MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 32, NO. 4, 2025
journal homepage: www.mais-journal.ru

Combinatorial Generation Algorithms Based on AND/OR Tree Structures for a Class of Algebraic Generating Functions

Y. V. Shablya[1]

DOI: 10.18255/1818-1015-2025-4-360-383

[1]Tomsk State University of Control Systems and Radioelectronics, Tomsk, Russia

MSC2020: 05A15, 68R05
Research article
Full text in Russian

Received November 1, 2025
Revised November 24, 2025
Accepted November 26, 2025

This paper proposes a systematic approach to developing combinatorial generation algorithms for sets of discrete structures whose cardinality is determined by the coefficients of algebraic generating functions and their powers. The study is based on the relationship between operations on generating functions and combinatorial sets. It uses the mathematical apparatus of AND/OR trees as a foundation, which allows combining combinatorial generation algorithms for simple substructures into complex combinatorial objects. The main theoretical result of the work is the derivation of new efficient recurrence formulas for calculating the values of the coefficients of algebraic generating functions and their powers with polynomial computational complexity $O((n_1+\ldots+n_m+m)\cdot n^2)$ for time and $O(n^2)$ for memory. Based on proven theorems on recurrence formulas, the proposed approach enables the construction of algorithms with polynomial computational complexity estimates, making them applicable to solving practical problems in applied discrete mathematics and theoretical computer science. Moreover, the use of coefficients of generating function powers expands the generation capabilities, since it allows us to construct not only objects of the original combinatorial set associated with the generating function, but also tuples of such objects. Validation of the proposed approach is demonstrated using examples of deriving recurrence formulas and generation algorithms based on them for classical numerical sequences, such as the Fibonacci, Pell, Catalan, Motzkin, and Schroder numbers. The proposed approach opens up new possibilities for solving problems of optimization, modeling, and coding complex discrete structures, for example, in fields such as bioinformatics and cryptography.

Keywords: discrete structure; combinatorial generation; algebraic generating function; functional equation; recurrence formula; AND/OR tree

INFORMATION ABOUT THE AUTHORS

Shablya, Yuriy V. (corresponding author) | ORCID iD: 0000-0002-9695-7493. E-mail: shablya-yv@mail.ru
PhD, Associate Professor

For citation: Y. V. Shablya, "Combinatorial generation algorithms based on AND/OR tree structures for a class of algebraic generating functions", *Modeling and Analysis of Information Systems*, vol. 32, no. 4, pp. 360–383, 2025. DOI: 10.18255/1818-1015-2025-4-360-383.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 32, № 4, 2025
сайт журнала: www.mais-journal.ru

DISCRETE MATHEMATICS IN RELATION TO COMPUTER SCIENCE

Алгоритмы комбинаторной генерации на основе структур деревьев И/ИЛИ для класса алгебраических производящих функций

Ю. В. Шабля[1]

DOI: 10.18255/1818-1015-2025-4-360-383

[1]Томский государственный университет систем управления и радиоэлектроники, Томск, Россия

УДК 519.111.3
Научная статья
Полный текст на русском языке

Получена 1 ноября 2025 г.
После доработки 24 ноября 2025 г.
Принята к публикации 26 ноября 2025 г.

В данной статье предложен систематический подход к разработке алгоритмов комбинаторной генерации для множеств дискретных структур, мощность которых задается коэффициентами алгебраических производящих функций и их степеней. Исследование базируется на наличии связи между операциями над производящими функциями и комбинаторными множествами. В качестве основы использован математический аппарат деревьев И/ИЛИ, который позволяет комбинировать алгоритмы комбинаторной генерации для простых подструктур в сложные комбинаторные объекты. При этом основным теоретическим результатом работы является вывод новых эффективных рекуррентных формул для вычисления значений коэффициентов алгебраических производящих функций и их степеней с полиномиальной вычислительной сложностью $O((n_1 + \ldots + n_m + m) \cdot n^2)$ по времени и $O(n^2)$ по памяти. На основе доказанных теорем о рекуррентных формулах, предложенный подход позволяет строить алгоритмы с полиномиальной оценкой вычислительной сложности, что делает их применимыми для решения практических задач в области прикладной дискретной математики и теоретической информатики. Кроме того, использование коэффициентов степеней производящих функций расширяет возможности генерации, так как это позволяет строить не только объекты исходного комбинаторного множества, связанного с производящей функцией, но и кортежи таких объектов. Апробация предложенного подхода показана на примерах получения рекуррентных формул и алгоритмов генерации на их основе для классических числовых последовательностей, таких как числа Фибоначчи, Пелля, Каталана, Моцкина и Шредера. Предложенный подход открывает новые возможности для решения задач оптимизации, моделирования и кодирования сложных дискретных структур, например, в таких областях как биоинформатика и криптография.

Ключевые слова: дискретная структура; комбинаторная генерация; алгебраическая производящая функция; функциональное уравнение; рекуррентная формула; дерево И/ИЛИ

ИНФОРМАЦИЯ ОБ АВТОРАХ

Шабля, Юрий Васильевич (автор для корреспонденции) | ORCID iD: 0000-0002-9695-7493. E-mail: shablya-yv@mail.ru
Канд. тех. наук, доцент

Для цитирования: Y. V. Shablya, “Combinatorial generation algorithms based on AND/OR tree structures for a class of algebraic generating functions”, *Modeling and Analysis of Information Systems*, vol. 32, no. 4, pp. 360–383, 2025.
DOI: 10.18255/1818-1015-2025-4-360-383.

Введение

Развитие информационных технологий и цифровизация всех сфер жизни человека приводит к экспоненциальному росту производимой, обрабатываемой и хранимой информации [1]. В свою очередь, это приводит к потребности в развитии существующего и разработке нового математического, алгоритмического и программного обеспечения для организации соответствующих информационных процессов. Комбинаторные алгоритмы играют важную роль в области теоретических основ информатики, так как являются эффективным инструментом при работе с дискретными структурами [2, 3]. Одной из общих задач, связанных с обработкой комбинаторных множеств, является построение алгоритмов комбинаторной генерации [4]. Алгоритмы комбинаторной генерации позволяют перенумеровать элементы комбинаторного множества (за счет применения алгоритма ранжирования), а также в обратную сторону реализовать их генерацию как в единичном варианте (алгоритм генерации по рангу или случайная генерация), так и исчерпывающим перебором (последовательная генерация). При этом каждый тип алгоритмов комбинаторной генерации специализируется на решении определенного рода практических задач.

В частности, алгоритмы последовательной генерации позволяют решить задачу полного перебора всех возможных структур при заданных значениях их параметров. Следовательно, такие алгоритмы можно применить при решении задач оптимизации дискретных структур, то есть когда происходит их перебор и по итогу отбирается структура с наилучшими характеристиками. Также алгоритмы комбинаторной генерации можно применить при решении задач моделирования сложных дискретных структур (например, в области биоинформатики [5—7] и криптографии [8, 9]). В таком случае сразу создаются структуры, удовлетворяющие заданным ограничениям, то есть отпадает потребность проверки соблюдения требуемых условий у произвольно созданной структуры.

В то же время комбинирование алгоритмов генерации по рангу и ранжирования позволяет задать биективное отображение между комбинаторным множеством и ограниченным подмножеством неотрицательных целых чисел. Алгоритм ранжирования позволяет представить заданную дискретную структуру в форме одного целого числа (ранг), то есть выполняется кодирование исходного информационного объекта. Алгоритм генерации по рангу позволяет восстановить исходную дискретную структуру из значения соответствующего ей ранга, то есть выполняется декодирование информационного объекта. Следовательно, на основе алгоритмов комбинаторной генерации можно организовать процесс кодирования информационных объектов, представляющих собой сложные дискретные структуры [10, 11].

Рассмотрим произвольный класс дискретных структур A. Обозначим через A_n подмножество объектов с заданным значением n некоторого количественного параметра (например, размер объекта), то есть $A_n \subseteq A$ и

$$\bigcup_{n \geqslant 0} A_n = A,$$

причем $A_i \cap A_j = \emptyset$ для $i \neq j$. Если записать числовую последовательность значений мощности комбинаторных множеств A_n для возрастающей последовательности значений параметра n, где $n \geqslant 0$, то ей можно сопоставить производящую функцию вида:

$$|A_0| + |A_1|\, x + |A_2|\, x^2 + |A_3|\, x^3 + \ldots = \sum_{n \geqslant 0} |A_n|\, x^n = \sum_{n \geqslant 0} a_n\, x^n = A(x).$$

Таким образом, производящая функция $A(x)$ представляет собой дополнительную характеристику исследуемого множества дискретных структур. Кроме того, изучение свойств таких производящих функций позволяет строить алгоритмы комбинаторной генерации для связанных с ними комбинаторных множеств. Данная статья посвящена вопросам разработки алгоритмов комбинаторной генерации для множеств, мощность которых определяется коэффициентами алгебраических производящих функций.

1. Метод построения алгоритмов комбинаторной генерации

В качестве теоретической базы для построения новых алгоритмов комбинаторной генерации, выбран метод на основе структур деревьев И/ИЛИ [12—14]. Такие деревья содержат узлы двух типов: ИЛИ-узлы (соответствуют операции сложения) и И-узлы (соответствуют операции умножения). Основная идея метода на основе деревьев И/ИЛИ заключается в представлении комбинаторных объектов в виде множества вариантов структуры дерева И/ИЛИ. Вариантом дерева И/ИЛИ называется структура, полученная выбором одного узла-потомка для каждого ИЛИ-узла и удаления всех невыбранных узлов-потомков ИЛИ-узлов, включая их поддеревья. При этом для построения структуры дерева И/ИЛИ требуется наличие формулы для вычисления мощности комбинаторного множества, которая должна состоять только из разрешенных операций (сложение, умножение и композиция, а также использование неотрицательных целых чисел). В результате, используя общие алгоритмы генерации по рангу вариантов дерева И/ИЛИ (алгоритмы 1 и 2), получаем соответствующие алгоритмы для множества исследуемых комбинаторных объектов.

