algorithms

MDPI

Article

Combinatorial Generation Algorithms for Some Lattice Paths Using the Method Based on AND/OR Trees †

Yuriy Shablya

Laboratory of Algorithms and Technologies for Discrete Structures Research, Tomsk State University of Control Systems and Radioelectronics, 634050 Tomsk, Russia; syv@fb.tusur.ru

† This paper is an extended version of our paper published in the proceedings book of the 5th Mediterranean International Conference of Pure & Applied Mathematics and Related Areas (MICOPAM 2022, Antalya, Turkey, 27–30 October 2022).

Abstract: Methods of combinatorial generation make it possible to develop algorithms for generating objects from a set of discrete structures with given parameters and properties. In this article, we demonstrate the possibilities of using the method based on AND/OR trees to obtain combinatorial generation algorithms for combinatorial sets of several well-known lattice paths (North-East lattice paths, Dyck paths, Delannoy paths, Schroder paths, and Motzkin paths). For each considered combinatorial set of lattice paths, we construct the corresponding AND/OR tree structure where the number of its variants is equal to the number of objects in the combinatorial set. Applying the constructed AND/OR tree structures, we have developed new algorithms for ranking and unranking their variants. The performed computational experiments have confirmed the obtained theoretical estimation of asymptotic computational complexity for the developed ranking and unranking algorithms.

Keywords: combinatorial generation; ranking; unranking; AND/OR tree; lattice path; Dyck path; Delannoy path; Schroder path; Motzkin path

Citation: Shablya, Y. Combinatorial Generation Algorithms for Some Lattice Paths Using the Method Based on AND/OR Trees. *Algorithms* **2023**, *16*, 266. https://doi.org/10.3390/a16060266

Academic Editors: Roberto Montemanni and George Karakostas

Received: 5 May 2023
Revised: 21 May 2023
Accepted: 23 May 2023
Published: 26 May 2023

1. Introduction

Combinatorial generation is a branch of science that lies at the intersection of computer science and combinatorics. This scientific direction is devoted to various methods that allow processing sets of discrete structures (combinatorial sets) in terms of generating elements of such sets. The following scientific monographs are devoted to a detailed description of the main concepts and significant results in combinatorial generation: Kreher and Stinson [1], Ruskey [2], and Knuth [3].

The main tasks of combinatorial generation are as follows:

- Listing: the sequential generation of all objects belonging to the combinatorial set;
- Ranking: the assignment of an individual number (a rank) to an object belonging to the combinatorial set (this requires some way to order the elements of the combinatorial set);
- Unranking: the generation of an object belonging to the combinatorial set by the value of its rank (this requires some way to order the elements of the combinatorial set);
- Random selection: the generation of random objects belonging to the combinatorial set.

In discrete mathematics, there are many typical classes of discrete structures (such as permutations, subsets, trees, lattice paths, and others), and each of them has its own specifics. In this article, we study combinatorial generation algorithms for the combinatorial sets of lattice paths [4,5].

A lattice path P is a sequence $P = (P_0, P_1, \ldots, P_k)$ of points P_i in the d-dimensional integer lattice (i.e., $P_i \in \mathbb{Z}^d$), where P_0 is the starting point and P_k is the end point. It is also required to specify a set of possible steps S in the lattice path, where each step $s_i \in S$ is

a vector in the d-dimensional integer lattice (i.e., $s_i \in \mathbb{Z}^d$). Furthermore, a lattice path can be represented as a sequence of steps performed, i.e., $P = (s_1, \ldots, s_k)$, where $s_i = \overrightarrow{P_{i-1}P_i}$.

Lattice paths are widely used in combinatorics, since they are a fairly simple combinatorial object in terms of their representation. In addition, they are well-suited to encoding various other combinatorial objects. Therefore, it is possible to study the properties of complex discrete structures by studying the properties of the corresponding lattice paths. A brief historical review of research related to lattice paths is presented by Humphreys [6]. A more detailed description of the main methods and results in lattice path enumeration is presented in [7].

An analysis of research papers shows that lattice paths are most often used to solve enumerative combinatorics problems (including other more complex structures that have a bijection with lattice paths). However, lattice paths can also be used to solve combinatorial generation problems (i.e., to obtain algorithms for generating such structures). For example, Zaks and Richards [8] developed the algorithms for ranking and unranking lattice paths in the $(t+1)$-dimensional integer lattice that begin at the point $(n_0, n_1, \ldots, n_t)$, end at the origin, and do not go below the hyperplane $x_0 = \sum_{i=1}^{t} (k_i - 1)x_i$. The use of such lattice paths helped them to obtain combinatorial generation algorithms for the ordered trees with $n_0 + 1$ leaves and n_i internal nodes having k_i child nodes where $i = 1, \ldots, t$. Bent [9] developed algorithms for ranking and unranking n-node binary trees by applying Dyck n-paths. In this case, the order in which the trees differ by one rotation was used, and the effect of these rotations on the lattice paths was studied. Parque and Miyashita [10] also studied n-node binary trees based on their corresponding Dyck n-paths and proposed an efficient algorithm for their exhaustive generation that uses $O(n)$ space and $O(1)$ time on average per tree. There are also studies that consider the development of combinatorial generation algorithms directly for lattice paths. For example, Barcucci, Bernini, and Pinzani [11,12] developed the algorithms for exhaustive generation of Motzkin and Schroder positive paths and their prefixes. Kuo [13] considered the North-East lattice paths with t turns and proposed an algorithm for their generation. In addition, the Combinatorial Object Server [14] has an implementation of an algorithm for generating Dyck n-paths.

The purpose of this work is to develop new combinatorial generation algorithms for different types of lattice paths based on a common approach. As an example, it is proposed to consider the following lattice paths: North-East lattice paths, Dyck paths, Delannoy paths, Schroder paths, and Motzkin paths.

2. Materials and Methods

There are several basic general methods for developing combinatorial generation algorithms, such as backtracking [1], the ECO-method [15], Flajolet's method [16], and Kruchinin's method [17]. This article discusses the application of the latter method, which is based on the representation of combinatorial sets in the form of an AND/OR tree structure. An AND/OR tree is a tree structure that contains nodes of two types: OR nodes (such nodes correspond to the union of sets, i.e., it is the union of elements of subsets) and AND nodes (such nodes correspond to the Cartesian product of sets, i.e., it is the combination of elements of subsets). A variant of an AND/OR tree is a tree structure obtained by removing all edges except one for each OR node. In this case, the number of variants of an AND/OR tree is equal to the number of objects of the corresponding combinatorial set.

The main restriction on the application of this method is the requirement to have the cardinality function of a given combinatorial set belonging to the algebra $\{\mathbb{N}, +, \times, R\}$ (i.e., usage of only natural numbers, addition and product operations, and the primitive recursion operator). This article continues the study presented in [18], where the possibility of constructing an AND/OR tree structure for each studied combinatorial set of lattice paths was shown. Therefore, using AND/OR tree structures, it is possible to develop new combinatorial generation algorithms for such lattice paths. A detailed description of the method for developing combinatorial generation algorithms based on AND/OR trees is presented in [17].

3. Results

In this section, we consider the main steps in developing combinatorial generation algorithms for combinatorial sets of several well-known lattice paths by applying the method based on AND/OR trees.

3.1. Combinatorial Generation Algorithms for North-East Lattice Paths

3.1.1. Combinatorial Set

A North-East lattice path is a lattice path in the plane which begins at $(0,0)$, ends at (n,m), and consists of steps $(0,1)$ and $(1,0)$ [19]. The step $(0,1)$ is called the North-step and is denoted by N; the $(1,0)$ step is called the East-step and is denoted by E.

Figure 1 shows all possible 20 variants of the considered North-East lattice paths for $n = 3$ and $m = 3$.

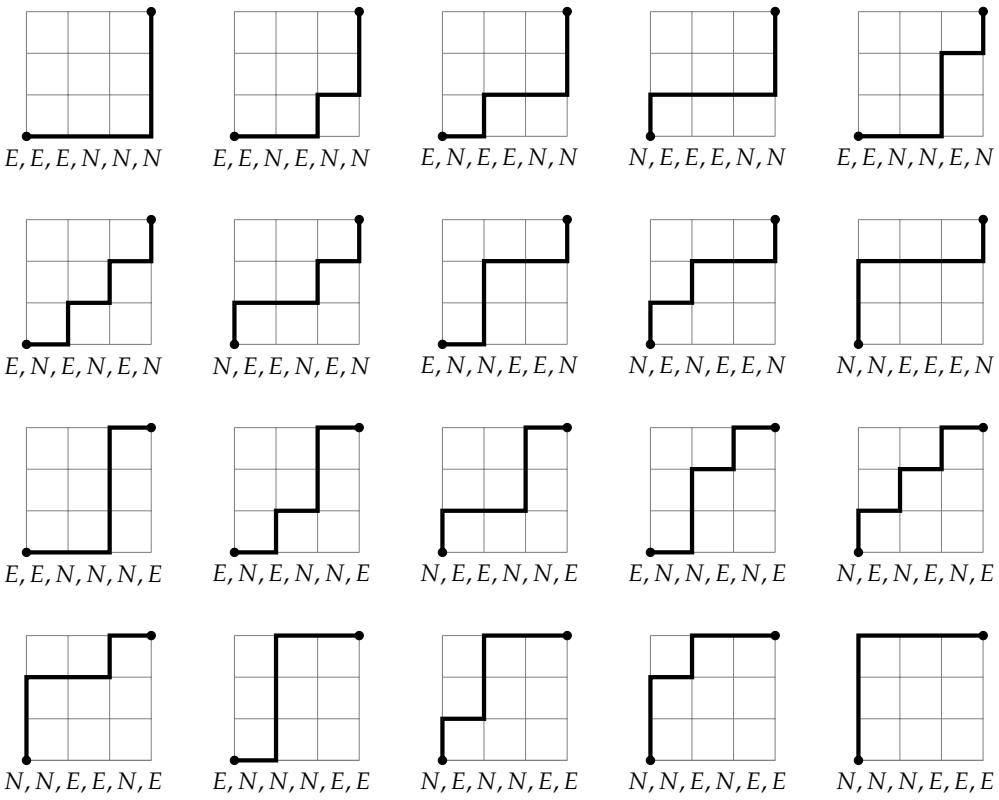


Figure 1. All North-East lattice paths beginning at $(0,0)$ and ending at $(3,3)$.

The total number of North-East lattice paths is defined by the following binomial coefficient (the sequence $A007318$ in OEIS [20]):

$$L_n^m = \binom{n+m}{m} = \binom{n+m}{n}. \tag{1}$$

The value of L_n^m can also be calculated using the following recurrence that belongs to the required algebra $\{\mathbb{N}, +, \times, R\}$:

$$L_n^m = L_n^{m-1} + L_{n-1}^m, \quad L_n^0 = L_0^m = 1. \tag{2}$$

In addition, the sequence of values of L_n^m is defined by the bivariate generating function

$$\sum_{n\geq 0}\sum_{m\geq 0} L_n^m x^n y^m = \frac{1}{1-x-y}.$$

3.1.2. AND/OR Tree Structure

Since Equation (2) satisfies the requirements of the applied method, the corresponding AND/OR tree structure for L_n^m can therefore be constructed (see Figure 2).

