mathematics

MDPI

Article

Method for Developing Combinatorial Generation Algorithms Based on AND/OR Trees and Its Application

Yuriy Shablya [1,*], Dmitry Kruchinin [1] and Vladimir Kruchinin [2]

[1] Department of Complex Information Security of Computer Systems, Tomsk State University of Control Systems and Radioelectronics, Tomsk 634050, Russia; kdv@fb.tusur.ru

[2] Institute of Innovation, Tomsk State University of Control Systems and Radioelectronics, Tomsk 634050, Russia; kru@2i.tusur.ru

* Correspondence: shablya-yv@mail.ru

Received: 6 May 2020; Accepted: 9 June 2020; Published: 12 June 2020

Abstract: In this paper, we study the problem of developing new combinatorial generation algorithms. The main purpose of our research is to derive and improve general methods for developing combinatorial generation algorithms. We present basic general methods for solving this task and consider one of these methods, which is based on AND/OR trees. This method is extended by using the mathematical apparatus of the theory of generating functions since it is one of the basic approaches in combinatorics (we propose to use the method of compositae for obtaining explicit expression of the coefficients of generating functions). As a result, we also apply this method and develop new ranking and unranking algorithms for the following combinatorial sets: permutations, permutations with ascents, combinations, Dyck paths with return steps, labeled Dyck paths with ascents on return steps. For each of them, we construct an AND/OR tree structure, find a bijection between the elements of the combinatorial set and the set of variants of the AND/OR tree, and develop algorithms for ranking and unranking the variants of the AND/OR tree.

Keywords: combinatorial generation; method; algorithm; AND/OR tree; Euler–Catalan's triangle; labeled Dyck path; ranking algorithm; unranking algorithm

MSC: 68R05; 05C05; 05A15

1. Introduction

Many information objects have a hierarchical or recursive structure. In this case, a tree structure is a convenient form of representing such information objects. This allows us to describe an information object by a combinatorial set and apply combinatorial generation algorithms for it. A combinatorial set is a finite set whose elements have some structure and there is an algorithm for constructing the elements of this set. The elements of combinatorial sets (combinatorial objects) like combinations, permutations, partitions, compositions, paths, graphs, trees, etc. play an important role in mathematics and computer science.

Knuth [1] gives a detailed overview of the formation and development of the direction related to designing combinatorial algorithms. In this overview, special attention is paid to the procedure for traversing all possible elements of a given combinatorial set. This problem can be studied as enumerating, listing, and generating elements of a given combinatorial set. On the other hand, Ruskey [2] introduces the concept of combinatorial generation and distinguishes the following four tasks in this area:

1. Listing: generating elements of a given combinatorial set sequentially;
2. Ranking: ranking (numbering) elements of a given combinatorial set;
3. Unranking: generating elements of a given combinatorial set in accordance with their ranks;
4. Random selection: generating elements of a given combinatorial set in random order.

Before applying combinatorial generation algorithms, it is necessary to develop them. General methods for developing combinatorial generation algorithms were studied by such researches as E.M. Reingold [3], D.L. Kreher [4], E. Barcucci [5,6], S. Bacchelli [7,8], A. Del Lungo [9,10], V. Vajnovszki [11–13], P. Flajolet [14,15], C. Martinez and X. Molinero [16–20], B.Y. Ryabko and Y.S. Medvedeva [21–23], and V.V. Kruchinin [24].

There are several basic general methods for developing combinatorial generation algorithms:

- backtracking [3,4]: this method is used for exhaustive generation and based on constructing a state space tree for a combinatorial set and obtaining feasible solutions on level l of the tree that are built up from partial solutions on level $l-1$;
- ECO-method [5–13]: this method is used for exhaustive generation and based on producing a set of new objects of size $n+1$, using a succession rule and starting from an object of size n;
- Flajolet's method [14–20]: this method can be applied for listing, ranking, and unranking a large family of unlabeled and labeled admissible combinatorial classes that can be obtained by using admissible combinatorial operators (disjoint union, Cartesian unlabeled product, labeled product, sequence, set, cycle, powerset, substitution, etc.);
- Ryabko's method [21–23]: this method can be applied for ranking and unranking combinatorial objects presented in the form of a word, this method operates with word prefixes and uses the divide-and-conquer paradigm;
- Kruchinin's method [24]: this method can be applied for listing, ranking, and unranking a combinatorial set represented in the form of an AND/OR tree structure for which the total number of its variants is equal to the value of the cardinality function of the combinatorial set.

Each of these methods claims the universality of its application in the development of new combinatorial generation algorithms. The study of these methods have shown the following results connected with their limitations and requirements [25]:

- the main characteristic, which is necessary for developing combinatorial generation algorithms, is the cardinality function of a combinatorial set;
- some methods (backtracking and ECO-method) are aimed only at the development of listing algorithms;
- there are restrictions on applying some methods (ECO-method and Flajolet's method) for combinatorial sets that are described by more than one parameter;
- most methods require the representation of a combinatorial object in a special form (for example, as a word, a sequence, a specification, or an AND/OR tree), but this is not always a trivial task and requires additional research;
- there are requirements for additional information describing a combinatorial set.

Also, there are many combinatorial generation algorithms that are based on features of the applied combinatorial set or that are based on simple counting techniques (for example, see [26–28]). Therefore, the methods used for developing such algorithms cannot be universal (they cannot be applied to develop new combinatorial generation algorithms for other combinatorial sets).

Thus, there is no universal general method that can be applied for developing new combinatorial generation algorithms. The main purpose of our research is to derive and improve general methods for developing combinatorial generation algorithms. In this paper, we consider and extend Kruchinin's method for developing combinatorial generation algorithms, which is based on the use of AND/OR trees. This method:

- allows us to develop all types of combinatorial generation algorithms (listing, ranking, and unranking algorithms);
- has no restrictions on the number of parameters that describe combinatorial sets (this allows us to consider complex discrete structures);
- requires only an expression of the cardinality function as additional information describing a combinatorial set.

However, to apply this method, it is necessary to know an expression of the cardinality function of a combinatorial set that must satisfy the following conditions:

- the expression of the cardinality function can contain only positive integers (let $\mathbb{N}$ denotes the set of natural numbers);
- the expression of the cardinality function can contain only such algebraic operations as addition (which is denoted by $+$) and multiplication (which is denoted by $\times$);
- the cardinality function may be recursively defined in terms of itself (let R denotes a primitive recursive function).

If an expression of the cardinality function f of a combinatorial set A satisfies the conditions presented above, then we will say that f belongs to the algebra $\{\mathbb{N}, +, \times, R\}$. The requirement of the cardinality function that belongs to the algebra $\{\mathbb{N}, +, \times, R\}$ is the main restriction of Kruchinin's method. If the required form of the cardinality function is unknown for a given combinatorial set, then this method cannot be applied to develop combinatorial generation algorithms.

The organization of this paper is as follows. Section 2 of this paper is devoted to a brief description of the main theoretical points of the used method for developing combinatorial generation algorithms. In Section 3, our modification of the original method is presented. The modification is based on applying the method of compositae from the theory of generating functions. To confirm the effectiveness of using the proposed modification of the original method, we develop new ranking and unranking algorithms for the following combinatorial sets: permutations, permutations with ascents, combinations, Dyck paths with return steps, labeled Dyck paths with ascents on return steps. The obtained results are shown in Section 4.

2. Method for Developing Combinatorial Generation Algorithms Based on AND/OR Trees

Kruchinin [24] introduces a method for developing combinatorial generation algorithms, which is based on the use of AND/OR trees. This method is based on representing a combinatorial set in the form of an AND/OR tree structure for which the total number of its variants is equal to the value of the cardinality function of the combinatorial set. Using an AND/OR tree structure, it is possible to develop listing, ranking, and unranking algorithms for a given combinatorial set. The effectiveness of this method is shown in the development of combinatorial generation algorithms for a large number of combinatorial sets (for example, permutations, combinations, partitions, compositions, the Fibonacci numbers, the Catalan numbers, the Stirling numbers, tree structures, and formal languages).

An AND/OR tree is a tree structure that contains nodes of two types: AND nodes and OR nodes. Figure 1 shows a way for representing nodes in an AND/OR tree.

