

Московский государственный университет
имени М.В.Ломоносова
Механико-математический факультет

На правах рукописи

Перпер Евгений Михайлович

Поиск подслоа в множестве слов и его приложение к
семантическому анализу текстов

Специальность 01.01.09 — дискретная математика и
математическая кибернетика

Диссертация

на соискание ученой степени
кандидата физико-математических наук

Научный руководитель:
доктор физико-математических наук,
профессор Э.Э.Гасанов

Москва – 2018

Оглавление

Введение	3
1 Общая характеристика работы	3
2 Содержание работы	9
1 Поиск подслова в множестве слов	16
1 Обзор литературы	16
2 Определения	20
3 Нижняя оценка сложности поиска подслова в множестве слов	24
4 Информационный граф, осуществляющий поиск с минимальной временной сложностью	25
5 Оценка порядка объемной сложности информационного графа, имеющего минимальную временную сложность	29
2 Поиск вхождений подслова в множестве слов	40
1 Определения	40
2 Нижняя оценка сложности поиска вхождений подслова в множестве слов	42
3 Нижняя оценка объема памяти алгоритма, решающего задачу поиска вхождений подслова	43
4 Решающий задачу поиска вхождений подслова информационный граф с минимальной по порядку объемной и временной сложностью	46
3 Поиск подслова с известной длиной в множестве слов	56
1 Определения	56
2 Оценки сложности поиска подслова с известной длиной в множестве слов при ограничениях на объем памяти	57
4 Построение модели нормативных актов	69
1 Постановка задачи	69
2 Построение связного синтаксического графа предложения	73
3 Упрощение синтаксического графа предложения	77
4 Отождествление сущностей	80
5 Построение логической формулы	84
6 Построение модели закона по логическим формулам	90
Приложение 1. Перечень правил синтаксического анализа	96
Приложение 2. Пример построения синтаксического графа предложения	104
Заключение	109
Список литературы	111

Введение

§ 1 Общая характеристика работы

Актуальность темы исследования

В работе рассматривается задача поиска в базе данных, представляющей собой множество слов, всех слов, содержащих некоторое заданное слово в качестве подслова. Такая задача часто возникает на практике, например, когда необходимо найти слово в словаре. В частности, если рассматриваемое слово — имя существительное, находящееся не в именительном падеже, или глагол, находящийся не в инфинитиве, в словаре его может не оказаться. Тем не менее, в словаре найдутся слова, содержащие некоторое подслово рассматриваемого слова, например, его основу. Желание проводить поиск этих слов как можно быстрее приводит к задаче, которой посвящена работа. В диссертации рассматриваются две вариации данной задачи: собственно задача поиска подслова в множестве слов, в которой требуется найти все слова из множества, содержащие данное подслово, а также задача поиска вхождений подслова в множестве слов, требующая найти не только каждое слово из множества, содержащее данное подслово, но и позицию в этом слове, с которой данное подслово начинается.

Широко исследована вариация задачи поиска вхождений подслова в множестве слов, в которой множество слов состоит из одного слова (это слово часто называют «текстом»). Некоторые алгоритмы, осуществляющие поиск вхождений образца в текст, можно обобщить на случай, когда нужно найти все вхождения образца в несколько текстов (например, такое обобщение описано в работе [1]). Задаче поиска вхождений подслова в слово посвящены, в частности, работы [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. Для задачи поиска в базе данных, рассматриваемой в настоящей работе, наибольший интерес представляют алгоритмы, где на основе текста строится некоторая структура, позволяющая для любого

быстро находить все вхождения произвольного подслова в текст. Самые быстрые алгоритмы, например, алгоритм, описанный в работе [13] (Ферраджина, Манцини, Мякинен и Наварро, 2006), позволяют осуществлять поиск за время, пропорциональное сумме длины подслова и количества вхождений подслова в текст. Алгоритмы, оптимальные по объему используемой памяти среди самых быстрых алгоритмов (в том числе алгоритм из работы [13]), требуют не более $O(n \log k)$ битов памяти, где n — длина текста, а k — количество букв в алфавите. В этих алгоритмах текст используется только при построении структуры, используемой при поиске, причем для некоторых текстов эта структура требует меньше памяти, чем сам текст.