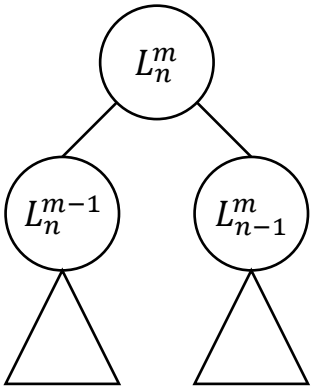


Figure 2. An AND/OR tree for L_n^m.

For this AND/OR tree structure, there are the following initial conditions:

- Each node labeled L_n^0 is a leaf node in the AND/OR tree for L_n^m;
- Each node labeled L_0^m is a leaf node in the AND/OR tree for L_n^m.

Figure 3 presents an example of the AND/OR tree structure for L_n^m where $n = 3$ and $m = 3$. The total number of its variants is equal to $L_3^3 = 20$. Since the obtained AND/OR tree structure does not contain AND nodes, each variant of such a tree is a path from the root to a leaf.

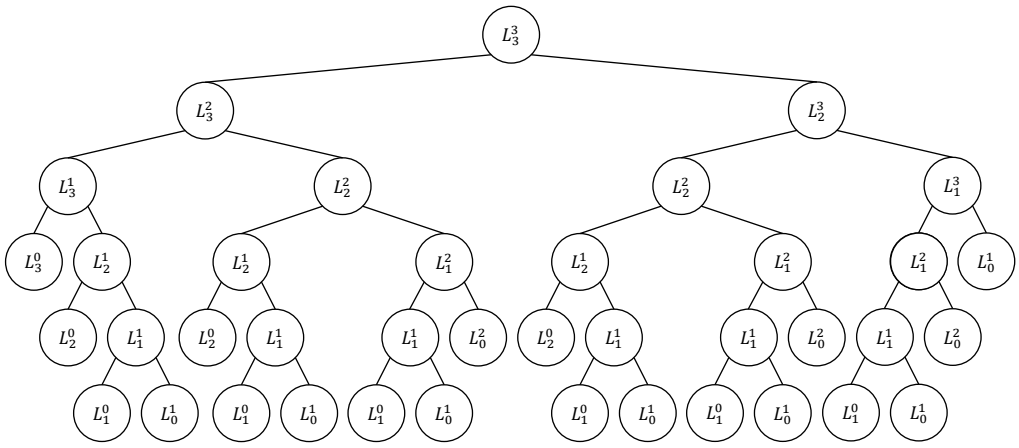


Figure 3. An AND/OR tree for L_3^3.

For a compact representation, a variant of the AND/OR tree for L_n^m is encoded by a sequence $v = (v_1, v_2, \ldots)$ of the selected children of the OR nodes in this tree (the left child corresponds to $v_i = 1$ and the right child corresponds to $v_i = 2$).

Theorem 1. *There is a bijection between the set of North-East lattice paths beginning at* $(0, 0)$ *and ending at* (n, m) *and the set of variants of the AND/OR tree for* L_n^m.

Proof. The total number of North-East lattice paths beginning at $(0, 0)$ and ending at (n, m) is equal to L_n^m. The total number of variants of the AND/OR tree for L_n^m presented in Figure 2 is also equal to L_n^m. Therefore, it is possible to associate each such lattice path with one specific variant of the AND/OR tree for L_n^m. A bijection between the set of North-East lattice paths beginning at $(0, 0)$ and ending at (n, m) and the set of variants of the AND/OR tree for L_n^m is defined by the following rules:

- Each selected left child of the OR node labeled L_n^m determines the addition of one North-step to the North-East lattice path obtained by the subtree of the node labeled L_n^{m-1}: the resulting lattice path is $(s_1, \ldots, s_{n+m-1}, N)$;
- Each selected right child of the OR node labeled L_n^m determines the addition of one East-step to the North-East lattice path obtained by the subtree of the node labeled L_{n-1}^m: the resulting lattice path is $(s_1, \ldots, s_{n+m-1}, E)$;
- Each leaf node labeled L_n^0 determines the lattice path from $(0,0)$ to $(n,0)$ that consists of n East-steps: the resulting lattice path is $(s_1, \ldots, s_n) = (E, \ldots, E)$;
- Each leaf node labeled L_0^m determines the lattice path from $(0,0)$ to $(0,m)$ that consists of m North-steps: the resulting lattice path is $(s_1, \ldots, s_m) = (N, \ldots, N)$.

□

The algorithms that implement the developed bijection rules have linear time complexity $O(n+m)$, since they require one pass to fill a sequence of $(n+m)$ elements. An example of applying these bijection rules is presented in Table 1.

Table 1. Ranking the set of North-East lattice paths beginning at $(0,0)$ and ending at $(3,3)$.

Lattice Path	Variant of AND/OR Tree	Rank
E,E,E,N,N,N	$(1,1,1)$	0
E,E,N,E,N,N	$(1,1,2,1)$	1
E,N,E,E,N,N	$(1,1,2,2,1)$	2
N,E,E,E,N,N	$(1,1,2,2,2)$	3
E,E,N,N,E,N	$(1,2,1,1)$	4
E,N,E,N,E,N	$(1,2,1,2,1)$	5
N,E,E,N,E,N	$(1,2,1,2,2)$	6
E,N,N,E,E,N	$(1,2,2,1,1)$	7
N,E,N,E,E,N	$(1,2,2,1,2)$	8
N,N,E,E,E,N	$(1,2,2,2)$	9
E,E,N,N,N,E	$(2,1,1,1)$	10
E,N,E,N,N,E	$(2,1,1,2,1)$	11
N,E,E,N,N,E	$(2,1,1,2,2)$	12
E,N,N,E,N,E	$(2,1,2,1,1)$	13
N,E,N,E,N,E	$(2,1,2,1,2)$	14
N,N,E,E,N,E	$(2,1,2,2)$	15
E,N,N,N,E,E	$(2,2,1,1,1)$	16
N,E,N,N,E,E	$(2,2,1,1,2)$	17
N,N,E,N,E,E	$(2,2,1,2)$	18
N,N,N,E,E,E	$(2,2,2)$	19

3.1.3. Ranking and Unranking Algorithms

Applying the general approach described in [17], we develop algorithms for ranking (Algorithm 1) and unranking (Algorithm 2) the variants of the AND/OR tree for L_n^m. In these algorithms, $()$ denotes an empty sequence and the function $\texttt{concat}(a,b)$ denotes merging sequences a and b.

Algorithm 1: An algorithm for ranking a variant of the AND/OR tree for L_n^m.

```
1 Rank_L(v = (v1, v2, ...), n, m)
2 begin
3     if n = 0 or m = 0 then r := 0
4     else
5         if v1 = 1 then r := Rank_L((v2, ...), n, m − 1)
6         else r := L_n^{m−1} + Rank_L((v2, ...), n − 1, m)
7     end
8     return r
9 end
```

Algorithm 2: An algorithm for unranking a variant of the AND/OR tree for L_n^m.

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

The developed algorithms have the following computational complexity:

- Algorithm 1 has at most m recursive calls where $v_1 = 1$ (each such recursive call requires one assignment) and has at most n recursive calls where $v_1 = 2$ (each such recursive call requires calculations of L_n^{m-1}). Applying Equation (1), the calculation of L_n^m has linear time complexity $O(m)$ for $m < n$ and $O(n)$ for $m > n$. Hence, Algorithm 1 has polynomial time complexity $O(nm)$ for $m < n$ and $O(m + n^2)$ for $m > n$;
- Algorithm 2 has at most $(n + m)$ recursive calls where each such recursive call requires calculations of L_n^{m-1}. Hence, Algorithm 2 has polynomial time complexity $O(m(n + m))$ for $m < n$ and $O(n(n + m))$ for $m > n$.

Table 1 presents an example of ranking the combinatorial set of all North-East lattice paths beginning at $(0, 0)$ and ending at $(3, 3)$.

3.2. Combinatorial Generation Algorithms for Dyck Paths

3.2.1. Combinatorial Set

A Dyck n-path is a lattice path in the plane which begins at $(0, 0)$, ends at (n, n), consists of steps $(0, 1)$ and $(1, 0)$, and never rises above the diagonal $y = x$ [21]. The set of Dyck n-paths is a subset of the North-East lattice paths beginning at $(0, 0)$ and ending at (n, n).

Figure 4 shows all possible 5 variants of the considered Dyck paths for $n = 3$.

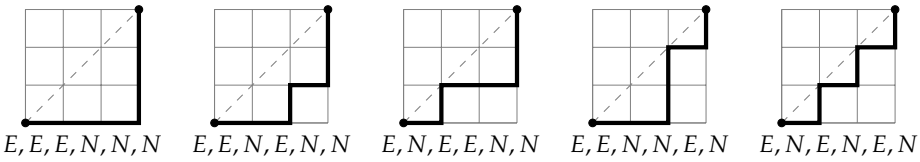


Figure 4. All Dyck paths beginning at $(0, 0)$ and ending at $(3, 3)$.

The total number of Dyck n-paths is defined by the Catalan number C_n (the sequence $A000108$ in OEIS [20]):

$$C_n = \frac{1}{n+1}\binom{2n}{n}. \tag{3}$$

The value of C_n can also be calculated using the following recurrence that belongs to the required algebra $\{\mathbb{N}, +, \times, R\}$:

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}, \quad C_0 = 1. \tag{4}$$

In addition, the sequence of values of C_n is defined by the generating function

$$\sum_{n\geq 0} C_n x^n = \frac{1 - \sqrt{1 - 4x}}{2x}.$$

3.2.2. AND/OR Tree Structure

Since Equation (4) satisfies the requirements of the applied method, the corresponding AND/OR tree structure for C_n can therefore be constructed (see Figure 5).

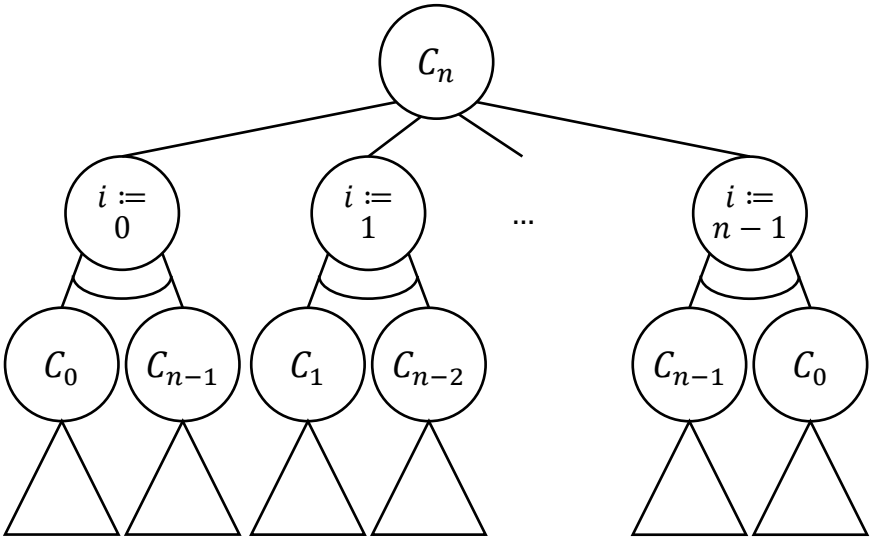


Figure 5. An AND/OR tree for C_n.