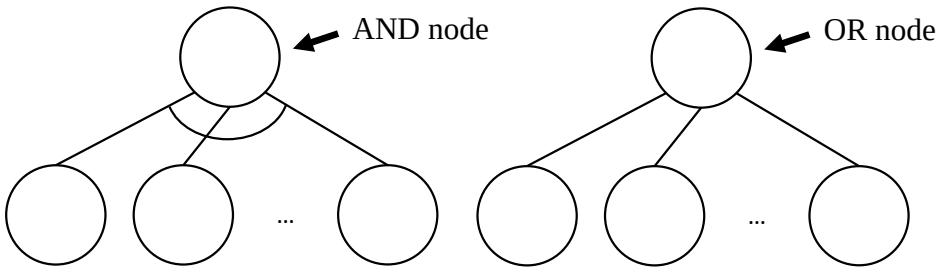

Figure 1. The representation of nodes in an AND/OR tree.

A variant of an AND/OR tree is a tree structure obtained by removing all edges except one for each OR node. Figure 2 shows an example of an AND/OR tree and all its variants.

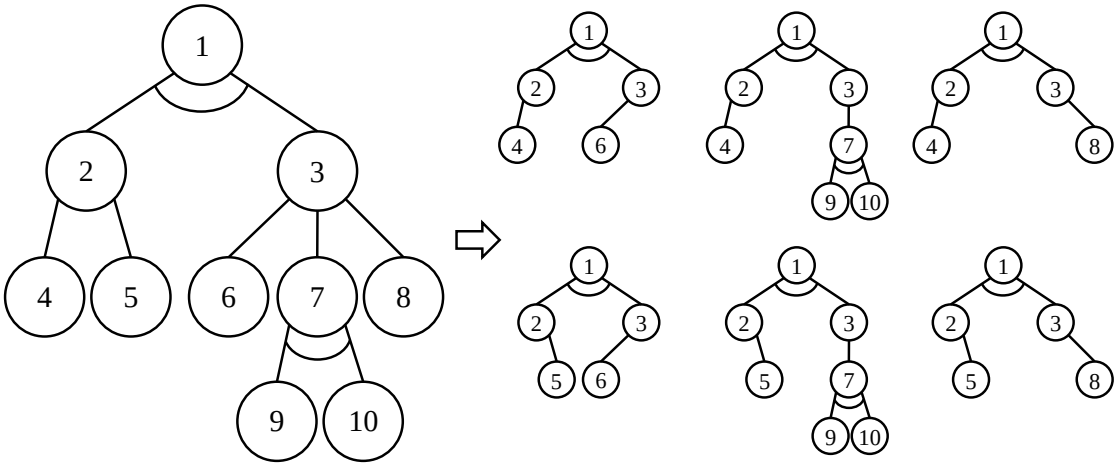

Figure 2. An AND/OR tree and all its variants.

If we know the cardinality function f of a combinatorial set A that belongs to the algebra $\{\mathbb{N}, +, \times, R\}$, then we can construct an AND/OR tree structure for which the total number of its variants is equal to the value of the cardinality function (Theorems 1–3 and Corollaries 1–2 in [24]. To do this, it is necessary to perform the following steps for the cardinality function f:

- each addition $+$ from f must be represented as an OR node of the AND/OR tree where all terms are represented as sons of the OR node;
- each multiplication $\times$ from f must be represented as an AND node of the AND/OR tree where all factors are represented as sons of the AND node;
- each coefficient $k \in \mathbb{N}$ from f must be represented as an OR node of the AND/OR where all sons are leaves and their number is equal to k;
- each recursive operation from f must be represented as recursion in the AND/OR tree (it will be denoted by a node with a triangle).

Thus, the method for developing combinatorial generation algorithms based on AND/OR trees can be written in the following form:

Input: The cardinality function f of a combinatorial set A that belongs to the algebra $\{\mathbb{N}, +, \times, R\}$.

Output: The combinatorial generation algorithms

$$\texttt{RankVariant}(v) : W(D) \to \mathbb{N}_{|W(D)|}$$

and

$$\texttt{UnrankVariant}(r) : \mathbb{N}_{|W(D)|} \to W(D),$$

where each variant v of an AND/OR tree D constructed for the combinatorial set A must correspond to a specific combinatorial object $a \in A$. That is, the bijection $A \leftrightarrow W(D)$ must be defined, where $W(D)$ is the set of all the variants v of the AND/OR tree D. The combination of this bijection with the algorithms $\texttt{RankVariant}(v)$ and $\texttt{UnrankVariant}(r)$ represents the desired combinatorial generation algorithms $\texttt{Rank}(a) : A \to \mathbb{N}$ and $\texttt{Unrank}(r) : \mathbb{N} \to A$ (for ranking and unranking elements of the combinatorial set A).

The ranking algorithm $\texttt{RankVariant}(v)$ allows us to associate each variant $v \in W(D)$ of the AND/OR tree D with a unique number $r \in \mathbb{N}_{|W(D)|} = \{0, 1, \ldots, |W(D)| - 1\}$ called the rank. The rank r of a combinatorial object a represented as a variant v of the corresponding AND/OR tree D is determined by the value of $l(z)$ for the node z of the variant v that is the root of the AND/OR tree D.

The value of $l(z)$ corresponds to a number for the node z that satisfies the condition

$$0 \leq l(z) < w(z),$$

where $w(z)$ shows the number of variants in the subtree of the node z.

The value of $l(z)$ is calculated according to the following rules:

1. If a node z of the variant v is a leaf of the AND/OR tree D, then

$$l(z) = 0;$$

2. If a node z of the variant v is an AND node of the AND/OR tree D, then

$$l(z) = l(s_1^{(z)}) + w(s_1^{(z)}) \left(l(s_2^{(z)}) + w(s_2^{(z)}) \left(\ldots \left(l(s_{n-1}^{(z)}) + w(s_{n-1}^{(z)}) l(s_n^{(z)}) \right) \ldots \right) \right),$$

where n is equal to the number of sons for the node z and $s_i^{(z)}$ is the i-th son of the node z;

3. If a node z of the variant v is an OR node of the AND/OR tree D, then

$$l(z) = l(s_k^{(z)}) + \sum_{i=1}^{k-1} w(s_i^{(z)}),$$

where k is the position of the son $s_k^{(z)}$ of the node z chosen in the variant v among all its sons in the AND/OR tree D.

For the general case, the rules for ranking the variants of an AND/OR tree was formalized and presented as Algorithm 1. To run this algorithm, it is necessary to start from the command `RankVariant`$(root, v, D)$, where *root* is the root of an AND/OR tree D.

Algorithm 1: A general algorithm for ranking the variants of an AND/OR tree.

```
1  RankVariant (z, v, D)
2  begin
3      if z = a leaf of the AND/OR tree D then l := 0
4      if z = an AND node of the AND/OR tree D then
5          n := the number of sons for the node z
6          l := RankVariant (s_n^(z), v, D)
7          for i := n − 1 to 1 do l := RankVariant (s_i^(z), v, D) + w(s_i^(z)) · l
8      end
9      if z = an OR node of the AND/OR tree D then
10         k := the position of the son of the node z chosen in the variant v among all its sons
11         l := RankVariant (s_k^(z), v, D)
12         for i := 1 to k − 1 do l := l + w(s_i^(z))
13     end
14     return l
15 end
```

The unranking algorithm `UnrankVariant`(r) performs the inverse operation to the algorithm `RankVariant`(v). That is, the algorithm `UnrankVariant`(r) allows us to associate each rank $r \in \mathbb{N}_{|W(D)|}$ with a unique variant v of the AND/OR tree D. The algorithm `UnrankVariant`(r) is based on the inverse actions to the calculations used in the algorithm `RankVariant`(v). For the general case, the rules for unranking the variants of an AND/OR tree were formalized and presented as Algorithm 2.

