Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Томский государственный университет систем управления и радиоэлектроники»
(ТУСУР)

На правах рукописи

Шабля Юрий Васильевич

АЛГОРИТМИЧЕСКОЕ ОБЕСПЕЧЕНИЕ КОМБИНАТОРНОЙ ГЕНЕРАЦИИ НА ОСНОВЕ ПРИМЕНЕНИЯ ТЕОРИИ ПРОИЗВОДЯЩИХ ФУНКЦИЙ

Специальность 05.13.17 — «Теоретические основы информатики»

Диссертация на соискание учёной степени кандидата технических наук

Научный руководитель:
доктор технических наук, профессор
Шелупанов Александр Александрович

Томск — 2019

Оглавление

Введение

Актуальность темы. Многие информационные объекты, в том числе с большим объемом данных, носят иерархическую или рекурсивную природу описания. В таком случае древовидная структура является естественной формой представления информационного объекта, что позволяет описать данный информационный объект формальным комбинаторным множеством и применить в дальнейшем алгоритмы комбинаторной генерации.

Комбинаторная генерация — раздел на стыке информатики и комбинаторики, развивающий методы и алгоритмы ранжирования и генерации комбинаторных множеств, таких как классы перестановок, разбиений, графов, деревьев, таблиц и многие другие [1–3]. Под ранжированием и генерацией комбинаторных множеств понимается нумерация объектов комбинаторных множеств, то есть их кодирование в виде чисел для удобства хранения информации о них, а также последующее восстановление из чисел самих объектов. Комбинаторная генерация тесно связана с процедурой индексации информационных объектов, а разработка эффективных алгоритмов комбинаторной генерации является актуальной задачей. Разработкой методов построения алгоритмов комбинаторной генерации занимались такие ученые, как Э. Рейнгольд [4], Д. Крехер [5], Е. Баркуччи [6; 7], С. Баччелли [8; 9], А. Дель Лунго [10; 11], В. Вайновски [12–14], Ф. Флажоле [15; 16], К. Мартинес и К. Мулинеро [17–20], Б.Я. Рябко [21; 22], Ю.С. Медведева [23; 24], В.В. Кручинин [25; 26].

Существующие универсальные методы построения алгоритмов комбинаторной генерации требуют представления рассматриваемых комбинаторных объектов в специальном виде за счет применения некоторых эвристик, что является трудной задачей ввиду отсутствия соответствующих методик. Предлагаемое научное исследование направлено на разработку и формализацию новых методов построения алгоритмов комбинаторной генерации. Для этого планируется применить для получения выражений функций мощности комбинаторных множеств математический аппарат теории производящих функций, который ранее не использовался в рамках рассматриваемой научной задачи. Производящие функции [27–29] являются мощным инструментом решения задач из самых разных областей математических наук, таких как комбинаторика, теория чисел, теория вероятностей и др.

Цели и задачи исследования. Целью диссертационной работы является развитие методов построения алгоритмов комбинаторной генерации за счет применения теории производящих функций.

Для достижения поставленной цели были решены следующие задачи:

1. Провести аналитический обзор современного состояния исследований в области разработки алгоритмов комбинаторной генерации.

2. Исследовать и формализовать метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ.

3. Исследовать метод получения явных выражений коэффициентов производящих функций.

4. Модифицировать метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ за счет применения теории производящих функций.

5. Провести апробацию модифицированного метода построения алгоритмов комбинаторной генерации, разработав и исследовав новые алгоритмы комбинаторной генерации.

6. Создать программное обеспечение для ранжирования и генерации по рангу элементов комбинаторных множеств на основе разработанных алгоритмов комбинаторной генерации.

Объект исследования. Объектом исследования являются методы и алгоритмы комбинаторной генерации.

Предмет исследования. Предметом исследования является построение новых эффективных алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ с использованием теории производящих функций.

Методы исследования. В диссертационной работе применялись методы построения алгоритмов комбинаторной генерации, получения явных выражений коэффициентов производящих функций, анализа алгоритмов, объектно-ориентированного программирования.

Научная новизна полученных результатов:

1. Предложен модифицированный метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ, который отличается применением метода получения явных выражений коэффициентов производящих функций для нахождения выражения функции мощности комбинаторного множества, для которого не известно выражение функции мощности, принадлежащее алгебре $\{\mathbb{N}, +, \times, R\}$.

2. Разработаны новые алгоритмы ранжирования и генерации по рангу для множества комбинаторных объектов, отражающих вторичную структуру РНК, отличающиеся от аналогов меньшей вычислительной сложностью, оценка которой равна $O(m^2(n-m))$.

3. Впервые созданы алгоритмы ранжирования и генерации по рангу для множества комбинаторных объектов, определяемых числовым треугольником Эйлера-Каталана.

Теоретическая значимость работы. Теоретическая значимость результатов диссертационной работы заключается в развитии методов построения алгоритмов комбинаторной генерации. Модифицированный метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ за счет применения теории производящих функций позволяет использовать данный метод для получения новых алгоритмов ранжирования и генерации по рангу как для известных, так и для новых комбинаторных множеств.

Практическая значимость работы. Предложенный модифицированный метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ расширяет перечень инструментов проведения исследований в области разработки алгоритмов комбинаторной генерации. Полученные в рамках апробации модифицированного метода алгоритмы комбинаторной генерации подтверждают эффективность и универсальность его применения для широкого разнообразия комбинаторных множеств.

Разработанное программное обеспечение для системы компьютерной алгебры «Maxima» позволяет в автоматизированном режиме решать задачи комбинаторной генерации по ранжированию и генерации по рангу элементов комбинаторных множеств. Это обеспечивает ускорение выполнения вычислений в рамках работы с алгоритмами комбинаторной генерации.

Практическая значимость результатов диссертационной работы подтверждается их внедрением в ООО «ПлантаПлюс», ООО «Удостоверяющий центр Сибири» и учебный процесс ФГБОУ ВО «ТУСУР», а также использованием в ходе выполнения научно-исследовательских работ. Например, в процессе создания программного продукта для работы с алгоритмами получения простых чисел в ООО «Удостоверяющий центр Сибири», комбинирование полученных результатов проверки генерируемых чисел с помощью теста простоты числа с результатами проверки теста, основанного на полученных с помощью разработанного программного обеспечения критериев простоты числа, позволило

сократить от 5 до 7% количество необнаруженных псевдопростых чисел, что является значимым результатом при обработке больших выборок чисел.

Положения, выносимые на защиту:

1. Модифицированный метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ позволяет строить алгоритмы ранжирования и генерации по рангу для таких комбинаторных множеств, для которых не известно выражение функции мощности, принадлежащее алгебре $\{\mathbb{N}, +, \times, R\}$, но известно выражение производящей функции для последовательности значений функции мощности.

Соответствует п. 3 паспорта специальности 05.13.17: Исследование методов и разработка средств кодирования информации в виде данных. Принципы создания языков описания данных, языков манипулирования данными, языков запросов. Разработка и исследование моделей данных и новых принципов их проектирования.

2. Оценка вычислительной сложности разработанных алгоритмов ранжирования и генерации по рангу для множества комбинаторных объектов, отражающих вторичную структуру РНК длины n с m пар нуклеотидов, соединенных водородной связью, равна $O(m^2(n-m))$, при этом не требуется вспомогательных предварительных вычислений.

Соответствует п. 3 паспорта специальности 05.13.17: Исследование методов и разработка средств кодирования информации в виде данных. Принципы создания языков описания данных, языков манипулирования данными, языков запросов. Разработка и исследование моделей данных и новых принципов их проектирования.

3. Оценка вычислительной сложности разработанных алгоритмов ранжирования и генерации по рангу для множества комбинаторных объектов, определяемых числовым треугольником Эйлера-Каталана EC_n^m, равна $O(nm(n-m))$, при этом не требуется вспомогательных предварительных вычислений.

Соответствует п. 3 паспорта специальности 05.13.17: Исследование методов и разработка средств кодирования информации в виде данных. Принципы создания языков описания данных, языков манипулирования данными, языков запросов. Разработка и исследование моделей данных и новых принципов их проектирования.

4. Алгоритмы комбинаторной генерации для создания программного обеспечения, автоматизирующего процесс ранжирования и генерации по рангу для комбинаторных множеств.

Соответствует п. 14 паспорта специальности 05.13.17: Разработка теоретических основ создания программных систем для новых информационных технологий.

Достоверность результатов. Достоверность результатов диссертационной работы обеспечивается сравнением разработанных алгоритмов с аналогичными алгоритмами других авторов, проверкой теоретических положений вычислительными экспериментами, положительным эффектом от внедрения полученных результатов.

Внедрение результатов работы. Результаты диссертационной работы использованы в ходе выполнения следующих научно-исследовательских работ:

– базовая часть государственного задания Министерства науки и высшего образования РФ (проект № 2.8172.2017/8.9 «Метод и модели определения уровня защищенности информационных систем» на 2017–2019 гг.);

– грант «Российского научного фонда» (проект № 18-71-00059 «Разработка алгоритмов и программного обеспечения индексирования больших объемов данных на основе новых методов комбинаторной генерации» на 2018–2020 гг.);

– грант «Российского фонда фундаментальных исследований» (проект № 18-31-00201 «Методы комбинаторной генерации на основе деревьев И/ИЛИ с применением теории производящих функций» на 2018–2019 гг.).

Результаты диссертационной работы использованы при систематизации архива проведения экспериментальных исследований в ООО «ПлантаПлюс», а также в процессе создания программного продукта для работы с алгоритмами получения простых чисел в ООО «Удостоверяющий центр Сибири».

Результаты диссертационной работы внедрены в учебный процесс ФГБОУ ВО «ТУСУР» на факультете безопасности и используются при изучении дисциплин «Дискретная математика» и «Теория игр и исследование операций».

Личный вклад автора. Постановка цели и задач научного исследования осуществлялась совместно с научным руководителем. Автором разработан модифицированный метод построения алгоритмов комбинаторной генерации на основе применения теории производящих функций; разработаны алгоритмы комбинаторной генерации для множества комбинаторных объектов, отражающих выражения обобщенного языка Дика, для множества комбинаторных объектов, отражающих вторичную структуру РНК, и для множества комбинаторных объектов, определяемых треугольником Эйлера-Каталана; создано программное обеспечение для ранжирования и генерации по рангу элементов комбинаторных множеств.

Апробация работы. Основные результаты диссертационной работы докладывались на следующих конференциях:

1. Международная научно-практическая конференция «Электронные средства и системы управления» (2015–2018 гг., г. Томск, ТУСУР).

2. Международная научно-техническая конференция студентов, аспирантов и молодых ученых «Научная сессия ТУСУР» (2016–2019 гг., г. Томск, ТУСУР).

3. The 3rd Algorithmic and Enumerative Combinatorics Summer School (1–5 августа 2016 г., Австрия, г. Хагенберг, Университет им. И. Кеплера).

4. The Mediterranean International Conference of Pure & Applied Mathematics and Related Areas (26–29 октября 2018 г., Турция, г. Анталья, Университет Акдениз).

5. XII Всероссийская научная конференция «Наука. Технологии. Инновации» (3–7 декабря 2018 г., г. Новосибирск, НГТУ).

6. V Международная научно-практическая конференция молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук» (22–24 апреля 2019 г., г. Тольятти, ТГУ).

7. The 32th International Conference of the Jangjeon Mathematical Society (17–19 июля 2019 г., г. Владивосток, ДВФУ).

8. The 2nd Mediterranean International Conference of Pure & Applied Mathematics and Related Areas (28–31 августа 2019 г., Франция, г. Париж, Университет Эври).

9. Томский IEEE семинар «Интеллектуальные системы моделирования, проектирования и управления» (2015–2019 гг., г. Томск, ТУСУР).

Публикации по теме диссертации. Основные результаты диссертационного исследования опубликованы в 21 работе, в том числе 4 публикации в рецензируемых журналах из перечня ВАК [30–33], 6 публикаций в научных журналах, индексируемых Web of Science и Scopus [34–39], 9 публикаций в тезисах и материалах научных конференций [40–48], получены 3 свидетельства о государственной регистрации программы для ЭВМ [49–51].

Объем и структура работы. Диссертация состоит из введения, четырех глав основной части, заключения, списка литературы и двух приложений. Полный объем диссертации составляет 123 страницы, включая 29 рисунков и 8 таблиц. Список литературы содержит 102 наименования.

Глава 1. Анализ современного состояния исследований в области разработки алгоритмов комбинаторной генерации

Комбинаторное множество — это конечное множество, элементы которого имеют некоторую структуру и имеется процедура построения элементов этого множества [26]. Элементы комбинаторных множеств (комбинаторные объекты), таких как сочетания, перестановки, размещения, разбиения, графы, деревья и так далее, играют важную роль в математике и информатике, а также имеют множество приложений [1; 2].

Д. Кнут [1] приводит обзор становления и развития направления, связанного с разработкой различного рода комбинаторных алгоритмов. При этом особое внимание уделяется задаче обхода всех возможных элементов некоторого комбинаторного множества, которую можно рассматривать как:

– перечисление всех комбинаторных объектов (enumerating) — подсчет общего количества элементов рассматриваемого комбинаторного множества;

– составление списка всех комбинаторных объектов (listing) — запись полного списка всех элементов рассматриваемого комбинаторного множества;

– генерация всех комбинаторных объектов (generating) — генерация всех требующихся элементов рассматриваемого комбинаторного множества и их поочередное посещение.

Отдельно рассмотрим генерацию комбинаторных объектов, так как при разработке комбинаторных алгоритмов зачастую необходимо, чтобы разрабатываемая программа могла поочередно представить требуемые комбинаторные объекты в памяти компьютера в виде некоторой структуры данных и их обработать [46]. Ф. Раски [2] вводит понятие комбинаторной генерации и выделяет в рамках данного направления следующие четыре задачи:

– listing — последовательная генерация элементов рассматриваемого комбинаторного множества;

– ranking — ранжирование (нумерация) элементов рассматриваемого комбинаторного множества;

– unranking — генерация элементов рассматриваемого комбинаторного множества в соответствии с их рангами (номерами);

– random selection — генерация элементов рассматриваемого комбинаторного множества в случайном порядке.

Последовательная генерация комбинаторных объектов обычно реализуется в виде функции Next. Данная функция получает на вход текущий комбинаторный объект и, возможно, некоторую дополнительную информацию, а на выходе генерирует следующий за ним комбинаторный объект. Генерация всех элементов рассматриваемого комбинаторного множества может быть осуществлена начиная с любого комбинаторного объекта или начиная с определенного комбинаторного объекта, задаваемого некоторой вспомогательной функцией First.

Ранжирование (нумерация) заключается в присвоении комбинаторным объектам рангов. Ранг комбинаторного объекта — это порядковый номер, указывающий позицию данного комбинаторного объекта в рамках некоторого упорядочения элементов комбинаторного множества. Соответствующий алгоритм ранжирования комбинаторных объектов обозначается через Rank [52–55]. Формальная запись работы алгоритма Rank:

$$r = \texttt{Rank}(a),$$

где $a \in A$ — элемент комбинаторного множества A; $r = 1, \ldots, |A|$ — ранг комбинаторного объекта a.

Алгоритм, который позволяет произвести обратное действие ранжированию, обозначается через Unrank [52–55]. С помощью алгоритма Unrank становится возможной генерация элементов рассматриваемого комбинаторного множества в соответствии с их рангами: зная ранг r комбинаторного объекта a, можно однозначно воспроизвести сам комбинаторный объект a. Формальная запись работы алгоритма Unrank:

$$a = \texttt{Unrank}(r).$$

Использование алгоритма Unrank делает возможным не только быструю генерацию комбинаторного объекта с заданным рангом, но и последовательную генерацию всех элементов рассматриваемого комбинаторного множества, а также их генерацию в случайном порядке. Последовательная генерация может быть реализована путем вызова алгоритма Unrank в цикле, который последовательно обходит все возможные значения рангов комбинаторных объектов. Генерация в случайном порядке может быть реализована путем вызова алгоритма Unrank для значений рангов, полученных с помощью генератора случайных чисел.

1.1 Методы построения алгоритмов комбинаторной генерации

Среди основных подходов к разработке алгоритмов комбинаторной генерации можно выделить:

- метод поиска с возвратом (backtracking) [4; 5];
- ECO-метод [6; 8];
- метод Ф. Флажоле [15; 17];
- метод Б.Я. Рябко [21; 22];
- метод В.В. Кручинина [25; 26].

Также существует множество алгоритмов комбинаторной генерации, при разработке которых использовались особенности рассматриваемого комбинаторного множества (примеры таких алгоритмов можно найти в работах Д. Кнута [1] и Ф. Раски [2]). Методы разработки таких алгоритмов носят эвристический характер и поэтому не могут быть универсальными, то есть их невозможно применить в полном объеме при разработке алгоритмов комбинаторной генерации для других комбинаторных множеств.

Рассмотрим подробно принцип работы каждого из приведенных основных подходов к разработке алгоритмов комбинаторной генерации с указанием имеющихся у них недостатков.

1.1.1 Метод поиска с возвратом

Для генерации комбинаторных объектов Э. Рейнгольд, Ю. Нивергельт и Н. Део в работе «Комбинаторные алгоритмы: Теория и практика» [4] предлагают использовать идею исчерпывающего поиска с возвратом (backtracking). Данный метод носит общий характер и находит свое применение в широком классе задач поиска.

Метод поиска с возвратом основан на последовательном расширении текущего частичного решения. При условии наступления ситуации, когда дальнейшее расширение текущего частичного решения невозможно, необходимо произвести возврат к частичному решению меньшего размера и попробовать расширить его. Поиск останавливается тогда, когда частичное решение больше нельзя расширить. Выполнение указанных действий позволяет перебрать все возможные достижимые варианты решения поставленной задачи.

Классическим примером задачи, которую можно решить с помощью алгоритма поиска с возвратом, можно назвать задачу о n ферзях [4]. Данная задача заключается в поиске возможных вариантов размещения n ферзей на поле доски размером $n \times n$, чтобы ни один из них не находился под боем другого.

Формализуем работу алгоритма поиска с возвратом для общего случая. Пусть допустимое решение задачи состоит из слова $(a_1, a_2, \ldots, a_k)$, которое удовлетворяет некоторым заданным ограничениям (некоторый предикат P). При этом каждое a_i является элементом конечного упорядоченного множества A_i. Работа алгоритма заключается в определении подмножества S_k множества A_k при заданных ограничениях, элементы которого являются кандидатами в a_k при заданных $(a_1, a_2, \ldots, a_{k-1})$, то есть возможно расширение текущего частичного решения из $(a_1, a_2, \ldots, a_{k-1})$ в $(a_1, a_2, \ldots, a_{k-1}, a_k)$. Если таких кандидатов в a_k нет ($S_k = \varnothing$), то возвращаемся и пробуем выбрать нового кандидата в a_{k-1}, и так далее.

Реализация такого алгоритма возможна как без использования рекурсии, так и с рекурсией (Алгоритм 1). При этом работа рекурсивного алгоритма начинается с вызова `Backtraking`$(1, ())$.

Алгоритм 1: Алгоритм поиска с возвратом (с рекурсией)

```
1 Backtracking (k, (a_1, a_2, ..., a_{k-1}))
2 begin
3 |   if P(a_1, a_2, ..., a_{k-1}) then Вывод (a_1, a_2, ..., a_{k-1})
4 |   S_k := Вычислить S_k
5 |   foreach a_k ∈ S_k do Backtracking (k + 1, (a_1, a_2, ..., a_{k-1}, a_k))
6 end
```

Для оптимизации алгоритма необходимо реализовать вычисления таким образом, чтобы программа не рассматривала заведомо недостижимые решения, тем самым удастся значительно сократить время нахождения всех решений. Такой результат можно получить с помощью дополнительного исследования поставленной задачи и использования некоторых эвристик при реализации алгоритма. Например, в работе Д. Крехера и Д. Стинсона [5] предложена оптимизация алгоритма поиска с возвратом за счет введения ограничивающих функций.

Применение метода поиска с возвратом для построения алгоритмов генерации комбинаторных объектов характеризуется следующими недостатками:

– метод применим только для последовательной генерации комбинаторных объектов;

– метод носит идейный характер (представлены только общие принципы работы метода без каких-либо конкретных инструкций по разработке алгоритмов комбинаторной генерации), поэтому требует больших усилий при разработке алгоритмов комбинаторной генерации для конкретных комбинаторных объектов.

1.1.2 ECO-метод

Е. Баркуччи, А. Дель Лунго, Э. Пергола и Н. Пинзани [6] предложили метод, который также претендует на некоторую универсальность применения при разработке алгоритмов комбинаторной генерации. В их работе описан общий метод, названный ECO-методом (Enumeration Combinatorial Object — перечислительный комбинаторный объект), который позволяет совершить перечисление всех комбинаторных объектов некоторого комбинаторного множества.

В ECO-методе используется идея перехода от комбинаторного объекта размера k к объекту размера $k+1$. Для перечисления всех комбинаторных объектов вводится набор правил вывода, на основе которых строится генерирующее дерево. Уровень дерева соответствует размерности комбинаторного объекта. В вершинах генерирующего дерева на k-ом уровне записывается число сыновей, а общее количество вершин на k-ом уровне дерева соответствует количеству комбинаторных объектов размера k.

Представим формальное описание ECO-метода. Пусть A — комбинаторное множество, а A_n — множество комбинаторных объектов размера n. Определим оператор θ над множеством A как следующее семейство функций:

$$\theta : A_n \to 2^{A_{n+1}},$$

где $2^{A_{n+1}}$ является множеством всех подмножеств A_n.

Если оператор θ удовлетворяет следующим условиям:

– для каждого $Y \in A_{n+1}$ существует $X \in A_n$ такой, что $Y \in \theta(X)$;

– если $X_1, X_2 \in A_n$ и $X_1 \neq X_2$, то $\theta(X_1) \cap \theta(X_2) = \varnothing$,

тогда $F_{n+1} = \{\theta(X) : X \in A_n\}$ является разбиением A_{n+1}, а θ называется ECO-оператором.

В соответствии с ECO-оператором вводится понятие ECO-правила Ω (или правило наследования — succession rule), которое представляет собой систему из корня (e_0) и множества правил вывода вида

$$(k) \rightsquigarrow (e_1(k))\,(e_2(k))\ldots(e_k(k))\,,$$

где e_0 и каждое $e_i(k), 1 \leqslant i \leqslant k$, являются натуральными числами, а каждое $e_i(k)$ показывает как получить всех сыновей $(e_1(k))\,(e_2(k))\ldots(e_k(k))$ для заданной метки (k). Более компактная запись ECO-правила:

$$\Omega : \begin{cases} (e_0), \\ (k) \rightsquigarrow (e_1(k))\,(e_2(k))\ldots(e_k(k))\,. \end{cases}$$

Дальнейшее развитие ECO-метода предложено в работе С. Баччелли, Е. Баркуччи, Э. Грацини и Э. Пергола [8]. В данной работе описан алгоритм исчерпывающей генерации комбинаторных объектов, базирующийся на идеях ECO-метода (Алгоритм 2).

Алгоритм 2: Алгоритм последовательной генерации комбинаторных объектов с помощью ECO-метода

```
1  ECO ()
2  begin
3      На основе ECO-правила Ω и размерности n комбинаторного объекта
         построить начальный комбинаторный объект в виде слова
         (a_1, a_2, ..., a_n)
4      while Существует такое a_i, которое можно изменить do
5          Найти такое a_i, которое можно изменить и у которого наибольший
             индекс i
6          if i ≠ 1 then
7              В соответствии с ECO-правилом Ω обновить значение a_i и
                 задать значения a_j (j = i + 1, ..., n) равными первому
                 элементу правила вывода из a_{j-1}
8              Вывод (a_1, a_2, ..., a_n)
9          end
10     end
11 end
```

В основе Алгоритма 2 лежит идея генерации начального комбинаторного объекта заданной размерности и последующий вывод из него остальных комбинаторных объектов на основе ECO-правила.

Несмотря на идейный характер, ECO-метод нашел свое применение для генерации объектов большого числа комбинаторных множеств, например:

– нечетные числа Фибоначчи, числа Каталана, факториалы [8];

– некоторые классы полимино [10; 11];

– перестановки, беспорядки, инволюции [12].

Применение ECO-метода для построения алгоритмов генерации комбинаторных объектов характеризуется следующими недостатками:

– метод применим только для последовательной генерации комбинаторных объектов;

– необходимо задать ECO-правило Ω;

– необходимо задать правила биективного отображения между комбинаторным множеством и множеством слов, кодирующих объекты данного множества;

– метод применим только для таких комбинаторных объектов, которые описываются лишь одним параметром — размерность n;

– метод носит идейный характер, поэтому требует больших усилий при разработке алгоритмов комбинаторной генерации для конкретных комбинаторных объектов.

1.1.3 Метод Ф. Флажоле

В работе Ф. Флажоле, П. Циммермана и Б. Катсема [15] приводится собственный алгоритм генерации в случайном порядке для помеченных комбинаторных объектов. Для применения данного алгоритма необходимо знать экспоненциальную производящую функцию, описывающую количество комбинаторных объектов в зависимости от заданной размерности объекта. При этом требуется получить декомпозицию производящей функции с использованием следующих допустимых операторов и на ее основе описать спецификацию класса комбинаторных объектов:

– ε — пустая структура размера 0;

– Z — одиночная структура размера 1;

– $\mathcal{A} + \mathcal{B}$ — операция объединения классов $\mathcal{A}$ и $\mathcal{B}$;

– $\mathcal{A} \star \mathcal{B}$ — операция произведения классов $\mathcal{A}$ и $\mathcal{B}$;

– $\texttt{Seq}(\mathcal{A})$ — операция формирования последовательностей из элементов класса $\mathcal{A}$;

– $\texttt{Set}(\mathcal{A})$ — операция формирования подмножеств из элементов класса $\mathcal{A}$ (возможно наличие ограничений на мощность формируемых подмножеств);

– $\texttt{Cycle}(\mathcal{A})$ — операция формирования ориентированных циклов из элементов класса $\mathcal{A}$.

Расширение данного метода представлено в цикле работ К. Мартинеса и К. Мулинеро [17–19]. В указанных работах описан перечень следующих дополнений к алгоритму:

– использование обыкновенных производящих функций для непомеченных комбинаторных объектов;

– использование дополнительных операторов, таких как $\mathcal{A} \times \mathcal{B}$, $\texttt{Subst}(\mathcal{A})$ и $\texttt{PowerSet}(\mathcal{A})$;

– расширение метода до возможности разработки алгоритмов последовательной генерации комбинаторных объектов, а также алгоритмов ранжирования и генерации в соответствии с их рангами.

Таблица 1.1 — Комбинаторные операторы и соответствующие им обыкновенные и экспоненциальные производящие функции

Комбинаторный оператор	Обыкновенная производящая функция	Экспоненциальная производящая функция
ε	1	1
Z	z	z
$\mathcal{A} + \mathcal{B}$	$A(z) + B(z)$	$A(z) + B(z)$
$\mathcal{A} \times \mathcal{B}$	$A(z) \cdot B(z)$	
$\mathcal{A} \star \mathcal{B}$		$A(z) \cdot B(z)$
$\texttt{Seq}(\mathcal{A})$	$\frac{1}{1-A(z)}$	$\frac{1}{1-A(z)}$
$\texttt{Set}(\mathcal{A})$	$\exp\left(\sum\limits_{n>0} \frac{A(z^n)}{n}\right)$	$\exp\left(A(z)\right)$
$\texttt{PowerSet}(\mathcal{A})$	$\exp\left(\sum\limits_{n>0} (-1)^{n-1} \frac{A(z^n)}{n}\right)$	$\exp\left(A(z)\right)$
$\texttt{Cycle}(\mathcal{A})$	$\sum\limits_{n>0} \frac{\varphi(n)}{n} \log\left(\frac{1}{1-A(z^n)}\right)$	$\log\left(\frac{1}{1-A(z)}\right)$
$\texttt{Subst}(\mathcal{A},\mathcal{B})$	$A(B(z))$	$A(B(z))$

Применение метода Ф. Флажоле для построения алгоритмов генерации комбинаторных объектов характеризуется следующими недостатками:

– необходимо знать выражение функции мощности комбинаторного множества;

– необходимо знать выражение производящей функции, описывающей количество комбинаторных объектов в зависимости от заданной размерности n комбинаторного объекта;

– необходимо вручную задать декомпозицию производящей функции с использованием допустимых операторов и на ее основе описать спецификацию класса комбинаторных объектов (если спецификация не была получена, то дальнейшая разработка алгоритмов комбинаторной генерации данным методом не является возможной);

– для каждого оператора, использованного в спецификации, необходимо задать правила генерации конкретного комбинаторного объекта;

– метод применим только для таких комбинаторных объектов, которые описываются лишь одним параметром — размерность n (так как рассматривается только случай одномерных производящих функций).

1.1.4 Метод Б.Я. Рябко

Для организации ранжирования комбинаторных объектов и их генерации в соответствии с их рангами в работах Б.Я. Рябко [21; 22] предлагается метод быстрого кодирования и декодирования слов. При этом каждое слово является представлением конкретного комбинаторного объекта заданной размерности.

Представим формальное описание метода Б.Я. Рябко. Пусть S — некоторый алфавит, S^n — множество всех слов длины n, полученных с помощью алфавита S. Рассмотрим подмножество $A \subset S^n$, которое представляет собой некоторое множество комбинаторных объектов размера n. Каждому комбинаторному объекту размера n соответствует слово $a = a_1 a_2 \dots a_n$, где $a \in A$ и $a_i \in S$. При этом для рассматриваемого комбинаторного объекта необходимо задать следующие вспомогательные функции:

$$P(a_i|a_1 \dots a_{i-1}) = \frac{N_A(a_1 \dots a_i)}{N_A(a_1 \dots a_{i-1})}, \quad P(a_1) = \frac{N_A(a_1)}{|A|},$$

$$q(a_i|a_1 \dots a_{i-1}) = \sum_{x<a_i} P(x|a_1 \dots a_{i-1}), \quad q(a_1) = \sum_{x<a_1} P(x),$$

где $N_A(a_1 \dots a_i)$ — количество слов множества A с префиксом $a_1 \dots a_i$.

Тогда ранг r комбинаторного объекта можно получить в виде порядкового номера, соответствующего объекту слова $a_1 \dots a_n$, в рамках лексикографического упорядочения множества A. При этом формула вычисления будет следующей:

$$r(a_1 \dots a_n) = |A|((q(a_1) + q(a_2|a_1)P(a_1)) + ((q(a_3|a_1a_2)+$$

$$+q(a_4|a_1a_2a_3)P(a_3|a_1a_2))P(a_2|a_1)P(a_1)) + \dots).$$

Также вводятся следующие вспомогательные обозначения:

$$\lambda_k^s = \lambda_{2k-1}^{s-1} + \rho_{2k-1}^{s-1} \cdot \lambda_{2k}^{s-1}, \quad \lambda_k^0 = q(a_i|a_1 \dots a_{i-1}), \quad \lambda_1^0 = q(a_1),$$

$$\rho_k^s = \rho_{2k-1}^{s-1} \cdot \rho_{2k}^{s-1}, \quad \rho_k^0 = P(a_i|a_1 \dots a_{i-1}), \quad \rho_1^0 = P(a_1),$$

Тогда для алгоритма ранжирования комбинаторных объектов методом Б.Я. Рябко можно применить формулу

$$\texttt{Ryabko\_Rank}(a_1a_2 \dots a_n) = |A|\lambda_1^{\lceil \log_2 n \rceil}.$$

Алгоритм генерации слова на основе известного ранга основан на обратных преобразованиях. Модификация данного алгоритма представлена в диссертации Ю.С. Медведевой [24].