Algorithm 1. Algorithm for unranking an AND/OR tree variant for the subtree of an OR-node

Алгоритм 1. Алгоритм генерации по рангу варианта дерева И/ИЛИ для поддерева ИЛИ-узла

```
1  Unrank_OR(r, node = (child_1, ..., child_n))
2  begin
3      sum := 0
4      for i := 1 to n do                          // Поиск выбранного узла child_I
5          if sum + w(child_i) > r then            // Если найден выбранный узел child_I
6              r := r − sum
7              I := i
8              object_I := Unrank(r, child_I)      // Генерация варианта поддерева узла child_I
9              break
10         end
11         sum := sum + w(child_i)
12     end
13     object := (object_I)                        // Выбор одного узла-потомка
14     return object
15 end
```

Algorithm 2. Algorithm for unranking an AND/OR tree variant for the subtree of an AND-node

Алгоритм 2. Алгоритм генерации по рангу варианта дерева И/ИЛИ для поддерева И-узла

```
1  Unrank_AND(r, node = (child_1, ..., child_n))
2  begin
3      for i := 1 to n do                          // Для каждого узла-потомка child_i
4          r_i := r  mod w(child_i)                // Ранг варианта поддерева узла child_i
5          r := ⌊ r / w(child_i) ⌋
6          object_i := Unrank(r_i, child_i)        // Генерация варианта поддерева узла child_i
7      end
8      object := (object_1, ..., object_n)        // Выбор всех узлов-потомков
9      return object
10 end
```

Алгоритмы 1 и 2 принимают на вход узел дерева И/ИЛИ, который представляется как упорядоченный набор его узлов-потомков $(child_1, \ldots, child_n)$. При этом функция $w(child)$ определяет количество вариантов в поддереве узла *child*.

2. Связь производящих функций с комбинаторными множествами и алгоритмами комбинаторной генерации на основе структур деревьев И/ИЛИ

Операции над производящими функциями влияют на связанные с ними множества информационных объектов. Если рассмотреть комбинаторные множества A_n и B_n, которые содержат объекты размера n (или любого другого количественного параметра) в количестве $|A_n|$ и $|B_n|$ соответственно, то на основе последовательностей значений $|A_n|$ и $|B_n|$ при разных n можно задать следующие производящие функции с коэффициентами a_n и b_n:

$$\sum_{n\geqslant 0} |A_n|\, x^n = \sum_{n\geqslant 0} a_n\, x^n = A(x),$$

$$\sum_{n\geqslant 0} |B_n|\, x^n = \sum_{n\geqslant 0} b_n\, x^n = B(x).$$

Выполнение операций над производящими функциями $A(x)$ и $B(x)$ формирует новое комбинаторное множество, связанное с элементами комбинаторных множеств A_n и B_n.

2.1. Сложение производящих функций

Если применить операцию сложения, то получаем следующую производящую функцию:

$$A(x) + B(x) = H(x) = \sum_{n\geqslant 0} h_n\, x^n = \sum_{n\geqslant 0} (a_n + b_n)\, x^n.$$

В таком случае производящая функция $H(x)$ задает последовательность значений мощности комбинаторных множеств H_n $(n \geqslant 0)$, элементы которых формируются путем дизъюнктивного объединения комбинаторных множеств A_n и B_n, то есть

$$H_n = A_n \sqcup B_n.$$

Таким образом, комбинаторное множество H_n содержит объекты размера n, каждый из которых соответствует объекту, принадлежащему либо A_n, либо B_n. Применяя метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ, по формуле для вычисления значений h_n строится соответствующая композиция структур деревьев И/ИЛИ (рисунок 1) и алгоритм генерации по рангу для комбинаторного множества H_n (алгоритм 3).

Заметим, что полученный алгоритм `Unrank_Hsum`(r, n) базируется на применении алгоритмов генерации по рангу для комбинаторного множества A_n (алгоритм `Unrank_A`(r, n)) и комбинаторного множеств B_n (алгоритм `Unrank_B`(r, n)). При этом алгоритмы `Unrank_A`(r, n) и `Unrank_B`(r, n) могут быть произвольной природы происхождения, что позволяет комбинировать алгоритмы комбинаторной генерации с наилучшими показателями эффективности.

2.2. Умножение производящих функций

Если применить операцию умножения, то получаем следующую производящую функцию:

$$A(x) \cdot B(x) = H(x) = \sum_{n\geqslant 0} h_n\, x^n = \sum_{n\geqslant 0} \left(\sum_{i=0}^{n} a_i \cdot b_{n-i} \right) x^n.$$

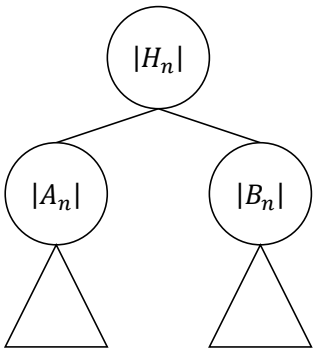

Fig. 1. AND/OR tree structure for H_n when $H(x) = A(x) + B(x)$

Рис. 1. Структура дерева И/ИЛИ для H_n при $H(x) = A(x) + B(x)$

Algorithm 3. Algorithm for unranking H_n when $H(x) = A(x) + B(x)$

Алгоритм 3. Алгоритм генерации по рангу для H_n при $H(x) = A(x) + B(x)$

```
1 Unrank_Hsum(r, n)
2 begin
3     if n = 0 then object := ε                                  // Пустой объект
4     else
5         if r < |A_n| then object := Unrank_A(r, n)             // Выбор узла-потомка с меткой |A_n|
6         else object := Unrank_B(r − |A_n|, n)                  // Выбор узла-потомка с меткой |B_n|
7     end
8     return object
9 end
```

В таком случае производящая функция $H(x)$ задает последовательность значений мощности комбинаторных множеств H_n ($n \geqslant 0$), элементы которых формируются путем декартова произведения комбинаторных множеств A_i и B_{n-i} для всех $i \in \{0, 1, \ldots, n\}$, то есть

$$H_n = (A_0 \times B_n) \cup (A_1 \times B_{n-1}) \cup \ldots \cup (A_{n-1} \times B_1) \cup (A_n \times B_0) = \bigcup_{i=0}^{n} A_i \times B_{n-i}.$$

Таким образом, комбинаторное множество H_n содержит объекты, представляющие собой упорядоченные пары вида $(object_a, object_b)$, где $object_a \in A_i$ и $object_b \in B_{n-i}$ ($i \in \{0, 1, \ldots, n\}$), так что суммарный размер объектов a и b равен n. Применяя метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ, по формуле для вычисления значений h_n строится соответствующая композиция структур деревьев И/ИЛИ (рисунок 2) и алгоритм генерации по рангу для комбинаторного множества H_n (алгоритм 4).

2.3. Возведение производящей функции в k-ю степень

Если применить операцию возведения в k-ю степень, то получаем следующую производящую функцию:

$$A(x)^k = H(x) = \sum_{n \geqslant 0} h_{n,k}\, x^n.$$

Из рекуррентной формулы

$$A(x)^k = A(x) \cdot A(x)^{k-1}, \quad A(x)^0 = 1,$$

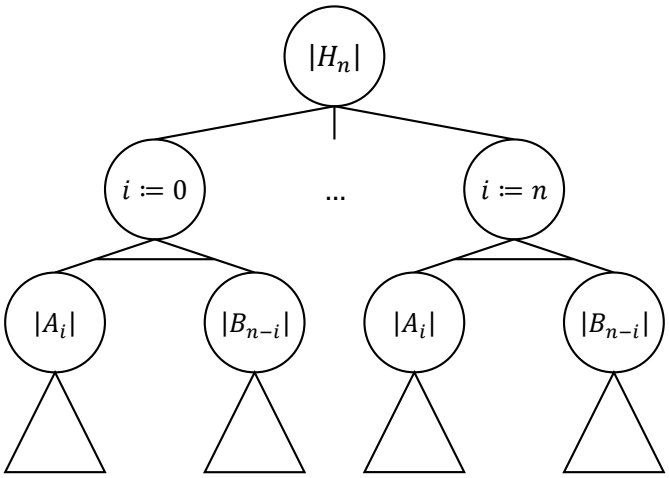

Fig. 2. AND/OR tree structure for H_n when $H(x) = A(x) \cdot B(x)$

Рис. 2. Структура дерева И/ИЛИ для H_n при $H(x) = A(x) \cdot B(x)$

Algorithm 4. Algorithm for unranking H_n when $H(x) = A(x) \cdot B(x)$

Алгоритм 4. Алгоритм генерации по рангу для H_n при $H(x) = A(x) \cdot B(x)$

```
1  Unrank_Hmul(r, n)
2  begin
3      if n = 0 then object := ε                                  // Пустой объект
4      else
5          sum := 0
6          for i := 0 to n do                          // Поиск выбранного узла с меткой i := I
7              if sum + |A_i| · |B_{n−i}| > r then     // Если найден выбранный узел с меткой i := I
8                  r := r − sum
9                  I := i
10                 break
11             end
12             sum := sum + |A_i| · |B_{n−i}|
13         end
14         r_1 := r  mod |A_I|                    // Ранг варианта поддерева узла с меткой |A_I|
15         r_2 := ⌊ r / |A_I| ⌋                   // Ранг варианта поддерева узла с меткой |B_{n−I}|
16         object_a := Unrank_A(r_1, I)
17         object_b := Unrank_B(r_2, n − I)
18         object := (object_a, object_b)
19     end
20     return object
21 end
```