Algorithm 2: A general algorithm for unranking the variants of an AND/OR tree.

```
1  UnrankVariant (r, D)
2  begin
3      Push (r, root) onto the stack
4      Add root to the variant v
5      while the stack is not empty do
6          Pop (l, z) from the stack
7          if z = an AND node of the AND/OR tree D then
8              n := the number of sons for the node z
9              for i := 1 to n do
10                 l_i := l  mod w(s_i^(z))
11                 Push (l_i, s_i^(z)) onto the stack
12                 Add s_i^(z) to the variant v
13                 l := ⌊ l / w(s_i^(z)) ⌋
14             end
15         end
16         if z = an OR node of the AND/OR tree D then
17             k := 1
18             sum := 0
19             while sum + w(s_k^(z)) ≤ l do
20                 sum := sum + w(s_i^(z))
21                 k := k + 1
22             end
23             l := l − sum
24             Push (l, s_k^(z)) onto the stack
25             Add s_k^(z) to the variant v
26         end
27     end
28     return v
29 end
```

3. Modification of the Method for Developing Combinatorial Generation Algorithms

The use of the method for developing combinatorial generation algorithms based on AND/OR trees has the following two restrictions:

Firstly, if we do not know the cardinality function of a combinatorial set that belongs to the algebra $\{\mathbb{N}, +, \times, R\}$, then we cannot construct the corresponding AND/OR tree for the combinatorial set. Therefore, this method for developing combinatorial generation algorithms cannot be applied without an AND/OR tree structure.

If an AND/OR tree structure is constructed for a combinatorial set, then there is a new problem. This problem is associated with finding a bijection between the elements of the combinatorial set and the set of variants of the AND/OR tree (that is, each variant v of an AND/OR tree D constructed for the combinatorial set A must correspond to a specific combinatorial object $a \in A$, and vice versa). A general approach for solving this problem does not exist, since each combinatorial set has its own and completely unique characteristics. We propose to use the following recommendation: it is necessary to consider the changes that occur in the structure of combinatorial objects when moving from one node

of an AND/OR tree to another (considering the type of a node, the label of a node and the selected sons) and show these changes in the bijection.

For solving the first problem, we propose to apply the theory of generating functions, since it is one of the basic approaches in modern combinatorics and generating functions are already known for many combinatorial sets. An ordinary generating function of a sequence $(a_n)_{n\geq 0}$ is the following formal power series [29]:

$$A(t) = a_0 + a_1 t + a_2 t^2 + \ldots = \sum_{n\geq 0} a_n t^n.$$

For the coefficients of the powers of generating functions, the notion of the compositae of a generating function was introduced in [30]. The composita of an ordinary generating function

$$G(t) = \sum_{n>0} g_n t^n$$

is the following function with two variables $G^{\Delta}(n,k)$, which is a coefficients function of the k-th power of the generating function $G(t)$:

$$(G(t))^k = \sum_{n\geq k} G^{\Delta}(n,k) t^n.$$

This mathematical apparatus provides such operations on compositae as shift, addition, multiplication, composition, reciprocation, and compositional inversion of generating functions. Such operations on compositae allow us to obtain explicit expressions for the coefficients of generating functions. To obtain an explicit expression for the coefficients of a generating function using the method of compositae, it is necessary to decompose the given generating function into functions for that the compositae are known and apply the corresponding operations to them. More detailed information about the compositae can be found in [30–35].

If for a given combinatorial set A we consider its subset $A_n \subset A$, which contains only the combinatorial objects of size n, then the cardinality function $f(n) = |A_n|$ of this combinatorial set A_n can be described by a generating function

$$F(t) = \sum_{n\geq 0} f_n t^n = \sum_{n\geq 0} f(n) t^n = \sum_{n\geq 0} |A_n| t^n.$$

Hence, to obtain an expression for the cardinality function $f(n)$ of a combinatorial set for which a generating function is known, we can apply the method of compositae for obtaining an explicit expression of the coefficients f_n of the generating function [25]. Moreover, the operations on compositae can be extended to the case of multivariate generating functions. This makes it possible to obtain expressions for the cardinality functions of combinatorial sets that are described by more than one parameter.

The obtained method for developing combinatorial generation algorithms with its modification is presented as a sequence of the following steps:

1. If the cardinality function f of a combinatorial set A that belongs to the algebra $\{\mathbb{N}, +, \times, R\}$ is known, then go to Step 4.
2. If the generating function F for the values of the cardinality function f of the combinatorial set A is known, then apply the method of compositae for obtaining an explicit expression of the coefficients of the generating function. Otherwise, the method for developing combinatorial generation algorithms cannot be applied.
3. If the cardinality function f of the combinatorial set A that belongs to the algebra $\{\mathbb{N}, +, \times, R\}$ is obtained, then go to Step 4. Otherwise, the method for developing combinatorial generation algorithms cannot be applied.
4. Using the cardinality function f of the combinatorial set A, construct the corresponding AND/OR tree D.

5. Find a bijection $A \leftrightarrow W(D)$ between the elements of the combinatorial set A and the set of variants v of the AND/OR tree D in the form of the algorithms $\texttt{ObjectToVariant}(a, D) : A \to W(D)$, where $a \in A$, and $\texttt{VariantToObject}(v, D) : W(D) \to A$, where $v \in W(D)$.
6. Find a bijection $W(D) \leftrightarrow \mathbb{N}_{|W(D)|}$ between the elements of the set of variants v of the AND/OR tree D and the finite set of natural numbers $\mathbb{N}_{|W(D)|} = \{0, 1, \ldots, |W(D)| - 1\}$, using the algorithms $\texttt{RankVariant}(root, v, D) : W(D) \to \mathbb{N}_{|W(D)|}$, where *root* is the root of an AND/OR tree D, $v \in W(D)$, and $\texttt{UnrankVariant}(r, D) : \mathbb{N}_{|W(D)|} \to W(D)$, where $r \in \mathbb{N}_{|W(D)|}$.

The combination of the algorithms defined in the last two steps of the modified method forms a bijection $A \leftrightarrow \mathbb{N}$ and represents the combinatorial generation algorithms $\texttt{Rank}(a) : A \to \mathbb{N}$ for ranking and $\texttt{Unrank}(r) : \mathbb{N} \to A$ for unranking elements of the combinatorial set A.

4. Application of the Modification of the Method for Developing Combinatorial Generation Algorithms

Next, we consider the process of developing ranking and unranking algorithms using the obtained method for developing combinatorial generation algorithms. We describe a combinatorial set, construct an AND/OR tree, find a bijection between the elements of the combinatorial set and the set of variants of the AND/OR tree, and develop algorithms for ranking and unranking the variants.

4.1. Combinatorial Set

Let us consider the following combinatorial object: a labeled Dyck n-path with m ascents on return steps. A Dyck n-path is a lattice path in the plane which begins at $(0, 0)$, ends at $(2n, 0)$, and consists of steps $(1, 1)$ called rises or up-steps and $(1, -1)$ called falls or down-steps [36]. A return step is a down-step at level 1 (a return to the ground level 0) [37]. In a labeled Dyck n-path with m ascents on return steps, each down-step has its own label (a unique value from 1 to n). If we consider the sequence of labels of down-steps of a Dyck n-path starting from $(0, 0)$, then it has exactly m ascents.

Figure 3 shows all possible variants of the considered labeled Dyck paths for $n = 3$ and $m = 1$.

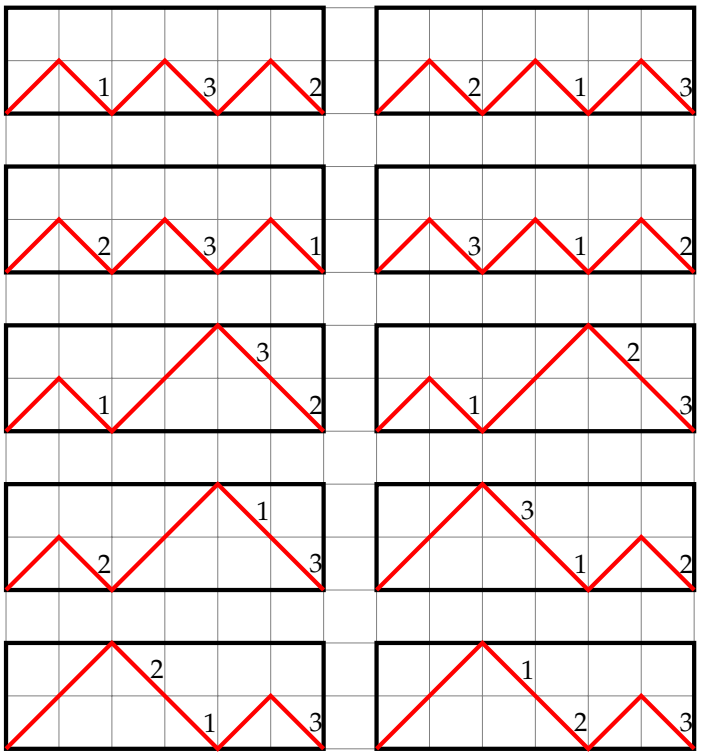

Figure 3. All labeled Dyck paths of size 3 with 1 ascent on return steps.