For this AND/OR tree structure, there is the following initial condition:

- Each node labeled C_0 is a leaf node in the AND/OR tree for C_n.

Figure 6 presents an example of the AND/OR tree structure for C_n where $n = 3$. The total number of its variants is equal to $C_3 = 5$.

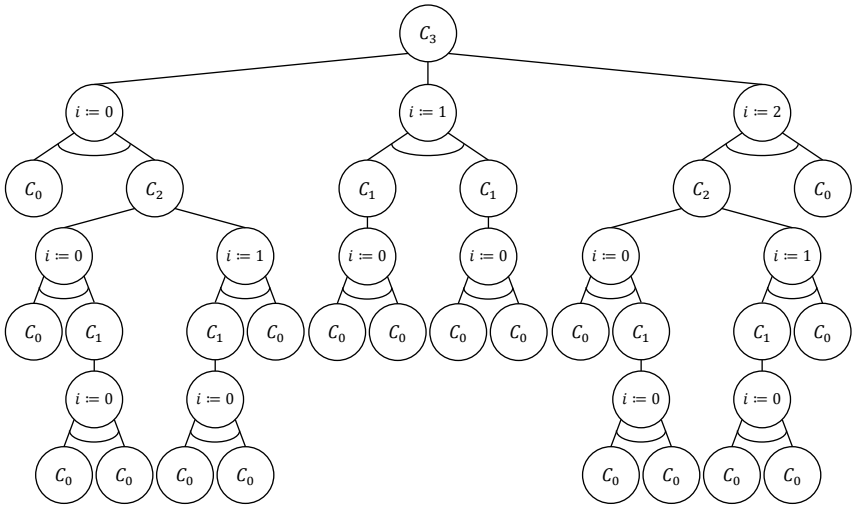


Figure 6. An AND/OR tree for C_3.

For a compact representation, a variant of the AND/OR tree for C_n is encoded by a sequence $v = (I, v_1, v_2)$, where the following apply:

- An empty sequence $v = ()$ corresponds to the selection of a leaf node labeled C_0;
- I corresponds to the selected value of i in the AND/OR tree for C_n;
- v_1 corresponds to the variant of the subtree of the node labeled C_I (the left subtree);
- v_2 corresponds to the variant of the subtree of the node labeled C_{n-1-I} (the right subtree).

Theorem 2. *There is a bijection between the set of Dyck n-paths and the set of variants of the AND/OR tree for C_n.*

Proof. The total number of Dyck n-paths is equal to C_n. The total number of variants of the AND/OR tree for C_n presented in Figure 5 is also equal to C_n. Therefore, it is possible to associate each such lattice path with one specific variant of the AND/OR tree for C_n. A bijection between the set of Dyck n-paths and the set of variants of the AND/OR tree for C_n is defined by the following rules:

- Each selected child of the OR node labeled C_n determines the addition of one East-step and one North-step to the Dyck $(n-1)$-path that merges the Dyck I-path obtained by the subtree of the node labeled C_I and consisting of $2I$ steps and the Dyck $(n-1-I)$-path obtained by the subtree of the node labeled C_{n-1-I} and consisting of $(2n-2I-2)$ steps: the resulting Dyck n-path is $(s_1, \ldots, s_{2I}, E, s_{2I+2}, \ldots, s_{2n-1}, N)$;
- The subtree of the node labeled C_I (the left subtree) determines the Dyck I-path of the form $(s_1, \ldots, s_{2I})$;
- The subtree of the node labeled C_{n-1-I} (the right subtree) determines the Dyck $(n-1-I)$-path of the form $(s_{2I+2}, \ldots, s_{2n-1})$;
- Each leaf node labeled C_0 determines the empty lattice path $()$.

□

The algorithms that implement the developed bijection rules have polynomial time complexity $O(n^2)$, since they make $2n$ recursive calls, where each such recursive call requires one pass to fill a sequence of $2n$ elements. An example of applying these bijection rules is presented in Table 2.

Table 2. Ranking the set of Dyck paths beginning at $(0,0)$ and ending at $(3,3)$.

Lattice Path	Variant of AND/OR Tree	Rank
E,E,E,N,N,N	$(0,(),(0,(),(0,(),())))$	0
E,E,N,E,N,N	$(0,(),(1,(0,(),()),()))$	1
E,N,E,E,N,N	$(1,(0,(),()),(0,(),()))$	2
E,E,N,N,E,N	$(2,(0,(),(0,(),())),())$	3
E,N,E,N,E,N	$(2,(1,(0,(),()),()),())$	4

3.2.3. Ranking and Unranking Algorithms

Applying the general approach described in [17], we developed algorithms for ranking (Algorithm 3) and unranking (Algorithm 4) the variants of the AND/OR tree for C_n.

The developed algorithms have the following the computational complexity:

- Algorithm 3 has at most n recursive calls, where each such recursive call requires calculations of C_n maximum $(2n-1)$ times. Applying Equation (3), the calculation of C_n has linear time complexity $O(n)$. Hence, Algorithm 3 has polynomial time complexity $O(n^3)$;
- Algorithm 4 has at most n recursive calls, where each such recursive call requires calculations of C_n maximum $(2n+1)$ times. Hence, Algorithm 4 has polynomial time complexity $O(n^3)$.

Table 2 presents an example of ranking the combinatorial set of all Dyck paths beginning at $(0,0)$ and ending at $(3,3)$.

Algorithm 3: An algorithm for ranking a variant of the AND/OR tree for C_n.

```
1  Rank_C(v = (I, v1, v2), n)
2  begin
3      if n = 0 then r := 0
4      else
5          sum := Σ_{i=0}^{I-1} C_i C_{n-1-i}
6          w1 := C(I)
7          l1 := Rank_C(v1, I)
8          l2 := Rank_C(v2, n - 1 - I)
9          r := sum + l1 + w1 l2
10     end
11     return r
12 end
```

Algorithm 4: An algorithm for unranking a variant of the AND/OR tree for C_n.

```
1  Unrank_C(r, n)
2  begin
3      if n = 0 then v := ()
4      else
5          sum := 0
6          for i := 0 to n - 1 do
7              s := C_i C_{n-1-i}
8              if sum + s > r then
9                  r := r - sum
10                 I := i
11                 break
12             end
13             sum := sum + s
14         end
15         w1 := C(I)
16         l1 := r  mod w1
17         l2 := ⌊r / w1⌋
18         v1 := Unrank_C(l1, I)
19         v2 := Unrank_C(l2, n - 1 - I)
20         v := (I, v1, v2)
21     end
22     return v
23 end
```

3.3. Combinatorial Generation Algorithms for Delannoy Paths

3.3.1. Combinatorial Set

A Delannoy path is a lattice path in the plane which begins at $(0,0)$, ends at (n,m), and consists of steps $(0,1)$, $(1,0)$ and $(1,1)$ [22]. The step $(1,1)$ is called the North-East-step and denoted by NE. Thus, the set of Delannoy paths is a generalization of the North-East lattice paths beginning at $(0,0)$ and ending at (n,m) by adding the North-East-steps.

Figure 7 shows all possible 25 variants of the considered Delannoy paths for $n = 3$ and $m = 2$.

The total number of Delannoy paths is defined by the Delannoy number D_n^m (the sequence $A008288$ in OEIS [20]):

$$D_n^m = \sum_{i=1}^{\min(n,m)} \binom{n}{i} \binom{m}{i} 2^i. \tag{5}$$

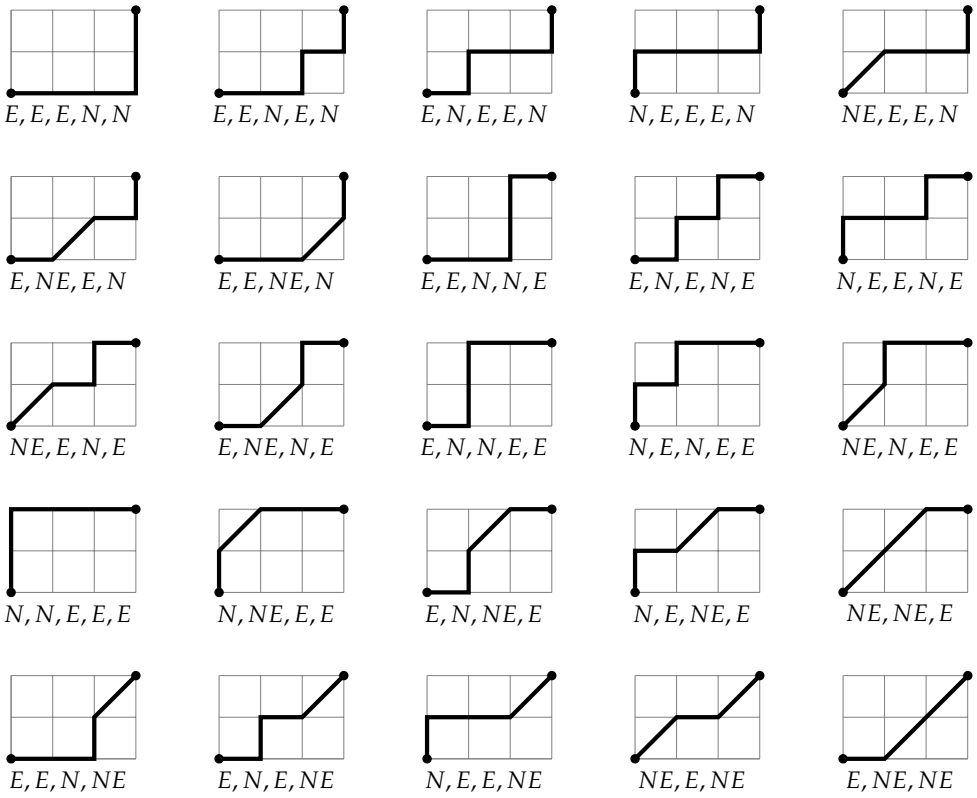


Figure 7. All Delannoy paths beginning at $(0,0)$ and ending at $(3,2)$.

