CMST 24(3) 187–209 (2018) DOI:10.12921/cmst.2018.0000027

On Half Iterates of Functions Defined on Finite Sets

Paweł Marcin Kozyra

Institute of Mathematics
University of Silesia in Katowice
Bankowa 14, 40-007 Katowice, Poland
E-mail: pawel_m_kozyra@wp.pl

Received: 30 April 2018; revised: 28 August 2018; accepted: 30 August 2018; published online: 30 September 2018

Abstract: Four algorithms determining all functional square roots (half iterates) and seven algorithms finding one functional square root of any function $f\colon X \to X$ defined on a finite set X, if these square roots exist, are presented herein. Time efficiency of these algorithms depending on the complexity of examined functions is compared and justification of correctness is given. Moreover, theorems which make finding half iterates possible in some cases or facilitate this task are formulated.
Key words: functional square root, half iterate, iterated function

I. INTRODUCTION

The *n-th iterate of a function* $f\colon X \to X$ is defined for non-negative integers in the following way:

- $f_0 := id_X$
- $f^{n+1} := f \circ f^n$

where id_X is the identity function on X and $f \circ g$ denotes function composition. Fractional iterates (iterative roots of n-th order) are defined as follows: $f^{\frac{1}{n}}$ is a function $g\colon X \to X$ such that $g^n = f$, for all $n \in \mathbb{N}$. In particular a functional square root (half iterate) of f is function g such that $g^2 = g \circ g = f$.

The literature pertaining to finding functional square roots involves mainly:

- works by Hellmuth Kneser's, who studied the half iterate of the exponential function [3]
- Charles Babbage's research from 1815 of the solutions of $f\big(f(x)\big) = x$ over $\mathbb{R}$, so called involutions of the real numbers [4].

For the given function h the solution Ψ of Schröder's equation

$$\Psi\big(h(x)\big) = s\Psi(x),$$

where the eigenvalue $s = h'(a)$ and $h(a) = a$ enables finding arbitrary functional n-roots [5–7]. In general, all functional iterates of h are given by $h_t(x) = \Psi^{-1}\big(s^t\Psi(x)\big)$, for $t \in \mathbb{R}$. In [8] M. Zdun dealt with the problem of existence and uniqueness of continuous iterative roots of homeomorphisms of the circle. Let $\mathbb{S}^1 := \{z \in \mathbb{C}\colon |z| = 1\}$ and $F\colon \mathbb{S}^1 \to \mathbb{S}^1$ be a homeomorphism without periodic points. Zdun showed that if the limit set of the orbit $\{F^k(z), k \in \mathbb{Z}\} = \mathbb{S}^1$, then F has exactly n iterative roots of n-th order. Otherwise, F either has no iterative roots of n-th order or F has infinitely many iterative roots depending on an arbitrary function.

To determine the functional square roots of functions defined on the finite sets the problem should initially be simplified. Let $S(n)$ denote the set of numbers $\{1, \ldots, n\}$ and $Var(n)$ denote the set of all functions $\alpha\colon S(n) \to S(n)$, for $n \in \mathbb{N}$. Note that for any function $f\colon X \to X$, where $X = \{x_1, \ldots, x_n\}$ is a finite set, there exists the function $\alpha\colon S(n) \to S(n)$ such that $f(x_i) = x_{\alpha(i)}$ for all $i \in S(n)$. Thus only half iterates in $Var(n)$ need to be considered. In order to find the half iterates of $\alpha \in Var(n)$ all functions $\beta \in Var(n)$ could be taken into consideration and checked whether $\beta(\beta(i)) = \alpha(i)$ for all $i \in S(n)$. Unfortunately, such procedure is extremely non-effective. The time of work of such an algorithm is relatively very long, even for n smaller than 10. Therefore, algorithms based upon other ideas have been invented, which can relatively quickly inform us whether there exist functional square roots and, if the roots exist, these algorithms can find them. The time of finding of half iterates is longer than the time of determining if they exist but

much shorter than the time of work of the primitive algorithm described previously.

Sometimes it is convenient to represent a function $\alpha \in Var(n)$ as corresponding to α the directed graph $G = (V, E)$ denoted by $G(\alpha)$, such that:

1. $V = S(n)$;
2. $e \in E$ iff $e = (k, l)$ and $\alpha(k) = l$ for some $k, l \in S(n)$.

Standard terminology from the graph theory (see [1, 2]) is used. If $G(\alpha)$ consists of many components then there is possible further simplification of the problem of finding half iterates of α. It will be proven that there exists a half iterate of α iff

1. for each component in $G(\alpha)$ there exists its square root (see Definition 7)
   or
2. for each component $G_1 = (V_1, E_1)$ of $G(\alpha)$ which has no square root there exists another component $G_2 = (V_2, E_2)$ of $G(\alpha)$ of the same type as G_1 (see Definition 6) such that there exists a square root of the graph being the union of these components – $G_3 = (V_1 \cup V_2, E_1 \cup E_2)$.

For convenience, functions $\alpha \in Var(n)$ with sequences $(\alpha(1), \ldots, \alpha(n))$ or with vectors $[\alpha(1), \ldots, \alpha(n)]$ will be identified. For example, consider $\alpha = (2, 3, 3)$. Then $G(\alpha)$ has the form as in Fig. 1.

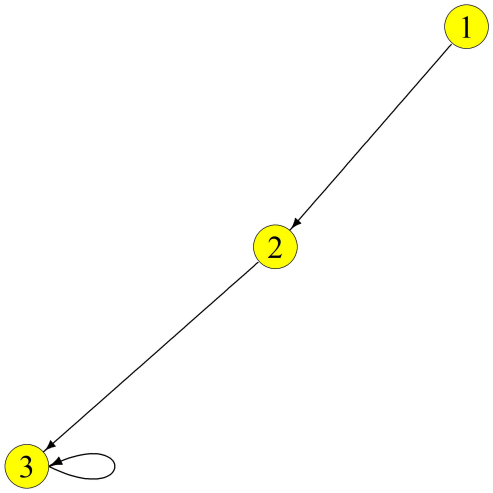

Fig. 1. Graph $G(\alpha)$ corresponding to function $\alpha = (2, 3, 3)$

Note that α has no functional square root. Suppose that such half iterate $\beta \in Var(3)$ exists. Then in particular $\beta(1) \in S(3)$. Suppose that $\beta(1) = 1$. Then $2 = \alpha(1) = \beta(1) = 1$ – contradiction. Similarly, if $\beta(1) = 2$, then $2 = \alpha(1) = \beta(2)$, hence $2 = \beta(2) = \alpha(2) = 3$ – contradiction, and if $\beta(1) = 3$, then $2 = \alpha(1) = \beta(3)$, so $\beta(2) = \alpha(3) = 3$, therefore $3 = \alpha(2) = \beta(3) = 2$ – contradiction. It is seen that the assumption of the existence of a functional square root of α results in contradiction.

All algorithms presented have some common procedures. One such common procedure is $ppfsr$ which determines all possible paths for the half iterates of α based on the above reasoning, i.e.

1. the algorithm attributes all pendants (vertices whose degree is 1) of $G(\alpha)$ to the set R;
2. in the next step for each cycle in $G(\alpha)$ the algorithm adds to the R one chosen element belonging to the cycle, if some component of $G(\alpha)$ forms this cycle (see Definition 5 from the next section);
3. Further algorithm checks for all $k \in R$ and $b \in S(n)$ if the assumption that β is a functional square root of α and $\beta(k) = b$ does not result in contradiction, as in the previous example. If not, the algorithm creates path $(k, \beta(k), \ldots, \beta^m(k))$, for some $m \in S(n)$. Obviously it must happen that $(\beta^i(k), \ldots, \beta^m(k))$ forms a cycle for some $i \in \{0, \ldots, m-1\}$.

It may happen that a half iterate does not exist although paths created by the algorithm $ppfsr$ exist for all elements in the set R also created in this algorithm. For example, consider $v = (3, 5, 3, 4, 3, 3, 4, 6, 1, 4)$. The graph $G(v)$ corresponding to v has the form seen in Fig. 2.

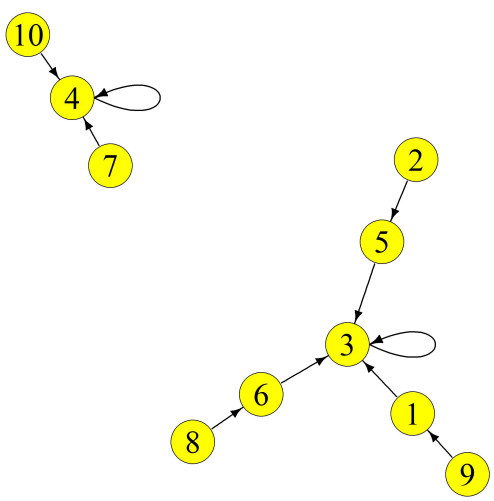

Fig. 2. Graph $G(v)$ corresponding to function $v = (3, 5, 3, 4, 3, 3, 4, 6, 1, 4)$

Then $ppfsr(v)$ returns the list of the sets of possible paths for the functional square roots of v:

$\big[\{[2,1,5,3,3],[2,6,5,3,3],[2,7,5,4,3,4],$
$[2,8,5,6,3,3],[2,9,5,1,3,3],[2,10,5,4,3,4]\},$
$\{[7,4,4],[7,3,4,3],[7,10,4,4],[7,1,4,3,4],$
$[7,5,4,3,4],[7,6,4,3,4]\},\{[8,1,6,3,3],$
$[8,5,6,3,3],[8,2,6,5,3,3],[8,7,6,4,3,4],$
$[8,9,6,1,3,3],[8,10,6,4,3,4]\},\{[9,5,1,3,3],$
$[9,6,1,3,3],[9,2,1,5,3,3],[9,7,1,4,3,4],$
$[9,8,1,6,3,3],[9,10,1,4,3,4]\},\{[10,4,4],$
$[10,3,4,3],[10,7,4,4],[10,1,4,3,4],$
$[10,5,4,3,4],[10,6,4,3,4]\}\big].$

In the next step using the procedure del_L all algorithms presented here remove all paths p such that for all paths q, beginning at some other point, paths p and q cannot be the sequences of the values of the consecutive iterates of some function, meaning there does not exist a function w such that $p = [r, w(r), \ldots, w^m(r)]$ and $q = [s, w(s), \ldots, w^n(s)]$ for some $r, s \in R, r \neq s, m, n \in \mathbb{N}$ and $p, q \in ppfsr(v)$. For example, consider the path $p := [2,1,5,3,3]$ and suppose that for some half iterate w of v holds: $p = [2, w(2), w^2(2), w^3(2), w^4(2)]$. Then for some path q beginning at 8 the same must happen, meaning $q = [8, w(8), \ldots, w^m(8)]$ for some $m \in \mathbb{N}$. But we can see that it is impossible. Thus p cannot be path determined by some functional square root of v. After removing such paths the following list is obtained:

$\big[\{[2,7,5,4,3,4],[2,10,5,4,3,4]\},$
$\{[7,3,4,3],[7,1,4,3,4],$
$[7,5,4,3,4],[7,6,4,3,4]\},$
$\{[8,7,6,4,3,4],[8,10,6,4,3,4]\},$
$\{[9,7,1,4,3,4],[9,10,1,4,3,4]\},$
$\{[10,3,4,3],[10,1,4,3,4],$
$[10,5,4,3,4],[10,6,4,3,4]\}\big].$

Now any pair of paths p and q from this list can be determined by some half iterate, but it is impossible to find paths such that each of these paths belong to exactly one set from the above list and all these paths can be determined by some function. Hence a half iterate of v does not exist, since every half iterate is a function. Algorithms presented differ depending on the way they find the collection of paths such that every path from this collection belongs to exactly one set in $del_L\big(ppfsr(v)\big)$ and there exists the function w such that every path from this collection is a sequence of the values of the consecutive iterates of w at some point belonging to the set R created in the second step of the algorithm $ppfsr$. It will be proved that if $v, w \in Var(n)$ for some $n \in \mathbb{N}$, then $w \circ w = v$ iff for every set $S \in del_L\big(ppfsr(v)\big)$ there exists exactly one path $p \in S$ such that $w(p[i]) = p[i+1]$ for all $i \in \{1, \ldots, len\}$, where len is the length of path p.

II. SPECIAL CASES. THE GRAPH THEORY POINT OF VIEW

In this section the result (Proposition 1) which provides a way of determining whether a function $\alpha \in Var(n)$ is its half iterate is presented. Moreover, Corollaries 11 and 13 enable determining the existence of the half iterate in some cases and Theorem 9 simplifies the problem of finding half iterates of α if the graph corresponding to α contains many cycles. Additionally, Propositions 14 and 17, describing the number of half iterates of constant and identity sequences, are presented.

Proposition 1. *Let $\alpha \in Var(n)$ for some $n \in \mathbb{N}$. Then α is its half iterate iff for each $k \in S(n)$ if $\alpha^{-1}(k) \neq \emptyset$ then $k \in \alpha^{-1}(k)$.*

Proof. ($\Longrightarrow$) Assume that $\alpha^2(k) = \alpha(k)$ for any $k \in S(n)$. Fix $k \in S(n)$ and assume that $l \in \alpha^{-1}(k)$. Then $\alpha(l) = k$ and $k = \alpha(l) = \alpha^2(l) = \alpha(k)$. So $k \in \alpha^{-1}(k)$.

($\Longleftarrow$) Assume that for each $k \in S(n)$ if $\alpha^{-1}(k) \neq \emptyset$ then $k \in \alpha^{-1}(k)$. Fix any $k \in S(n)$. Then $\alpha(k) = m$ for some $m \in S(n)$ and $\alpha^{-1}(m) \neq \emptyset$. By assumption, $\alpha(m) = m$. Hence $\alpha(k) = \alpha(m)$ and $\alpha^2(m) = \alpha(m) = m$. Thus $\alpha^2(k) = \alpha^2(m) = m = \alpha(k)$. □

If the graph $G(\alpha)$ corresponding to a function α has many components, then the problem of finding the half iterates of α can be simplified by the following theorem. Before we formulate this statement we introduce several definitions. Assume that $G = (V, E)$ is a directed graph. Let $\mathbb{Z}_k$ denote the set $\{0, \ldots, k-1\}$ for $k \in \mathbb{N}$.

Definition 2. A sequence $(a_0, \ldots, a_{k-1}) \in V^k$ is called *a cycle of length $k \in \mathbb{N}$ in graph G* (in other words, graph G *contains a cycle* $(a_0, \ldots, a_{k-1}) \in V^k$) iff $card\{a_0, \ldots, a_{k-1}\} = k$ and $(a_i, a_{(i+1) \mod k}) \in E$ for all $i \in \mathbb{Z}_k$.
A function $\alpha \in Var(n)$ *contains a cycle* $(a_1, \ldots, a_k) \in S(n)^k$ iff $G(\alpha)$ contains this cycle.

Definition 3. Two cycles $(a_0, \ldots, a_{k-1}), (b_0, \ldots, b_{k-1}) \in V^k$ of graph G are *equivalent* iff for all $i \in \mathbb{Z}_k$ $b_i = a_{(i+m) \mod k}$ for some $m \in \mathbb{N}_0$. We shall identify equivalent cycles.