The total number of labeled Dyck n-paths with m ascents on return steps is defined by the elements of the Euler–Catalan number triangle (the sequence $A316773$ in [38]). That is, the cardinality function of this combinatorial set is equal to the Euler–Catalan numbers, which are denoted by EC_n^m [39].

Applying the method of compositae for obtaining explicit expressions for the coefficients of generating functions, we have found a generating function and the following explicit formula [39]:

$$EC_n^m = \begin{cases} 1, & \text{for } n = m = 0; \\ \sum\limits_{k=m+1}^{n} \frac{n!}{k!} CT_n^k E_k^m, & \text{otherwise,} \end{cases} = \begin{cases} 1, & \text{for } n = m = 0; \\ \sum\limits_{k=m+1}^{n} CT_n^k C_n^k E_k^m P_{n-k}, & \text{otherwise,} \end{cases} \quad (1)$$

where CT_n^m is the transposed Catalan triangle, C_n^m is the number of m-combinations of n elements, E_n^m is the Euler triangle, and P_n is the number of permutations of n elements.

4.2. AND/OR Tree

Equation (1) belongs to the algebra $\{\mathbb{N}, +, \times, R\}$, and there are the well-known formulas for the components of Equation (1), which also belong to the required algebra:

- the elements of the transposed Catalan triangle (the sequence $A033184$ in [38]) show the number of Dyck n-paths with m return steps and can be calculated using the following recurrence [37]:

$$CT_n^m = CT_{n-1}^{m-1} + CT_n^{m+1}, \quad CT_n^0 = 0, \quad CT_n^n = 1; \quad (2)$$

- the number of m-combinations of n elements (the sequence $A007318$ in [38]) can be calculated using the following recurrence [40]:

$$C_n^m = C_{n-1}^m + C_{n-1}^{m-1}, \quad C_n^n = C_n^0 = 1; \quad (3)$$

- the elements of the Euler triangle (the sequence $A173018$ in [38]) show the number of permutations of n elements with m ascents and can be calculated using the following recurrence [41]:

$$E_n^m = (m+1)E_{n-1}^m + (n-m)E_{n-1}^{m-1}, \quad E_n^{n-1} = E_n^0 = 1; \quad (4)$$

- the number of permutations of n elements (the sequence $A000142$ in [38]) can be calculated using the following recurrence [40]:

$$P_n = nP_{n-1}, \quad P_0 = 1. \quad (5)$$

Since Equation (1) and all the formulas for its components belong to the algebra $\{\mathbb{N}, +, \times, R\}$, we can construct the AND/OR tree structure for EC_n^m, which is presented in Figure 4. A bijection between the labeled Dyck n-paths with m ascents on return steps and the variants of the AND/OR tree is defined by the following rules:

- the number of return steps in a Dyck path is determined by the value of k (the selected son of the OR node labeled EC_n^m in a variant of the AND/OR tree);
- the subtree of the node labeled CT_n^k determines the version of a Dyck n-path with k return steps;
- the subtree of the node labeled C_n^k determines k values given from the set of n values, which are used as the labels for the return steps (the remaining $n-k$ values are used as the labels for the remaining $n-k$ down-steps);
- the subtree of the node labeled E_k^m determines the version of a permutation of the labels for k return steps, which form exactly m ascents;
- the subtree of the node labeled P_{n-k} determines the version of a permutation of the labels for the remaining $n-k$ down-steps.

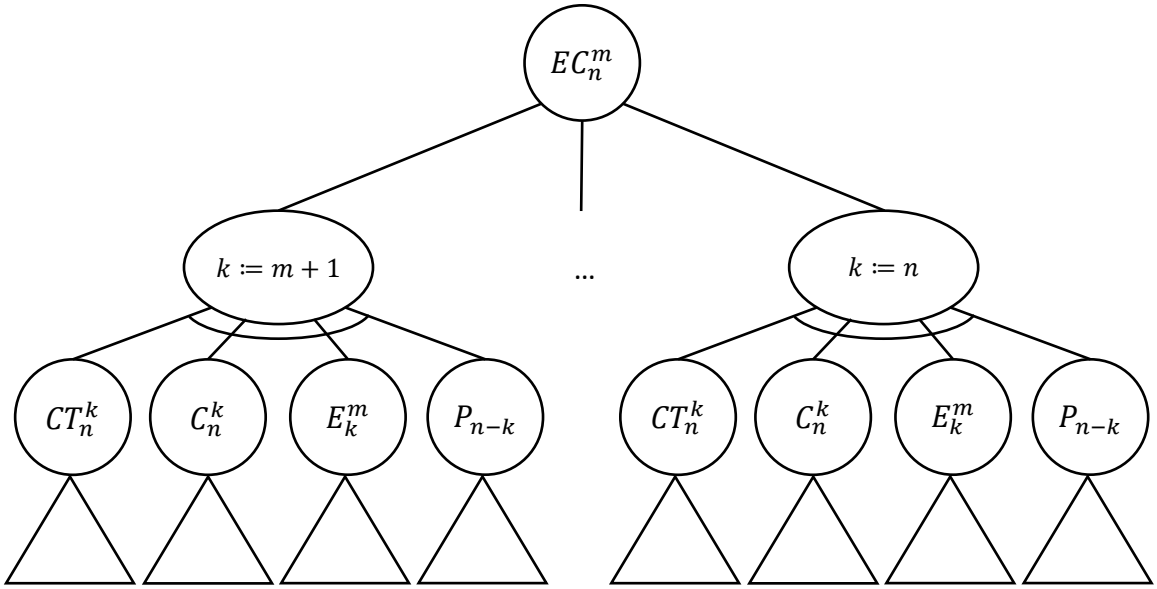

Figure 4. An AND/OR tree for EC_n^m.

Next, we need to construct AND/OR tree structures for all the subtrees of the AND/OR tree for EC_n^m. We also need to find bijections between the variants of these AND/OR trees and the corresponding combinatorial objects.

Since Equation (2) belongs to the algebra $\{\mathbb{N}, +, \times, R\}$, we can construct the AND/OR tree structure for CT_n^m, which is presented in Figure 5. A bijection between the Dyck n-paths with m return steps and the variants of the AND/OR tree is defined by the following rules:

- if a Dyck n-path is not completely filled and it is at the ground level 0, then it is necessary to add an up-step;
- each selected left son of the AND/OR tree (the node labeled CT_{n-1}^{m-1}) corresponds to a down-step in a Dyck n-path;
- each selected right son of the AND/OR tree (the node labeled CT_n^{m+1}) corresponds to an up-step in a Dyck n-path;
- if a leaf of the AND/OR tree is reached (when $m = n$), then it is necessary to add n down-steps.

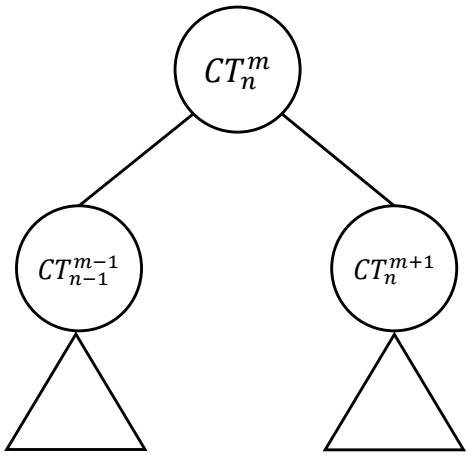

Figure 5. An AND/OR tree for CT_n^m.

For a compact representation, we encode a Dyck n-path by a sequence $a = (a_1, \ldots, a_{2n})$, where $a_i =$'u' encodes an up-step and $a_i =$'d' encodes a down-step. We also encode a variant of an AND/OR tree by a sequence $v = (v_1, v_2, \ldots)$ of the selected sons of the OR nodes in this tree (the left son corresponds to $v_i = 0$ and the right son corresponds to $v_i = 1$). An example of applying the obtained bijection for Dyck paths with return steps is presented in Table A1.