В общем виде алгоритм генерации комбинаторных объектов по рангу можно записать как:

Алгоритм 3: Алгоритм генерации комбинаторных объектов по рангу с помощью метода Б.Я. Рябко

```
1  Ryabko_Unrank (r)
2  begin
3      z := r/|A|
4      for k := 1 to n do
5          Найти x_k, для которого λ_k^0|_{a_k:=x_k} ⩽ z < λ_k^0|_{a_k:=x_k+1}
6          a_k := x_k
7          z := (z − λ_k^0)/ρ_k^0
8      end
9      return (a_1 a_2 ... a_n)
10 end
```

Применение метода Б.Я. Рябко для построения алгоритмов генерации комбинаторных объектов характеризуется следующими недостатками:

– необходимо знать выражение функции мощности комбинаторного множества;

– необходимо знать выражение вспомогательных функций, описывающих количество слов, кодирующих комбинаторные объекты, с заданным префиксом;

– необходимо задать правила биективного отображения между комбинаторным множеством и множеством слов, кодирующих объекты данного множества.

1.1.5 Метод В.В. Кручинина

В работах В.В. Кручинина [25; 26] предлагается метод разработки алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ. Данный метод базируется на представлении комбинаторных множеств в виде структуры дерева И/ИЛИ, с помощью которого можно построить алгоритмы последовательной генерации комбинаторных объектов, а также их ранжирования и генерации в соответствии с их рангами.

Чтобы применить данный метод необходимо для рассматриваемого комбинаторного множества построить дерево И/ИЛИ, число вариантов которого должно совпадать со значением функции мощности комбинаторного множества. В работах [25; 26] показана возможность построения таких деревьев И/ИЛИ для комбинаторных множеств, для которых известно выражение функции мощности f, которая принадлежит алгебре $\{\mathbb{N}, +, \times, R\}$, где R — оператор примитивной рекурсии.

Для ранжирования комбинаторного объекта, представленного в виде структуры варианта v дерева И/ИЛИ D, необходимо предварительно провести сопоставление варианта v дерева D с самим деревом. Ранг комбинаторного объекта определяется значением функции $l(z)$ для узла z варианта v, который в дереве D является корнем. При этом значение функции $l(z)$ для каждого варианта v дерева И/ИЛИ D является уникальным, что делает возможным однозначное сопоставление между вариантами дерева И/ИЛИ и значениями функции $l(z)$.

Значение $l(z)$ соответствует некоторому номеру узла z, для которого выполняется условие

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

где $w(z)$ — число вариантов в поддереве узла z, и вычисляется по следующим правилам:

Если узел z варианта v в дереве D является листом, то

$$l(z) = 0.$$

Если узел z варианта v в дереве D является И-узлом, то

$$l(z) = l\left(s_1^{(z)}\right) + w\left(s_1^{(z)}\right) \cdot$$

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

где n — количество сыновей узла z, $s_i^{(z)}$ — i-й сын узла z.

Если узел z варианта v в дереве D является ИЛИ-узлом, то:

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

где k — порядковый номер сына узла z среди всех его сыновей, который выбран в варианте v.

Алгоритм генерации комбинаторного объекта в соответствии с заданным рангом основан на обратных преобразованиях.

Эффективность этого метода показана на генерации объектов достаточно большого числа комбинаторных множеств: сочетания, разложения, числа Фибоначчи, разбиения, композиции, перестановки, числа Каталана, деревья, выражения некоторых языков [26].

Применение метода В.В. Кручинина для построения алгоритмов генерации комбинаторных объектов характеризуется следующими недостатками:

– необходимо знать выражение функции мощности комбинаторного множества, которое принадлежит алгебре $\{\mathbb{N}, +, \times, R\}$;

– необходимо задать правила биективного отображения между комбинаторным множеством и множеством вариантов дерева И/ИЛИ, кодирующих объекты данного множества.

1.2 Сравнение методов построения алгоритмов комбинаторной генерации на примере множеств перестановок и сочетаний

Рассмотрим подробно процесс разработки алгоритмов комбинаторной генерации с помощью каждого метода для таких классических комбинаторных множеств, как перестановки и сочетания [45].

Перестановка n элементов — это упорядоченный набор всех элементов множества n различных элементов [56]. Если взять множество элементов и переставить все элементы местами, то все возможные такие наборы и будут составлять множество всех перестановок. Функция мощности множества всех перестановок n элементов определяется выражением

$$P_n = n!.$$

Сочетание из n элементов по m — это набор m элементов, выбранных из множества n различных элементов [56]. Наборы, отличающиеся только порядком следования элементов, считаются одинаковыми. Функция мощности множества всех сочетаний из n элементов по m определяется выражением

$$C_n^m = \frac{n!}{m!(n-m)!}.$$

1.2.1 Метод поиска с возвратом

Для метода поиска с возвратом возможна только разработка алгоритмов последовательной генерации комбинаторных объектов. На основе общего алгоритма метода поиска с возвратом был получен следующий алгоритм последовательной генерации множества перестановок n элементов (Алгоритм 4).

Для запуска выполнения Алгоритма 4 необходимо вызвать команду `Backtracking_PermutationGen`(1,()). Данный алгоритм можно представить как левосторонний обход генерирующего дерева, приведенного на рисунке 1.1.

Рассмотрим ход работы алгоритма для частного случая $n = 3$:

$$\varnothing \to \{1\} \to \{1,2\} \to \mathbf{\{1,2,3\}} \to \{1,2\} \to \{1\} \to \{1,3\} \to \mathbf{\{1,3,2\}} \to$$
$$\to \{1,3\} \to \{1\} \to \varnothing \to \{2\} \to \{2,1\} \to \mathbf{\{2,1,3\}} \to \{2,1\} \to \{2\} \to$$
$$\to \{2,3\} \to \mathbf{\{2,3,1\}} \to \{2,3\} \to \{2\} \to \varnothing \to \{3\} \to \{3,1\} \to \mathbf{\{3,1,2\}} \to$$
$$\to \{3,1\} \to \{3\} \to \{3,2\} \to \mathbf{\{3,2,1\}} \to \{3,2\} \to \{3\} \to \varnothing$$

Алгоритм 4: Алгоритм последовательной генерации множества перестановок, полученный с помощью метода поиска с возвратом

```
1  Backtracking_PermutationGen (k, (a_1, a_2, ..., a_{k-1}))
2  begin
3      if k > n then Вывод (a_1, a_2, ..., a_{k-1})
4      else
5          for i := 1 to n do
6              if i ∉ {a_1, a_2, ..., a_{k-1}} then
7                  Backtracking_PermutationGen (k + 1, (a_1, a_2, ..., a_{k-1}, i))
8              end
9          end
10     end
11 end
```

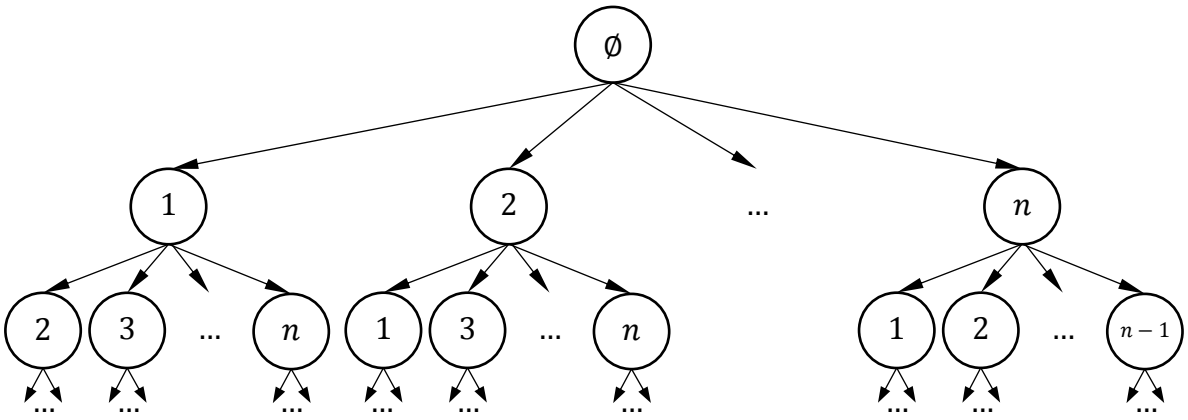


Рисунок 1.1 — Генерирующее дерево для множества перестановок

Оценка вычислительной сложности Алгоритма 4 равна $O(n!)$.

Аналогично был получен алгоритм последовательной генерации множества сочетаний из n элементов по m (Алгоритм 5).

Для запуска выполнения Алгоритма 5 необходимо вызвать команду `Backtracking_CombinationGen`(1,()). Данный алгоритм можно представить как левосторонний обход генерирующего дерева, приведенного на рисунке 1.2.

Рассмотрим ход работы алгоритма для частного случая $n = 4, m = 2$:

$$\varnothing \to \{1\} \to \mathbf{\{1,2\}} \to \{1\} \to \mathbf{\{1,3\}} \to \{1\} \to \mathbf{\{1,4\}} \to \{1\} \to \varnothing \to$$
$$\to \{2\} \to \mathbf{\{2,3\}} \to \{2\} \to \mathbf{\{2,4\}} \to \{2\} \to \varnothing \to \{3\} \to \mathbf{\{3,4\}} \to \{3\} \to \varnothing$$

Оценка вычислительной сложности Алгоритма 5 равна $O(C_n^m)$.

Алгоритм 5: Алгоритм последовательной генерации множества сочетаний, полученный с помощью метода поиска с возвратом

```
1  Backtracking_CombinationGen (k, (a_1, a_2, ..., a_{k-1}))
2  begin
3      if k > m then Вывод (a_1, a_2, ..., a_{k-1})
4      else
5          if k = 1 then i_0 := 1
6          else i_0 := a_{k-1} + 1
7          for i := 1 to n − m + k do
8              Backtracking_CombinationGen (k + 1, (a_1, a_2, ..., a_{k-1}, i))
9          end
10     end
11 end
```

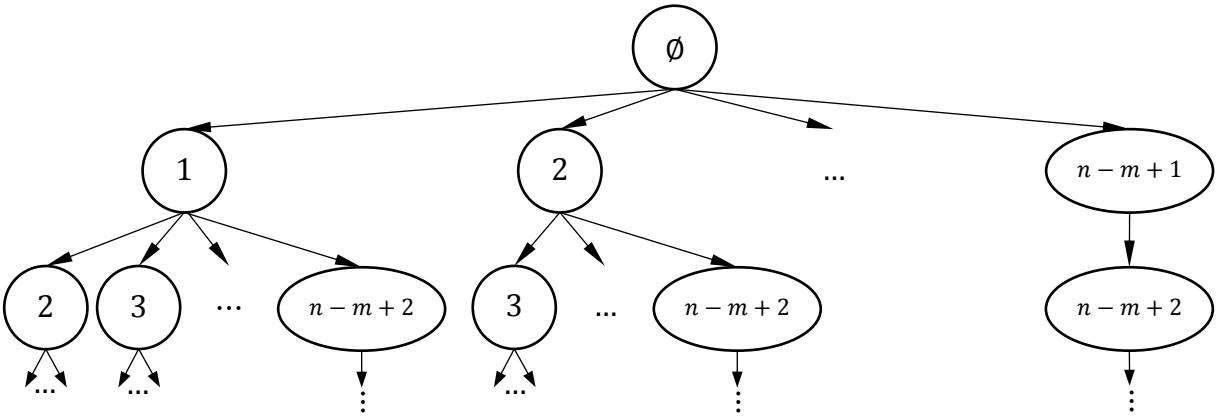


Рисунок 1.2 — Генерирующее дерево для множества сочетаний

1.2.2 ECO-метод

Для ECO-метода возможна только разработка алгоритмов последовательной генерации комбинаторных объектов. На основе общего алгоритма ECO-метода с помощью ECO-правила

$$\Omega : \begin{cases} (2), \\ (k) \rightsquigarrow (k+1)^k \end{cases}$$

был получен алгоритм последовательной генерации множества перестановок n элементов (Алгоритм 6).

Алгоритм 6: Алгоритм последовательной генерации множества перестановок, полученный с помощью ECO-метода

```
1 ECO_PermutationGen (k, (a_1, a_2, ..., a_{k-1}))
2 begin
3     if k > n then Вывод (a_1, a_2, ..., a_{k-1})
4     else
5         for i := 1 to k do
6             ECO_PermutationGen (k + 1, (a_1, ..., a_{k-i}, k, a_{k-i+1}, ..., a_{k-1}))
7         end
8     end
9 end
```

Для запуска выполнения Алгоритма 6 необходимо вызвать команду `ECO_PermutationGen`(1,()). Данный алгоритм можно представить как левосторонний обход генерирующего дерева, приведенного на рисунке 1.3.

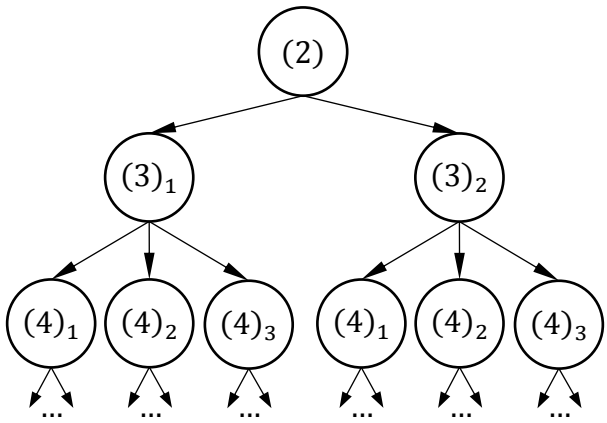


Рисунок 1.3 — Генерирующее дерево для множества перестановок

Рассмотрим ход работы алгоритма для частного случая $n = 3$:

Путь генерирующего дерева:	Полученная перестановка:
$(2)(3)_1(4)_1$	123
$(2)(3)_1(4)_2$	132
$(2)(3)_1(4)_3$	312
$(2)(3)_2(4)_1$	213
$(2)(3)_2(4)_2$	231
$(2)(3)_2(4)_3$	321

Оценка вычислительной сложности Алгоритма 6 равна $O(n!)$.

Применение ECO-метода для получения алгоритма последовательной генерации множества сочетаний из n элементов по m является невозможным, так как данный комбинаторный объект описывается двумя параметрами.

1.2.3 Метод Ф. Флажоле

Для множества перестановок n элементов известно следующее выражение экспоненциальной производящей функции:

$$P(z) = \sum_{n \geqslant 0} P_n \frac{z^n}{n!} = \frac{1}{1-z}.$$

Данная производящая функция соответствует следующей спецификации:

$$\mathcal{P} = \mathtt{Seq}(Z).$$

Известна оценка вычислительной сложности получаемых с помощью данного метода алгоритмов генерации комбинаторных объектов по известному рангу, которая равна $O(n^2)$ [57]. Данная оценка не учитывает временные затраты на реализацию вычислений в соответствии с правилами генерации конкретного комбинаторного объекта.

Применение метода Ф. Флажоле для получения алгоритмов комбинаторной генерации множества сочетаний из n элементов по m является невозможным, так как данный комбинаторный объект описывается двумя параметрами.

1.2.4 Метод Б.Я. Рябко

Для разработки алгоритмов комбинаторной генерации для множества перестановок n элементов были получены следующие выражения вспомогательных функций:

$$P(a_i | a_1 \dots a_{i-1}) = \frac{1}{n-i+1}, \quad P(a_1) = \frac{1}{n},$$

$$q(a_i | a_1 \dots a_{i-1}) = \frac{a_i - 1 - r_i}{n-i+1}, \quad q(a_1) = \frac{a_1 - 1}{n},$$

где r_i — количество значений в префиксе $a_1 \dots a_{i-1}$ меньших, чем a_i.

При этом упрощенная запись формулы вычисления ранга имеет вид

$$\mathtt{Ryabko\_PermutationRank}(a_1 a_2 \dots a_n) = \sum_{i=1}^{n} (a_i - 1 - r_i)(n-i)!.$$

Для разработки алгоритмов комбинаторной генерации для множества сочетаний из n элементов по m были получены следующие выражения вспомогательных функций:

Способ 1: Сочетание из n элементов по m представлено в виде слова $a_1a_2\dots a_m$, где a_i — значение выбранного элемента. Тогда

$$P(a_i|a_1\dots a_{i-1}) = \frac{C_{n-a_i}^{m-i}}{C_{n-a_{i-1}}^{m-i+1}}, \quad P(a_1) = \frac{C_{n-a_1}^{m-1}}{C_n^m},$$

$$q(a_i|a_1\dots a_{i-1}) = \sum_{x=a_{i-1}+1}^{a_1-1} \frac{C_{n-x}^{m-i}}{C_{n-a_{i-1}}^{m-i+1}}, \quad q(a_1) = \sum_{x=1}^{a_1-1} \frac{C_{n-x}^{m-1}}{C_n^m}.$$

При этом упрощенная запись формулы вычисления ранга имеет вид

$$\texttt{Ryabko\_CombinationRank}(a_1a_2\dots a_m) = \sum_{i=1}^{m} \sum_{x=a_{i-1}+1}^{a_1-1} C_{n-x}^{m-i}.$$

Способ 2: Сочетание из n элементов по m представлено в виде слова $a_1a_2\dots a_n$, где a_i указывает выбран ли этот элемент (значение 1) или нет (значение 0). Тогда

$$P(a_i|a_1\dots a_{i-1}) = \frac{a_i(m-r_i) + (1-a_i)(n-i+1-m+r_i)}{n-i+1},$$

$$P(a_1) = \frac{n-m-na_1+2ma_1}{n},$$

$$q(a_i|a_1\dots a_{i-1}) = \frac{a_i(n-i+1-m+r_i)}{n-i+1}, \quad q(a_1) = \frac{a_1(n-m)}{n},$$

где $r_i = \sum_{k=1}^{i-1} a_k$ — количество единиц в префиксе $a_1\dots a_{i-1}$.

При этом упрощенная запись формулы вычисления ранга имеет вид

$$\texttt{Ryabko\_CombinationRank}(a_1a_2\dots a_n) = \sum_{i=1}^{n} a_i C_{n-i}^{m-r}.$$

Известна оценка вычислительной сложности получаемых с помощью данного метода алгоритмов ранжирования и генерации по рангу комбинаторных объектов, которая равна $O(n\log^c n)$ и которая также соответствует оценке требуемого объема памяти [21]. Данная оценка не учитывает временные затраты на реализацию вычислений значения функции мощности комбинаторного множества, а также на реализацию вычислений в соответствии с правилами биективного отображения между комбинаторным множеством и множеством слов, кодирующих объекты данного множества.

1.2.5 Метод В.В. Кручинина

Функция мощности множества всех перестановок n элементов определяется следующим выражением, принадлежащим алгебре $\{\mathbb{N}, +, \times, R\}$:

$$f(n) = P_n = nP_{n-1},$$

где $P_n = n!$ и $P_1 = 1$.

На рисунке 1.4 представлено дерево И/ИЛИ для множества перестановок n элементов.

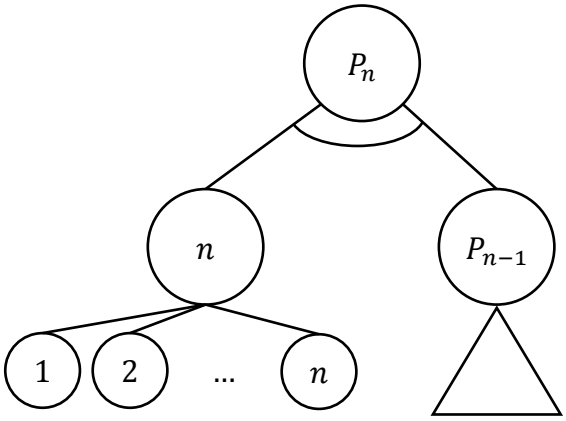


Рисунок 1.4 — Дерево И/ИЛИ для множества перестановок n элементов

Для определения биекции рассмотрим какие изменения происходят в структуре сочетаний при переходе от одного узла дерева И/ИЛИ к другому:

Выбор одного из n листов-сыновей узла по левой ветви дерева показывает набор возможных вариантов позиции размещения элемента под номером n в перестановке n элементов. Соответственно, номер листа, выбранного в варианте дерева, указывает позицию элемента под номером n в перестановке n элементов. Далее данный элемент под номером n не учитывается, и аналогичные рассуждения применяются к перестановке $n-1$ элементов (правая ветвь дерева).

В результате с помощью общих алгоритмов ранжирования и генерации по рангу вариантов дерева И/ИЛИ, а также правил определения биекции между множеством всех вариантов дерева И/ИЛИ и множеством перестановок были получены следующие алгоритмы ранжирования и генерации по рангу перестановок n элементов (оценка вычислительной сложности Алгоритма 7 равна $O(n^2)$, для Алгоритма 8 равна $O(n)$):

Алгоритм 7: Алгоритм ранжирования множества перестановок, полученный с помощью метода В.В. Кручинина

```
1 Permutation_Rank ((a_1, a_2, ..., a_n), n)
2 begin
3     if n = 1 then return 0
4     l_or := Найти k, для которого a_{k+1} = n
5     l_and := l_or + n · Permutation_Rank ((a_1, ..., a_{l_or}, a_{l_or+2}, ..., a_n), n − 1)
6     return l_and
7 end
```

Алгоритм 8: Алгоритм генерации по рангу множества перестановок, полученный с помощью метода В.В. Кручинина

```
1 Permutation_Unrank (r, n)
2 begin
3     if n = 1 then return 1
4     l_or := r  mod n
5     (a_1, a_2, ..., a_{n−1}) := Permutation_Unrank (⌊r/n⌋, n − 1)
6     return (a_1, ..., a_{l_or}, n, a_{l_or+1}, ..., a_{n−1})
7 end
```

Функция мощности множества всех сочетаний из n элементов по m определяется следующим выражением, принадлежащим алгебре $\{\mathbb{N}, +, \times, R\}$:

$$f(n,m) = C_n^m = C_{n-1}^m + C_{n-1}^{m-1},$$

где $C_n^m = \frac{n!}{m!(n-m)!}$ и $C_n^n = C_n^0 = 1$.

На рисунке 1.5 представлено дерево И/ИЛИ для множества сочетаний из n элементов по m.

Для определения биекции рассмотрим какие изменения происходят в структуре сочетаний при переходе от одного узла дерева И/ИЛИ к другому:

– При переходе по левой ветви дерева наблюдается уменьшение на единицу параметра n, при этом параметр m не изменяется. Данное изменение говорит о том, что элемент под номером n не был выбран для набора из m элементов и среди оставшихся нерассмотренных $n-1$ элементов множества необходимо все также отобрать m элементов.

– При переходе по правой ветви дерева наблюдается уменьшение на единицу параметров n и m. Данное изменение говорит о том, что элемент под номером n был выбран для набора из m элементов и среди оставшихся нерассмотренных $n-1$ элементов множества осталось отобрать $m-1$ элементов.

– Отдельно рассмотрим случаи C_n^n и C_n^0. Если на текущий момент имеется n элементов и среди них необходимо выбрать n элементов, тогда следует выбрать все имеющиеся элементы. Если же среди имеющихся n элементов необходимо выбрать 0 элементов, тогда следует не выбрать все имеющиеся элементы.

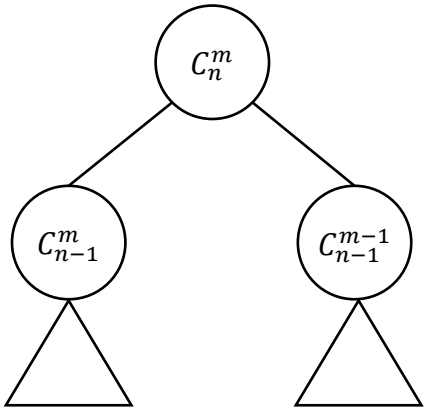


Рисунок 1.5 — Дерево И/ИЛИ для множества сочетаний из n элементов по m

В результате были получены следующие алгоритмы ранжирования и генерации по рангу сочетаний из n элементов по m (оценка вычислительной сложности Алгоритма 9 равна $O(n^2)$, для Алгоритма 10 равна $O(n^2)$):

Алгоритм 9: Алгоритм ранжирования множества сочетаний, полученный с помощью метода В.В. Кручинина

```
1 Combination_Rank ((a_1, a_2, ..., a_n), n, m)
2 begin
3 |   if m = 0 then return 0
4 |   if m = n then return 0
5 |   if a_n = 0 then l_or := Combination_Rank ((a_1, a_2, ..., a_{n-1}), n − 1, m)
6 |   else l_or := Combination_Rank ((a_1, a_2, ..., a_{n-1}), n − 1, m − 1) + C(n−1, k)
7 |   return l_or
8 end
```

Алгоритм 10: Алгоритм генерации по рангу множества сочетаний, полученный с помощью метода В.В. Кручинина

```
1  Combination_Unrank (r, n, m)
2  begin
3      if m = 0 then
4          (a_1, a_2, ..., a_n) := (0, 0, ..., 0)
5          return (a_1, a_2, ..., a_n)
6      end
7      if m = n then
8          (a_1, a_2, ..., a_n) := (1, 1, ..., 1)
9          return (a_1, a_2, ..., a_n)
10     end
11     if r < C(n-1, k) then
12         (a_1, a_2, ..., a_n) := Combination_Unrank (r, n - 1, m)
13         return (a_1, a_2, ..., a_n, 0)
14     end
15     else
16         (a_1, a_2, ..., a_n) := Combination_Unrank (r - C(n-1, k), n - 1, m - 1)
17         return (a_1, a_2, ..., a_n, 1)
18     end
19 end
```

При необходимости разработки алгоритма последовательной генерации комбинаторных объектов, во-первых, можно его реализовать путем вызова в цикле алгоритма генерации по рангу, который последовательно обходит все возможные значения рангов комбинаторных объектов. Во-вторых, можно воспользоваться общим алгоритмом последовательной генерации комбинаторных объектов, в основе которого лежит идея перехода от одного варианта дерева И/ИЛИ к другому с помощью функции Next [26].

При необходимости разработки алгоритма генерации комбинаторных объектов в случайном порядке можно его реализовать путем вызова алгоритма генерации по рангу для значений рангов, полученных с помощью генератора случайных чисел.

1.3 Выводы по главе

В настоящее время существует несколько универсальных методов построения алгоритмов комбинаторной генерации. Проведенный анализ таких методов показал, что:

– часть методов (метод поиска с возвратом и ECO-метод) направлены только на разработку алгоритмов последовательной генерации комбинаторных объектов;

– существуют ограничения на возможность применения части методов (ECO-метод и метод Ф. Флажоле) для комбинаторных множеств, описываемых более чем одним параметром;

– большинство методов требуют представления комбинаторного объекта в специальном виде (слово, последовательность, спецификация, дерево И/ИЛИ), что не всегда является тривиальной задачей и требует дополнительного исследования;

– существуют требования к наличию дополнительной информации, описывающей комбинаторное множество.

В таблице 1.2 представлены результаты сравнения выявленных основных характеристик методов построения алгоритмов комбинаторной генерации:

– характеристика «Listing»: имеется возможность построения алгоритмов последовательной генерации комбинаторных объектов;

– характеристика «Ranking/Unranking»: имеется возможность построения алгоритмов ранжирования и генерации комбинаторных объектов в соответствии с заданным рангом;

– характеристика «Более одного параметра»: имеется возможность применения метода для комбинаторных множеств, описываемых более чем одним параметром;

– характеристика «Биекция»: имеется требование представления комбинаторного объекта в специальном виде;

– характеристика «Дополнительные требования»: имеются требования к наличию дополнительной информации, описывающей комбинаторное множество.

Таблица 1.2 — Сравнение характеристик общих методов построения алгоритмов комбинаторной генерации

Метод	Характеристика				
	Listing	Ranking/ Unranking	Более одного параметра	Биекция	Дополнительные требования
Метод поиска с возвратом	+	−	+	−	
ECO-метод	+	−	−	+	ECO-правило
Метод Ф. Флажоле	+	+	−	+	Функция мощности, производящая функция, спецификация
Метод Б.Я. Рябко	+	+	+	+	Функция мощности, вспомогательные функции
Метод В.В. Кручинина	+	+	+	+	Функция мощности из алгебры $\{\mathbb{N}, +, \times, R\}$

Для дальнейшего исследования выбран метод комбинаторной генерации В.В. Кручинина, основанный на применении деревьев И/ИЛИ, так как данный метод:

– позволяет разрабатывать все типы алгоритмов комбинаторной генерации (listing, ranking и unranking);

– не имеет ограничений на количество параметров, которыми описываются комбинаторные множества, что позволяет рассматривать более сложные дискретные структуры;

– требует в качестве дополнительной информации, описывающей комбинаторное множество, только выражение функции мощности, на основе которой строится структура дерева И/ИЛИ.

Однако при этом необходимо знать выражение функции мощности комбинаторного множества, принадлежащее алгебре $\{\mathbb{N}, +, \times, R\}$. Это является одним из недостатков применения данного метода для разработки алгоритмов комбинаторной генерации, который в рамках диссертационного исследования предлагается устранить за счет применения математического аппарата производящих функций.

Глава 2. Математический инструментарий для построения алгоритмов комбинаторной генерации на основе применения теории производящих функций

В данной главе представлены полученные результаты разработки модифицированного метода построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ за счет применения теории производящих функций.

2.1 Метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ

На основе полученных результатов проведенного анализа современного состояния исследований в области разработки алгоритмов комбинаторной генерации для дальнейшего исследования выбран метод комбинаторной генерации В.В. Кручинина [25; 26], основанный на применении деревьев И/ИЛИ. Метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ базируется на представлении комбинаторных множеств в виде структуры дерева И/ИЛИ, число вариантов которого должно совпадать со значением функции мощности комбинаторного множества [44]. С помощью такого дерева И/ИЛИ можно строить алгоритмы последовательной генерации комбинаторных объектов, их ранжирования и генерации в соответствии с их рангами.

Деревом И/ИЛИ называется дерево, которое содержит узлы двух типов: И-узел и ИЛИ-узел. На рисунке 2.1 представлено схематическое изображение узлов дерева И/ИЛИ.