Remark 4. If $\bar{a} = (a_0, \ldots, a_{k-1}), \bar{b} = (b_0, \ldots, b_{k-1})$ are two cycles in $G(\alpha)$ for some $\alpha \in Var(n)$ and $a_i = b_j$ for some $i, j \in \mathbb{Z}_k$ then $\bar{a}$ and $\bar{b}$ are equivalent.

Definition 5. A graph (component of graph) $G = (V, E)$ *forms a cycle* iff there exists a sequence of all elements of V which is a cycle of length $card(V)$ in graph G.
A function $\alpha \in Var(n)$ *forms a cycle* $(a_0, \ldots, a_k)$ iff $G(\alpha)$ forms this cycle.

Note any cycles belonging to any component of $G(\alpha)$ are equivalent for any $\alpha \in Var(n)$ and $n \in \mathbb{N}$.

Definition 6. Two components of the graph G are *the same type* if they contain the cycles of the same length.

For example, consider the graph seen in Fig. 3.

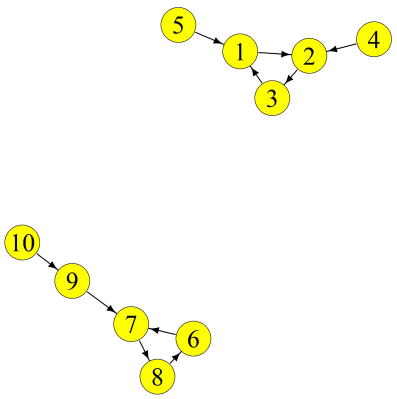

Fig. 3. Graph including two components of the same type

It is seen that two cycles $(1,2,3)$ and $(6,7,8)$ of this graph have the same length equal to 3. Hence two components containing these cycles are the same type.

Definition 7. The graph $G' = (V, E')$ is *a square root of a directed graph* $G = (V, E)$ iff

1. out-deg(u)=1 in G and in G' for every vertex $u \in V$ and
2. if $u, v \in V$ are any vertices then $(u, v) \in E$ iff there exists vertex $w \in V$ such that $(u, w) \in E'$ and $(w, v) \in E'$.

Remark 8. Note that if $v \in Var(n)$ and $w \in Var(n)$ is a half iterate of v then $G(w) = \big(S(n), E'\big)$ is a square root of $G(v) = \big(S(n), E\big)$. Certainly, $G(w)$ satisfies the first condition of definition of a square root, since w is a function. Assume that $k, l \in S(n)$ and $(k, l) \in E$. Then by definition of $G(v)$, $v(k) = l$. Since w is a half iterate of v, so $w\big(w(k)\big) = l$. Take $m := w(k)$. Then $(k, m) \in E'$ and $(m, l) \in E'$. Conversely, if $(k, m) \in E'$ and $(m, l) \in E'$ for some $m \in S(n)$, then by definition of $G(w)$ it follows that: $w(k) = m$ and $w(m) = l$, hence $v(k) = w\big(w(k)\big) = w(m) = l$, since w is a half iterate of v. Thus $(k, l) \in E$. Similarly, if $v \in Var(n)$ and $G' = \big(S(n), E'\big)$ is a square root of $G(v) = \big(S(n), E\big)$, then $G' = G(w)$ for some half iterate of v. It suffices to define w as follows: for any $k \in S(n)$ put $w(k) := l$, where $l \in S(n)$ is the only one vertex such that $(k, l) \in E'$. Such vertex exists by definition of the square root.

Now the theorem can be formulated:

Theorem 9. *Let $\alpha \in Var(n)$ for some $n \in \mathbb{N}$. Then there exists a half iterate of α iff*

1. *for each component in $G(\alpha)$ there exists its square root or*
2. *for each component $G_1 = (V_1, E_1)$ of $G(\alpha)$ which has no square root there exists another component $G_2 = (V_2, E_2)$ of $G(\alpha)$ of the same type as G_1 such that there exists a square root of the graph being the union of these components – $G_3 = (V_1 \cup V_2, E_1 \cup E_2)$.*

Proof. $(\Longrightarrow)$
Assume that β is a half iterate of α and suppose that some component $G_1 = (V_1, E_1)$ of $G(\alpha)$ has no square root. Let $\bar{a} := (a_0, \ldots, a_{k-1})$ be the cycle in G_1. Define $b_i := \beta(a_i)$, for $i \in \mathbb{Z}_k$. Note that $\alpha(b_i) = \beta^2\big(\beta(a_i)\big) = \beta\big(\alpha(a_i)\big) = \beta(a_{i+1}) = b_{i+1}$ for $i \in \mathbb{Z}_{k-1}$ if $k \geq 2$ and $\alpha(b_{k-1}) = \alpha\big(\beta(a_{k-1})\big) = \beta\big(\alpha(a_{k-1})\big) = \beta(a_0) = b_0$. Therefore, $\bar{b} := (b_0, \ldots, b_{k-1})$ is a cycle of length k in $G(\alpha)$ different from the cycle $\bar{a}$, since $b_i \neq a_j$ for all $i, j \in \mathbb{Z}_k$. Suppose that $b_i = a_j$ for some $i, j \in \mathbb{Z}_k$. Then $\bar{a}$ and $\bar{b}$ are equivalent. So $b_i = a_{(i+m) \mod k}$ for some $m \in \mathbb{N}_0$. Fix any vertex $c_0 \in V_1$. Then $\alpha^n(c_0) = a_i$ for some $n \in \mathbb{N}$ and $i \in \mathbb{Z}_k$. Let l be the smallest such n. Define $c_i := \alpha^i(c_0)$ and $d_i := \beta(c_i)$ for $i \in \mathbb{Z}_l$. Note that $\alpha(d_{l-1}) = \alpha\big(\beta(c_{l-1})\big) = \beta\big(\alpha(c_{l-1})\big) = \beta(a_i) = b_i = a_{(i+m) \mod k} \in V_1$ for some $i \in \mathbb{Z}_k$. So $d_{l-1} \in V_1$, since $(d_{l-1}, a_{(i+m) \mod k}) \in E_1$ and G_1 is a component. Similarly, $\alpha(d_j) = d_{j+1}$ and $d_j \in V_1$ for $j \in \mathbb{Z}_{l-1}$. In particular, $d_0 = \beta(c_0) \in V_1$. Thus $\beta(V_1) \subseteq V_1$. Define $G' = (V_1, E')$, where $E' := \{(k, l) \colon k, l \in V_1 \wedge \beta(k) = l\}$. Then G' is a square root of G_1 – contradiction. Therefore, $b_i \neq a_j$ for all $i, j \in \mathbb{Z}_k$ and $\bar{b}$ must belong to component $G_2 = (V_2, E_2)$ different from G_1 but the same type as G_1 since $\bar{a}$ and $\bar{b}$ have the same length. Let $E'' := \{(k, l) \colon k, l \in V_1 \cup V_2 \wedge \beta(k) = l\}$. Then $G'' := (V_1 \cup V_2, E'')$ is a square root of $G_3 := (V_1 \cup V_2, E_1 \cup E_2)$.

$(\Longleftarrow)$
Assume that $G(\alpha) = (V, E)$ is the union of $G_1, \ldots, G_m$, where $G_i = (V_i, E_i)$ is either a component of $G(\alpha)$ or union of two components of $G(\alpha)$ of the same type, and for each G_i there exists its square root – $G_i' = (V_i, E_i')$, for $i \in S(m)$. Then $V = \bigcup_{i=1}^{m} V_i$ and $V_i \cap V_j = \emptyset$ for all $i \neq j$, $i, j \in S(m)$. Define $\beta \in Var(n)$ in the following way:

$\beta(k) := l$, if $k \in V_i$ and $(k, l) \in E_i'$ for some $i \in S(m)$, for any $k \in V$.

Then β is a correctly defined function. It will be shown that $\beta^2 = \alpha$: Take any $k \in V$. Then there exists exactly one $i \in S(m)$ such that $k \in V_i$. Since G_i' is a square root of G_i, so there exists exactly one vertex $l \in V_i$ such that $(k, l) \in E_i'$. Similarly, there exists exactly one vertex $p \in V_i$ such that $(l, p) \in E_i'$. Thus by definition of β, $\beta(k) = l$ and $\beta(l) = p$, hence $\beta^2(k) = p$. On the other hand, by definition of a square root, $(k, p) \in E_i$, thus $\alpha(k) = p = \beta^2(k)$. $\square$

Return to the previous example. Note that graph G from Fig. 3 corresponds to the function $\alpha = (2,3,1,2,1,7,8,6,7,9)$. The above theorem can be used to show that there exists a half iterate of α. It can be seen that the second component of the graph has no square root. It is easy to find square roots of G. It suffices to intersperse vertices 10 and 9 with 4 or 5. Hence two square roots of G can be obtained, as seen in Fig. 4.

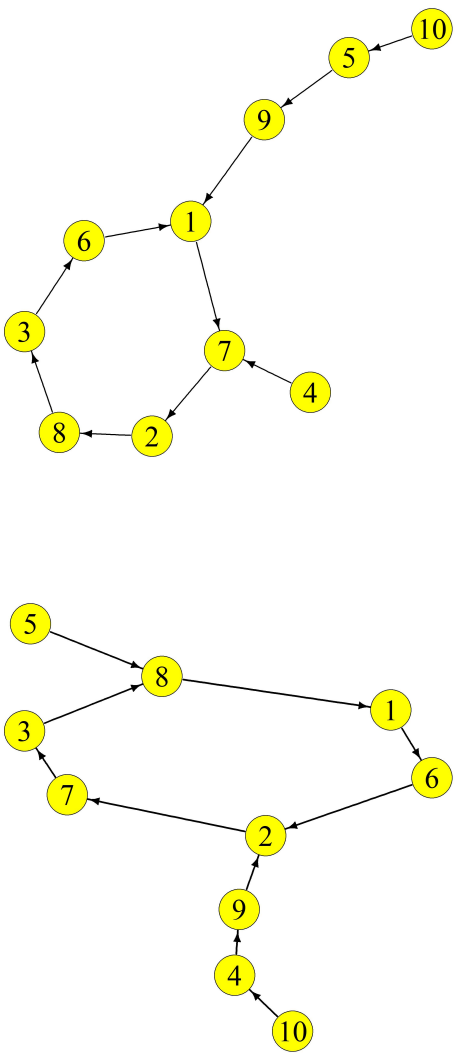

Fig. 4. The square roots of graph G from Fig. 3

By the form of square roots of G, it is easy to determine the half iterates of α: $\beta_1 = (7,8,6,7,9,1,2,3,1,5)$ and $\beta_2 = (6,7,8,9,8,2,3,1,2,4)$.

How the existence of a half iterate depends on the lengths of cycles formed by any functions will be shown.

Proposition 10. *Assume that $\alpha \in Var(n)$, $G(\alpha)$ consists of one component including cycle $\bar{a} := (a_0, \ldots, a_{k-1})$ of length k and $A := \{a_0, \ldots, a_{k-1}\}$.*

1. *If there exists half iterate $\beta \in Var(n)$ of α, then k is odd number and $\beta|_A = \alpha^{(k+1)/2}|_A$.*
2. *If $\beta \in Var(n)$ and α forms the cycle $\bar{a}$, then $\beta^2 = \alpha$ iff $\beta = \alpha^{(n+1)/2}$.*

Proof. Assume that $\beta \in Var(n)$ and $\beta^2 = \alpha$. Define $b_i := \beta(a_i)$ for all $i \in \mathbb{Z}_k$. Note that $\bar{b} := (b_0, \ldots, b_{k-1})$ is a cycle in $G(\alpha)$, since $\alpha(b_i) = \alpha(\beta(a_i)) = \beta(\alpha(a_i)) = \beta(a_{i+1 \mod k}) = b_{i+1 \mod k}$ for all $i \in \mathbb{Z}_k$. Thus $\bar{b}$ is equivalent with $\bar{a}$, since $G(\alpha)$ contains one cycle. Hence there exists $m \in \mathbb{Z}_k$ such that $b_i = a_{i+m \mod k}$ for all $i \in \mathbb{Z}_k$. In particular,

$$\begin{aligned} a_1 =& \alpha(a_0) = \beta^2(a_0) = \beta(b_0) = \\ =& \beta(a_m) = b_m = a_{2m \mod k}. \end{aligned}$$

So $2m = k+1$ and $m = \frac{k+1}{2}$. Therefore, k is odd number and $\beta(a_i) = b_i = a_{i+(k+1)/2 \mod k} = \alpha^{(k+1)/2}(a_i)$ for all $i \in \mathbb{Z}_k$.

If α forms the cycle $\bar{a}$, then α contains this cycle and by previous reasoning, $\beta = \alpha^{(n+1)/2}$, since here $A = S(n)$ and $k = n$. On the other hand, if $\beta = \alpha^{(n+1)/2}$, then $\beta^2(a_i) = \alpha^{n+1}(a_i) = \alpha(a_i)$ for all $i \in \mathbb{Z}_n$, thus $\beta^2 = \alpha$. □

Corollary 11. *If $\alpha \in Var(n)$ contains an odd number of nonequivalent cycles of even lengths, then its half iterate does not exist.*

Proof. Consider $G(\alpha)$. By assumption, there must exist some component G' of $G(\alpha)$ which contains a cycle c of even length for which there does not exist another component of $G(\alpha)$ of the same type as G'. By Proposition 10 and Remark 8, there does not exist a square root of G'. By Theorem 9, there does not exist a half iterate of α. □

For example, consider $\alpha = (2,3,4,1,2,3,8,9,10,11,9)$. Then $G(\alpha)$ has the form seen in Fig. 5.

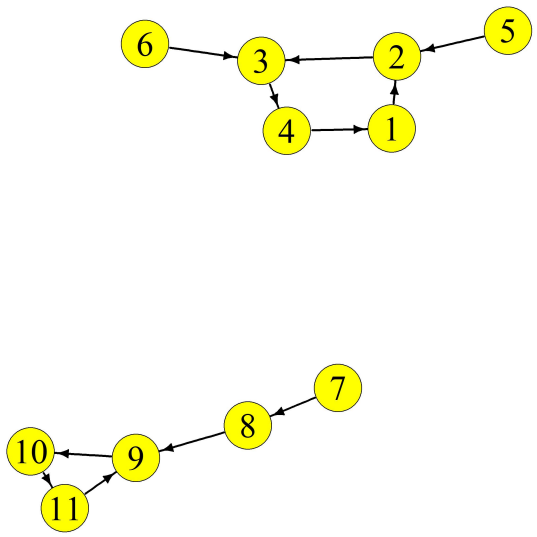

Fig. 5. $G(\alpha)$, where $\alpha = (2,3,4,1,2,3,8,9,10,11,9)$

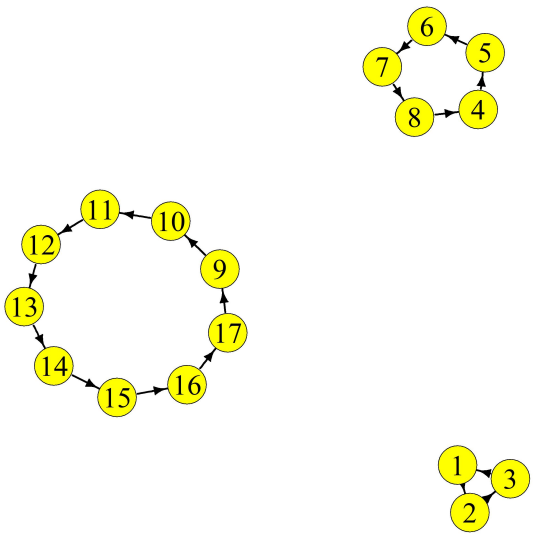

Fig. 6. $G(\alpha)$, where
$\alpha = (2,3,1,5,6,7,8,4,10,11,12,13,14,15,16,17,9)$

It can be seen that $G(\alpha)$ contains two components. One of these components contains one cycle of length 4, the second – one cycle of length 3. Thus α contains one cycle of even length. By Corollary 11, there does not exist a half iterate of α.