следует, что коэффициенты производящей функции $H(x)$ для $n, k \geqslant 0$ определяются по формуле

$$h_{n,k} = \begin{cases} 1, & n = k = 0; \\ a_n, & n \geqslant 0 \text{ и } k = 1; \\ \sum\limits_{i=0}^{n} a_i \cdot h_{n-i,k-1}, & n \geqslant 0 \text{ и } k > 1; \\ 0, & \text{иначе.} \end{cases}$$

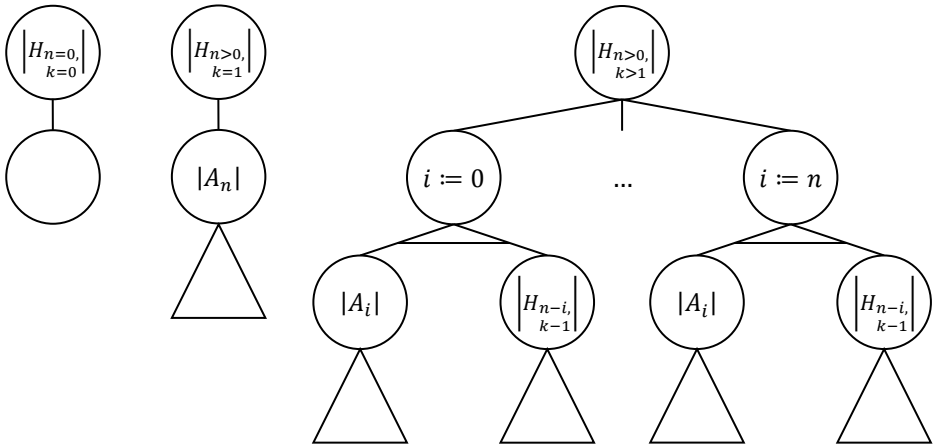

Fig. 3. AND/OR tree structures for $H_{n,k}$ when $H(x) = A(x)^k$

Рис. 3. Структуры дерева *И/ИЛИ* для $H_{n,k}$ при $H(x) = A(x)^k$

В таком случае производящая функция $H(x)$ задает последовательность значений мощности комбинаторных множеств $H_{n,k}$ $(n, k \geqslant 0)$, элементы которых формируются путем декартова произведения k различных комбинаторных множеств A_i $(i \in \{0, 1, \ldots, n\})$, так что суммарное значение индексов равно n, то есть

$$H_{n,k} = \bigcup_{i_1+\ldots+i_k=n} A_{i_1} \times \ldots \times A_{i_k}.$$

Таким образом, комбинаторное множество $H_{n,k}$ содержит объекты, представляющие собой кортежи вида $(object_1, \ldots, object_k)$, где $object_j \in A_{i_j}$ $(j \in \{1, \ldots, k\})$, так что суммарный размер объектов $object_j$ равен n. Применяя метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ, по формуле для вычисления значений $h_{n,k}$ строится соответствующая рекурсивная композиция структур деревьев И/ИЛИ (рисунок 3) и рекурсивный алгоритм генерации по рангу для комбинаторного множества $H_{n,k}$ (алгоритм 5).

3. Алгоритмы комбинаторной генерации для класса алгебраических производящих функций

Приведенные в разделе 2 примеры показывают, как выполнение операций над производящими функциями влияет на связанные с ними комбинаторные множества. Таким образом, декомпозиция производящей функции задает способ комбинирования алгоритмов комбинаторной генерации для простых структур, из которых формируется исследуемая сложная структура. В свою очередь, это расширяет возможности разработки новых алгоритмов комбинаторной генерации, поскольку определяет дополнительный путь к решению данной задачи. В частности, на идее декомпозиции производящей функции основан общий метод построения алгоритмов комбинаторной генерации, представленный в цикле работ [15—17]. Однако данный метод не применим к производящим функциям, которые задаются функциональными уравнениями. Поэтому актуальной является задача разработки метода построения алгоритмов комбинаторной генерации на основе производящих функций, заданных функциональными уравнениями.

Рассмотрим класс алгебраических производящих функций [18].

Определение 1. Производящая функция $A(x)$ называется алгебраической, если существуют многочлены вида $P_i(x) = \sum_{j=0}^{n_i} p_{i,j}\, x^j$, где $i \in \{0, 1, \ldots, m\}$, которые не все равны нулю и для которых выполняется

$$P_0(x) + P_1(x) \cdot A(x) + P_2(x) \cdot A(x)^2 + \ldots + P_m(x) \cdot A(x)^m = 0. \quad (1)$$

Algorithm 5. Algorithm for unranking $H_{n,k}$ when $H(x) = A(x)^k$

Алгоритм 5. Алгоритм генерации по рангу для $H_{n,k}$ при $H(x) = A(x)^k$

```
1  Unrank_Hpow(r, n, k)
2  begin
3      if n = 0 and k = 0 then object := ε                          // Пустой объект
4      else if k = 1 then object := Unrank_A(r, n)
5      else
6          sum := 0
7          for i := 0 to n do                          // Поиск выбранного узла с меткой i := I
8              if sum + |A_i| · |H_{n−i,k−1}| > r then // Если найден выбранный узел с меткой i := I
9                  r := r − sum
10                 I := i
11                 break
12             end
13             sum := sum + |A_i| · |H_{n−i,k−1}|
14         end
15         r_1 := r  mod |A_I|                    // Ранг варианта поддерева узла с меткой |A_I|
16         r_2 := ⌊ r / |A_I| ⌋                  // Ранг варианта поддерева узла с меткой |H_{n−I,k−1}|
17         object_a := Unrank_A(r_1, I)
18         object_h := Unrank_Hpow(r_2, n − I, k − 1)
19         object := concat((object_a), object_h)                   // Конкатенация кортежей
20     end
21     return object
22 end
```

Если $p_{1,0} = -1$, то функциональное уравнение (1) может быть представлено в следующем виде:

$$P_0(x) + P_{1^*}(x) \cdot A(x) + P_2(x) \cdot A(x)^2 + \ldots + P_m(x) \cdot A(x)^m = A(x),$$

где $P_{1^*}(x) = P_1(x) + 1$, то есть

$$p_{1^*,j} = \begin{cases} 0, & j = 0; \\ p_{1,j}, & j > 0. \end{cases}$$

Теорема 1. *Для алгебраической производящей функции*

$$A(x) = \sum_{n>0} a_n x^n,$$

удовлетворяющей функциональному уравнению

$$A(x) = \sum_{i=0}^{m} P_i(x) \cdot A(x)^i, \tag{2}$$

где $P_i(x) = \sum\limits_{j=0}^{n_i} p_{i,j} x^j$ *и* $p_{1,0} = 0$, *справедлива следующая рекуррентная формула:*

$$a_n = \begin{cases} 0, & n \leqslant 0; \\ \sum\limits_{i=0}^{m} \left(\sum\limits_{\substack{j+j_1+\ldots j_i=n; \\ j\geqslant 0;\ j_1,\ldots,j_i\geqslant 1}} p_{i,j} \cdot a_{j_1} \cdot \ldots \cdot a_{j_i} \right), & n > 0. \end{cases} \tag{3}$$

Доказательство. Условие $p_{1,0} = 0$ позволяет выполнить преобразование функционального уравнения (2) к виду функционального уравнения (1), что подтверждает соответствие $A(x)$ классу алгебраических производящих функций.

По условию задано, что у производящей функции $A(x)$ отсутствует свободный коэффициент, то есть $a_0 = 0$ и $A(0) = 0$. Также по условию у производящей функции $A(x)$ отсутствует переменная x с отрицательным значением показателя степени, поэтому $a_n = 0$ для $n < 0$.

Уравнение (2) можно представить в следующем виде:

$$\sum_{n>0} a_n x^n = \left(\sum_{j=0}^{n_0} p_{0,j} x^j\right) + \left(\sum_{j=0}^{n_1} p_{1,j} x^j\right) \cdot \left(\sum_{j_1>0} a_{j_1} x^{j_1}\right) + \ldots + \left(\sum_{j=0}^{n_m} p_{m,j} x^j\right) \cdot \left(\sum_{j_1>0} a_{j_1} x^{j_1}\right) \cdot \ldots \cdot \left(\sum_{j_m>0} a_{j_m} x^{j_m}\right)$$

или

$$\sum_{n>0} a_n x^n = \sum_{i=0}^{m} \left(\left(\sum_{j=0}^{n_i} p_{i,j} x^j\right) \cdot \left(\sum_{j_1>0} a_{j_1} x^{j_1}\right) \cdot \ldots \cdot \left(\sum_{j_i>0} a_{j_i} x^{j_i}\right)\right).$$

Слева коэффициент a_n расположен при x^n, значит в правой части уравнения необходимо определить коэффициент при такой же степени переменной. Справа имеется $m + 1$ слагаемых, что соответствует количеству многочленов $P_i(x)$, где $i \in \{0, \ldots, m\}$, поэтому в рамках каждого слагаемого нужно найти коэффициент при x^n и просуммировать их. По условию у всех многочленов $P_i(x)$ и у производящей функции $A(x)$ отсутствует переменная x с отрицательным значением показателя степени. Следовательно, в рамках каждого слагаемого требуется собрать комбинацию из $i + 1$ одночленов (один одночлен из $P_i(x)$ и i одночленов из $A(x)$) так, чтобы сумма показателей степеней переменной x была равна n. Таким образом, требуется рассмотреть все композиции числа n длиной $i + 1$, то есть $j + j_1 + \ldots + j_i = n$. При этом по условию $j \in \{0, \ldots, n_i\}$, а из условия $a_0 = 0$ следует, что $j_1, \ldots, j_i \in \{1, \ldots, n - 1\}$. В результате получаем искомую рекуррентную формулу (3) для вычисления коэффициентов a_n производящей функции $A(x)$. □

Также отметим, что условие $p_{1,0} = 0$ гарантирует отсутствие в правой части уравнения (2) слагаемого с производящей функцией $A(x)$ без умножения на переменную x, то есть $p_{1,0} \cdot a_n = 0$. Кроме того, условие $a_0 = 0$ гарантирует отсутствие a_n в $p_{i,j} \cdot a_{j_1} \cdot \ldots \cdot a_{j_i}$ для $i > 1$. В противном случае будет получено некорректное рекуррентное соотношение, в рамках которого вычисление коэффициента a_n потребует значения коэффициента a_n.