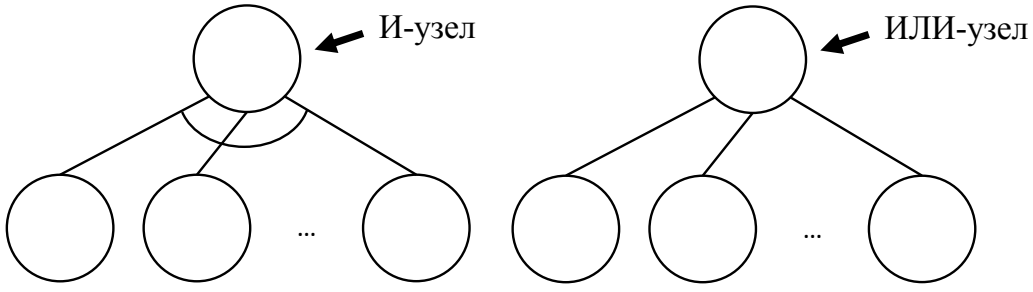


Рисунок 2.1 — Схематическое изображение узлов дерева И/ИЛИ

Вариантом дерева И/ИЛИ называется дерево, которое получается из данного путем отсечения у всех ИЛИ-узлов всех дуг кроме одной. На рисунке 2.2 показан пример структуры дерева И/ИЛИ и всех его вариантов.

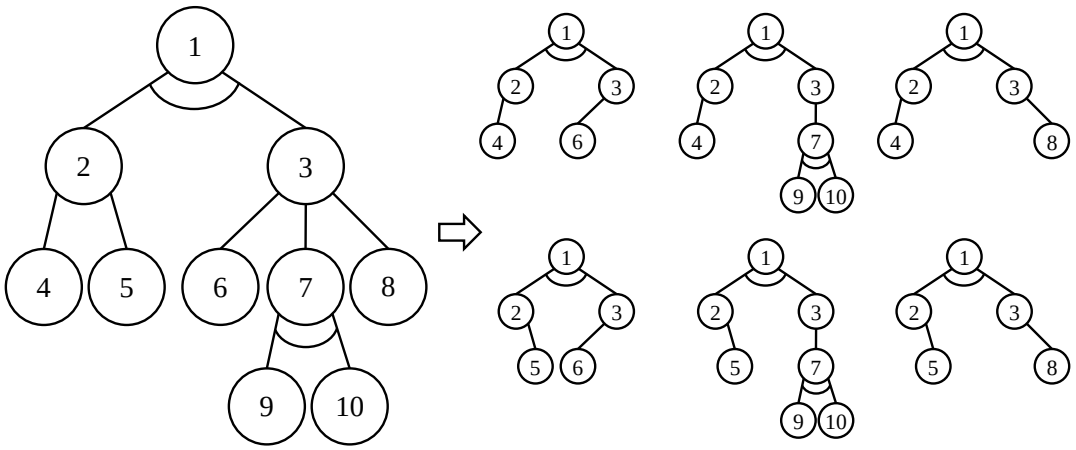


Рисунок 2.2 — Дерево И/ИЛИ и все его варианты

Если для комбинаторного множества A известно выражение функции мощности $f = |A|$, которое принадлежит алгебре $\{\mathbb{N}, +, \times, R\}$, где R — оператор примитивной рекурсии, тогда существует возможность построения дерева И/ИЛИ, число вариантов которого будет совпадать со значением функции мощности комбинаторного множества [58]. Для этого необходимо для заданной функции мощности f комбинаторного множества A:

– все операции сложения $+$ из f представить в виде ИЛИ-узла дерева И/ИЛИ, где все слагаемые — это сыновья данного ИЛИ-узла;

– все операции произведения $\times$ из f представить в виде И-узла дерева И/ИЛИ, где все множители — это сыновья данного И-узла;

– все коэффициенты $k \in \mathbb{N}$ из f представить в виде ИЛИ-узла дерева И/ИЛИ, у которого все сыновья являются листами и их количество равно k;

– все рекурсивные операции из f представить в виде схемы рекурсивной композиции дерева И/ИЛИ (обозначение: узел с треугольником).

Таким образом, метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ можно записать в следующем виде:

Вход: Функция мощности комбинаторного множества $f \in \{\mathbb{N}, +, \times, R\}$.

Выход: Алгоритмы комбинаторной генерации

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

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

при этом каждый вариант v дерева И/ИЛИ D, построенного для некоторого комбинаторного множества A, должен однозначно соответствовать конкретному комбинаторному объекту $a \in A$, то есть должна быть определена биекция $A \leftrightarrow W(D)$, где $W(D)$ — множество всех вариантов v дерева И/ИЛИ D. Совокупность данной биекции с алгоритмами `RankVariant` и `UnrankVariant` представляют собой искомые алгоритмы комбинаторной генерации $\texttt{Rank}(a) : A \to \mathbb{N}$ и $\texttt{Unrank}(r) : \mathbb{N} \to A$.

Алгоритм `RankVariant` определяет соответствие между элементами множества $W(D)$ всех вариантов дерева И/ИЛИ D и элементами конечного множества натуральных чисел $\mathbb{N}_{|W(D)|} = \{0, 1, \ldots, |W(D)| - 1\}$. Данный алгоритм ранжирования позволяет для каждого варианта $v \in W(D)$ дерева И/ИЛИ D поставить в соответствие некоторый уникальный порядковый номер $r \in \mathbb{N}_{|W(D)|}$, называемый рангом. В основе работы алгоритма `RankVariant` заложены правила вычисления значения $l(z)$, приведенные в подразделе 1.1.5 настоящей диссертационной работы. Для общего случая данный алгоритм ранжирования вариантов дерева И/ИЛИ был формализован и представлен в виде рекурсивного Алгоритма 11 [41]. Для запуска выполнения Алгоритма 11 необходимо вызвать команду $\texttt{RankVariant}(root, v, D)$, где $root$ — корень дерева И/ИЛИ D.

Алгоритм `UnrankVariant` выполняет обратное действие к алгоритму `RankVariant`. Данный алгоритм генерации по рангу позволяет для каждого ранга $r \in \mathbb{N}_{|W(D)|}$ поставить в соответствие некоторый вариант $v \in W(D)$ дерева И/ИЛИ D. В основе работы алгоритма `UnrankVariant` заложены обратные действия к вычислениям, использующимся в алгоритме `RankVariant`. Для общего случая данный алгоритм генерации вариантов дерева И/ИЛИ по рангу был формализован и представлен в виде Алгоритма 12. Для запуска выполнения Алгоритма 12 необходимо вызвать команду $\texttt{UnrankVariant}(r, D)$.

Если для рассматриваемого комбинаторного множества A построена структура дерева И/ИЛИ D, то возникает проблема определения биекции между элементами комбинаторного множества A и множества вариантов дерева И/ИЛИ D (то есть каждому варианту v дерева И/ИЛИ D должен однозначно соответствовать только один конкретный комбинаторный объект $a \in A$ и наоборот). Формализованного подхода к решению данной проблемы не существует, так как каждое комбинаторное множество обладает своими и, зачастую, совершенно уникальными характеристиками, но при этом можно использовать

Алгоритм 11: Алгоритм ранжирования вариантов дерева И/ИЛИ

```
1  RankVariant (z, v, D)
2  begin
3      if z = лист дерева D then return 0
4      if z = И-узел дерева D then
5          n := Количество сыновей узла z
6          l := RankVariant (s_n^(z), v, D)
7          for i := n − 1 to 1 do l := RankVariant (s_i^(z), v, D) + w(s_i^(z)) · l
8          return l
9      end
10     if z = ИЛИ-узел дерева D then
11         k := Порядковый номер выбранного в варианте v сына узла z
               среди всех его сыновей
12         l := RankVariant (s_k^(z), v, D)
13         for i := 1 to k − 1 do l := l + w(s_i^(z))
14         return l
15     end
16 end
```

следующие рекомендации: нужно рассмотреть какие изменения происходят в структуре комбинаторных объектов при переходе от одного узла дерева И/ИЛИ к другому и отразить данные изменения в биекции. Например, это могут быть изменения параметров комбинаторного объекта при переходе с одного уровня дерева на другой или при переходе по разным ветвям дерева.

Если же для рассматриваемого комбинаторного множества A не известно выражение функции мощности $f = |A|$, принадлежащее алгебре $\{\mathbb{N}, +, \times, R\}$, тогда структура дерева И/ИЛИ для такого комбинаторного множества не будет построена и, соответственно, применение метода построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ становится невозможным. Предлагается решить эту проблему за счет применения математического аппарата производящих функций, так как он является основополагающим в современной комбинаторике и для многих комбинаторных множеств уже известны выражения производящих функций или существует возможность получения таких выражений.

Алгоритм 12: Алгоритм генерации вариантов дерева И/ИЛИ по рангу

```
1  UnrankVariant (r, D)
2  begin
3      Поместить в стек пару (r, root)
4      Записать root в вариант v
5      while Стек ≠ Пустой стек do
6          Вытащить из стека пару (l, z)
7          if z = И-узел дерева D then
8              n := Количество сыновей узла z
9              for i := 1 to n do
10                 l_i := l  mod w(s_i^(z))
11                 Поместить в стек пару (l_i, s_i^(z))
12                 Записать s_i^(z) в вариант v
13                 l := ⌈ l / w(s_i^(z)) ⌉
14             end
15         end
16         if z = ИЛИ-узел дерева D then
17             n := Количество сыновей узла z
18             sum := 0
19             for i := 1 to n do
20                 if sum + w(s_i^(z)) > l then
21                     l := l − sum
22                     Поместить в стек пару (l, s_i^(z))
23                     Записать s_i^(z) в вариант v
24                     Прервать работу цикла for
25                 end
26                 sum := sum + w(s_i^(z))
27             end
28         end
29     end
30     return v
31 end
```

2.2 Метод получения явных выражений коэффициентов производящих функций

Для решения проблемы необходимости получения выражений функций мощности комбинаторных множеств предлагается использовать теорию производящих функций, а именно метод получения явных выражений коэффициентов производящих функций.

Производящие функции [27–29] являются мощным инструментом решения задач из самых разных областей математических наук, таких как комбинаторика, теория чисел, теория вероятностей и др. Основное преимущество использования производящих функций заключается в том, что они позволяют представить бесконечные числовые последовательности в компактной форме и при этом имеется соответствующий математический аппарат для исследования таких числовых последовательностей через их производящие функции.

Согласно Р.П. Стэнли [59], обыкновенной производящей функцией (или производящей функцией) произвольной числовой последовательности $(a_n)_{n\geqslant 0}$ называется формальный степенной ряд вида

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

В цикле работ [60–63] для коэффициентов степеней производящих функций было введено понятие композиты производящей функции, которое легло в основу математического аппарата степеней производящих функций. Композитой обыкновенной производящей функции $G(t) = \sum\limits_{n>0} g_n t^n$ называется функция $G^{\Delta}(n,k)$, которая является функцией коэффициентов k-й степени производящей функции $G(t)$

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

Математический аппарат степеней производящих функций обеспечивает такие операции над композитами, как сдвиг, сложение, умножение, композиция, а также определение взаимных и обратных композит. Такой набор операций над композитами позволяет получать явные выражения для коэффициентов производящих функций.

Например, если для заданной производящей функции $G(t) = \sum\limits_{n>0} g_n t^n$ известна ее композита $G^{\Delta}(n,k)$, тогда значения коэффициентов g_n могут быть вычислены как

$$g_n = G^{\Delta}(n,1).$$

Также если существует возможность представления заданной производящей функции $G(t)$ в виде композиции двух производящих функций

$$G(t) = R(F(t)) = \sum_{n\geqslant 0} g_n t^n,$$

где $R(t) = \sum\limits_{n\geqslant 0} r_n t^n$ и $F(t) = \sum\limits_{n>0} f_n t^n$, тогда значения коэффициентов g_n могут быть вычислены как

$$g_n = \begin{cases} r_0, & n = 0; \\ \sum\limits_{k=1}^{n} F^{\Delta}(n,k) r_k, & n > 0. \end{cases} \tag{2.1}$$

Таким образом, для получения явных выражений коэффициентов производящих функций с использованием математического аппарата степеней производящих функций необходимо декомпозировать заданную производящую функцию на функции, для которых известны значения композит, и затем применить к ним соответствующие операции. В работе [64] представлено алгоритмическое и программное обеспечение для данного метода получения явных выражений коэффициентов производящих функций.

Если для некоторого комбинаторного множества A рассмотреть его подмножество $A_n \subset A$, которое содержит только комбинаторные объекты размерности n, тогда функция мощности $f(n) = |A_n|$ такого комбинаторного множества A_n может быть описана некоторой производящей функцией

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

Следовательно, для получения выражения функции мощности $f(n)$ комбинаторного множества, для которого известна производящая функция, можно использовать описанный выше метод получения явных выражений коэффициентов производящих функций [48]. Однако, если рассматриваемый комбинаторный объект описывается более чем одним параметром (например, сочетание

из n элементов по m описывается размерностью множества различных элементов — параметр n, а также количеством выбранных элементов из этого множества — параметр m), тогда применение данного метода получения явных выражений коэффициентов производящих функций становится сложной задачей, так как соответствующий ему математический аппарат степеней производящих функций определен только для одномерных производящих функций.

По результатам проведенных исследований композиций производящих функций, в которых внешняя функция описывается двумя формальными переменными, данный метод был дополнен для следующих случаев:

1. Рассмотрим композицию двух производящих функций

$$G(x,y) = R(F(x),y) = \sum_{n\geqslant 0}\sum_{m\geqslant 0} g_{n,m}x^n y^m,$$

где $R(x,y) = \sum\limits_{n\geqslant 0}\sum\limits_{m\geqslant 0} r_{n,m}x^n y^m$ и $F(t) = \sum\limits_{n>0} f_n t^n$. Если зафиксировать переменную y как константу, то $R(x,y) = R_y(x) = \sum\limits_{n\geqslant 0} r_n x^n$, где $r_n = \sum\limits_{m\geqslant 0} r_{n,m}y^m$.

Тогда, используя (2.1), значения коэффициентов $g_{n,m}$ могут быть вычислены как

$$g_{n,m} = \begin{cases} r_{0,m}, & n = 0; \\ \sum\limits_{k=1}^{n} F^{\Delta}(n,k)r_{k,m}, & n > 0. \end{cases} \tag{2.2}$$

2. Рассмотрим композицию двух производящих функций

$$G(x,y) = R(F_1(x),F_2(y)) = \sum_{n\geqslant 0}\sum_{m\geqslant 0} g_{n,m}x^n y^m,$$

где $R(x,y) = \sum\limits_{n\geqslant 0}\sum\limits_{m\geqslant 0} r_{n,m}x^n y^m$, $F_1(t) = \sum\limits_{n>0} a_n t^n$ и $F_2(t) = \sum\limits_{n>0} b_n t^n$.

Тогда значения коэффициентов $g_{n,m}$ могут быть вычислены как

$$g_{n,m} = \begin{cases} r_{0,0}, & m = 0, n = 0; \\ \sum\limits_{k=1}^{n} F_1^{\Delta}(n,k)r_{k,0}, & m = 0, n > 0; \\ \sum\limits_{l=1}^{m} F_2^{\Delta}(m,l)r_{0,l}, & m > 0, n = 0; \\ \sum\limits_{l=1}^{m} F_2^{\Delta}(m,l)\sum\limits_{k=1}^{n} F_1^{\Delta}(n,k)r_{k,l}, & m > 0, n > 0. \end{cases}$$

Аналогичные соотношения могут быть получены и для более сложных вариантов композиций производящих функций. При этом основным ограничением является требование того, что декомпозирование заданной производящей

функции должно осуществляться в рамках одной формальной переменной, а остальные переменные в этот момент фиксируются как константы. То есть в качестве внутренних производящих функций композиции должны быть одномерные производящие функции.

В качестве примера рассмотрим процесс получения явного выражения коэффициентов производящей функции транспонированного числового треугольника Каталана (числовая последовательность A033184 в OEIS [65]). Известно следующее выражение для производящей функции транспонированного числового треугольника Каталана [66]:

$$CT(x,y) = \frac{y \cdot C(x)}{1 - y \cdot C(x)} = \sum_{n>0} \sum_{m>0} CT_n^m x^n y^m. \tag{2.3}$$

Данную производящую функцию можно представить в виде композиции $C(x,y) = R(C(x),y)$ производящих функций

$$R(x,y) = \frac{yx}{1-yx} = \sum_{n>0} \sum_{m>0} \delta_{n,m} x^n y^m,$$

$$C(x) = \frac{1-\sqrt{1-4x}}{2} = \sum_{n>0} C_{n-1} x^n, \tag{2.4}$$

где $\delta_{n,m}$ — символ Кронекера; C_n — число Каталана [67]; $C(x)$ — производящая функция чисел Каталана.

Для производящей функции $C(x)$ известно выражение ее композиты [60]

$$C^{\Delta}(n,k) = \frac{k}{2n-k} \binom{2n-k}{n}. \tag{2.5}$$

Используя (2.2), получим известное явное выражение коэффициентов производящей функции транспонированного числового треугольника Каталана [66]

$$CT_n^m = \begin{cases} 0, & n = 0; \\ \sum\limits_{k=1}^{n} C^{\Delta}(n,k)\delta_{k,m}, & n > 0. \end{cases} = \begin{cases} 0, & n = 0; \\ \frac{m}{2n-m}\binom{2n-m}{n}, & n > 0. \end{cases} \tag{2.6}$$

Полученное выражение (2.6) представляет собой выражение функции мощности для комбинаторных множеств, число элементов которых описывается элементами транспонированного числового треугольника Каталана. Однако применение данного выражения для построения алгоритмов комбинаторной

генерации для таких множеств с помощью метода, приведенного в разделе 2.1, является невозможным, так как оно не принадлежит требуемой алгебре $\{\mathbb{N}, +, \times, R\}$ (присутствует операция деления).

В рамках дополнительных исследований, направленных на изучение и апробацию математического аппарата степеней производящих функций, были получены следующие результаты:

При изучении целочисленных свойств композиции обыкновенных и экспоненциальных производящих функций [36;37] были доказаны теоремы, применение которых позволяет получать новые критерии простоты числа. Например, для композиции производящих функций $G(x) = B(E(x)) = \sum\limits_{n\geqslant 0} \frac{g_n}{n!}x^n$, где $B(x) = \sum\limits_{n\geqslant 0} b_n x^n$ $(b_n \in \mathbb{Z})$ и $E(x) = \sum\limits_{n>0} \frac{e_n}{n!}x^n$ $(e_n \in \mathbb{Z})$, если n является простым числом, тогда

$$g_n - e_n b_1 \equiv 0 \mod n.$$

На основе доказанных теорем и их следствий было разработано программное обеспечение «Primality Criterion Generator» [31; 49], которое можно использовать для генерации новых критериев простоты числа для поиска простых чисел и последующего их применения в криптографии с открытым ключом [68; 69]. Также было разработано программное обеспечение «Primality Test Analyser» [30;50], которое предназначено для анализа тестов простоты числа, построенных на основе новых критериев простоты числа, и их сравнения с другими новыми или известными тестами простоты числа.

Применение метода получения явных выражений коэффициентов производящих функций было апробировано при решении задачи получения явных выражений полиномов на основе известных для них выражений производящих функций [34; 70]. В качестве результата были получены явные выражения для большого перечня полиномов, что подтвердило эффективность применения метода получения явных выражений коэффициентов производящих функций. Также были проведены работы по автоматизации данного процесса с помощью системы компьютерной алгебры «Mathematica» [32; 38; 71].

Исследование задачи получения выражений коэффициентов обратных производящих функций [35] позволило расширить возможности применения метода получения явных выражений полиномов, что привело к получению новых тождеств для таких полиномов, производящие функции которых имеют вид $F(t)^x G(t)^\alpha$ [72].

2.3 Модифицированный метод построения алгоритмов комбинаторной генерации

Применение метода получения явных выражений коэффициентов производящих функций для нахождения выражения функции мощности заданного комбинаторного множества позволяет воспользоваться методом построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ для таких комбинаторных множеств, для которых не известно выражение функции мощности, принадлежащее алгебре $\{\mathbb{N}, +, \times, R\}$, но известно выражение производящей функции для последовательности значений функции мощности. Также расширение данного метода получения явных выражений коэффициентов производящих функций для случая композиции производящих функций, в которой внешняя функция является функцией нескольких переменных, делает возможным нахождение выражений функций мощности для комбинаторных множеств, которые описываются более чем одним параметром.

Однако существуют ограничения на возможность применения данного метода, так как выражение функции мощности заданного комбинаторного множества, получаемое в результате применения метода получения явных выражений коэффициентов производящих функций, не всегда соответствует требуемой алгебре $\{\mathbb{N}, +, \times, R\}$. Например, такая ситуация может возникнуть когда композита производящей функции не принадлежит алгебре $\{\mathbb{N}, +, \times, R\}$.

Достоинством применения данного метода является то, что по виду структуры композиции производящих функций, полученной для производящей функции последовательности значений функции мощности заданного комбинаторного множества, зачастую, можно получить некоторое представление о свойствах элементов данного комбинаторного множества. Например, так как представленная в виде композиции производящая функция (2.3) содержит в качестве внутренней функции по формальной переменной x производящую функцию чисел Каталана $C(x)$, то элементы комбинаторных множеств, для которых выражение функции мощности соответствует формуле (2.6), по параметру n будут иметь схожие свойства с числами Каталана.

Запишем модифицированный метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ в виде последовательности шагов [33]:

Шаг 1. Если известно выражение функции мощности f комбинаторного множества A, принадлежащее алгебре $\{\mathbb{N}, +, \times, R\}$, то переход на шаг 4.

Шаг 2. Если известно выражение производящей функции F для последовательности значений функции мощности f комбинаторного множества A, то применить метод получения явных выражений коэффициентов производящих функций, иначе дальнейшее применение метода невозможно.

Шаг 3. Если получено выражение функции мощности f комбинаторного множества A, принадлежащее алгебре $\{\mathbb{N}, +, \times, R\}$, то переход на шаг 4, иначе дальнейшее применение метода невозможно.

Шаг 4. На основе выражения функции мощности f комбинаторного множества A построить структуру дерева И/ИЛИ D.

Шаг 5. Определить биекцию $A \leftrightarrow W(D)$ между элементами комбинаторного множества A и множества всех вариантов v дерева И/ИЛИ D в виде алгоритмов $\texttt{ObjectToVariant}(a, D) : A \to W(D)$, где $a \in A$, и $\texttt{VariantToObject}(v, D) : W(D) \to A$, где $v \in W(D)$.

Шаг 6. Определить биекцию $W(D) \leftrightarrow \mathbb{N}_{|W(D)|}$ между элементами множества всех вариантов v дерева И/ИЛИ D и конечного множества натуральных чисел $\mathbb{N}_{|W(D)|} = \{0, 1, \ldots, |W(D)| - 1\}$ с помощью алгоритмов $\texttt{RankVariant}(root, v, D) : W(D) \to \mathbb{N}_{|W(D)|}$, где $root$ — корень дерева И/ИЛИ D, $v \in W(D)$, и $\texttt{UnrankVariant}(r, D) : \mathbb{N}_{|W(D)|} \to W(D)$, где $r \in \mathbb{N}_{|W(D)|}$.

Совокупность алгоритмов, определенных на последних двух шагах модифицированного метода (Шаг 5 и Шаг 6), формируют биекцию $A \leftrightarrow \mathbb{N}$ и представляют собой алгоритмы комбинаторной генерации $\texttt{Rank}(a) : A \to \mathbb{N}$ и $\texttt{Unrank}(r) : \mathbb{N} \to A$. Разработанный модифицированный метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ отличается применением метода получения явных выражений коэффициентов производящих функций для нахождения выражения функции мощности комбинаторного множества. Изменение оригинального метода заключается в наличии дополнительных шагов (Шаг 2 и Шаг 3), которые в случае успешного их выполнения позволяют воспользоваться методом построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ.

2.4 Выводы по главе

Основным результатом данной главы является разработанный модифицированный метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ. В отличие от оригинальной версии метода, в предлагаемой модификации применяется метод получения явных выражений коэффициентов производящих функций для нахождения выражения функции мощности комбинаторного множества. Данное дополнение позволяет в случае успешного получения выражения функции мощности, принадлежащего алгебре $\{\mathbb{N}, +, \times, R\}$, воспользоваться методом построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ для таких комбинаторных множеств, для которых не известно выражение функции мощности, принадлежащее алгебре $\{\mathbb{N}, +, \times, R\}$, но известно выражение производящей функции для последовательности значений функции мощности.

Также в ходе проведения дополнительных исследований было представлено расширение метода получения явных выражений коэффициентов производящих функций для некоторых случаев композиции производящих функций, в которой внешняя функция является функцией нескольких переменных. Данное дополнение метода делает возможным нахождение выражений функций мощности для комбинаторных множеств, которые описываются более чем одним параметром.

Глава 3. Апробация модифицированного метода построения алгоритмов комбинаторной генерации

В данной главе представлены полученные результаты апробации модифицированного метода построения алгоритмов комбинаторной генерации на примере следующих комбинаторных множеств:

– множество комбинаторных объектов, отражающих выражения обобщенного языка Дика;

– множество комбинаторных объектов, отражающих вторичную структуру РНК;

– множество комбинаторных объектов, определяемых треугольником Эйлера-Каталана.

3.1 Алгоритмы комбинаторной генерации для множества комбинаторных объектов, отражающих выражения обобщенного языка Дика

Язык Дика — это контекстно-свободный язык над конечным алфавитом $\{a, b\}$, порождаемый контекстно-свободной грамматикой $S \to \varepsilon|aSbS$ [73].

Примером множества выражений языка Дика (слова Дика) является множество последовательностей правильно вложенных скобок, исследование которых необходимо при организации работы трансляторов языков программирования высокого уровня [4]. В данном случае символ a представляет собой символ открывающейся скобки '(', а символ b — символ закрывающейся скобки ')'. В качестве примера рассмотрим все возможные последовательности правильно вложенных скобок, которые могут быть получены из трех пар открывающейся и закрывающейся скобок:

$$()()(), ()(()), (())(), (()()), ((())).$$

Комбинаторное множество последовательностей правильно вложенных скобок определяется параметром n — количество пар открывающейся и закрывающейся скобок. При этом длина выражения языка Дика будет составлять $2n$.

Функция мощности такого комбинаторного множества A_n определяется значением n-го числа Каталана:

$$f(n) = |A_n| = C_n,$$

где $C_n = \frac{1}{n+1}\binom{2n}{n}$ — это n-е число Каталана [67].

Числа Каталана образуют следующую целочисленную последовательность (A000108 в OEIS [65]):

$$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, \ldots$$

Числа Каталана являются одним из важнейших объектов комбинаторики, так как их значения описывают огромный набор разнообразных комбинаторных множеств. Перечень из более чем 200 комбинаторных интерпретаций чисел Каталана представлена в работах Р. Стенли [74; 75]. Кроме того, наличие связи между множеством выражений языка Дика и числами Каталана позволяет задать взаимно-однозначное соответствие между выражениями языка Дика и элементами других комбинаторных множеств, мощность которых также определяется значением чисел Каталана.

Также существует обобщение языка Дика. В данном случае рассматривается контекстно-свободный язык над конечным алфавитом $\{a_1, b_1, a_2, b_2, \ldots, a_m, b_m\}$, порождаемый контекстно-свободной грамматикой $S \to \varepsilon | a_1 S b_1 S | a_2 S b_2 S | \ldots | a_m S b_m S$. Если рассматривать в качестве выражений обобщенного языка Дика последовательности правильно вложенных скобок, то это будет соответствовать последовательностям правильно вложенных скобок m типов. Например, имеется ровно 8 возможных последовательностей правильно вложенных скобок двух типов, которые могут быть получены из двух пар открывающейся и закрывающейся скобок:

$$()(), (()), ()[], ([]), [](), [()], [][], [[]].$$

Комбинаторное множество последовательностей правильно вложенных скобок разных типов определяется двумя параметрами: n — количество пар открывающейся и закрывающейся скобок, m — количество типов скобок. Функция мощности такого комбинаторного множества $A_{n,m}$ определяется формулой

$$f(n,m) = |A_{n,m}| = C_n^m = C_n \overline{P}_m^n, \tag{3.1}$$

где $\overline{P}_n^m = n^m$ — это количество m-перестановок n элементов с повторениями.

Здесь для каждой последовательности правильно вложенных n пар скобок, число которых определяется значением n-го числа Каталана (C_n), рассматриваются все возможные варианты использования скобок, число которых определяется количеством перестановок n пар скобок, взятых из множества m пар скобок разных типов с возможностью повторений ($\overline{P}_m^n$).

Исследования, связанные с разработкой алгоритмов ранжирования и генерации по рангу для множества выражений языка Дика и множества выражений обобщенного языка Дика, представлены в работах [23;76;77]. В частности, в диссертационной работе Ю.С. Медведевой [24] представлены эффективные с точки зрения оценки вычислительной сложности алгоритмы ранжирования и генерации по рангу как для множества выражений языка Дика, так и для множества выражений обобщенного языка Дика. Для представленных алгоритмов оценка вычислительной сложности, а также оценка требуемого объема памяти равна $O(\log^3 n \log\log n)$. В данном случае для разработки алгоритмов комбинаторной генерации был применен метод Б.Я. Рябко.

Рассмотрим подробно процесс разработки алгоритмов ранжирования и генерации по рангу для множества выражений обобщенного языка Дика с помощью модифицированного метода построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ. Выражение (3.1) для функции мощности комбинаторного множества выражений обобщенного языка Дика принадлежит требуемой алгебре $\{\mathbb{N}, +, \times, R\}$, поэтому на основе данного выражения можно построить соответствующую комбинаторному множеству структуру дерева И/ИЛИ (рисунок 3.1).

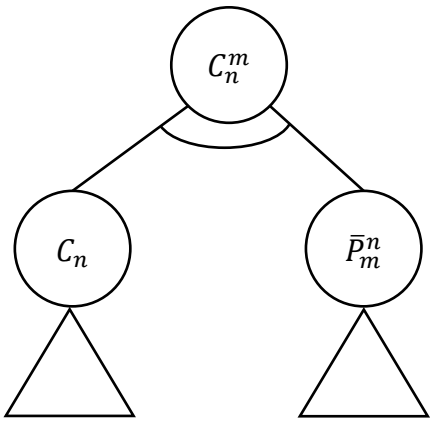


Рисунок 3.1 — Дерево И/ИЛИ для множества выражений обобщенного языка Дика

Таким образом, чтобы получить алгоритмы комбинаторной генерации на основе построенного дерева И/ИЛИ, необходимо разработать алгоритмы комбинаторной генерации для множества последовательностей правильно вложенных n пар скобок, число которых определяется значением C_n, и для множества m-перестановок n элементов с повторениями, число которых определяется значением $\overline{P}_n^m$.

В диссертационной работе [26] представлены алгоритмы комбинаторной генерации для множества выражений языка Дика и множества последовательностей правильно вложенных n пар скобок. Здесь в качестве выражения функции мощности множества выражений языка Дика длины $2n$, принадлежащего алгебре $\{\mathbb{N}, +, \times, R\}$, использовано известное рекуррентное соотношение для чисел Каталана [75]

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

В основе этой формулы лежит правило о том, что любая непустая последовательность правильно вложенных скобок s может быть представлена в форме $(s_1)s_2$, где s_1 и s_2 — последовательности правильно вложенных скобок. Данному выражению соответствует структура дерева И/ИЛИ, представленная на рисунке 3.2.

