ALGORYTMY

Vol. X • No. 17 • 1973

INSTYTUT MASZYN MATEMATYCZNYCH

A L G O R Y T M Y
Vol. X, N° 17 1973

Adres Redakcji: Warszawa, ul. Krzywickiego 34, tel. 28-37-29

Druk IMM pap. offset kl. III g. 70 zam. 212/72 nakł. 550 egz. GP-II-1416/68

W

TREŚĆ
СОДЕРЖАНИЕ
CONTENTS

ALGORYTMY, X, N° 17, 1973, 5-22

681.322.001.12:519.4

PROCESSES AND PROCESSORS

by Józef WINKOWSKI
Computing Center
of Polish Academy of Sciences

Received 25.09.1972

In the paper some categorical models for processes and processors are proposed. Processes are considered as categories which are some orderings. Processors are considered as some categories. A process of a processor is defined as a linear process with a functor into the given processor. A composed process is defined as a subcategory of the product of components. Composed processors are defined in a similar way. A definition of an interaction of processors is given. The above mentioned notions are illustrated by examples.

INTRODUCTION

The computation process notion is very essential in the computer science. It is especially clear in simulation where not only final result is important but also the computation process run and its similarity to the simulated process. Therefore, a need arises to precise the process and computation process notion and to explain the nature of processes.

It appears natural to consider a process as a category of a particular kind. Here, we shall use the notion of so called small category only. Such a category is a set A with a composition rule according to which an element $\alpha\beta \in A$ is assigned to certain elements $\alpha \in A$, $\beta \in A$ in such a way that the following conditions are satisfied:

(C1) if $\alpha\beta$, $\beta\gamma$ are defined then $(\alpha\beta)\gamma$, $\alpha(\beta\gamma)$ are defined and identic,

(C2) if $(\alpha\beta)\gamma$ or $\beta(\gamma\alpha)$ is defined then $\beta\gamma$ is defined,

(C3) each element α has a left unit α^- and a right unit α^+ (an element ε is called a left (right) unit of α iff $\varepsilon\alpha(\alpha\varepsilon)$ is defined and $\varepsilon\beta = \beta, \gamma\varepsilon = \gamma$ for every β, γ for which $\varepsilon\beta, \gamma\varepsilon$ is defined).

The elements of A are called morphisms. These of them which are units of others are called objects. We denote objects by small latin letters. A category all morphisms and objects of which are respectively morphisms and objects of a category A is called a subcategory of A. The product of categories A, B is the category with pairs of morphisms of A and B as morphisms and with the following composition rule:

$$(\alpha_1, \beta_1)\cdot(\alpha_2, \beta_2) = (\alpha_1\alpha_2, \beta_1\beta_2)$$

It is denoted by $A \times B$. In a similar way the product of any family of categories can be defined. A mapping f of a category A into a category B is called a functor if it preserves objects and if $f(\alpha)\ f(\beta)$ is defined and

$$f(\alpha)\ f(\beta) = f(\alpha\beta)$$

whenever $\alpha\beta$ is defined. If there exists the inverse functor then f is called isomorphism and categories A, B are said to be isomorphic. Projections of the product $A \times B$ onto A and onto B are clearly functors. The injection of a subcategory A of a category B into B is a functor.

PROCESSES

In any process one can indicate states of realisation and segments between states. Some of such segments are composed of others. Then the last follow immediately one after another. A segment composed of a segment α and of a segment β which follows immediately after α we denote by $\alpha\beta$.

One can agree that if β follows immediately after α and γ follows immediately after β then γ follows immediately after $\alpha\beta$ and $\beta\gamma$ follows immediately after α. Moreover, the

composition can be considered as an associative operation so that the condition (C1) is satisfied.

Farther, if γ follows immediately after $\alpha\beta$ then it follows immediately after β so that (C2) is satisfied. Finally, the condition (C3) means that for every segment α there exist some boundary segments α^-, α^+ which correspond respectively to the initial state and to the final one.

Because a segment α should represent all that what take place between states α^- and α^+ the following condition should be satisfied:
(P1) if $\alpha^- = \beta^-$ and $\alpha^+ = \beta^+$ then $\alpha = \beta$.

Moreover, all segments containing a non-boundary segment should be non-boundary. Hence
(P2) if $\alpha\beta$ is an object then α, β are objects.

As result, any process can be considered as a category in which the conditions (P1), (P2) are satisfied. Morphisms of such a category we call process segments. Segments which are objects we call process states or boundary segments. If the composition $\alpha\beta$ of segments α, β is defined then we say that β follows immediately after α.

Example 1. Let $x_0, x_1, x_2, \ldots$ be a sequence of elements of a set X. Let us consider a process A which consist in successive attaining the values $x_0, x_1, x_2, \ldots$ by a quantity. Segments $x_m, \ldots, x_{m+n}$ of the sequence $x_0, x_1, x_2, \ldots$ can be considered as segments of the process. Each segment $x_m, \ldots, x_n$ $(m \leq n)$ we denote by $m|x|n$. For segments $m|x|n$, $n|x|p$ one can say on composition

$$m|x|n \cdot n|x|p = m|x|p$$

The segments of the form $m|x|m$ are boundary or process states.*

Any subcategory of a process can be considered as a process. Therefore, such a subcategory will be called a subprocess of the given process. For instance, the subcategory formed of

segments $m|x|n$ with even m, n is a subprocess of the above described process.

The product A of some processes A_i can be understood as a process composed of independently running processes A_i.

Example 2. Let A be the process from the example 1. Let B be a similar process which consists in successive attaining the values $y_0, y_1, y_2, \ldots$ from a set Y by another quantity. Then the product $A \times B$ can be considered as a process which consist in independent one of another attaining the above mentioned values by both quantities. *

If some processes are in a relationship then they form a process which would be called a boundle of processes. Relationship among processes means that the boundle is a subprocess of the product of components. Formally, by a boundle of some processes A_i we mean a process A isomorphic with a subprocess A' of the product $\prod A_i$ with images $a_i(A') = A_i$ under projections $\prod A_i \xrightarrow{a_i} A_i$. If μ denotes the suitable injection of A into $\prod A_i$ then the family of functors $A \xrightarrow{\mu a_i} A_i$ we call a boundle construction. Here, μa_i denotes the function defined by the formula $\mu a_i(\alpha) = a_i(\mu(\alpha))$.

Example 3. If the processes A, B from the example 2 are related in such a way that both quantities attain alternately their successive values then morphisms composed of pairs:

$$(0|x|0,0|y|0),(0|x|1,0|y|0),(1|x|1,0|y|0),$$
$$(1|x|1,0|y|1),(1|x|1,1|y|1),\ldots$$

are segments of the resulting boundle. The boundle together with projections restricted to morphisms of the boundle is a boundle construction.

If the first quantity attains the value x_n after attaining y_{n-1} but before attaining y_{n+1} by the second quantity and conversely and values x_n, y_n can be attained in arbitrary order then we have a boundle with the segments composed of:

$(0|x|0,0|y|0)$, $(0|x|1,0|y|0)$, $(0|x|0,0|y|1)$,
$(1|x|1,1|y|1)$, $(1|x|2,1|y|1)$, ... ∎

The above described processes are discrete in such a sense that each segment α has a finite number of representations of the form $\alpha = \beta\gamma$. Clearly, there are processes which are not discrete.

Example 4. Let us consider a process which consists in attaining a value $x(t) \in X$ at each moment t from the set $\mathcal{R}$ of all real numbers by a quantity. The described process run is a function x defined in $\mathcal{R}$ and with values in X. Functions of the form $x|[a,b]$ with $a \leq b$ can be taken as process segments. For segments $x|[a,b]$, $x|[b,c]$ one can consider their composition

$$x|[a,b] \cdot x|[b,c] = x|[a,c]$$

All the functions of the form $x|[t,t]$ are boundary segments or process states. Exactly one value $x(t)$ corresponds to any such state $x|[t,t]$. ∎

The above described process as well as the process from the example 1 and the first process from the example 3 are linear i.e. the identity $\alpha^- = \beta^-$ implies $\alpha = \beta\gamma$ or $\beta = \alpha\gamma$.

Any process is isomorphic with the category which is the following ordering in the set of states:

$$a \leq b \text{ iff } a = \alpha^- \text{ and } b = \alpha^+ \text{ for a segment } \alpha$$

with the composition rule which assigns the pair

$$a \leq b \cdot b \leq c = a \leq c$$

to the pairs $a \leq b$, $b \leq c$. It suffices to assign the pair $\alpha^- \leq \alpha^+$ to each segment α. If the process is linear then so defined ordering is linear. If it is discrete then the ordering is discrete (a finite number of states occours between arbitrary two states). A weaker ordering of a subset of the set of states corresponds to any subprocess. The product ordering corresponds to the product of processes. Any segment α can be interpreted as the set of states which satisfy the conditions $\alpha^- \leq a \leq \alpha^+$.

Example 5. The process from the example 1 is isomorphic with the category $\mathcal{N}$ which is the natural ordering of the set $\{0,1,2,\ldots\}$ with the composition rule

$$m \leqslant n \;.\; n \leqslant p = m \leqslant p$$

The first process from the example 3 is isomorphic with the subcategory $\mathcal{M}_1$ of $\mathcal{N} \times \mathcal{N}$ with morphisms composed of

$$(0 \leqslant 0, 0 \leqslant 0), (0 \leqslant 1, 0 \leqslant 0), (1 \leqslant 1, 0 \leqslant 0), (1 \leqslant 1, 0 \leqslant 1),$$
$$(1 \leqslant 1, 1 \leqslant 1), (1 \leqslant 2, 1 \leqslant 1), (2 \leqslant 2, 1 \leqslant 1), \ldots$$

The subcategory $\mathcal{M}_1$ is isomorphic with $\mathcal{N}$.

The second process from the example 3 is isomorphic with the subcategory $\mathcal{M}_2$ of $\mathcal{N} \times \mathcal{N}$ with morphisms composed of

$$(0 \leqslant 0, 0 \leqslant 0), (0 \leqslant 1, 0 \leqslant 0), (0 \leqslant 0, 0 \leqslant 1), (1 \leqslant 1, 1 \leqslant 1),$$
$$(1 \leqslant 2, 1 \leqslant 1), (1 \leqslant 1, 1 \leqslant 2), (2 \leqslant 2, 2 \leqslant 2), \ldots$$

The process $\mathcal{M}_2$ is not isomorphic with $\mathcal{N}$. ∎

PROCESSORS

A processor is a device in which some actions can be realized. Certain actions can be composed of others. Then the last should can follow immediately one after another.

One can agree that if β is admissible after α, i.e. β can follow immediately after α, and if γ is admissible after β then γ is admissible after $\alpha\beta$ and $\beta\gamma$ is admissible after α. The composition of actions can be considered as an associative operation. If γ is admissible after $\alpha\beta$ then it is admissible after β. Finally, for each action α one can consider empty actions α^-, α^+ which represent the initial and the final state of α. As result, the set of processor actions with the composition operation is a category. Morphisms of such a category we call actions. Objects are called states or empty actions. If the composition $\alpha\beta$ of actions α, β is defined then we say that β is admissible after α.

Example 6. An abstract machine i.e. a relational structure $M = (S, R)$ with a carrier S (set of states) and a relation $R \subseteq S \times S$ (transition relation) can be considered as a processor in the above meaning. It suffices to define actions as pairs $(s, \bar{s})$ which belong to the reflexive and transitive closure of R and define a composition rule in the following way:

$$(s,\bar{s})\,(\bar{s},\bar{\bar{s}}) = (s,\bar{\bar{s}})$$

We obtain a processor M^{*} with states of the form (s,s) where $s \in S$.

The machine M can be also represented as a category $M^{\square}$ in another way. Namely, M can be considered as the graph with elements of S as vertices and with pairs belonging to R as directed arcs. Then actions can be defined as directed paths (including "empty" paths which reduce to a single vertex). The set W_M of all such paths with the natural composition rule is a category. Moreover, if an equivalence $\sim$ in W_M is given such that:

(E1) equivalent paths have the same origin and the same extremity,

(E2) if $\alpha \sim \alpha'$ and $\alpha\beta$ is defined then $\alpha\beta \sim \alpha'\beta$; likewise if $\gamma\alpha$ is defined then $\gamma\alpha \sim \gamma\alpha'$,

then a composition rule can be defined for classes modulo $\sim$ by the formula

$$[\alpha] \cdot [\beta] = [\alpha\beta]$$

where α, β are arbitrary representatives of $[\alpha], [\beta]$ respectively. It gives a new category which describes the machine. In the case of the equivalence:

$\alpha \sim \beta$ iff α has the same origin and extremity as β,

the result category is isomorphic with M^{*}. *

Example 7. An automaton can be considered as a processor with actions $(s, \omega, \bar{s})$ where:

(A1) ω is an element of a monoid Ω with a homomorphism h into the monoid of all mappings of a set S into S,

(A2) $\bar{s} = [h(\omega)](s)$,

(A3) $(s, \omega, \bar{s}) \cdot (\bar{s}, \bar{\omega}, \bar{\bar{s}}) = (s, \omega\bar{\omega}, \bar{\bar{s}})$

If Ω is a monoid freely generated by a subset $\Omega_0 \subseteq \Omega$ then it can be considered as the monoid of words over Ω_0. Therefore, we have an automaton which operates in a discrete manner. It has internal states in the set S, the input alphabet Ω_0, and the transfer function δ which is defined by the formula $\delta(\omega, s) = [h(\omega)](s)$ for all $s \in S$, $\omega \in \Omega_0$.