Однако комбинаторным множествам свойственно использование пустого объекта для случая объекта размера $n = 0$. В таком случае связанная с данным комбинаторным множеством производящая функция будет иметь единичный свободный коэффициент ($a_0 = 1$) и становится невозможным применение рекуррентной формулы (3). Тогда для получения требуемого вида нужно преобразовать производящую функцию путем вычитания ее свободного коэффициента, то есть $A(x) := A(x) - a_0$, либо путем домножения на переменную x, то есть $A(x) := x \cdot A(x)$.

Применение рекуррентной формулы (3) является неэффективным с точки зрения вычислительной сложности. В частности, на каждом уровне рекурсии при вычислении значения a_n требуется для каждого $i \in \{0, \ldots, m\}$ определить все композиции числа n длиной $i + 1$ в виде суммы $j + j_1 + \ldots + j_i = n$, где $j \in \{0, \ldots, n_i\}$ и $j_1, \ldots, j_i \in \{1, \ldots, n - 1\}$. Временная сложность перебора всех искомых композиций числа n равна $O(\sum_{i=0}^{m} \binom{n}{i})$ при условии наличия алгоритма генерации каждой последующей композиции с постоянной временной сложностью $O(1)$. Для $m < \frac{n}{2}$ эта оценка не превышает $O(n^m)$, а для худшего случая $m = n$ не превышает $O(2^n)$. В то же время каждая композиция генерирует множество новых рекурсивных вызовов для вычисления значений $a_{j_1}, \ldots, a_{j_i}$, при этом максимальная глубина рекурсии равна n. Таким образом, итоговая сложность вычисления значения a_n по рекуррентной формуле (3) является экспоненциальной, что делает ее неприменимой при решении реальных задач.

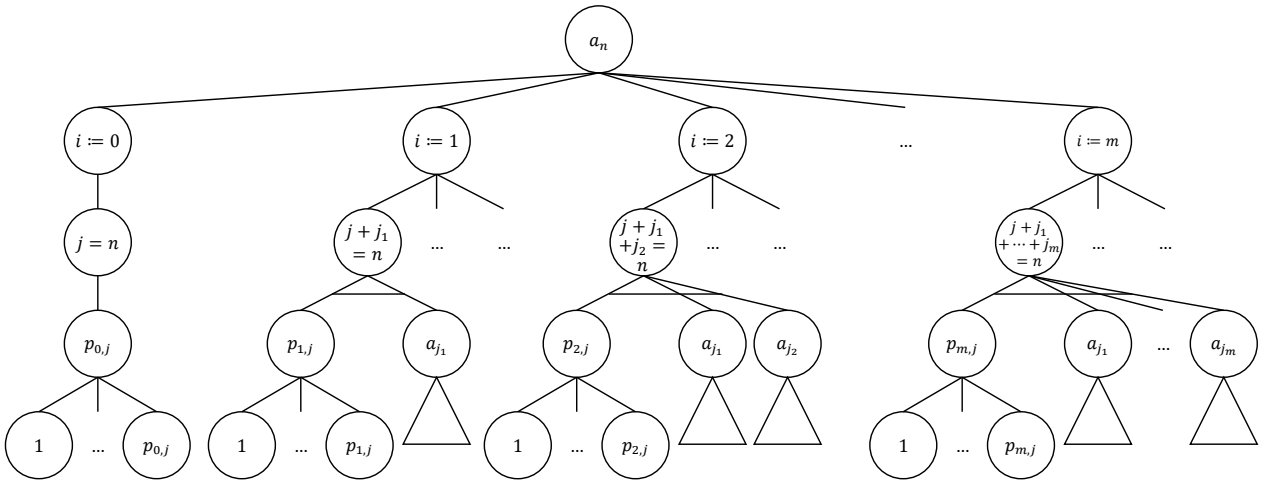

Fig. 4. AND/OR tree structure for a_n using the formula (3)

Рис. 4. Структура дерева И/ИЛИ для a_n при использовании формулы (3)

Чтобы исключить дублирование вычислений $a_1, \ldots, a_{n-1}$ на разных уровнях рекурсивных вызовов, можно организовать их хранение с использованием $O(n)$ дополнительной памяти. Тогда временная сложность вычисления значения a_n по рекуррентной формуле (3) станет равна $O(m \cdot n^{m+1})$:

- требуется вычислить n значений $a_1, \ldots, a_n$;
- для каждого из них требуется рассмотреть значения параметра $i \in \{0, \ldots, m\}$;
- для каждого случая параметра i требуется определить все композиции числа n, количество которых не превышает n^m.

Отметим, что в данном случае предполагается модель вычислений с использованием равномерной функции стоимости [19]. В рамках данной модели вычислений принимается за постоянную величину стоимость каждой машинной операции вне зависимости от размера обрабатываемых чисел, то есть временная сложность арифметических операций с числами равна $O(1)$. Такой подход широко распространен в научных работах, связанных с разработкой комбинаторных алгоритмов, и позволяет сравнить вычислительную сложность алгоритмов без привязки к архитектуре используемой вычислительной системы.

Кроме того, если выполняется условие $p_{i,j} \in \mathbb{Z}_{\geqslant 0}$ для $i \in \{0, \ldots, m\}$ и $j \in \{0, \ldots, n_i\}$, то по рекуррентной формуле (3) можно построить соответствующую рекурсивную структуру дерева И/ИЛИ (рисунок 4). Полученная структура дерева И/ИЛИ характеризуется экспоненциальным ростом в ширину при увеличении уровня дерева. В свою очередь, это приводит к экспоненциальной временной сложности алгоритма генерации по рангу, разработанного на основе такой древовидной структуры.

Для уменьшения вычислительных затрат предлагается рассмотреть коэффициенты k-й степени производящей функции $A(x)$.

Теорема 2. *Для алгебраической производящей функции*

$$A(x) = \sum_{n>0} a_n \, x^n,$$

удовлетворяющей функциональному уравнению

$$A(x) = \sum_{i=0}^{m} P_i(x) \cdot A(x)^i, \tag{4}$$

где $P_i(x) = \sum\limits_{j=0}^{n_i} p_{i,j}\, x^j$, $p_{i,j} \in \mathbb{Z}_{\geqslant 0}$ *и* $p_{1,0} = 0$, *и ее* k*-й степени*

$$A(x)^k = \sum_{n \geqslant k} a_{n,k}\, x^n,$$

где $k \geqslant 0$, *справедлива следующая рекуррентная формула:*

$$a_n = \begin{cases} 0, & n \leqslant 0; \\ p_{0,1}, & n = 1; \\ \sum\limits_{i=0}^{m} \sum\limits_{j=0}^{\min(n_i, n-i)} p_{i,j} \cdot a_{n-j,i}, & n > 1. \end{cases} \tag{5}$$

где

$$a_{n,k} = \begin{cases} 1, & n = k = 0; \\ (p_{0,1})^n, & n = k > 0; \\ \sum\limits_{i=0}^{m} \sum\limits_{j=0}^{\min(n_i, n-k+1-i)} p_{i,j} \cdot a_{n-j,k-1+i}, & n > k > 0; \\ 0, & \textit{иначе.} \end{cases} \tag{6}$$

Доказательство. Значение $a_n = 0$ для $n \leqslant 0$ следует из доказательства теоремы 1.

Основываясь на формуле (4), производящая функция $A(x)$ может быть представлена как сложение $m + 1$ производящих функций вида

$$B_i(x) = \sum_{n>0} b_{i,n}\, x^n = P_i(x) \cdot A(x)^i,$$

где $i \in \{0, \ldots, m\}$, то есть

$$A(x) = B_0(x) + B_1(x) + \ldots + B_m(x) = \sum_{i=0}^{m} B_i(x).$$

Следовательно, коэффициенты a_n связаны с коэффициентами $b_{i,n}$ следующим выражением для случая $n > 0$:

$$a_n = b_{0,n} + b_{1,n} + \ldots + b_{m,n} = \sum_{i=0}^{m} b_{i,n}.$$

В свою очередь, каждая производящая функция $B_i(x)$ является результатом умножения двух производящих функций $P_i(x)$ и $A(x)^i$:

$$B_i(x) = \sum_{n>0} b_{i,n}\, x^n = P_i(x) \cdot A(x)^i = \left(\sum_{n \geqslant 0} p_{i,n}\, x^n \right) \cdot \left(\sum_{n \geqslant i} a_{n,i}\, x^n \right).$$

Следовательно, коэффициенты $b_{i,n}$ связаны с коэффициентами $p_{i,n}$ и $a_{n,i}$ следующим выражением для случая $n > 0$:

$$b_{i,n} = \sum_{j=0}^{n} p_{i,j} \cdot a_{n-j,i}.$$

Комбинируя формулы для a_n и $b_{i,n}$, для случая $n > 0$ получаем

$$a_n = \sum_{i=0}^{m} \sum_{j=0}^{n} p_{i,j} \cdot a_{n-j,i}. \tag{7}$$

Учитывая тот факт, что $a_0 = 0$ для производящей функции $A(x)$, то в разложении в степенной ряд для ее k-й степени отсутствуют переменные с показателем степени меньше k, то есть $a_{n,k} = 0$ для $n < k$. Тогда для $a_{n-j,i}$ в формуле (7) имеем ограничение сверху для параметра j, а именно $j \leqslant n - i$. Также по условию $p_{i,j} = 0$ для $j > n_i$, поэтому для параметра j в формуле (7) имеем дополнительное ограничение сверху, а именно $j \leqslant n_i$.