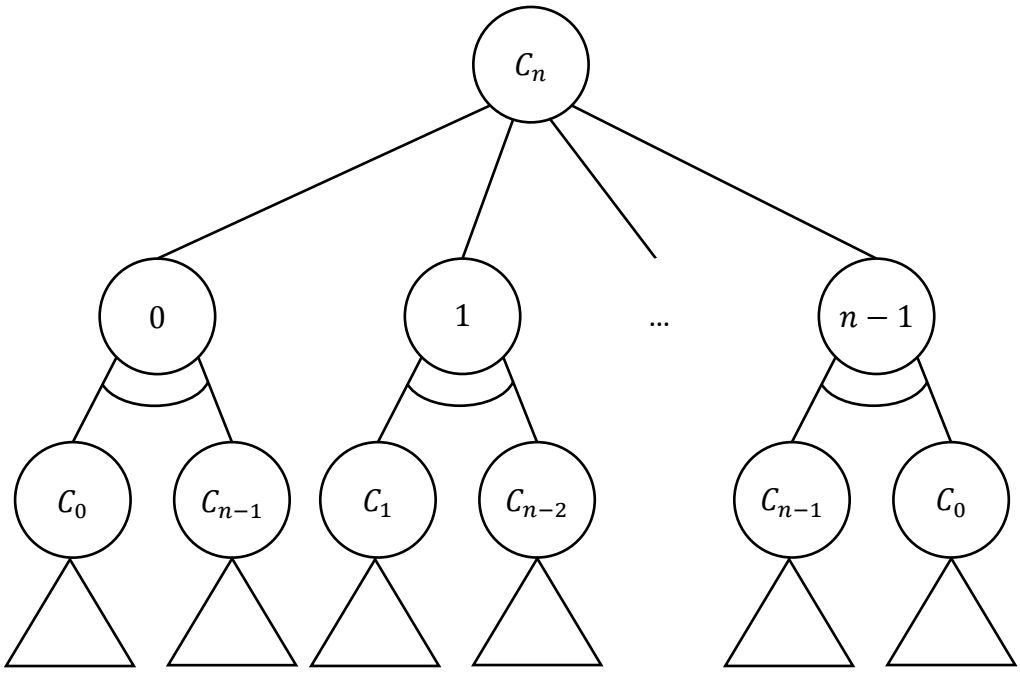


Рисунок 3.2 — Дерево И/ИЛИ для множества выражений языка Дика

Биекция между элементами комбинаторного множества и множества вариантов дерева И/ИЛИ определяется следующим правилом:

На каждом уровне дерева И/ИЛИ в рамках ИЛИ-узла выбирается одна из дуг. Все дуги ИЛИ-узла нумеруются порядковым номером k слева направо, начиная с номера 0. Результатом выбора дуги k получаем И-узел, состоящий из пары ИЛИ-узлов, помеченных как C_k и C_{n-1-k}. При этом значение k показывает какое количество раз в выражении $aSbS$ в рамках вывода из первого нетерминального символа S будет применено правило вывода непустой строки $S \to aSbS$, а значение $n-1-k$ показывает какое количество раз будет применено правило вывода непустой строки $S \to aSbS$ в рамках вывода из второго нетерминального символа S. Например, если рассматривать последовательности правильно вложенных n пар скобок, то значение k покажет количество пар скобок в s_1, а значение $n-1-k$ покажет количество пар скобок в s_2 для последовательности $(s_1)s_2$.

Алгоритм 13: Алгоритм отображения последовательности a правильно вложенных n пар скобок в вариант v дерева И/ИЛИ

```
1  CatalanBracketsToVariant (a, n) := CatalanBracketsToVariant (a, n, 1)
2  CatalanBracketsToVariant (a, n, k)
3  begin
4      v := ()
5      if n = 0 then return ()
6      if a_k = ')' then return (0)
7      else
8          while a_k = '(' do
9              vv := CatalanBracketsToVariant (a, n, k + 1)
10             v := concat ( v, vv )
11             k := k + 2(vv_1 + 1)
12             if k > 2n then break
13         end
14         if k > 2n then return v
15         else return concat ( (|v|), v )
16     end
17 end
```

Алгоритм 14: Алгоритм отображения варианта v дерева И/ИЛИ в последовательность a правильно вложенных n пар скобок

```
1  VariantToCatalanBrackets (v, n)
2  begin
3      (a_1, ..., a_2n) := (0, ..., 0)
4      k := 1
5      for i := 1 to 2n do
6          if a_i = 0 then
7              a_i := '('
8              a_{i+1+2v_k} := ')'
9              k := k + 1
10         end
11     end
12     return (a_1, ..., a_2n)
13 end
```

Для компактного представления будем кодировать варианты дерева И/ИЛИ последовательностью значений номеров выбранных в нем дуг ИЛИ-узлов, полученных при его левостороннем обходе. В Алгоритме 13 и Алгоритме 14 представлена реализация биекции между элементами комбинаторного множества последовательностей правильно вложенных скобок и множества вариантов дерева И/ИЛИ. Алгоритм 13 для последовательности $a = (a_1, \ldots, a_{2n})$ правильно вложенных n пар скобок ($a_i \in \{$ '(', ')' $\}$) определяет соответствующий вариант $v = (v_1, \ldots, v_n)$ дерева И/ИЛИ. Алгоритм 14 выполняет обратное преобразование: для заданного варианта $v = (v_1, \ldots, v_n)$ дерева И/ИЛИ определяется соответствующая последовательность $a = (a_1, \ldots, a_{2n})$ правильно вложенных n пар скобок. Здесь функция `concat` выполняет слияние последовательностей в одну, то есть для двух последовательностей $a = (a_1, \ldots, a_n)$ и $b = (b_1, \ldots, b_m)$ получаем в качестве результата последовательность

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

На основе общих алгоритмов ранжирования (Алгоритм 11) и генерации по рангу (Алгоритм 12) вариантов дерева И/ИЛИ были получены соответствующие алгоритмы (Алгоритм 15 и Алгоритм 16) для дерева И/ИЛИ для множества последовательностей правильно вложенных скобок.

Алгоритм 15: Алгоритм ранжирования варианта v дерева И/ИЛИ, соответствующего последовательности правильно вложенных n пар скобок

```
1  RankVariant_CatalanBrackets (v, n)
2  begin
3      if n = 0 then return 0
4      else
5          k := v_1
6          w := Σ_{i=0}^{k-1} C_i C_{n-i-1}
7          c_1 := RankVariant_CatalanBrackets ((v_2, ..., v_{k+1}), k)
8          c_2 := RankVariant_CatalanBrackets ((v_{k+2}, ..., v_n), n - k - 1)
9          return w + c_1 + C_k c_2
10     end
11 end
```

Алгоритм 16: Алгоритм генерации по рангу варианта v дерева И/ИЛИ, соответствующего последовательности правильно вложенных n пар скобок

```
1  UnrankVariant_CatalanBrackets (r, n)
2  begin
3      if n = 0 then return ()
4      else
5          sum := 0
6          for i := 0 to n - 1 do
7              w := C_i C_{n-i-1}
8              if sum + w > r then
9                  l := r - sum
10                 c_1 := UnrankVariant_CatalanBrackets (l mod C_i, i)
11                 c_2 := UnrankVariant_CatalanBrackets (⌈l / C_i⌉, n - i - 1)
12                 return concat ( (i), c_1, c_2 )
13             end
14         end
15     end
16 end
```

Совокупность Алгоритма 13 и Алгоритма 15 в виде последовательного их применения образуют алгоритм `CatalanBrackets_Rank`(a, n) для ранжирования множества последовательностей правильно вложенных n пар скобок. Совокупность Алгоритма 16 и Алгоритма 14 в виде последовательного их применения образуют алгоритм `CatalanBrackets_Unrank`(r, n) для генерации по рангу множества последовательностей правильно вложенных n пар скобок. В таблице 3.1 представлен пример полученных результатов для случая $n = 4$.

Таблица 3.1 — Пример ранжирования множества последовательностей правильно вложенных n пар скобок при $n = 4$

Последовательность правильно вложенных скобок	Код варианта дерева И/ИЛИ	Ранг
( ) ( ) ( ) ( )	(0,0,0,0)	0
( ) ( ) (( ))	(0,0,1,0)	1
( ) (( )) ( )	(0,1,0,0)	2
( ) (( ) ( ))	(0,2,0,0)	3
( ) ((( )))	(0,2,1,0)	4
(( )) ( ) ( )	(1,0,0,0)	5
(( )) (( ))	(1,0,1,0)	6
(( ) ( )) ( )	(2,0,0,0)	7
((( ))) ( )	(2,1,0,0)	8
(( ) ( ) ( ))	(3,0,0,0)	9
(( ) (( )))	(3,0,1,0)	10
((( )) ( ))	(3,1,0,0)	11
((( ) ( )))	(3,2,0,0)	12
(((( ))))	(3,2,1,0)	13

Для оценки вычислительной сложности разработанных алгоритмов комбинаторной генерации будем использовать асимптотические оценки [78].

Оценка вычислительной сложности Алгоритма 13 равна $O(n^2)$. Данное значение складывается из количества итераций цикла while (максимум n раз) и количества рекурсивных вызовов (n раз).

Оценка вычислительной сложности Алгоритма 14 равна $O(n)$. Данное значение складывается из количества итераций цикла for ($2n$ раз).

Оценка вычислительной сложности Алгоритма 15 равна $O(n^3)$. Данное значение складывается из количества итераций суммирования при расчете значения w (максимум $n-1$ раз), вычислительной сложности $O(n)$ для расчета значения C_n и количества рекурсивных вызовов ($2n$ раз).

Оценка вычислительной сложности Алгоритма 16 равна $O(n^3)$. Данное значение складывается из количества итераций цикла for (n раз), вычислительной сложности $O(n)$ для расчета значения C_n и количества рекурсивных вызовов (n раз).

Таким образом, оценка вычислительной сложности алгоритмов комбинаторной генерации `CatalanBrackets_Rank`(a, n) и `CatalanBrackets_Unrank`(r, n) равна $O(n^3)$. Если предварительно вычислить все значения C_i для $i = 1, \ldots, n$ и хранить их в памяти, тогда оценка вычислительной сложности снижается до $O(n^2)$.

Далее рассмотрим процесс разработки алгоритмов ранжирования и генерации по рангу для множества m-перестановок n элементов с возможностью повторений. В качестве выражения функции мощности множества m-перестановок n элементов с повторениями, принадлежащего алгебре $\{\mathbb{N}, +, \times, R\}$, использована следующая формула:

$$\overline{P}_n^m = n^m = n\overline{P}_n^{m-1}, \quad \overline{P}_n^0 = 1, \quad \overline{P}_0^m = 0.$$

В основе этой формулы лежит правило о том, что каждый из m элементов перестановки может быть выбран n различными способами. Данному выражению соответствует структура дерева И/ИЛИ, представленная на рисунке 3.3.

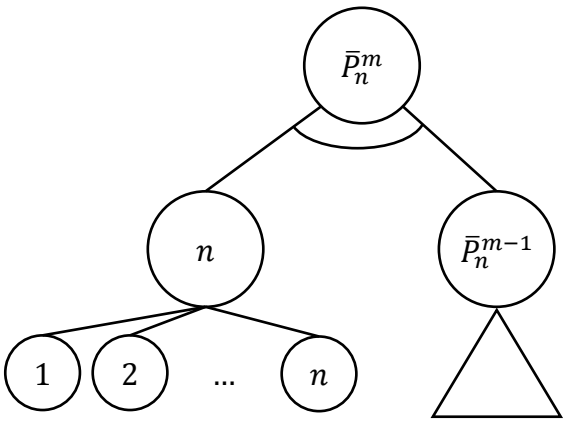


Рисунок 3.3 — Дерево И/ИЛИ для множества m-перестановок n элементов с повторениями

Биекция между элементами комбинаторного множества и множества вариантов дерева И/ИЛИ определяется следующим правилом:

На каждом уровне дерева И/ИЛИ в рамках ИЛИ-узла (левая ветвь дерева И/ИЛИ) выбирается одна из n дуг. Все дуги ИЛИ-узла нумеруются порядковым номером слева направо, начиная с номера 1. Значение выбранной дуги показывает значение элемента перестановки под номером m. Далее в рамках правой ветви дерева И/ИЛИ аналогичные рассуждения применяются к элементу перестановки под номером $m - 1$.

Для компактного представления будем кодировать варианты дерева И/ИЛИ последовательностью значений номеров выбранных в нем дуг ИЛИ-узлов, полученных при его левостороннем обходе. Следовательно, биекция между элементами комбинаторного множества m-перестановок n элементов с повторениями и множества вариантов дерева И/ИЛИ реализуется путем обращения порядка следования элементов. То есть для заданной перестановки m элементов $a = (a_1, \ldots, a_m)$, взятых из множества n элементов с возможностью повторений ($a_i \in \{1, \ldots, n\}$), соответствующий вариант $v = (v_1, \ldots, v_m)$ дерева И/ИЛИ определяется как $v = (v_1, \ldots, v_m) = (a_m, \ldots, a_1)$. Аналогично для заданного варианта $v = (v_1, \ldots, v_m)$ дерева И/ИЛИ определяется соответствующая перестановка $a = (a_1, \ldots, a_m) = (v_m, \ldots, v_1)$.

На основе общих алгоритмов ранжирования (Алгоритм 11) и генерации по рангу (Алгоритм 12) вариантов дерева И/ИЛИ были получены соответствующие алгоритмы (Алгоритм 17 и Алгоритм 18) для дерева И/ИЛИ для множества m-перестановок n элементов с повторениями.

Алгоритм 17: Алгоритм ранжирования варианта v дерева И/ИЛИ, соответствующего m-перестановке n элементов с повторениями

```
1 RankVariant_mPermutationR (v, n, m)
2 begin
3     if m = 0 then return 0
4     if n = 0 then return 0
5     r := v_1 − 1 + n · RankVariant_mPermutationR ((v_2, ..., v_m), n, m − 1)
6     return r
7 end
```

Алгоритм 18: Алгоритм генерации по рангу варианта v дерева И/ИЛИ, соответствующего m-перестановке n элементов с повторениями

```
1 UnrankVariant_mPermutationR (r, n, m)
2 begin
3     if m = 0 then return ()
4     if n = 0 then return ()
5     p := r  mod n
6     r := ⌈r/n⌉
7     v :=concat ( (p + 1), UnrankVariant_mPermutationR (r, n, m − 1) )
8     return v
9 end
```

Разработанные Алгоритм 17 и Алгоритм 18 в совокупности с правилами биекции между элементами комбинаторного множества и множества вариантов дерева И/ИЛИ образуют алгоритм $\texttt{mPermutationR\_Rank}(a, n, m)$ для ранжирования и алгоритм $\texttt{mPermutationR\_Unrank}(r, n, m)$ для генерации по рангу множества m-перестановок n элементов с повторениями. Оценка вычислительной сложности алгоритмов комбинаторной генерации $\texttt{mPermutationR\_Rank}(a, n, m)$ и $\texttt{mPermutationR\_Unrank}(r, n, m)$ равна $O(m)$. Данное значение для обоих алгоритмов складывается из количества рекурсивных вызовов (m раз). В таблице 3.2 представлен пример полученных результатов для случая $n = 4$ и $m = 2$.

Таблица 3.2 — Пример ранжирования множества m-перестановок n элементов с повторениями при $n = 4$ и $m = 2$

Перестановка	Код варианта дерева И/ИЛИ	Ранг	Перестановка	Код варианта дерева И/ИЛИ	Ранг
(1,1)	(1,1)	0	(3,1)	(1,3)	8
(1,2)	(2,1)	1	(3,2)	(2,3)	9
(1,3)	(3,1)	2	(3,3)	(3,3)	10
(1,4)	(4,1)	3	(3,4)	(4,3)	11
(2,1)	(1,2)	4	(4,1)	(1,4)	12
(2,2)	(2,2)	5	(4,2)	(2,4)	13
(2,3)	(3,2)	6	(4,3)	(3,4)	14
(2,4)	(4,2)	7	(4,4)	(4,4)	15

На основе разработанных алгоритмов ранжирования и генерации по рангу вариантов дерева И/ИЛИ, представленного на рисунке 3.2, (Алгоритм 15 и Алгоритм 16) и вариантов дерева И/ИЛИ, представленного на рисунке 3.3, (Алгоритм 17 и Алгоритм 18) можно представить соответствующие алгоритмы для дерева И/ИЛИ, представленного на рисунке 3.1, (Алгоритм 19 и Алгоритм 20). Варианты данного дерева И/ИЛИ будем кодировать парой последовательностей $v = (v_1, v_2)$, первая из которых v_1 соответствует коду поддерева по левой ветви дерева И/ИЛИ, а вторая v_2 — коду поддерева по правой ветви.

Алгоритм 19: Алгоритм ранжирования варианта v дерева И/ИЛИ для множества выражений обобщенного языка Дика

```
1 RankVariant_GeneralizedCatalanBrackets (v, n, m)
2 begin
3 |   l_1 := RankVariant_CatalanBrackets (v_1, n)
4 |   l_2 := RankVariant_mPermutationR (v_2, m, n)
5 |   return l_1 + l_2 · C_n
6 end
```

Алгоритм 20: Алгоритм генерации по рангу варианта v дерева И/ИЛИ для множества выражений обобщенного языка Дика

```
1 UnrankVariant_GeneralizedCatalanBrackets (r, n, m)
2 begin
3 |   l_1 := r  mod C_n
4 |   l_2 := ⌈r / C_n⌉
5 |   v_1 := UnrankVariant_CatalanBrackets (l_1, n)
6 |   v_2 := UnrankVariant_mPermutationR (l_2, m, n)
7 |   return (v_1, v_2)
8 end
```

Оценка вычислительной сложности Алгоритма 19 складывается из оценки вычислительной сложности алгоритма `RankVariant_CatalanBrackets`, равной $O(n^3)$, и алгоритма `RankVariant_mPermutationR`, равной $O(n)$, а также из оценки вычислительной сложности $O(n)$ для расчета значений C_n и $\overline{P}_m^n$, то есть $O(n^3 + n + n) = O(n^3)$. Аналогично получаем оценку $O(n^3)$ для Алгоритма 20.

Биекция между элементами множества последовательностей правильно вложенных n пар скобок m типов и множества вариантов дерева И/ИЛИ определяется следующим правилом:

С помощью кода поддерева по левой ветви дерева И/ИЛИ получаем последовательность правильно вложенных n пар скобок одного типа. Далее с помощью кода поддерева по правой ветви дерева И/ИЛИ получаем перестановку n пар скобок, взятых из множества m пар скобок разных типов с возможностью повторений. Комбинируя полученные результаты, получаем итоговую последовательность правильно вложенных n пар скобок m типов. В таблице 3.3 представлен пример полученных результатов для случая $n = 2$ и $m = 2$.

Таблица 3.3 — Пример ранжирования множества последовательностей правильно вложенных n пар скобок m типов при $n = 2$ и $m = 2$

Последовательность правильно вложенных скобок двух типов	Последовательность правильно вложенных скобок	Код варианта левого поддерева	Перестановка	Код варианта правого поддерева	Ранг
( ) ( )	( ) ( )	(0,0)	(1,1)	(1,1)	0
(( ))	(( ))	(1,0)	(1,1)	(1,1)	1
( ) [ ]	( ) ( )	(0,0)	(1,2)	(2,1)	2
([ ])	(( ))	(1,0)	(1,2)	(2,1)	3
[ ] ( )	( ) ( )	(0,0)	(2,1)	(1,2)	4
[( )]	(( ))	(1,0)	(2,1)	(1,2)	5
[ ] [ ]	( ) ( )	(0,0)	(2,2)	(2,2)	6
[[ ]]	(( ))	(1,0)	(2,2)	(2,2)	7

Таким образом, совокупность разработанных в данном разделе диссертационной работы алгоритмов комбинаторной генерации реализует алгоритмы ранжирования и генерации по рангу для множества выражений обобщенного языка Дика на примере множества последовательностей правильно вложенных n пар скобок m типов. Оценка вычислительной сложности полученных алгоритмов `GeneralizedCatalanBrackets_Rank`(a, n, m) и `GeneralizedCatalanBrackets_Unrank`(r, n, m) равна $O(n^3)$, что является хуже по сравнению с существующими аналогами, представленными в работе [23]. Однако это показывает возможность применения разработанного модифицированного метода для получения новых алгоритмов комбинаторной генерации.

3.2 Алгоритмы комбинаторной генерации для множества комбинаторных объектов, отражающих вторичную структуру РНК

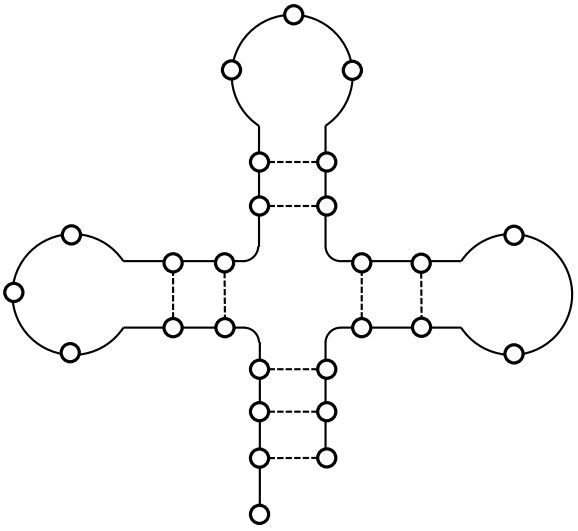
Рибонуклеиновая кислота (РНК) — это один из основных и наиболее важных элементов клеточной структуры любого живого организма [79].

Молекула РНК представляет собой цепочку нуклеотидов, состоящих из азотистых оснований четырех видов: A (Аденин), C (Цитозин), G (Гуанин) и U (Урацил). Вид последовательности нуклеотидов в цепочке РНК образует так называемую первичную структуру РНК. С точки зрения математической записи первичная структура РНК является последовательностью элементов множества $\{A, C, G, U\}$, то есть $a = (a_1, a_2, \ldots)$, где $a_i \in \{A, C, G, U\}$.

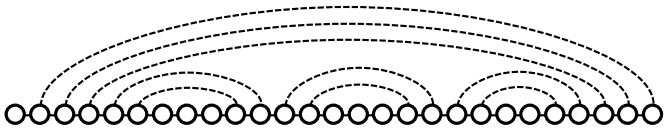
Также в цепочке РНК могут образовываться дополнительные водородные связи между двумя основаниями. Классические Уотсон-Криковские пары [79] могут возникать для пары азотистых оснований A и U, а также для C и G. Таким образом, за счет образования водородных связей в цепочке РНК появляются всевозможные изгибы и петли, что, в свою очередь, усложняет вид структуры молекулы РНК. Вид цепочки РНК с учетом образовавшихся водородных связей представляет собой так называемую вторичную структуру РНК. С точки зрения математической записи вторичная структура РНК может быть представлена в виде структуры графа, в котором помеченные вершины являются нуклеотидами с указанием типа азотистого основания, а ребра показывают наличие связи между нуклеотидами [80].

Молекулы РНК играют важную роль в клеточной обработке нуклеиновых кислот и экспрессии генов, а глубокое понимание данных процессов может быть получено за счет детального изучения, в том числе, вторичной структуры РНК [81]. Например, одним из таких инструментов детального изучения молекул РНК является предсказание вида вторичной структуры РНК по известной последовательности ее нуклеотидов [82–86]. Большая практическая и теоретическая значимость, а также актуальность проведения исследований над вторичными структурами РНК подтверждаются огромным перечнем современных научных работ (например, в [87–90]). При этом поддержка исследований в данной области наук за счет разработки вычислительных алгоритмов является одним из основных направлений современной биоинформатики [91].

На рисунке 3.4 приведен пример представления вторичной структуры цепочки РНК, состоящей из 27 нуклеотидов и 9 водородных связей. Рисунок 3.4(а) показывает вторичную структуру РНК в виде цепочки нуклеотидов (сплошная линия), в которой водородные связи между основаниями (штриховые линии) образовали форму «листа клевера». На рисунке 3.4(б) данная цепочка РНК изображена в виде прямой линии. На рисунке 3.4(в) вторичная структура РНК закодирована последовательностью правильно вложенных скобок с возможностью пропуска скобки.



а)



б)

*(((((***))((***))((**)))))

в)

Рисунок 3.4 — Пример представления вторичной структуры цепочки РНК:
а) в виде структуры «листа клевера»; б) в виде прямой цепочки;
в) в виде последовательности скобок

Рассмотрим комбинаторное множество A_n первичных структур РНК. Данное множество определяется одним параметром n (количество нуклеотидов в цепочке РНК) и является подмножеством множества n-перестановок m элементов с повторениями при $m = 4$. То есть происходит отбор n элементов из множества $m = 4$ элементов $\{A, C, G, U\}$ с возможностью повторений. Таким образом, в качестве алгоритмов комбинаторной генерации для рассматриваемого множества первичных структур РНК длины n могут быть использованы разработанные в разделе 3.1 алгоритмы `mPermutationR_Rank`(a, m, n) и `mPermutationR_Unrank`(r, m, n) при $m = 4$, оценка вычислительной сложности которых равна $O(n)$.

Далее рассмотрим комбинаторное множество $A_{n,m}$ вторичных структур РНК. Данное множество определяется двумя параметрами: n — количество нуклеотидов в цепочке РНК (длина цепочки РНК), m — количество пар нуклеотидов, соединенных водородной связью. Исследования, связанные с разработкой алгоритмов комбинаторной генерации для множества вторичных структур РНК, представлены в работах [92; 93].

В частности, в статье [93] представлены алгоритмы генерации набора m случайных вторичных структур РНК длины n (без уточнения количества пар нуклеотидов, соединенных водородной связью). Оценка вычислительной сложности представленных в работе алгоритмов равна $O(mn\log n)$, при этом требуется предварительный шаг вычислений с оценкой вычислительной сложности равной $O(n^2)$. Для построения алгоритмов комбинаторной генерации применен метод Ф. Флажоле. При этом вид вторичной структуры РНК кодируется последовательностью правильно вложенных скобок с возможностью пропуска скобки.

В статье [92] представлены алгоритмы ранжирования и генерации по рангу для множества вторичных структур РНК длины n с m пар нуклеотидов, соединенных водородной связью. Оценка вычислительной сложности представленных в работе алгоритмов равна $O(n^2+m^2)$, при этом требуется предварительный шаг вычислений с оценкой вычислительной сложности равной $O((n-m)|A_{n,m}|)$, в рамках которого строится вспомогательная таблица. Для построения алгоритмов комбинаторной генерации применен подход, основанный на использовании префиксов числовых последовательностей (аналогично подходу, применяемому в методе Б.Я. Рябко). При этом вид вторичной структуры РНК кодируется упорядоченным деревом, которое в свою очередь также кодируется в виде E-последовательности [92].

Функция мощности рассматриваемого комбинаторного множества $A_{n,m}$ определяется формулой [92]

$$f(n,m)=|A_{n,m}|=S_n^m=S_{n-1}^m+\sum_{i=0}^{m-1}\sum_{j=2i}^{n-2(m-i)-1}S_{n-2-j}^{m-1-i}S_j^i, \tag{3.2}$$

где $S_n^0=1$ и $S_n^m=0$ для $m\geqslant n/2$.

Также значение S_n^m можно вычислить по формуле [65; 94]

$$S_n^m=\frac{1}{n-m}\binom{n-m}{m}\binom{n-m}{m+1}. \tag{3.3}$$

Существует взаимосвязь значений S_n^m с числами Нараяна [65; 95], которые в качестве комбинаторной интерпретации представляют собой разбиение комбинаторных множеств, определяемых числами Каталана, на подмножества в соответствии с некоторым дополнительным параметром. Например, одна из комбинаторных интерпретаций чисел Каталана — количество путей Дика длины $2n$, тогда числа Нараяна показывают общее количество путей дика длины $2n$ с m пиками [47; 95].

Числа Нараяна $N_{n,m}$ определяются следующей производящей функцией двух формальных переменных [65]:

$$N(x,y) = \sum_{n>0} \sum_{m>0} N_{n,m} x^n y^m = \frac{1 - x - xy - \sqrt{(1 - x - xy)^2 - 4x^2 y}}{2x},$$

где

$$N_{n,m} = \frac{1}{n} \binom{n}{m-1} \binom{n}{m}.$$

Взаимосвязь значений S_n^m с числами Нараяна $N_{n,m}$ можно представить через взаимосвязь выражений их производящих функций:

$$S(x,y) = \sum_{n \geqslant 0} \sum_{m \geqslant 0} S_n^m x^n y^m = 1 + \frac{N(x,xy)}{xy}.$$

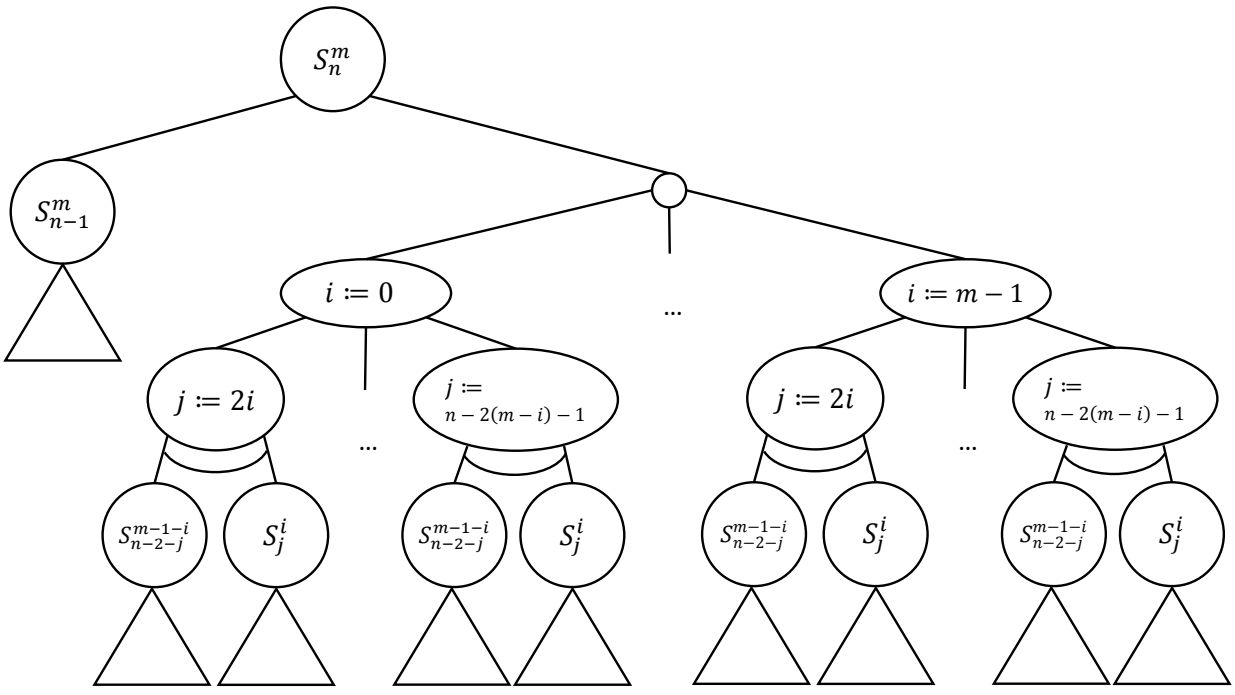


Рисунок 3.5 — Дерево И/ИЛИ для множества вторичных структур РНК

Рассмотрим подробно процесс разработки алгоритмов ранжирования и генерации по рангу для множества вторичных структур РНК длины n с m пар нуклеотидов, соединенных водородной связью, с помощью модифицированного метода построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ. Выражение (3.2) для функции мощности комбинаторного множества вторичных структур РНК принадлежит требуемой алгебре $\{\mathbb{N}, +, \times, R\}$, поэтому на основе данного выражения можно построить соответствующую комбинаторному множеству структуру дерева И/ИЛИ (рисунок 3.5).