If Ω is the monoid of the mappings

$$\omega : [0,t) \longrightarrow X$$

with operation which assigns to every

$$\omega_1 : [0,t_1) \longrightarrow X$$
$$\omega_2 : [0,t_2) \longrightarrow X$$

the concatenation

$$\omega_1\omega_2 : [0,t_1 + t_2) \longrightarrow X$$

defined by the formula

$$[\omega_1\omega_2](t) = \begin{cases} \omega_1(t) & \text{for } 0 \leq t < t_1 \\ \omega_2(t - t_1) & \text{for } t_1 \leq t < t_1 + t_2 \end{cases}$$

then the obtained automaton can be considered as operating in a continous manner. Internal states are elements of S. The automaton being in the state s at the moment u and receiving at any moment $u \leq t < v$ the signal $\omega(t-u)$ defined by the mapping

$$\omega : [0, v-u) \longrightarrow X$$

attains the state $\bar{s} = [h(\omega)](s)$ at v. ∎

Subcategories of processors can be considered as processors. The product P of processors P_i can be considered as a processor composed of independent processors P_i.

Example 8. Independent abstract machines $M_1 = (S_1, R_1)$, $M_2 = (S_2, R_2)$ form a system which can be identified with the machine $M = (S_1 \times S_2, \bar{R}_1 \cup \bar{R}_2)$ where

$$((s_1,s_2),(\bar{s}_1,\bar{s}_2)) \in \bar{R}_1 \text{ iff } (s_1,\bar{s}_1) \in R_1 \text{ and } s_2 = \bar{s}_2,$$
$$((s_1,s_2),(\bar{s}_1,\bar{s}_2)) \in \bar{R}_2 \text{ iff } (s_2,\bar{s}_2) \in R_2 \text{ and } s_1 = \bar{s}_1$$

Assigning the action

$$\mu((s_1,s_2),(\bar{s}_1,\bar{s}_2)) = ((s_1,\bar{s}_1),(s_2,\bar{s}_2))$$

of $M_1^{\mathfrak{M}} \times M_2^{\mathfrak{M}}$ to any action $((s_1,s_2),(\bar{s}_1,\bar{s}_2))$ of $M^{\mathfrak{M}}$ we have an isomorphism of $M^{\mathfrak{M}}$ onto $M_1^{\mathfrak{M}} \times M_2^{\mathfrak{M}}$. $\mathfrak{M}$

Sometimes, processors which form a system are in relationships. The relationships cause that the resulting processor is a subcategory of the product of processors. Formally, by a system of processors P_i we mean any processor P isomorphic with a subcategory P' of the product $\prod P_i$. The family of the functors $P \xrightarrow{\nu_i} P_i$ which are obtained by composing the suitable isomorphism with projections $\prod P_i \xrightarrow{p_i} P_i$ we call a system construction.

Example 9. Let abstract machines $M_1 = (S_1,R_1)$, $M_2 = (S_2,R_2)$ be combined by relationships $S_1 = V^{A_1}$, $S_2 = V^{A_2}$ with $A_1 \cap A_2 \neq \emptyset$, where V, A_1, A_2 are arbitrary non-empty sets. The resulting system may be identified with the machine $M = (S, \bar{R}_1 \cup \bar{R}_2)$ where

$$S = \{(s_1,s_2) \in S_1 \times S_2 : s_1 | A_1 \cap A_2 = s_2 | A_1 \cap A_2\},$$

$$((s_1,s_2),(\bar{s}_1,\bar{s}_2)) \in \bar{R}_1 \text{ iff } (s_1,s_2) \in S \text{ and } (s_1,\bar{s}_1) \in R_1 \text{ and } s_2|A_2 \setminus A_1 = \bar{s}_2|A_2 \setminus A_1,$$

$$((s_1,s_2),(\bar{s}_1,\bar{s}_2)) \in \bar{R}_2 \text{ iff } (s_1,s_2) \in S \text{ and } (s_2,\bar{s}_2) \in R_2 \text{ and } s_1|A_1 \setminus A_2 = \bar{s}_1|A_1 \setminus A_2$$

The machine M_1 works in the system as the machine $\hat{M}_1 = (S_1, \hat{R}_1)$ with

$(s_1, \bar{s}_1) \in \hat{R}_1$ iff there exist $s_2 \in S_2$ and $\bar{s}_2 \in S_2$ such that $((s_1, s_2), (\bar{s}_1, \bar{s}_2)) \in \bar{R}_1 \cup \bar{R}_2$

Similarly, M_2 works as $\hat{M}_2 = (S_2, \hat{R}_2)$ with

$(s_2, \bar{s}_2) \in \hat{R}_2$ iff there exist $s_1 \in S_1$ and $\bar{s}_1 \in S_1$ such that $((s_1, s_2), (\bar{s}_1, \bar{s}_2)) \in \bar{R}_1 \cup \bar{R}_2$

Assigning the pairs

$$\mu_1((s_1, s_2), (\bar{s}_1, \bar{s}_2)) = (s_1, \bar{s}_1)$$
$$\mu_2((s_1, s_2), (\bar{s}_1, \bar{s}_2)) = (s_2, \bar{s}_2)$$

to each action $((s_1, s_2), (\bar{s}_1, \bar{s}_2))$ of $M^{\mathfrak{M}}$ we have functors

$$\mu_1 : M^{\mathfrak{M}} \longrightarrow (\hat{M}_1)^{\mathfrak{M}}$$
$$\mu_2 : M^{\mathfrak{M}} \longrightarrow (\hat{M}_2)^{\mathfrak{M}}$$

which uniquely determine a functor

$$\mu : M^{\mathfrak{M}} \longrightarrow (\hat{M}_1)^{\mathfrak{M}} \times (\hat{M}_2)^{\mathfrak{M}}$$

If actions α, α' of $M^{\mathfrak{M}}$ differ then $\mu_1(\alpha) \neq \mu_1(\alpha')$ or $\mu_2(\alpha) \neq \mu_2(\alpha')$ so that μ is an injection. Hence, the processor $M^{\mathfrak{M}}$ is isomorphic with the subcategory of $(\hat{M}_1)^{\mathfrak{M}} \times (\hat{M}_2)^{\mathfrak{M}}$ which is the subcategory with morphisms of the form

$$((s_1, \bar{s}_1), (s_2, \bar{s}_2))$$

where $((s_1, s_2), (\bar{s}_1, \bar{s}_2))$ is a morphism of $M^{\mathfrak{M}}$. The family formed by functors $M^{\mathfrak{M}} \xrightarrow{\mu_1} (\hat{M}_1)^{\mathfrak{M}}$, $M^{\mathfrak{M}} \xrightarrow{\mu_2} (\hat{M}_2)^{\mathfrak{M}}$ is a system construction.

It should be emphasized that because of influence one machine onto another it was necessary to extend transfer relations of M_1 and M_2 and to consider these machines as $\hat{M}_1$ and $\hat{M}_2$. The same refers to the processors. However, the forming

$M^{\mathfrak{M}}$ of $(\hat{M}_1)^{\mathfrak{M}}$, $(\hat{M}_2)^{\mathfrak{M}}$ seems to be more natural than similar an operation in the case of M, $\hat{M}_1$, $\hat{M}_2$ (the composition rule remains such as in $(\hat{M}_1)^{\mathfrak{M}} \times (M_2)^{\mathfrak{M}}$). *

Example 10. If machines M_1, M_2 from the previous example are combined in such a way that for every $(s_1,s_2) \in U \leq S$ a transfer

$$((s_1,s_2),(\bar{s}_1,\bar{s}_2)) \in \bar{R}_1$$

is realized and for every $(s_1,s_2) \in T = S \setminus U$ a transfer

$$((s_1,s_2),(\bar{s}_1,\bar{s}_2)) \in \bar{R}_2$$

is realized then they form a system which can be identified with the machine $M = (S,R)$ where

$$((s_1,s_2),(\bar{s}_1,\bar{s}_2)) \in R \text{ iff } (s_1,s_2) \in U \text{ and } ((s_1,s_2),(\bar{s}_1,\bar{s}_2)) \in R_1$$
$$\text{or}$$
$$(s_1,s_2) \in T \text{ and } ((s_1,s_2),(\bar{s}_1,\bar{s}_2)) \in \bar{R}_2$$

This system can be considered as the least subcategory of $(\hat{M}_1)^{\mathfrak{M}} \times (\hat{M}_2)^{\mathfrak{M}}$ with morphisms

$$((s_1,\bar{s}_1),(s_2,\bar{s}_2))$$

satisfying the condition

$$((s_1,s_2),(\bar{s}_1,\bar{s}_2)) \in R. \quad *$$

Example 11. Let us consider automata P_1, P_2 such as that described in the example 7 with state sets S_1, S_2, with monoids Ω_1, Ω_2 where

$$\Omega_1 \ni \omega_1 : [0,t) \longrightarrow S_2$$
$$\Omega_2 \ni \omega_2 : [0,t) \longrightarrow S_1,$$

and with homomorphisms h_1, h_2 of Ω_1, Ω_2 into the monoids of the mappings of S_1 into S_1 and S_2 into S_2 respectively. If P_1,

P_2 are connected in such a way that P_1 receives the states of P_2 as input signals and conversely then the resulting system can be treated as the subcategory P of $P_1 \times P_2$ formed by morphisms $\big((s_1, \omega_1, \bar{s}_1), (s_2, \omega_2, \bar{s}_2)\big)$ which satisfy the following conditions:

(1) ω_1, ω_2 have the same domain $[0,t)$,

(2) $s_1 = \omega_2(0)$, $s_2 = \omega_1(0)$,

(3) for every $0 < \tau < t$, $\big[h_1(\omega_1)|[0,\tau))\big](s_1) = \omega_2(\tau)$ and $\big[h_2(\omega_2)|[0,\tau))\big](s_2) = \omega_1(\tau)$. ∎

PROCESSES OF PROCESSORS

A processor considered as a whole works in a linear manner. In any particular activity only some actions are realized. They correspond to some segments of the suitable process and the composition of actions corresponds to the composition of segments. Therefore, by a process of a processor P we mean a linear process A with a functor

$$f : A \longrightarrow P$$

(or this functor considered as a mapping).

Example 12. The process $\mathcal{N}$ from the example 5 with a functor f which assigns an action

$$f(m \leq n) = \big(s(m), s(n)\big)$$

of the processor M^* from the example 6 to any pair $m \leq n$ of $\mathcal{N}$ is a process of M^*. The function f alone is a process such as that described in the example 1. The successive states $s(0)$, $s(1)$, $s(2)$,... are such that all the pairs

$$\big(s(m), s(n)\big), \quad m \leq n$$

are actions of M^*. ∎

Example 13. The process A from the example 4 with a functor f which assigns an action

$$f\big(x|[a,b]\big) = \big(s, \omega_{a,b}, \bar{s}\big)$$

of the automaton from the example 7 with

$$\omega_{a,b} : [0,b-a) \longrightarrow X$$

defined by the formula

$$\omega_{a,b}(t) = x(a+t)$$

to every segment $x|[a,b]$ of A is a process of the automaton.*

Example 14. Let us consider the process $\mathcal{M}_1$ from the example 5 and the processor M^* from the example 8. If there is a functor f which assigns an action

$$f(m_1 \leq n_1, m_2 \leq n_2) = \big((s_1(m_1), s_2(m_2)), (s_1(n_1), s_2(n_2))\big)$$

to every morphism $(m_1 \leq n_1, m_2 \leq n_2)$ of $\mathcal{M}_1$ then the $\mathcal{M}_1$ with f is a process of M^*. However, in general, does not exist such a functor in the case of the processor M^* from the example 9. But if such a functor exists all the diagrams

$$\begin{array}{ccc} & f(m_1 \leq n_1, m_2 \leq n_2) & \\ f(m_1 \leq q_1, m_2 \leq q_2) & \square & f(n_1 \leq p_1, n_2 \leq p_2) \\ & f(q_1 \leq p_1, q_2 \leq p_2) & \end{array}$$

are commutative. *

Any process $A \xrightarrow{f} P$ of a system P of P_i's with the construction $\{P \xrightarrow{\nu_i} P_i\}$ is determined by the processes $A \xrightarrow{f\nu_i} P_i$ of particular P_i's.