Отдельно рассмотрим случай для a_1. Согласно формуле (7) с учетом указанных выше ограничений сверху для параметра j, получаем

$$a_1 = p_{0,0} \cdot a_{1,0} + p_{0,1} \cdot a_{0,0} + p_{1,0} \cdot a_{1,1}.$$

По определению $A(x)^0 = 1$, поэтому $a_{0,0} = 1$ и $a_{n,0} = 0$ для $n \neq 0$. Также по условию задано, что $p_{1,0} = 0$. Упрощая выражение для a_1, получаем $a_1 = p_{0,1}$. Таким образом, комбинация полученных формул предоставляет искомую рекуррентную формулу (5) для вычисления коэффициентов a_n производящей функции $A(x)$.

Чтобы получить рекуррентную формулу (6) для вычисления коэффициентов $a_{n,k}$ производящей функции $A(x)^k$, умножим левую и правую части уравнения (4) на $A(x)^{k-1}$, тогда получим

$$A(x)^k = \sum_{i=0}^{m} P_i(x) \cdot A(x)^{k-1+i}.$$

Основываясь на данной формуле, производящая функция $A(x)^k$ может быть представлена как сложение $m + 1$ производящих функций, каждая из которых, в свою очередь, является результатом умножения двух производящих функций $P_i(x)$ и $A(x)^{k-1+i}$. Следовательно, по аналогии с выводом формулы (7), для случая $n > 0$ и $k > 0$ получаем

$$a_{n,k} = \sum_{i=0}^{m} \sum_{j=0}^{n} p_{i,j} \cdot a_{n-j,k-1+i}. \tag{8}$$

Кроме того, ранее было показано, что $a_{n,k} = 0$ для $n < k$, а также что $a_{n,k} = 1$ для $n = k = 0$ и $a_{n,k} = 0$ для $n \neq k = 0$. Тогда для $a_{n-j,k-1+i}$ в формуле (8) имеем ограничение сверху для параметра j, а именно $j \leqslant n - k + 1 - i$.

Отдельно рассмотрим случай для $a_{n,k}$, где $n = k$. Согласно формуле (8) с учетом указанных выше ограничений сверху для параметра j, получаем

$$a_{n,n} = p_{0,0} \cdot a_{n,n-1} + p_{0,1} \cdot a_{n-1,n-1} + p_{1,0} \cdot a_{n,n}.$$

Учитывая тот факт, что $a_0 = 0$ для производящей функции $A(x)$, то из уравнения (4) следует, что $p_{0,0} = 0$. Также по условию задано, что $p_{1,0} = 0$. Упрощая выражение для $a_{n,n}$, получаем

$$a_{n,n} = p_{0,1} \cdot a_{n-1,n-1}.$$

Далее данное выражение рекурсивно вызывается для $a_{n-1,n-1}$ и повторяется до достижения коэффициента $a_{0,0} = 1$:

$$a_{n,n} = p_{0,1} \cdot a_{n-1,n-1} = p_{0,1} \cdot \left(p_{0,1} \cdot a_{n-2,n-2}\right) = \ldots = p_{0,1}^n.$$

Таким образом, комбинация полученных формул предоставляет искомую рекуррентную формулу (6) для вычисления коэффициентов $a_{n,k}$ производящей функции $A(x)^k$. □

Полученные рекуррентные формулы (5) и (6) характеризуются лучшими оценками вычислительной сложности по сравнению с использованием рекуррентной формулы (3).

Теорема 3. *Вычисление значения $a_{n,k}$ по рекуррентной формуле (6) требует $O((n_1 + \ldots + n_m + m) \cdot n^2)$ операций и $O(n^2)$ памяти.*

Доказательство. Реализуем последовательное вычисление значений $a_{i,j}$ для $0 \leqslant j \leqslant i \leqslant n$ с сохранением полученных результатов. Таким образом, нужно вычислить $(n+1)^2/2$ значений $a_{i,j}$, которые образуют числовой треугольник и могут храниться в таблице размера $n+1$ на $n+1$. Следовательно, требуется $O(n^2)$ памяти.

Согласно рекуррентной формуле (6), для вычисления каждого значения $a_{i,j}$ требуется просуммировать $(m+1) \cdot (n+1)$ произведений вида $p_{i,j} \cdot a_{n-j,k-1+i}$. Учитывая тот факт, что $p_{i,j} = 0$ для $j > n_i$, то остается не более $n_i + 1$ ненулевых коэффициентов $p_{i,j}$ для каждого значения параметра $i \in \{0, \ldots, m\}$. Следовательно, для вычисления каждого значения $a_{i,j}$ требуется просуммировать $n_1 + \ldots + n_m + m$ произведений вида $p_{i,j} \cdot a_{n-j,k-1+i}$. Тогда временная сложность вычисления значения $a_{n,k}$ по рекуррентной формуле (6) равна $O((n_1 + \ldots + n_m + m) \cdot n^2)$, так как:

- требуется вычислить $\frac{(n+1)^2}{2}$ значений $a_{i,j}$;
- для каждого значения $a_{i,j}$ требуется просуммировать $n_1 + \ldots + n_m + m$ произведений вида $p_{i,j} \cdot a_{n-j,k-1+i}$.

□

Так как $a_n = a_{n,1}$, а также для вычисления a_n по рекуррентной формуле (5) требуется расчет значений $a_{n-j,i}$, поэтому вычисление значения a_n характеризуется аналогичными оценками вычислительной сложности.

Следствие 1. *Вычисление значения a_n по рекуррентной формуле (5) требует $O((n_1 + \ldots + n_m + m) \cdot n^2)$ операций и $O(n^2)$ памяти.*

Кроме того, если выполняется условие $p_{i,j} \in \mathbb{Z}_{\geqslant 0}$ для $i \in \{0, \ldots, m\}$ и $j \in \{0, \ldots, n_i\}$, то по рекуррентным формулам (5) и (6) можно построить соответствующие рекурсивные композиции структур деревьев И/ИЛИ (рисунки 5 и 6). Также структура дерева И/ИЛИ для a_n может быть дополнена случаем $a_0 = 1$, чтобы была возможность учета пустого объекта.

На основе полученных структур деревьев И/ИЛИ для a_n и $a_{n,k}$ строятся рекурсивные алгоритмы генерации по рангу для соответствующих комбинаторных множеств (алгоритмы 6 и 7).

В данных алгоритмах генерации по рангу используются обозначения для вспомогательных функций, реализующих правила построения конкретного комбинаторного объекта:

- функция `GetObject`(r, p) реализует построение комбинаторного объекта на основе правила под номером r из набора правил p;
- функция `ConvertObject`$(r, p, object)$ реализует правило преобразования комбинаторного объекта *object* на основе правила под номером r из набора правил p.

Временная сложность алгоритмов генерации по рангу равна $O((n_1 + \ldots + n_m + m) \cdot n \cdot O(a_{n,k}))$, так как:

- каждый рекурсивный вызов требует $n_1 + \ldots + n_m + m$ раз вычислить значение $a_{i,j}$ в ходе выполнения двух вложенных циклов for;
- глубина рекурсии не превышает n, так как останавливается при достижении условия $n = 0$ или $n = k$.

Если для вычисления значения a_n использовать рекуррентную формулу (5) с временной сложностью $O((n_1 + \ldots + n_m + m) \cdot n^2)$, то алгоритм генерации по рангу будет иметь временную сложность $O((n_1 + \ldots + n_m + m)^2 \cdot n^3)$. Дополнительное повышение скорости работы алгоритма генерации

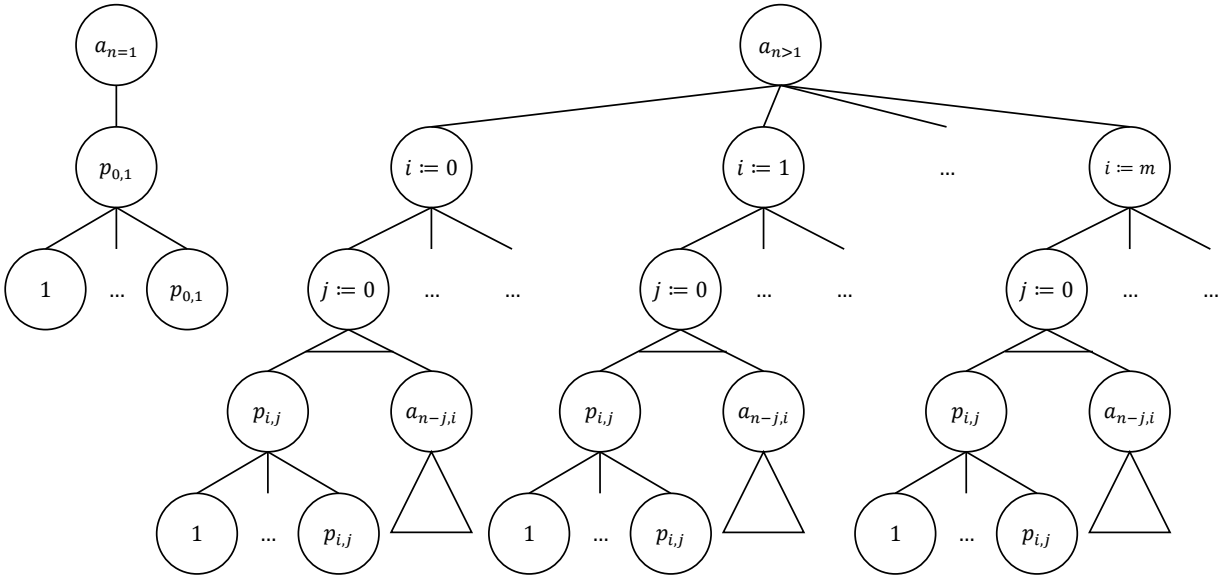

Fig. 5. AND/OR tree structures for a_n using the formula (5)