Since Equation (3) belongs to the algebra $\{\mathbb{N}, +, \times, R\}$, we can construct the AND/OR tree structure for C_n^m, which is presented in Figure 6. A bijection between the m-combinations of n elements and the variants of the AND/OR tree is defined by the following rules:

- each selected left son of the AND/OR tree (the node labeled C_{n-1}^m) determines that the element n is not selected in an m-combination of n elements;
- each selected right son of the AND/OR tree (the node labeled C_{n-1}^{m-1}) determines that the element n is selected in an m-combination of n elements;
- if a leaf of the AND/OR tree is reached (when $m = 0$), then it is necessary to not select all n elements in an m-combination of n elements;
- if a leaf of the AND/OR tree is reached (when $m = n$), then it is necessary to select all n elements in an m-combination of n elements.

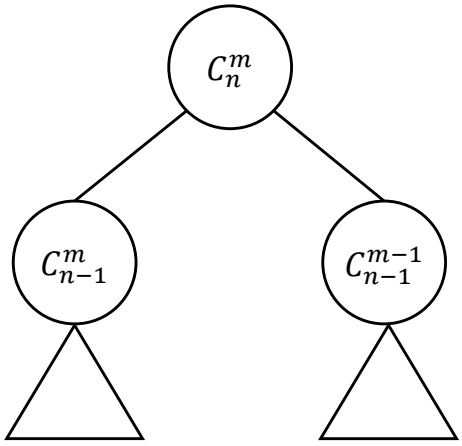

Figure 6. An AND/OR tree for C_n^m.

For a compact representation, we encode an m-combination of n elements by a sequence $a = (a_1, \ldots, a_n)$, where $a_i = 0$ encodes that the i-th element is not selected and $a_i = 1$ encodes that the i-th element is selected. We also encode a variant of an AND/OR tree by a sequence $v = (v_1, v_2, \ldots)$ of the selected sons of the OR nodes in this tree (the left son corresponds to $v_i = 0$ and the right son corresponds to $v_i = 1$). An example of applying the obtained bijection for combinations is presented in Table A2.

Since Equation (4) belongs to the algebra $\{\mathbb{N}, +, \times, R\}$, we can construct the AND/OR tree structure for E_n^m, which is presented in Figure 7. A bijection between the permutations of n elements with m ascents and the variants of the AND/OR tree is defined by the following rules:

- each selected left son of the OR node labeled E_n^m determines that the element n does not add an ascent in a permutation of n elements, and the selected son of the OR node labeled $m + 1$ determines the position of the element n in the permutation (there are exactly $m + 1$ possible positions);
- each selected right son of the OR node labeled E_n^m determines that the element n adds an ascent in a permutation of n elements, and the selected son of the OR node labeled $n - m$ determines the position of the element n in the permutation (there are exactly $n - m$ possible positions);

- if a leaf of the AND/OR tree is reached (when $m = 0$), then it is necessary to arrange all n elements in a permutation of n elements in decreasing order;
- if a leaf of the AND/OR tree is reached (when $m = n - 1$), then it is necessary to arrange all n elements in a permutation of n elements in increasing order.

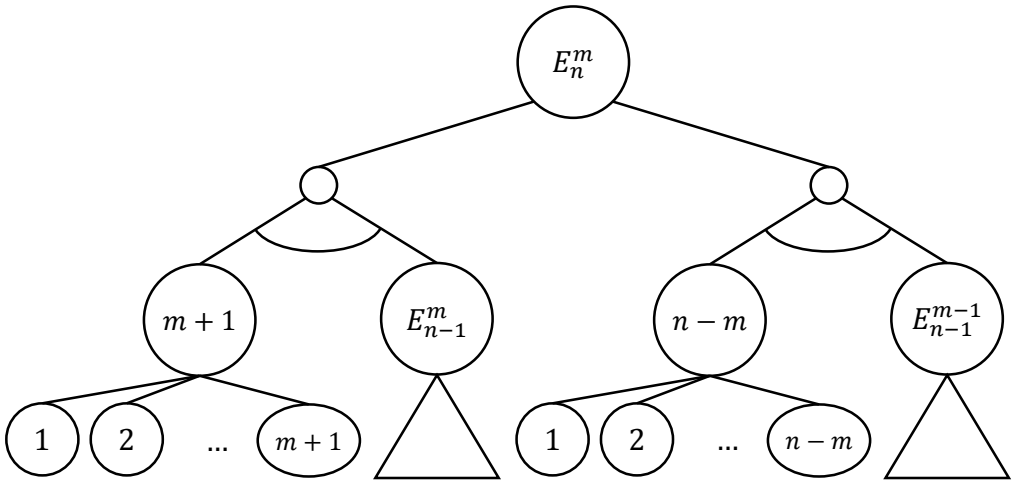

Figure 7. An AND/OR tree for E_n^m.

For a compact representation, we encode a variant of an AND/OR tree by a sequence $v = (v_1, v_2, \ldots)$ of the selected sons of the OR nodes in this tree, where each v_i is represented as a pair $(v_{i,1}, v_{i,2})$. In this pair: $v_{i,1} = 0$ corresponds to the left son of the OR node labeled E_n^m and $v_{i,2}$ determines the selected son of the OR node labeled $m + 1$; $v_{i,1} = 1$ corresponds to the right son of the OR node labeled E_n^m, $v_{i,2}$ determines the selected son of the OR node labeled $n - m$. An example of applying the obtained bijection for permutations with ascents is presented in Table A3.

Since Equation (5) belongs to the algebra $\{\mathbb{N}, +, \times, R\}$, we can construct the AND/OR tree structure for P_n, which is presented in Figure 8. A bijection between the permutations of n elements and the variants of the AND/OR tree is defined by the following rules:

- each selected son of the OR node labeled n determines the position of the element n in a permutation of n elements;
- if a leaf of the AND/OR tree is reached (when $n = 0$), then it is necessary to form an empty permutation.

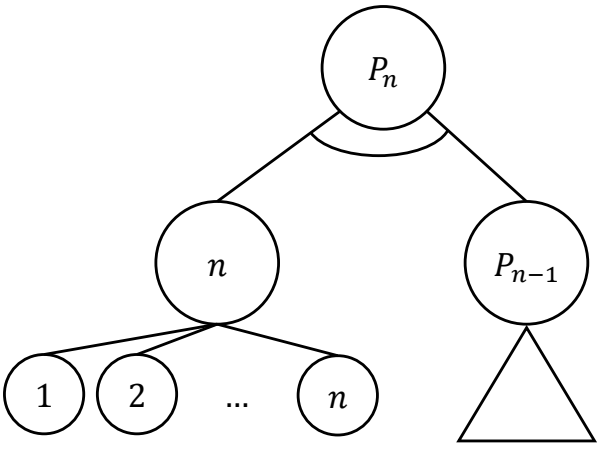

Figure 8. An AND/OR tree for P_n.

For a compact representation, we encode a variant of an AND/OR tree by a sequence $v = (v_1, v_2, \ldots)$ of the selected sons of the OR nodes in this tree. An example of applying the obtained bijection for permutations is presented in Table A4.

We have constructed AND/OR trees for all the combinatorial sets presented in this paper. Hence, we can develop algorithms for ranking and unranking the variants of the AND/OR trees.

4.3. Ranking and Unranking Algorithms

Based on Algorithms 1 and 2, we can develop algorithms for ranking and unranking the variants of the constructed AND/OR trees for EC_n^m, CT_n^m, C_n^m, E_n^m, and P_n.

For the AND/OR tree for CT_n^m, which is presented in Figure 5, we develop an algorithm for ranking its variants (Algorithm 3) and an algorithm for unranking its variants (Algorithm 4). Combining the developed algorithms with the derived rules for the bijection, we get algorithms for ranking and unranking the combinatorial set of Dyck n-paths with m return steps.