Theorem 1. For every linear process A with functors $A \xrightarrow{\varphi_i} P_i$ there exists at most one process $A \xrightarrow{f} P$ of the system P with construction $\{P \xrightarrow{\nu_i} P_i\}$ such that $f\nu_i = \varphi_i$ for all i.

Proof. Let $\prod P_i \xrightarrow{p_i} P_i$ be projections. There exists unique functor φ such that $\varphi p_i = \varphi_i$ for all i. On the other hand there exists an unique injection $P \xrightarrow{\nu} \prod P_i$ such that $\nu p_i = \nu_i$ for all i. Hence, the functor f satisfies the condition $f\nu = \varphi$. But may be at most one such a functor because ν is an injection. ∎

The above theorem implies, for instance, that the functor f from the example 14 is uniquely determined by functors $\mathcal{N} \xrightarrow{f_1} M_1^{\mathcal{M}}$, $\mathcal{N} \xrightarrow{f_2} M_2^{\mathcal{M}}$ defined as in the example 12.

If processors P_i which form a system P with a construction $\{P \xrightarrow{\nu_i} P_i\}$ work, the processes $A_i \xrightarrow{f_i} P_i$ are related in such a way that processes A_i form a boundle A with a construction $\{A \xrightarrow{\mu_i} A_i\}$ and there exists a functor $A \xrightarrow{f} P$ such that all the diagrams

$$\begin{array}{ccc} A & \xrightarrow{f} & P \\ {\scriptstyle \mu_i}\downarrow & & \downarrow{\scriptstyle \nu_i} \\ A_i & \xrightarrow{f_i} & P_i \end{array}$$

are commutative. The functor $A \xrightarrow{f} P$ will be called a boundle of processes $A_i \xrightarrow{f_i} P_i$. In general, it is not a process of P because the process A does not need to be linear.

Example 15. Let $\mathcal{N} \xrightarrow{f_1} M_1^{\mathcal{M}}$, $\mathcal{N} \xrightarrow{f_2} M_2^{\mathcal{M}}$ be some processes of $M_1^{\mathcal{M}}$, $M_2^{\mathcal{M}}$ from the example 8 such as that described in the example 12. Then the functor $\mathcal{N} \times \mathcal{N} \xrightarrow{f} M^{\mathcal{M}}$ where

$$f(m_1 \leqslant n_1, m_2 \leqslant n_2) = \big((s_1(m_1), s_2(m_2)), (s_1(n_1), s_2(n_2))\big)$$

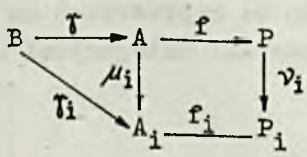
is a boundle of $\mathcal{N} \xrightarrow{f_1} M_1^{\mathcal{M}}$, $\mathcal{N} \xrightarrow{f_2} M_2^{\mathcal{M}}$. The restriction of f to the subprocess $\mathcal{M}_2$ gives another boundle. ∎

The meaning of the boundle of processes of processors notion is partially given in the following theorem.

Theorem 2. Any process $B \xrightarrow{f} P$ of a system P of processors P_i is of the form $B \xrightarrow{\tau} A \xrightarrow{\xi} P$ where $A \xrightarrow{\xi} P$ is a boundle of some processes $A_i \xrightarrow{f_i} P_i$.

Proof. Let P has the construction $\{P \xrightarrow{\nu_i} P_i\}$ and let $\prod P_i \xrightarrow{p_i} P_i$ be projections and $P \xrightarrow{\nu} \prod P_i$ the injection which satisfies the conditions $\nu p_i = \nu_i$. We define $\varphi_i = f\nu_i$ and choose processes $A_i \xrightarrow{f_i} P_i$ such that $\varphi_i = \beta_i f_i$, where $B \xrightarrow{\beta_i} A_i$ transforms B onto A_i. Let $\prod A_i \xrightarrow{a_i} A_i$ be projections. There exists $B \xrightarrow{\beta} \prod A_i$ such that $\beta a_i = \beta_i$ for all i. Because the process B is linear its image A under β is a subcategory of $\prod A_i$. Clearly, it is a boundle of A_i's. The family $\{A \xrightarrow{\mu} \prod A_i \xrightarrow{a_i} A_i\}$ with inclusion μ is a boundle construction. There exists a functor $B \xrightarrow{\gamma} A$ such that $\gamma\mu = \beta$. The action $g(\gamma(\alpha)) = f(\alpha)$ may be assigned to every morphism $\gamma(\alpha)$ of A. Because $f(\alpha) \neq f(\alpha')$ implies $\varphi_i(\alpha) \neq \varphi_i(\alpha')$ for a certain i we have $\gamma(\alpha) \neq \gamma(\alpha')$ so that g is a function. Due to linearity of B it is a functor. By definition of g we have $f = \gamma g$. ∎

Any boundle $A \xrightarrow{f} P$ of $A_i \xrightarrow{f_i} P_i$ can be considered as an interaction of P_i's in the system P. Such an interaction can be understood as a model of a theory which describes some relationship among the functors $P \xrightarrow{\nu_i} P_i$, $A \xrightarrow{\mu_i} A_i$; $A_i \xrightarrow{f_i} P_i$, $A \xrightarrow{f} P$. The fact that the ordering which corresponds to A is not linear means that some results of activities of particular processors P_i are independent of order in which processors P_i work. In other words, processes $A_i \xrightarrow{f_i} P_i$ may run in parallel to some extent. The interaction reflects some conditions only which are satisfied by particular processes $A_i \xrightarrow{f_i} P_i$. In general, there is a number of processes $B \xrightarrow{g} P$ of P which satisfy such conditions. However, for some conditions, such processes should factor through A, i.e. should exist functors $B \xrightarrow{\gamma} A$, $B \xrightarrow{\gamma_i} A_i$ such that $g = \gamma f$ and diagrams



are commutative. Therefore, an interaction of P_i's satisfying some given conditions is useful for finding a process of P which satisfies these conditions.

Example 16. If we want to find a process $\mathcal{N} \xrightarrow{f} M^{\mathcal{M}}$ of $M^{\mathcal{M}}$ from the example 9 and such that $s_1(n)$ $(s_2(n))$ is attained after $s_2(n-1)$ $(s_1(n-1))$ but before $s_2(n+1)$ $(s_1(n+1))$ then we try to find a boundle $\mathcal{M}_2 \xrightarrow{g} M^{\mathcal{M}}$ of some $\mathcal{N} \xrightarrow{f_1} (\hat{M}_1)^{\mathcal{M}}$, $\mathcal{N} \xrightarrow{f_2} (\hat{M}_2)^{\mathcal{M}}$. The process $\mathcal{N} \xrightarrow{f} M^{\mathcal{M}}$ we are looking for should be of the form $\mathcal{N} \xrightarrow{T} \mathcal{M}_2 \xrightarrow{g} M^{\mathcal{M}}$. ∎

FINAL REMARKS

The presented way for describing of processes and processors has many advantages. At first, very simple notions which correspond to real things are used. Next, discrete as well as non-discrete processes and processors can be treated in the same way. Third, the proposed notion of processor includes the abstract machine notion (a set of states with a transition relation), the automaton notion and others like these. But the most important fact is that composed processes, systems of processors, and interactions of processors which form some systems can be defined in a natural way. It gives some possibilities to develop an unified theory of dynamics of objects of arbitrary complexity.

All processes and processors described in this paper are very simple as categories. Any process is simply an ordering with the natural composition rule. Any abstract machine may be represented as a quasi-ordering with the natural composition rule. However, considerations in terms of morphisms and objects are very convenient. This is one reason for use the notions of category theory. Moreover, some processors (like automata or abstract machines considered as categories of paths or as categories of classes of paths) can not be represented as quasi-orderings. But still they can be represented as categories. This is another reason for use the categorical notions.

References

[1] BLIKLE A.: Iterative Systems; an Algebraic Approach, Bull. Acad. Polon. Sci., Sér. Sci. Math. Astronom. Phys., 20(1972), 51-55

[2] KALMAN R.E., FALB P.L., ARBIB M.A.: Topics in Mathematical Systems Theory, Mc Graw-Hill, 1969

[3] MITCHELL B.: Theory of Categories, New York, 1965

[4] PAWLAK Z.: Programmed Computers (in Polish), Algorytmy, 10(1969), 5-19

[5] WINKOWSKI J.: Interacting Abstract Machines, Bull. Acad. Polon. Sci., Sér. Sci. Math. Astronom. Phys., 20(1972), 181-184

[6] WINKOWSKI J.: Composed Abstract Machines, ibid., 20(1972), 407-411

PROCESY I PROCESORY

Streszczenie

Praca stanowi próbę potraktowania procesów i procesorów jako kategorii. Procesy rozważa się jako kategorie stanowiące uporządkowania a procesory jako kategorie. Proces procesora jest definiowany jako proces liniowy z funktorem w dany procesor. Procesy złożone definiuje się jako podkategorie produktu składowych. Podobnie określa się systemy procesorów. Podana jest definicja współdziałania wielu procesorów. Wymienione pojęcia są zilustrowane przykładami.

ПРОЦЕССЫ И ПРОЦЕССОРЫ

Резюме

Предлагаемая робота является попыткой представления процессов и процессоров как категории. Процессы рассматриваются как категории будущие порядками. Процессоры рассматриваются просто как категории. Процесс конкретного процессора определяется как линейный процесс с функтором в этот процессор. Сложный процесс определяется как подкатегория произведения слагаемых. Сложные процессы определяются подобным образом. Вводится тоже понятие воздействия нескольких процессов. Все рассматриваемые понятия обяснены примерами.

ALGORYTMY, X, N° 17, 1973, 23-34

681.322.004.14:519.4

AN AUTOMATIZATION OF THE COMMUTATOR CALCULUS

by Emanuel CZYŻO
The Warsaw University Computation Centre

Received 25.07.1972

The paper deals with some computational problems in group theory. In this paper the word processing system - WP and the collecting algorithm - COLLECTING are described. The system WP is useful in computational problems in group theory, such as commutator calculus and similar problems. The collecting algorithm is an implementation of Hall collecting process for an arbitrary word written in generators of the nilpotent free group G of a given degree of nilpotency $(nil \leq 11)$, generated by $x_1, \ldots, x_r$ $(r \leq 9)$. It is written in the form of an ALGOL procedure which uses the WP system.

1. INTRODUCTION

The experiments connected with the application of computers in mathematical scientific research gain ever increasing practical importance now. Interesting results from not only a theoretical but also practical point of view have been achieved with the aid of computers in the field of mathematical logic and especially in proving mathematical theorems.

The research having the purpose of using computers in the abstract algebra are more and more often performed. Those applications in particular deal with such parts of algebra as algebraic number theory, semigroup and group theory.

The experiments concerned with applications of computers in the group theory are frequent now. It is obvious because of

the importance of this part of algebra. The most significant of up to now achieved results deal with such problems as coset enumeration, subgroup lattices and representations of finite groups (cf. [1]).

This paper deals with some computational problems in group theory. The paper consists of two parts.

The first part is a presentation of the word processing system WP designed for efficient performance of Hall collecting process and similar problems in the group theory.

In the second part the collecting algorithm is described. The algorithm is an implementation of the Hall collecting process described in [6] for an arbitrary word written in generators of the nilpotent free group $G(x_1, x_2, \ldots, x_r)$ of a given degree of nilpotency (nil), generated by $x_1, x_2, \ldots, x_r$.

The algorithm allows us to decide in a simple way whether or not the word $g(x_1, x_2, \ldots x_r)$ represents the identity of the group, it also plays an important part in the construction of algorithms for deciding other problems in nilpotent groups (cf. [7]).

The terminology and notations of this paper are in accordance with those of [6] and [3].

2. COMMUTATOR CALCULUS

The importance of automatization of the commutator calculus has been increasing lately because of the positive results obtained when deciding some problems for nilpotent groups.

P. Hall and M. Hall have mainly contributed to the development of the commutator calculus. The investigation of a lower central series developed by P. Hall allowed to make the commutator calculus a systematic theory.

We will not give the description of the theory here. The basic notions of the theory such as definitions of commutators

and basic commutators, and the description of a collecting process, together with many theoretical results are described in M. Hall's book [6].

The notion of basic commutators introduced by M. Hall [6] is assumed to be known. Let $F(x_1, x_2, \ldots, x_r)$ denote a free group generated by $x_1, x_2, \ldots, x_r$. The number $M(r, n)$ of basic commutators of weight n, and r generators is given by Witt's formula

$$M(r, n) = \frac{1}{n} \sum_{d} \mu(d)\, r^{\frac{n}{d}}$$

where $\mu(d)$ is a Möbius function on integers and d takes all the values of the divisors of n. In [2] the numerical values of $M(r, n)$ for $r \leqslant 9$ and $n \leqslant 11$ are given.