Corollary 12. *If there exists a half iterate of some $\alpha \in Var(n)$ and α contains a cycle $\bar{c}$ of even length k, then α contains another (nonequivalent with $\bar{c}$) cycle of the same length k.*

Proof. A straightforward conclusion from Theorem 9 and Proposition 10. □

Corollary 13. *If $\alpha \in Var(n)$ contains the cycles of odd lengths which sum to n, then there exists a half iterate of α.*

Proof. Assume that α contains the cycles of odd length $l_1,\ldots,l_m$ which are formed by functions $c_1,\ldots,c_m \in Var(n)$, respectively, and $l_1+\cdots+l_m = n$. Then $\alpha = c_1\circ\cdots\circ c_m$ and $c_i^{l_i+1} = c_i$ for all $i \in S(m)$. Let $L := LCM(l_1,\ldots,l_m)$ be the least common multiple of $l_1,\ldots,l_m$. Then $c_i^{L+1} = c_i$ for all $i \in S(m)$, hence $\alpha^{L+1} = \alpha$. Let $\beta := \alpha^{(L+1)/2}$. Then β is correctly defined, since $2|L+1$ and $\beta^2 = \alpha$. □

Let $\alpha = (2,3,1,5,6,7,8,4,10,11,12,13,14,15,16,17,9)$. Then $G(\alpha)$ is such as it is seen in Fig. 6.

Note that $\alpha \in Var(17)$ contains three cycles of length 3, 5 and 9. By Corollary 13, there exists a half iterate of α. Namely, take $L := LCM(3,5,9) = 45$ and $\beta := \alpha^{(L+1)/2} = = \alpha^{23} = (3,1,2,7,8,4,5,6,14,15,16,17,9,10,11,12,13)$. Then $\beta^2 = \alpha$.

The following result enables us to determine $\varphi(n)$ – the number of all half iterates of identity sequences $\alpha = (1,\ldots,n)$ for $n \in \mathbb{N}$.

Proposition 14.

$$\varphi(n) = 1 + \sum_{i=1}^{\left[\frac{n}{2}\right]} \frac{1}{i!} \prod_{j=1}^{i} \binom{n-2(j-1)}{2}$$

for all $n \in \mathbb{N}$, where $[x]$ denotes the greatest integer number smaller or equal to x.

Proof. Let $\alpha \in Var(n)$. Note that the number of all square roots of $G(\alpha)$ which have one pair of connected distinct vertices is equal to $\binom{n}{2}$. If we have an edge between two fixed vertices, then other two vertices can be chosen from $n-2$ vertices by $\binom{n-2}{2}$ ways. But if we have fixed distinct vertices v_1 and v_2 and later we will choose distinct vertices v_3 and v_4 then the final result will be the same as if we firstly chose vertices v_3 and v_4 and next vertices v_1 and v_2. Therefore, the number of all square roots of $G(\alpha)$ which have two pairs of connected distinct vertices is equal to $\frac{\binom{n}{2}\binom{n-2}{2}}{2}$. Similarly, the number of all square roots of $G(\alpha)$ which have k pairs of connected distinct vertices is equal to $\frac{1}{k!}\prod_{j=1}^{k}\binom{n-2(j-1)}{2}$. Moreover, we can have at most $[\frac{n}{2}]$ pairs of connected distinct

vertices. Thus $\varphi(n) = 1 + \sum_{i=1}^{\left[\frac{n}{2}\right]} \frac{1}{i!} \prod_{j=1}^{i} \binom{n-2(j-1)}{2}$, since by Proposition 1, $G(\alpha)$ is also its square root. □

Lemma 15.

$$\sum_{i=0}^{\left[\frac{n}{2}\right]} \binom{n}{2i} = \sum_{i=0}^{\left[\frac{n-1}{2}\right]} \binom{n}{2i+1} = 2^{n-1} \text{ for all } n \in \mathbb{N}.$$

Proof. Note that if n is an odd number, then $\sum_{i=0}^{\left[\frac{n}{2}\right]} \binom{n}{2i} = \sum_{i=0}^{\frac{n-1}{2}} \binom{n}{2i} = \sum_{i=0}^{\frac{n-1}{2}} \binom{n}{n-2i} = \sum_{i=0}^{\left[\frac{n-1}{2}\right]} \binom{n}{2i+1}$. Thus the statement is true for odd numbers.

Assume now that n is an even number. Then

$$2^{n-1} = \sum_{i=0}^{\frac{n}{2}-1} \binom{n-1}{2i} + \sum_{i=0}^{\frac{n}{2}-1} \binom{n-1}{2i+1} = \sum_{i=0}^{\frac{n}{2}-1} \binom{n}{2i+1} = \sum_{i=0}^{\left[\frac{n-1}{2}\right]} \binom{n}{2i+1},$$

since

$$\binom{n-1}{2i} + \binom{n-1}{2i+1} = \binom{n}{2i+1}.$$

Moreover,

$$2^n = \sum_{i=0}^{\left[\frac{n-1}{2}\right]} \binom{n}{2i+1} + \sum_{i=0}^{\left[\frac{n}{2}\right]} \binom{n}{2i}.$$

Therefore,

$$\sum_{i=0}^{\left[\frac{n}{2}\right]} \binom{n}{2i} = 2^n - 2^{n-1} = 2^{n-1} = \sum_{i=0}^{\left[\frac{n-1}{2}\right]} \binom{n}{2i+1}.$$

□

Corollary 16. $\varphi(n) \geq 2^{n-1}$.

Proof. Assume that $n \in \mathbb{N}$. Note that $\frac{1}{i!} \prod_{j=1}^{i} \binom{n-(j-1)2}{2} = \frac{1}{i!} \frac{n!}{(n-2i)!2^i} \geq \frac{n!}{(n-2i)!(2i)!} = \binom{n}{2i}$, since $(2i)! \geq i!2^i$ for all $i \in \left\{1, \ldots, \left[\frac{n}{2}\right]\right\}$. Therefore, by Proposition 14 and Lemma 15, $\varphi(n) \geq \sum_{i=0}^{\left[\frac{n}{2}\right]} \binom{n}{2i} = 2^{n-1}$. □

It follows from the above corollary that the problem of finding all half iterates of some sequences belongs to the complexity class EXPTIME.

It turns out that there exist sequences from $Var(n)$ for which the number of its half iterates may be even greater than $\varphi(n)$. Let $\psi(n)$ denote the number of all half iterates of a constant sequence $\alpha = (i, \ldots, i)$ of length n for some $i \in S(n)$. For a given sequence $\bar{k} = (k_1, \ldots, k_m)$ and $n \in \mathbb{N}$

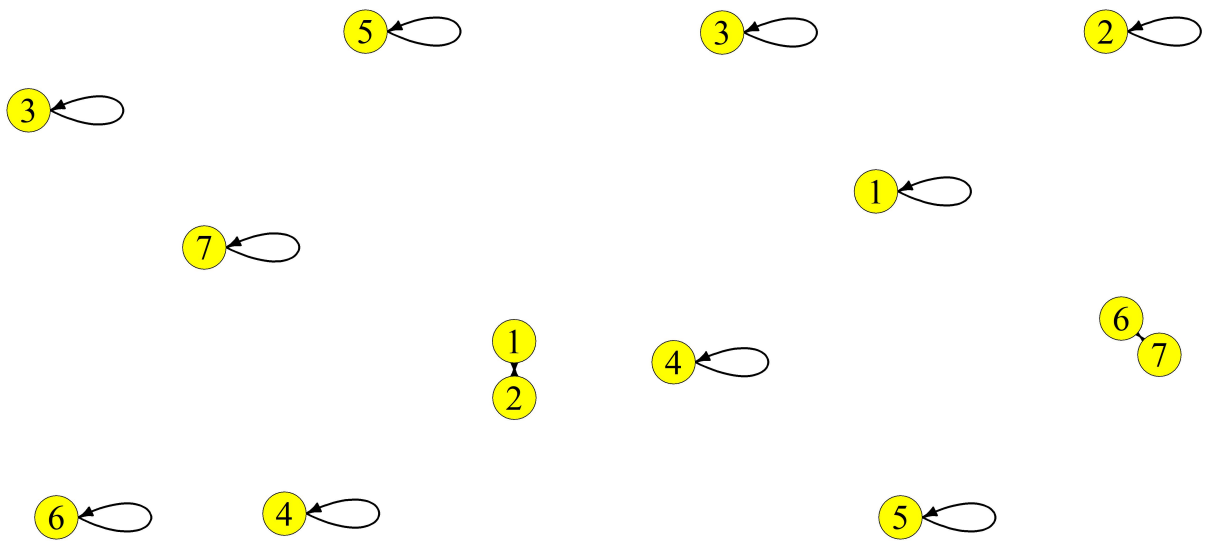

Fig. 7. Some square roots of graph $G(\alpha)$ which have one pair of connected distinct vertices

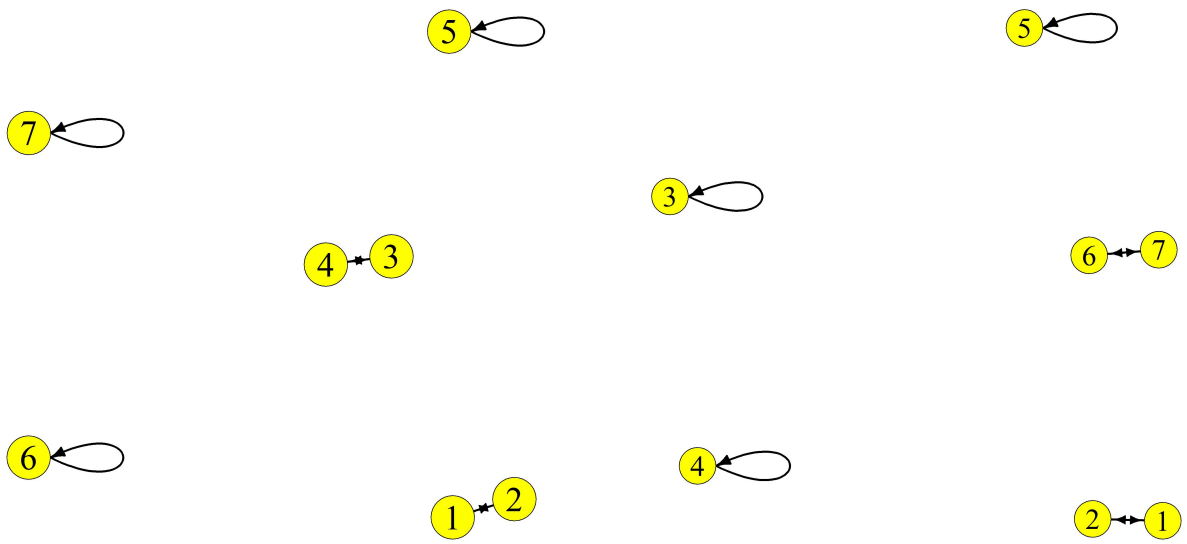

Fig. 8. Some square roots of graph $G(\alpha)$ which have two pairs of connected distinct vertices

let

$$P_{n,\bar{k}} := \prod_{i=1}^{m} \left(n - i - \sum_{j=1}^{i-1} k_j \right) \cdot \binom{n - i - \sum_{j=1}^{i-1} k_j - 1}{k_i}.$$

For a given $n \in \mathbb{N}$ and $m \in S(n-1)$ let

$$K_{n,m} := \Big\{ (k_1, \ldots, k_m) \in {\mathbb{N}_0}^m \colon k_1 \leq \cdots \leq k_m \wedge \wedge \sum_{i=1}^{m} k_i = n - m - 1 \Big\}.$$

For a given sequence $\bar{k} = (k_1, \ldots, k_n)$ let $R(\bar{k}) := \prod_{i=1}^{m} r_i!$, where $m := \max(\bar{k}) + 1$, $r_i := \#\{j \in S(n) \colon k_j = i - 1\}$ and $S(n) := \{1, \ldots, n\}$.

Proposition 17. *Under the above notations and assumptions it holds:*

$$\psi(n) = \begin{cases} 1 & \text{if } n \leq 2 \\ \sum_{m=1}^{n-1} \sum_{\bar{k} \in K_{n,m}} \frac{P_{n,\bar{k}}}{R(\bar{k})} & \text{otherwise} \end{cases}$$

Proof. Without loss of generality we can assume that $\alpha = \underbrace{(1, \ldots, 1)}_{n}$. Note that β is a half iterate of α or equivalently $G(\beta) = (V, E)$ is a square root of $G(\alpha)$ iff all of the following conditions are satisfied:

1. there exist $m \in S(n-1)$ and m vertices $v_1, \ldots, v_m \in V$ such that $(v_i, 1) \in E$ for all $i \in S(m)$;
2. for all $i \in S(m)$ there exist k_i vertices $v_{i,1}, \ldots, v_{i,k_i}$ such that $(v_{i,j}, v_i) \in E$ for all $j \in S(k_i)$ or such vertices do not exist (then we admit $k_i = 0$);
3. $1 + m + \sum_{i=1}^{m} k_i = n$.

If $n = 1$ or $n = 2$, then α is it only half iterate, hence $\psi(1) = \psi(2) = 1$.

Assume now that $n \geq 3$. Note that the first of vertices – v_1 – can be chosen by $n - 1$ ways. There can exist other k_1 vertices satisfying the second condition which can be chosen by $\binom{n-2}{k_1}$ ways, where $0 \leq k_1 \leq n - 2$. The second vertex v_2 can be chosen by $n - 2 - k_1$ ways and there can exist k_2 others vertices satisfying the second condition, which can be chosen by $\binom{n-3-k_1}{k_2}$ ways, where $0 \leq k_2 \leq n - 3 - k_1$, and so forth. Note also that if we firstly choose vertex v_1 with k_1 vertices satisfying the second condition and next we choose vertex v_2 with k_2 vertices satisfying the second condition, then we obtain the same configuration as if we firstly chose vertex v_2 with k_2 vertices satisfying the second condition and next vertex v_1 with k_1 vertices satisfying the second condition. Therefore, in order to calculate the number of unique configurations we can assume that $k_1 \leq \cdots \leq k_m$. Thus the numbers $k_1, \ldots, k_m \in K_{n,m}$. It may happen that $k_i = \cdots = k_j$ for some $1 \leq i < j \leq m$. Therefore, in order to obtain the number of all unique configurations we must divide $P_{n,\bar{k}}$ by $R(\bar{k})$. So for a given sequence $\bar{k} = (k_1, \ldots, k_m) \in K_{n,m}$, we can assume that

$k_1 \leq \cdots \leq k_m$ and there exist $\frac{P_{n,\bar{k}}}{R(\bar{k})}$ possible half iterates of α which have m vertices satisfying condition 1, and k_i vertices satisfying conditions 2 and 3 for all $i \in S(m)$. Hence for $m \in S(n-1)$ there exist $\sum_{\bar{k} \in K_{n,m}} \frac{P_{n,\bar{k}}}{R(\bar{k})}$ possible square roots of $G(\alpha)$ which have m vertices satisfying condition 1. Since $m \in S(n-1)$, we obtain thesis. □

Remark 18. Values of the function ψ increase very rapidly and more quickly than φ. Numerical examples show that for a given n there does not exist $\alpha \in Var(n)$ such that the number of all half iterates of α is greater than $\psi(n)$. The table below presents the first 20 values of functions φ and ψ. It is seen from the numerical data from Tab. 1, that 2^n is lower bound for ψ and $2^{n^{1.4}}$ is upper bound for ψ for $n \geq 5$.