Некоторые исследования по задаче поиска вхождений подслова в слово, в частности, [14, 15, 16], посвящены нижним оценкам памяти алгоритмов. В частности, в работе [14] Э.Д. Демейн и А. Лопес-Ортис показали, что нижняя оценка объема памяти самых быстрых алгоритмов пропорциональна объему памяти, необходимому для хранения текста. В целом, однако, исследований, посвященных нижним оценкам объема памяти для задачи поиска вхождений подслова в слово, значительно меньше, чем исследований, посвященных верхним оценкам объема памяти для той же задачи. Подробнее о работах, посвященных поиску подслова в слове, рассказано в параграфе 1 главы 1.

В настоящей работе используется информационно-графовая модель, предназначенная для формализации задач поиска в базах данных. В работах [17, 18] можно найти подробное описание модели и обоснование ее актуальности и полезности для построения и анализа алгоритмов поиска информации в базах данных. В данной модели алгоритм поиска описывается ориентированным графом. Вершины и ребра графа этого графа нагружены элементами данных, а также функциями, определенными на множестве возможных запросов к базе данных. Данная модель удобна для получения оценок (в том числе нижних) объема памяти, используемой при поиске, так как в этой модели объем памяти определяется числом ребер графа. При получении оценок предполагается, что при построении графа можно использовать только функции из некоторого заранее определенного множества. Для минимального по времени алгоритма поиска подслова в множестве слов получена нижняя оценка необходимой памяти по-

рядка pn^2 , где p — число слов в базе данных, а n — длина каждого из этих слов (считаем, что все слова в базе данных имеют одинаковую длину). Для случая, когда предполагается, что длина подслова подается на вход алгоритма вместе с самим подсловом, получена нижняя оценка сложности алгоритма, а также найдены верхние оценки сложности алгоритма при ограничениях на объем памяти. Для поиска вхождений подслова в множестве слов предложен алгоритм, фактически представляющий собой модификацию обобщенного суффиксного дерева. Он требует $O(pn)$ памяти. Показано, что время работы предложенного алгоритма превышает нижнюю оценку не более чем в три раза, а объем памяти, нужной для этого алгоритма, превышает нижнюю оценку не более чем в $2k + 11$ раз, где k — число букв в алфавите (k считается константой).

Далее в работе предложенные алгоритмы поиска подслова и вхождений подслова в множестве слов используются в процессе решения задачи автоматического выделения смысла некоторого текста на естественном языке. Эта задача требует многократного поиска слов в различных словарях, поэтому время работы алгоритмов ее решения существенно зависит от эффективности используемых алгоритмов поиска.

Как правило, для произвольного текста трудно формально определить, в чем состоит его смысл, но юридический документ описывает некоторые алгоритмы действий в определенных ситуациях, и эти алгоритмы можно считать смыслом данного документа. В частности, закон, посвященный бухгалтерскому учету, описывает схемы вычисления значений упомянутых в законе объектов. Можно считать, что смысл закона выделен правильно, если и только если вычисленные по этим схемам значения объектов совпадают со значениями, вычисленными в соответствии с текстом закона. Задача построения таких схем и исследуется в данной работе. Предложен алгоритм, позволяющий строить такие схемы.

На первом этапе работы алгоритма осуществляется синтаксический анализ каждого предложения текста закона. Программы, проводящие синтаксический анализ текстов, как правило, основаны либо на машинном обучении на синтаксически размеченных корпусах [19, 20, 21, 22], либо на использовании некоторого набора правил, указывающих, как находить синтаксические связи между

словами из предложения [23, 24, 25]. Более подробно об этих двух типах программ рассказано в параграфе 1 главы 4. Алгоритм синтаксического анализа, предложенный в настоящей работе, относится ко второму типу.

Следующий этап работы алгоритма построения схем, вычисляющих значения объектов в соответствии с текстом закона, состоит в семантическом анализе текста. Существуют программы, проводящие такой анализ для текста на русском языке, см., например, [23, 26]. В настоящей работе целью семантического анализа является построение формул логики предикатов, описывающих отношения между рассматриваемыми в тексте объектами. В работе применяется определенный набор правил, позволяющий строить такие формулы по синтаксически разобранным предложениям текста.

На последнем этапе создается модель закона, представляющая собой совокупность схем вычисления значений объектов, упомянутых в законе. Каждая такая схема является своеобразным аналогом алгебраического дерева вычислений [27].

