq j$ has computed the value $W_k[i]$ correctly. Given that, if node i has the value $W_i[j]$ incorrect, now it can compute it and set a correct value to $W_i[j]$ relying on its neighbours' information. The proof is done. □

We will now consider a number of moves the algorithm makes to stabilize. Let $v_i(j)$ be the number of nodes in the connected component containing the node i in the graph with $\{i,j\}$ edge cut, and let $c_i(j)$ be the number of the value $W_i[j]$ changes

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 25/02/2026 11:07:11

during execution of algorithm R1. Now we comprise lemma from [7] that will allow us to deduce about the complexity of our algorithm.

Lemma 4. When algorithm R1 has stabilized $c_i(j) \leq v_i(j)$ for every node i and its neighbour j.

The proof of Lemma 4 is similar to that for lemma 3, so it is omitted.

The number of moves in our algorithm is the same as in that for the network tree without weights, so the following lemma from [7] also holds.

Lemma 5. The rule R1 is executed at most $n(n-1)$ times.

PROOF. For every pair of adjacent nodes i, j the property $v_i(j) + v_j(i) = n$ holds, so from Lemma 4 the total number of times that the values $W_i[j]$ and $W_j[i]$ change is at most n. Since the network system is a tree, so the number of edges is $n-1$, the result follows. □

3.2 Phase two — rooting the tree

Once the R1 phase has stabilized the system, every node can determine the weight of the tree. To show this, we introduce a predicate similar to that from [7], which in our algorithm states whether from the node i point of view, it can determine correct weight of the whole tree. The following predicate:

$$wCorrect_i = \left(\forall_{j \in N(i)} \;\; (\; W_i[j] = w_i + \sum_{k \in N(i) - \{j\}} W_k[i] \;) \right)$$

is evaluated in order to run the following part of the algorithm. It is worth mentioning that for any node i, the predicate $wCorrect_i$ is true iff the rule R1 is not active for the node.

Given that the system has been stabilized, every node can determine the weight of the tree. The following lemma gives the method how the node i can calculate the weight of the whole tree.

Lemma 6. If $wCorrect_i$ is true then $W_i[j]+W_j[i] = W_i[k]+W_k[i]$ for the neighbours j, k of node i.

PROOF. Since $wCorrect_i$ is true:

$$\begin{aligned} W_i[j] + W_j[i] &= w_i + \textstyle\sum_{q \in N(i)-\{j\}} W_q[i] + W_j[i] \\ &= w_i + \textstyle\sum_{q \in N(i)} W_q[i] \\ &= w_i + \textstyle\sum_{q \in N(i)-\{k\}} W_q[i] + W_k[i] \\ &= W_i[k] + W_k[i] \end{aligned}$$

□

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 25/02/2026 11:07:11

This way each node can determine whether any of its neighbours has component weight bigger than half of the weight of the whole tree. If this occurs, the node itself cannot be the weighted centroid, and the neighbour is closer to it. In fact, it can be the centroid itself. On the other hand, if for a node its every neighbour has component weight less than half of the whole tree weight, then this node is considered the weighted centroid of the tree.

There can be a situation, where two adjacent nodes can be considered as the weighted centroid, actually when two neighbours have component weight exactly equal to half of the whole tree weight. In such a case we arbitrarily choose the node with greater ID.

All the time the $wCorrect_i$ predicate is true for the node i, from its point of view it can determine the weight of the whole tree, but it is not necessarily correct. It is correct after the rule R1 has stabilized in the whole system. Apart from the above correctness, the weight of the entire tree calculated by node i we denote by $\mathcal{W}_i$.

We present below four further rules of our algorithm. Their meaning is to make a pointer to the centroid: after stabilization if a node is centroid then it points to itself, otherwise the node points to the neighbour closer to the centroid. So each node i contains the variable p_i whose integer value points to the neighbour closer to the centroid or to itself if it is the centroid itself.

```
: R2
    If $(wCorrect_i) \wedge (\forall_{j \in N(i)} W_j[i] < \mathcal{W}_i/2) \wedge (p_i \neq i)$
    then $p_i := i$
: R3
    If $(wCorrect_i) \wedge (\exists_{j \in N(i)} W_j[i] > \mathcal{W}_i/2) \wedge (p_i \neq j)$
    then $p_i := j$
: R4
    If $(wCorrect_i) \wedge (\exists_{j \in N(i)} W_j[i] = \mathcal{W}_i/2) \wedge (ID_i > ID_j) \wedge (p_i \neq i)$
    then $p_i := i$
: R5
    If $(wCorrect_i) \wedge (\exists_{j \in N(i)} W_j[i] = \mathcal{W}_i/2) \wedge (ID_i < ID_j) \wedge (p_i \neq j)$
    then $p_i := j$
```

In all above rule guards there is $wCorrect_i$ predicate. This makes the rules inactive if from the node i point of view weights of components have not been calculated yet. Thus for a node any of the rules R2-R5 can be active if and only if the rule R1 is inactive.

The meaning of the rule R2 is to indicate in the node i that the weighted centroid is only one (unique) and i is the one. The rule R3 makes the p_i pointing to the neighbour of i closer to the centroid if i is not the one. The rules R4 and R5 are activated only if there are two weighted centroid nodes. Then the one with greater ID becomes elected: by itself (the rule R4) and by the other one (the rule R5).

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 25/02/2026 11:07:11

Lemma 7. The algorithm R1–R5 stabilizes on every weighted tree network after at most $2n^2 - n$ moves.