III. ALGORITHMS FINDING ALL HALF ITERATES

In this section we deal with algorithms determining all half iterates of any function $\alpha \in Var(n)$ for $n \in \mathbb{N}$. In a description of the algorithms below the following notation will be used:

1. $|x|$ denotes the number of elements of set, list, sequence or vector x;
2. [] denotes an empty list or vector;
3. the operator = denotes the equality operator;
4. the operator := denotes the assignment operator;
5. for a given list, sequence or vector $v = (v_1, \ldots, v_n)$ let $set(v) := \{v_1, \ldots, v_n\}$;
6. ind(a,v) denotes the index of the first appearance of a in vector or list v; in the following algorithms indices are numbered from 1
7. $S(n) := \{1, \ldots, n\}$ for $n \in \mathbb{N}$.

We will also use the following notions.

Definition 19. Let $\alpha \in Var(n)$, $n \in \mathbb{N}$ and $G(\alpha) = (S(n), E)$. A sequence $\bar{p} = (a_1, \ldots, a_k)$ is called a *path* in graph $G(\alpha)$ iff $(a_i, a_{i+1}) \in E$ for each $i \in S(k-1)$ and $|set(\bar{p})| = k$ or $|set(\bar{p})| = k-1$ and there exists exactly one $j < k$ such that $a_j = a_k$.

If $\bar{p} = (a_1, \ldots, a_k)$ is a path and $a_j = a_k$ for some $j < k$ and cycles $c = (b_0, \ldots, b_{k-1-j})$ and $(a_j, a_{j+1}, \ldots, a_{k-1})$ are equivalent, then we say that path $\bar{p}$ *terminates* in the cycle c.

III. 1. Auxiliary algorithms

To begin some auxiliary algorithms needed in consecutive algorithms are given. Procedure *det_cyc(v)* determines all cycles in $G(v)$ and returns the list of them – L.

```
det_cyc(v)
n := |v|;
S := {1,...,n};
L:=[];
while S ≠ ∅ do
        a:=min(S); b:=v[a]; U:=[a]; c:=1;
        while (b ∉ U and b ∈ S) do
                a:=b; add a~at the end of U; c:=c
                    ↪ +1; b:=v[a]
        end do;
        if b in S then
                i:=ind(b,U);
                put w:=U[i .. c] and add b at the
                    ↪ end of it;
                push w into L;
        end if;
        S := S \ set(U)
end do;
return L;
```

Let k_i denote the number of executions of the internal loop in the ith execution of the external loop. Then if the external loop is executed r times, then $\sum_{i=1}^{r} k_i + r = n$. So time complexity of this algorithm is equal to $t_1 + rt_2 + nt_3 + (n-r)t_4 + st_5$, for some time t_1, t_2, t_3, t_4, t_5, $1 \leq s \leq r \leq n$, where s is number of cases when $b \in S$. Therefore, time complexity of this algorithm is $O(n)$.

Procedure *pen(v)* determines the set of pendants of $G(v)$ (vertices of $G(v)$ whose degree is 1).

```
pen(v)
n := |v|;
S := {1...,n};
R := S \ set(v);
return R;
```

It is a linear time algorithm.

Definition 20. A sequence $(a_0, \ldots, a_n)$ of vertices of $G(v)$ is called a *possible path* for square roots of $G(v)$ beginning at a k if $a_0 = k$, (a_i, a_{i+2}) is an edge of $G(v)$, for all $i \in \{0, \ldots, n-2\}$, $a_j = a_n$ and (a_{n-1}, a_{j+1}) is an edge of $G(v)$ for some $j < n$ and $a_s \neq a_t$ for all $s, t < n$.

Algorithm *ppfsrbp(v,k)* determines possible paths for square roots of $G(v)$ beginning at a k.

```
ppfsrbp(v,k)
S := ∅; n := |v|;
for b0 from 1 to n do
        initialize vector w of length n;
        a := k; b := b0; L := [];
        while a ∉ L do
                w[a] := b; add a at the end of L;
                c := a; a := b; b := v[c];
        end do;
        if b = w[a] then add a~at the end of L;
            ↪ S := S ∪ {L} end if;
end do;
return S;
```

Let k_i denote the number of executions of the internal loop 'while' in the ith execution of the external loop, r be the number of cases when the condition $b = w[a]$ is satisfied and $K := \sum_{i=1}^{n} k_i$. Then $1 \leq k_i \leq n$ for all $i \in \{1, \ldots, n\}$, $n \leq K \leq n^2$, $0 \leq r \leq n$ and time complexity of this algorithm is equal to $t_1 + nt_2 + (K+n)t_3 + Kt_4 + rt_5$ for some times t_1,t_2,t_3,t_4,t_5. Hence in the most optimistic case this algorithm is with time complexity $O(n)$ and in the most pessimistic case is $O(n^2)$.

Definition 21. A cycle $\bar{c} = (a_0, \ldots, a_{k-1})$ is *isolated* in $\alpha \in Var(n)$ iff $\alpha(k) \notin \bar{c}$ for all $k \in S(n) \setminus \{a_0, \ldots, a_{k-1}\}$.

Procedure *ppfsr(v)* determines the list L of possible paths for square roots of $G(v)$.

```
ppfsr(v)
n := |v|; S := {1,...,n}; L:=[]; R:=pen(v); Cc:=
    ↪ det_cyc(v);
for w in Cc do
        if w is isolated then R := R ∪ {w[1]};end if
            ↪ ;
end do;
for k in R do add ppfsrbp(v,k) at the end of L;
    ↪ end do;
return L;
```

Let $m = |Cc|$ and $k = |R|$. Then $1 \le m, k \le n$. Note that if $m = n$, then each cycle is isolated and if $m = 1$, then the sole cycle is not isolated. Moreover, checking whether the ith cycle w_i is isolated lasts $(r_i + 1)t_1 + t_2$, where t_1,t_2 are some times, $2 \le |w_i| \le n + 1$ and $0 \le r_i \le n - |w_i| + 1$ is the number of cases when $v[j] \notin w_i$. Commands from the first line of the algorithm are with time complexity $O(n)$. Time complexity of further part of the algorithm is equal to $\sum_{i=1}^{m}[(r_i + 1)t_1 + t_2] + k(f(n) + t_3) = \sum_{i=1}^{m} r_i t_1 + m(t_1 + t_2) + k(f(n) + t_3)$, where $f(n)$ denotes time complexity of $ppfsrbp(v, k)$. Therefore, in the most optimistic case this algorithm is with time complexity $O(n)$ and in the most pessimistic case time complexity is $O(n^3)$.

Procedure *bttssr(p,q)* checks whether paths p and q created according to the procedure *ppfsr(v)* can belong to the same square root of $G(v)$.

```
bttssr(p,q)
m := |p|; n := |q|; j:=0;
for i to m do
        if p[i] ∈ q then j:=ind(p[i], q); break; end
            ↪ if
end do;
bel:=(j = 0 or (j > 0 and p[i + 1] = q[j + 1]));
return bel;
```

Let r denote the number of cases when $p[i] \notin q$ and $v \in Var(N)$. Then $0 \le r \le m$, $1 \le j \le n$, $2 \le m, n \le N + 1$ and time complexity of this algorithm is equal to $t_1 + r \cdot n \cdot t_2 + sgn(m - r)(jt_2 + t_3) + t_4$. Therefore, in the most optimistic case time complexity of this algorithm is $O(1)$ and in the most pessimistic case its time complexity is $O(N^2)$.

Remark 22. Algorithm *bttssr* is based on the fact: If a path p of length m, a path q of length n are some paths created according to the procedure *ppfsr*; moreover, p and q have common element $k = p[i] = q[j]$ and k is their first common element, then $p[i + 1] = q[j + 1]$ iff $p[i \dots m] = q[j \dots n]$.

Proof. By procedure $apfsrbp$, $i < m$ and $j < n$. Note that $p[i + 2] = v\big[p[i]\big] = v\big[q[i]\big] = q[i + 2]$. Similarly, $p[i + 3] = v\big[p[i + 1]\big] = v\big[q[i + 1]\big] = q[i + 3]$, etc. By induction, the assertion is true. □

Procedure *del(S,T)* deletes every path p in a set S (T) such that $bttssr(p, q)$ is false for every path q in a set T (S). $S = ppfsrbp(v, k)$ and $T = ppfsrbp(v, l)$ for some $k, l \in S(N)$, where $N = |v|$.

```
del(S,T)
m := |S|; n := |T|; So := ∅; To := ∅;
for u in S do
        for v in T do
         if bttssr(u, v) then So := So ∪ {u};break;
             ↪ end if;
        end do;
```

Tab. 1. The first 20 values of functions φ, ψ, $\psi(n) - 2^n$ and $2^{n^{1.4}} - \psi(n)$

n	$\varphi(n)$	$\psi(n)$	$\psi(n) - 2^n$	$2^{n^{1.4}} - \psi(n)$
1	1	1	-1	1
2	2	1	-3	5.229065842
3	4	3	-5	22.20322980
4	10	10	-6	114.8805084
5	26	41	9	692.3050650
6	76	196	132	4798.190266
7	232	1057	929	37785.40661
8	764	6322	6066	3.34621624110^{5}
9	2620	41393	40881	3.30464446910^{6}
10	9496	293608	292584	3.61424695210^{7}
11	35696	2237921	2235873	4.35194523910^{8}
12	140152	18210094	18205998	5.74026996910^{9}
13	568504	157329097	157320905	8.25764063210^{10}
14	2390480	1436630092	1436613708	1.29053056810^{12}
15	10349536	13810863809	13810831041	2.18355163510^{13}
16	46206736	139305550066	139305484530	3.98735721510^{14}
17	211799312	1469959371233	1469959240161	7.83616351310^{15}
18	997313824	16184586405328	16184586143184	1.65309377510^{17}
19	4809701440	185504221191745	185504220667457	3.73457290510^{18}
20	23758664096	2208841954063318	2208841953014742	9.01552309110^{19}

```
end do;
for v in T do
        for u in S do
         if bttssr(v, u) then T_o := T_o ∪ {v}; break;
              ↪ end if;
        end do;
end do;
return [S_o, T_o]
```

Let $S := \{u_1, \ldots, u_m\}$, $T := \{v_1, \ldots, v_n\}$, $f(i,j)$ denote time complexity of $bttssr(u_i, v_j)$,

$$r(i) = \begin{cases} \min\{j \in S(n) \colon bttssr(u_i, v_j)\} & \text{if} \exists_{j \in S(n)} \colon \\ & bttssr(u_i, v_j) \\ n & \text{otherwise} \end{cases}$$

and

$$s(i) = \begin{cases} \min\{j \in S(m) \colon bttssr(u_j, v_i)\} & \text{if } \exists_{j \in S(m)} \colon \\ & bttssr(u_j, v_i) \\ m & \text{otherwise} \end{cases}$$

Then $1 \leq r(i) \leq n$ for all $i \in S(m)$, $1 \leq s(i) \leq m$ for all $i \in S(n)$, $1 \leq m, n \leq N$ and time complexity of this algorithm is approximately equal to $\sum_{i=1}^{m} \sum_{j=1}^{r(i)} f(i,j) + \sum_{i=1}^{n} \sum_{j=1}^{s(i)} f(j,i)$, since times of adding an element to a set is irrelevant in comparison with $f(i,j)$. Hence in the most optimistic case its time complexity is $O(1)$ and in the most pessimistic case it is with time complexity $O(N^4)$.

Algorithm *del_L(L)* executes *del(S,T)* for all distinct S, T from the list L created in the procedure *ppfsr(v)* for some $v \in Var(N)$.

```
del_L(L)
n := |L|; L_o := L;
for i from 1 to n−1 do for j from i+1 to n do
        L1:=del(L_o[i], L_o[j]);
        L_o[i]:=L1[1];
        L_o[j]:=L1[2]
end do; end do;
return L_o;
```

Note that $1 \leq n = |ppfsr(v)| \leq N$, where $N = |v|$. If $f(i,j)$ denotes time complexity of $del(L_o[i], L_o[j])$, then time complexity of this algorithm is equal to $t_1 + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \big(f(i,j) + t_2\big)$. So in the most optimistic case its time complexity is $O(1)$ and in the most pessimistic case – $O(N^6)$.

Proposition 23. *If $\alpha, \beta \in Var(n)$ for some $n \in \mathbb{N}$, then $\beta \circ \beta = \alpha$ iff for each set $S \in del\_L\big(ppfsr(\alpha)\big)$ there exists exactly one path $p \in S$ such that $\beta(p[i]) = p[i+1]$ for all $i \in \{1, \ldots, len\}$, where len is the length of path p.*

Proof. ($\Longrightarrow$) Assume that β is a half iterate of α. Fix any set $S \in del_L\big(ppfsr(\alpha)\big)$. Then all paths in S begin with a common element $k \in R$, where R is the set created in the procedure $ppfsr(\alpha)$. Thus $S = ppfsrbp(\alpha, k)$. Let $m \in \mathbb{N}$ be the smallest $n \in \mathbb{N}$ such that $\beta^n(k) = \beta^i(k)$ for some $0 \leq i < n$. Put $p := [k, \beta(k), \ldots, \beta^m(k)]$. Then $p \in S$ and $\beta(p[i]) = p[i+1]$ for all $i \in \{1, \ldots, len\}$, where len is the length of path p. Moreover, p is only path in S satisfying this condition, since β is a function.