In the theory of basic commutators developed by M. Hall a standard form of elements of a nilpotent free group is known. It is given in the following theorem.

Basis theorem

If F is the free group with free generators $x_1, \ldots, x_r$ and if in a sequence of basic commutators $c_1, \ldots, c_t$ are those of weights 1, 2, ..., n, then an arbitrary element f of F has a unique representation,

$$f = c_1^{e_1} c_2^{e_2} \ldots c_t^{e_t} \pmod{F_{n+1}} \qquad (2.0)$$

The basic commutators of weight n form a basis for the free Abelian group F_n/F_{n+1} (cf. [6], p. 175).

Let $g = g(x_1, x_2, \ldots, x_r)$ be a word written in generators of the nilpotent free group G of a given nil; then the collecting process described by M. Hall [6] gives an effective method of finding the presentation (2.0) for the element g.

Collecting of commutators may be carried out according to the following rules

$$vu = u\,v\,(v, u) \tag{2.1}$$

$$vu^{-1} = u^{-1}\,v\,v_2\,v_4 \cdots v_3^{-1}\,v_1^{-1} \pmod{F_{n+1}} \tag{2.2}$$

$$v^{-1}u = u\,v^{-1}\,w_2\,w_4 \cdots w_3^{-1}w_1^{-1} \pmod{F_{n+1}} \tag{2.3}$$

$$v^{-1}u^{-1} = u^{-1}\,v_1\,v_3 \cdots v_4^{-1}\,v_2^{-1}\,v^{-1} \pmod{F_{n+1}} \tag{2.4}$$

where $v_0 = v$, $v_{i+1} = (v_i, u)$ for $i = 0, 1, \ldots$

and $w_1 = (v, u)$, $w_{i+1} = (w_i, v)$ for $i = 1, 2, \ldots$

The collecting process gives therefore a possibility to decide a word problem in the case of a nilpotent free group of a given degree of nilpotency. It also plays an important part in the construction of algorithms for deciding other problems in nilpotent groups of a given nil.

Decision problems for nilpotent groups are considered by A.W. Mostowski in [7]. He gives effective methods of deciding the problem of inclusion of the element $g = g(x_1, x_2, \ldots, x_r)$ to the subgroup H generated by the words $h_1(x_1, x_2, \ldots, x_r), \ldots, h_m(x_1, x_2, \ldots, x_r)$ and the finiteness problem for nilpotent groups of a given degree of nilpotency on the base of the so-called subgroup theorem.

On the base of a collecting algorithm, considered in the section 4, we hope to implement the algorithms on a computer.

3. WORD PROCESSING SYSTEM

3.1. According to Cannon (cf. [1], p. 10) one of the most important problems is an automatization of the so called tedious algebra comprising first of all commutator calculus.

The basic problem consists in the elaboration of a suitable programming language for a possibly extensive class of group-theoretical problems. However, the present knowledge of these problems being insufficient has not allowed to define the language yet.

The research dealing with construction of algorithms for some specified classes of grouptheoretical problems is nevertheless under way. The experiments bring some results obtained by elaborated algorithms and simultaneously allow better characteristics of considered problem classes, having in mind the aspect of their further automatization.

3.2. A group G may be expressed up to isomorphism by the set of generators $x_1, x_2, \ldots, x_r$ and the set of relations

$$r_1(x_1, x_2, \ldots, x_r) = \ldots = r_m(x_1, x_2, \ldots, x_r) = 1$$

where 1 is the identity of the group and $r_1, \ldots, r_m$ are the words written in the group generators.

The element $g \in G$ is represented by the word $g(x_1, x_2, \ldots, x_r)$ written in the generators and grouptheoretical operations are then expressed by adequate operations on words. There are many computational problems in the group theory which could be formulated in terms of words written in some alphabets and some operations defined on them. Because of lack of a suitable machine independent programming language for these problems (cf. [1], p. 10) the author has worked out a word processing system WP which facilitates formulation and solving of a certain class of grouptheoretical problems with the aid of computers. Among other applications of the system the research on automatization of commutator calculus is carried out.

3.3. The word processing system WP has been described in [3]. We confine here ourselves to giving definitions of a letter, of a word and of a basic word. The operations defined by the system take these objects as arguments.

An e l e m e n t (word element, letter) is an ordered quadruple of integers

$$L = (a : l, I, r)$$

a - address of the element, $a > 0$

l - left link, $l \geq 0$

r - right link, $r \geq 0$

I - information part of the element (ordinal number of the letter).

BL = (a : 0, I, r) is an initial element, EL = (a : 1, I, 0) is a final element and VL = (a : 1, 0, r) is a dummy element.

Let us define a relation $)($ on pairs of letters. Let $L_1 = (a_1 : l_1, I_1, r_1)$ and $L_2 = (a_2 : l_2, I_2, r_2)$

$$L_1)(L_2 \overset{\text{def}}{=} r_1 = a_2 \wedge l_2 = a_1$$

Let U denote the finite sequence of letters $L_1, L_2, \ldots, L_k$. U is a w o r d if

1. $L_1)(L_2)(\ldots)(L_k$
2. $a_i \neq a_j$ for $i \neq j$

The pair $< B, E >$ where B is the address of the letter L_1 and E is the address of the letter L_k, is the address of the word U.

The number of letters occuring in the word is called the length of the word. A word consisting of only dummy letters is called a dummy word. The word of the length 0 will be treated as a dummy word with the address $<0,0>$.

The sequence of letters U is called a b a s i c w o r d if

1. U is a word
2. U contains both initial and final elements
3. U does not contain dummy elements.

The system consists of two sets of procedures: basic procedures and special purpose procedures. These procedures enable processing of words. Words have been defined as double linked lists. They contain only one sort of elements called letters. The information part of a word element is an ordinal number of a letter (cf. sec. 4).

The WP system has been designed mainly for efficient performance of the collecting process. But, it is useful also

in other problems. The system may be useful for programming some algorithms for semigroups defined by a set of generators and relations. The algorithm generating all words equivalent to the given one may serve as an example. Another field of possible applications is studying formal languages (for example algorithms for syntactical inference). The system will be expanded in the course of further experiments especially dealing with automatization of the commutator calculus.

4. THE COLLECTING ALGORITHM

4.1. The basic part in the commutator calculus plays the collecting process described by M. Hall in [6], for its helpfulness in solving many problems of the theory of nilpotent groups (see sec. 2).

The first computer application to this problem is due to H. Felsch. He wrote a program which implements the collecting process for words of the type $(x_1\ x_2)^n$. Campbell and Lamberth wrote a SLIP program to carry out the collecting process on words of a nilpotent group of finite class, written either in group generators or basic commutators. Cannon has written a similar program to carry out commutator collection on-line (cf. [1], p. 9).

The author has worked out the program HALL + in 1969 and to follow the procedure HALL PLUS, which is an implementation of the collecting process for positive words expressed in generators of a group. The maximal weight of commutators in the process was 11 ($\mathrm{nil} \leqslant 11$) and the number of group generators did not exceed 9 ($r \leqslant 9$) (cf. [2]).

4.2. The collecting algorithm described here is an implementation of the full process of collecting commutators i.e. collecting process for an arbitrary word (containing also negative powers of letters) written in generators of a nilpotent free group of a given degree of nilpotency.

The algorithm has been written in the form of an algol procedure COLLECTING. The procedure has been elaborated with the aid of WP system (cf. [4]). The procedure has been based on the same assumption as the program HALL +.

We assume that the set

$$X = \{x_1, x_2, \ldots, x_r\} \quad \text{where} \quad r \leqslant 9$$

is a set of generators of a nilpotent free group and the maximal degree of nilpotency is 11 (nil $\leqslant$ 11). Those limitations come out of the accepted method of numbering commutators and of the standard length of the machine word length. The experiments proved that the limitations are not essential.

The element $g = b_1 b_2 \ldots b_k$ of a group, where b_i $(i = 1, 2, \ldots, k)$ are either basic commutators or their inversions, is represented in the form of a basic word (cf. sec. 3) in the collecting algorithm.

Numbering and ordering of commutators are defined according to [2]. The commutator number is the information part of the letter. The ordinal number of the commutator u is denoted by $N(u)$. We define additionally $N(u^{-1}) = -N(u)$. Complete information about the commutator is included in its number i.e. the number of digits in the decimal expansion of the commutator number defines the weight of the commutator and its sign determines whether it is a commutator or its inversion.

Implementation of each (2.1) - (2.4) rules run in three steps. At the start ordinal numbers of letters v and u are changed, the weight of the commutator v and the length of the word being linked, are determined. If the length of the linked word is greater than zero, then it is constructed by placing letters in the order of their increasing weights by CREATE LETTER procedure (cf. [3]). The word includes only commutators of weights not greater than nil. At the end the constructed word is linked to the basic word.

The possible reduction of letters in the basic word is performed after carrying out each of the rules. We assume that the basic word

$$g = x_{i_1}^{e_1} x_{i_2}^{e_2} \dots x_{i_k}^{e_k}$$

is in a reduced form i.e. there is no such m that $i_m = i_{m+1}$ and $e_m = -e_{m+1}$ $(1 \leq m < k)$. However reducible subwords may occur in the basic word in the course of performing collecting process. The reduction is limited to the noncollected part of the basic word. Because of the reduction we save some places in the store and avoid superfluous operations.

If the weight of the minimal commutator is greater than a half of the maximal weight, all the rules of transposition (2.1) - (2.4) come to the rule

$$v^a u^b = u^b v^a$$

because the weights of commutators v_{i+1} and w_{i+1} occuring in the rules are greater than the nil then. From this moment we deal only with the sorting of letters. The sorting of letters in the basic word is a final operation of the collecting algorithm. The details of the collecting algorithm can be found in [4].

5. APPLICATIONS OF THE COLLECTING ALGORITHM

5.1. Computation of the standard form of an element

The collecting process for words of the form

$$(\varkappa) \qquad g(x_1, x_2, \dots, x_r) = (x_1 x_2 \dots x_r)^n$$

was carried out with the aid of COLLECTING procedure for $n \leq 4$, $r \leq 4$ and suitable values of the nil parameter.

The example below is the standard form of the element $(\varkappa)$ for nil = 4, r = 4, n = 4.

STANDARD FORM

c1↑4 c2↑4 c3↑4 c4↑4 c21↑6 c31↑6 c32↑6 c41↑6 c42↑6 c43↑6 c211↑4
c212↑14 c213↑20 c214↑20 c311↑4 c312↑14 c313↑14 c314↑20 c322↑4
c323↑14 c324↑20 c411↑4 c412↑14 c413↑14 c414↑14 c422↑4 c423↑14
c424↑14 c433↑4 c434↑14 c2111 c2112↑11 c2113↑15 c2114↑15
c2122↑11 c2123↑50 c2124↑50 c2133↑25 c2134↑70 c2144↑25 c3111
c3112↑11 c3113↑11 c3114↑15 c3121↑11 c3122↑11 c3123↑36 c3124↑50
c3133↑11 c3134↑50 c3144↑25 c3221↑11 c3222 c3223↑11 c3224↑15
c3231↑25 c3233↑11 c3234↑50 c3244↑25 c4111 c4112↑11 c4113↑11
c4114↑11 c4121↑11 c4122↑11 c4123↑36 c4124↑36 c4131↑11 c4132↑11
c4133↑11 c4134↑36 c4144↑11 c4221↑11 c4222 c4223↑11 c4224↑11
c4231↑11 c4232↑11 c4233↑11 c4234↑36 c4241↑25 c4244↑11
c4321↑11 c4331↑11 c4332↑11 c4333 c4334↑11 c4341↑25 c4342↑25
c4344↑11ø

(where for example c4344↑11 denotes the 11th power of the basic commutator $(((x_4, x_3), x_4), x_4)$).

The obtained presentations were used for calculation of exponents in Hall's formula. On the base of this calculations a classification of basic commutators of the weights 3 and 4 was made. Details can be found in the paper by K. Dałek [5].

All the experiments were carried out on GIER ocmputer in the Warsaw University Computation Centre.

5.2. Finding a normal basis

The elements $g_1, g_2, \ldots, g_k$ constitute a normal basis of the group G if any element $g \in G$ can be uniquely represented in the form

$$g = g_1^{a_1} g_2^{a_2} \ldots g_k^{a_k}$$

where $a_1, a_2, \ldots, a_k$ are integers.

According to the theorem from section 2 we have that basic commutators $c_1, c_2, \ldots, c_t$ form the normal basis in a nilpotent free group.