Algorithm 3: An algorithm for ranking the variants of the AND/OR tree for CT_n^m.

```
1 RankVariant_CT (v = (v_1, v_2, ...), n, m)
2 begin
3     if m = n then r = 0
4     else
5         if v_1 = 0 then r := RankVariant_CT ((v_2, ...), n − 1, m − 1)
6         else r := RankVariant_CT ((v_2, ...), n, m + 1) + CT_{n−1}^{m−1}
7     end
8     return r
9 end
```

Algorithm 4: An algorithm for unranking the variants of the AND/OR tree for CT_n^m.

```
1 UnrankVariant_CT (r, n, m)
2 begin
3     if m = n then v := ()
4     else
5         if r < CT_{n−1}^{m−1} then v :=concat ((0), UnrankVariant_CT (r, n − 1, m − 1))
6         else v :=concat ((1), UnrankVariant_CT (r − CT_{n−1}^{m−1}, n, m + 1))
7     end
8     return v
9 end
```

In these algorithms, () denotes an empty sequence and a function `concat` denotes merging sequences. That is, if we have a sequence $a = (a_1, \ldots, a_n)$ and a sequence $b = (b_1, \ldots, b_m)$, then we can get the following sequence:

$$\texttt{concat}(a, b) = (a_1, \ldots, a_n, b_1, \ldots, b_m).$$

For the AND/OR tree for C_n^m, which is presented in Figure 6, we develop an algorithm for ranking its variants (Algorithm 5) and an algorithm for unranking its variants (Algorithm 6). Combining the developed algorithms with the derived rules for the bijection, we get algorithms for ranking and unranking the combinatorial set of m-combinations of n elements.

Algorithm 5: An algorithm for ranking the variants of the AND/OR tree for C_n^m.

```
1 RankVariant_C (v = (v_1, v_2, ...), n, m)
2 begin
3     if m = 0 or m = n then r = 0
4     else
5         if v_1 = 0 then r := RankVariant_C ((v_2, ...), n − 1, m)
6         else r := RankVariant_C ((v_2, ...), n − 1, m − 1) + C_{n−1}^m
7     end
8     return r
9 end
```

Algorithm 6: An algorithm for unranking the variants of the AND/OR tree for C_n^m.

```
1 UnrankVariant_C (r, n, m)
2 begin
3     if m = 0 or m = n then v := ()
4     else
5         if r < C_{n−1}^m then v :=concat ((0), UnrankVariant_C (r, n − 1, m))
6         else v :=concat ((1), UnrankVariant_C (r − C_{n−1}^m, n − 1, m − 1))
7     end
8     return v
9 end
```

For the AND/OR tree for E_n^m, which is presented in Figure 7, we develop an algorithm for ranking its variants (Algorithm 7) and an algorithm for unranking its variants (Algorithm 8). Combining the developed algorithms with the derived rules for the bijection, we get algorithms for ranking and unranking the combinatorial set of permutations of n elements with m ascents.

Algorithm 7: An algorithm for ranking the variants of the AND/OR tree for E_n^m.

```
1  RankVariant_E (v = ((v_{1,1}, v_{1,2}), (v_{2,1}, v_{2,2}), ...), n, m)
2  begin
3      if m = 0 or m = n − 1 then r = 0
4      else
5          if v_{1,1} = 0 then
6              l_1 := v_{1,2} − 1
7              l_2 := RankVariant_E (((v_{2,1}, v_{2,2}), ...), n − 1, m)
8              r := l_1 + (m + 1)l_2
9          end
10         else
11             l_1 := v_{1,2} − 1
12             l_2 := RankVariant_E (((v_{2,1}, v_{2,2}), ...), n − 1, m − 1)
13             r := l_1 + (n − m)l_2 + (m + 1)E_{n−1}^m
14         end
15     end
16     return r
17 end
```

Algorithm 8: An algorithm for unranking the variants of the AND/OR tree for E_n^m.

```
1  UnrankVariant_E (r, n, m)
2  begin
3      if m = 0 or m = n then v := ()
4      else
5          if r < (m + 1)E^m_{n−1} then
6              l_1 := r mod m + 1
7              l_2 := ⌊r/(m+1)⌋
8              v :=concat (((0, l_1 + 1)), UnrankVariant_E (l_2, n − 1, m))
9          end
10         else
11             r := r − (m + 1)E^m_{n−1}
12             l_1 := r mod n − m
13             l_2 := ⌊r/(n−m)⌋
14             v :=concat (((1, l_1 + 1)), UnrankVariant_E (l_2, n − 1, m − 1))
15         end
16     end
17     return v
18 end
```

For the AND/OR tree for P_n, which is presented in Figure 8, we develop an algorithm for ranking its variants (Algorithm 9) and an algorithm for unranking its variants (Algorithm 10). Combining the developed algorithms with the derived rules for the bijection, we get algorithms for ranking and unranking the combinatorial set of permutations of n elements.

Algorithm 9: An algorithm for ranking the variants of the AND/OR tree for P_n.

```
1  RankVariant_P (v = (v_1, v_2, ...), n)
2  begin
3      if n = 0 then r = 0
4      else
5          l_1 := v_1 − 1
6          l_2 := RankVariant_P ((v_2, ...), n − 1)
7          r := l_1 + nl_2
8      end
9      return r
10 end
```

Algorithm 10: An algorithm for unranking the variants of the AND/OR tree for P_n.

```
1  UnrankVariant_P (r, n)
2  begin
3      if n = 0 then v := ()
4      else
5          l_1 := r mod n
6          l_2 := ⌊r/n⌋
7          v :=concat ((l_1 + 1), UnrankVariant_P (l_2, n − 1))
8      end
9      return v
10 end
```

For the AND/OR tree for EC_n^m, which is presented in Figure 4, we develop an algorithm for ranking its variants (Algorithm 11) and an algorithm for unranking its variants (Algorithm 12). Combining the developed algorithms with the derived rules for the bijection, we get algorithms for ranking and unranking the combinatorial set of labeled Dyck n-paths with m ascents on return steps.

Algorithm 11: An algorithm for ranking the variants of the AND/OR tree for EC_n^m.

```
RankVariant_EC (v = (k, v1, v2, v3, v4), n, m)
begin
    l1 := RankVariant_CT (v1, n, k)
    l2 := RankVariant_C (v2, n, k)
    l3 := RankVariant_E (v3, k, m)
    l4 := RankVariant_P (v4, n − k)
    r := l1 + CT_n^k (l2 + C_n^k (l3 + E_k^m l4)) + Σ_{i=m+1}^{k−1} CT_n^i C_n^i E_i^m P_{n−i}
    return r
end
```

Algorithm 12: An algorithm for unranking the variants of the AND/OR tree for EC_n^m.

```
UnrankVariant_EC (r, n, m)
begin
    k := m + 1
    sum := 0
    while sum + CT_n^k C_n^k E_k^m P_{n−k} ≤ r do
        sum := sum + CT_n^k C_n^k E_k^m P_{n−k}
        k := k + 1
    end
    r := r − sum
    l1 := r mod CT_n^k
    r := ⌊ r / CT_n^k ⌋
    l2 := r mod C_n^k
    r := ⌊ r / C_n^k ⌋
    l3 := r mod E_k^m
    l4 := ⌊ r / E_k^m ⌋
    v1 := UnrankVariant_CT (l1, n, k)
    v2 := UnrankVariant_C (l2, n, k)
    v3 := UnrankVariant_E (l3, k, m)
    v4 := UnrankVariant_P (l4, n − k)
    v = (k, v1, v2, v3, v4)
    return v
end
```

In these algorithms, we use all the above mentioned algorithms for ranking and unranking the variants of the AND/OR trees for CT_n^m, C_n^m, E_n^m, and P_n. For a compact representation, a variant of the AND/OR tree for EC_n^m is encoded by a sequence $v = (k, v_1, v_2, v_3, v_4)$, where:

- k is the label of the selected son of the OR node labeled EC_n^m in a variant of the AND/OR tree;
- v_1 corresponds to the variant of the subtree of the node labeled CT_n^k;
- v_2 corresponds to the variant of the subtree of the node labeled C_n^k;

- v_3 corresponds to the variant of the subtree of the node labeled E_k^m;
- v_4 corresponds to the variant of the subtree of the node labeled P_{n-k}.

5. Conclusions

In this paper, we study methods for developing combinatorial generation algorithms and present basic general methods for solving this task. We consider one of these methods, which is based on AND/OR trees, and extend it by using the mathematical apparatus of the theory of generating functions.