Proof. The first phase of the algorithm (stabilization of the rule R1) takes no more than $n^2 - n$ moves. The second phase consisting of the rules R2-R5 gives the following number of moves: every node can make one move according to the rules R2-R5 before any node triggers the rule R1. This gives n moves. Furthermore, after every R1 move a node can make one of the R2-R5 rules move — this gives extra $n^2 - n$ moves. All in all, it gives $n^2 - n + n + n^2 - n = 2n^2 - n$ moves. □

Lemma 8. After stabilization of the R1-R5 algorithm, the pointer values p_i for every node i in the weighted tree determine a rooted tree with the weighted centroid as the root.

Proof. It is obvious that execution of the R2-R5 rules does not have any influence on the rule R1. Thus the rule R1 will stabilize with correct values $W_i[j]$, for every node i and its neighbour j.

Let us now assume that the entire algorithm has stabilized the weighted tree network. Take a look at the nodes that are not the weighted centroid. Any such node i has exactly one neighbour j for which $W_j[i] > \mathcal{W}/2$, where $\mathcal{W}$ is the weight of the entire tree. For these nodes i the rule R3 may apply after stabilization of the rule R1, but none of the R2, R4, R5 rules is applicable. After possible application of the rule R3, p_i will point to the neighbour which is part of the connected component with its weight greater than $\mathcal{W}/2$. So these nodes point to the neighbour which is part of the subtree containing the weighted centroid.

Now let us assume that there is a unique weighted centroid. Thus there exists no node with $W_j[i] = \mathcal{W}/2$. The centroid i has all the neighbours j with the property $W_j[i] < \mathcal{W}/2$. Then the node i which is the unique centroid may apply the rule R2, pointing to itself as a centroid node. But it is also impossible that i can make a move according to the rules R3-R5. Given that all the other nodes may apply only the rule R3 , the whole tree has been stabilized with orientation to the root denoted.

The other case is when there are two nodes with the weighted centroid property. This means that there is a pair of adjacent nodes p and q such that $W_p[q] = W_q[p] = \mathcal{W}/2$. Then no node i in the tree can have $W_j[i] < \mathcal{W}/2$ for all its neighbours. Thus the nodes with the weighted centroid property cannot apply the rules R2 or R3, and all other nodes can apply only the rule R3. The non-centroid nodes will properly set their p variables pointing towards the centroid node, as shown above. The weighted centroid nodes are adjacent so one of them can apply the rule R4 and the other one can apply the rule R5 if necessary. So the one with greater ID becomes a leader and the other one points to the leader. This proves the last case. The proof is done. □

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 25/02/2026 11:07:11

To end up we can state the following corollary.

Corollary 1. Starting with any global state of weighted tree network system, the algorithm R1-R5 stabilizes in at most $2n^2 - n$ moves having the values p_i for every node i pointing towards the just elected leader as one of the weighted centroid nodes.

The final state of the system Fig. 1 is shown in Fig. 5. The arrows symbolize the pointers p_i for every node i. We have two weighted centroids in the example (nodes 4 and 5). The leader is marked with the circle.

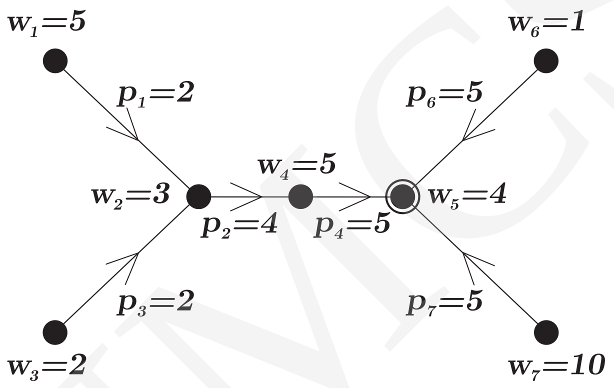

Fig. 5. A state of variable p_i for every node i in a tree after stabilization of the algorithm R1–R5. The weighted centroid node elected by the algorithm is circled.

4 Conclusions

We have presented the self-stabilizing algorithm for finding a centroid in the weighted trees. It is generalization of the algorithm [**7**] for tree networks without weights (precisely with all node weights equal to 1). Our algorithm can be applied in the networks with the weights for electing a node which is the center of the network. One can think of it as limelight of the network.

In the future work we would like to study the weighted centroid and the weighted median in the self-stabilizing systems with topologies other than trees. Some results connected with the above subject can be found in [**9**].

References

[1] Dijkstra E. W., Self-stabilizing in spite of distributed control, Communications of the ACM 17 (1974): 643.
[2] Schneider M., Self-Stabilization, ACM Computing Surveys 25 (1) (1993).
[3] Dolev S., Self-stabilization, The MIT Press (2000).

UMCS

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 25/02/2026 11:07:11

[4] Harary F., Graph Theory, Addison-Wesley (1972).
[5] Zelinka B., Medians and peripherians of trees, Archivum Mathematicum 4 (2) (1968): 87.
[6] Bruell S. C., Ghosh S., Karaata M. H., Pemmaraju S. V., Self-stabilizing algorithms for finding centers and medians of trees, SIAM Journal on Computing 29 (1999): 600.
[7] Blair J. R. S., Manne F., Efficient self-stabilizing algorithms for tree networks, Proceedings of the 23rd International Conference on Distributed Computing Systems, (ICDCS) Providence, Rhode Island (2003): 20.
[8] Chepoi V., Fevat T., Godard E., Vaxès Y., A self-stabilizing algorithm for the median problem in partial rectangular grids and their relatives, Algorithmica 62 (2012): 146.
[9] Bielak H., Pańczyk M., A self-stabilizing algorithm for median election in some weighted graphs (in preparation).

UMCS