($\Longleftarrow$) Assume that for each set $S \in del_L\big(ppfsr(v)\big)$ there exists exactly one path $p \in S$ such that $\beta(p[i]) = p[i+1]$ for all $i \in \{1, \ldots, len\}$, where len is the length of path p. Fix any $l \in S(n)$. Then there exists $k \in R$ and the smallest $m \in \mathbb{N}$ such that $l = \alpha^m(k)$, where R is the set created in the procedure $ppfsr(\alpha)$. Let p be the only path in $ppfsrbp(\alpha, k)$ such that $\beta(p[i]) = p[i+1]$ for all $i \in \{1, \ldots, len\}$. Then there exists a function w such that $p = [k, w(k), \ldots, w^M(k)]$ where $M \in \mathbb{N}$ is the smallest number $L \in \mathbb{N}$ such that $w^i(k) = w^L(k)$ for some $i < L$, $\alpha\big(w^j(k)\big) = w^{j+2}(k)$ for all $j \leq M-2$ and $\alpha\big(w^{M-1}(k)\big) = w^{i+1}(k)$. Therefore, $l = \alpha^m(k) = w^{2m}(k) = w^j(k)$, for some $j < M$, since $w^M(k) = w^i(k)$ for some $i < M$. Now assume that $j \leq M-2$. Then $\beta^2(l) = \beta^2(p[j+1]) = p[j+3] = w^{j+2}(k) = \alpha\big(w^j(k)\big) = \alpha(l)$. If $j = M-1$, then $\beta^2(l) = \beta^2(p[M]) = \beta(p[M+1]) = \beta\big(w^M(k)\big) = \beta\big(w^i(k)\big) = \beta(p[i+1]) = p[i+2] = w^{i+1}(k) = \alpha\big(w^{M-1}(k)\big) = \alpha(l)$. So β is a half iterate of α, since l was arbitrary. □

Procedure *mp(L,m)* matches up paths belonging to the first m sets from the list L created in the procedure $del\_L(ppfsr(v))$ in such a way that every two different paths belong to the same square root of $G(v)$ for some $v \in Var(N)$.

```
mp(L,m)
n := |L|; S_o := ∅;
if m ≤ n and ∅ ∉ L then
  if m = 1 then
      k := |L[1]|;
      for i from 1 to k do S_o := S_o ∪ {[i]} end do
    else
      S_i := mp(L, m − 1); nL:=L[m]; t := |nL|;
      for i to t do
         for ind in S_i do
          ok:=true
          for j to m−1 do
            if not bttssr(L[j][ind[j]], nL[i])
                 ↪ then
                 ok:=false;break;
            end if
          end do;
          if ok then
            add i at the end of ind and
                 ↪ S_o := S_o ∪ {ind}
          end if
        end do
      end do
  end if
end if;
return S_o
```

Under notation from the algorithm above we have $1 \leq k, n, t \leq N$ and $0 \leq m \leq n$. Let $f(L,m)$ denote time complexity of *mp(L,m)*, $g(L,m) = |mp(L,m)|$, $h(ind,i,j)$ denote time complexity of $bttssr(L[j][ind[j]], nL[i])$, $R(ind,i) = \{j \leq m-1 \colon \neg bttssr(L[j][ind[j]], nL[i])\}$ and

$$r(ind,i) = \begin{cases} \min(R(ind,i)) & \text{if } R(ind,i) \neq \emptyset \\ m & \text{otherwise} \end{cases}$$

If $\emptyset \in L$, then this algorithm is with time complexity $O(n)$, so in the most optimistic case its time complexity is $O(1)$ and in the most pessimistic case – $O(N)$.

Otherwise $f(L,1) = nt_1 + kt_2$, $g(L,1) = k$,

$$f(L,m) = nt_1 + f(L,m-1) + t_3 + \sum_{i=1}^{t} \sum_{ind \in S\_i} \sum_{j=1}^{\min\left(m-1, r(ind,i)\right)} h(ind,i,j) + g(L,m)t_4$$

and

$$g(L,m) = \sum_{i=1}^{t} \sum_{ind \in S\_i} \mathbf{1}_{r(ind,i)=m}(ind,i), \text{ where}$$

$$\mathbf{1}_{r(ind,i)=m}(ind,i) = \begin{cases} 1 & \text{if } r(ind,i) = m \\ 0 & \text{otherwise} \end{cases}$$

Hence $mp(L,1)$ is with time complexity $O(\max(n,k))$, that is, in the most optimistic case its time complexity is $O(1)$ and in the most pessimistic case – $O(N)$.

Note that if $g(L,m-1) = 0$, then $g(L,m) = 0$ and $f(L,m) = nt_1 + f(L,m-1) + t_3$. Therefore, in the most optimistic case, that is, if $g(L,2) = 0$, $n = k = t = 1$ then $f(L,m) = mt_1 + t_2 + (m-1)t_3 + \sum_{ind \in mp(L,1)} h(ind,1,1)$, hence time complexity of $mp(L,m)$ is $O(m)$.

In the most pessimistic case if $n = k = t = N$ and $g(L,m) = t^{m-1}k$ we have $f(L,m) = Nt_1 + f(L,m-1) + t_3 + N^{m+2}(m-1) + N^m t_4$ and $mp(L,m)$ is with time complexity $O(N^{m+2})$.

Procedure *as(p,q)* returns two boolean variable. The first is true iff paths p and q have common elements and the second is true iff they contain the same cycle.

```
as(p,q)
m := |p|; n := |q|; c:=false; sc:=false;
if p[m] ∈ q then
        i:=ind(p[m],q); c:=true; j:=ind(p[m],p);
        if p[j+1] = q[i+1] then sc:=true end if
else if p[m-1] ∈ q then
        i:=ind(p[m-1],q); c:=true;
        if p[m] = q[i+1] then sc:=true end if end
            ↪ if
end if
return [c, sc];
```

The algorithm above is very efficient, its time complexity is at most $O(N)$ for $\alpha \in Var(N)$, but its correctness demands a justification.

Proposition 24. *Let $p = (p_1,\ldots,p_m) \in S$, $q = (q_1,\ldots,q_n) \in T$ and $S,T \in del\_L(ppfsr(\alpha))$ for some sequence $\alpha \in Var(N)$. Then p and q have a common element iff $p_m \in q$ or $p_{m-1} \in q$ (alternatively $q_n \in p$ or $q_{n-1} \in p$).*

Moreover, if $p_m \in q$, then p and q contain equivalent cycles iff $p_{j+1} = q_{i+1}$, where $i := ind(p_m,q)$ and $j := ind(p_m,p)$. If $p_{m-1} \in q$, then p and q contain equivalent cycles iff $p_m = q_{i+1}$, where $i := ind(p_{m-1},q)$.

Proof. Let $i := \min\{k \in S(m) : p_k \in q\}$. Then $i < m$, $p_i = q_j$ for some $j < n$ and $\alpha^k(p_i) = \alpha^k(q_j) \in set(p) \cap set(q)$ for each $k \in \mathbb{N}_0$. In particular, $\alpha^{[\frac{m-i}{2}]}(p_i) \in set(p) \cap set(q)$. Hence $p_{m-1} \in q$ or $p_m \in q$, since

$$\alpha^{[\frac{m-i}{2}]}(p_i) = \begin{cases} p_m & \text{if } 2|m-i \\ p_{m-1} & \text{otherwise} \end{cases}$$

The reverse implication is obvious.

Assume now that, $p_m \in q$, $i := ind(p_m,q)$, $j := ind(p_m,p)$, $k := ind(q_n,q)$ and $p_{j+1} = q_{i+1}$. Then $c_1 = (p_j, p_{j+1},\ldots,p_{m-1})$ and $c_2 = (q_k, q_{k+1},\ldots,q_{n-1})$ are cycles in $G(\beta)$ for some $\beta \in Var(N)$ such that $\beta \circ \beta = \alpha$, if such β exists. Note that $q_{i+2} = \alpha(q_i) = \alpha(p_j) = p_{j+2}$, $q_{i+3} = \alpha(q_{i+1}) = \alpha(p_{j+1}) = p_{j+3}$ and so on. Hence $q_{i+l} = p_{j+l}$ for all $l < n-i$. In particular, $q_{n-2} = p_{j+n-2-i}$ and $q_{n-1} = p_{j+n-1-i}$. Therefore, $q_k = q_n = \alpha(q_{n-2}) = \alpha(p_{j+n-2-i}) = p_{j+n-i}$ and $q_{k+1} = \alpha(q_{n-1}) = \alpha(p_{j+n-1-i}) = p_{j+n-i+1}$ and hence $q_{k+l} = p_{j+n-i+l}$ for all $l \le i-k$. In particular, $p_m = p_j = q_i = p_{j+n-k}$, thus $m-j = n-k$ and the cycles c_1 and c_2 are equivalent. If $p_{m-1} \in q$, then proof is analogous to that above. □

Procedure *as_v(v,L)* adds v to the list L iff v has the same destination cycle as paths in L or L is an empty list. If $v[1]$ is equal to the first elements of paths belonging to a set S from the list L, then S is replaced by $S \cup \{v\}$, otherwise $\{v\}$ is pushed back into L. Procedure returns the (modified) list L and boolean variable $ch = true$ iff the list L was modified.

```
as_v(v,L)
N := |L|; L_o:=L; ch:=false;
if N > 0 then if as(v, L_o[1][1])[2] then
        ch:=true; is:=false;
        for i from 1 to N do S:=L_o[i];
            if S[1][1] = v[1] then
                is:=true; L_o[i] := S ∪ {v};
                    ↪ break;
            end if
        end do;
        if not is then add {v} at the end of L_o
            ↪ end if; end if
else L_o:=[{v}]; ch:=true end if;
return (L_o, ch);
```

Procedure *as_L(L0)* groups paths according to their destination cycle and its beginning. Its argument is the result of $del\_L(ppfsr(\alpha))$ for some $\alpha \in Var(n)$. The procedure returns a list of lists $L_1,\ldots,L_k$. Each list L_i contains sets $S_{i_1},\ldots,S_{i_{m_i}}$. Each of these sets contains paths beginning at the same vertex. Paths belonging to all sets from a given list L_i contain paths containing the same cycle.

```
as_L(L0)
LL:=[[]];
for S in L0 do for v in S do
        N := |LL|; Is:=false;
        for i from 1 to N do
            r:=as_v(v, LL[i]);
```

```
                if r[2] then LL[i]:=r[1]; Is:=true
                    ↪ ; break; end if
        end do;
        if not Is then add [{v}] at the end of LL
            ↪ end if
end do end do;
return LL;
```

$|L0|$ is equal to the number of pendants of $\alpha \in Var(n)$, possibly increased by the number of isolated cycles of α. Hence $1 \le |L0| \le n$. Moreover, $0 \le |S| \le n$ for each $S \in L0$. $N = |LL|$ cannot exceed the number of cycles of odd length, possibly increased by the number of pairs of cycles of the same length. Therefore, $1 \le N \le n + \binom{n}{2}$, since if α is a cycle of odd length – k, then $del\_L(ppfsr(\alpha)) = [\{\alpha^{(k+1)/2}\}]$ and if α contains n cycles of length 1, then number of pairs of these cycles is equal to $\binom{n}{2}$. For each $i \in \{1, \dots, N\}$ it holds $1 \le |LL[i]| \le n-1$. Moreover, as is algorithm with time complexity at most $O(n)$. Hence in the most optimistic case this algorithm is with time complexity $O(1)$ and in the most pessimistic case its time complexity is $O(n^5)$.

For a list of sets Ind procedure *prop_seqs1(Ind,m)* returns the set of sequences $(a_1, \dots, a_m)$ of length m such that

1. $a_i \in Ind[i]$
2. if $a_i \ne a_j$ for some $i, j \in S(m)$ then there does not exist $U \in Ind$ such that $\{a_i, a_j\} \subseteq U$

```
prop_seqs1(Ind,m)
S := ∅; n := |Ind|;
if m ≤ n then
 if m = 1 then for k in Ind[1] do S := S ∪ {[k]} end
     ↪ do
 else PS:=prop_seqs1(Ind, m−1);
  for u in PS do for k in Ind[m] do
   ok:=true;
   for U in Ind do
    if |U ∩ {k, set(u)}| > 1 then ok:=false; break;end
        ↪ if;
   end do
   if ok then psuh back k into u and S := S ∪ {u}
       ↪ end if
  end do end do
 end if
end if;
return S
```

The first argument of the procedure above is a list Ind of n sets, where n is equal to the number of cycles of an $\alpha \in Var(N)$. For any $i \in \{1, \dots, n\}$ the ith set contains such indices j of list $LL := as\_L(del\_L(ppfsr(\alpha)))$ that any path from $LL[j]$ has a common element with the ith cycle. So if α contains k cycles of the same length l as the ith cycle, then $k \cdot l \le N$ and the ith set from Ind can contain at most $(k-1) \cdot l \le N - l$ elements. For example, if the ith cycle is of the form $(1, 2, 3)$ and there exists another cycle of the same length $(4, 5, 6)$, then each of three cycles $(1, 4, 2, 5, 3, 6)$,$(1, 5, 2, 6, 3, 4)$,$(1, 6, 2, 4, 3, 5)$ can belong to $G(\beta)$ for a half iterate β of α. Summing up, $1 \le n \le N$ and $0 \le |Ind[i]| \le N - 1$. Assume that $g(m) := |prop\_seqs1(Ind, m)|$ and $f(m)$ denotes time complexity of this algorithm for m. Then $f(m) \in [1 + f(m-1); f(m-1) + g(m-1) \cdot (N-1) \cdot N]$, $f(1) \in [1; N-1]$, $g(1) \in [0; N-1]$ and $g(m) \in [0; g(m-1) \cdot (N-1) \cdot N]$. Hence $g(m) \in [0; (N-1) \cdot ((N-1) \cdot N)^{m-1}]$ and $f(m) \in \left[m; \sum_{i=0}^{m-1} (N-1) \cdot ((N-1) \cdot N)^i\right]$. Hence in the most optimistic case time complexity of this algorithm is $O(m)$ and in the most pessimistic case – $O(N^{2m-1})$.

For a list L from $as\_L(del\_L(ppfsr(\alpha)))$, for any sequence w from the set W, for any $ind \in Ind = mp(L, m)$, where m is the number of elements in L, and for any path p equal to $L[i][ind[i]]$ for some $i \in S(m)$, the procedure $part\_sq\_roots(L, W, n)$ creates sequences u of length n such that $u(p[j]) = p[j+1]$ for any $j \in S(len)$, where len is the length of path p, and $u(j) = w(j)$ for $j \in S(n) \setminus set(p)$. If $W = \emptyset$ then vectors w of length n are created such that $w(p[j]) = p[j+1]$ for any $j \in S(len)$, where len is the length of path p, $w(j) = 0$ for $j \in S(n) \setminus set(p)$.

```
part_sq_roots(L,W,n)
m := |L|; Ind:=mp(L, m); W_o := ∅;
if Ind ≠ ∅ then
 if W = ∅ then
  for ind in Ind do
    initialize vector w consisting of n zeros;
    for i to m do
      p:=L[i][ind[i]]; k := |p|;
      for j to k−1 do w[p[j]]:=p[j+1] end do
    end do;
    W_o := W_o ∪ {w}
  end do
 else
  for w in W do
    for ind in Ind do
      u:=w;
      for i to m do
        p:=L[i][ind[i]]; k := |p|;
        for j to k−1 do u[p[j]]:=p[j+1] end do
      end do;
      W_o := W_o ∪ {u}
    end do
  end do
end if; end if;
return W_o;
```

Assume that $\alpha \in Var(N)$. Then $1 \le m \le N-1$, $0 \le Ind \le N^m$ and in the most optimistic case time complexity of this algorithm is $O(m)$. If $W = \emptyset$, then in the most pessimistic case – $O(N^{m+2})$ and $|W_o| = |Ind| \le N^m$, otherwise $|W_o| = |W| \cdot |Ind|$, where W can be a set returned by this procedure for another list L, and in the most pessimistic case time complexity is $O(|W|N^{m+2})$.