The value of D_n^m can also be calculated using the following recurrence that belongs to the required algebra $\{\mathbb{N}, +, \times, R\}$:

$$D_n^m = D_n^{m-1} + D_{n-1}^m + D_{n-1}^{m-1}, \quad D_n^0 = D_0^m = 1. \tag{6}$$

In addition, the sequence of values of D_n^m is defined by the bivariate generating function

$$\sum_{n\geq 0} \sum_{m\geq 0} D_n^m x^n y^m = \frac{1}{1-x-y-xy}.$$

3.3.2. AND/OR Tree Structure

Since Equation (6) satisfies the requirements of the applied method, the corresponding AND/OR tree structure for D_n^m can therefore be constructed (see Figure 8).

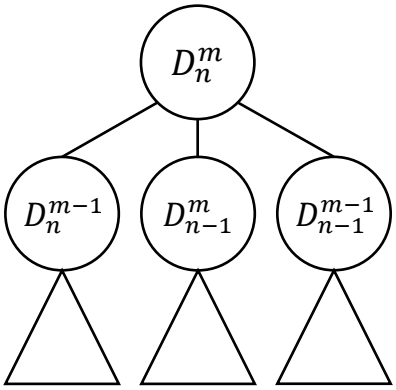


Figure 8. An AND/OR tree for D_n^m.

For this AND/OR tree structure, there are the following initial conditions:

- Each node labeled D_n^0 is a leaf node in the AND/OR tree for D_n^m;
- Each node labeled D_0^m is a leaf node in the AND/OR tree for D_n^m.

Figure 9 presents an example of the AND/OR tree structure for D_n^m where $n = 3$ and $m = 2$. The total number of its variants is equal to $D_3^2 = 25$. Since the obtained AND/OR tree structure does not contain AND nodes, each variant of such a tree is a path from the root to a leaf.

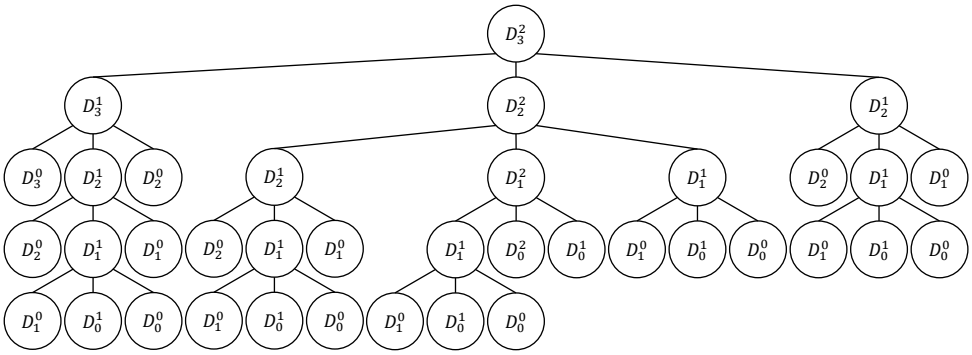


Figure 9. An AND/OR tree for D_3^2.

For a compact representation, a variant of the AND/OR tree for D_n^m is encoded by a sequence $v = (v_1, v_2, \ldots)$ of the selected children of the OR nodes in this tree (the left child corresponds to $v_i = 1$, the middle child corresponds to $v_i = 2$ and the right child corresponds to $v_i = 3$).

Theorem 3. *There is a bijection between the set of Delannoy paths beginning at* $(0, 0)$ *and ending at* (n, m) *and the set of variants of the AND/OR tree for* D_n^m.

Proof. The total number of Delannoy paths beginning at $(0, 0)$ and ending at (n, m) is equal to D_n^m. The total number of variants of the AND/OR tree for D_n^m presented in Figure 8 is also equal to D_n^m. Therefore, it is possible to associate each such lattice path with one specific variant of the AND/OR tree for D_n^m. A bijection between the set of Delannoy paths beginning at $(0, 0)$ and ending at (n, m) and the set of variants of the AND/OR tree for D_n^m is defined by the following rules:

- Each selected left child of the OR node labeled D_n^m determines the addition of one North-step to the Delannoy path obtained by the subtree of the node labeled D_n^{m-1} and consisting of k steps: the resulting lattice path is $(s_1, \ldots, s_k, N)$;

- Each selected middle child of the OR node labeled D_n^m determines the addition of one East-step to the Delannoy path obtained by the subtree of the node labeled D_{n-1}^m and consisting of k steps: the resulting lattice path is $(s_1, \ldots, s_k, E)$;
- Each selected right child of the OR node labeled D_n^m determines the addition of one North-East-step to the Delannoy path obtained by the subtree of the node labeled D_{n-1}^{m-1} and consisting of k steps: the resulting lattice path is $(s_1, \ldots, s_k, NE)$;
- Each leaf node labeled D_n^0 determines the Delannoy path from $(0,0)$ to $(n,0)$ that consists of n East-steps: the resulting lattice path is $(s_1, \ldots, s_n) = (E, \ldots, E)$;
- Each leaf node labeled D_0^m determines the Delannoy path from $(0,0)$ to $(0,m)$ that consists of m North-steps: the resulting lattice path is $(s_1, \ldots, s_m) = (N, \ldots, N)$.

□

The algorithms that implement the developed bijection rules have linear time complexity $O(n+m)$, since they require one pass to fill a sequence of maximum $(n+m)$ elements. An example of applying these bijection rules is presented in Table 3.

Table 3. Ranking the set of Delannoy paths beginning at $(0,0)$ and ending at $(3,2)$.

Lattice Path	Variant of AND/OR Tree	Rank
E,E,E,N,N	$(1,1)$	0
E,E,N,E,N	$(1,2,1)$	1
E,N,E,E,N	$(1,2,2,1)$	2
N,E,E,E,N	$(1,2,2,2)$	3
NE,E,E,N	$(1,2,2,3)$	4
E,NE,E,N	$(1,2,3)$	5
E,E,NE,N	$(1,3)$	6
E,E,N,N,E	$(2,1,1)$	7
E,N,E,N,E	$(2,1,2,1)$	8
N,E,E,N,E	$(2,1,2,2)$	9
NE,E,N,E	$(2,1,2,3)$	10
E,NE,N,E	$(2,1,3)$	11
E,N,N,E,E	$(2,2,1,1)$	12
N,E,N,E,E	$(2,2,1,2)$	13
NE,N,E,E	$(2,2,1,3)$	14
N,N,E,E,E	$(2,2,2)$	15
N,NE,E,E	$(2,2,3)$	16
E,N,NE,E	$(2,3,1)$	17
N,E,NE,E	$(2,3,2)$	18
NE,NE,E	$(2,3,3)$	19
E,E,N,NE	$(3,1)$	20
E,N,E,NE	$(3,2,1)$	21
N,E,E,NE	$(3,2,2)$	22
NE,E,NE	$(3,2,3)$	23
E,NE,NE	$(3,3)$	24

3.3.3. Ranking and Unranking Algorithms

Applying the general approach described in [17], we develop algorithms for ranking (Algorithm 5) and unranking (Algorithm 6) the variants of the AND/OR tree for D_n^m.

Algorithm 5: An algorithm for ranking a variant of the AND/OR tree for D_n^m.

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

Algorithm 6: An algorithm for unranking a variant of the AND/OR tree for D_n^m.

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

The developed algorithms have the following computational complexity:

- Algorithm 5 has at most m recursive calls where $v_1 = 1$ (each such recursive call requires one assignment), has at most n recursive calls where $v_1 = 2$ (each such recursive call requires calculations of D_n^{m-1}), and has at most $\min(n, m)$ recursive calls where $v_1 = 3$ (each such recursive call requires calculations of D_n^{m-1} and D_{n-1}^m). Applying Equation (5), the calculation of D_n^m has polynomial time complexity $O(m^2)$ for $m < n$ and $O(n^2)$ for $m > n$. Hence, Algorithm 5 has polynomial time complexity $O(m^2(n+m))$ for $m < n$ and $O(m+n^3)$ for $m > n$;
- Algorithm 6 has at most $(n+m)$ recursive calls where each such recursive call requires calculations of D_n^{m-1} or D_{n-1}^m. Hence, Algorithm 2 has polynomial time complexity $O(m^2(n+m))$ for $m < n$ and $O(n^2(n+m))$ for $m > n$.

Table 3 presents an example of ranking the combinatorial set of all Delannoy paths beginning at $(0,0)$ and ending at $(3,2)$.

3.4. Combinatorial Generation Algorithms for Schroder Paths

3.4.1. Combinatorial Set

A Schroder n-path is a lattice path in the plane which begins at $(0,0)$, ends at (n,n), consists of steps $(0,1)$, $(1,0)$, and $(1,1)$, and never rises above the diagonal $y = x$ [23]. The set of Schroder n-paths is a subset of the Delannoy paths beginning at $(0,0)$ and ending at (n,n).

Figure 10 shows all possible 22 variants of the considered Schroder paths for $n = 3$.

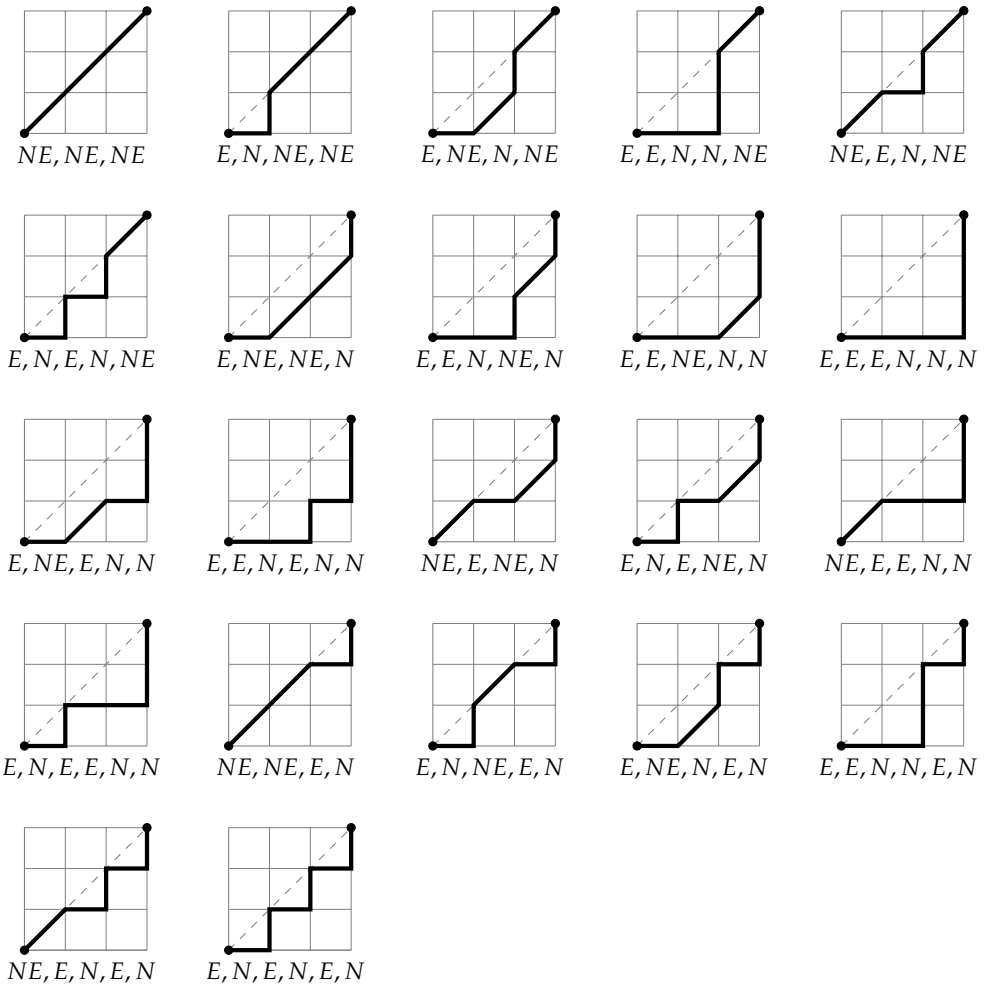


Figure 10. All Schroder paths beginning at $(0,0)$ and ending at $(3,3)$.