Биекция между элементами комбинаторного множества и множества вариантов дерева И/ИЛИ определяется следующим правилом:

Вторичная структура цепочки РНК кодируется последовательностью $a = (a_1, \ldots, a_n)$ правильно вложенных скобок с возможностью пропуска скобки, где $a_i \in \{$ '(', ')', '*' $\}$, '*' обозначает пропуск скобки. Если $a_1 =$ '*', то это соответствует переходу по левой ветви от корня в дереве И/ИЛИ, в вариант дерева записывается значение 0 и далее рассматривается последовательность $(a_2, \ldots, a_n)$. Иначе выполняем переход по правой ветви от корня в дереве И/ИЛИ и последовательность $a = (a_1, \ldots, a_n)$ рассматривается в форме '('$,s_1,$')'s_2, где $s_1 = (a_2, \ldots, a_{n-J-1})$ и $s_2 = (a_{n-J+1}, \ldots, a_n)$ — последовательности правильно вложенных скобок с возможностью пропуска скобки. При этом вводятся вспомогательные параметры: I — количество пар скобок в последовательности s_2, J — количество элементов в последовательности s_2. В вариант дерева записывается набор значений $(0, I, J)$ и далее аналогичным образом рассматриваются последовательности $s_1 = (a_2, \ldots, a_{n-J-1})$ и $s_2 = (a_{n-J+1}, \ldots, a_n)$.

В Алгоритме 21 и Алгоритме 22 представлена реализация биекции между элементами комбинаторного множества вторичных структур РНК и множества вариантов дерева И/ИЛИ. Алгоритм 21 для вторичной структуры РНК длины n с m пар нуклеотидов, соединенных водородной связью, которая представлена в виде последовательности правильно вложенных m пар скобок с возможностью пропуска скобки, определяет соответствующий вариант v дерева И/ИЛИ. Алгоритм 22 выполняет обратное преобразование: для заданного варианта v дерева И/ИЛИ определяется соответствующая вторичная структура РНК длины n с m пар нуклеотидов, соединенных водородной связью, которая представлена в виде последовательности правильно вложенных m пар скобок с возможностью пропуска скобки.

Алгоритм 21: Алгоритм отображения последовательности a правильно вложенных m пар скобок с возможностью пропуска скобки в вариант v дерева И/ИЛИ

```
1  RNASSToVariant (a, n, m)
2  begin
3      if m = 0 then return ()
4      if a_1 = '*' then v := (0, RNASSToVariant ((a_2, ..., a_n), n − 1, m) )
5      else
6          s_1 := 0
7          s_2 := 0
8          I := 0
9          J := 0
10         for i := 1 to n do
11             if a_i = '(' then s_1 := s_1 + 1
12             if a_i = ')' then s_2 := s_2 + 1
13             if s_1 = s_2 then
14                 I := m − s_1
15                 J := n − i
16                 break
17             end
18         end
19         vv_1 := RNASSToVariant ((a_2, ..., a_{n−J−1}), n − 2 − J, m − 1 − I)
20         vv_2 := RNASSToVariant ((a_{n−J+1}, ..., a_n), J, I)
21         v := (1, I, J, vv_1, vv_2)
22     end
23     return v
24 end
```

Оценка вычислительной сложности Алгоритма 21 равна $O(m(n+m))$. Данное значение складывается из количества итераций цикла for (n раз), сложности $O(m)$ для расчета v и количества рекурсивных вызовов (m раз).

Оценка вычислительной сложности Алгоритма 22 равна $O(nm)$. Данное значение складывается из вычислительной сложности $O(n)$ для расчета последовательности a и количества рекурсивных вызовов (m раз).

Алгоритм 22: Алгоритм отображения варианта v дерева И/ИЛИ в последовательность a правильно вложенных m пар скобок с возможностью пропуска скобки

```
1  VariantToRNASS (v, n, m)
2  begin
3      if m = 0 then
4          (a_1, ..., a_n) := (0, ..., 0)
5          return (a_1, ..., a_n)
6      end
7      if v_1 := 0 then a :=concat ( 0, VariantToRNASS (v_2, n − 1, m) )
8      else
9          I := v_2
10         J := v_3
11         aa_1 := VariantToRNASS (v_4, n − 2 − J, m − 1 − I)
12         aa_2 := VariantToRNASS (v_5, J, I)
13         a :=concat ( '(', aa_1, ')', aa_2 )
14     end
15     return a
16 end
```

На основе общих алгоритмов ранжирования (Алгоритм 11) и генерации по рангу (Алгоритм 12) вариантов дерева И/ИЛИ были получены соответствующие алгоритмы (Алгоритм 23 и Алгоритм 24) для дерева И/ИЛИ для множества вторичных структур РНК длины n с m пар нуклеотидов, соединенных водородной связью, которые представлены в виде последовательности правильно вложенных m пар скобок с возможностью пропуска скобки.

Совокупность Алгоритма 21 и Алгоритма 23 в виде последовательного их применения образует алгоритм $\texttt{RNASS\_Rank}(a, n, m)$ для ранжирования множества вторичных структур РНК длины n с m пар нуклеотидов, соединенных водородной связью. Совокупность Алгоритма 24 и Алгоритма 22 в виде последовательного их применения образуют алгоритм $\texttt{RNASS\_Unrank}(r, n, m)$ для генерации по рангу множества вторичных структур РНК длины n с m пар нуклеотидов, соединенных водородной связью.

Алгоритм 23: Алгоритм ранжирования варианта v дерева И/ИЛИ для множества вторичных структур РНК

```
1  RankVariant_RNASS (v, n, m)
2  begin
3      if m = 0 then return 0
4      if m ⩾ n/2 then return 0
5      if v_1 = 0 then r := RankVariant_RNASS (v_2, n − 1, m)
6      else
7          I := v_2
8          J := v_3
9          r := S_{n−1}^m + Σ_{i=0}^{I−1} Σ_{j=2i}^{n−2(m−i)−1} S_{n−2−j}^{m−1−i} S_j^i + Σ_{j=2I}^{J−1} S_{n−2−j}^{m−1−I} S_j^I
10         r := r+ RankVariant_RNASS (v_4, n − 2 − J, m − 1 − I)
11         r := r + S_{n−2−J}^{m−1−I} RankVariant_RNASS (v_5, J, I)
12     end
13     return r
14 end
```

Оценка вычислительной сложности Алгоритма 23 равна $O(m^2(n-m))$. Данное значение складывается из количества итераций суммирования при расчете значения r (максимум $(m-1)$ раз для внешнего и $(n-2m)$ раз для внутреннего оператора суммирования), вычислительной сложности $O(m)$ для расчета S_n^m (при $m < n-2m$) и количества рекурсивных вызовов (m раз).

Оценка вычислительной сложности Алгоритма 24 равна $O(m^2(n-m))$. Данное значение складывается из количества итераций циклов for (m раз для внешнего и $(n-2m)$ раз для внутреннего цикла), вычислительной сложности $O(m)$ для расчета S_n^m (при $m < n-2m$) и количества рекурсивных вызовов (m раз).

Таким образом, оценка вычислительной сложности разработанных алгоритмов комбинаторной генерации `RNASS_Rank`(a, n, m) и `RNASS_Unrank`(r, n, m) равна $O(m^2(n-m))$. Если предварительно вычислить все значения S_j^i для $j = 1, \ldots, n$ и $i = 1, \ldots, m$ за время $O(nm^2)$ и хранить их в памяти в виде таблицы размера $n \times m$, тогда оценка вычислительной сложности снижается до $O(m(n-m))$.

Алгоритм 24: Алгоритм генерации по рангу варианта v дерева И/ИЛИ для множества вторичных структур РНК

```
1  UnrankVariant_RNASS (r, n, m)
2  begin
3      if m = 0 then return ()
4      if r < S^m_{n-1} then v := (0, UnrankVariant_RNASS (r, n − 1, m) )
5      else
6          r := r − S^m_{n-1}
7          sum := 0
8          flag := 0
9          I := 0
10         for i := 0 to m − 1 do
11             J := 2i
12             for j := 2i to n − 2(m − i) − 1 do
13                 w := S^{m-1-i}_{n-2-j} S^i_j
14                 if sum + w > r then
15                     flag := 1
16                     break
17                 end
18                 else sum := sum + w
19                 J := J + 1
20             end
21             if flag = 1 then break
22             I := I + 1
23         end
24         r := r − sum
25         w := S^{m-1-I}_{n-2-J}
26         vv_1 := RNASSToVariant (r mod w, n − 2 − J, m − 1 − I)
27         vv_2 := RNASSToVariant (⌈r/w⌉, J, I)
28         v := (1, I, J, vv_1, vv_2)
29     end
30     return v
31 end
```

В таблице 3.4 представлен пример полученных результатов для случая $n = 8$ и $m = 3$.

Таблица 3.4 — Пример ранжирования множества вторичных структур РНК длины n с m пар нуклеотидов, соединенных водородной связью, при $n = 8$ и $m = 3$

Последовательность правильно вложенных скобок	Код варианта дерева И/ИЛИ	Ранг
$*(((*)))$	(0,(1,0,0,(1,0,0,(1,0,0,(),()),()),()))	0
$(*((*)))$	(1,0,0,(0,(1,0,0,(1,0,0,(),()),())),())	1
$((*(*)))$	(1,0,0,(1,0,0,(0,(1,0,0,(),())),()),())	2
$(((**)))$	(1,0,0,(1,0,0,(1,0,0,(),()),()),())	3
$(((*)*))$	(1,0,0,(1,0,0,(1,0,1,(),()),()),())	4
$(((*))*)$	(1,0,0,(1,0,1,(1,0,0,(),()),()),())	5
$((*)(*))$	(1,0,0,(1,1,3,(),(1,0,0,(),())),())	6
$(((*)))*$	(1,0,1,(1,0,0,(1,0,0,(),()),()),())	7
$((*))(*)$	(1,1,3,(1,0,0,(),()),(1,0,0,(),()))	8
$(*)((*))$	(1,2,5,(),(1,0,0,(1,0,0,(),()),()))	9

Разработанные алгоритмы ранжирования `RNASS_Rank`(a, n, m) и генерации по рангу `RNASS_Unrank`(r, n, m) для множества комбинаторных объектов, отражающих вторичную структуру РНК длины n с m пар нуклеотидов, соединенных водородной связью, имеют лучшую совокупную оценку вычислительной сложности в сравнении с существующими аналогами. Оценка вычислительной сложности разработанных алгоритмов равна $O(m^2(n - m))$, при этом не требуется никаких вспомогательных предварительных вычислений. Оценка вычислительной сложности аналогов разработанных алгоритмов, которые представлены в статье [92], равна $O(n^2 + m^2)$. Однако в данном случае необходим предварительный шаг вычислений с оценкой вычислительной сложности равной $O((n-m)|A_{n,m}|)$, где $|A_{n,m}|$ определяется биномиальными коэффициентами (выражения (3.2) и (3.3)), то есть является факториальной. Таким образом, в совокупности с предварительными вычислениями итоговая вычислительная сложность аналогов разработанных алгоритмов является факториальной, что является худшей оценкой по сравнению с разработанными алгоритмами.

3.3 Алгоритмы комбинаторной генерации для множества комбинаторных объектов, определяемых числовым треугольником Эйлера-Каталана

Рассмотрим процесс разработки алгоритмов ранжирования и генерации по рангу с помощью модифицированного метода построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ для комбинаторного множества, для которого не известно выражение функции мощности, принадлежащее алгебре $\{\mathbb{N}, +, \times, R\}$, но известно выражение производящей функции для последовательности значений функции мощности.

В качестве такого комбинаторного множества рассмотрим множество комбинаторных объектов, определяемых числовым треугольником Эйлера-Каталана (числовая последовательность A316773 в OEIS [65]). Элементы числового треугольника Эйлера-Каталана $EC_{n,m}$ определяются следующей экспоненциальной производящей функцией двух формальных переменных [65]:

$$EC(x,y) = \sum_{n \geqslant 0} \sum_{m \geqslant 0} \frac{EC_{n,m}}{n!} x^n y^m = \frac{y-1}{y - e^{C(x)(y-1)}},$$

где $C(x)$ — производящая функция чисел Каталана (2.4).

Таблица 3.5 — Несколько первых значений $EC_{n,m}$

$n \setminus m$	0	1	2	3	4	5	6	7	8
0	1								
1	1	0							
2	3	1	0						
3	19	10	1	0					
4	193	119	23	1	0				
5	2721	1806	466	46	1	0			
6	49171	34017	10262	1502	87	1	0		
7	1084483	770274	255795	47020	4425	162	1	0	
8	28245729	20429551	7235853	1539939	193699	12525	303	1	0

Производящая функция $EC(x,y)$ имеет связь с производящими функциями чисел Каталана C_n и чисел Эйлера первого рода E_n^m [40]. Данный факт позволяет получать различные комбинаторные интерпретации для значений $EC_{n,m}$.

Например, в качестве одной из комбинаторных интерпретаций значение $EC_{n,m}$ является значением функции мощности для комбинаторного множества помеченных двоичных деревьев размера n с m подъемами на левой ветви [43]. Это такая структура дерева, которая состоит из n вершин, каждая из которых помечена уникальным значением от 1 до n и имеет максимум два сына (левый и правый). Под левой ветвью дерева понимается дерево, полученное путем удаления всех правых сыновей. Если рассмотреть последовательность значений пометок вершин левой ветви дерева, начиная от корня, то в ней будет ровно m подъемов (ровно m значений последовательности будут больше, чем предшествующее им значение). На рисунке 3.6 показаны все возможные варианты таких деревьев для $n = 3$ и $m = 1$.

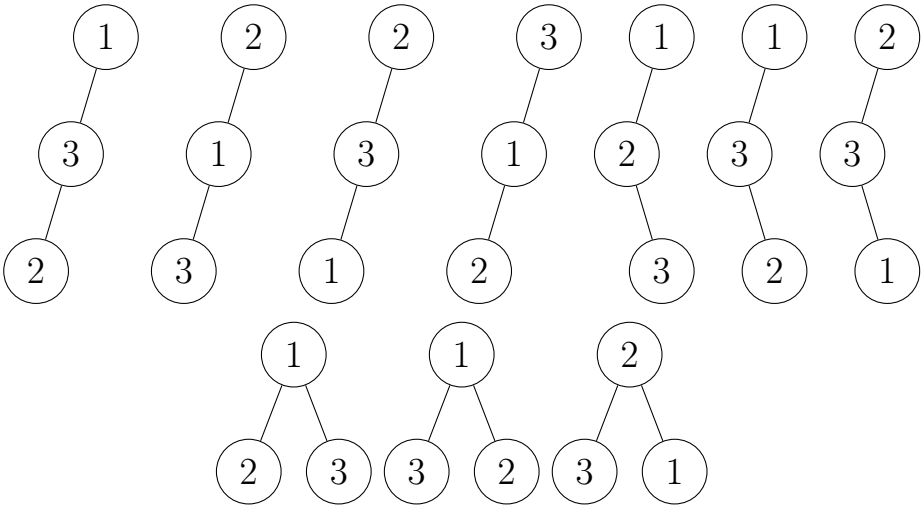


Рисунок 3.6 — Все возможные варианты помеченных двоичных деревьев размера $n = 3$ с $m = 1$ подъемом на левой ветви

Еще одна комбинаторная интерпретация: значение $EC_{n,m}$ является значением функции мощности для комбинаторного множества помеченных путей Дика длины $2n$ с m подъемами на возвратных шагах [43]. Это такие пути Дика длины $2n$, в которых каждый спуск помечен уникальным значением от 1 до n. Если рассмотреть последовательность значений пометок спусков пути Дика, то в ней будет ровно m подъемов (ровно m значений последовательности будут больше, чем предшествующее им значение). Путь Дика длины $2n$ — это решетчатый путь на верхней плоскости (выше оси абсцисс), который начинается в координате (0,0), заканчивается в ($2n$,0) и состоит из шагов (1,1), называемых подъемами, и $(1, -1)$, называемых спусками [96]. Возвратный шаг — это спуск, возвращающий путь Дика на ось абсцисс. На рисунке 3.7 показаны все возможные варианты таких путей Дика для $n = 3$ и $m = 1$.

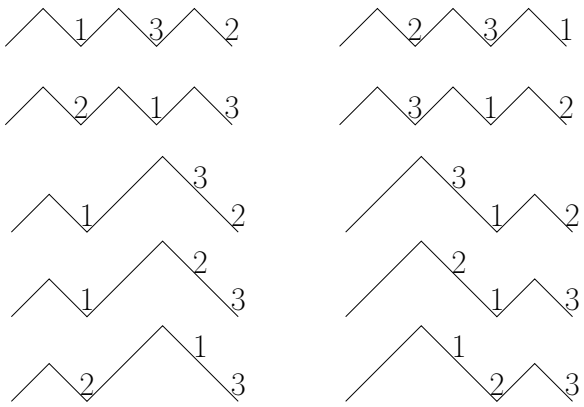


Рисунок 3.7 — Все возможные варианты помеченных путей Дика длины $2n = 6$ с $m = 1$ подъемом на возвратных шагах

Чтобы построить структуру дерева И/ИЛИ для множества комбинаторных объектов, определяемых числовым треугольником Эйлера-Каталана, необходимо получить выражение функции мощности, принадлежащее алгебре $\{\mathbb{N}, +, \times, R\}$. Для этого воспользуемся предложенным в разделе 2.2 методом получения явных выражений коэффициентов производящих функций.

Представим производящую функцию $EC(x,y)$ в виде композиции

$$EC(x,y) = E\left(C(x),y\right), \tag{3.4}$$

где

$$E(x,y) = \sum_{n\geqslant 0}\sum_{m\geqslant 0}\frac{E_n^m}{n!}x^n y^m = \frac{y-1}{y-e^{x(y-1)}}$$

это производящая функция чисел Эйлера первого рода E_n^m [39; 56].

Композита $C^{\Delta}(n,k)$ производящей функции Каталана $C(x)$ (2.4) определяется выражением (2.5), которое при $n > 0$ совпадает с явным выражением коэффициентов производящей функции транспонированного числового треугольника Каталана CT_n^k (2.6). Используя (2.2) для композиции (3.4), получим

$$\begin{aligned} EC_{n,m} &= \begin{cases} n!E_0^m & n=0; \\ n!\sum\limits_{k=1}^{n} C^{\Delta}(n,k)E_k^m, & n>0. \end{cases} \\ &= \begin{cases} 1, & n=0, m=0; \\ n!\sum\limits_{k=m+1}^{n} CT_n^k \frac{E_k^m}{k!}, & n>0, m>0. \end{cases} \\ &= \begin{cases} 1, & n=0, m=0; \\ \sum\limits_{k=m+1}^{n} CT_n^k E_k^m P_n^{n-k}, & n>0, m>0. \end{cases} \end{aligned} \tag{3.5}$$

Используемое в выражении (3.5) значение P_n^m соответствует количеству всех m-перестановок n элементов.

Рассмотрим подробно процесс разработки алгоритмов ранжирования и генерации по рангу на примере множества помеченных путей Дика длины $2n$ с m подъемами на возвратных шагах с помощью модифицированного метода построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ. Выражение (3.5) для функции мощности комбинаторного множества помеченных путей Дика длины $2n$ с m подъемами на возвратных шагах принадлежит требуемой алгебре $\{\mathbb{N}, +, \times, R\}$, поэтому на основе данного выражения можно построить соответствующую комбинаторному множеству структуру дерева И/ИЛИ (рисунок 3.8).

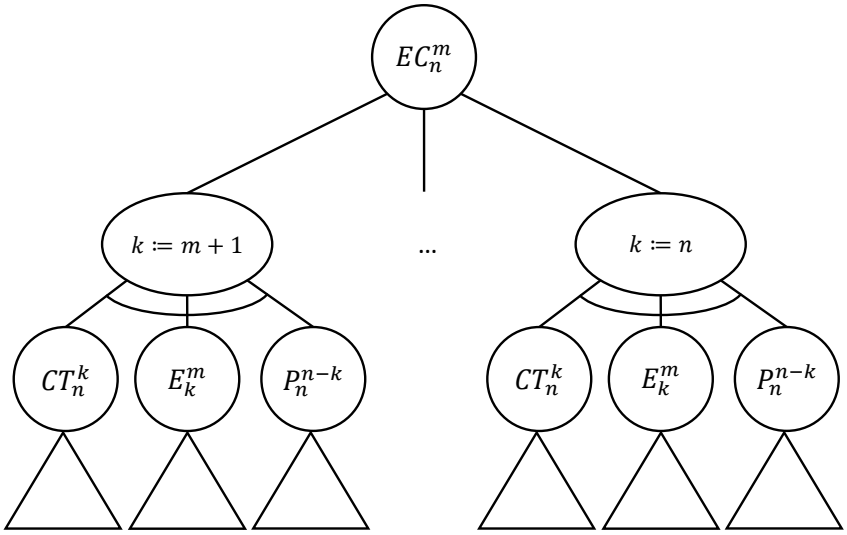


Рисунок 3.8 — Дерево И/ИЛИ для множества помеченных путей Дика длины $2n$ с m подъемами на возвратных шагах

Биекция между элементами комбинаторного множества помеченных путей Дика длины $2n$ с m подъемами на возвратных шагах и множества вариантов дерева И/ИЛИ определяется следующим правилом:

– количество возвратных шагов определяет значение параметра k (выбирается соответствующая ветвь ИЛИ-узла дерева И/ИЛИ);

– поддерево с вершиной CT_n^k определяет вариант пути Дика длины $2n$ с k возвратными шагами;

– поддерево с вершиной E_k^m определяет вариант перестановки пометок k возвратных шагов, чтобы они образовывали ровно m подъемов;

– поддерево с вершиной P_n^{n-k} определяет вариант значений пометок и вариант их перестановки для оставшихся (не возвратных) $n-k$ спусков.

Таким образом, чтобы получить алгоритмы комбинаторной генерации на основе построенного дерева И/ИЛИ, необходимо разработать алгоритмы комбинаторной генерации для множеств: путей Дика длины $2n$ с m возвратными шагами, число которых определяется значением CT_n^m [66]; перестановок n элементов с m подъемами, число которых определяется значением E_n^m; m-перестановок n элементов, число которых определяется значением P_n^m.

В диссертационной работе [26] представлены алгоритмы комбинаторной генерации для множества перестановок n элементов с m подъемами. Здесь в качестве выражения функции мощности множества перестановок n элементов с m подъемами, принадлежащего алгебре $\{\mathbb{N}, +, \times, R\}$, использовано известное рекуррентное соотношение для чисел Эйлера первого рода [95]

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

Данному выражению соответствует структура дерева И/ИЛИ, представленная на рисунке 3.9.

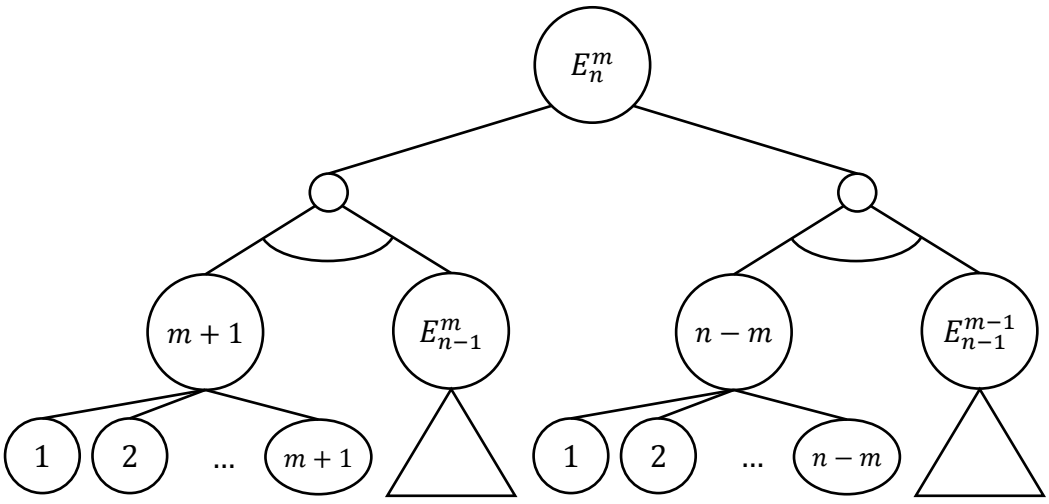


Рисунок 3.9 — Дерево И/ИЛИ для множества перестановок n элементов с m подъемами

Биекция между элементами комбинаторного множества и множества вариантов дерева И/ИЛИ определяется следующим правилом [42]:

При переходе по левой ветви наблюдается уменьшение на единицу параметра n, при этом параметр m не изменяется. Данное изменение говорит о том, что элемент со значением n не увеличивает количество подъемов в перестановке n элементов, а номер выбранного листа указывает соответствующий вариант размещения элемента со значением n в перестановке из возможных $m+1$ вариантов. При переходе по правой ветви наблюдается уменьшение на единицу

параметров n и m. Данное изменение говорит о том, что элемент со значением n увеличивает количество подъемов в перестановке n элементов, а номер выбранного листа указывает соответствующий вариант размещения элемента со значением n в перестановке из возможных $n - m$ вариантов.

Оценка вычислительной сложности алгоритмов комбинаторной генерации для множества перестановок n элементов с m подъемами:

– для алгоритма `PermutationAToVariant`(a, n, m) — $O(n^2)$;
– для алгоритма `VariantToPermutationA`(v, n, m) — $O(n^2)$;
– для алгоритма `RankVariant_PermutationA`(v, n, m) — $O(nm^2)$;
– для алгоритма `UnrankVariant_PermutationA`(r, n, m) — $O(nm^2)$.

Также в диссертационной работе [26] представлены алгоритмы комбинаторной генерации для множества m-перестановок n элементов. Здесь в качестве выражения функции мощности множества m-перестановок n элементов, принадлежащего алгебре $\{\mathbb{N}, +, \times, R\}$, использована формула

$$P_n^m = \frac{n!}{(n-m)!} = \frac{n!}{m!(n-m)!} m! = C_n^m P_m.$$

Данному выражению соответствует структура дерева И/ИЛИ, представленная на рисунке 3.10.

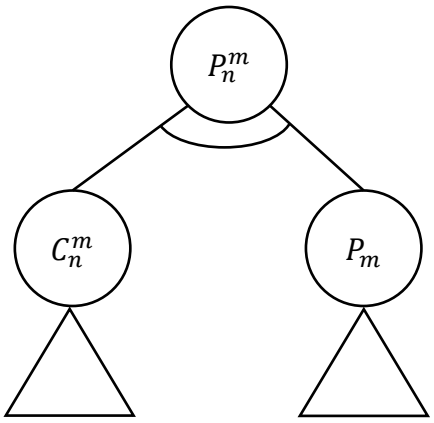


Рисунок 3.10 — Дерево И/ИЛИ для множества m-перестановок n элементов

Биекция между элементами комбинаторного множества и множества вариантов дерева И/ИЛИ определяется следующим правилом:

Левое поддерево с вершиной C_n^m определяет какие m элементов будут отобраны из множества n элементов для их участия в перестановке. Правое поддерево с вершиной P_m определяет вариант перестановки для отобранных m элементов.

Оценка вычислительной сложности алгоритмов комбинаторной генерации для множества m-перестановок n элементов:

– для алгоритма `mPermutationToVariant`$(a, n, m) - O(m^2)$;
– для алгоритма `VariantTomPermutation`$(v, n, m) - O(m^2)$;
– для алгоритма `RankVariant_mPermutation`$(v, n, m) - O(nm)$;
– для алгоритма `UnrankVariant_mPermutation`$(r, n, m) - O(nm)$.

Рассмотрим процесс разработки алгоритмов ранжирования и генерации по рангу для множества путей Дика длины $2n$ с m возвратными шагами. В качестве выражения функции мощности множества путей Дика длины $2n$ с m возвратными шагами, принадлежащего алгебре $\{\mathbb{N}, +, \times, R\}$, использована следующая формула [66]:

$$CT_n^m = CT_{n-1}^{m-1} + CT_n^{m+1}, \quad CT_n^0 = 0, \quad CT_n^n = 1.$$

Данному выражению соответствует структура дерева И/ИЛИ, представленная на рисунке 3.11.

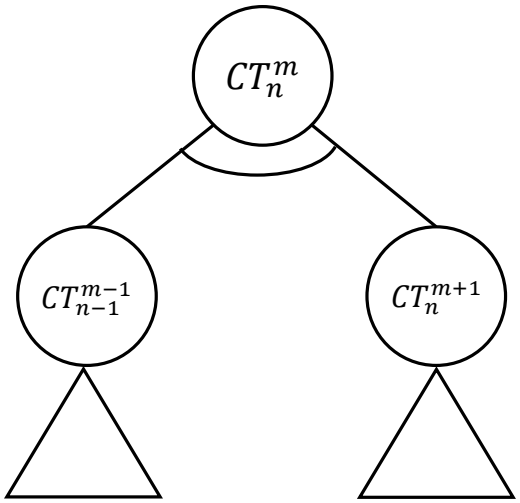


Рисунок 3.11 — Дерево И/ИЛИ для множества путей Дика длины $2n$ с m возвратными шагами

Биекция между элементами комбинаторного множества и множества вариантов дерева И/ИЛИ определяется следующим правилом:

Если путь Дика находится на оси абсцисс, то необходимо совершить один подъем. Каждый переход по левой ветви дерева И/ИЛИ соответствует спуску, а переход по правой ветви — подъему. Если дошли до листа дерева И/ИЛИ ($m = n$), тогда необходимо сделать дополнительно n спусков.

Для компактного представления будем кодировать варианты дерева И/ИЛИ последовательностью значений номеров выбранных в нем дуг ИЛИ-узлов, полученных при его левостороннем обходе (0 соответствует переходу по левой ветви, 1 — переходу по правой ветви). Также пути Дика длины $2n$ будем кодировать в виде последовательности шагов $a = (a_1, \ldots, a_{2n})$, где $a_i \in \{\text{'u','d'}\}$, 'u' — обозначение шага-подъема, 'd' — обозначение шага-спуска.