A.W. Mostowski [7] has proved that if the group G has a normal basis which fulfils certain requirements there exists a normal basis for its subgroups.

According to the theorem known as the subgroup theorem (cf. [7]) effective algorithms for deciding some problems in nilpotent groups of a given degree of nilpotency, can be constructed.

On the base of the collecting algorithm, an algorithm for finding a normal basis for the subgroup H, generated by the words $h_1, h_2, \ldots, h_m$, is being constructed. Obtained results of this and of other problems (see sec. 2) will be the subject of next publications.

The author wishes to thank Dr A.W. Mostowski whose remarks and helpful suggestions were of a great value for the author.

References

[1] CANNON J.J.: Computers in Group Theory: a Survey, Comm. ACM 12 (1969), pp. 3-12.

[2] CZYŻO E.: An Attempt of Mechanization of Hall Collecting Process, Algorytmy 15 (1972), pp. 5-17.

[3] CZYŻO E.: Computers in Group Theory: Word Processing System (in Polish), Sprawozdania IMM i ZON Uniwersytetu Warszawskiego, No 29 (1971).

[4] CZYŻO E.: Computers in Group Theory: Collecting Algorithm (in Polish), Sprawozdania IMM i ZON Uniwersytetu Warszawskiego, No 31 (1972).

[5] DAŁEK K.: Computation of Exponents in Hall Formula, Bull. Ac. Pol. Sci. ser. math., Vol. XIX, No 8 (1971), pp. 711-718.

[6] HALL M.: The Theory of Groups, New York, 1959.

[7] MOSTOWSKI A.W.: Computational Algorithms for Deciding Some Problems for Nilpotent Groups, Fund. Math., LIX(1966), pp. 137-152.

AUTOMATYZACJA RACHUNKU KOMUTATORÓW

Streszczenie

Praca dotyczy automatyzacji pewnej klasy problemów w teorii grup. W artykule opisany został system WP oraz algorytm wybierania. System WP składa się z dwóch zestawów procedur, zapisanych w języku GIER Algol 4, umożliwiających operowanie na obiektach zwanych słowami. Z pomocą tego systemu opracowany został m.in. algorytm wybierania. Algorytm wybierania jest realizacją procesu wybierania Hall'a dla dowolnego słowa zapisanego w generatorach grupy nilpotentnej wolnej G generowanej przez $x_1,\ldots, x_r$ $(r \leq 9)$ o danym stopniu nilpotentności $(\mathrm{nil} \leq 11)$. Algorytm został opracowany w formie procedury o nazwie COLLECTING.

АВТОМАТИЗАЦИЯ ВЫЧИСЛЕНИЯ КОММУТАТОРОВ

Резюме

В работе рассматривается вопрос автоматизации некоторого класса проблем теории групп. В статье представлена система WP и собирательный алгоритм COLLECTING. Система WP полезна для таких вычислительных методов в теории групп, как вычисление коммутаторов и смежных вопросов. Собирательный алгоритм является реализацией собирательного процесса для любого слова представленного в терминах генераторов свободной нильпотентной группы G порожденной элементами $x_1,\ldots,x_r (r \leq 9)$ с данной степенью нильпотентности $(\mathrm{nil} \leq 11)$. Алгоритм разработан в виде процедуры COLLECTING.

Warszawa, ul. Krzywickiego 34
A.M.

ALGORYTMY, X, N° 17, 1973, 35-44 681.322.056.06.004.15:519.12

SORTOWANIE METODĄ PROSTYCH PRZESTAWIEŃ

Ryszard RABAN
Studium Doktoranckie
Politechniki Warszawskiej

Pracę złożono 15.04.1972

Artykuł zawiera opis i dokładną analizę algorytmów sortowania metodą przestawień prostych oraz wyprowadzenie wzoru na średnią ilość przestawień prostych (S) dla algorytmów optymalnych, tzn. realizujących sortowanie zbioru n-elementowego przy minimalnej ilości przestawień prostych. Na podstawie tych ogólnych rozważań, w drugiej części artykułu, zanalizowany został algorytm sortowania przez wstawienie (by inserting) oraz określona średnia ilość porównań (T) jaką należy wykonać przy sortowaniu zbioru n-elementowego.

1. WSTĘP

W literaturze [1], [2], [3] i [4] opisana jest metoda sortowania przez wstawianie (by inserting). Praca niniejsza ma na celu ścisłe określenie efektywności tego algorytmu. Za wskaźnik efektywności przyjęto średnią liczbę porównań potrzebną do posortowania zbioru n-elementowego.

2. OKREŚLENIA WSTĘPNE

Określenia i pojęcia wprowadzone na wstępie odnosić się będą do skończonego zbioru n-elementowego. Elementy tego zbioru oznaczać będziemy a_i dla $i = 1, 2, \ldots, n$.

Taki n-elementowy zbiór można uporządkować na n! sposobów. Dla jednoznacznego określenia położenia elementu a_i w danym uporządkowaniu (permutacji) przypiszemy temu elementowi liczbę całkowitą p_i. Wartość jej określać będzie numer pozycji (liczonej z lewa na prawo), na której znajduje się element a_i. Aby całkowicie określić permutację n-elementową wystarczy podać ciąg pozycji

$$P = \{p_1, p_2, \ldots, p_n\}$$

Np. dla zbioru pięciu elementów a_1, a_2, a_3, a_4, a_5 ciąg pozycji $P = \{2, 4, 1, 3, 5\}$ określa następujące uporządkowanie tych elementów a_3, a_1, a_4, a_2, a_5.

W dalszym ciągu mówiąc o uporządkowaniu (permutacji) P będziemy rozumieli takie uporządkowanie zbioru n-elementowego, które wyznaczone jest przez ciąg pozycji P.

Będziemy mówili, że w permutacji P element a_i poprzedza element a_j, i będziemy pisali $a_i \underset{P}{<} a_j$ jeśli $p_i < p_j$.

Jeżeli będziemy rozpatrywali dwie permutacje P_1 i P_2 zbioru n-elementowego, to dla dowolnych dwóch elementów tego zbioru a_i oraz a_j (dla $i, j = 1, 2, \ldots, n$ oraz $i \neq j$) zachodzą następujące związki

$$\left(a_i \underset{P_1}{<} a_j \text{ i } a_i \underset{P_2}{<} a_j\right) \text{ lub } \left(a_j \underset{P_1}{<} a_i \text{ i } a_j \underset{P_2}{<} a_i\right) \quad (1)$$

lub

$$\left(a_i \underset{P_1}{<} a_j \text{ i } a_j \underset{P_2}{<} a_i\right) \text{ lub } \left(a_j \underset{P_1}{<} a_i \text{ i } a_i \underset{P_2}{<} a_j\right) \quad (2)$$

W przypadku gdy spełnione są zależności (1) powiemy, że w permutacji P_2 para elementów a_i i a_j jest uporządkowana zgodnie z P_1, gdy natomiast spełnione są zależności (2) powiemy, że w permutacji P_2 para elementów a_i i a_j jest uporządkowana niezgodnie z P_1.

Przestawieniem dwóch elementów nazywać będziemy zamianę ich pozycji. Oznacza to, że jeśli w permutacji P dokonamy przestawienia elementów a_i i a_j, to w powstałej permutacji P', $p'_i = p_j$ oraz $p'_j = p_i$, natomiast pozycje związane z pozostałymi elementami pozostaną niezmienione.

Przestawieniem prostym nazywać będziemy przestawienie takich dwóch elementów a_i i a_j, dla których spełniona jest zależność $p_i = p_j + 1$ lub $p_j = p_i + 1$. Oznacza to, że przestawieniem prostym jest przestawienie elementów znajdujących się na dwóch sąsiednich pozycjach.

Drogą prostych przestawień możemy przekształcić dowolną permutację P w ustaloną z góry permutację końcową P_0. Jeśli permutacja końcowa P_0 jest ciągiem elementów uporządkowanych wg pewnej ustalonej relacji liniowo porządkującej zbiór elementów a_i $(i = 1,\ldots, n)$, to proces przekształcania dowolnej permutacji P w permutację P_0 jest sortowaniem zbioru n-elementowego.

3. SORTOWANIE METODĄ PRZESTAWIEŃ PROSTYCH

Niech permutacja P_0 będzie uporządkowanym ciągiem elementów. W dowolnej permutacji P możemy określić $\binom{n}{2}$ par różnych elementów. W tej liczbie znajdują się pary uporządkowane zgodnie z P_0 oraz pozostałe uporządkowane niezgodnie z P_0.

Wykażemy, że minimalna liczba przestawień prostych konieczna do przekształcenia permutacji P w P_0 równa jest liczbie par uporządkowanych niezgodnie z P_0 w permutacji P.

Wykazanie prawdziwości powyższej tezy opiera się na stwierdzeniu, że jedno proste przestawienie dokonane w permutacji P przekształca ją w permutację P', w której tylko para elementów przestawianych jest uporządkowana niezgodnie z P, wszystkie pozostałe pary są natomiast uporządkowane zgodnie z P. Wynika to z kolei z tego, że po przestawieniu dwóch elementów znajdujących się na sąsiednich pozycjach uporządkowanie tych

elementów względem pozostałych elementów nie ulega zmianie. Oznacza to, że wszystkie pary elementów, z wyjątkiem pary przestawianej, będą posiadały uporządkowanie zgodne z permutacją, na której dokonano przestawienia. Tak więc dokonując w permutacji P przestawienia prostego dwóch elementów tworzących parę uporządkowaną niezgodnie z P_0 tworzymy tym samym permutację P', w której liczba par uporządkowanych niezgodnie z P_0 jest mniejsza o jeden od liczby par uporządkowanych niezgodnie z P_0 w permutacji P. Następnie w P' dokonując przestawienia prostego dwóch elementów uporządkowanych niezgodnie z P_0 tworzymy permutację P'', która posiada liczbę par uporządkowanych niezgodnie z P_0 o dwa mniejszą od liczby par uporządkowanych niezgodnie z P_0 w permutacji P. Kontynuując to postępowanie po liczbie kroków równej liczbie par uporządkowanych niezgodnie z P_0 w permutacji P uzyskamy permutację końcową P_0. Ponieważ każdy krok związany jest z jednym przestawieniem prostym, liczba prostych przestawień jest również równa liczbie par uporządkowanych niezgodnie z P_0 w permutacji P. Jest to minimalna liczba przestawień prostych konieczna do przekształcenia permutacji P w permutację P_0, gdyż za pomocą jednego przestawienia prostego można uzyskać zmianę uporządkowania tylko jednej pary elementów.

Wnioskiem z powyższych rozważań jest stwierdzenie, że każdy algorytm sortowania przez przestawienia proste będzie realizował sortowanie przy minimalnej liczbie przestawień, jeśli przestawienia dokonywane będą tylko na parach elementów uporządkowanych niezgodnie z P_0.

W celu określenia średniej liczby przestawień prostych przy sortowaniu zbioru n-elementowego udowodnimy na wstępie, że rozkład liczby permutacji zawierających tę samą liczbę par uporządkowanych niezgodnie z P_0 jest symetryczny. Inaczej mówiąc, że liczba permutacji zawierających k par uporządkowanych niezgodnie z P_0 jest równa liczbie permutacji zawierających $\binom{n}{2} - k$ par uporządkowanych niezgodnie z P_0.

Dowód powyższej tezy rozpoczniemy od udowodnienia twierdzenia pomocniczego, mówiącego, że istnieje dokładnie jedna taka permutacja $\bar{P}_0$, która zawiera $\binom{n}{2}$ par uporządkowanych niezgodnie z P_0. (W P_0 wszystkie pary elementów są uporządkowane niezgodnie z P_0).

Załóżmy, że istnieją dwie różne permutacje $\bar{P}_0$ i $\bar{P}'_0$ zawierające $\binom{n}{2}$ par uporządkowanych niezgodnie z P_0. Ponieważ $\bar{P}'_0$ jest permutacją różną od $\bar{P}_0$, musi więc zawierać co najmniej jedną parę uporządkowaną niezgodnie z $\bar{P}_0$. Ale para ta byłaby uporządkowana zgodnie z P_0 (w $\bar{P}_0$ wszystkie pary są uporządkowane niezgodnie z P_0), co przeczy początkowemu założeniu. Tak więc istnieje dokładnie jedna permutacja $\bar{P}_0$, w której wszystkie pary są uporządkowane niezgodnie z P_0.

Permutację $\bar{P}_0$ można wyznaczyć w następujący sposób. Jeżeli permutacja P_0 opisana jest ciągiem pozycji

$$P_0 = \{p_1, p_2, \ldots, p_n\}$$

to $\bar{P}_0$ jest opisana następującym ciągiem pozycji

$$\bar{P}_0 = \{\bar{p}_1, \bar{p}_2, \ldots, \bar{p}_n\}$$

przy czym $\bar{p}_i = n - p_i + 1$. Łatwo sprawdzić, że dla każdej pary elementów a_i i a_j $p_i < p_j$ wtedy i tylko wtedy, gdy $\bar{p}_i > \bar{p}_j$. Nie ma więc w $\bar{P}_0$ ani jednej pary uporządkowanej zgodnie z P_0.