The total number of Schroder n-paths is defined by the Schroder number S_n (the sequence $A006318$ in OEIS [20]):

$$S_n = \sum_{i=0}^{n} \frac{1}{i+1} \binom{n+i}{i} \binom{n}{i}. \tag{7}$$

The value of S_n can also be calculated using the following recurrence that belongs to the required algebra $\{\mathbb{N}, +, \times, R\}$:

$$S_n = S_{n-1} + \sum_{i=0}^{n-1} S_i S_{n-1-i}, \quad S_0 = 1. \tag{8}$$

In addition, the sequence of values of S_n is defined by the generating function

$$\sum_{n \geq 0} S_n x^n = \frac{1 - x - \sqrt{1 - 6x + x^2}}{2x}.$$

3.4.2. AND/OR Tree Structure

Since Equation (8) satisfies the requirements of the applied method, the corresponding AND/OR tree structure for S_n can therefore be constructed (see Figure 11).

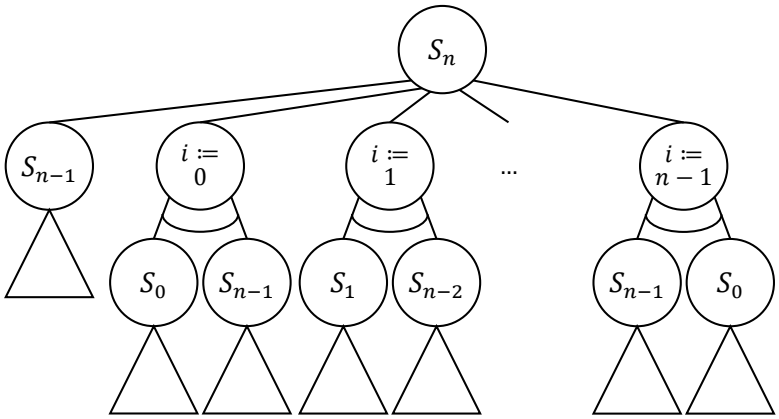


Figure 11. An AND/OR tree for S_n.

For this AND/OR tree structure, there is the following initial condition:

- Each node labeled S_0 is a leaf node in the AND/OR tree for S_n.

Figure 12 presents an example of the AND/OR tree structure for S_n where $n = 3$. The total number of its variants is equal to $S_3 = 22$.

For a compact representation, a variant of the AND/OR tree for S_n is encoded by a sequence v:

1. If the left child of the OR nodde labeled S_n is selected, then $v = (I, v_1)$, where
   - $I = -1$;
   - v_1 corresponds to the variant of the subtree of the node labeled S_{n-1};
2. Otherwise $v = (I, v_1, v_2)$, where
   - I corresponds to the selected value of i in the AND/OR tree for S_n;
   - v_1 corresponds to the variant of the subtree of the node labeled S_I;
   - v_2 corresponds to the variant of the subtree of the node labeled S_{n-1-I};
3. An empty sequence $v = ()$ corresponds to the selection of a leaf node labeled S_0.

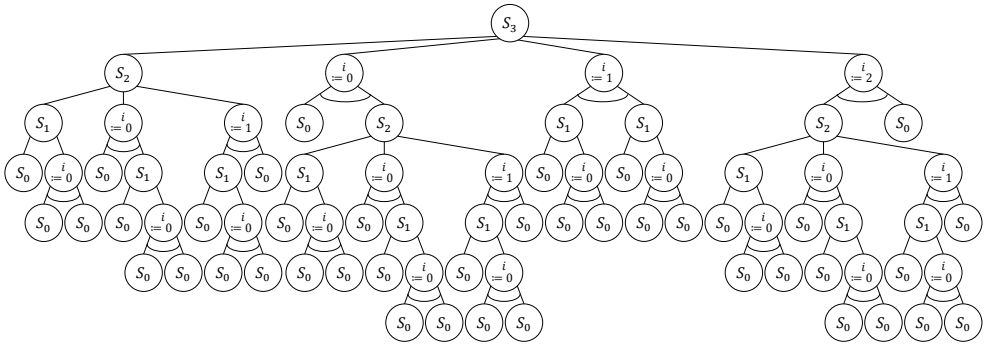


Figure 12. An AND/OR tree for S_3.

Theorem 4. *There is a bijection between the set of Schroder n-paths and the set of variants of the AND/OR tree for S_n.*

Proof. The total number of Schroder n-paths is equal to S_n. The total number of variants of the AND/OR tree for S_n presented in Figure 11 is also equal to S_n. Therefore, it is possible to associate each such lattice path with one specific variant of the AND/OR tree for S_n. A bijection between the set of Schroder n-paths and the set of variants of the AND/OR tree for S_n is defined by the following rules:

- Each selected left child of the OR node labeled S_n determines the addition of one North-East-step to the Schroder $(n-1)$-path obtained by the subtree of the node labeled S_{n-1} and consisting of k steps: the resulting Schroder n-path is $(s_1, \ldots, s_k, NE)$;

- Each selected child of the OR node labeled S_n determines the addition of one East-step and one North-step to the Schroder $(n-1)$-path that merges the Schroder I-path obtained by the subtree of the node labeled S_I and consisting of k_1 steps and the Schroder $(n-1-I)$-path obtained by the subtree of the node labeled S_{n-1-I} and consisting of k_2 steps: the resulting Schroder n-path is $(s_1, \ldots, s_{k_1}, E, s_{k_1+2}, \ldots, s_{k_1+1+k_2}, N)$;
- The subtree of the node labeled S_I (the left subtree) determines the Schroder I-path of the form $(s_1, \ldots, s_{k_1})$;
- The subtree of the node labeled S_{n-1-I} (the right subtree) determines the Schroder $(n-1-I)$-path of the form $(s_{k_1+2}, \ldots, s_{k_1+1+k_2})$;
- Each leaf node labeled S_0 determines the empty lattice path $()$.

□

The algorithms that implement the developed bijection rules have polynomial time complexity $O(n^2)$, since they make n recursive calls where each such recursive call requires one pass to fill a sequence of maximum $2n$ elements. An example of applying these bijection rules is presented in Table 4.

Table 4. Ranking the set of Schroder paths beginning at $(0,0)$ and ending at $(3,3)$.

Lattice Path	Variant of AND/OR Tree	Rank
NE, NE, NE	$(-1,(-1,(-1,())))$	0
E, N, NE, NE	$(-1,(-1,(0,(),())))$	1
E, NE, N, NE	$(-1,(0,(),(-1,())))$	2
E, E, N, N, NE	$(-1,(0,(),(0,(),())))$	3
NE, E, N, NE	$(-1,(1,(-1,()),()))$	4
E, N, E, N, NE	$(-1,(1,(0,(),()),()))$	5
E, NE, NE, N	$(0,(),(-1,(-1,())))$	6
E, E, N, NE, N	$(0,(),(-1,(0,(),())))$	7
E, E, NE, N, N	$(0,(),(0,(),(-1,())))$	8
E, E, E, N, N, N	$(0,(),(0,(),(0,(),())))$	9
E, NE, E, N, N	$(0,(),(1,(-1,()),()))$	10
E, E, N, E, N, N	$(0,(),(1,(0,(),()),()))$	11
NE, E, NE, N	$(1,(-1,()),(-1,()))$	12
E, N, E, NE, N	$(1,(0,(),()),(-1,()))$	13
NE, E, E, N, N	$(1,(-1,()),(0,(),()))$	14
E, N, E, E, N, N	$(1,(0,(),()),(0,(),()))$	15
NE, NE, E, N	$(2,(-1,(-1,())),())$	16
E, N, NE, E, N	$(2,(-1,(0,(),())),())$	17
E, NE, N, E, N	$(2,(0,(),(-1,())),())$	18
E, E, N, N, E, N	$(2,(0,(),(0,(),())),())$	19
NE, E, N, E, N	$(2,(1,(-1,()),()),())$	20
E, N, E, N, E, N	$(2,(1,(0,(),()),()),())$	21

3.4.3. Ranking and Unranking Algorithms

Applying the general approach described in [17], we develop algorithms for ranking (Algorithm 7) and unranking (Algorithm 8) the variants of the AND/OR tree for S_n.

The developed algorithms have the following the computational complexity:

- Algorithm 7 has at most n recursive calls where $I = -1$ (each such recursive call requires one assignment) and has at most n recursive calls where $I \neq -1$ (each such recursive call requires calculations of S_n maximum $2n$ times). Applying Equation (7), the calculation of S_n has polynomial time complexity $O(n^2)$. Hence, Algorithm 7 has polynomial time complexity $O(n^4)$;
- Algorithm 8 has at most n recursive calls where $I = -1$ (each such recursive call requires calculations of S_n) and has at most n recursive calls where $I \neq -1$ (each such recursive call requires calculations of S_n maximum $(2n+2)$ times). Hence, Algorithm 8 has polynomial time complexity $O(n^4)$.

Algorithm 7: An algorithm for ranking a variant of the AND/OR tree for S_n.

```
1  Rank_S(v = (I, v_1, ...), n)
2  begin
3      if n = 0 then r := 0
4      else
5          if I = −1 then r := Rank_S(v_1, n − 1)
6          else
7              sum := S(n − 1) + Σ_{i=0}^{I−1} S_i S_{n−1−i}
8              w_1 := S(I)
9              l_1 := Rank_S(v_1, I)
10             l_2 := Rank_S(v_2, n − 1 − I)
11             r := sum + l_1 + w_1 l_2
12         end
13     end
14     return r
15 end
```

Algorithm 8: An algorithm for unranking a variant of the AND/OR tree for S_n.

```
1  Unrank_S(r, n)
2  begin
3      if n = 0 then v := ()
4      else
5          if r < S_{n−1} then
6              I := −1
7              v_1 := Unrank_S(r, n − 1)
8              v := (I, v_1)
9          end
10         else
11             r := r − S_{n−1}
12             sum := 0
13             for i := 0 to n − 1 do
14                 s := S_i S_{n−1−i}
15                 if sum + s > r then
16                     r := r − sum
17                     I := i
18                     break
19                 end
20                 sum := sum + s
21             end
22             w_1 := S(I)
23             l_1 := r mod w_1
24             l_2 := ⌊r / w_1⌋
25             v_1 := Unrank_S(l_1, I)
26             v_2 := Unrank_S(l_2, n − 1 − I)
27             v := (I, v_1, v_2)
28         end
29     end
30     return v
31 end
```

Table 4 presents an example of ranking the combinatorial set of all Schroder paths beginning at $(0,0)$ and ending at $(3,3)$.