В Алгоритме 25 и Алгоритме 26 представлена реализация биекции между элементами комбинаторного множества путей Дика длины $2n$ с m возвратными шагами и множества вариантов дерева И/ИЛИ.

Алгоритм 25: Алгоритм отображения последовательности a для пути Дика длины $2n$ с m возвратными шагами в вариант v дерева И/ИЛИ

```
1  DyckPathRSToVariant (a, n, m)
2  begin
3      v := ()
4      l := 0
5      k := 0
6      for i := 1 to 2n do
7          if l = 0 then l := l + 1
8          else
9              if a_i ='d' then
10                 l := l − 1
11                 v := concat ( v, 0 )
12             end
13             else
14                 l := l + 1
15                 k := k + 1
16                 v := concat ( v, 1 )
17             end
18             if k = n − m then return v
19         end
20     end
21 end
```

Алгоритм 26: Алгоритм отображения варианта v дерева И/ИЛИ в последовательность a для пути Дика длины $2n$ с m возвратными шагами

```
1  VariantToDyckPathRS (v, n, m)
2  begin
3      a := (a_1, ..., a_2n)
4      l := 0
5      k := 0
6      for i := 1 to 2n do
7          if l = 0 then
8              l := l + 1
9              a_i := 'u'
10         end
11         else
12             k := k + 1
13             if k > |v| then
14                 l := l − 1
15                 a_i := 'd'
16             end
17             else
18                 if v_k = 0 then
19                     l := l − 1
20                     a_i := 'd'
21                 end
22                 else
23                     l := l + 1
24                     a_i := 'u'
25                 end
26             end
27         end
28     end
29     return a
30 end
```

Алгоритм 25 для последовательности $a = (a_1, \ldots, a_{2n})$ для пути Дика длины $2n$ с m возвратными шагами определяет соответствующий вариант v дерева И/ИЛИ. Алгоритм 26 выполняет обратное преобразование.

На основе общих алгоритмов ранжирования (Алгоритм 11) и генерации по рангу (Алгоритм 12) вариантов дерева И/ИЛИ были получены соответствующие алгоритмы (Алгоритм 27 и Алгоритм 28) для дерева И/ИЛИ для множества путей Дика длины $2n$ с m возвратными шагами.

Алгоритм 27: Алгоритм ранжирования варианта v дерева И/ИЛИ, соответствующего пути Дика длины $2n$ с m возвратными шагами

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

Алгоритм 28: Алгоритм генерации по рангу варианта v дерева И/ИЛИ, соответствующего пути Дика длины $2n$ с m возвратными шагами

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

Совокупность Алгоритма 25 и Алгоритма 27 в виде последовательного их применения образует алгоритм `DyckPathRS_Rank`(a, n, m) для ранжирования множества путей Дика длины $2n$ с m возвратными шагами. Совокупность Алгоритма 28 и Алгоритма 26 в виде последовательного их применения образует алгоритм `DyckPathRS_Unrank`(r, n, m) для генерации по рангу множества путей Дика длины $2n$ с m возвратными шагами. В таблице 3.6 представлен пример полученных результатов для случая $n = 5$ и $m = 2$.

Таблица 3.6 — Пример ранжирования множества путей Дика длины $2n$ с m возвратными шагами при $n = 5$ и $m = 2$

Путь Дика	Код варианта дерева И/ИЛИ	Ранг
u d u u d u d u d d	(0,1,0,1,0,1)	0
u d u u d u u d d d	(0,1,0,1,1)	1
u d u u u d d u d d	(0,1,1,0,0,1)	2
u d u u u d u d d d	(0,1,1,0,1)	3
u d u u u u d d d d	(0,1,1,1)	4
u u d d u u d u d d	(1,0,0,1,0,1)	5
u u d d u u u d d d	(1,0,0,1,1)	6
u u d u d d u u d d	(1,0,1,0,0,1)	7
u u d u d u d d u d	(1,0,1,0,1)	8
u u d u u d d d u d	(1,0,1,1)	9
u u u d d d u u d d	(1,1,0,0,0,1)	10
u u u d d u d d u d	(1,1,0,0,1)	11
u u u d u d d d u d	(1,1,0,1)	12
u u u u d d d d u d	(1,1,1)	13

Оценка вычислительной сложности Алгоритма 25 и Алгоритма 26 равна $O(n)$. Данное значение складывается из количества итераций цикла for ($2n$ раз).

Для Алгоритма 27 и Алгоритма 28 оценка вычислительной сложности равна $O((n-m)^2)$. Данное значение складывается из вычислительной сложности $O(n-m)$ для расчета значения CT_n^m и количества рекурсивных вызовов ($2n - m$ раз).

Используя алгоритмы комбинаторной генерации для множеств путей Дика длины $2n$ с m возвратными шагами, перестановок n элементов с m подъемами и m-перестановок n элементов, можно построить алгоритмы комбинаторной генерации для множества помеченных путей Дика длины $2n$ с m подъемами на возвратных шагах. Для этого необходимо применить общие алгоритмы ранжирования (Алгоритм 11) и генерации по рангу (Алгоритм 12) вариантов дерева И/ИЛИ для дерева, представленного на рисунке 3.8.

В результате были получены соответствующие алгоритмы (Алгоритм 29 и Алгоритм 30) для дерева И/ИЛИ для множества помеченных путей Дика длины $2n$ с m подъемами на возвратных шагах.

В данных алгоритмах варианты дерева И/ИЛИ (рисунок 3.8) представлены тройкой последовательностей $v = (v_1,v_2,v_3)$, первая из которых v_1 соответствует коду поддерева по левой ветви выбранного И-узла дерева И/ИЛИ, вторая v_2 — коду поддерева по средней ветви, третья v_3 — коду поддерева по правой ветви. При этом биекция между элементами комбинаторного множества помеченных путей Дика длины $2n$ с m подъемами на возвратных шагах и множества вариантов дерева И/ИЛИ определяется по представленным ранее правилам.

Алгоритм 29: Алгоритм ранжирования варианта v дерева И/ИЛИ, соответствующего помеченному пути Дика длины $2n$ с m подъемами на возвратных шагах

```
1 RankVariant_DyckPathARS (v, n, m)
2 begin
3     k := |v_2| / 2
4     l_1 := RankVariant_DyckPathRS (v_1, n, k)
5     l_2 := RankVariant_PermutationA (v_2, k, m)
6     l_3 := RankVariant_mPermutation (v_3, n, n − k)
7     r := l_1 + CT_n^k (l_2 + E_k^m l_3) + Σ_{i=m+1}^{k−1} CT_n^i E_i^m P_n^{n−i}
8     return r
9 end
```

Алгоритм 30: Алгоритм генерации по рангу варианта v дерева И/ИЛИ, соответствующего помеченному пути Дика длины $2n$ с m подъемами на возвратных шагах

```
1  UnrankVariant_DyckPathARS (r, n, m)
2  begin
3      sum := 0
4      for i := m + 1 to n do
5          s := CT_n^i E_i^m P_n^{n-i}
6          if sum + s > r then
7              r := r − sum
8              k := i
9              break
10         end
11         else sum := sum + s
12     end
13     l_1 := r  mod CT_n^k
14     r := ⌈ r / CT_n^k ⌉
15     l_2 := r  mod E_k^m
16     l_3 := ⌈ r / E_k^m ⌉
17     v_1 := UnrankVariant_DyckPathRS (l_1, n, k)
18     v_2 := UnrankVariant_PermutationA (l_2, k, m)
19     v_3 := UnrankVariant_mPermutation (l_3, n, n − k)
20     return (v_1, v_2, v_3)
21 end
```

Оценка вычислительной сложности Алгоритма 29 складывается из оценки вычислительной сложности $O((n-m)^2)$ алгоритма `RankVariant_DyckPathRS`, $O(nm^2)$ для алгоритма `RankVariant_PermutationA`, $O(nm)$ для алгоритма `RankVariant_mPermutation`, а также из оценки вычислительной сложности $O(nm(n-m))$ для расчета суммы, то есть $O((n-m)^2+nm^2+nm+nm(n-m)) = O(nm(n-m))$. Аналогично получаем оценку $O(nm(n-m))$ для Алгоритма 30.

3.4 Выводы по главе

Основным результатом данной главы является алгоритмическое обеспечение, разработанное в ходе апробации представленного в главе 2 модифицированного метода построения алгоритмов комбинаторной генерации.

В разделе 3.1 были разработаны алгоритмы ранжирования и генерации по рангу для множества выражений обобщенного языка Дика на примере множества последовательностей правильно вложенных n пар скобок m типов. Оценка вычислительной сложности полученных алгоритмов комбинаторной генерации равна $O(n^3)$, что является хуже по сравнению с существующими аналогами. Однако это показывает возможность применения разработанного модифицированного метода для получения новых алгоритмов комбинаторной генерации.

В разделе 3.2 были разработаны алгоритмы ранжирования и генерации по рангу для множества комбинаторных объектов, отражающих вторичную структуру РНК длины n с m пар нуклеотидов, соединенных водородной связью. Оценка вычислительной сложности разработанных алгоритмов равна $O(m^2(n-m))$. По сравнению с существующими аналогами данная оценка является лучшей, так как не требуется никаких предварительных вычислений.

В разделе 3.3 были разработаны алгоритмы ранжирования и генерации по рангу для множества комбинаторных объектов, определяемых треугольником Эйлера-Каталана, на примере множества помеченных путей Дика длины $2n$ с m подъемами на возвратных шагах. Оценка вычислительной сложности разработанных алгоритмов равна $O(m^2(n-m))$. Для данного комбинаторного множества не было известно выражение функции мощности, принадлежащее алгебре $\{\mathbb{N}, +, \times, R\}$, но было известно выражение производящей функции $EC(x,y)$ для последовательности значений функции мощности. В результате с помощью метода получения явных выражений коэффициентов производящих функций (Шаг 2 модифицированного метода) было получено выражение функции мощности EC_n^m, принадлежащее алгебре $\{\mathbb{N}, +, \times, R\}$.

Таким образом, по итогам апробации представленного в главе 2 модифицированного метода построения алгоритмов комбинаторной генерации можно сделать вывод о том, что применение данного метода является эффективным с точки зрения построения новых алгоритмов комбинаторной генерации как для известных комбинаторных множеств, так и для новых.

Глава 4. Программная реализация разработанных алгоритмов комбинаторной генерации

В качестве проверки работоспособности и демонстрации результатов выполнения разработанных в главе 3 алгоритмов комбинаторной генерации, а также для автоматизации процессов вычислений в рамках данных алгоритмов, было разработано программное обеспечение для ранжирования и генерации по рангу элементов комбинаторных множеств.

4.1 Выбор средства программной реализации

Для программной реализации была выбрана система компьютерной алгебры «Maxima» [97].

Maxima — программа для выполнения математических вычислений, символьных преобразований и построения графиков [98; 99]. На сегодняшний день пакет достаточно активно развивается, и во многих отношениях не уступает другим развитым системам компьютерной математики.

К основным достоинствам использования Maxima можно отнести:

– возможность свободного использования (Maxima относится к классу свободных программ и распространяется по лицензии GNU GPL);

– возможность функционирования под управлением различных операционных систем (в частности, Linux и Windows);

– небольшой размер программы;

– широкий класс решаемых задач;

– возможность работы как в консольной версии программы, так и с использованием одного из графических интерфейсов.

Выбор Maxima для программной реализации обусловлен, во-первых, тем, что функциональные возможности Maxima аналогичны возможностям таких передовых систем компьютерной алгебры, как Mathematica [100] и Maple [101]. Во-вторых, Maxima является бесплатной и распространяется с открытым исходным кодом, что в свою очередь способствует ее широкому применению в научной деятельности.

4.2 Программное обеспечение для ранжирования и генерации по рангу элементов комбинаторных множеств

4.2.1 Структура программного обеспечения

На основе системы компьютерной алгебры «Maxima» было разработано программное обеспечение для ранжирования и генерации по рангу элементов комбинаторных множеств [51]. В приложении А представлено полученное свидетельство о государственной регистрации программы для ЭВМ.

На рисунке 4.1 представлена структура разработанного программного обеспечения для ранжирования и генерации по рангу элементов комбинаторных множеств.

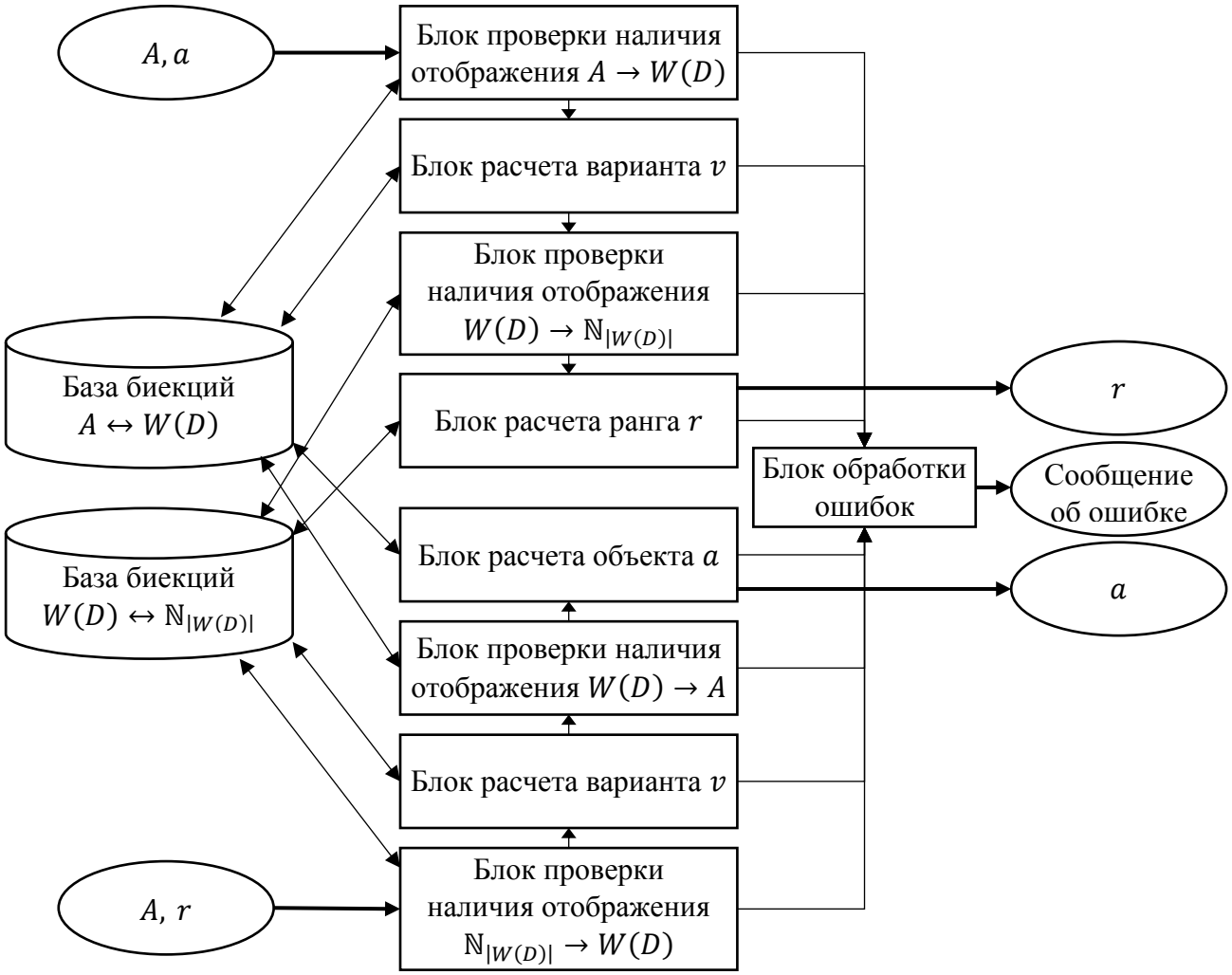


Рисунок 4.1 — Структура программного обеспечения для ранжирования и генерации по рангу элементов комбинаторных множеств

Структура разработанного программного обеспечения состоит из следующих элементов:

1. Блок ввода данных: пользователю необходимо предоставить информацию о рассматриваемом комбинаторном множестве A (название комбинаторного множества и значения описывающих его параметров), а также указать конкретный комбинаторный объект $a \in A$ или ранг комбинаторного объекта $r \in \{0, \ldots, |A| - 1\}$.

2. База биекций $A \leftrightarrow W(D)$: представляет собой набор алгоритмов $\texttt{ObjectToVariant}(a) : A \to W(D)$ и $\texttt{VariantToObject}(v) : W(D) \to A$ для некоторого перечня комбинаторных множеств. В программе реализованы все алгоритмы данного вида, которые представлены в главе 3 диссертационной работы.

3. База биекций $W(D) \leftrightarrow \mathbb{N}_{|W(D)|}$: представляет собой набор алгоритмов $\texttt{RankVariant}(v) : W(D) \to \mathbb{N}_{|W(D)|}$ и $\texttt{UnrankVariant}(r) : \mathbb{N}_{|W(D)|} \to W(D)$ для некоторого перечня комбинаторных множеств. В программе реализованы все алгоритмы данного вида, которые представлены в главе 3 диссертационной работы.

4. Проверочные блоки: на основе предоставленной от пользователя информации о комбинаторном множестве программа проверяет наличие в базах биекций требуемых для расчетов алгоритмов комбинаторной генерации. Если для заданного комбинаторного множества в базах биекций имеются соответствующие алгоритмы комбинаторной генерации, то программа переходит к расчетным блокам, иначе выполняется переход к блоку обработки ошибок.

5. Расчетные блоки: в зависимости от предоставленной пользователем информации программа с помощью имеющихся в базах биекций алгоритмов комбинаторной генерации производит требуемые расчеты. Если пользователем задан конкретный комбинаторный объект a, то сначала выполняется расчет варианта v с помощью соответствующего комбинаторному множеству алгоритма $\texttt{ObjectToVariant}(a)$, а затем по полученному варианту v выполняется расчет ранга r с помощью соответствующего комбинаторному множеству алгоритма $\texttt{RankVariant}(v)$. Если пользователем задан ранг r, то сначала выполняется расчет варианта v с помощью соответствующего комбинаторному множеству алгоритма $\texttt{UnrankVariant}(r)$, а затем по полученному варианту v выполняется расчет комбинаторного объекта a с помощью соответствующего комбинаторному множеству алгоритма $\texttt{VariantToObject}(v)$.

6. Блок обработки ошибок: выполняет обработку ошибок, которые могут возникнуть в ходе выполнения вычислений (как на основе внесенных в программу проверочных функций, так и с помощью встроенного в Maxima обработчика ошибок).

7. Блок вывода данных: выполняет вывод полученных в ходе выполнения вычислений результатов. В случае успешного выполнения всех вычислений пользователю отображается ранг r комбинаторного объекта или собственно комбинаторный объект a (в зависимости от поставленной пользователем задачи). При возникновении ошибок в ходе работы программы пользователю будет предоставлено соответствующее сообщение об ошибке.

Разработанное программное обеспечение представляет собой набор из подключаемых к Maxima файлов-библиотек:

– файл «CombGen.wxm» — реализует общий интерфейс взаимодействия с пользователем по работе с алгоритмами ранжирования и генерации по рангу на примере некоторого абстрактного комбинаторного множества (родительский класс для всех конкретных комбинаторных множеств);

– файл «CombGenLib.wxm» — реализует функцию справочника, который по заданному названию комбинаторного множества подключает к работе соответствующий файл;

– файлы вида «CombGen_AName.wxm», где AName является названием конкретного комбинаторного множества — содержит программную реализацию алгоритмов комбинаторной генерации для соответствующего комбинаторного множества;

Наличие общего интерфейса взаимодействия, а также возможность подключения дополнительных файлов с программной реализацией алгоритмов комбинаторной генерации позволяют в дальнейшем расширять перечень комбинаторных множеств, работу с которыми будет поддерживать разработанное программное обеспечение. Для этого необходимо реализовать алгоритмы комбинаторной генерации для некоторого комбинаторного множества в виде отдельного файла «CombGen_AName.wxm» и внести соответствующую информацию о функциях данного файла в файл «CombGenLib.wxm».

4.2.2 Пример взаимодействия с программным обеспечением

Все взаимодействие с разработанным программным обеспечением происходит в системе компьютерной алгебре «Maxima», поэтому первым шагом требуется запустить интерфейс работы с системой Maxima (например, графический пользовательский интерфейс «wxMaxima» [102]). Далее необходимо подключить разработанное программное обеспечение, указав путь к файлу «CombGen.wxm», введя команду

load(”CombGen.wxm”);

Чтобы приступить к работе с алгоритмами комбинаторной генерации, необходимо ввести описание комбинаторного множества в формате

ОбозначениеМножества : [”НазваниеМножества”, Параметр1 : ЗначениеПараметра1, Параметр2 : ЗначениеПараметра2, . . .];

Например, для рассмотренного в разделе 3.2 множества вторичных структур РНК длины n с m пар нуклеотидов, соединенных водородной связью, описание комбинаторного множества будет иметь следующий вид:

A:[”RNASecondaryStructure”, n:8, m:3];

Для того, чтобы узнать значение функции мощности комбинаторного множества A, следует ввести команду

fObject(A);

Рассмотрим работу алгоритма генерации по рангу объектов комбинаторного множества. Для этого необходимо указать значение ранга r и для заданного описания комбинаторного множества A ввести команду

a:Unrank(A,r);

В результате отобразится запись представления в программе полученного объекта a комбинаторного множества A. Например, для множества вторичных структур РНК длины n с m пар нуклеотидов, соединенных водородной связью, отобразится последовательность правильно вложенных m пар скобок с возможностью пропуска скобки, в которой пропуск скобки соответствует значению 0, открывающаяся скобка — значению 1, закрывающаяся скобка — значению -1.

Обратное действие по ранжированию объекта a комбинаторного множества A выполняется с помощью команды

r:Rank(A,a);

На рисунке 4.2 представлен пример выполнения вышеописанных команд по работе с разработанным программным обеспечением для ранжирования и генерации по рангу элементов комбинаторных множеств.

```
(%i1)   load("CombGen.wxm");
(%o1)   CombGen.wxm

(%i2)   A:["RNASecondaryStructure",n:8,m:3];
(A)     [RNASecondaryStructure,8,3]

(%i3)   fObject(A);
(%o3)   10

(%i5)   r:5;
        a:Unrank(A,r);
(r)     5
(a)     [1,1,1,0,-1,-1,0,-1]

(%i6)   r:Rank(A,a);
(r)     5
```

Рисунок 4.2 — Пример взаимодействия с разработанным программным обеспечение для ранжирования и генерации по рангу элементов комбинаторных множеств

```
(%i7)   for r:0 thru fObject(A)-1 do (print(r,a:Unrank(A,r),Rank(A,a)));
0 [0,1,1,1,0,-1,-1,-1] 0
1 [1,0,1,1,0,-1,-1,-1] 1
2 [1,1,0,1,0,-1,-1,-1] 2
3 [1,1,1,0,0,-1,-1,-1] 3
4 [1,1,1,0,-1,0,-1,-1] 4
5 [1,1,1,0,-1,-1,0,-1] 5
6 [1,1,0,-1,1,0,-1,-1] 6
7 [1,1,1,0,-1,-1,-1,0] 7
8 [1,1,0,-1,-1,1,0,-1] 8
9 [1,0,-1,1,1,0,-1,-1] 9
(%o7)   done
```

Рисунок 4.3 — Пример генерации всех объектов комбинаторного множества

Для генерации всех объектов комбинаторного множества A или, например, для проверки правильности работы алгоритмов комбинаторной генерации можно с помощью цикла for выполнить генерацию комбинаторных объектов со значением ранга $r = 0, \ldots, |A| - 1$. На рисунке 4.3 представлен пример полученных результатов после выполнения указанных действий для множества вторичных структур РНК длины $n = 8$ с $m = 3$ пар нуклеотидов, соединенных водородной связью.

На рисунке 4.4 показаны несколько примеров отображаемых пользователю сообщений о возникших в ходе выполнения вычислений ошибках.

```
(%i10)      A:["RNAPrimaryStructure",n:8,m:3]$
            r:10$
            a:Unrank(A,r);
(A)         done
В базе отсутствует биекция N_|W(D)|< - >W(D) для комбинаторного множества
RNAPrimaryStructure !

(%i13)      A:["RNASecondaryStructure",n:8,m:3]$
            r:10$
            a:Unrank(A,r);
(A)         false
Введено неправильное значение ранга! Ранг должен быть в интервале от 0 до  9 .

(%i16)      A:["RNASecondaryStructure",n:8,m:3]$
            a:[1,0,1,1,2,-1,-1,-1]$
            Rank(A,a);
(A)         false
В [1,0,1,1,2,-1,-1,-1] имеются элементы, которые не равны 1, -1 или 0!
(%o16)      false

(%i19)      A:["RNASecondaryStructure",n:8,m:3]$
            a:[1,0,1,1,-1,-1,-1]$
            Rank(A,a);
Количество элементов в [1,0,1,1,-1,-1,-1] не равно 8 !
(%o19)      false

(%i22)      A:["RNASecondaryStructure",n:8,m:3]$
            a:[1,0,1,-1,0,0,-1,0]$
            Rank(A,a);
Количество элементов в [1,1,-1,-1] не равно 6 !
(%o22)      false
```

Рисунок 4.4 — Пример отображаемых сообщений об ошибках

4.2.3 Проверка достоверности разработанных алгоритмов комбинаторной генерации

Используя разработанное программное обеспечение для ранжирования и генерации по рангу элементов комбинаторных множеств, было проведено исследование достоверности разработанных в главе 3 алгоритмов комбинаторной генерации. Для этого был составлен код программы, представленный на рисунке 4.5, который выполняет генерацию всех объектов заданного комбинаторного множества с последующей проверкой достоверности полученных результатов.

В данном случае проверка достоверности разработанных алгоритмов комбинаторной генерации заключается в следующем:

1. Программа получает на вход описание комбинаторного множества A;

2. Создаются два пустых массива:

– MasR1 для хранения информации о заданном ранге r некоторого комбинаторного объекта a;

– MasR2 для хранения информации о полученном в ходе выполнения функции Rank(A,a) ранге r по информации о комбинаторном объекте a, полученном в ходе выполнения функции Unrank(A,r) по заданному рангу r;

3. В цикле для всех значений ранга $r \in \{0, \ldots, |A| - 1\}$ последовательно выполняются функции Unrank(A,r) и Rank(A,a), при этом происходит заполнение массивов MasR1 и MasR2 соответствующими значениями;

4. На основе полученных результатов выполняется проверка условий:

– если в ходе вычислений блок обработки ошибок не отобразил каких-либо сообщений, следовательно все сгенерированные с помощью функции Unrank(A,r) комбинаторные объекты соответствуют описанию комбинаторного множества A и их количество совпадает с значением функции мощности $|A|$;

– если массив MasR1 полностью совпадает с массивом MasR2, следовательно функции Unrank(A,r) и Rank(A,a) выполняют обратные друг для друга преобразования и для каждого комбинаторного объекта a однозначно соответствует некоторое значение ранга $r \in \{0, \ldots, |A| - 1\}$;

5. Если проверка условий выполнена успешно, следовательно для заданного описания комбинаторного множества A алгоритмы Unrank(A,r) и Rank(A,a) реализуют биективное отображение между комбинаторным множеством A и множеством рангов комбинаторных объектов $\{0, \ldots, |A| - 1\}$, то есть полученные в ходе выполнения алгоритмов результаты являются достоверными.

```
MasR1:[]$
MasR2:[]$
for r:0 thru fObject(A)-1 do
(
   MasR1:endcons(r,MasR1),
   a:Unrank(A,r),
   rr:Rank(A,a),
   MasR2:endcons(rr,MasR2)
);
if (MasR1=MasR2) then true else false;
```

Рисунок 4.5 — Код проверки достоверности полученных результатов

Представленный код проверки достоверности полученных результатов был применен к разработанным в главе 3 алгоритмам комбинаторной генерации, а именно:

– для всех возможных описаний множества последовательностей правильно вложенных n пар скобок m типов при $0 \leqslant n \leqslant 6$, $0 \leqslant m \leqslant 6$ (общее количество проверенных комбинаторных объектов равно $\sum\limits_{n=0}^{6} \sum\limits_{m=0}^{6} C_n^m = 9413279$);

– для всех возможных описаний множества комбинаторных объектов, отражающих вторичную структуру РНК длины n с m пар нуклеотидов, соединенных водородной связью, при $0 \leqslant n \leqslant 20$, $0 \leqslant m < n/2$ (общее количество проверенных комбинаторных объектов равно $\sum\limits_{n=0}^{20} \sum\limits_{m=0}^{n/2} S_{n,m} = 4273935$);

– для всех возможных описаний множества помеченных путей Дика длины $2n$ с m подъемами на возвратных шагах при $0 \leqslant m < n \leqslant 7$ (общее количество проверенных комбинаторных объектов равно $\sum\limits_{n=0}^{7} \sum\limits_{m=0}^{n} EC_{n,m} = 2262611$).

Для всех рассмотренных описаний комбинаторных множеств итог применения кода проверки достоверности полученных результатов является положительным, то есть разработанные для данных комбинаторных множеств алгоритмы Unrank(A,r) и Rank(A,a) реализуют биективное отображение между комбинаторным множеством и множеством рангов комбинаторных объектов. Суммарно корректность работы разработанных алгоритмов комбинаторной генерации была апробирована на более чем $1{,}5 \cdot 10^7$ комбинаторных объектов. Данный вычислительный эксперимент подтверждает достоверность разработанных алгоритмов комбинаторной генерации.

4.2.4 Результаты вычислительных экспериментов

Используя программную реализацию разработанных в главе 3 алгоритмов комбинаторной генерации, был проведен ряд вычислительных экспериментов, направленных на выявление зависимостей времени вычислений от значений параметров исследуемого комбинаторного множества.

Вычислительные эксперименты проводились на компьютере со следующими характеристиками:

– Процессор: Intel(R) Core(TM) i7-3610QM, 2.3 ГГц;

– Оперативная память: 4 ГБ;

– Операционная система: 32-разрядная Windows 7 Профессиональная;

– Maxima: Maxima версия 5.43.0, wxMaxima версия 19.05.7.

Для оценки времени вычислений использовалась встроенная команда «showtime:true» программы Maxima, которая для каждой выполненной в Maxima команды отображает затраченное процессором время на вычисления.

В разделе 3.2 представлены следующие оценки вычислительной сложности алгоритмов комбинаторной генерации для множества вторичных структур РНК длины n с m пар нуклеотидов, соединенных водородной связью:

– оценка вычислительной сложности $O(m^2(n-m))$ для алгоритма `UnrankVariant_RNASS`(r,n,m) (Алгоритм 24);

– оценка вычислительной сложности $O(m^2(n-m))$ для алгоритма `RankVariant_RNASS`(v,n,m) (Алгоритм 23).