Рис. 5. Структуры дерева И/ИЛИ для a_n при использовании формулы (5)

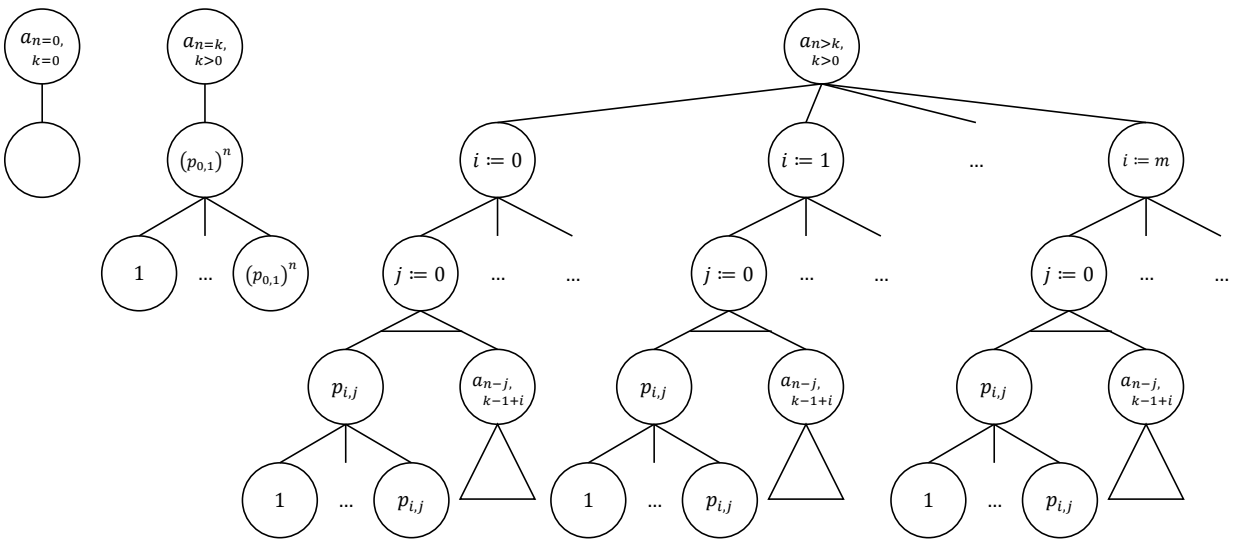

Fig. 6. AND/OR tree structures for $a_{n,k}$ using the formula (6)

Рис. 6. Структуры дерева И/ИЛИ для $a_{n,k}$ при использовании формулы (6)

по рангу может быть достигнуто за счет получения явной формулы для вычисления значения a_n. Таким образом, получаемые алгоритмы комбинаторной генерации обладают полиномиальной вычислительной сложностью, что делает их применимыми при решении реальных задач. Кроме того, коэффициенты $a_{n,k}$ расширяют возможности генерации, так как позволяют строить не только объекты исходного комбинаторного множества, связанного с производящей функцией $A(x)$, но и кортежи таких объектов для случая $k > 1$.

4. Примеры разработки алгоритмов комбинаторной генерации

Далее рассмотрим несколько примеров использования результатов данной статьи для частных случаев алгебраических производящих функций, удовлетворяющих требованиям теоремы 2.

Algorithm 6. Algorithm for unranking a_n using the formula (5)

Алгоритм 6. Алгоритм генерации по рангу для a_n при использовании формулы (5)

```
1  Unrank_A(r, n)
2  begin
3      if n = 0 then object := ε                                  // Пустой объект
4      if n = 1 then object := GetObject(r, p_{0,1})              // Правила построения объекта
5      else
6          sum := 0
7          for i := 0 to m do                                     // Поиск выбранного узла с меткой i := I
8              for j := 0 to min(n_i, n − k + 1 − i) do            // Поиск выбранного узла с меткой j := J
9                  if sum + p_{i,j} · a_{n−j,i} > r then           // Если найдены выбранные узлы
10                     r := r − sum
11                     I := i
12                     J := j
13                     break
14                 end
15                 sum := sum + p_{i,j} · a_{n−j,i}
16             end
17         end
18         r_1 := r mod p_{I,J}                                   // Ранг варианта поддерева узла с меткой p_{I,J}
19         r_2 := ⌊ r / p_{I,J} ⌋                                 // Ранг варианта поддерева узла с меткой a_{n−J,I}
20         subobject := Unrank_Apow(r_2, n − J, I)
21         object := ConvertObject(r_1, p_{I,J}, subobject)       // Правила преобразования объекта
22     end
23     return object
24 end
```

Пример 1. Рассмотрим функциональное уравнение, которое определяет производящую функцию для последовательности чисел Фибоначчи (числовая последовательность $A000045$ в OEIS [20]):

$$A(x) = x + (x + x^2) \cdot A(x).$$

Согласно теореме 2 получаем следующую рекуррентную формулу для коэффициентов производящей функции $A(x)^k$:

$$a_{n,k} = \begin{cases} 1, & n = k; \\ a_{n-1,k-1} + a_{n-1,k} + a_{n-2,k}, & n > k > 0; \\ 0, & \text{иначе.} \end{cases} \tag{9}$$

Воспользуемся одной из известных комбинаторных интерпретаций для чисел Фибоначчи: a_n показывает количество композиций числа $n-1$ с использованием только единиц и двоек. Тогда $a_{n,k}$ показывает количество кортежей длины k, где каждый элемент представляет собой композицию числа $n_i - 1$ с использованием только единиц и двоек, причем $n_1 + n_2 + \ldots + n_k = n$ и общая сумма единиц и двоек равна $n-k$. Используя алгоритм 7 для формулы (9), получаем алгоритм для генерации таких кортежей (алгоритм 8).

В данном алгоритме используются следующие правила построения и преобразования объекта:

- если выполняется условие $n = k$ (строка 3 алгоритма), то в кортеже длины k элементов каждый элемент представляет собой пустой объект ε;

Algorithm 7. Algorithm for unranking $a_{n,k}$ using the formula (6)

Алгоритм 7. Алгоритм генерации по рангу для $a_{n,k}$ при использовании формулы (6)

```
1  Unrank_Apow(r, n, k)
2  begin
3      if n = 0 and k = 0 then object := ε                          // Пустой объект
4      else if n = k then object := GetObject(r, p^n_{0,1})         // Правила построения объекта
5      else
6          sum := 0
7          for i := 0 to m do                                       // Поиск выбранного узла с меткой i := I
8              for j := 0 to min(n_i, n − k + 1 − i) do              // Поиск выбранного узла с меткой j := J
9                  if sum + p_{i,j} · a_{n−j,k−1+i} > r then         // Если найдены выбранные узлы
10                     r := r − sum
11                     I := i
12                     J := j
13                     break
14                 end
15                 sum := sum + p_{i,j} · a_{n−j,k−1+i}
16             end
17         end
18         r_1 := r mod p_{I,J}                                     // Ранг варианта поддерева узла с меткой p_{I,J}
19         r_2 := ⌊r / p_{I,J}⌋                                     // Ранг варианта поддерева узла с меткой a_{n−J,k−1+I}
20         subobject := Unrank_Apow(r_2, n − J, k − 1 + I)
21         object := ConvertObject(r_1, p_{I,J}, subobject)         // Правила преобразования объекта
22     end
23     return object
24 end
```

- если выполняется условие $r < a_{n-1,k-1}$ (строка 5 алгоритма), то к кортежу длины $k-1$ элементов в конец добавляется новый элемент, представляющий собой пустой объект ε;
- если выполняется условие $r - a_{n-1,k-1} < a_{n-1,k}$ (строка 11 алгоритма), то в кортеже длины k элементов к последнему элементу добавляется единица к записи композиции числа;
- иначе выполняется условие $r - a_{n-1,k-1} - a_{n-1,k} < a_{n-2,k}$ (строка 15 алгоритма), тогда в кортеже длины k элементов к последнему элементу добавляется двойка к записи композиции числа.

В таблице 1 представлены примеры результатов генерации комбинаторных объектов путем запуска алгоритма 8 для $n = 5$ и $k \in \{1, 2, 3, 4\}$.

Пример 2. Рассмотрим функциональное уравнение, которое определяет производящую функцию для последовательности чисел Пелля (числовая последовательность $A000129$ в OEIS [20]):

$$A(x) = x + (2x + x^2) \cdot A(x).$$

Согласно теореме 2 получаем следующую рекуррентную формулу для коэффициентов производящей функции $A(x)^k$:

$$a_{n,k} = \begin{cases} 1, & n = k; \\ a_{n-1,k-1} + 2 \cdot a_{n-1,k} + a_{n-2,k}, & n > k > 0; \\ 0, & \text{иначе.} \end{cases} \tag{10}$$

Algorithm 8. Algorithm for unranking $a_{n,k}$ using the formula (9)

Алгоритм 8. Алгоритм генерации по рангу для $a_{n,k}$ при использовании формулы (9)

```
1  Unrank_Apow(r, n, k)
2  begin
3      if n = k then object := (object_1, ..., object_k) = (ε, ..., ε)
4      else
5          if r < a_{n-1,k-1} then
6              (object_1, ..., object_{k-1}) := Unrank_Apow(r, n − 1, k − 1)
7              object := (object_1, ..., object_{k-1}, ε)
8          end
9          else
10             r := r − a_{n-1,k-1}
11             if r < a_{n-1,k} then
12                 (object_1, ..., object_k) := Unrank_Apow(r, n − 1, k)
13                 object := (object_1, ..., object_k + 1)
14             end
15             else
16                 r := r − a_{n-1,k}
17                 (object_1, ..., object_k) := Unrank_Apow(r, n − 2, k)
18                 object := (object_1, ..., object_k + 2)
19             end
20         end
21     end
22     return object
23 end
```