Using an AND/OR tree structure, it is possible to develop listing, ranking, and unranking algorithms for a given combinatorial set. However, the use of the method for developing combinatorial generation algorithms based on AND/OR trees has the following restriction: the cardinality function of a combinatorial set must belong to the algebra $\{\mathbb{N}, +, \times, R\}$. For solving this problem, we propose to apply the method of compositae for obtaining explicit expression of the coefficients of generating functions, since the theory of generating functions is one of the basic approaches in combinatorics. The limitation of this method is that it can be applied only for a combinatorial set for which a generating function is known.

As a result, we formalize the proposed idea in our modification of the original method for developing combinatorial generation algorithms. In addition, one of the main contributions of the paper is the application of this method. To confirm the effectiveness of using the proposed method, we develop new ranking and unranking algorithms for the following combinatorial sets: labeled Dyck n-paths with m ascents on return steps, Dyck n-paths with m return steps, m-combinations of n elements, permutations of n elements with m ascents, permutations of n elements. For each of them, we construct an AND/OR tree, find a bijection between the elements of the combinatorial set and the set of variants of the AND/OR tree, and develop algorithms for ranking and unranking the variants of the AND/OR tree.

All the developed algorithms have been realized in the computer algebra system "Maxima" and validated by exhaustive generation for fixed values of combinatorial set parameters. It also has shown that all the developed algorithms have polynomial time complexity. Several examples of applying the obtained results can be found in Appendix A.

As further research, we will consider the development of new and effective combinatorial generation algorithms in the field of applied mathematics. For example, it can be done for combinatorial sets that represent different types of chemical compounds [42], molecular structures such as RNA and DNA [43–45], etc. We also plan further improvements to the presented method for developing combinatorial generation algorithms, for example, through the usage of other types of trees [46].

Author Contributions: Investigation, Y.S., V.K., and D.K.; methodology, V.K.; writing—original draft preparation, Y.S.; writing—review and editing, D.K. All authors have read and agreed to the published version of the manuscript.

Funding: The reported study was supported by the Russian Science Foundation (project no. 18-71-00059).

Acknowledgments: The authors would like to thank the referees for their helpful comments and suggestions.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Examples of Ranking the Elements of Combinatorial Sets

Table A1. Ranking the combinatorial set of Dyck n-paths with m return steps for $n = 5$ and $m = 2$.

Dyck path	Variant of AND/OR Tree	Rank
u d u u d u d u d d	$(0,1,0,1,0,1)$	0
u d u u d u u d d d	$(0,1,0,1,1)$	1
u d u u u d d u d d	$(0,1,1,0,0,1)$	2
u d u u u d u d d d	$(0,1,1,0,1)$	3

Table A1. *Cont.*

Dyck path	Variant of AND/OR Tree	Rank
u d u u u u d d d d	$(0,1,1,1)$	4
u u d d u u d u d d	$(1,0,0,1,0,1)$	5
u u d d u u u d d d	$(1,0,0,1,1)$	6
u u d u d d u u d d	$(1,0,1,0,0,1)$	7
u u d u d u d d u d	$(1,0,1,0,1)$	8
u u d u u d d d u d	$(1,0,1,1)$	9
u u u d d d u u d d	$(1,1,0,0,0,1)$	10
u u u d d u d d u d	$(1,1,0,0,1)$	11
u u u d u d d d u d	$(1,1,0,1)$	12
u u u u d d d d u d	$(1,1,1)$	13

Table A2. Ranking the combinatorial set of m-combinations of n elements for $n = 5$ and $m = 2$.

Combination	Variant of AND/OR Tree	Rank
$(1,1,0,0,0)$	$(0,0,0)$	0
$(1,0,1,0,0)$	$(0,0,1,0)$	1
$(0,1,1,0,0)$	$(0,0,1,1)$	2
$(1,0,0,1,0)$	$(0,1,0,0)$	3
$(0,1,0,1,0)$	$(0,1,0,1)$	4
$(0,0,1,1,0)$	$(0,1,1)$	5
$(1,0,0,0,1)$	$(1,0,0,0)$	6
$(0,1,0,0,1)$	$(1,0,0,1)$	7
$(0,0,1,0,1)$	$(1,0,1)$	8
$(0,0,0,1,1)$	$(1,1)$	9

Table A3. Ranking the combinatorial set of permutations of n elements with m ascents for $n = 4$ and $m = 2$.

Permutation	Variant of AND/OR Tree	Rank
$(4,1,2,3)$	$((0,1))$	0
$(1,4,2,3)$	$((0,2))$	1
$(1,2,4,3)$	$((0,3))$	2
$(3,4,1,2)$	$((1,1),(0,1))$	3
$(3,1,2,4)$	$((1,2),(0,1))$	4
$(1,3,4,2)$	$((1,1),(0,2))$	5
$(1,3,2,4)$	$((1,2),(0,2))$	6
$(2,3,4,1)$	$((1,1),(1,1))$	7
$(2,3,1,4)$	$((1,2),(1,1))$	8
$(2,4,1,3)$	$((1,1),(1,2))$	9
$(2,1,3,4)$	$((1,2),(1,2))$	10

Table A4. Ranking the combinatorial set of permutations of n elements for $n = 4$.

Permutation	Variant of AND/OR Tree	Rank
$(4,3,2,1)$	$(1,1,1,1)$	0
$(3,4,2,1)$	$(2,1,1,1)$	1
$(3,2,4,1)$	$(3,1,1,1)$	2
$(3,2,1,4)$	$(4,1,1,1)$	3
$(4,2,3,1)$	$(1,2,1,1)$	4
$(2,4,3,1)$	$(2,2,1,1)$	5
$(2,3,4,1)$	$(3,2,1,1)$	6
$(2,3,1,4)$	$(4,2,1,1)$	7
$(4,2,1,3)$	$(1,3,1,1)$	8
$(2,4,1,3)$	$(2,3,1,1)$	9
$(2,1,4,3)$	$(3,3,1,1)$	10

Table A4. *Cont.*

Permutation	Variant of AND/OR Tree	Rank
(2, 1, 3, 4)	(4, 3, 1, 1)	11
(4, 3, 1, 2)	(1, 1, 2, 1)	12
(3, 4, 1, 2)	(2, 1, 2, 1)	13
(3, 1, 4, 2)	(3, 1, 2, 1)	14
(3, 1, 2, 4)	(4, 1, 2, 1)	15
(4, 1, 3, 2)	(1, 2, 2, 1)	16
(1, 4, 3, 2)	(2, 2, 2, 1)	17
(1, 3, 4, 2)	(3, 2, 2, 1)	18
(1, 3, 2, 4)	(4, 2, 2, 1)	19
(4, 1, 2, 3)	(1, 3, 2, 1)	20
(1, 4, 2, 3)	(2, 3, 2, 1)	21
(1, 2, 4, 3)	(3, 3, 2, 1)	22
(1, 2, 3, 4)	(4, 3, 2, 1)	23