Работоспособность предложенного алгоритма проверена на примере положения по бухгалтерскому учету ПБУ 6/01[28].

Цель работы

Целями работы являются:

- получение нижних оценок объема памяти алгоритмов поиска в базах данных в зависимости от времени поиска информации при некоторых ограничениях на используемые при поиске функции;
- построение алгоритма поиска слов и вхождений слов в множестве слов с характеристиками по объему и памяти, оптимальными при некоторых ограничениях на используемые при поиске функции;
- получение оценок сложности алгоритма поиска слов для случая, когда длина подслова подается на вход алгоритма вместе с самим подсловом, при ограничениях на объем памяти и используемые при поиске функции;
- создание алгоритма перевода текстов нормативных документов на русском языке в формальный язык логики предикатов;

- проверка работоспособности данного алгоритма на примере положения по бухгалтерскому учету ПБУ 6/01.

Методы исследования

В работе используются методы дискретной математики, теории сложности управляющих систем, математической логики, математической лингвистики.

Научная новизна

Для задачи поиска подслова в множестве слов найдено значение минимального времени поиска при некоторых ограничениях на используемые при поиске функции и получена нижняя оценка объема памяти, необходимого для алгоритмов, осуществляющих поиск за минимальное время. Для задачи поиска вхождений подслова в множестве слов найдено значение минимального времени поиска при некоторых ограничениях на используемые при поиске функции и построен алгоритм, осуществляющий поиск за минимальное по порядку время и требующий минимальной по порядку памяти. Для задачи поиска вхождений подслова в множестве слов в случае, когда длина подслова подается на вход алгоритма вместе с самим подсловом, и при некоторых ограничениях на используемые при поиске функции, найдена нижняя оценка сложности поиска, а также получены верхние оценки сложности поиска при ограничениях на объем памяти.

В работе получен перечень правил, позволяющий по тексту нормативно-правового акта построить его синтаксический граф, а затем — формулы логики предикатов, позволяющие вычислять значения объектов, о которых идет речь в нормативно-правовом акте.

Практическая ценность

Алгоритмы поиска, описанные в работе, можно использовать для быстрого нахождения, например, всех слов в словаре, содержащих данный корень, с использованием небольшого объема памяти.

Результаты работы могут быть использованы в ERP-системах (Enterprise Resource Planning — Управление ресурсами предприятия), частью которых являются программы, заполняющие различные формы отчетности. Сейчас эти

программы пишутся вручную. При каждом изменении в законах, определяющих правила вычисления значений, которыми нужно заполнять формы отчетности, приходится находить, как именно эти изменения затронули программу, и соответствующим образом программу изменять. В данной работе описан способ автоматического построения схемы вычисления значений, которыми заполняются формы отчетности. Это позволило создать компьютерную программу, вычисляющую значения объектов в соответствии с инструкциями, описанными в нормативном акте. Последнее позволяет избавиться от труда программистов.

Описанные в работе методы обработки текста, написанного на естественном языке, можно использовать при решении различных задач математической лингвистики.

Апробация работы

1. Семинар «Теория автоматов» под руководством академика, профессора, д.ф-м.н. В.Б. Кудрявцева (2013–2017 гг., неоднократно);
2. Семинар «Вопросы сложности алгоритмов поиска» под руководством проф., д.ф-м.н. Э.Э.Гасанова (2010–2017 гг., неоднократно);
3. XI Международная конференция «Интеллектуальные системы и компьютерные науки» (2016, Москва, МГУ).
4. XII Международный семинар «Дискретная математика и ее приложения» (2016 г., Москва, МГУ);
5. Международная конференция студентов, аспирантов и молодых ученых «Ломоносов» (2013 г., 2014 г., 2015 г., Москва, МГУ);
6. Научная конференция «Ломоносовские чтения» (2013 г., 2014 г., 2016 г., Москва, МГУ);
7. XI Международный семинар «Дискретная математика и ее приложения», посвященный 80-летию со дня рождения академика О.Б. Лупанова (2012, Москва, МГУ);
8. X Международная конференция «Интеллектуальные системы и компьютерные науки» (2011, Москва, МГУ).