3.5. Combinatorial Generation Algorithms for Motzkin Paths

3.5.1. Combinatorial Set

A Motzkin n-path is a lattice path in the plane which begins at $(0,0)$, ends at (n,n), consists of steps $(0,2)$, $(2,0)$ and $(1,1)$, and never rises above the diagonal $y = x$ [24]. The step $(0,2)$ is the double North-step, the step $(2,0)$ is the double East-step. The set of Motzkin n-paths is a subset of the Schroder n-paths.

Figure 13 shows all possible 4 variants of the considered Motzkin paths for $n = 3$.

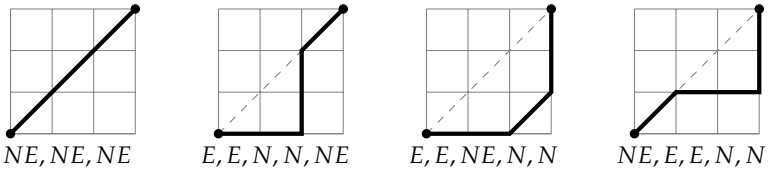


Figure 13. All Motzkin paths beginning at $(0,0)$ and ending at $(3,3)$.

The total number of Motzkin n-paths is defined by the Motzkin number M_n (the sequence $A001006$ in OEIS [20]):

$$M_n = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \frac{1}{i+1} \binom{n}{2i} \binom{2i}{i}. \tag{9}$$

The value of M_n can also be calculated using the following recurrence that belongs to the required algebra $\{\mathbb{N}, +, \times, R\}$:

$$M_n = M_{n-1} + \sum_{i=0}^{n-2} M_i M_{n-2-i}, \quad M_0 = M_1 = 1. \tag{10}$$

In addition, the sequence of values of M_n is defined by the generating function

$$\sum_{n \geq 0} M_n x^n = \frac{1 - x - \sqrt{1 - 2x - 3x^2}}{2x^2}.$$

3.5.2. AND/OR Tree Structure

Since Equation (10) satisfies the requirements of the applied method, the corresponding AND/OR tree structure for M_n can therefore be constructed (see Figure 14).

For this AND/OR tree structure, there is the following initial condition:

- each node labeled M_0 or M_1 is a leaf node in the AND/OR tree for M_n.

Figure 15 presents an example of the AND/OR tree structure for M_n where $n = 3$. The total number of its variants is equal to $M_3 = 4$.

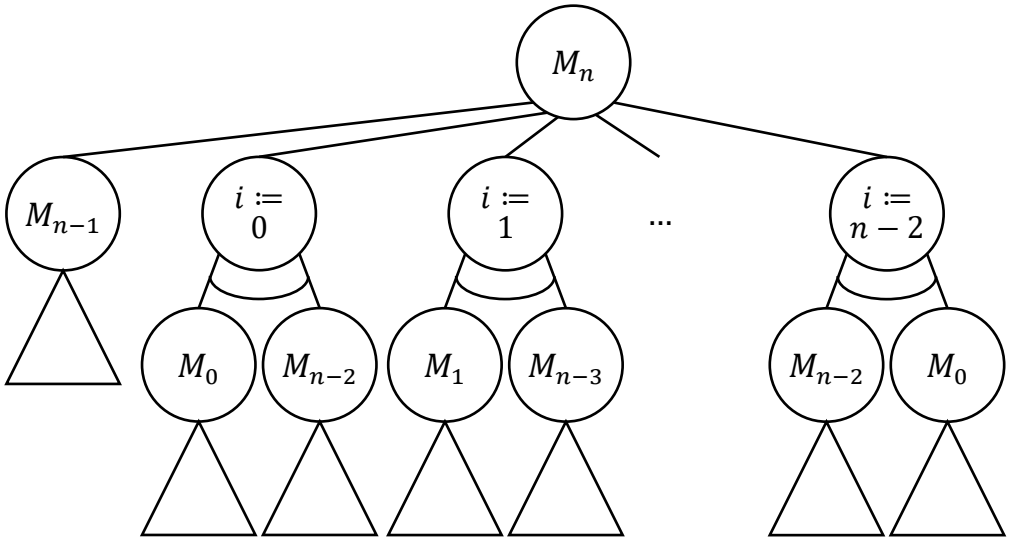


Figure 14. An AND/OR tree for M_n.

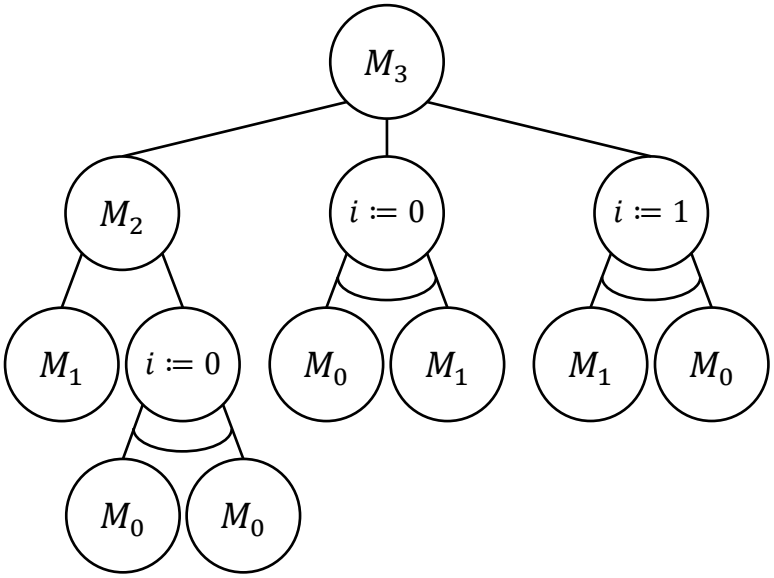


Figure 15. An AND/OR tree for M_3.

For a compact representation, a variant of the AND/OR tree for M_n is encoded by a sequence v:

1. If the left child of the OR node labeled M_n is selected, then $v = (I, v_1)$, where
   - $I = -1$;
   - v_1 corresponds to the variant of the subtree of the node labeled M_{n-1};
2. Otherwise $v = (I, v_1, v_2)$, where
   - I corresponds to the selected value of i in the AND/OR tree for M_n;
   - v_1 corresponds to the variant of the subtree of the node labeled M_I;
   - v_2 corresponds to the variant of the subtree of the node labeled M_{n-2-I};
3. An empty sequence $v = ()$ corresponds to the selection of a leaf node labeled M_0.

Theorem 5. *There is a bijection between the set of Motzkin n-paths and the set of variants of the AND/OR tree for M_n.*

Proof. The total number of Motzkin n-paths is equal to M_n. The total number of variants of the AND/OR tree for M_n presented in Figure 14 is also equal to M_n. Therefore, it is possible to associate each such lattice path with one specific variant of the AND/OR tree for M_n. A bijection between the set of Motzkin n-paths and the set of variants of the AND/OR tree for M_n is defined by the following rules:

- each selected left child of the OR node labeled M_n determines the addition of one North-East-step to the Motzkin $(n-1)$-path obtained by the subtree of the node labeled M_{n-1} and consisting of k steps: the resulting Motzkin n-path is $(s_1, \ldots, s_k, NE)$;
- each selected child of the OR node labeled M_n determines the addition of two East-steps and North-steps to the Motzkin $(n-2)$-path that merges the Motzkin I-path obtained by the subtree of the node labeled M_I and consisting of k_1 steps and the Motzkin $(n-2-I)$-path obtained by the subtree of the node labeled M_{n-2-I} and consisting of k_2 steps: the resulting Motzkin n-path is $\left(s_1, \ldots, s_{k_1}, E, E, s_{k_1+3}, \ldots, s_{k_1+2+k_2}, N, N\right)$;
- the subtree of the node labeled M_I (the left subtree) determines the Motzkin I-path of the form $\left(s_1, \ldots, s_{k_1}\right)$;
- the subtree of the node labeled M_{n-2-I} (the right subtree) determines the Motzkin $(n-2-I)$-path of the form $\left(s_{k_1+3}, \ldots, s_{k_1+2+k_2}\right)$;
- each leaf node labeled M_1 determines the lattice path from $(0,0)$ to $(1,1)$ that consists of one North-East-step: the resulting lattice path is $(s_1) = (NE)$;
- each leaf node labeled M_0 determines the empty lattice path $()$.

□

The algorithms that implement the developed bijection rules have polynomial time complexity $O(n^2)$, since they make n recursive calls where each such recursive call requires one pass to fill a sequence of maximum $2n$ elements. An example of applying these bijection rules is presented in Table 5.

Table 5. Ranking the set of Motzkin paths beginning at $(0,0)$ and ending at $(3,3)$.

Lattice path	Variant of AND/OR tree	Rank
NE, NE, NE	$(-1, (-1, (-1, ())))$	0
E, E, N, N, NE	$(-1, (0, (), ()))$	1
E, E, NE, N, N	$(0, (), (-1, ()))$	2
NE, E, E, N, N	$(1, (-1, ()), ())$	3

3.5.3. Ranking and Unranking Algorithms

Applying the general approach described in [17], we develop algorithms for ranking (Algorithm 9) and unranking (Algorithm 10) the variants of the AND/OR tree for M_n.

The developed algorithms have the following the computational complexity:

- Algorithm 9 has at most n recursive calls where $I = -1$ (each such recursive call requires one assignment) and has at most n recursive calls where $I \neq -1$ (each such recursive call requires calculations of M_n maximum $(2n-2)$ times). Applying Equation (9), the calculation of M_n has polynomial time complexity $O(n^2)$. Hence, Algorithm 9 has polynomial time complexity $O(n^4)$;
- Algorithm 10 has at most n recursive calls where $I = -1$ (each such recursive call requires calculations of M_n) and has at most n recursive calls where $I \neq -1$ (each such recursive call requires calculations of M_n maximum $2n$ times). Hence, Algorithm 10 has polynomial time complexity $O(n^4)$.