References

1. Knuth, D.E. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*; Addison-Wesley Professional: Boston, MA, USA, 2011.
2. Ruskey, F. Combinatorial generation. Working Version (1j-CSC 425/520). Available online: http://page.math.tu-berlin.de/~felsner/SemWS17-18/Ruskey-Comb-Gen.pdf (accessed on 1 May 2020).
3. Reingold, E.M.; Nievergelt, J.; Deo, N. *Combinatorial Algorithms: Theory and Practice*; Prentice-Hall: Upper Saddle River, NJ, USA, 1977.
4. Kreher, D.L.; Stinson, D.R. *Combinatorial Algorithms: Generation, Enumeration, and Search*; CRC Press: Boca Raton, FL, USA, 1999.
5. Barcucci, E.; Del Lungo, A.; Pergola, E.; Pinzani, R. ECO: A methodology for the enumeration of combinatorial objects. *J. Differ. Equ. Appl.* **1999**, *5*, 435–490. [CrossRef]
6. Barcucci, E.; Del Lungo, A.; Pergola, E. Random generation of trees and other combinatorial objects. *Theoret. Comput. Sci.* **1999**, *218*, 219–232. [CrossRef]
7. Bacchelli, S.; Barcucci, E.; Grazzini, E.; Pergola, E. Exhaustive generation of combinatorial objects by ECO. *Acta Inform.* **2004**, *40*, 585–602. [CrossRef]
8. Bacchelli, S.; Ferrari, L.; Pinzani, R.; Sprugnoli, R. Mixed succession rules: The commutative case. *J. Combin. Theory Ser. A* **2010**, *117*, 568–582.
9. Del Lungo, A.; Frosini, A.; Rinaldi, S. ECO method and the exhaustive generation of convex polyominoes. *Lect. Notes Comput. Sci. Discret. Math. Theor. Comput. Sci.* **2003**, *2731*, 129–140.
10. Del Lungo, A.; Duchi, E.; Frosini, A.; Rinaldi, S. On the generation and enumeration of some classes of convex polyominoes. *Electron. J. Combin.* **2004**, *11*, 1–46. [CrossRef]
11. Vajnovszki, V. Generating involutions, derangements, and relatives by ECO. *Discrete Math. Theor. Comput. Sci.* **2010**, *12*, 109–122.
12. Vajnovszki, V. An efficient Gray code algorithm for generating all permutations with a given major index. *J. Discret. Algorithms* **2014**, *26*, 77–88. [CrossRef]
13. Do, P.T.; Tran, T.T.H.; Vajnovszki, V. Exhaustive generation for permutations avoiding (colored) regular sets of patterns. *Discret. Appl. Math.* **2019**, *268*, 44–53. [CrossRef]
14. Flajolet, P.; Zimmerman, P.; Cutsem, B. A calculus for the random generation of combinatorial structures. *Theoret. Comput. Sci.* **1994**, *132*, 1–35. [CrossRef]
15. Sedgewick, R.; Flajolet, P. *An Introduction to the Analysis of Algorithms*, 2nd ed.; Addison-Wesley: Boston, MA, USA, 2013.
16. Martinez, C.; Molinero, X. A generic approach for the unranking of labeled combinatorial classes. *Random Struct. Algorithms* **2001**, *19*, 472–497. [CrossRef]
17. Martinez, C.; Molinero, X. Generic algorithms for the generation of combinatorial objects. *Lect. Notes Comput. Sci. Math. Found. Comput. Sci.* **2003**, *2747*, 572–581.
18. Martinez, C.; Molinero, X. An experimental study of unranking algorithms. *Lect. Notes Comput. Sci. Exp. Effic. Algorithms* **2004**, *3059*, 326–340.

19. Martinez, C.; Molinero, X. Efficient iteration in admissible combinatorial classes. *Theoret. Comput. Sci.* **2005**, *346*, 388–417. [CrossRef]
20. Molinero, X.; Vives, J. Unranking algorithms for combinatorial structures. *Int. J. Appl. Math. Inf.* **2015**, *9*, 110–115.
21. Ryabko, B.Y. Fast enumeration of combinatorial objects. *Discret. Math. Appl.* **1998**, *8*, 163–182. [CrossRef]
22. Medvedeva, Y.S.; Ryabko, B.Y. Fast enumeration algorithm for words with given constraints on run lengths of ones. *Probl. Inf. Transm.* **2010**, *46*, 390–399. [CrossRef]
23. Medvedeva, Y. Fast enumeration of words generated by Dyck grammars. *Math. Notes* **2014**, *96*, 68–83. [CrossRef]
24. Kruchinin, V.V. *Methods for Developing Algorithms for Ranking and Unranking Combinatorial Objects Based on AND/OR Trees*; V-Spektr: Tomsk, Russia, 2007. (In Russian)
25. Shablya, Y.; Kruchinin, D.; Kruchinin, V. Application of the method of compositae in combinatorial generation. In Proceedings of the Book of the 2nd Mediterranean International Conference of Pure and Applied Mathematics and Related Areas (MICOPAM2019), Paris, France, 28–31 August 2019; pp. 91–94.
26. Hartman, P.; Sawada, J. Ranking and unranking fixed-density necklaces and Lyndon words. *Theoret. Comput. Sci.* **2019**, *791*, 36–47. [CrossRef]
27. Pai, K.J.; Chang, J.M.; Wu, R.Y.; Chang, S.C. Amortized efficiency of generation, ranking and unranking left-child sequences in lexicographic order. *Discrete Appl. Math.* **2019**, *268*, 223–236. [CrossRef]
28. Amani, M.; Nowzari-Dalini, A. Efficient generation, ranking, and unranking of (k, m)-ary trees in B-order. *Bull. Iranian Math. Soc.* **2019**, *45*, 1145–1158. [CrossRef]
29. Stanley, R.P. *Enumerative Combinatorics*, 2nd ed.; Cambridge University Press: Cambridge, MA, USA, 2011; Volume 1.
30. Kruchinin, D.V.; Kruchinin, V.V. A method for obtaining generating functions for central coefficients of triangles. *J. Integer Seq.* **2012**, *15*, 12.9.3.
31. Kruchinin, D.V.; Kruchinin, V.V. Explicit formulas for some generalized polynomials. *Appl. Math. Inf. Sci.* **2013**, *7*, 2083–2088. [CrossRef]
32. Kruchinin, V.V.; Kruchinin, D.V. Composita and its properties. *J. Anal. Number Theory* **2014**, *2*, 37–44.
33. Kruchinin, D.V.; Shablya, Y.V. Explicit formulas for Meixner polynomials. *Int. J. Math. Math. Sci.* **2015**, *2015*, 620569. [CrossRef]
34. Kruchinin, D.V.; Shablya, Y.V.; Kruchinin, V.V.; Shelupanov, A.A. A method for obtaining coefficients of compositional inverse generating functions. In Proceedings of the International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2015), Rhodes, Greece, 22–28 September 2015; Volume 1738, p. 130003.
35. Kruchinin, D.V.; Kruchinin, V.V. Explicit formula for reciprocal generating function and its application. *Adv. Stud. Contemp. Math. Kyungshang* **2019**, *29*, 365–372.
36. Banderier, C.; Krattenthaler, C.; Krinik, A.; Kruchinin, D.; Kruchinin, V.; Nguyen, D.; Wallner, M. Explicit formulas for enumeration of lattice paths: Basketball and the kernel method. In *Lattice Path Combinatorics and Applications*; Springer: Cham, Switzerland, 2019; pp. 78–118.
37. Deutsch, E. Dyck path enumeration. *Discret. Math.* **1999**, *204*, 167–202. [CrossRef]
38. Sloane, N.J.A. The On-Line Encyclopedia of Integer Sequences. Available online: https://oeis.org (accessed on 1 May 2020).
39. Shablya, Y.; Kruchinin, D. Euler–Catalan's number triangle and its application. *Symmetry* **2020**, *12*, 600. [CrossRef]
40. Graham, R.L.; Knuth, D.E.; Patashnik, O. *Concrete Mathematics*, 2nd ed.; Addison-Wesley: Boston, MA, USA, 1994.
41. Petersen, T.K. *Eulerian Numbers*; Birkhauser Advanced Texts; Basler Lehrbucher, Birkhauser/Springer: New York, NY, USA, 2015.
42. Shimizu, T.; Fukunaga, T.; Nagamochi, H. Unranking of small combinations from large sets. *J. Discret. Algorithms* **2014**, *29*, 8–20. [CrossRef]
43. Seyedi-Tabari, E.; Ahrabian, H.; Nowzari-Dalini, A. A new algorithm for generation of different types of RNA. *Int. J. Comput. Math.* **2010**, *87*, 1197–1207. [CrossRef]
44. Nebel, M.E.; Scheid, A.; Weinberg, F. Random generation of RNA secondary structures according to native distributions. *Algorithms Mol. Biol.* **2011**, *6*, 24. [CrossRef] [PubMed]

45. Chee, Y.M.; Chrisnata, J.; Kiah, H.M.; Nguyen, T.T. Efficient encoding/decoding of irreducible words for codes correcting tandem duplications. In Proceedings of the 2018 IEEE International Symposium on Information Theory (ISIT), Vail, CO, USA, 17–22 June 2018.
46. Pop, P.C. The generalized minimum spanning tree problem: An overview of formulations, solution procedures and latest advances. *Eur. J. Oper. Res.* **2020**, *283*, 1–15. [CrossRef]