Table 1. Results of generating objects using the algorithm 8 for $n = 5$ and $k \in \{1, 2, 3, 4\}$

Таблица 1. Результаты генерации объектов с помощью алгоритма 8 для $n = 5$ и $k \in \{1, 2, 3, 4\}$

Ранг	Мощность комбинаторного множества и его элементы			
	$a_{5,1} = 5$	$a_{5,2} = 10$	$a_{5,3} = 9$	$a_{5,4} = 4$
$r = 0$	$(1 + 1 + 1 + 1)$	$(1 + 1 + 1, \varepsilon)$	$(1 + 1, \varepsilon, \varepsilon)$	$(1, \varepsilon, \varepsilon, \varepsilon)$
$r = 1$	$(2 + 1 + 1)$	$(2 + 1, \varepsilon)$	$(2, \varepsilon, \varepsilon)$	$(\varepsilon, 1, \varepsilon, \varepsilon)$
$r = 2$	$(1 + 2 + 1)$	$(1 + 2, \varepsilon)$	$(1, 1, \varepsilon)$	$(\varepsilon, \varepsilon, 1, \varepsilon)$
$r = 3$	$(1 + 1 + 2)$	$(1 + 1, 1)$	$(\varepsilon, 1 + 1, \varepsilon)$	$(\varepsilon, \varepsilon, \varepsilon, 1)$
$r = 4$	$(2 + 2)$	$(2, 1)$	$(\varepsilon, 2, \varepsilon)$	–
$r = 5$	–	$(1, 1 + 1)$	$(1, \varepsilon, 1)$	–
$r = 6$	–	$(\varepsilon, 1 + 1 + 1)$	$(\varepsilon, 1, 1)$	–
$r = 7$	–	$(\varepsilon, 2 + 1)$	$(\varepsilon, \varepsilon, 1 + 1)$	–
$r = 8$	–	$(1, 2)$	$(\varepsilon, \varepsilon, 2)$	–
$r = 9$	–	$(\varepsilon, 1 + 2)$	–	–

Воспользуемся одной из известных комбинаторных интерпретаций для чисел Пелля: a_n показывает количество композиций числа $n - 1$ с использованием только единиц и двоек, причем имеется два вида единиц (например, единицы двух цветов). Тогда $a_{n,k}$ показывает количество кортежей длины k, где каждый элемент представляет собой композицию числа $n_i - 1$ с использованием только единиц (двух видов) и двоек, причем $n_1 + n_2 + \ldots + n_k = n$ и общая сумма единиц и двоек равна

Algorithm 9. Algorithm for unranking $a_{n,k}$ using the formula (10)

Алгоритм 9. Алгоритм генерации по рангу для $a_{n,k}$ при использовании формулы (10)

```
1  Unrank_Apow(r, n, k)
2  begin
3      if n = k then object := (object_1, ..., object_k) = (ε, ..., ε)
4      else
5          if r < a_{n-1,k-1} then
6              (object_1, ..., object_{k-1}) := Unrank_Apow(r, n − 1, k − 1)
7              object := (object_1, ..., object_{k-1}, ε)
8          end
9          else
10             r := r − a_{n-1,k-1}
11             if r < 2 · a_{n-1,k} then
12                 r_1 := r mod 2
13                 r_2 := ⌊r/2⌋
14                 (object_1, ..., object_k) := Unrank_Apow(r_2, n − 1, k)
15                 if r_1 = 0 then object := (object_1, ..., object_k + 1)
16                 else object := (object_1, ..., object_k + 1)
17             end
18             else
19                 r := r − 2 · a_{n-1,k}
20                 (object_1, ..., object_k) := Unrank_Apow(r, n − 2, k)
21                 object := (object_1, ..., object_k + 2)
22             end
23         end
24     end
25     return object
26 end
```

$n-k$. Используя алгоритм 7 для формулы (10), получаем алгоритм для генерации таких кортежей (алгоритм 9).

В таблице 2 представлены примеры результатов генерации комбинаторных объектов путем запуска алгоритма 9 для $n = 4$ и $k \in \{1, 2, 3, 4\}$.

Пример 3. Рассмотрим функциональное уравнение, которое определяет производящую функцию для последовательности чисел Каталана (числовая последовательность $A000108$ в OEIS [20]), которые разделены нулями (числовая последовательность $A126120$ в OEIS [20]):

$$A(x) = x + x \cdot A(x)^2.$$

Согласно теореме 2 получаем следующую рекуррентную формулу для коэффициентов производящей функции $A(x)^k$:

$$a_{n,k} = \begin{cases} 1, & n = k; \\ a_{n-1,k-1} + a_{n-1,k+1}, & n > k > 0; \\ 0, & \text{иначе.} \end{cases} \tag{11}$$

Воспользуемся одной из известных комбинаторных интерпретаций для чисел Каталана [21]: a_n показывает количество правильных скобочных последовательностей длины $n-1$. Тогда $a_{n,k}$ пока-

Table 2. Results of generating objects using the algorithm 9 for $n = 4$ and $k \in \{1, 2, 3, 4\}$

Таблица 2. Результаты генерации объектов с помощью алгоритма 9 для $n = 4$ и $k \in \{1, 2, 3, 4\}$

Ранг	Мощность комбинаторного множества и его элементы			
	$a_{4,1} = 12$	$a_{4,2} = 14$	$a_{4,3} = 6$	$a_{4,4} = 1$
$r = 0$	$(1 + 1 + 1)$	$(1 + 1, \varepsilon)$	$(1, \varepsilon, \varepsilon)$	$(\varepsilon, \varepsilon, \varepsilon, \varepsilon)$
$r = 1$	$(1 + 1 + 1)$	$(1 + 1, \varepsilon)$	$(1, \varepsilon, \varepsilon)$	–
$r = 2$	$(1 + 1 + 1)$	$(1 + 1, \varepsilon)$	$(\varepsilon, 1, \varepsilon)$	–
$r = 3$	$(1 + 1 + 1)$	$(1 + 1, \varepsilon)$	$(\varepsilon, 1, \varepsilon)$	–
$r = 4$	$(1 + 1 + 1)$	$(2, \varepsilon)$	$(\varepsilon, \varepsilon, 1)$	–
$r = 5$	$(1 + 1 + 1)$	$(1, 1)$	$(\varepsilon, \varepsilon, 1)$	–
$r = 6$	$(1 + 1 + 1)$	$(1, 1)$	–	–
$r = 7$	$(1 + 1 + 1)$	$(1, 1)$	–	–
$r = 8$	$(2 + 1)$	$(1, 1)$	–	–
$r = 9$	$(2 + 1)$	$(\varepsilon, 1 + 1)$	–	–
$r = 10$	$(1 + 2)$	$(\varepsilon, 1 + 1)$	–	–
$r = 11$	$(1 + 2)$	$(\varepsilon, 1 + 1)$	–	–
$r = 12$	–	$(\varepsilon, 1 + 1)$	–	–
$r = 13$	–	$(\varepsilon, 2)$	–	–

зывает количество кортежей длины k, где каждый элемент представляет собой правильную скобочную последовательность длины $n_i - 1$, причем $n_1 + n_2 + \ldots + n_k = n$ и общее количество открывающих и закрывающих скобок равно $n - k$. Используя алгоритм 7 для формулы (11), получаем алгоритм для генерации таких кортежей (алгоритм 10).

В данном алгоритме используются следующие правила построения и преобразования объекта:

- если выполняется условие $n = k$ (строка 3 алгоритма), то в кортеже длины k элементов каждый элемент представляет собой пустой объект ε;
- если выполняется условие $r < a_{n-1,k-1}$ (строка 5 алгоритма), то к кортежу длины $k-1$ элементов в конец добавляется новый элемент, представляющий собой пустой объект ε;
- иначе выполняется условие $r - a_{n-1,k-1} < a_{n-1,k+1}$ (строка 9 алгоритма), тогда в кортеже длины $k+1$ элементов предпоследний элемент обрамляется открывающей и закрывающей скобками и к нему присоединяется последний элемент кортежа.

В таблице 3 представлены примеры результатов генерации комбинаторных объектов путем запуска алгоритма 10.

Пример 4. Рассмотрим функциональное уравнение, которое определяет производящую функцию для последовательности чисел Моцкина (числовая последовательность $A001006$ в OEIS [20]):

$$A(x) = x + x \cdot A(x) + x \cdot A(x)^2.$$

Согласно теореме 2 получаем следующую рекуррентную формулу для коэффициентов производящей функции $A(x)^k$:

$$a_{n,k} = \begin{cases} 1, & n = k; \\ a_{n-1,k-1} + a_{n-1,k} + a_{n-1,k+1}, & n > k > 0; \\ 0, & \text{иначе.} \end{cases}$$

Разработка алгоритма генерации по рангу кортежей, мощность которых определяется значением $a_{n,k}$, осуществляется аналогично предыдущим примерам на основе какой-либо известной комбинаторной интерпретации для значений a_n.