Algorithm 9: An algorithm for ranking a variant of the AND/OR tree for M_n.

```
1  Rank_M(v = (I, v_1, ...), n)
2  begin
3      if n = 0 then r := 0
4      else
5          if I = −1 then r := Rank_M(v_1, n − 1)
6          else
7              sum := M(n − 1) + Σ_{i=0}^{I−1} M_i M_{n−2−i}
8              w_1 := M(I)
9              l_1 := Rank_M(v_1, I)
10             l_2 := Rank_M(v_2, n − 2 − I)
11             r := sum + l_1 + w_1 l_2
12         end
13     end
14     return r
15 end
```

Algorithm 10: An algorithm for unranking a variant of the AND/OR tree for M_n.

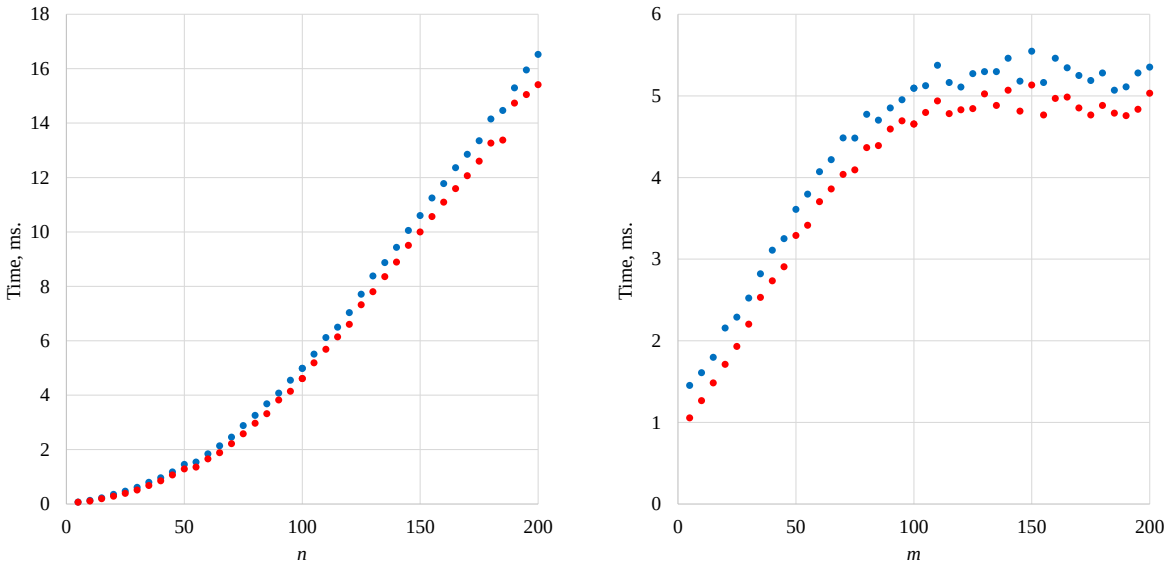
```
1  Unrank_M(r, n)
2  begin
3      if n = 0 then v := ()
4      else
5          if r < M_{n−1} then
6              I := −1
7              v_1 := Unrank_M(r, n − 1)
8              v := (I, v_1)
9          end
10         else
11             r := r − M_{n−1}
12             sum := 0
13             for i := 0 to n − 1 do
14                 s := M_i M_{n−2−i}
15                 if sum + s > r then
16                     r := r − sum
17                     I := i
18                     break
19                 end
20                 sum := sum + s
21             end
22             w_1 := M(I)
23             l_1 := r mod w_1
24             l_2 := ⌊r / w_1⌋
25             v_1 := Unrank_M(l_1, I)
26             v_2 := Unrank_M(l_2, n − 2 − I)
27             v := (I, v_1, v_2)
28         end
29     end
30     return v
31 end
```

Table 5 presents an example of ranking the combinatorial set of all Motzkin paths beginning at $(0,0)$ and ending at $(3,3)$.

3.6. Computational Experiments

We have also performed a computational experiment aimed at testing the obtained computational complexity of the developed ranking and unranking algorithms. For this purpose, we implemented these algorithms in the computer algebra system Maxima [25] on a laptop (Intel i7-9750H, 2.6 GHz, Windows 10, 64 bit). We then measured the elapsed time of the algorithms for ranking and unranking variants of the AND/OR trees for L_n^m, C_n, D_n^m, S_n and M_n with different values of the parameters n and m. All calculations were made for the variants of the AND/OR trees with the maximum rank (because this is usually the hardest case to compute due to the need to traverse the entire AND/OR tree structure).

For the maximum rank $r := L_n^m - 1$, Algorithms 1 and 2 have polynomial time complexity $O(nm)$ for $m < n$ and $O(n^2)$ for $n < m$. Figure 16a presents the average time for unranking and ranking such variants of the AND/OR tree for L_n^m where $m = 100$ and n is in the range of 5 to 200 with step 5. This figure confirms polynomial time complexity $O(n^2)$ for $n < 100$ and linear time complexity $O(n)$ for $n > 100$ when $m = 100$. Figure 16b presents the average time for unranking and ranking variants of the AND/OR tree for L_n^m where $n = 100$ and m is in the range of 5 to 200 with step 5. This figure confirms linear time complexity $O(m)$ for $m < 100$ and constant time complexity $O(1)$ for $m > 100$ when $n = 100$. For other fixed values of m (or n), the general form of the dependence on n (or m) does not change.



(**a**) (**b**)

Figure 16. Average time for unranking (**blue**) and ranking (**red**) variants of the AND/OR tree for L_n^m: (**a**) dependence on n when $m = 100$; and (**b**) dependence on m when $n = 100$.

For the maximum rank $r := C_n - 1$, Algorithms 3 and 4 have polynomial time complexity $O(n^3)$. Figure 17 presents the average time for unranking and ranking such variants of the AND/OR tree for C_n where n is in the range of 1 to 50 with step 1. This figure confirms polynomial time complexity $O(n^3)$.

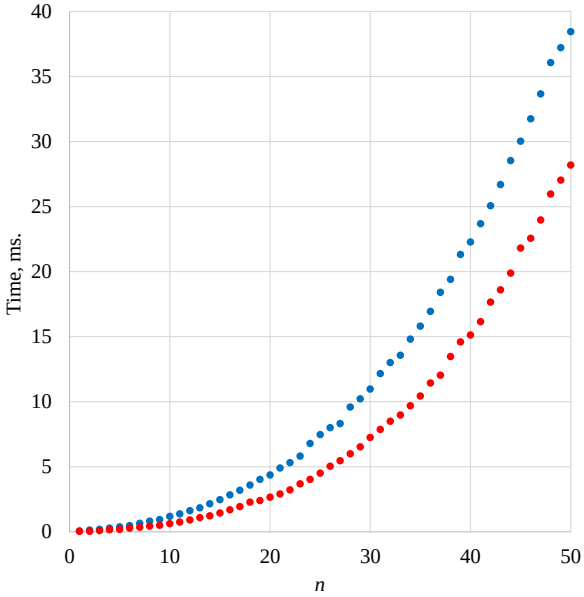
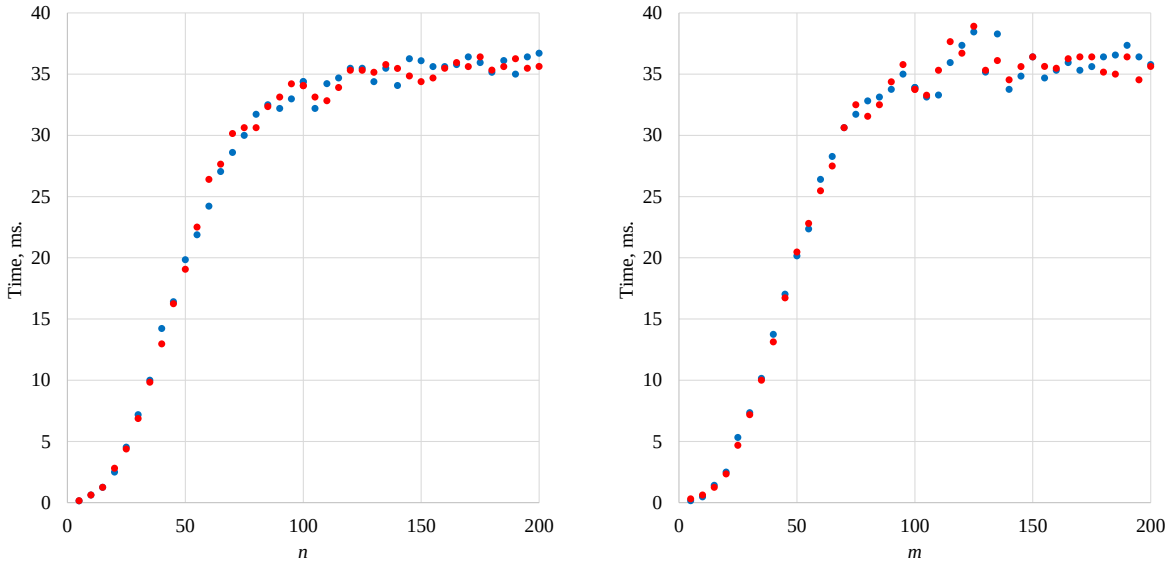


Figure 17. Average time for unranking (**blue**) and ranking (**red**) variants of the AND/OR tree for C_n.

For the maximum rank $r := D_n^m - 1$, Algorithms 5 and 6 have the following polynomial time complexity (due to all special cases of the calculation of Equation (5)): $O(m^3)$ for $2m < n$, $O(m^2(n-m))$ for $m < n < 2m$, $O(n^2(m-n))$ for $n < m < 2n$, and $O(n^3)$ for $2n < m$. Figure 18a presents the average time for unranking and ranking such variants of the AND/OR tree for D_n^m where $m = 50$ and n is in the range of 5 to 200 with step 5. Figure 18b presents the average time for unranking and ranking variants of the AND/OR tree for D_n^m where $n = 50$ and m is in the range of 5 to 200 with step 5. These figures confirm the derived polynomial time complexity for all special cases. For other fixed values of m (or n), the general form of the dependence on n (or m) does not change.



(**a**) (**b**)

Figure 18. Average time for unranking (**blue**) and ranking (**red**) variants of the AND/OR tree for D_n^m: (**a**) dependence on n when $m = 50$; and (**b**) dependence on m when $n = 50$.

For the maximum rank $r := S_n - 1$, Algorithms 7 and 8 have polynomial time complexity $O(n^4)$. Figure 19 presents the average time for unranking and ranking such variants of the AND/OR tree for S_n where n is in the range of 1 to 50 with step 1. This figure confirms polynomial time complexity $O(n^4)$.