В качестве проверки зависимости алгоритмов `UnrankVariant_RNASS`(r,n,m) и `RankVariant_RNASS`(v,n,m) от значений параметра n был проведен вычислительный эксперимент, в рамках которого был зафиксирован второй параметр $m=2$ и была рассмотрена работа алгоритмов для комбинаторных объектов с максимальным значением ранга $r=S_n^m-1$. В данном случае время вычислений будет максимальным, так как происходит обход всего дерева И/ИЛИ, что соответствует максимальному значению количества итераций суммирования при расчете выражения (3.2). Значение параметра n варьировалось от 10 до 500 с шагом 10, все вычисления повторялись 100 раз.

На рисунке 4.6 представлены полученные результаты средних значений времени вычислений по каждому алгоритму (расчет доверительных интервалов производился с надежностью 0,99).

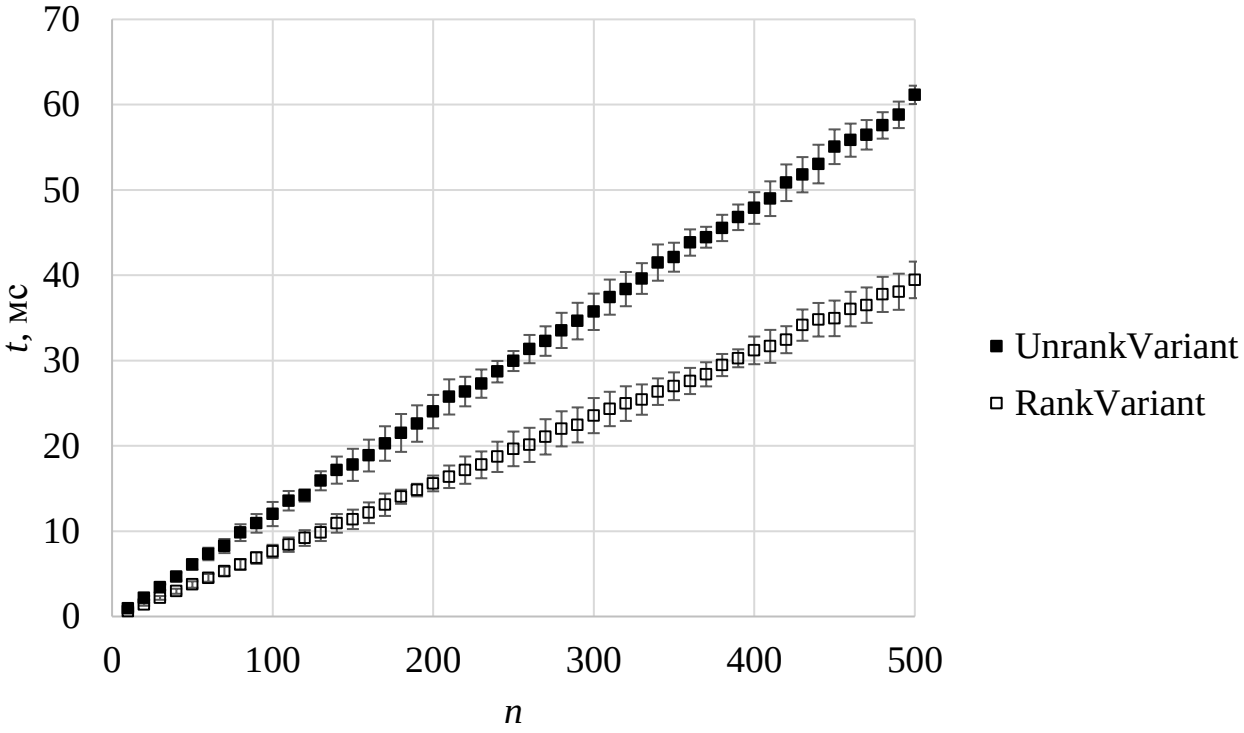


Рисунок 4.6 — Среднее время вычислений в зависимости от параметра n (на основе максимального значения ранга r при $m = 2$)

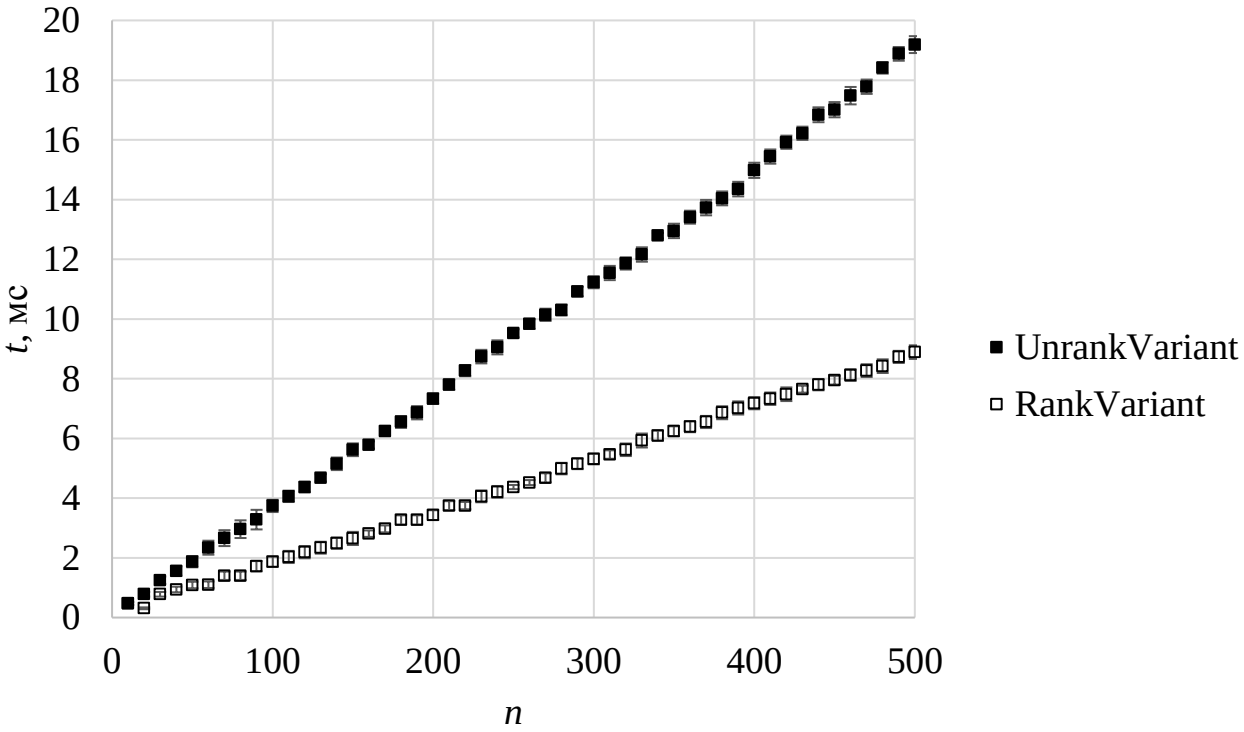


Рисунок 4.7 — Среднее время вычислений в зависимости от параметра n (на основе минимального значения ранга r при $m = 2$)

Также был проведен вычислительный эксперимент для фиксированного значения $m = 2$, в рамках которого была рассмотрена работа алгоритмов `UnrankVariant_RNASS`(r, n, m) и `RankVariant_RNASS`(v, n, m) для комбинаторных объектов с минимальным значением ранга $r = 0$. В данном случае время вычислений будет минимальным, так как количество итераций суммирования при расчете выражения (3.2) становится нулевым. Значение параметра n варьировалось от 10 до 500 с шагом 10, все вычисления повторялись 100 раз.

На рисунке 4.7 представлены полученные результаты средних значений времени вычислений по каждому алгоритму (расчет доверительных интервалов производился с надежностью 0,99). Результаты проведенных вычислительных экспериментов показывают наличие линейной зависимости от параметра n и подтверждают полученные теоретические оценки вычислительной сложности.

Однако такая линейная зависимость от параметра n наблюдается только при условии $m < n - 2m$. Это связано с тем, что для расчета S_n^m в выражении (3.3) при $m > n - 2m$ с целью оптимизации времени вычислений меняется схема расчета биномиального коэффициента $\binom{n-m}{m}$ и вычислительная сложность для расчета S_n^m становится равной $O(n - 2m)$. Тогда итоговая вычислительная сложность алгоритмов `UnrankVariant_RNASS`(r, n, m) и `RankVariant_RNASS`(v, n, m) при $m > n - 2m$ становится равной $O(m(n - m)^2)$, то есть должна наблюдаться квадратичная зависимость времени вычислений от значений параметра n. При этом итоговая оценка $O(m^2(n - m))$ не противоречит данной особенности работы программы, так как является асимптотической оценкой сверху.

Для исследования данной особенности был проведен вычислительный эксперимент для фиксированного значения $m = 30$. При таком значении параметра m требуемое условие $m > n - 2m$ будет выполняться при $n < 3m$, то есть при $n < 90$. Также для рассматриваемого комбинаторного множества должно выполняться условие $n > 2m$, то есть $n > 60$. Таким образом, значение параметра n варьировалось от 61 до 90 с шагом 1, все вычисления повторялись 100 раз для комбинаторных объектов с максимальным значением ранга $r = S_n^m - 1$.

На рисунке 4.8 представлены полученные результаты средних значений времени вычислений по каждому алгоритму (расчет доверительных интервалов производился с надежностью 0,99), которые подтверждают наличие квадратичной зависимости от параметра n при $m > n - 2m$.

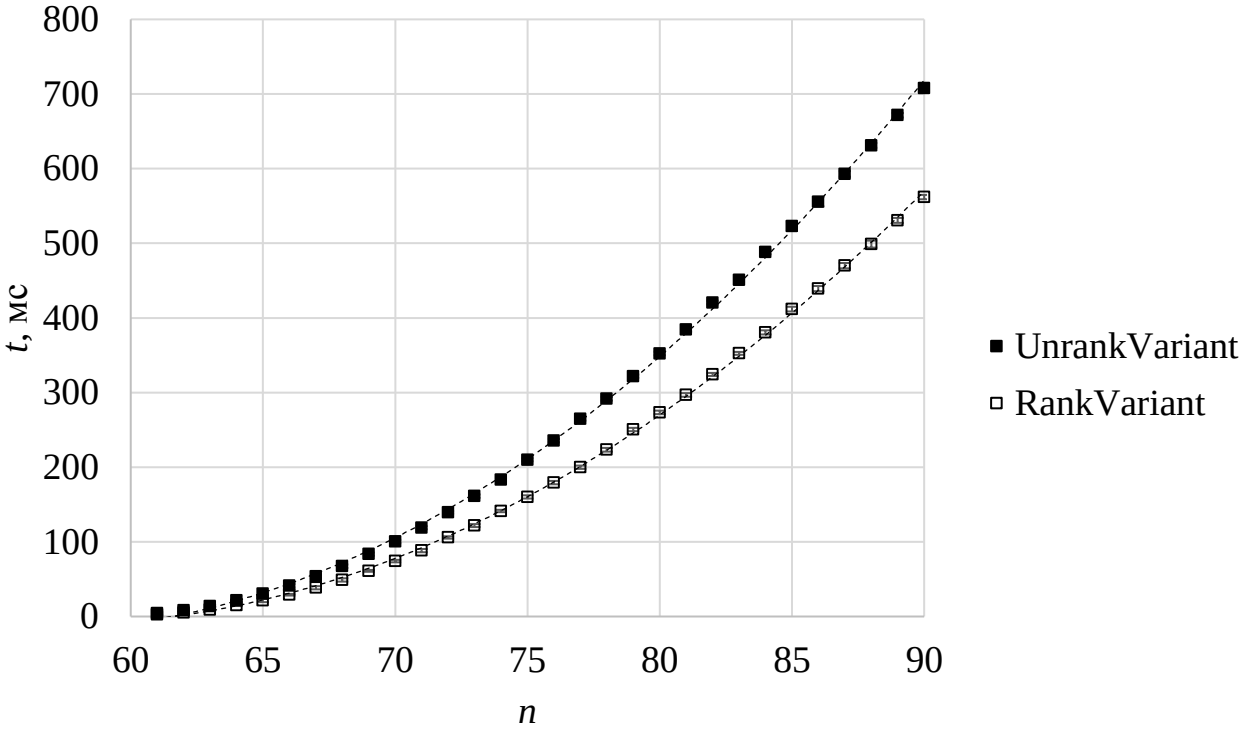


Рисунок 4.8 — Среднее время вычислений в зависимости от параметра n (на основе максимального значения ранга r при $m = 30$, $m > n - 2m$)

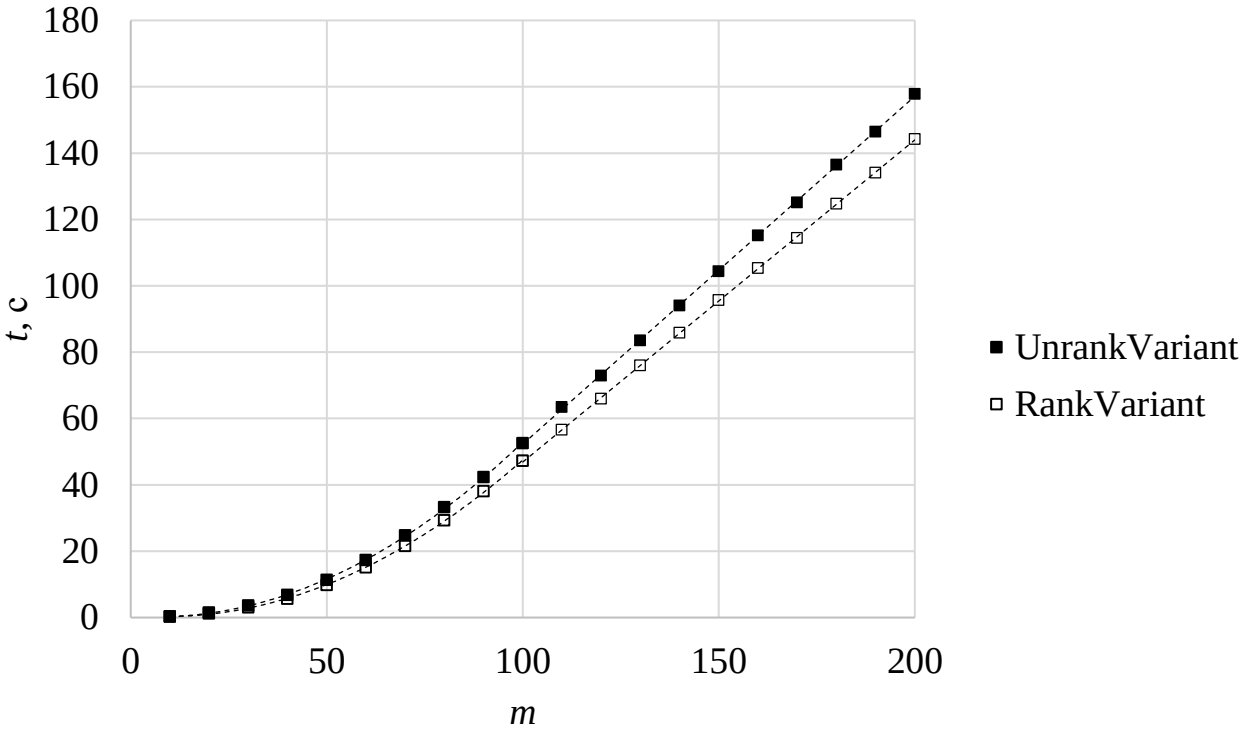


Рисунок 4.9 — Время вычислений в зависимости от параметра m (на основе максимального значения ранга r при $n = 100 + 2m$)

Для проверки зависимости алгоритмов `UnrankVariant_RNASS`(r, n, m) и `RankVariant_RNASS`(v, n, m) от значений параметра m был проведен вычислительный эксперимент, в рамках которого был зафиксирован второй параметр $n = 100 + 2m$. При таком значении параметра n разница $n - 2m$, которая присутствует в оценке вычислительной сложности данных алгоритмов, становится постоянной. Значение параметра m варьировалось от 1 до 200 с шагом 10, все вычисления повторялись 1 раз для комбинаторных объектов с максимальным значением ранга $r = S_n^m - 1$.

На рисунке 4.9 представлены полученные результаты значений времени вычислений по каждому алгоритму, которые подтверждают наличие квадратичной зависимости от параметра m (при $m < n - 2m$ и постоянном значении разницы $n - 2m$) и линейной зависимости от параметра m (при $m > n - 2m$ и постоянном значении разницы $n - 2m$).

Аналогично были проведены вычислительные эксперименты, направленные на выявление зависимостей времени вычислений от значений параметров, для комбинаторного множества помеченных путей Дика длины $2n$ с m подъемами на возвратных шагах. В разделе 3.3 представлены следующие оценки вычислительной сложности алгоритмов комбинаторной генерации для данного множества:

– оценка вычислительной сложности $O(nm(n - m))$ для алгоритма `UnrankVariant_DyckPathARS`(r, n, m) (Алгоритм 30);

– оценка вычислительной сложности $O(nm(n - m))$ для алгоритма `RankVariant_DyckPathARS`(v, n, m) (Алгоритм 29).

В качестве проверки зависимости указанных алгоритмов от значений параметра n был проведен вычислительный эксперимент, в рамках которого был зафиксирован второй параметр $m = 2$. Значение параметра n варьировалось от 10 до 200 с шагом 10, все вычисления повторялись 100 раз для комбинаторных объектов с максимальным значением ранга $r = EC_n^m - 1$.

В качестве проверки зависимости указанных алгоритмов от значений параметра m был проведен вычислительный эксперимент, в рамках которого был зафиксирован второй параметр $n = 50 + m$. При таком значении параметра n выражение $O(nm(n - m))$ становится равным $O(m^2)$. Значение параметра m варьировалось от 5 до 100 с шагом 5, все вычисления повторялись 100 раз для комбинаторных объектов с максимальным значением ранга $r = EC_n^m - 1$.

На рисунках 4.10 и 4.11 представлены полученные результаты средних значений времени вычислений по каждому алгоритму, которые подтверждают наличие квадратичной зависимости от параметра n и квадратичной зависимости от параметра m (при постоянном значении разницы $n - m$).

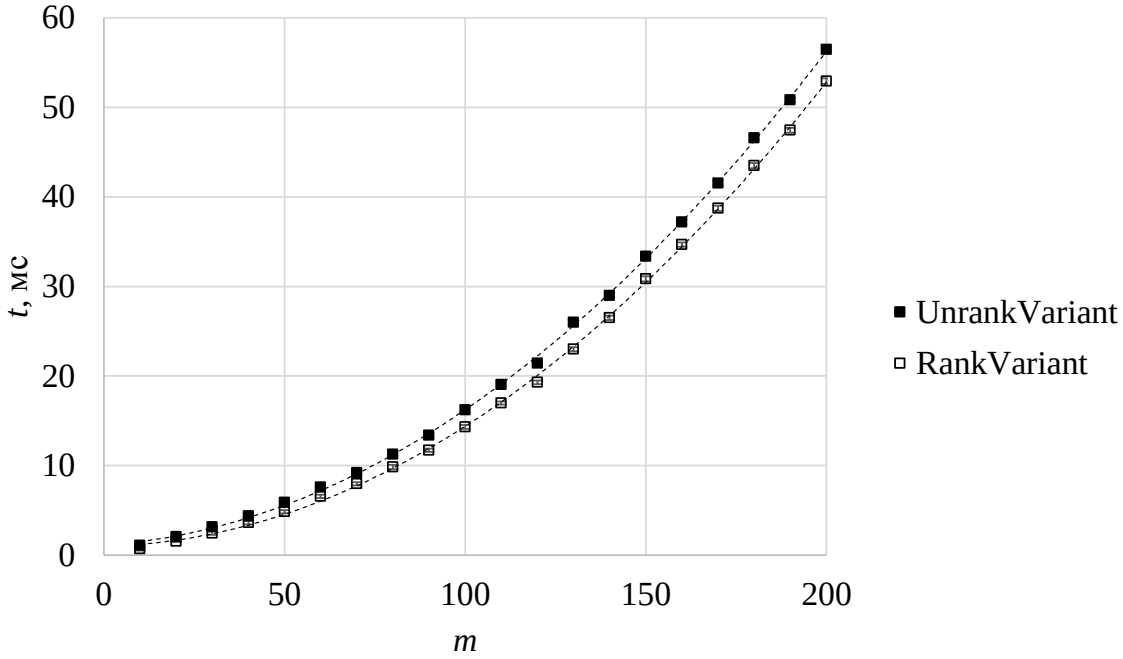


Рисунок 4.10 — Среднее время вычислений в зависимости от параметра n (на основе максимального значения ранга r при $m = 2$)

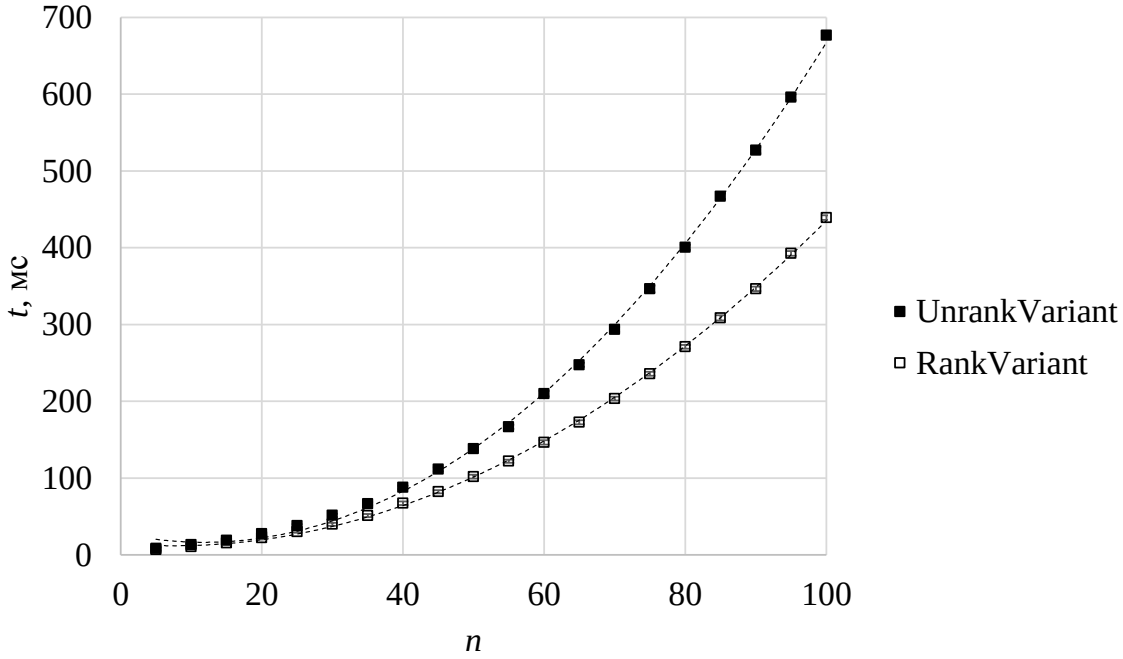


Рисунок 4.11 — Среднее время вычислений в зависимости от параметра n (на основе максимального значения ранга r при $n = 50 + m$)

4.3 Внедрение результатов диссертационной работы

Достоверность полученных результатов диссертационного исследования подтверждается их внедрением в деятельность ООО «ПлантаПлюс» и ООО «Удостоверяющий центр Сибири», которое оказало положительный эффект.

Результаты диссертационной работы были внедрены в деятельность ООО «ПлантаПлюс», которая занимается химическим и биологическим исследованием и производством микробиологических препаратов для растениеводства, в процессе работы над систематизацией архива проведения экспериментальных исследований (Приложение Б).

На основе применения описанного в диссертационной работе математического аппарата, для архива проведения экспериментальных исследований был применен подход его представления в виде комбинаторного множества, для которого впоследствии была получена соответствующая ему структура дерева И/ИЛИ. Для этого каждый набор параметров, фиксируемых в рамках проведения эксперимента, был представлен в виде И-узла, при этом для каждого параметра фиксировалось конечное множество его возможных значений, которые были представлены в виде сыновей ИЛИ-узла. Это позволило объединить разрозненные базы данных и систематизировать имеющийся архив проведения экспериментальных исследований, что, в свою очередь, способствует повышению эффективности работы с архивом за счет сокращения времени на поиск результатов требуемого эксперимента.

Применение предложенного модифицированного метода построения алгоритмов ранжирования и генерации по рангу для полученного представления в виде структуры дерева И/ИЛИ позволило выполнить кодирование хранящихся результатов экспериментов. Положительным эффектом от кодирования является сокращение на 17% общего объема базы данных архива проведения экспериментальных исследований.

Результаты диссертационной работы были внедрены в деятельность ООО «Удостоверяющий центр Сибири», которая занимается организацией защищенного электронного документооборота и предоставляет услуги по защите информации, в процессе создания программного продукта для работы с алгоритмами получения простых чисел (Приложение Б).

Целочисленные свойства композиции обыкновенных и экспоненциальных производящих функций, которые были получены в рамках исследований, направленных на изучение и апробацию математического аппарата степеней производящих функций для модификации метода комбинаторной генерации, были применены в работе алгоритмов генерации простых чисел. Используемый в алгоритмах генерации простых чисел тест проверки простоты числа был дополнен новыми критериями простоты числа. Данные критерии простоты числа были получены с помощью разработанного программного обеспечения для генерации новых критериев простоты числа.

Комбинирование полученных результатов проверки генерируемых чисел с помощью теста простоты числа с результатами проверки теста, основанного на новых критериях простоты числа, позволило сократить от 5 до 7% количество необнаруженных псевдопростых чисел, что является значимым результатом при обработке больших выборок чисел.

Также результаты диссертационной работы используются в учебном процессе на факультете безопасности ТУСУР при чтении курса лекций и проведении практических занятий по дисциплинам «Дискретная математика» и «Теория игр и исследование операций» для подготовки специалистов по защите информации (Приложение Б).

В курсе «Дискретная математика» используются результаты по разработке представлений элементов комбинаторных множеств в различных формах записи, в том числе и в форме структуры дерева И/ИЛИ, а также по разработке алгоритмов отображения элементов комбинаторных множеств в варианты дерева И/ИЛИ. Это позволяет студентам ознакомиться с процессом реализации биективного отображения между двумя множествами.

В курсе «Теория игр и исследование операций» используется предложенный модифицированный метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ, позволяющий студентам ознакомиться с процессом разработки алгоритмов кодирования объектов комбинаторных множеств и применить их на практике.

Освоение студентами предложенного алгоритмического обеспечения позволяет сформировать навыки работы с дискретными структурами, их перечисления и кодирования.

Кроме того, разработанные алгоритмы ранжирования и генерации по рангу для множества комбинаторных объектов, отражающих вторичную структуру РНК, могут быть использованы для сжатия баз данных, хранящих информацию о таких объектах.

Например, рассмотрим случай хранения информации о вторичной структуре РНК длины n с m пар нуклеотидов, соединенных водородной связью, в виде последовательности $a = (a_1, \ldots, a_n)$ правильно вложенных m пар скобок с возможностью пропуска скобки. Предположим, что $n = 100$ (реальная длина цепочки молекулы РНК может варьироваться от 80 нуклеотидов до нескольких десятков тысяч нуклеотидов). Если хранить информацию в байтовом представлении десятичных цифр, тогда чтобы хранить тройку (n, m, a) потребуется максимум 105 байт:

$$\lceil \log_{10}(n+1) \rceil + \lceil \log_{10} m_{max} \rceil + |a| = 3 + 2 + 100 = 105.$$

Если же для последовательности a выполнить алгоритм ранжирования, тогда будет достаточно хранить тройку (n, m, r), которая потребует максимум 44 байта (при $m = 27$ и $S_n^m = S_{100}^{27} = 1252591483604977377949623319717323658 24$):

$$\lceil \log_{10}(n+1) \rceil + \lceil \log_{10} m \rceil + \lceil \log_{10} S_n^m \rceil = 3 + 2 + 39 = 44.$$

Таким образом, достигается сжатие хранящейся информации в более чем 2,3 раза. При $n = 100$ и $m \neq 27$ коэффициент сжатия становится еще больше. Например, для $n = 100$ при $m = 1$ и $S_n^m = S_{100}^1 = 4851$ достигается максимальный коэффициент сжатия, равный 13:

$$\lceil \log_{10}(n+1) \rceil + \lceil \log_{10} m \rceil + \lceil \log_{10} S_n^m \rceil = 3 + 1 + 4 = 8.$$

4.4 Выводы по главе

Основным результатом данной главы является разработанное программное обеспечение для ранжирования и генерации по рангу элементов комбинаторных множеств, которая автоматизирует процессы вычислений в рамках разработанных в главе 3 алгоритмов комбинаторной генерации. Наличие общего интерфейса взаимодействия, а также возможность подключения дополнительных файлов с программной реализацией алгоритмов комбинаторной генерации позволяют в дальнейшем расширять перечень комбинаторных множеств, работу с которыми будет поддерживать разработанное программное обеспечение.

С использованием разработанной программы был проведен ряд вычислительных экспериментов, направленных на выявление зависимостей времени вычислений от значений параметров исследуемых комбинаторных множеств. Результаты проведенных вычислительных экспериментов полностью подтверждают полученные теоретические оценки вычислительной сложности для разработанных в разделе 3.2 алгоритмов комбинаторной генерации для множества вторичных структур РНК длины n с m пар нуклеотидов, соединенных водородной связью, и для разработанных в разделе 3.3 алгоритмов комбинаторной генерации для множества помеченных путей Дика длины $2n$ с m подъемами на возвратных шагах.

Заключение

В диссертационной работе решена задача развития методов построения алгоритмов комбинаторной генерации на основе применения теории производящих функций за счет использования метода получения явных выражений коэффициентов производящих функций.

Основные результаты диссертационной работы:

1. Проведен аналитический обзор современного состояния исследований в области разработки алгоритмов комбинаторной генерации, который показал, что в настоящее время существует несколько универсальных методов построения алгоритмов комбинаторной генерации: метод поиска с возвратом, ECO-метод, метод Ф. Флажоле, метод Б.Я. Рябко, метод В.В. Кручинина. При этом: метод поиска с возвратом и ECO-метод направлены только на разработку алгоритмов последовательной генерации комбинаторных объектов; существуют ограничения на возможность применения ECO-метода и метода Ф. Флажоле для комбинаторных множеств, описываемых более чем одним параметром; большинство методов требует представления комбинаторного объекта в специальном виде; существуют требования к наличию дополнительной информации, описывающей комбинаторное множество.

2. Проведено исследование метода построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ и метода получения явных выражений коэффициентов производящих функций. Используя данные методы предложен модифицированный метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ. Предложенный модифицированный метод отличается применением метода получения явных выражений коэффициентов производящих функций для нахождения выражения функции мощности комбинаторного множества и позволяет построить алгоритмы ранжирования и генерации по рангу для таких комбинаторных множеств, для которых не известно требуемое выражение функции мощности, но известно выражение производящей функции для последовательности значений функции мощности.