Algorithm 10. Algorithm for unranking $a_{n,k}$ using the formula (11)

Алгоритм 10. Алгоритм генерации по рангу для $a_{n,k}$ при использовании формулы (11)

```
1  Unrank_Apow(r, n, k)
2  begin
3      if n = k then object := (object_1, ..., object_k) = (ε, ..., ε)
4      else
5          if r < a_{n-1,k-1} then
6              (object_1, ..., object_{k-1}) := Unrank_Apow(r, n − 1, k − 1)
7              object := (object_1, ..., object_{k-1}, ε)
8          end
9          else
10             r := r − a_{n-1,k-1}
11             (object_1, ..., object_{k+1}) := Unrank_Apow(r, n − 1, k + 1)
12             object := (object_1, ..., concat("(", object_k, ")", object_{k+1}))
13         end
14     end
15     return object
16 end
```

Table 3. Results of generating objects using the algorithm 10

Таблица 3. Результаты генерации объектов с помощью алгоритма 10

Ранг	Мощность комбинаторного множества и его элементы			
	$a_{9,1} = 14$	$a_{8,2} = 14$	$a_{7,3} = 9$	$a_{8,4} = 14$
$r = 0$	(”(((()))”)	(”((()))”, ε)	(”(())”, ε, ε)	(”(())”, ε, ε, ε)
$r = 1$	(”((()()))”)	(”(()())”, ε)	(”()()”, ε, ε)	(”()()”, ε, ε, ε)
$r = 2$	(”((())())”)	(”(())()”, ε)	(”()”, ”()”, ε)	(”()”, ”()”, ε, ε)
$r = 3$	(”(()(()))”)	(”()(())”, ε)	(ε, ”(())”, ε)	(ε, ”(())”, ε, ε)
$r = 4$	(”(()()())”)	(”()()()”, ε)	(ε, ”()()”, ε)	(ε, ”()()”, ε, ε)
$r = 5$	(”((()))()”)	(”(())”, ”()”)	(”()”, ε, ”()”)	(”()”, ε, ”()”, ε)
$r = 6$	(”(()())()”)	(”()()”, ”()”)	(ε, ”()”, ”()”)	(ε, ”()”, ”()”, ε)
$r = 7$	(”(())(())”)	(”()”, ”(())”)	(ε, ε, ”(())”)	(ε, ε, ”(())”, ε)
$r = 8$	(”()((()))”)	(ε, ”((()))”)	(ε, ε, ”()()”)	(ε, ε, ”()()”, ε)
$r = 9$	(”()(()())”)	(ε, ”(()())”)	–	(”()”, ε, ε, ”()”)
$r = 10$	(”(())()()”)	(”()”, ”()()”)	–	(ε, ”()”, ε, ”()”)
$r = 11$	(”()(())()”)	(ε, ”(())()”)	–	(ε, ε, ”()”, ”()”)
$r = 12$	(”()()(())”)	(ε, ”()(())”)	–	(ε, ε, ε, ”(())”)
$r = 13$	(”()()()()”)	(ε, ”()()()”)	–	(ε, ε, ε, ”()()”)

Пример 5. Рассмотрим функциональное уравнение, которое определяет производящую функцию для последовательности больших чисел Шредера (числовая последовательность $A006318$ в OEIS [20]):

$$A(x) = x + x \cdot A(x) + A(x)^2.$$

Согласно теореме 2 получаем следующую рекуррентную формулу для коэффициентов производящей функции $A(x)^k$:

$$a_{n,k} = \begin{cases} 1, & n = k; \\ a_{n-1,k-1} + a_{n-1,k} + a_{n,k+1}, & n > k > 0; \\ 0, & \text{иначе.} \end{cases}$$

Разработка алгоритма генерации по рангу кортежей, мощность которых определяется значением $a_{n,k}$, осуществляется аналогично предыдущим примерам на основе какой-либо известной комбинаторной интерпретации для значений a_n.

Пример 6. Рассмотрим функциональное уравнение, которое определяет производящую функцию для числовой последовательности $A001002$ в OEIS [20]:

$$A(x) = x + A(x)^2 + A(x)^3.$$

Согласно теореме 2 получаем следующую рекуррентную формулу для коэффициентов производящей функции $A(x)^k$:

$$a_{n,k} = \begin{cases} 1, & n = k; \\ a_{n-1,k-1} + a_{n,k+1} + a_{n,k+2}, & n > k > 0; \\ 0, & \text{иначе.} \end{cases}$$

Разработка алгоритма генерации по рангу кортежей, мощность которых определяется значением $a_{n,k}$, осуществляется аналогично предыдущим примерам на основе какой-либо известной комбинаторной интерпретации для значений a_n.

Заключение

В данной статье показано, что операции над производящими функциями (такие как сложение, умножение и возведение в степень), которые связаны с дискретными структурами, формируют новые комбинаторные объекты. В свою очередь, применение аппарата деревьев И/ИЛИ позволяет строить новые алгоритмы комбинаторной генерации, в том числе путем комбинирования существующих алгоритмов для подструктур исследуемого комбинаторного объекта.

В качестве основного результата предложен систематический подход к разработке алгоритмов комбинаторной генерации для множеств дискретных структур, определяемых классом алгебраических производящих функций и их степеней. На основе доказанных рекуррентных формул, предложенный подход позволяет строить алгоритмы с полиномиальной оценкой вычислительной сложности как с точки зрения временных затрат, так и с точки зрения затрат по памяти. Кроме того, использование коэффициентов степеней производящих функций расширяет возможности генерации, так как это позволяет строить не только объекты исходного комбинаторного множества, связанного с производящей функцией, но и кортежи таких объектов. Продемонстрированные примеры получения рекуррентных формул и алгоритмов генерации на их основе для классических числовых последовательностей (числа Фибоначчи, Пелля, Каталана, Моцкина, Шредера) подтверждают применимость и универсальность предложенного подхода.

Перспективными направлениями дальнейших исследований может стать расширение класса поддерживаемых функциональных уравнений, а также адаптация подхода к задачам с использованием вероятностных моделей. Кроме того, актуальной является направление оптимизации создаваемых алгоритмов комбинаторной генерации за счет реализации параллельных вычислений.

References

[1] S. Kemp. “Digital 2024: Global overview report”. (2024), [Online]. Available: https://datareportal.com/reports/digital-2024-global-overview-report (visited on 11/01/2025).

[2] D. E. Knuth, *The art of computer programming. Volume 4A: Combinatorial algorithms, part 1*. USA: Addison-Wesley, 2011, 912 pp.

[3] D. E. Knuth, *The art of computer programming. Volume 4B: Combinatorial algorithms, part 2*. USA: Addison-Wesley, 2022, 736 pp.

[4] D. L. Kreher and D. R. Stinson, *Combinatorial algorithms: Generation, enumeration, and search*. USA: CRC Press, 1999, 342 pp.

[5] E. Seyedi-Tabari, H. Ahrabian, and A. Nowzari-Dalini, "A new algorithm for generation of different types of RNA", *International Journal of Computer Mathematics*, vol. 87, no. 6, pp. 1197–1207, 2010. DOI: 10.1080/00207160802140049.

[6] M. E. Nebel, A. Scheid, and F. Weinberg, "Random generation of RNA secondary structures according to native distributions", *Algorithms for Molecular Biology*, vol. 6, p. 24, 2011. DOI: 10.1186/1748-7188-6-24.

[7] E. Onokpasa, S. Wild, and P. W. H. Wong, "RNA secondary structures: From ab initio prediction to better compression, and back", in *Data Compression Conference*, 2023, pp. 278–287. DOI: 10.1109/DCC55655.2023.00036.

[8] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers, "Format-preserving encryption", *Lecture Notes in Computer Science: International Workshop on Selected Areas in Cryptography*, vol. 5867, pp. 295–312, 2009. DOI: 10.1007/978-3-642-05445-7_19.

[9] D. Luchaup, T. Shrimpton, T. Ristenpart, and S. Jha, "Formatted encryption beyond regular languages", in *ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 1292–1303. DOI: 10.1145/2660267.266035.

[10] A. V. Goldberg and M. Sipser, "Compression and ranking", *SIAM Journal on Computing*, vol. 20, no. 3, pp. 524–536, 1991. DOI: 10.1137/0220034.

[11] Y. V. Shablya, "Compression of information objects using combinatorial generation methods based on AND/OR trees", *Proceedings of TUSUR University*, vol. 27, no. 4, pp. 74–79, 2024, in Russian. DOI: 10.21293/1818-0442-2024-27-4-74-79.

[12] Y. V. Shablya, D. V. Kruchinin, and V. V. Kruchinin, "Method for developing combinatorial generation algorithms based on AND/OR trees and its application", *Mathematics*, vol. 8, no. 6, p. 962, 2020. DOI: 10.3390/math8060962.

[13] Y. V. Shablya, "A method for constructing combinatorial generation algorithms for context-free languages based on AND/OR tree structures", *Information Technologies*, vol. 31, no. 9, pp. 465–476, 2025. DOI: 10.17587/it.31.465-476.

[14] Y. V. Shablya and D. V. Kruchinin, "Algorithms for ranking and unranking the combinatorial set of RNA secondary structures", *Discrete Mathematics, Algorithms and Applications*, p. 2 550 059, 2025. DOI: 10.1142/S1793830925500594.

[15] P. Flajolet, P. Zimmerman, and B. Cutsem, "A calculus for the random generation of combinatorial structures", *Theoretical Computer Science*, vol. 132, no. 1–2, pp. 1–35, 1994. DOI: 10.1016/0304-3975(94)90226-7.

[16] C. Martinez and X. Molinero, "A generic approach for the unranking of labeled combinatorial classes", *Random Structures and Algorithms*, vol. 19, no. 3–4, pp. 472–497, 2001. DOI: 10.1002/rsa.10025.

[17] C. Martinez and X. Molinero, "Efficient iteration in admissible combinatorial classes", *Theoretical Computer Science*, vol. 346, no. 2–3, pp. 388–417, 2005. DOI: 10.1016/j.tcs.2005.08.028.

[18] R. P. Stanley, *Enumerative combinatorics. Volume 2*. USA: Cambridge University Press, 1999, 600 pp.

[19] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The design and analysis of computer algorithms*. USA: Addison-Wesley, 1974, 470 pp.

[20] "The on-line encyclopedia of integer sequences". (2025), [Online]. Available: http://www.oeis.org/ (visited on 11/01/2025).

[21] D. V. Kruchinin, V. V. Kruchinin, and Y. V. Shablya, "On some properties of generalized Narayana numbers", *Quaestiones Mathematicae*, vol. 45, no. 12, pp. 1949–1963, 2022. DOI: 10.2989/16073606.2021.1980448.