Niech $P_1, P_2, \ldots, P_m$ będą permutacjami zawierającymi k par uporządkowanych niezgodnie z P_0. Wynika z tego, że $\bar{P}_1, \bar{P}_2, \ldots, \bar{P}_m$ są permutacjami zawierającymi k par uporządkowanych niezgodnie z $\bar{P}_0$. Ale wszystkie pary zawarte w $\bar{P}_0$ są uporządkowane niezgodnie z P_0, tak więc każda permutacja zawierająca k par uporządkowanych niezgodnie z $\bar{P}_0$ posiada tym samym k par uporządkowanych zgodnie z P_0. Oznacza to, że każda permutacja zawierająca k par uporządkowanych niezgodnie z $\bar{P}_0$ zawiera jednocześnie $\binom{n}{2} - k$ par uporządkowanych niezgodnie z P_0. Wynika stąd, że liczba permu-

tacji zawierających k par uporządkowanych niezgodnie z P_0 równa jest liczbie permutacji zawierających $\binom{n}{2}$ - k par uporządkowanych niezgodnie z P_0.

Kończy to dowód twierdzenia o symetrii rozkładu liczby permutacji zawierających tę samą liczbę par uporządkowanych niezgodnie z P_0.

Z powyższego dowodu wynika prosty wniosek, że rozkład liczby permutacji, których przekształcenia do permutacji P_0 dokonuje się za pomocą tej samej liczby przestawień prostych (przy zastosowaniu algorytmu optymalnego) jest symetryczny.

Określmy na zbiorze wszystkich możliwych permutacji n-elementowych zmienną losową, która dla każdej permutacji równa jest liczbie przestawień prostych, które wykonujemy przekształcając tę permutację (wg optymalnego algorytmu) w pewną ustaloną permutację P_0. Jeśli założymy, że wszystkie permutacje mogą wystąpić z równym prawdopodobieństwem, to na podstawie wniosku z twierdzenia o symetrii rozkładu liczby permutacji zawierających tę samą liczbę par uporządkowanych niezgodnie z P_0 możemy stwierdzić, że zmienna losowa określona wyżej ma rozkład symetryczny. Korzystając z tej właściwości rozkładu możemy policzyć wartość oczekiwaną tej zmiennej losowej jako średnią arytmetyczną dwóch jej krańcowych wartości. Tymi krańcowymi wartościami są: 0 dla permutacji P_0 oraz $\binom{n}{2}$ dla permutacji $\overline{P}_0$. Wartość oczekiwana, będąca jednocześnie średnią liczbą przestawień prostych (S) przy sortowaniu zbioru n-elementowego, wynosi w związku z tym:

$$S = \frac{0 + \binom{n}{2}}{2} = \frac{n(n-1)}{4}$$

Jest to średnia liczba przestawień prostych dla każdego algorytmu sortującego przez przestawienia proste, który realizuje sortowanie przy minimalnej liczbie przestawień.

4. SORTOWANIE PRZEZ WSTAWIENIA

W praktycznej realizacji algorytmu sortowania średnia liczba przestawień prostych nie może stanowić pełnego wskaźnika efektywności. W najgorszym bowiem przypadku należałoby sprawdzić (przez porównanie) $\binom{n}{2}$ par elementów, celem określenia, czy ich uporządkowanie jest zgodne czy niezgodne z permutacją P_0 stanowiącą zbiór uporządkowany, przy czym wykonanoby średnio $\frac{1}{2}\binom{n}{2}$ przestawień. Oznacza to, że średnio połowa porównań nie będzie wprowadzała konkretnych postępów w sortowaniu. Każde natomiast porównanie wymaga sprowadzenia z pamięci porównywanych elementów, co wpływa na całkowity czas realizacji algorytmu. Dlatego dobry algorytm powinien charakteryzować się jak najmniejszą liczbą tych "niepotrzebnych" porównań, które nie prowadzą do przestawień.

Algorytmem, który wydaje się być efektywny pod tym względem jest sortowanie przez wstawienia (by inserting). Ogólnie metoda ta polega na dołączaniu do już uporządkowanego zbioru jednego elementu i drogą przestawień prostych wstawienia go na odpowiednie miejsce.

Załóżmy, że w zbiorze n-elementowym mamy na k pierwszych pozycjach elementy posortowane. Element stojący na k+1 pozycji dołączamy do uporządkowanej części zbioru w sposób następujący: porównujemy go z elementem znajdującym się na pozycji k-tej i jeśli stwierdzimy, że powinien on poprzedzać ten element dokonujemy przestawienia prostego tych elementów, dalej porównujemy dołączany element (znajdujący się już na pozycji k-tej) z elementem znajdującym się aktualnie na pozycji k-1 i przestawiamy w tym wypadku, gdy dołączany element powinien poprzedzać ten element. Przestawienia kontynuujemy aż do napotkania elementu, który poprzedza element dołączany. Napotkanie takiego elementu oznacza, że dołączany element został wstawiony na właściwą pozycję. Do powstałego w ten sposób uporządkowanego zbioru k+1 elementów dołączamy następnie element znajdujący się na pozycji k+2 itd.

Model sortowania, określony w sposób rekurencyjny, zaczyna działać od dołączania elementu do zbioru jednoelementowego (element na pozycji pierwszej), następnie do zbioru dwuelementowego itd. aż do wyczerpania elementów sortowanego zbioru. Każdy więc element (oprócz pierwszego) jest na określonym etapie sortowania elementem dołączanym.

Oznaczmy przez q_i liczbę przestawień prostych związaną z dołączaniem elementu znajdującego się w wyjściowym zbiorze na pozycji i-tej. Każde przestawienie związane jest z jednym porównaniem pary elementów. Tak więc element znajdujący się w wyjściowym zbiorze na pozycji i-tej jest porównywany q_i razy z elementami, z którymi jest przestawiany oraz raz z elementem, na którym kończy się proces dołączania. Stąd całkowita liczba porównań T równa jest

$$T = \sum_{i=2}^{n} (q_i + 1) = (n - 1) + \sum_{i=2}^{n} q_i .$$

Ale suma wszystkich przestawień prostych $\sum_{i=2}^{n} q_i$ równa jest średniej liczbie przestawień prostych, a więc $\frac{1}{2}\binom{n}{2}$. Stąd:

$$T = (n - 1) + \frac{1}{2}\binom{n}{2} = \frac{(n - 1)(n + 4)}{4} .$$

Na zakończenie należy zauważyć, że w tym algorytmie liczba porównań, z których nie wynikają przestawienia jest równa n - 1, a więc znacznie mniej od średniej liczby przestawień prostych. Stąd wniosek o dużej efektywności tego algorytmu wśród algorytmów wykorzystujących metodę przestawień prostych.

Klasa algorytmów przedstawiona powyżej nie stanowi grupy najszybszych algorytmów sortowania. Niewątpliwą jednak zaletą tych algorytmów jest realizacja sortowania bez koniecz-

ności rezerwowania w pamięci dodatkowego obszaru roboczego. Algorytmy te mogą więc być stosowane wszędzie tam, gdzie zbiór sortowany tak wypełnia pamięć wewnętrzną, że nie można zarezerwować odpowiednio dużego obszaru roboczego.

Literatura

[1] GOTLIEB C.C.: High-Speed Data Processing, McGraw-Hill, New York 1958.

[2] GOTLIEB C.C.: Sorting on Computers, Comm. of the ACM, 1963:6,5.

[3] MEADOW C.T.: The Analysis of Information Systems, John Wiley and Sons, Inc., New York 1967.

[4] GLUŠKOV V.M., GLADUN V.P., LOZINSKIJ L.S., POGREBINSKIJ S.B.: Obrabotka informacjonnych massivov v avtomatizirovannych sistemach upravlenija, Kijev 1970.

СОРТИРОВКА МЕТОДОМ ПРОСТОЙ ПЕРЕСТАНОВКИ

Резюме

Целью статьи является определить эффективность алгоритма сортировки методом вставки, принимая указателем эффективности среднее число сравнений.

Первая часть работы заключает общий анализ класса алгоритмов сортировки, использующих метод простых перестановок. Определяя сортировку как преобразование любой n-элементной перестановки P в некую перестановку P_0, определено минимальное число простых перестановок, нужное для сортировки n-элементного множества, а потом усредниьая эту величину для всех возможных перестановок P, получено среднее число простых перестановок (S) в процессе сортировки n-элементного множества.

На основе этих общих рассуждений, во второй части статьи проведено анализ алгоритма сортировки методом вставки, а также выведено формулу для среднего числа сравнений (T) в процессе сортировки n-элементного множества.

SORTING BY THE METHOD OF SIMPLE EXCHANGE

Summary

The article aims at a determination of the sorting algorithm by inserting after the coefficient of effectiveness of mean number comparison had been accepted.

The first part of the paper includes a general analysis of a class of sorting algorithms by simple exchange method. Sorting is understood as a transformation of any n-element P permutation into a certain, fixed in advance, n-element P_0 permutation. Due to this the minimum number of simple exchange is determined, needed for an n-element set sorting. Then, finding the mean value of this magnitude, after all possible permutations P, the mean number of simple exchange (S) was obtained while sorting the n-element set.

On the basis of these general considerations the second part of the paper comprises an analysis of the sorting algorithm by inserting. Also the formula is written for the mean number of comparisons (T) while sorting an n-element set.

ALGORYTMY, X, N° 17, 1973, 45-57 681.322.056.06.004.15:519.5

OCENA EFEKTYWNOŚCI ALGORYTMU SCALANIA DWÓCH UPORZĄDKOWANYCH ZBIORÓW

Henryk RYBIŃSKI
Studium Doktoranckie
Politechniki Warszawskiej

Pracę złożono 15.04.1972

W artykule dokonano oceny efektywności procesu scalania dwóch uporządkowanych zbiorów. Wyznaczono dokładne wyrażenie na średnią liczbę porównań realizowanych w procesie scalania. Na podstawie uzyskanego wyniku podano oszacowanie średniej liczby porównań realizowanych przez algorytmy sortowania Two-Way Merge Sort i Insertion Sort.

1. WSTĘP

Do najczęściej stosowanych procedur w systemach przetwarzania danych należą procedury sortowania. Istnieje cała klasa algorytmów sortowania realizujących porządkowanie zbiorów danych na drodze kolejnych porównań parami elementów sortowanych. Często stosowaną miarą efektywności algorytmów tej klasy jest wartość oczekiwana liczby porównań wykonywanych w procesie porządkowania przy założeniu, że każda początkowa sekwencja danych jest jednakowo prawdopodobna. W klasie tej wyróżnić można taką podklasę algorytmów sortowania, w których daje się wyodrębnić pewne kroki charakteryzujące się scalaniem wcześniej uporządkowanych podzbiorów zbioru porządkowanego.

Celem niniejszej pracy jest ocena efektywności procesu scalania dwóch zbiorów uporządkowanych. Ocena ta oparta jest na wyznaczeniu średniej liczby porównań realizowanych podczas scalania. Jest ona przydatna przy badaniu efektywności algorytmów sortowania wyróżnionej wyżej podklasy.

Dalej oznaczać będziemy przez $\overline{\overline{P}}$ moc zbioru P. Przyjmijmy dla uproszczenia, że zbiory scalane A i B spełniają $A \subset N$ i $B \subset N$, gdzie N oznacza zbiór liczb naturalnych, oraz, że relacją liniowo porządkującą każdy z tych zbiorów jest relacja $<$. O zbiorze $P \subset N$ będziemy mówili, że jest uporządkowany w pamięci maszyny cyfrowej, jeżeli dla dowolnych elementów $p_r \in P$ i $p_s \in P$ znajdujących się na miejscach pamięci o numerach r i s spełnione jest $p_r < p_s$ wtedy i tylko wtedy gdy $r < s$. Załóżmy, że zbiory A i B są uporządkowane w pamięci. Przyjmijmy $\overline{\overline{A}} = k$ oraz $\overline{\overline{B}} = l$. Ponadto niech $A \cap B = \emptyset$. Wynikiem scalenia takich zbiorów jest zbiór $A \cup B$ uporządkowany w pamięci maszyny cyfrowej.

Przez $\left|\begin{smallmatrix} a_i \\ b_j \end{smallmatrix}\right|$ oznaczać będziemy czynność porównania elementów $a_i \in A$ i $b_j \in B$. Wynik porównania determinuje właściwe ustawienie wzajemne elementów w pamięci i wybór kolejnego porównania, ewentualnie zakończenie procesu scalania.