III. 2. Procedures finding all half iterates

Now it is possible to go to the procedures finding all half iterates. This begins with the simplest procedure $sq\_roots(v)$. This procedure is treated as a point of reference and checks whether other procedures return the same results. Procedure *sq_roots(v)* is very primitive and takes a long time.

```
sq_roots(v)
```

```
n := |v|;
var:=Var(n); S := ∅;
for w in var do
  if v = w∘w then S := S ∪ {w} end if
end do;
return S
```

Time complexity of this algorithm is exponential: $n \cdot n^n = 2^{\ln_2(n)(n+1)}$.

Procedure *sq_roots1(v,opt)* has two options, it is based on Proposition 23 and returns the set of all half iterates of v. At first the algorithm creates a list of possible paths for square roots of $G(v)$, according to the procedure $ppfsr(v)$ described in the previous subsection. Next, some of these paths are deleted from this list and a list DP is created according to the algorithm del_L. In the next stage from each set from the list DP indices of paths which belong to the same square roots of $G(v)$ are selected according to the procedure $mp(DP, m)$, where $m = |DP|$. If result Ind of this procedure is empty, then a square root of v does not exist. Otherwise, from paths selected from DP by indices from Ind all half iterates of v are created. If $opt = 2$ then for each half iterate w this procedure determines values of w for arguments which do not occur in the previous steps. If $opt = 1$ the procedure can write the same value down to the memory of $w[i]$ many times.

```
sq_roots1(v,opt)
n := |v|; DP:=del_L(ppfsr(v));
m := |DP|; Ind:=mp(DP, m); W := ∅;
if Ind ≠ ∅ then
  for ind in Ind do
    initialize vector w of length n;
    if opt = 2 then U := ∅ end if;
    for i to m do
      p:=DP[i][ind[i]]; k := |p|;
      if opt = 2 then
        S_p := {p[1],...,p[k−1]}
      end if;
      if opt = 1 then
          for j to k−1 do w[p[j]]:=p[j+1] end do
        else for j to k−1 do
          if p[j] ∉ U then w[p[j]]:=p[j+1]
            else break
          end if
        end do;
        U := U ∪ S_p
      end if
    end do;
    W := W ∪ {w}
  end do
end if;
return W
```

Assume that $v \in Var(n)$. Then $0 \leq |Ind| \leq n^m$, $1 \leq m \leq n$, $2 \leq k \leq n$ and by reasoning from the previous subsection, time complexity of this algorithm is at least $O(n)$ and at most $O(n^{m+2})$.

Procedure *sq_roots2(v)* is designed for determining all half iterate of $v \in Var(n)$ if v has many cycles. Otherwise, its use is less profitable than the use of the previous algorithm *sq_roots1(v)*. Firstly, the algorithm initializes set $WW := \emptyset$ and determines list LL of possible paths and list Cyc of n_C cycles of v according to the procedures $as\_L(del\_L(ppfsr(v)))$ and $det\_cyc(v)$, respectively, described in the previous subsection. In the next step, if LL does not contain an empty sublist, then a list L_C of n_C sets is created such that for any $i \in \{1, \ldots, n\_C\}$ the ith set contains such indices j of list LL that any path from $LL[j]$ has a common element with the ith cycle. In order to check whether two paths have a common element it is used function 'as', described in the previous subsection. Next, if list L_C does not contain the empty set, then the set of sequences P_S is created according to the procedure $prop\_seqs1(L\_C, n\_C)$ described in the previous subsection. We shall show that $P\_S \neq \emptyset$ is a necessary condition for existence of half iterates of $v \in Var(n)$.

Proposition 25. *Let $\alpha \in Var(n)$, $c_1, \ldots, c_k$ be all cycles of α and $LL := as\_L(del\_L(ppfsr(\alpha))) = [L_1, \ldots, L_m]$. Define*

$$Ind_i := \{j \in S(m) \colon set(L_j[1][1]) \cap set(c_i) \neq \emptyset\}$$

for each $i \in S(k)$ and let

$$PS := \{(a_1, \ldots, a_k) \in \mathbb{N}^k \colon \forall_{i \in S(k)} \colon a_i \in Ind_i \\ \wedge |Ind_i \cap \{a_1, \ldots, a_k\}| = 1\}.$$

Then if there exists a half iterate of α, then $PS \neq \emptyset$.

Proof. Assume that a half iterate β of $\alpha \in Var(n)$ exists. Fix any cycle $c_i = (a_0, \ldots, a_{l_i-1})$ of length l_i and $x \in c_i$. There are two possibilities: either $\beta(x) \in c_i$ or $\beta(x) \in c_j$ for some $j \neq i$ such that the cycle $c_j = (b_0, \ldots, b_{l_i-1})$ has the same length as c_i. If $\beta(x) \in c_i$, then there exists $y \in S(n)$ and $l \geq l_i$ such that $q := (y, \beta(y), \ldots, \beta^l(y))$ is a path in $G(\beta)$ terminating in the cycle c_i' and belonging to L_j for some $j \in S(m)$, where $c_i' = (a_0, a_r, a_{2r \mod l_i}, \ldots, a_{(l_i-1)r \mod l_i})$, and $r := \frac{l_i+1}{2}$. If $\beta(x) \notin c_i$, then there exists $y \in S(n)$ and $l \geq 2l_i$ such that $q := (y, \beta(y), \ldots, \beta^l(y))$ is a path in $G(\beta)$ terminating in the cycle $c_{i,j}$ and $q \in L_j$ for some $j \in S(m)$, where $c_{i,j} = (a_0, b_s, a_1, b_{s+1 \mod l_i}, \ldots, a_{l_i-1}, b_{s+l_i-1 \mod l_i})$ and $s < l_i$. Either way, for each $i \in S(k)$ there exists $j \in S(m)$ such that $set(L_j[1][1]) \cap set(c_i) \neq \emptyset$, since $L_j[1][1])$ and q terminate in equivalent cycles. So for each $i \in S(k)$ we can choose $a_i \in Ind_i$.

Now suppose that for some $i \in S(k)$ there exists $i' \in S(k) \setminus \{i\}$ such that $a_{i'} \in Ind_i$ and $a_{i'} \neq a_i$. Then for some $y, l \in S(n)$ there exists a path $r := (y, \beta(y), \ldots, \beta^l(y))$ such that $r \in L_{a_{i'}}$ and $set(r) \cap set(c_i) \neq \emptyset$, since r and $L_{a_{i'}}[1][1]$ terminate in equivalent cycles and $set(L_{a_{i'}}[1][1]) \cap set(c_i) \neq \emptyset$. From the first part of the proof we know that for some $z, l' \in S(n)$ there exists a path $q := (z, \beta(z), \ldots, \beta^{l'}(z))$ such that $q \in L_{a_i}$ and $set(q) \cap set(c_i) \neq \emptyset$. By reasoning analogous with the proof of Proposition 24 it follows that q and r must terminate in equivalent cycles and it leads to a contradiction, since $a_{i'} \neq a_i$ and each cycle from L_{a_i} is not equivalent with any cycle in $L_{a_{i'}}$. □

Remark 26. Under assumptions and notations from the previous proposition there exist sequences $\alpha \in Var(n)$

for which $PS \neq \emptyset$ and there does not exist a half iterate of α. Consider $\alpha = (3,5,3,4,3,3,4,6,1,4)$. Then $G(\alpha)$ has two cycles $c_1 = (3)$, $c_2 = (4)$, $LL = [[\{[2,7,5,4,3,4],[2,10,5,4,3,4]\},$
$\{[7,3,4,3],[7,1,4,3,4],[7,5,4,3,4],[7,6,4,3,4]\},$
$\{[8,7,6,4,3,4],[8,10,6,4,3,4]\},$
$\{[9,7,1,4,3,4],[9,10,1,4,3,4]\},$
$\{[10,3,4,3],[10,1,4,3,4],[10,5,4,3,4],[10,6,4,3,4]\}]]$, $Ind_1 = Ind_2 = \{1\}$, $PS = \{(1,1)\}$ and α has no half iterate. Thus $P\_S \neq \emptyset$ is a necessary but insufficient condition for existence of half iterates of $\alpha \in Var(n)$.

If P_S is not the empty set, then for each sequence $p\_s \in P\_S$ set $W := \emptyset$ is initialized and for each $i \in \{1,\ldots,n\_C\}$ if $p\_s[i]$ did not appear earlier, then the result of the procedure $part\_sq\_roots(LL[p\_s[i]], W, n)$, described in the previous subsection, is written down to the set W. If $W \neq \emptyset$ and the first sequence from W does not contain 0 (i.e. for a given p_s there exists a half iterate of v), then $WW := WW \cup W$. Eventually the set WW is returned.

```
sq_roots2(v)
n := |v|; WW := ∅; esr:=true;
Cyc:=det_cyc(v); LL:=as_L(del_L(ppfsr(v)));
if LL does not contain an empty sublist then
  n_C := |Cyc|; L_C:=[]; n_LL := |LL|;
  for i to n_C do
    S := ∅; p:=Cyc[i];
    for j to n_LL do
      L:=LL[j]; q:=L[1][1];
      if as(p, q)[1] then S := S ∪ {j} end if
    end do;
    add S at the end of L_C;
  end do;
  if ∅ ∉ L_C then
    P_S:=prop_seqs1(L_C, n_C);
    if P_S ≠ ∅ then
      for p_s in P_S do
        W := ∅; v_j:=[];
        for i to n_C do
          j:=p_s[i];
          if j ∉ v_j then
            add j at the end of v_j;
            W:=part_sq_roots(LL[j], W, n)
          end if
        end do;
        if W ≠ ∅ and 0 ∉ W[1] then
          WW := WW ∪ W
        end if
      end do
    end if
  end if
end if;
return WW
```

Note that $1 \leq n\_C \leq n$, $0 \leq |P\_S| \leq (n-1)^{n\_C} n^{n\_C-1}$ and $1 \leq n\_LL \leq n + \binom{n}{2}$. So by reasoning from the previous subsection, if LL contains an empty sublist, then in the most optimistic case time complexity of this algorithm is $O(n)$ and in the most pessimistic case – $O(n^6)$. Time complexity of determining list L_C in the most optimistic case is $O(1)$ and in the most pessimistic case – $O(n^4)$. If $\emptyset \notin L\_C$ then time complexity of $prop\_seqs1(L\_C, n\_C)$ in the most optimistic case is $O(1)$ and in the most pessimistic case – $O(n^{2n-1})$. If $P\_S \neq \emptyset$, then time complexity of the further part of the algorithm is equal $|P\_S|n\_C$ multiplied by time complexity of $part\_sq\_roots(LL[j], W, n)$. So in the most optimistic case it is $O(1)$ and in the most pessimistic case – $O(n^{n^2})$.

III. 3. Comparison of procedures determining all half iterates

Before dealing with comparison of procedures determining all half iterates for any function $\alpha \in Var(n)$, some procedures needed for testing the time of work of the procedures finding all half iterates are described. Procedure *gen_rand_var(len,n)* generates n random functions belonging to $Var(len)$. Procedure *gen_rand_sq(len,n)* generates n random functions from $Var(len)$ for which there exists a half iterate.

In the first column of Tabs. 2 and 3 the lengths of sequences of samples of 100 random sequences which have a half iterate are seen. In the second column of these tables numbers of procedures are presented: 1 – *sq_roots(v)*, 2 – $sq\_roots1(v,1)$, 3 – $sq\_roots1(v,2)$, 4 – *sq_roots2(v)*. Average work times of these procedures for these samples are in the third and sixth columns of Tab. 2 and in the third and fifth columns of Tab. 3. Variances of the work time of these procedures are in the fourth and seventh columns of Tab. 2 and in the fourth and sixth columns of Tab. 3. Percentages of results which are the same as results of the procedure of reference *sq_roots(v)* for which this field in this column is blank we can see in the fifth and eighth columns of Tab. 2. Data from columns 3–5 of Tab. 2 and columns 3–4 of Tab. 3 concern variances generated by the procedure $gen\_rand\_var(len, 100)$; however, data from columns 6–8 of Tab. 2 and columns 5–6 of Tab. 3 concern variances generated by the procedure $gen\_rand\_sq(len, 100)$.

It can be seen in Tab. 2 that in each case all procedures return the same result as the procedure of reference – 1. Note that the average work times of the first procedure *sq_roots(v)* are much longer than other procedures even for relatively short sequences. It is especially seen for sequences for which there does not exist a half iterate. Therefore, for sequences of length greater than 6 only the remaining procedures 2 – 4 are compared. It can also be seen that the average work time of the fourth procedure *sq_roots2(v)* is the shortest in seven cases, namely for sequences generated by $gen\_rand\_sq(len, 100)$ and $len \in \{4,5,7,10,11,12,13\}$, while average time of the second procedure *sq_roots1(v,1)* is the shortest in two cases: for sequences generated by $gen\_rand\_sq(len, 100)$ and $len \in \{8,14\}$ and average time of the third procedure *sq_roots1(v,2)* is the shortest in two cases: for sequences generated by $gen\_rand\_sq(len, 100)$ and $len \in \{6,9\}$. We can also see that for $len \in \{12,14\}$ the average work times and variances of work times of these procedures working for sequences generated by $gen\_rand\_sq(len, 100)$ are much longer than in others cases. It is consistent with the fact that some sequences may have more than 2^{n-1} half iterates and

Tab. 2. Mean, variance of work time in $[s]$ of procedures 1 – sq_roots, 2 – sq_roots1 with option 1, 3 – sq_roots1 with option 2 and 3 – sq_roots2 and their compatibility with sq_roots for testing sets being the result of the procedures $gen\_rand\_var(len, 100)$ and $gen\_rand\_sq(len, 100)$ and for $len \in \{4, \ldots, 6\}$

		Variances			Variances with a half iterate		
len	Proc	Mean	Var	Comp	Mean	Var	Comp
4	1	0.00627	0.00006		0.00702	0.000081	
	2	0.00016	0.000003	100%	0.00031	0.000005	100%
	3	0.00031	0.000005	100%	0.00048	0.000008	100%
	4	0.00061	0.000014	100%	0	0	100%
5	1	0.1861	0.001455		0.16797	0.001576	
	2	0.00016	0.000003	100%	0.00079	0.000012	100%
	3	0.00016	0.000003	100%	0.00158	0.000023	100%
	4	0.00047	0.000007	100%	0.00076	0.000011	100%
6	1	26.32578	5.202791		25.4475	0.729389	
	2	0.00048	0.000008	100%	0.00405	0.000131	100%
	3	0.00046	0.000007	100%	0.00189	0.000032	100%
	4	0.00031	0.000005	100%	0.00219	0.00003	100%

others – only one half iterate. Therefore, the work times may differ very much, depending on numbers of half iterates quasi-randomly sequences of generated by procedure $gen\_rand\_sq(len, 100)$.

IV. ALGORITHMS FINDING ONE HALF ITERATE IF IT EXISTS

Additional algorithms to finding one half iterate, if it exists, were invented for the sake of the long work time of procedures described in the previous section. Procedures from this section are based on similar ideas, therefore part of the auxiliary procedures used in these procedures is the same as in the previous section. Below only additional procedures which do not occur earlier are listed.