3. Проведена апробация модифицированного метода построения алгоритмов комбинаторной генерации, в ходе которой разработаны новые алгоритмы комбинаторной генерации:

– для множества выражений обобщенного языка Дика на примере множества последовательностей правильно вложенных n пар скобок m типов (оценка вычислительной сложности полученных алгоритмов равна $O(n^3)$, что является хуже по сравнению с существующими аналогами, однако это показывает возможность применения разработанного модифицированного метода для получения новых алгоритмов комбинаторной генерации);

– для множества комбинаторных объектов, отражающих вторичную структуру РНК длины n с m пар нуклеотидов, соединенных водородной связью (оценка вычислительной сложности равна $O(m^2(n-m))$, что по сравнению с существующими аналогами является лучшей, так как не требуется никаких предварительных вычислений);

– для множества комбинаторных объектов, определяемых треугольником Эйлера-Каталана, на примере множества помеченных путей Дика длины $2n$ с m подъемами на возвратных шагах (оценка вычислительной сложности разработанных алгоритмов равна $O(m^2(n-m))$).

4. На основе разработанных алгоритмов комбинаторной генерации создано программное обеспечение для ранжирования и генерации по рангу элементов комбинаторных множеств. Наличие общего интерфейса взаимодействия, а также возможность подключения дополнительных файлов с программной реализацией алгоритмов комбинаторной генерации позволяют в дальнейшем расширять перечень комбинаторных множеств, работу с которыми будет поддерживать разработанное программное обеспечение. Результаты проведенных вычислительных экспериментов полностью подтверждают полученные теоретические оценки вычислительной сложности для разработанных алгоритмов комбинаторной генерации.

Полученные результаты диссертационной работы использованы в ходе выполнения научно-исследовательских работ в рамках базовой части государственного задания Министерства науки и высшего образования РФ (проект № 2.8172.2017/8.9), гранта «Российского научного фонда» (проект № 18-71-00059) и гранта «Российского фонда фундаментальных исследований» (проект № 18-31-00201). Также результаты диссертационной работы использованы при систематизации архива проведения экспериментальных исследований в ООО «ПлантаПлюс», применены в процессе создания программного продукта для работы с алгоритмами получения простых чисел в ООО «Удостоверяющий центр Сибири», и внедрены в учебный процесс ФГБОУ ВО «ТУСУР».

Список литературы

1. *Кнут Д. Э.* Искусство программирования. Том 4, A. Комбинаторные алгоритмы, часть 1. — М.: Вильямс, 2013. — 960 с.

2. *Ruskey F.* Combinatorial generation. Working version (1j-CSC 425/520). — 2003. — 311 p. — URL: http://page.math.tu-berlin.de/~felsner/SemWS17-18/Ruskey-Comb-Gen.pdf.

3. *Arndt J.* Matters computational: Ideas, algorithms, source code. — Germany: Springer, 2011. — 966 p.

4. *Рейнгольд Э., Нивергельт Ю., Део Н.* Комбинаторные алгоритмы: Теория и практика. — М.: Мир, 1980. — 433 с.

5. *Kreher D. L., Stinson D. R.* Combinatorial algorithms: Generation, enumeration, and search. — USA: CRC Press, 1999. — 329 p.

6. ECO: A methodology for the enumeration of combinatorial objects / E. Barcucci, A. Del Lungo, E. Pergola, R. Pinzani // *Journal of Difference Equations and Applications*. — 1999. — Vol. 5, no. 4–5. — P. 435–490.

7. *Barcucci E., Del Lungo A., Pergola E.* Random generation of trees and other combinatorial objects // *Theoretical Computer Science*. — 1999. — Vol. 218, no. 2. — P. 219–232.

8. Exhaustive generation of combinatorial objects by ECO / S. Bacchelli, E. Barcucci, E. Grazzini, E. Pergola // *Acta Informatica*. — 2004. — Vol. 40, no. 8. — P. 585–602.

9. Mixed succession rules: The commutative case / S. Bacchelli, L. Ferrari, R. Pinzani, R. Sprugnoli // *Journal of Combinatorial Theory, Series A*. — 2010. — Vol. 117, no. 5. — P. 568–582.

10. *Del Lungo A., Frosini A., Rinaldi S.* ECO method and the exhaustive generation of convex polyominoes // *Lecture Notes in Computer Science: Discrete Mathematics and Theoretical Computer Science*. — 2003. — Vol. 2731. — P. 129–140.

11. On the generation and enumeration of some classes of convex polyominoes / A. Del Lungo, E. Duchi, A. Frosini, S. Rinaldi // *Electronic Journal of Combinatorics*. — 2004. — Vol. 11, no. 1 R. — P. 1–46.

12. *Vajnovszki V.* Generating involutions, derangements, and relatives by ECO // *Discrete Mathematics and Theoretical Computer Science*. — 2010. — Vol. 12, no. 1. — P. 109–122.

13. *Vajnovszki V.* An efficient Gray code algorithm for generating all permutations with a given major index // *Journal of Discrete Algorithms*. — 2014. — Vol. 26. — P. 77–88.

14. *Do P. T., Tran T. T. H., Vajnovszki V.* Exhaustive generation for permutations avoiding (colored) regular sets of patterns // *Discrete Applied Mathematics*. — 2019. — P. 1–10.

15. *Flajolet P., Zimmerman P., Cutsem B.* A calculus for the random generation of combinatorial structures // *Theoretical Computer Science*. — 1994. — Vol. 132, no. 1–2. — P. 1–35.

16. *Sedgewick R., Flajolet P.* An introduction to the analysis of algorithms. — Second edition. — USA: Addison-Wesley, 2013. — 593 p.

17. *Martinez C., Molinero X.* A generic approach for the unranking of labeled combinatorial classes // *Random Structures and Algorithms*. — 2001. — Vol. 19, no. 3–4. — P. 472–497.

18. *Martinez C., Molinero X.* Generic algorithms for the generation of combinatorial objects // *Lecture Notes in Computer Science: Mathematical Foundations of Computer Science*. — 2003. — Vol. 2747. — P. 572–581.

19. *Martinez C., Molinero X.* An experimental study of unranking algorithms // *Lecture Notes in Computer Science: Experimental and Efficient Algorithms*. — 2004. — Vol. 3059. — P. 326–340.

20. *Martinez C., Molinero X.* Efficient iteration in admissible combinatorial classes // *Theoretical Computer Science*. — 2005. — Vol. 346, no. 2–3. — P. 388–417.

21. *Рябко Б. Я.* Быстрая нумерация комбинаторных объектов // *Дискретная математика*. — 1998. — Т. 10, № 2. — С. 101–119.

22. *Медведева Ю. С., Рябко Б. Я.* Быстрый алгоритм нумерации слов с заданными ограниче-ниями на длины серий единиц // *Проблемы передачи информации.* — 2010. — Т. 46, № 4. — С. 130–139.

23. *Medvedeva Y.* Fast enumeration of words generated by Dyck grammars // *Mathematical Notes.* — 2014. — Vol. 96, no. 1–2. — P. 68–83.

24. *Медведева Юлия Сергеевна.* Быстрая нумерация комбинаторных объектов, находящая применение в системах передачи и хранения информации : дис. ... канд. техн. наук : 05.13.17. — Новосибирск, 2015. — 113 с.

25. *Кручинин В. В.* Методы построения алгоритмов генерации и нумерации комбинаторных объектов на основе деревьев И/ИЛИ. — Томск: В-Спектр, 2007. — 200 с.

26. *Кручинин Владимир Викторович.* Методы, алгоритмы и программное обеспечение комбинаторной генерации : дис. ... д-ра техн. наук : 05.13.11. — Томск, 2010. — 387 с.

27. *Srivastava H. M., Manocha H. L.* A treatise on generating functions. — USA: Ellis Horwood Limited, 1984. — 569 p.

28. *Wilf H. S.* Generatingfunctionology. — Second edition. — USA: Academic Press, 1994. — 228 p.

29. *Ландо С. К.* Лекции о производящих функциях. — 3 изд. — М.: Издательство МЦНМО, 2007. — 144 с.

30. *Кручинин Д. В., Шабля Ю. В.* Программное обеспечение для анализа тестов простоты натурального числа // *Доклады ТУСУРа.* — 2014. — № 4 (34). — С. 95–99.

31. *Шабля Ю. В., Кручинин Д. В., Шелупанов А. А.* Генератор критериев простоты натурального числа на основе свойств композиции производящих функций // *Доклады ТУСУРа.* — 2015. — № 4 (38). — С. 97–101.

32. Сравнительный анализ вычислительных способов нахождения коэффициентов ряда Тейлора в математических пакетах / В. С. Мельман, Ю. В. Шабля, Д. В. Кручинин, В. В. Кручинин // *Доклады ТУСУРа.* — 2017. — Т. 20, № 4. — С. 71–74.

33. *Шабля Ю. В., Кручинин Д. В.* Модификация метода построения алгоритмов комбинаторной генерации на основе применения теории производящих функций // *Доклады ТУСУРа.* — 2019. — Т. 22, № 3. — 6 с.

34. *Kruchinin D. V., Shablya Y. V.* Explicit formulas for Meixner polynomials // *International Journal of Mathematics and Mathematical Sciences.* — 2015. — Vol. 2015. — P. 1–5.

35. A method for obtaining coefficients of compositional inverse generating functions / D. V. Kruchinin, Y. V. Shablya, V. V. Kruchinin, A. A. Shelupanov // Proceeding of International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2015). — Vol. 1738. — 2016. — P. 1–4.

36. Integer properties of a composition of exponential generating functions / D. V. Kruchinin, Y. V. Shablya, O. O. Evsutin, A. A. Shelupanov // Proceeding of International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2016). — Vol. 1863. — 2017. — P. 1–4.

37. Properties of a composition of exponential and ordinary generating functions / D. V. Kruchinin, Y. V. Shablya, V. V. Kruchinin, A. A. Shelupanov // *Communications in Mathematics and Applications.* — 2018. — Vol. 9, no. 4. — P. 705–711.

38. A library for calculating polynomials based on compositae of generating functions / D. V. Kruchinin, V. S. Melman, Y. V. Shablya, A. A. Shelupanov // Proceeding of International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2018). — Vol. 2116. — 2019. — P. 1–4.

39. Explicit formulas for the Eulerian numbers of the second kind / D. V. Kruchinin, V. V. Kruchinin, Y. V. Shablya, A. A. Shelupanov // Proceeding of International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2018). — Vol. 2116. — 2019. — P. 1–4.

40. *Shablya Y. V.* The Catalan and Euler number triangles and their application // Материалы международной научно-технической конференции студентов, аспирантов и молодых ученых «Научная сессия ТУСУР-2018» (16-18 мая 2018 г.). — Т. 4. — Томск: 2018. — С. 216–219.

41. *Шабля Ю. В., Мельман В. С., Репкин А. С.* Исследование метода построения алгоритмов генерации комбинаторных объектов на основе деревьев И/ИЛИ // Материалы докладов XIV международной научно-практической конференции «Электронные средства и системы управления» (28-30 ноября 2018 г.), в 2 частях. — Т. 2. — Томск: В-Спектр, 2018. — С. 20–22.

42. *Шабля Ю. В., Репкин А. С., Кручинин Д. В.* Представление комбинаторных объектов в виде структуры дерева И/ИЛИ // Материалы докладов XIV международной научно-практической конференции «Электронные средства и системы управления» (28-30 ноября 2018 г.), в 2 частях. — Т. 2. — Томск: В-Спектр, 2018. — С. 12–15.

43. *Shablya Y. V., Kruchinin D. V.* Euler-Catalan's number triangle and its application // Proceedings book of the Mediterranean International Conference of Pure & Applied Mathematics and Related Areas (26-29 October 2018). — Antalya: 2018. — P. 212–216.

44. *Шабля Ю. В., Репкин А. С., Кручинин Д. В.* Анализ метода разработки алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ // Сборник научных трудов XII всероссийской научной конференции «Наука. Технологии. Инновации» (3-7 декабря 2018 г.), в 9 частях. — Т. 2. — Новосибирск: Изд-во НГТУ, 2018. — С. 78–82.

45. *Шабля Ю. В., Кручинин Д. В., Репкин А. С.* Сравнение подходов к разработке алгоритмов комбинаторной генерации на примере множеств перестановок и сочетаний // Материалы V международной научно-практической конференции молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук» (22-24 апреля 2019 г.). — Тольятти: 2019. — С. 75–80.

46. *Репкин А. С., Филиппов Г. А., Шабля Ю. В.* Анализ современного состояния исследований в области разработки алгоритмов комбинаторной генерации // Сборник избранных статей научной сессии ТУСУРа (22-24 мая 2019 г.), в 2 частях. — Т. 2. — Томск: В-Спектр, 2019. — С. 35–37.

47. *Kruchinin V. V., Kruchinin D. V., Shablya Y. V.* On some properties of generalized Narayana numbers // Abstracts book of the 32th International Conference of the Jangjeon Mathematical Society (17-19 July 2019). — 2019.

48. *Shablya Y. V., Kruchinin D. V.* Application of the method of compositae in combinatorial generation // Proceedings book of the 2nd Mediterranean International Conference of Pure & Applied Mathematics and Related Areas (28-31 August 2019). — Paris: 2019.

49. *Шабля Ю. В., Кручинин Д. В., Мельман В. С.* Программа «PCG: Primality Criterion Generator» для генерации критериев простоты числа // Свидетельство о государственной регистрации программы для ЭВМ №2018611242 от 26.01.2018 г.

50. *Шабля Ю. В., Кручинин Д. В., Мельман В. С.* Программа «PTA: Primality Test Analyser» для анализа тестов простоты числа // Свидетельство о государственной регистрации программы для ЭВМ №2018611241 от 26.01.2018 г.

51. *Шабля Ю. В., Кручинин Д. В.* Программа для ранжирования и генерации по рангу элементов комбинаторных множеств // Свидетельство о государственной регистрации программы для ЭВМ №2019660020 от 29.07.2019 г.

52. *Loehr N.* Combinatorics: Discrete mathematics and its applications. — USA: CRC Press, 2017. — 610 p.

53. Ranking and unranking of well-formed parenthesis strings: A unified approach / R.-Y. Wu, J.-M. Chang, A.-H. Chen, C.-L. Liu // *Chiang Mai Journal of Science.* — 2012. — Vol. 39, no. 4. — P. 648–659.

54. *Rizzi R., Tomescu A. I.* Ranking, unranking and random generation of extensional acyclic digraphs // *Information Processing Letters.* — 2013. — Vol. 113, no. 5–6. — P. 183–187.

55. *Mikawa K., Tanaka K.* Lexicographic ranking and unranking of derangements in cycle notation // *Discrete Applied Mathematics.* — 2014. — Vol. 166. — P. 164–169.

56. *Graham R. L., Knuth D. E., Patashnik O.* Concrete mathematics. — Second edition. — USA: Addison-Wesley, 1994. — 657 p.

57. *Molinero X., Vives J.* Unranking algorithms for combinatorial structures // *International Journal of Applied Mathematics and Informatics.* — 2015. — Vol. 9. — P. 110–115.

58. *Кручинин В. В.* Представление множеств деревьями И/ИЛИ // *Доклады ТУСУРа.* — 2008. — Т. 1, № 17. — С. 107–112.

59. *Stanley R. P.* Enumerative combinatorics. Volume 1. — Second edition. — USA: Cambridge University Press, 2011. — 640 p.

60. *Кручинин В. В., Кручинин Д. В.* Степени производящих функций и их применение. — Томск: Издательство ТУСУРа, 2013. — 236 с.

61. *Kruchinin D. V., Kruchinin V. V.* A method for obtaining generating functions for central coefficients of triangles // *Journal of Integer Sequences.* — 2012. — Vol. 15, no. 9. — P. 1–10.

62. *Kruchinin D. V., Kruchinin V. V.* Application of a composition of generating functions for obtaining explicit formulas of polynomials // *Journal of Mathematical Analysis and Applications.* — 2013. — Vol. 404, no. 1. — P. 161–171.

63. *Kruchinin V. V., Kruchinin D. V.* Composita and its properties // *Journal of Analysis and Number Theory.* — 2014. — Vol. 2, no. 2. — P. 37–44.

64. *Перминова Мария Юрьевна.* Алгоритмы и программный модуль получения явных выражений коэффициентов производящих функций : дис. ... канд. техн. наук : 05.13.17. — Томск, 2017. — 113 с.

65. *Sloane N. J. A.* The on-line encyclopedia of integer sequences [Электронный ресурс]. — URL: https://oeis.org (дата обращения: 01.08.2019).

66. *Deutsch E.* Dyck path enumeration // *Discrete Mathematics.* — 1999. — Vol. 204, no. 1–3. — P. 167–202.

67. *Stanley R. P.* Enumerative combinatorics. Volume 2. — USA: Cambridge University Press, 2001. — 595 p.

68. *Yan S. Y.* Primality testing and integer factorization in public-key cryptography. — Second edition. — USA: Springer, 2009. — 371 p.

69. Актуальные направления развития методов и средств защиты информации / А. А. Шелупанов, О. О. Евсютин, А. А. Конев и др. // *Доклады ТУСУРа.* — 2017. — Т. 20, № 3. — С. 11–24.

70. Метод получения явных выражений полиномов на основе степеней производящих функций и его реализация / Д. В. Кручинин, В. В. Кручинин, А. А. Шелупанов и др. — Томск: В-Спектр, 2017. — 172 с.

71. Realization of a method for calculating Bell polynomials based on compositae of generating functions / V. S. Melman, Y. V. Shablya, D. V. Kruchinin, A. A. Shelupanov // *Journal of Informatics and Mathematical Sciences*. — 2018. — Vol. 10, no. 4. — P. 659–672.

72. *Kruchinin D. V., Kruchinin V. V., Shablya Y. V.* Obtaining explicit formulas and identities for polynomials defined by generating functions of the form $F(t)^x G(t)^a$ // Polynomials — Theory and application. — IntechOpen, 2019.

73. *Гросс М., Лантен А.* Теория формальных грамматик. — М.: Мир, 1971. — 296 с.

74. *Stanley R. P.* Catalan addendum. — 2013. — 96 p. — URL: http://www-math.mit.edu/~rstan/ec/catadd.pdf.

75. *Stanley R. P.* Catalan numbers. — USA: Cambridge University Press, 2015. — 215 p.

76. *Liebehenschel J.* Ranking and unranking of a generalized Dyck language and the application to the generation of random trees // *Seminaire Lotharingien de Combinatoire*. — 2000. — Vol. 32, no. 4. — P. 880–903.

77. *Liebehenschel J.* Lexicographical generation of a generalized Dyck language // *SIAM Journal on Computing*. — 2003. — Vol. 43. — P. 1–19.

78. *Абрамов С. А.* Лекции о сложности алгоритмов. — М.: Издательство МЦНМО, 2009. — 256 с.

79. *Зенгер В.* Принципы структурной организации нуклеиновых кислом. — М.: Мир, 1987. — 584 с.

80. *Reidys C.* Combinatorial computational biology of RNA: Pseudoknots and neutral networks. — USA: Springer, 2011. — 257 p.

81. *Waterman M. S.* Introduction to computational biology. — USA: CRC Press, 2018. — 448 p.

82. *Montaseri S., Zare-Mirakabad F., Ganjtabesh M.* Evaluating the quality of SHAPE data simulated by k-mers for RNA structure prediction // *Journal of Bioinformatics and Computational Biology.* — 2017. — Vol. 15, no. 6.

83. *Legendre A., Angel E., Tahi F.* Bi-objective integer programming for RNA secondary structure prediction with pseudoknots // *BMC Bioinformatics.* — 2018. — Vol. 19, no. 1.

84. *Kabir M. R., Zahra F. T., Islam M. R.* RNA structure prediction using chemical reaction optimization // *Advances in Intelligent Systems and Computing.* — 2019. — Vol. 755. — P. 587–598.

85. A review on RNA secondary structure prediction algorithms / I. K. Oluoch, A. Akalin, Y. Vural, Y. Canbay // International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT 2018). — 2019.

86. A new method of RNA secondary structure prediction based on convolutional neural network and dynamic programming / H. Zhang, C. Zhang, Z. Li et al. // *Frontiers in Genetics.* — 2019. — Vol. 10.

87. Integrative analysis of somatic mutations in non-coding regions altering RNA secondary structures in cancer genomes / F. He, R. Wei, L. Zhou, Z. Huang et al. // *Scientific Reports.* — 2019. — Vol. 9, no. 1.

88. *Leger S., Costa M. B. W., Tulpan D.* Pairwise visual comparison of small RNA secondary structures with base pair probabilities // *BMC Bioinformatics.* — 2019. — Vol. 20, no. 1.

89. *Surujon D., Ponty Y., Clote P.* Small-world networks and RNA secondary structures // *Journal of Computational Biology.* — 2019. — Vol. 26, no. 1. — P. 16–26.

90. Global importance of RNA secondary structures in protein-coding sequences / M. Fricke, R. Gerst, B. Ibrahim et al. // *Bioinformatics.* — 2019. — Vol. 35, no. 4. — P. 579–583.

91. *Picardi E.* RNA bioinformatics. — USA: Humana Press, 2015. — 415 p.

92. *Seyedi-Tabari E., Ahrabian H., Nowzari-Dalini A.* A new algorithm for generation of different types of RNA // *International Journal of Computer Mathematics.* — 2010. — Vol. 87, no. 6. — P. 1197–1207.

93. *Nebel M. E., Scheid A., Weinberg F.* Random generation of RNA secondary structures according to native distributions // *Algorithms for Molecular Biology.* — 2011. — Vol. 6, no. 1.

94. *Panayotopoulos A., Vlamos P.* Cutting degree of meanders // IFIP Advances in Information and Communication Technology. — Vol. 382. — 2012. — P. 480–489.

95. *Petersen T. K.* Eulerian numbers. — USA: Birkhauser Advanced Texts Basler Lehrbucher, 2015. — 463 p.

96. *Ландо С. К.* Введение в дискретную математику. — М.: Издательство МЦНМО, 2014. — 264 с.

97. Maxima, a Computer Algebra System [Электронный ресурс]. — URL: http://maxima.sourceforge.net/ (дата обращения: 01.08.2019).

98. *Стахин Н. А.* Основы работы с системой аналитических (символьных) вычислений Maxima. — М., 2008. — 86 с.

99. *Чичкарев Е. А.* Компьютерная математика с Maxima. — М.: ALT Linux, 2012. — 384 с.

100. Wolfram Mathematica: Современные технические вычисления [Электронный ресурс]. — URL: http://www.wolfram.com/mathematica/ (дата обращения: 01.08.2019).

101. Maplesoft – Software for mathematics, online learning, engineering [Электронный ресурс]. — URL: https://www.maplesoft.com/ (дата обращения: 01.08.2019).

102. wxMaxima [Электронный ресурс]. — URL: https://wxmaxima-developers.github.io/wxmaxima/ (дата обращения: 01.08.2019).

Приложение А

Свидетельства о государственной регистрации программ для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2019660020

Программа для ранжирования и генерации по рангу элементов комбинаторных множеств

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего образования «Томский государственный университет систем управления и радиоэлектроники» (ТУСУР) (RU)***

Авторы: ***Шабля Юрий Васильевич (RU),***
Кручинин Дмитрий Владимирович (RU)

Заявка № **2019618795**
Дата поступления **17 июля 2019 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***29 июля 2019 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2018611242

Программа «PCG: Primality Criterion Generator» для генерации критериев простоты числа

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего образования «Томский государственный университет систем управления и радиоэлектроники» (ТУСУР) (RU)***

Авторы: ***Шабля Юрий Васильевич (RU), Кручинин Дмитрий Владимирович (RU), Мельман Вадим Сергеевич (RU)***

Заявка № **2017662540**
Дата поступления **04 декабря 2017 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***26 января 2018 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2018611241

Программа «PTA: Primality Test Analyser» для анализа тестов простоты числа

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего образования «Томский государственный университет систем управления и радиоэлектроники» (ТУСУР) (RU)***

Авторы: ***Шабля Юрий Васильевич (RU), Кручинин Дмитрий Владимирович (RU), Мельман Вадим Сергеевич (RU)***

Заявка № **2017662547**
Дата поступления **04 декабря 2017 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***26 января 2018 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

Приложение Б

Акты внедрения

ПЛАНТА плюс
Биотехнологическое предприятие

Общество с ограниченной ответственностью «ПлантаПлюс»
Фрунзе пр., 20, оф. 411, 413, г. Томск, 634029, тел.: (3822) 202-334, http://www.planta-plus.ru
ИНН / КПП: 7017102395 / 701701001, ОГРН: 1047000204096, р/с: 40702810506290004355, к/с: 30101810500000000728,
ПАО «Томскпромстройбанк» г. Томск, БИК: 046902728

УТВЕРЖДАЮ
Директор ООО «ПлантаПлюс»
_____ И. М. Галактионов
«__» _____ 2019 г.

АКТ
о внедрении результатов диссертационной работы
Шабля Юрия Васильевича

Комиссия в составе:

председатель комиссии – Кулятов Дмитрий Васильевич, начальник лаборатории промышленной микробиологии ООО «ПлантаПлюс»

члены комиссии:

- Николаева Дарья Леонидовна, научный сотрудник лаборатории промышленной микробиологии ООО «ПлантаПлюс»

- Кузьминская Елена Анатольевна, химик-лаборант ООО «ПлантаПлюс».

составили настоящий акт о том, что результаты диссертационной работы Шабля Ю.В. внедрены в деятельность ООО «ПлантаПлюс» в процессе работы над систематизацией архива проведения экспериментальных исследований.

На основе применения описанного в диссертационной работе Шабля Ю.В. математического аппарата, для архива проведения экспериментальных исследований был применен подход его представления в

виде комбинаторного множества, для которого было получено его представление в виде структуры дерева И/ИЛИ. Это позволило объединить разрозненные базы данных и систематизировать имеющийся архив проведения экспериментальных исследований, что, в свою очередь, способствует повышению эффективности работы с архивом за счет сокращения времени на поиск результатов требуемого эксперимента.

Применение предложенного в диссертационной работе Шабля Ю. В. модифицированного метода построения алгоритмов ранжирования и генерации по рангу для полученного представления в виде структуры дерева И/ИЛИ позволило выполнить кодирование хранящихся результатов экспериментов. Положительным эффектом от кодирования является сокращение на 17% общего объема базы данных архива проведения экспериментальных исследований.

Председатель комиссии _____ Кулятов Д.В

Члены комиссии: _____ Николаева Д.Л.

_____ Кузьминская Е.А.

Удостоверяющий центр Сибири

Общество с ограниченной ответственностью
«Удостоверяющий центр Сибири»

634009, г. Томск, пр-т Ленина, 110
ИНН/КПП/ОГРН 7017311494/701701001/1127017020767

тел: (382 2) 900-111
факс: (382 2) 900-111
e-mail: office@udcs.ru
http:// www.udcs.ru

УТВЕРЖДАЮ
Директор ООО «УЦ Сибири»
_____ А.В. Перфильев
«22» 09 2019 г.

АКТ
о внедрении результатов диссертационной работы
Шабля Юрия Васильевича

Комиссия в составе:

- Перфильев А.В., Директор – председатель комиссии;
- Хабибулин Д.И., руководитель технической службы;
- Саевец А.М., специалист по ЗИ

составила настоящий акт о том, что результаты диссертационной работы Шабля Ю.В. внедрены в деятельность ООО «УЦ Сибири» в процессе создания программного продукта для работы с алгоритмами получения простых чисел.

Целочисленные свойства композиции обыкновенных и экспоненциальных производящих функций, полученные в диссертационной работе Шабля Ю.В. в рамках исследований, направленных на изучение и апробацию математического аппарата степеней производящих функций для модификации метода комбинаторной генерации, были применены в работе алгоритмов генерации простых чисел. Используемый в алгоритмах генерации простых чисел тест проверки простоты числа был дополнен новыми критериями простоты числа. Данные критерии простоты числа были получены с помощью разработанного Шабля Ю.В. программного обеспечения для генерации новых критериев

простоты числа. Комбинирование полученных результатов проверки генерируемых чисел с помощью теста простоты числа с результатами проверки теста, основанного на новых критериях простоты числа, позволило сократить от 5 до 7% количество необнаруженных псевдопростых чисел, что является значимым результатом при обработке больших выборок чисел.

Председатель комиссии _____ А.В. Перфильев

Члены комиссии: _____ Д.И. Хабибулин

_____ А.М. Саевец

Общество с ограниченной ответственностью Удостоверяющий Центр Сибири РОССИЯ г. Томск ОГРН 1127017020767 ИНН 7017161454

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

УТВЕРЖДАЮ
Директор департамента образования ТУСУР
_____ П.Е. Троян
«12» 09 2019 г.

АКТ
о внедрении результатов диссертационной работы
Шабля Юрия Васильевича в учебный процесс

Комиссия в составе:

- Давыдова Е.М., к.т.н., декан факультета безопасности ТУСУР – председатель комиссии;
- Конев А.А., к.т.н., доцент кафедры КИБЭВС ТУСУР;
- Сарин К.С., к.т.н., доцент кафедры КИБЭВС ТУСУР;
- Кручинин Д.В., к.ф.-м.н., доцент кафедры КИБЭВС ТУСУР

составила настоящий акт о нижеследующем.

Результаты диссертационной работы Шабля Ю.В. используются в учебном процессе на факультете безопасности ТУСУР при чтении курса лекций и проведении практических занятий по дисциплинам «Дискретная математика» и «Теория игр и исследование операций» для подготовки специалистов по защите информации, обучающихся по специальностям «10.03.01 – Информационная безопасность», «10.05.02 – Информационная безопасность телекоммуникационных систем», «10.05.03 – Информационная безопасность автоматизированных систем» и «10.05.04 – Информационно-аналитические системы безопасности».

В курсе «Дискретная математика» используются результаты работы Шабля Ю.В. по разработке представлений элементов комбинаторных множеств в различных формах записи, в том числе и в форме структуры дерева И/ИЛИ, а также

по разработке алгоритмов отображения элементов комбинаторных множеств в варианты дерева И/ИЛИ. Это позволяет студентам ознакомиться с процессом реализации биективного отображения между двумя множествами.

В курсе «Теория игр и исследование операций» используется предложенный Шабля Ю.В. модифицированный метод построения алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ, позволяющий студентам ознакомиться с процессом разработки алгоритмов кодирования объектов комбинаторных множеств и применить их на практике.

Кроме того, студенты факультета безопасности имеют возможность ознакомиться с результатами диссертационного исследования в ходе выполнения групповых проектов, научно-исследовательских и дипломных работ.

Освоение студентами предложенного Шабля Ю.В. алгоритмического обеспечения позволяет сформировать навыки работы с дискретными структурами, их перечисления и кодирования.

Настоящий акт составлен в 3 (трех) экземплярах.

Давыдова Е.М.
к.т.н., декан факультета
безопасности ТУСУР
«12» 09 2019 г.

Конев А.А.
к.т.н., доцент кафедры
КИБЭВС ТУСУР
«12» 09 2019 г.

Сарин К.С.
к.т.н., доцент кафедры
КИБЭВС ТУСУР
«12» 09 2019 г.

Кручинин Д.В.
к.ф.-м.н., доцент кафедры
КИБЭВС ТУСУР
«12» 09 2019 г.