Przy ocenie efektywności algorytmu scalania wygodnie jest przedstawić go za pomocą następującej tablicy:

$$
\begin{array}{ccccccc}
\left|\begin{smallmatrix} a_1 \\ b_1 \end{smallmatrix}\right| & \rightarrow & \left|\begin{smallmatrix} a_1 \\ b_2 \end{smallmatrix}\right| & \rightarrow & \left|\begin{smallmatrix} a_1 \\ b_3 \end{smallmatrix}\right| & \rightarrow \cdots \rightarrow & \left|\begin{smallmatrix} a_1 \\ b_l \end{smallmatrix}\right| \\
\downarrow & & \downarrow & & \downarrow & & \downarrow \\
\left|\begin{smallmatrix} a_2 \\ b_1 \end{smallmatrix}\right| & \rightarrow & \left|\begin{smallmatrix} a_2 \\ b_2 \end{smallmatrix}\right| & \rightarrow & \left|\begin{smallmatrix} a_2 \\ b_3 \end{smallmatrix}\right| & \rightarrow \cdots \rightarrow & \left|\begin{smallmatrix} a_2 \\ b_l \end{smallmatrix}\right| \\
\downarrow & & \downarrow & & \downarrow & & \\
\cdots & & \cdots & & \cdots & & \\
\cdots & & \cdots & & \cdots & & \\
\downarrow & & \downarrow & & \downarrow & & \downarrow \\
\left|\begin{smallmatrix} a_k \\ b_1 \end{smallmatrix}\right| & \rightarrow & \left|\begin{smallmatrix} a_k \\ b_2 \end{smallmatrix}\right| & \rightarrow & \left|\begin{smallmatrix} a_k \\ b_3 \end{smallmatrix}\right| & \rightarrow \cdots \rightarrow & \left|\begin{smallmatrix} a_k \\ b_l \end{smallmatrix}\right|
\end{array}
$$

Algorytm rozpoczyna działanie od porównania $\left|\begin{matrix}a_1\\b_1\end{matrix}\right|$, po czym w zależności od wyniku przechodzi do porównania $\left|\begin{matrix}a_1\\b_2\end{matrix}\right|$, gdy $a_1 > b_1$ lub do $\left|\begin{matrix}a_2\\b_1\end{matrix}\right|$, gdy $a_1 < b_1$. Od porównania $\left|\begin{matrix}a_i\\b_j\end{matrix}\right|$ algorytm w zależności od wyniku przechodzi do porównania $\left|\begin{matrix}a_i\\b_{j+1}\end{matrix}\right|$, gdy $a_i > b_j$ lub do $\left|\begin{matrix}a_{i+1}\\b_j\end{matrix}\right|$, gdy $a_i < b_j$. Proces scalania kończy się na porównaniu $\left|\begin{matrix}a_i\\b_1\end{matrix}\right|$, gdy $a_i > b_1$, $i = 1, \ldots, k$ lub na porównaniu $\left|\begin{matrix}a_k\\b_j\end{matrix}\right|$, gdy $a_k < b_j$, $j = 1, \ldots 1$.

2. EFEKTYWNOŚĆ SCALANIA

Załóżmy, że dla dwóch zbiorów A i B jest $A \cap B = \emptyset$, oraz $\overline{\overline{A}} = k$, $\overline{\overline{B}} = l$. Mówić będziemy, że w permutacji elementów zbioru $A \cup B$ umieszczonej w pamięci maszyny na kolejnych k+l miejscach zachowane jest uporządkowanie elementów zbioru A, jeżeli zbiór ten pozostaje uporządkowany w pamięci.

TWIERDZENIE 2.1. Istnieje $\binom{k+l}{k}$ takich permutacji elementów zbioru $A \cup B$, w których zachowane jest uporządkowanie elementów zbioru A oraz uporządkowanie elementów zbioru B.

Dowód: Oznaczmy zbiór szukanych permutacji przez Q, wówczas szukana liczba permutacji jest równa $\overline{\overline{Q}}$. Przez dowolne permutowanie elementów b_i zbioru B względem siebie na pozycjach zajmowanych przez elementy zbioru B można dla każdego ciągu $q \in Q$ uzyskać l! ciągów, w których warunkiem jest jedynie zachowanie uporządkowania elementów a_i zbioru A. Zbiór R ciągów, w których warunkiem jest uporządkowanie a_i jest więc mocy $\overline{\overline{Q}} \cdot l!$.

Dalej, dla każdego ciągu $r \in R$ można uzyskać k! ciągów permutując elementy a_i zbioru A. Ponieważ daje to zbiór wszystkich możliwych permutacji k+l elementów, zatem

$$\overline{\overline{Q}} \cdot (k!)(l!) = (k+l)! \qquad \text{stąd zaś}$$

$$\overline{\overline{Q}} = \frac{(k+l)!}{(k!) \cdot (l!)} = \binom{k+l}{k} \qquad \text{c.n.d.}$$

Tak więc w wyniku scalania uporządkowanych zbiorów A i B zostanie utworzona jedna z $\binom{k+l}{k}$ permutacji elementów zbioru $A \cup B$. Każde uporządkowanie końcowe zależne jest od przypadkowych wartości elementów zbiorów scalanych. Ponieważ wystąpienie w zbiorach scalanych każdego elementu należącego do zbioru liczb naturalnych N jest jednakowo prawdopodobne, więc każde uporządkowanie końcowe jest również jednakowo prawdopodobne.

Z jednoznaczności działania algorytmu scalania wynika, że każde z możliwych uporządkowań końcowych elementów zbiorów A i B realizowane jest przez dokładnie jedną sekwencję porównań.

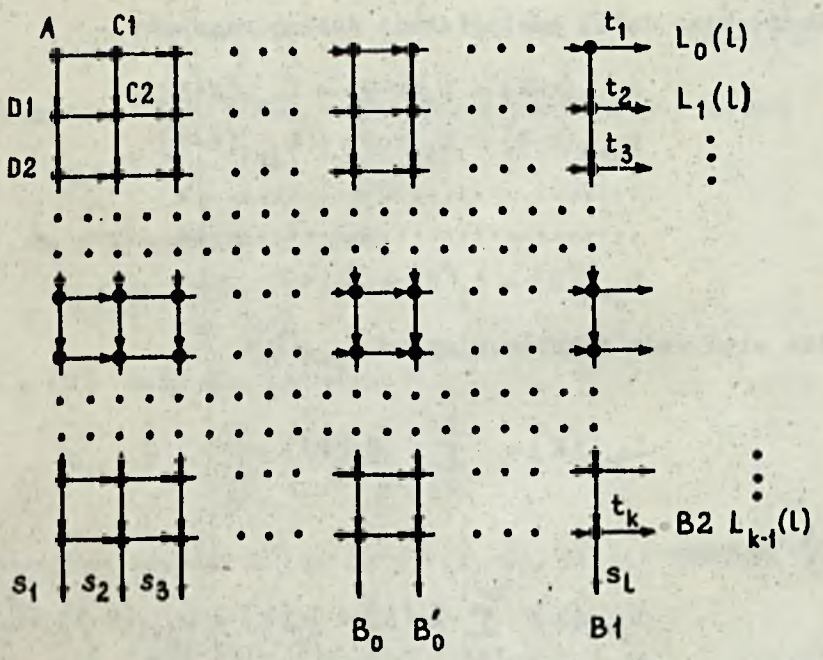
Z jednoznaczności działania algorytmu wynika także, że sekwencja porównań zakończona porównaniem $\left|\begin{smallmatrix} a_i \\ b_1 \end{smallmatrix}\right|$, $1 \leq i \leq k-1$, lub $\left|\begin{smallmatrix} a_k \\ b_j \end{smallmatrix}\right|$, $1 \leq j \leq l-1$, daje tylko jedno uporządkowanie końcowe. Jedynie sekwencjom zakończonym porównaniem $\left|\begin{smallmatrix} a_k \\ b_1 \end{smallmatrix}\right|$ odpowiadają w zależności od wyniku dwa możliwe uporządkowania końcowe: $\ldots\ldots a_k b_1$, lub $\ldots\ldots b_1 a_k$. Wynika to z faktu, że porównanie $\left|\begin{smallmatrix} a_k \\ b_1 \end{smallmatrix}\right|$ niezależnie od wyniku zawsze kończy proces scalania.

Powyższe rozumowanie prowadzi do wniosku, że wyznaczenie średniej liczby porównań w procesie scalania sprowadza się do wyznaczenia średniej długości drogi, zaczynającej się w punkcie A prostokąta (rys. 1) i kończącej się strzałką s_p, $p = 1, \ldots, l$, lub t_q, $q = 1, \ldots k$, przy założeniu, że każda z dróg jest jednakowo prawdopodobna, zaś długość drogi jest rozumiana jako liczba strzałek. Istotnie, tak rozumiana długość drogi jest równa liczbie porównań w sekwencji porównań realizującej scalanie, a ponadto każdej drodze odpowiada jedno uporządkowanie końcowe (każdej sekwencji porównań zakończonej w tablicy scalania porównaniem $\left|\begin{smallmatrix} a_k \\ b_1 \end{smallmatrix}\right|$ odpowiada na rys. 1 jedna z dwóch dróg, zależnie od wyniku tego porównania).

Zauważmy, że długość wszystkich sekwencji zakończonych porównaniem $\left|\begin{smallmatrix} a_k \\ b_{j+1} \end{smallmatrix}\right|$ jest dla danego j, $j = 0, 1, \ldots, l-1$, jednakowa i równa się $k+j$; podobnie długość wszystkich sekwencji

zakończonych porównaniem $\left|\begin{matrix} a_{i+1} \\ b_1 \end{matrix}\right|$ jest jednakowa dla danego i, i = 0, 1,..., k-1, i równa jest l+i. Oznaczmy liczbę możliwych uporządkowań końcowych, odpowiadających wszystkim sekwencjom zakończonym porównaniem $\left|\begin{matrix} a_k \\ b_{j+1} \end{matrix}\right|$, przez $L_j(k)$. Innymi słowy $L_j(k)$ jest to liczba dróg w prostokącie na rys. 1 z punktu A do punktu B_0. Zgodnie z tym $L_{l-1}(k)$ jest równe liczbie możliwych przejść do punktów B_1 i B_2 łącznie. Z rysunku wynika wprost, że $L_0(k)$ dla dowolnego $k \geqslant 1$ (tj. liczba sekwencji zakończonych porównaniem $\left|\begin{matrix} a_k \\ b_1 \end{matrix}\right|$) jest równe 1. Podobnie dla $l \geqslant 1$ mamy $L_{l-1}(1) = 1$. Ponadto, ponieważ liczba przejść w prostokącie o długościach boków $(j+1)$ x l jest równa liczbie przejść w prostokącie $lx(j+1)$ więc prawdziwe jest

$$L_j(k) = L_{k-1}(j+1) \qquad (1)$$



Rys. 1

W zbiorze dróg do punktu B_o' (rys. 1) o długości k+j+1 wyróżnić można dwa dopełniające się podzbiory, z których jeden obejmuje drogi przechodzące przez punkt C1, a drugi przez D1; ponieważ liczba dróg przechodzących przez C1 równa jest $L_j(k)$, zaś liczba dróg przechodzących przez D1 jest $L_{j+1}(k-1)$, zatem

$$L_{j+1}(k) = L_j(k) + L_{j+1}(k-1)$$

Zbiór dróg przechodzących przez D1 rozbijamy na dwa podzbiory, z których jeden obejmuje drogi przechodzące przez punkt C2, a drugi przez D2. Zauważając, że liczba dróg wychodzących z D1 przez C2 do B_o' równa jest $L_j(k-1)$, zaś liczba dróg przechodzących przez D2 jest równa $L_{j+1}(k-2)$, mamy

$$L_{j+1}(k-1) = L_j(k-1) + L_{j+1}(k-2)$$

Postępując dalej analogicznie możemy uzyskać:

$$L_{j+1}(k-2) = L_j(k-2) + L_{j+1}(k-3)$$
$$L_{j+1}(k-3) = L_j(k-3) + L_{j+1}(k-4)$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$L_{j+1}(2) = L_j(2) + L_j(1)$$