IV. 1. Additional auxiliary procedures

Procedure *mp_s(L)* works as the procedure *mp(L)* but it finds one adjustment.

```
mp_s(L)
m := |L|; N:=[]; esr:=true;
if 0 < m then
  for i to m do add |L[i]| at the end of N end do
end if;
Initialize sequence ind_o consisting of m zeros;
if 0 ∉ N and 0 < m then
  Initialize sequence ind = (1...,1) of length m;
  if 1 < m then
    i:=1; L_p:=[];
    while i ≤ m do
      q:=L[i][ind[i]];
      ok:=true;
      for p in L_p do
        if not bttssr(p, q) then ok:=false; break
            ↪ end if
      end do
      if ok then add q at the end of L_p; i:=i+1
        else if ind[i] < N[i] then ind[i]:=ind[i
            ↪ ]+1
        else while ind[i] = N[i] and 1 < i
          do i:=i−1 end do;
          if ind[i] < N[i]
            then ind[i]:=ind[i]+1; ind[i+1 .. m
                ↪ ]:=1;
            if 1 < i then L_p:=L_p[1 .. i−1]
              else L_p:=[] end if
            else esr:=false; break;
          end if
        end if
      end if
    end do
  end if
  else esr:=false
end if;
if esr then ind_o:=ind end if;
ind_o
```

For a list L being some list from *as_L(del_L(ppfsr(v)))* and for a vector w, the procedure *part_sq_r(L,w)* looks for a path p in $L[i]$ for any $i \in \{1, \ldots, m\}$, where m is the number of elements of L, such that $p[j]$ does not belong to the set U of elements of paths q determined in such a way in the previous steps for $k < i$ or $w[p[j]] = p[j+1]$ for any $j \in \{1, \ldots, len\}$, where len is the length of path p. If $p[j] \notin U$, then the procedure writes down $w[p[j]] := p[j+1]$, thus changes the vector w.

```
part_sq_r(L,w)
n := |w|; esr:=true; m := |L|; N:=[];
if m > 0 then for i to m do add |L[i]| at the end of
    ↪ N; end do end if;
if 0 ∉ N and m > 0 then
  if m > 1 then
    initialize vector ind = (1,...,1) of length m; i
        ↪ :=1;
    initialize vector U of length (m+1); U[1] := ∅;
    while i ≤ m do
```

Tab. 3. Mean, variance of work time in $[s]$ of procedures 2 – sq_roots1 with option 1, 3 – sq_roots1 with option 2 and 3 – sq_roots2 for testing sets being the result of the procedures $gen\_rand\_var(len, 100)$ and $gen\_rand\_sq(len, 100)$ and for $len \in \{7, \ldots, 14\}$

		Variances		Variances with a half iterate	
len	Proc	Mean	Var	Mean	Var
7	2	0.00063	0.000024	0.00376	0.000075
	3	0.00046	0.000007	0.00435	0.000118
	4	0.00047	0.000007	0.00283	0.000042
8	2	0.00032	0.000005	0.05097	0.188292
	3	0.00061	0.000009	0.05624	0.2092
	4	0.00064	0.00001	0.05138	0.197869
9	2	0.00079	0.000017	0.00843	0.000343
	3	0.00062	0.000014	0.00719	0.000195
	4	0.00109	0.000021	0.00751	0.000269
10	2	0.0011	0.000016	0.01298	0.001068
	3	0.00079	0.000012	0.0108	0.0009
	4	0.00061	0.000009	0.00732	0.000268
11	2	0.00126	0.000033	0.16857	0.482426
	3	0.00092	0.000013	0.17836	0.525644
	4	0.00063	0.00001	0.12151	0.298582
12	2	0.00032	0.000005	3.84986	1333.825
	3	0.0011	0.000021	4.14747	1555.371
	4	0.00139	0.00002	3.12142	891.8630
13	2	0.00157	0.000037	0.16018	0.29617
	3	0.00108	0.000016	0.16141	0.309504
	4	0.00173	0.000024	0.03701	0.009897
14	2	0.00093	0.000014	3.63565	1166.562
	3	0.00122	0.000017	3.88134	1337.609
	4	0.00206	0.000044	3.77739	1360.451

```
      a_v:=L[i][ind[i]]; k:=|a_v|; ok:=true;
      for j to k-1 do
        if a_v[j] ∉ U[i] then w[a_v[j]]:=a_v[j+1]
          else if w[a_v[j]] ≠ a_v[j+1] then ok:=false
              ↪ ; break end if
        end if
      end do;
      if ok then U[i+1]:=U[i] ∪ set(a_v); i:=i+1
        else if ind[i] < N[i] then ind[i]:=ind[i
            ↪ ]+1
          else while ind[i] = N[i] and i > 1 do i
              ↪ :=i-1 end do;
            if ind[i] < N[i] then ind[i]:=ind[i
                ↪ ]+1; ind[i+1 .. m]:=1
              else esr:=false; break;
            end if
        end if
      end if
    end do
    else a_v:=L[1][1]; k:=|a_v|;
      for j to k-1 do w[a_v[j]]:=a_v[j+1] end do;
  end if
  else esr:=false
end if;
return (w, esr);
```

Assume that $v \in Var(n)$. $0 \leq m \leq n$ and $0 \leq N[i] \leq n$ for each $i \in S(m)$. Algorithms *mp_s(L)* and *part_sq_r(L,w)* work similarly: If the condition $0 \notin N$ and $m > 1$ is satisfied, then in the loop 'while $i \leq m$' if ok:=true, then i is increased by 1, otherwise ind is changed in the same manner. Hence their time complexity is similar. In the most optimistic case (if $m = 0$) these algorithms are with time complexity $O(1)$. Let ind_o be the last state of ind after exit from loop 'while' in these algorithms. Then the condition $ok = false$ is satisfied at most $\sum_{i=1}^{m-1} (ind_o[i] - 1) \cdot \prod_{j=i+1}^{m} N[j] + ind_o[m]$ times. The number of cases when $ok = true$ is not greater than $N[1] \cdot \ldots \cdot N[m-1] \cdot m$. Either way in the most pessimistic case these algorithms are exponential time.

For a path p, for a subset P of a set of pendants, and for a list L of sets of paths, the procedure *fp(p,P,L)* finds the index $k \in \{1, \ldots, n\}$, where n is the number of elements of L, and the path q such that if $p[2] \in P$ then $q = p[2..|p|]$ and $q \in L[k]$, otherwise $p = q[2..|q|]$ and $q \in L[k]$,

```
fp(p, P, L)
q:=[]; k:=0; n := |L|;
if p[2] ∈ P then
  r := p[2..|p|];
  for i to n do
    S:=L[i];
    if r ∈ S then
      q:=r; k:=i; break;
    end if
  end do
  else for i to n do
    S:=L[i];
    add S[1][1] at the beginning of p;
    if p ∈ S then q:=p; k:=i; break; end if
  end do
end if;
return (k, q);
end proc;
```

The procedure *fps(p,P,L)* works similarly to the procedure *fp(p,P,L)*, but *fps(p,P,L)* finds all paths q and indices k such that $p = q[2..|q|]$ and $q \in L[k]$ or path $q = p[2..|p|]$ and index k such that $q \in L[k]$.

```
fps(p, P, L)
L_o:=[]; n := |L|;
if p[2] ∈ P then
  r := p[2..|p|];
  for i to n do
    S:=L[i];
    if r ∈ S then
      add [i, r] at the end of L_o; break;
    end if
  end do
  else for i to n do
    S:=L[i];
    add S[1][1] at the beginning of p;
    if p ∈ S then
      add [i, p] at the end of L_o;
    end if
  end do
end if;
L_o
end proc;
```

If $\alpha \in Var(N)$, then both algorithms fp(p, P, L) and fps(p, P, L) are in the most pessimistic case with time complexity $O(N^2)$ and in the most optimistic case – $O(1)$.

The procedures *part_sq_r1(P0,L,w)* and *part_sq_r2(P0,L,w)* work similarly to the procedure *part_sq_r(L,w)* but for each path p found in a similar way as in the procedure *part_sq_r* these procedure additionally find paths according to the procedures *fp* and *fps*, respectively. If it is impossible to find paths which can belong to a square root in the same way, then these procedures activate the procedure *part_sq_r(L,w)*. The idea behind these algorithms is based on the fact that each path belonging to a square root $G(\beta)$ of $G(\alpha)$ for any $\alpha \in Var(n)$ consists of two interspersed paths from $G(\alpha)$. In particular, if a path p from $G(\beta)$ belongs to $as\_L(del\_L(ppfsr(\alpha)))$ and consists of paths belonging to distinct components from $G(\alpha)$ of the same type, then there exists other path q belonging to $as\_L(del\_L(ppfsr(\alpha)))$ such that $q = p[2..|p|]$ or $p = q[2..|q|]$.

```
part_sq_r1(P0, L, w)
n := |w|; esr:=true; m := |L|;
P := {L[1][1][1],...,L[m][1][1]} ∩ P0;
N:=[]; w_o:=w;
if 0 < m then
  for i to m do add |L[i]| at the end of N; end do
end if;
if 0 ∉ N and 0 < m then
  if 1 < m then
    initialize sequence ind := (1,...,1) of length m;
    i:=1;
    initialize lists U and R consisting of m+1
        ↪ empty sets;
    esr1:=true;
    while i ≤ m do
      if i ∉ R[i] then
      n_i:=i+1;
      while n_i ∈ R[i] and n_i ≤ m do n_i:=n_i+1;end
          ↪ do;
      p:=L[i][ind[i]]; iip :=' p[1] ∈ P'; q:=[];i_q
          ↪ :=0;
      if iip then res:=fp(p,P,L);i_q:=res[1];q:=
          ↪ res[2];end if;
      k := |p|;
      if |q| > k and i_q>i and i_q ∉ R[i] then p:=q;k
          ↪ :=k+1; end if;
      c:=true;
      for j from 1 to (k−1) do  if p[j] ∉ U[i] then
          ↪ w_o[p[j]]:=p[j+1];
        else if w_o[p[j]] ≠ p[j+1] then c:=false;
            ↪ break;end if;
      end if;  end do;
      if c then U[n_i] := U[i] ∪ set(p);
        if iip  and i_q>i then R[n_i] := R[i] ∪ {i_q};
          else  R[n_i]:=R[i];
        end if;
        i:=n_i;
        else if ind[i]<N[i] then ind[i]:=ind[i]+1;
          else p_i:=i−1;
            while p_i>0 and (ind[p_i]=N[p_i] or
                ↪ p_i ∈ R[i])) do
              p_i:=p_i−1;
            end do;
            if p_i>0  then i:=p_i;
              ind[i]:=ind[i]+1;ind[(i+1)..m]:=1;
              else esr1:=false;break;
            end if;
        end if;
      end if;
      else n_i:=i+1;
        while n_i ∈ R[i] do n_i:=n_i+1; end do;
        R[n_i]:=R[i];U[n_i]:=U[i];i:=n_i;
      end if;
    end do;
  if not esr1 then (w_o,esr):=part_sq_r(L,w); end
      ↪ if;
  else  p:=L[1][1];k := |p|;
    for j from 1 to (k−1) do w_o[p[j]]:=p[j+1];
        ↪ end do;
  end if;
  else esr:=false;
end if;
return (w_o,esr);

part_sq_r2(P0,L,w);
n := |w|;
esr:=true;#'esr=true iff there exists square root'
m := |L|;
P := {L[1][1][1],...,L[m][1][1]} ∩ P0;
N:=[];  w_o:=w;
```

```
if m>0 then for i from 1 to m do
  add |L[i]| at the end of N;
end do;end if;
if 0 ∉ N and m>0 then
  if m>1 then i:=1;
    initialize sequence ind := (1,...,1) of length m;
    initialize lists U and R of m+1 empty sets;
    esr2:=true;
    while i ≤ m do
      if i ∉ R[i] then n_i:=i+1;
        while n_i ∈ R[i] and n_i ≤ m do n_i:=n_i+1;
            ↪ end do;
        p:=L[i][ind[i]];  iip :=' p[1] ∈ P';  k := |p|;  c
            ↪ :=true;
        for j from 1 to (k−1) do  if p[j] ∉ U[i] then
          w_o[p[j]]:=p[j+1];
          else if w_o[p[j]] ≠ p[j+1] then c:=false;
              ↪ break;end if;
         end if; end do;
        if c then  U_q := ∅; R_q := ∅;
          if iip then Q:=fps(p,P,L);  if |Q| > 0
              ↪ then
            for u in Q do i_q:=u[1];q:=u[2];
              if i_q>i and |q| < k then
                  ↪ R_q := R_q ∪ {i_q}; end if;
              if i_q>i and |q| > k and q[1] ∉ U[i] then
                R_q := R_q ∪ {i_q};
                U_q := U_q ∪ {q[1]};w_o[q[1]]:=q[2];
              end if;
            end do;
          end if;end if;
          U[n_i] := U[i] ∪ set(p) ∪ U_q;
          R[n_i] := R[i] ∪ R_q;  i:=n_i;
          else if ind[i]<N[i] then ind[i]:=ind[i
              ↪ ]+1;
            else p_i:=i−1;
              while p_i>0 and (ind[p_i]=N[p_i] or
                  ↪ p_i ∈ R[i])) do
                p_i:=p_i−1;
              end do;
              if p_i>0  then i:=p_i;
                ind[i]:=ind[i]+1;ind[(i+1)..m]:=1;
                else esr2:=false;break;
              end if;
          end if;
        end if;
        else n_i:=i+1; while n_i ∈ R[i] do n_i:=n_i
            ↪ +1; end do;
          R[n_i]:=R[i];U[n_i]:=U[i];i:=n_i;
      end if;
    end do;
    if not esr2 then (w_o,esr):=part_sq_r(L,w) end
        ↪ if;
    else  p:=L[1][1];k := |p|;
      for j from 1 to (k−1) do w_o[p[j]]:=p[j+1];
          ↪ end do;
  end if;
  else esr:=false;
end if;
return (w_o,esr);
```

Similarly to the procedure *part_sq_r(L,w)*, also the above two algorithms are in the most optimistic case with time complexity $O(1)$ and in the most pessimistic case they are exponential time.