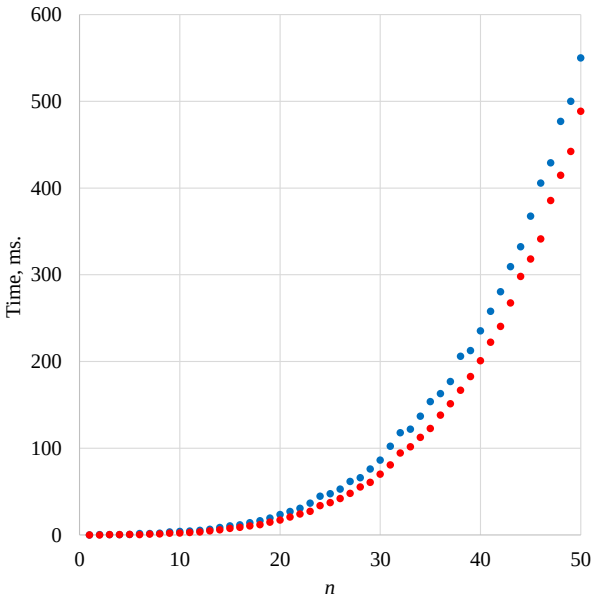


Figure 19. Average time for unranking (**blue**) and ranking (**red**) variants of the AND/OR tree for S_n.

For the maximum rank $r := M_n - 1$, Algorithms 9 and 10 have polynomial time complexity $O(n^4)$. Figure 20 presents the average time for unranking and ranking such variants of the AND/OR tree for M_n where n is in the range of 1 to 50 with step 1. This figure confirms polynomial time complexity $O(n^4)$.

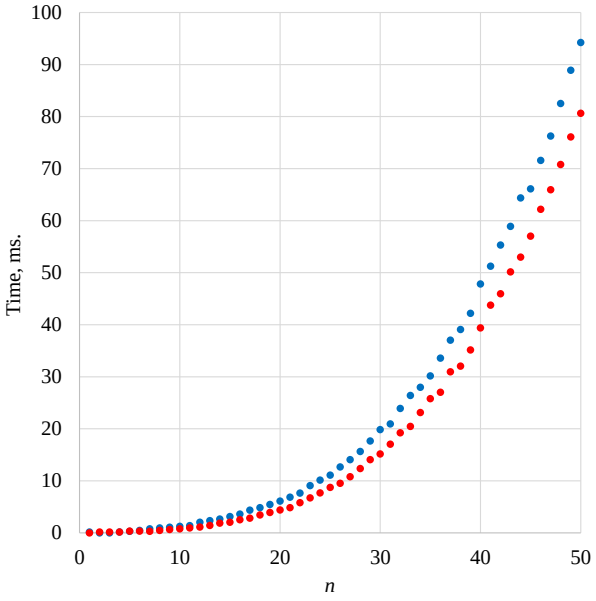


Figure 20. Average time for unranking (**blue**) and ranking (**red**) variants of the AND/OR tree for M_n.

4. Discussion

In this article, we have demonstrated the possibilities of using the method based on AND/OR trees to obtain combinatorial generation algorithms for combinatorial sets of several well-known lattice paths. In particular, the following lattice paths were considered: North-East lattice paths, Dyck paths, Delannoy paths, Schroder paths, and Motzkin paths. For each of these combinatorial sets of lattice paths, there is an expression of its cardinality function that belongs to the algebra $\{\mathbb{N}, +, \times, R\}$. This fact made it possible to construct AND/OR tree structures for these combinatorial sets, in which the number of their variants is equal to the number of objects in the combinatorial set. To associate each considered lattice path with one specific variant of the corresponding AND/OR tree structure, the bijection rules were derived.

Applying the constructed AND/OR tree structures, we have developed algorithms for ranking and unranking their variants. All developed algorithms have a recursive form since the applied AND/OR tree structures were built by using recurrence relations. The developed algorithms, together with the obtained bijection rules, make it possible to encode a given lattice path as a single number (by applying ranking algorithms). For example, this reduces the amount of data stored if it is necessary to store information about a large number of lattice paths. It is also possible to recover each lattice path by decoding the corresponding rank (by applying unranking algorithms). In addition, the unranking algorithms make it possible to form a sample of random lattice paths with the given properties by applying them to random rank values. If it is necessary to generate all objects with the given parameters (for example, when an exhaustive search is needed to solve an optimization problem), then we can run the unranking algorithm for all rank values. Furthermore, the exhaustive generation can be performed by sequentially traversing all variants of the considered AND/OR tree structure.

The developed combinatorial generation algorithms for lattice paths have polynomial time complexity, which is determined by the maximum number of recursive calls (the height of an AND/OR tree), the maximum number of child nodes of a given node (the width of an AND/OR tree), and the computational complexity of calculating the value of the cardinality function. Assuming algebraic operations with numbers in $O(1)$, the performed computational experiments confirmed the obtained theoretical estimation of asymptotic computational complexity for the developed ranking and unranking algorithms.

To reduce the time complexity of these algorithms, we can add a preliminary step where all the required values of the cardinality function are calculated and stored. In this case, the computational complexity of algorithms will depend only on the size of the AND/OR tree structure. However, this improvement requires additional memory space. In addition, if an AND/OR tree structure contains OR nodes where the number of children depends on the AND/OR tree parameters (for example, as AND/OR trees for C_n, S_n and M_n), then the computational complexity of ranking and unranking algorithms can be reduced as follows:

- By calculating *sum* (for example, see Line 5 in Algorithm 3 or Lines 5–14 in Algorithm 4), applying a simpler explicit formula without using the summation operator;
- By a preliminary search of the value of the selected child of the OR node (for example, the parameter I defined in Lines 6–14 in Algorithm 4), applying an approximate formula (as in [26]).

Thus, the main contribution of this article is the derivation of bijections between two sets of discrete structures (the set of lattice paths and the set of AND/OR tree variants), as well as new algorithms for their generation. The scheme used in this article to develop combinatorial generation algorithms for lattice paths can also be applied to the classes of more complex lattice paths that have additional types of steps (for example, skew Dyck paths [27] or skew Dyck paths with catastrophes [28]) or depend on more parameters (for example, lattice paths associated with the generalized Narayana numbers [29]). The main restriction is that the cardinality function for such lattice paths must belong to the required algebra $\{\mathbb{N}, +, \times, R\}$.

Funding: This research was funded by the Russian Science Foundation, grant number 22-71-10052.

Data Availability Statement: Source code can be made available on request.

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

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

References

1. Kreher, D.L.; Stinson, D.R. *Combinatorial Algorithms: Generation, Enumeration, and Search*; CRC Press: Boca Raton, FL, USA, 1999.
2. Ruskey, F. Combinatorial Generation. Available online: https://page.math.tu-berlin.de/~felsner/SemWS17-18/Ruskey-Comb-Gen.pdf (accessed on 1 May 2023).
3. Knuth, D.E. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*; Addison-Wesley Professional: Boston, MA, USA, 2011.
4. Stanley, R.P. *Enumerative Combinatorics: Volume 1*, 2nd ed.; Cambridge University Press: New York, NY, USA, 2012.
5. Wallner, M. Combinatorics of Lattice Paths and Tree-Like Structures. Ph.D. Thesis, Institute of Discrete Mathematics and Geometry, Vienna University of Technology, Vienna, Austria, 2016.
6. Humphreys, K. A history and a survey of lattice path enumeration. *J. Statist. Plann. Inference* **2010**, *140*, 2237–2254. [CrossRef]
7. Krattenthaler, C. Lattice path enumeration. In *Handbook of Enumerative Combinatorics*; Bona, M., Ed.; CRC Press: New York, NY, USA, 2015; pp. 589–678.
8. Zaks, S.; Richards, D. Generating trees and other combinatorial objects lexicographically. *SIAM J. Comput.* **1979**, *8*, 73–81. [CrossRef]
9. Bent, S.W. Ranking trees generated by rotations. *Lect. Notes Comput. Sci.* **1990**, *447*, 132–142.
10. Parque, V.; Miyashita, T. An efficient scheme for the generation of ordered trees in constant amortized time. In Proceedings of the 15th International Conference on Ubiquitous Information Management and Communication (IMCOM), Seoul, Republic of Korea, 4–6 January 2021.
11. Barcucci, E.; Bernini, A.; Pinzani, R. Exhaustive generation of positive lattice paths. In Proceedings of the 11th International Conference on Random and Exhaustive Generation of Combinatorial Structures (GASCom), Athens, Greece, 18–20 June 2018.
12. Barcucci, E.; Bernini, A.; Pinzani, R. Exhaustive generation of some lattice paths and their prefixes. *Theoret. Comput. Sci.* **2021**, *878–879*, 47–52. [CrossRef]
13. Kuo, T. From enumerating to generating: A linear time algorithm for generating 2*D* lattice paths with a given number of turns. *Algorithms* **2015**, *8*, 190–208. [CrossRef]
14. The Combinatorial Object Server. Available online: http://combos.org/ (accessed on 1 May 2023).
15. 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]
16. Flajolet, P.; Zimmerman, P.; Cutsem, B. A calculus for the random generation of combinatorial structures. *Theoret. Comput. Sci.* **1994**, *132*, 1–35. [CrossRef]
17. Shablya, Y.; Kruchinin, D.; Kruchinin, V. Method for developing combinatorial generation algorithms based on AND/OR trees and its application. *Mathematics* **2020**, *8*, 962. [CrossRef]
18. Shablya, Y.; Tokareva, A. Development of combinatorial generation algorithms for some lattice paths using the method based on AND/OR trees. In Proceedings of the 5th Mediterranean International Conference of Pure & Applied Mathematics and Related Areas (MICOPAM), Antalya, Turkey, 27–30 October 2022.
19. Mohanty, G. *Lattice Path Counting and Applications*; Academic Press: New York, NY, USA, 1979.
20. The On-Line Encyclopedia of Integer Sequences. Available online: https://oeis.org/ (accessed on 1 May 2023).
21. Mansour, T. Statistics on Dyck paths. *J. Integer Seq.* **2005**, *9*, 06.1.5.
22. Banderier, C.; Schwer, S. Why Delannoy numbers? *J. Statist. Plann. Inference* **2005**, *135*, 40–54. [CrossRef]
23. Shapiro, L.W.; Sulanke, R.A. Bijections for the Schroder numbers. *Math. Mag.* **2000**, *73*, 369–376. [CrossRef]
24. Oste, R.; Van der Jeugt, J. Motzkin paths, Motzkin polynomials and recurrence relations. *Electron. J. Combin.* **2015**, *22*, P2.8. [CrossRef] [PubMed]
25. Maxima, a Computer Algebra System. Available online: https://maxima.sourceforge.io/ (accessed on 1 May 2023).
26. Kruchinin, V.; Shablya, Y.; Kruchinin, D. Unranking small combinations of a large set in co-lexicographic order. *Algorithms* **2022**, *15*, 36. [CrossRef]
27. Deutsch, E.; Munarini, E.; Rinaldi, R. Skew Dyck paths. *J. Statist. Plann. Inference* **2010**, *140*, 2191–2203. [CrossRef]

28. Prodinger, H. Skew Dyck paths with catastrophes. *Discrete Math. Lett.* **2022**, *10*, 9–13.
29. Kruchinin, D.; Kruchinin, V.; Shablya, Y. On some properties of generalized Narayana numbers. *Quaest. Math.* **2022**, *45*, 1949–1963. [CrossRef]