Wynika stąd wzór rekurencyjny na $L_{j+1}(k)$:

$$L_{j+1}(k) = \sum_{i=1}^{k} L_j(i) \qquad (2)$$

Z (2) wynika:

$$L_{j+1}(k) = \sum_{i=1}^{k-1} L_j(i) + L_j(k) = L_{j+1}(k-1) + L_j(k)$$

analogicznie:

$$L_j(k) = L_j(k-1) + L_{j-1}(k)$$
$$L_{j-1}(k) = L_{j-1}(k-1) + L_{j-2}(k)$$
$$\cdots\cdots\cdots\cdots\cdots\cdots$$
$$\cdots\cdots\cdots\cdots\cdots\cdots$$
$$L_2(k) = L_2(k-1) + L_1(k)$$
$$L_1(k) = L_1(k-1) + L_0(k) = L_1(k-1) + L_0(k-1)$$

stąd zaś mamy:

$$L_{j+1}(k) = \sum_{i=0}^{j+1} L_i(k-1) \tag{3}$$

TWIERDZENIE 2.2. Dla każdego k naturalnego oraz $j \geq 0$ $L_j(k)$ wyraża się wzorem:

$$L_j(k) = \binom{k+j-1}{j} \tag{4}$$

Dowód: Ponieważ $L_0(k) = 1$, więc bezpośrednio z (2) wynika

$$L_1(k) = k,$$

załóżmy, że dla pewnego j jest

$$L_j(k) = \binom{k+j-1}{j}$$

wówczas z (2) mamy dla każdego k

$$L_{j+1}(k) = \sum_{i=1}^{k} \binom{i+j-1}{j}$$

Dowód zatem sprowadza się do wykazania, że dla każdego k zachodzi:

$$\sum_{i=1}^{k} \binom{i+j-1}{j} = \binom{k+j}{j+1} \tag{5}$$

Równość (5) jest spełniona dla k=1. Załóżmy, że jest ona spełniona także dla k=p, tj.

$$L_{j+1}(p) = \sum_{i=1}^{p} \binom{i+j-1}{j} = \binom{p+j}{j+1}.$$

Stąd dla p+1 jest

$$L_{j+1}(p+1) = \sum_{i=1}^{p+1} \binom{i+j-1}{j} = \sum_{i=1}^{p} \binom{i+j-1}{j} + \binom{p+j}{j} =$$

$$= \binom{p+j}{j+1} + \binom{p+j}{j} = \binom{p+j+1}{j+1} \qquad \text{c.n.d.}$$

Na podstawie powyższych twierdzeń można wyznaczyć średnią liczbę porównań realizowanych w procesie scalania uporządkowanych zbiorów A i B odpowiednio k i l-elementowych. Dla ustalenia uwagi załóżmy, że $l \leqslant k$, tj. $k-l = p$, $p \geqslant 0$. Oznaczmy przez $h_i(k,l)$ liczbę końcowych uporządkowań zbioru $A \cup B$, dla realizacji których algorytm scalania dokonuje l+i porównań, mamy wówczas:

$$h_0(k, l) = L_0(l)$$
$$h_1(k, l) = L_1(l)$$
$$\cdots\cdots\cdots\cdots\cdots$$
$$\cdots\cdots\cdots\cdots\cdots$$
$$h_{p-1}(k, l) = L_{p-1}(l)$$
$$h_p(k, l) = L_p(l) + L_0(k)$$
$$h_{p+1}(k, l) = L_{p+1}(l) + L_1(k)$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$h_{k-2}(k,l) = L_{k-2}(l) + L_{k-2-p}(k) = L_{k-2}(l) + L_{l-2}(k)$$
$$h_{k-1}(k,l) = L_{k-1}(l) + L_{l-1}(k) = 2L_{k-1}(l) = 2L_{l-1}(k)$$

przy czym ostatnia równość wynika z (1).

Ponieważ średnia liczba porównań realizowanych podczas scalania wyraża się wzorem

$$s(k,l) = \sum_{i=0}^{k-1} \frac{1}{\binom{k+l}{k}} \cdot h_i(k,l) \cdot (l+i),$$

więc

$$s(k,l) = \frac{1}{\binom{k+l}{k}} \left[\sum_{i=0}^{k-1} L_i(l).(l+i) + \sum_{i=0}^{l-1} L_i(k).(k+i) \right] =$$

$$= \frac{1 + \frac{l(l+1)}{1!} + \frac{l.(l+1).(l+2)}{2!} + \ldots + \frac{l.(l+1).\ldots.(l+k-2).(l+k-1)}{(k-1)!}}{\binom{k+l}{k}} +$$

$$+ \frac{k + \frac{k(k+1)}{1!} + \frac{k.(k+1).(k+2)}{2!} + \ldots + \frac{k.(k+1).\ldots.(k+l-2).(k+l-1)}{(l-1)!}}{\binom{k+l}{k}} =$$

$$= \frac{1}{\binom{k+l}{k}} \cdot \left[l \cdot \sum_{i=0}^{k-1} L_i(l+1) + k \cdot \sum_{i=0}^{l-1} L_i(k+1) \right] =$$

$$= \frac{1}{\binom{k+l}{k}} \cdot \left[l \cdot L_{k-1}(l+2) + k.L_{l-1}(k+2) \right],$$

przy czym ostatnia równość wynika z zastosowania równania (3).

Dalej po przekształceniu tego wyrażenia uzyskujemy wzór na wartość średniej liczby porównań

$$s(k,l) = \frac{l.k}{(l+1)} + \frac{l.k}{(k+1)} \qquad (6)$$

W dalszej części pracy pokazano możliwości wykorzystania tej oceny dla dokładnego wyznaczenia efektywności dwóch algorytmów sortowania, z których jeden jest znany pod nazwą TWO-WAY MERGE SORT (sortowanie przez scalanie), zaś drugi - INSERTION SORT (sortowanie przez dostawianie).

3. OCENA EFEKTYWNOŚCI ALGORYTMÓW TWO-WAY MERGE SORT ORAZ INSERTION SORT

Dla wyznaczenia średniej wartości liczby porównań realizowanych przez algorytm TWO-WAY MERGE SORT (T-WMS) przyjmijmy upraszczająco, że zbiór sortowany liczy 2^n elementów, wówczas porządkowanie tego zbioru będzie realizowane w n krokach, przy czym w każdym kroku następuje scalanie uporządkowanych podzbiorów o tej samej liczbie elementów. Proces powtarza się, aż liczba elementów w zbiorze scalonym obejmie cały zbiór sortowany. Gdy więc sortowany jest zbiór 2^n elementów, to w i-tym kroku scalane są podzbiory uporządkowane liczące po 2^{i-1} elementów. Pokazuje to poniższa tabela:

Nr kroku	liczba podzbiorów	długość poj. podzbioru	średnia liczba porównań w kroku
1	2^n	1	$2^{n-1} \cdot s(1,1)$
2	2^{n-1}	2	$2^{n-2} \cdot s(2,2)$
3	2^{n-2}	4	$2^{n-3} \cdot s(4,4)$
.....	.....	.....	.....
.....	.....	.....	.....
i	2^{n-i+1}	2^{i-1}	$2^{n-1} \cdot s(2^{i-1}, 2^{i-1})$
.....	.....	.....	.....
n	2	2^{n-1}	$1 \cdot s(2^{n-1}, 2^{n-1})$

ponieważ zaś wg (6)

$$s(2^i, 2^i) = \frac{2^{2i+1}}{2^i + 1}$$

więc średnia liczba porównań S dla tego algorytmu jest

$$S = \sum_{i=1}^{n} 2^{n-i} \cdot s\left(2^{i-1}, 2^{i-1}\right) = \sum_{i=1}^{n} \frac{2^{n+i-1}}{2^{i-1}+1} =$$

$$= 2^n \sum_{i=1}^{n}\left(1 - \frac{1}{2^{i-1}+1}\right) = 2^n \cdot n - 2^n \cdot \sum_{i=1}^{n} \frac{1}{2^{i-1}+1} \qquad (7)$$

Na podstawie tego wzoru można oszacować S następująco:

$$S \leqslant 2^n (n-1) + 1$$

Dolne ograniczenie S można oszacować zapisując

$$S = \sum_{i=1}^{n} 2^{n-i}\left[2^i - 2 + \frac{2}{2^{i-1}+1}\right] = \qquad (7a)$$

$$= n \cdot 2^n - 2\left(2^n - 1\right) + 2^{n+1} \sum_{i=1}^{n} \frac{1}{2^i\left(2^{i-1}+1\right)}$$

Z wzoru (7a) wynika

$$S > 2^n\left(n - \frac{4}{3}\right)$$

zatem można zapisać

$$2^n\left(n - \frac{4}{3}\right) < S \leqslant 2^n (n-1) + 1$$

Wzór (6) daje też możliwość wyznaczenia efektywności algorytmu INSERTION SORT (IS). Ogólnie metoda ta polega na dołączeniu do już uporządkowanego zbioru jednego elementu i drogą przestawień wstawienia go na odpowiednie miejsce. Tak więc wyróżnić można w przypadku sortowania N elementów N-1 kroków, przy czym krok i-ty jest w zasadzie scalaniem uporządkowanych zbiorów: i-elementowego i jednoelementowego. Zatem zgodnie z (6) średnia liczba porównań w kroku i-tym jest

$$s_i = \frac{i \cdot 1}{i+1} + \frac{1 \cdot 1}{2}$$

Stąd zaś

$$S = \sum_{i=1}^{N-1} \left(\frac{i}{i+1} + \frac{i}{2} \right) = \frac{1}{2} \cdot \frac{(N-1) \cdot N}{2} + (N-1) - \sum_{i=1}^{N-1} \frac{1}{i+1} =$$

$$= \frac{1}{4} (N-1) \cdot (N+4) - \sum_{i=1}^{N-1} \frac{1}{i+1} . \qquad (8)$$

Korzystając z definicji stałej Eulera uzyskujemy dla dużych N dość dokładną aproksymację wyrażenia na S:

$$S = \frac{1}{4} (N-1) \cdot (N+4) - \ln (N+1) + 0{,}4228.$$

Literatura

[1] BEUS H.L.: The Use of Information in Sorting, J. of the ACM, No 3, July 1970.

[2] BOSE R.C., NELSON R.I.: A Sorting Problem, J. of the ACM, 1962:6,5.

[3] GLUŠKOV V.M. i inni: Obrabotka informacjonnych massivov v avtomatizirovannych sistemach upravlenija, Izd. Naukova Dumka, Kijev 1970.

[4] GOTLIEB C.C.: Sorting on Computers, Comm. of the ACM, 1963:6,5.

[5] MEADOW C.T.: The Analysis of Information Systems, John Wiley and Sons, NY 1967.

ОЦЕНКА ЭФФЕКТИВНОСТИ АЛГОРИТМА СОЕДИНЕНИЯ ДВУХ УПОРЯДОЧЕННЫХ МНОЖЕСТВ

Резюме

Рассмотрен процесс соединения двух упорядоченных множеств. Алгоритм соединения представлен в виде таблицы соединений. Принято, что любой элемент может выступать в соединённых множествах с одинаковой вероятностью. Показано, что среднее число сравнений в процессе соединения равняется средней длине пути в некотором определённом прямоугольнике. Потом определено среднее число сравнений. Полученный результат послужил для оценки эффективности двух алгоритмов сортировки, известных под названием "Two-Way-Merge-Sort" и "Insertion Sort".

THE EVALUATION OF THE MERGING ALGORITHM EFFECTIVENESS OF TWO ORDERED SETS

Summary

The process of two ordered sets merging is considered. The merging algorithm is presented in the form of a merging table. It has been accepted that the appearance of every element in merged sets is equally probable, and with this assumption it was shown that the mean number of comparisons realized in the merging process equals the mean length of a way in a certain defined rectangle. Then a mean number of comparisons was determined. The obtained result served to evaluate the effectiveness of two sorting algorithms known as Two-Way Merge Sort and Insertion Sort.

WYDAWNICTWA IMM

Branżowy Ośrodek Informacji Naukowo-Technicznej i Ekonomicznej Instytutu Maszyn Matematycznych wydaje:

ALGORYTMY - półrocznik, zawiera artykuły na temat teorii programowania i zastosowania elektronicznych maszyn cyfrowych. Do nabycia w księgarni ORWN PAN oraz w Domach Książki. Cena zeszytu 40,- zł.

PRACE IMM - 3 numery w roku, zawierają publikacje naukowe i badawcze pracowników IMM w zakresie projektowania i budowy elektronicznych maszyn cyfrowych oraz systemów przetwarzania informacji. Do nabycia w księgarni ORWN PAN oraz w Domach Książki. Cena zeszytu 60,- zł.

Elektroniczna Technika Obliczeniowa - NOWOŚCI - kwartalnik, zawiera artykuły przeglądowe z dziedziny maszyn matematycznych, opracowane na podstawie najnowszej literatury światowej. Prenumeratę prowadzi Centrala Kolportażu Prasy i Wydawnictw "RUCH". Cena prenumeraty krajowej 240,- zł rocznie.

Automatyzacja Przetwarzania Informacji - INFORMACJA EKSPRESOWA - miesięcznik. Prenumeratę prowadzi Centrala Kolportażu Prasy i Wydawnictw "RUCH". Cena prenumeraty krajowej 240,- zł rocznie.

Cena zł 40,–