The procedure *prop_seqs(Ind)* works similarly to the procedure *prop_seqs1* from section 3.1, but the procedure *prop_seqs* determines sequences of length m, where m is the number of elements of Ind and *prop_seqs* uses Cartesian product of sets from Ind.

```
prop_seqs(Ind)
S := ∅;
T:= Cartesian product of sets from Ind;
for u in T do ok:=true;
  for U in Ind do if |U ∩ set(u)| > 1 then ok:=false;
      ↪ break; end if end do;
  if ok then S := S ∪ {u} end if
end do;
return S;
```

The procedure *init_prop_seq(Ind, N)* determines the first proper sequence, i.e. satisfying the properties satisfied by sequences determined by the procedures *prop_seqs* and *prop_seqs1* if such sequence exists.

```
init_prop_seq(Ind,N);
n := |N|;
initialize sequence ind := (1,...,1) of length n;
i:=2;  eps:=true;#'eps=true iff there exists
    ↪ proper sequence';
while i ≤ n do cor:=true;
  for U in Ind do if
      ↪ |{Ind[1][ind[1]],...,Ind[i][ind[i]]} ∩ U| > 1 then
    cor:=false;break;
  end if;end do;
  if cor then i:=i+1;
    else if ind[i]<N[i] then ind[i]:=ind[i]+1;
      else while i>0 and  ind[i]=N[i]  do i:=i−1;
          ↪ end do;
        if i>0 then ind[i]:=ind[i]+1; ind[(i+1)..n
            ↪ ]:=1;
          else eps:=false; break;
        end if;
    end if;
  end if;
end do;
return [eps,ind];
```

The procedure *next_prop_seq(Ind, N,ind)* determines the next (after ind) sequence satisfying the properties satisfied by sequences determined by the procedures *prop_seqs* and *prop_seqs1* if such a sequence exists.

```
next_prop_seq:=proc(Ind, N, ind)
n := |Ind|; enps:=true; i:=n; ind_o:=ind;
if ind_o[i] < N[i] then ind_o[i]:=ind_o[i]+1
  else while i > 0 and ind_o[i] = N[i] do i:=i−1
      ↪ end do;
  if i > 0 then ind_o[i]:=ind_o[i]+1; ind_o[i+1 ..
      ↪ n]:=1
    else enps:=false
  end if;
end if;
if enps then while i ≤ n do cor:=true;
  for U in Ind do if
      ↪ |{Ind[1][ind[1]],...,Ind[i][ind[i]]} ∩ U| > 1 then
    cor:=false; break;
  end if end do;
  if cor then i:=i+1
    else if ind_o[i] < N[i] then ind_o[i]:=ind_o[i
        ↪ ]+1
      else while i > 0 and ind_o[i] = N[i] do i:=i
          ↪ −1 end do;
        if i > 0 then ind_o[i]:=ind_o[i]+1; ind_o[
            ↪ i+1 .. n]:=1
```

```
          else enps:=false; break;
        end if;
    end if;
  end if
end do; end if;
return [enps, ind_o]
```

The above algorithms *prop_seqs(Ind)*, *init_prop_seq(Ind, N)* and *next_prop_seq(Ind, N,ind)* are in the most optimistic case with time complexity $O(1)$ and in the most pessimistic case they are exponential time.

IV. 2. Procedures finding one half iterate

Each of the following procedures returns one half iterate if it exists and a zero-vector otherwise.

The procedure *sq_root(v)* invokes the procedure *part_sq_r(L,w)* for $L = del\_L(ppfsr(v))$ and zero-vector $w = [0, \ldots, 0]$ of length the same as the length of vector v.

```
sq_root(v)
n := |v|;
initialize sequence w of length n zeros;
L:=del_L(ppfsr(v));
w_o:=part_sq_r(L, w)[1];
return w_o;
```

The procedure *sq_root1(v)* has the most compact form. Apart from the procedures *del_L(ppfsr(v))*, it uses the procedure *mp_s* on the result of the former. If the result of *mp_s* does not contain 0, then a half iterate exists, otherwise it does not exist.

```
sq_root1:=proc(v)
n := |v|;
initialize sequence w of n zeros;
L:=del_L(ppfsr(v));
ind:=mp_s(L); m := |ind|;
if 0 ∉ ind then
  for i to m do
    p:=L[i][ind[i]]; k := |p|;
    for j to k−1 do w[p[j]]:=p[j+1] end do
  end do
end if;
return w;
```

The procedure *sq_root2(v,opt)* firstly determines *as_L(del_L(ppfsr(v)))* and writes down its result to the variable LL, w is initiated as zero-vector of length of vector v. If LL does not contain an empty list then the procedure creates list L_C in the following way: for any cycle p in v the procedure creates a set of indices j of lists from the list LL such that cycles of p and $q := LL[j][1][1]$ have a common element, i.e. $as(p, q)[1]$ is true; next, this set is added to the list L_C. If L_C contains the empty set then the half iterate of v does not exist. Otherwise, the procedure creates the set P_S in the following way: if $opt = 1$, then the procedure $P\_S := prop\_seqs(L\_C)$ is used and if $opt = 2$, then $P\_S := prop\_seqs1(L\_C, n\_C)$, where n_C is the number of cycles in v. In the next steps for each vector p_s in P_S the procedure computes $part\_sq\_r(LL[j], w)$, for each unique element j of vector p_s. If $w \circ w = v$ the loop is interrupted.

```
sq_root2(v, opt)
n := |v|;
initialize vector w_o of n zeros;
esr:=true; Cyc:=det_cyc(v);
LL:=as_L(del_L(ppfsr(v)));
if [] ∉ LL then
  n_C := |Cyc|; L_C:=[];
  n_LL := |LL|;
  for i to n_C do
    S := ∅; p:=Cyc[i];
    for j to n_LL do
      L:=LL[j]; q:=L[1][1];
      if as(p, q)[1] then
        S := S ∪ {j};
      end if
    end do;
    add S at the end of L_C;
  end do;
  if ∅ ∈ L_C then
    esr:=false
    else if opt = 1 then
      P_S:=prop_seqs(L_C)
      else P_S:=prop_seqs1(L_C, n_C)
    end if;
    if P_S ≠ ∅ then
      for p_s in P_S do
        initialize sequence w of n zeros;
        v_j:=[];
        for i to n_C do j:=p_s[i];
          if j ∉ v_j then
            add j at the end of v_j;
            res:=part_sq_r(LL[j], w);
            if res[2] then
              w:=res[1]
              else break
            end if
          end if
        end do;
        if w ∘ w = v then
          w_o:=w; break
        end if
      end do
      else esr:=false
    end if
  end if
end if;
return w_o;
end proc
```

The procedure *sq_root3(v,opt)* works similarly to the procedure *sq_root2(v,opt)*, but *sq_root3* determines sequences by the procedures $init\_prop\_seq(L\_C, N)$ and $next\_prop\_seq(L\_C, N, ind)$ which have the same properties as sequences in the set P_S, instead of creating the set P_S of all such sequences. Moreover, the procedure *sq_root2* uses only the function *part_sq_r*, and the procedure *sq_root3* uses this procedure only if $opt = 1$, otherwise if $opt = 2$ it uses the procedure *part_sq_r1*, otherwise it uses the procedure *part_sq_r2*.

```
sq_root3:=proc(v, opt)
n := |v|;
initialize sequence w_o consisting of n zeros
esr:=true; Cyc:=det_cyc(v);
P0:=pen(v); LL:=[];
LL:=as_L(del_L(ppfsr(v)));
if [] ∉ LL then
  n_C := |Cyc|; L_C:=[];
  n_LL := |LL|;
  for i to n_C do
    S := ∅; p:=Cyc[i];
    for j to n_LL do
      L:=LL[j]; q:=L[1][1];
```

Tab. 4. Means and variances of work times in $[s]$ of tested procedures for various testing sets being the result of the procedure $gen\_rand\_sq(len, 100)$ for $len \in \{10, 20, 30, 40, 50, 60\}$

Proc	len	Mean	Var	len	Mean	Var
1	10	0.00203	0.000028	20	0.01076	0.000141
2		0.00248	0.000033		0.01284	0.000169
3		0.11591	1.212388		0.02311	0.005965
4		0.00375	0.000094		0.01344	0.000253
5		0.0036	0.000044		0.01233	0.00009
6		0.0033	0.000056		0.0136	0.000189
7		0.00377	0.000075		0.01314	0.000152
1	30	0.03972	0.00786	40	0.06077	0.004162
2		0.09151	0.206574		0.06907	0.006832
3		0.036	0.006812		0.06028	0.003625
4		0.03512	0.005899		0.05518	0.003687
5		0.0358	0.005883		0.05545	0.003013
6		0.03173	0.001543		0.06066	0.003128
7		0.02981	0.000772		0.06172	0.003799
1	50	0.13374	0.057481	60	2.64657	215.3860
2		0.42329	2.435236		18.81421	11061.65
3		0.10285	0.017674		4.85367	930.0744
4		0.09783	0.016412		0.91503	27.68707
5		0.09591	0.015526		0.89928	27.37115
6		0.07823	0.002296		0.72904	25.10524
7		0.08065	0.002874		0.4708	2.532989

```
    if as(p, q)[1] then S := S ∪ {j} end if
  end do;
  add S at the end of L_C;
end do;
N:=[];
for S in L_C do add |S| at the end of N; end do;
if ∅ ∈ L_C then esr:=false
  else initialize sequence ind := (1,...,1) of
      ↪ length n_C;
  ips:=init_prop_seq(L_C, N);
  initialize sequence w of n zeros; v_j:=[];
  if ips[1] then
    ind:=ips[2];
    if opt = 1 then
      for i to n_C do
        j:=L_C[i][ind[i]];
        if j ∉ v_j then
          add j at the end of v_j;
          res:=part_sq_r(LL[j], w);
          if res[2] then
            w:=res[1]
            else break
          end if
        end if
      end do
      else if opt = 2 then
        for i to n_C do j:=L_C[i][ind[i]];
          if j ∉ v_j then
            add j at the end of v_j;
            res:=part_sq_r1(P0, LL[j], w);
            if res[2] then
              w:=res[1]
              else break
            end if
          end if
        end do
        else for i to n_C do j:=L_C[i][ind[i]];
          if j ∉ v_j then
            add j at the end of v_j;
            res:=part_sq_r2(P0, LL[j], w);
            if res[2] then
              w:=res[1]
              else break
            end if
          end if
        end do
      end if
    end if;
    if w ∘ w = v then
      w_o:=w
      else while esr and w ∘ w ≠ v do
        initialize sequence w of n zeros; v_j
            ↪ :=[];
        nps:=next_prop_seq(L_C, N, ind);
        if nps[1] then
          ind:=nps[2];
          if opt = 1 then
            for i to n_C do
              j:=L_C[i][ind[i]];
              if j ∉ v_j then
                add j at the end of v_j;
                res:=part_sq_r(LL[j], w);
                if res[2] then
                  w:=res[1]
                  else break
                end if
```

```
          end if
        end do
        else if opt = 2 then
          for i to n_C do
            j:=L_C[i][ind[i]];
            if j ∉ v_j then
              add j at the end of v_j;
              res:=part_sq_r1(P0, LL[j], w);
              if res[2] then
                w:=res[1]
                else break
              end if
            end if
          end do
          else for i to n_C do
            j:=L_C[i][ind[i]];
            if j ∉ v_j then
              add j at the end of v_j;
              res:=part_sq_r2(P0, LL[j], w);
              if res[2] then
                w:=res[1]
                else break
              end if
            end if
          end do
        end if
      end if;
      if w∘w=v then w_o:=w; break end if
      else esr:=false
    end if
  end do
 end if
 else esr:=false
end if
```

```
  end if
end if;
return w_o;
end proc
```

It follows from the previous sections that all of the above algorithms finding a half iterate of $\alpha \in Var(n)$: *sq_root*, *sq_root1*, *sq_root2* and *sq_root3* are in the most optimistic case with time complexity $O(n)$ and in the most pessimisitc case they are exponential time.

IV. 3. Comparison of procedures determining one half iterate

In a similar way the means and variances of work times of procedures finding one half iterate, if it exists, were compared. All of these procedures returned the same proportion of sequences for which there exist half iterates.

Tab. 4 contains:

- the number of the procedure in the first column; 1 corresponds to the procedure *sq_root(v)*, 2 – *sq_root1(v)*, 3 – $sq\_root2(v,1)$, 4 – $sq\_root2(v,2)$, 5 – $sq\_root3(v,1)$, 6 – $sq\_root3(v,2)$, 7 – $sq\_root3(v,3)$;
- lengths of sequences in columns 2 and 5;
- averages work times of examined procedures in columns 3 and 6;
- variances of work times of examined procedures in columns 4 and 7.

Tab. 5. Means and variances of work time in $[s]$ of tested procedures for various testing sets being the result of the procedure $gen\_rand\_var(len, 100)$ for $len \in \{10, 20, 30, 40, 50, 60\}$

Proc	len	Mean	Var	len	Mean	Var
1	10	0.00032	0.000005	20	0.00435	0.000098
2		0.0022	0.00006		0.00362	0.000094
3		0.00062	0.000009		0.00268	0.000086
4		0.00078	0.000012		0.00297	0.000038
5		0.00031	0.000005		0.00251	0.000033
6		0.00108	0.000016		0.00328	0.000041
7		0.00031	0.000005		0.00277	0.000035
1	30	0.00951	0.000139	40	0.01905	0.000204
2		0.00981	0.000186		0.01749	0.000074
3		0.00795	0.000166		0.01874	0.000183
4		0.00986	0.000107		0.01672	0.000101
5		0.0074	0.000077		0.01833	0.000208
6		0.00982	0.000086		0.02076	0.000328
7		0.00738	0.000107		0.01719	0.000077
1	50	0.03359	0.000306	60	0.0544	0.000421
2		0.03142	0.000349		0.05013	0.000526
3		0.03151	0.00033		0.0486	0.000443
4		0.0344	0.000413		0.05364	0.00066
5		0.03302	0.000428		0.06288	0.00111
6		0.03166	0.000299		0.0582	0.000865
7		0.03283	0.000355		0.04511	0.000193

Investigated procedures were used for sequences for which there exist half iterates generated by the procedure $gen\_rand\_sq(len, 100)$ for $len \in \{10, 20, 30, 40, 50, 60\}$. It can be seen that in each case *sq_root(v)* is faster than *sq_root1(v)* and $sq\_root2(v, 1)$ is slower than $sq\_root2(v, 2)$.

Tab. 5 contains the same columns as the previous table, but it concerns data generated by the procedure $gen\_rand\_var(len, 100)$ for $len \in \{10, 20, 30, 40, 50, 60\}$. It is seen that both means and variances of work times of examined procedures are much shorter than in the previous table, since sequences generated by $gen\_rand\_var(len, 100)$ do not have to have a half iterate and examined algorithms are able to verify this quickly.

Acknowledgements

The author is grateful to Charlotte Stępień for checking excerpts from the text. All algorithms were implemented and performed with help of Maple software version 18.

References

[1] J. Gross, J. Yellen, *Handbook of Graph Theory*, CRC Press, 2003.

[2] M.N.S. Swamy, K. Thulasiraman, *Graphs: Theory and Algorithms*. Wiley, 1992.

[3] Kneser, H. *Reelle analytische Lösungen der Gleichung $\Phi(\Phi(x)) = e^x$ und verwandter Funktionalgleichungen*. Journal fur die reine und angewandte Mathematik. **187**, 56–67 (1950).

[4] Gray J., Parshall, K. Episodes in the History of Modern Algebra (1800–1950), American Mathematical Society, ISBN 978-0-8218-4343-7, 2007.

[5] E. Schröder, Über iterirte Functionen. *Mathematische Annalen*, **3** (2), 296–322 (1870).

[6] G. Szekeres, *Regular iteration of real and complex functions* Acta Mathematica **100**, (3–4) 361–376 (1958).

[7] T. Curtright, C. Zachos, X. Jin, *Approximate solutions of functional equations*, Journal of Physics A **44** (40): 405205 (2011).

[8] M.C. Zdun, *On iterative roots of homeomorphisms of the circle*, Bull. Pol. Acad. Sci. Math **48** (2), 203-213.

Paweł Marcin Kozyra is an Assistant at the Department of Mathematics, Faculty of Mathematics, Physics and Chemistry, University of Silesia in Katowice. He received his master's degree in Mathematics from this department, University of Silesia in Katowice, in 2005, his master's degree in Computer Science from the Faculty of Automatic Control, Electronics and Computer Science, Silesian University of Technology, in 2011, and obtained his PhD in Mathematics from the Institute of Mathematics of the Polish Academy of Sciences in 2017. His research fields include bounds on the moments of linear combinations of order statistics and kth records, discrete mathematics and mathematical theory of music.