Публикации

Основное содержание диссертации опубликовано в 8 печатных работах, 5 из которых опубликованы в рецензируемых научных изданиях, определенных п.2.3 Положения о присуждении ученых степеней в Московском государственном университете имени М.В.Ломоносова. В работе, написанной в соавторстве, результаты автора диссертации являются определяющими.

Структура и объем работы

Диссертация состоит из введения и трех глав. Объем диссертации — 114 страниц. Список литературы содержит 38 наименований.

§ 2 Содержание работы

В главе 1 рассматривается задача поиска подслова в множестве слов, притом предполагается, что при поиске можно использовать лишь функции из определенного множества. Приводится нижняя оценка времени работы алгоритма, осуществляющего такой поиск. Также приводится нижняя оценка объема памяти, требуемого для поиска за наименьшее время, и описан алгоритм, производящий поиск за минимально возможное время и требующий минимальной по порядку памяти.

Будем использовать следующие обозначения:

Обозначим через N_k множество, содержащее все натуральные числа от 1 до k . Будем считать, что слово длины s над алфавитом N_k представлено набором длины n , первые s элементов которого — из множества N_k , а оставшиеся элементы — нули. Длину слова w обозначим через $l(w)$.

Назовем библиотекой множество, содержащее некоторые слова длины n над алфавитом N_k . Будем рассматривать задачу информационного поиска (ЗИП) I_1 , состоящую в перечислении для произвольного запроса x (слова длины не более n над алфавитом N_k) всех слов из некоторой библиотеки, содержащих x в качестве подслова. Назовем эту задачу задачей поиска подслова. Множество всех запросов обозначим через X .

При решении задачи поиска подслова в настоящей работе используется информационно-графовая модель данных [17, 18]. В данной модели база дан-

ных представлена в виде ориентированного графа. Некоторым вершинам графа сопоставлено одно слово из библиотеки. Кроме того, ребрам и вершинам графа сопоставлены функции из некоторого базового множества. Единственным аргументом каждой функции из базового множества является запрос. Для информационного графа задано функционирование: в результате вычисления функции, сопоставленной вершине или ребру, запрос может перейти по ребру из вершины в другую вершину; изначально запрос находится в корне графа. Если запрос проходит в вершину, которой сопоставлено слово из библиотеки, то это слово попадает в ответ информационного графа на запрос. Если множество слов, оказавшихся в ответе информационного графа на запрос, совпадает с множеством слов из библиотеки, содержащих запрос в качестве подслова, то будем говорить, что информационный граф решает задачу поиска подслова.

Базовое множество \mathcal{F}_1 , используемое при решении задачи поиска подслова, состоит из функции, тождественно равной 1, которая сопоставляется ребрам для безусловного перехода по ним, а также набора функций $g_i(x)$, сопоставляемых вершинам. Значение функции $g_i(x)$ равно i -й букве запроса x , а если длина запроса меньше i , то $k + 1$. Обозначим через $\mathcal{U}(I_1, \mathcal{F}_1)$ множество всех информационных графов над базовым множеством \mathcal{F}_1 , решающих задачу поиска подслова.

Объемом $Q(U)$ информационного графа U назовем число ребер в графе U . Эта величина соответствует объему памяти, используемой информационным графом. Еще одна характеристика информационного графа U — его сложность $T(U, x)$ на некотором запросе x — характеризует время, в течение которого граф обрабатывает запрос. Будем считать, что сложность вычисления функции $g_i(x)$ равна 1, а сложность вычисления функции, тождественно равной 1, равна некоторому числу t , причем $0 < t < 1$.

Обозначим через $d(I_1, x)$ количество слов из библиотеки, содержащих x как подслово. Величину $\min(l(x) + 1, n) + t \cdot (d(I_1, x) - 1)$ будем обозначать как $R(I_1, \mathcal{F}_1, x)$. Как будет видно из теорем 1 и 2, для некоторых библиотек данная величина представляет собой минимальную сложность информационного графа, решающего задачу поиска подслова, на запросах, которым удовлетворяет хотя бы одно слово из библиотеки.

Теорема 1. При $k \geq 3$ среди всех библиотек $V \subseteq N_k^n$ из p слов, где $p \leq (k-1)^n$, найдется такая библиотека V , что для задачи I_1 для любых $U \in \mathcal{U}(I_1, \mathcal{F}_1)$ и $x \in X$ выполняется следующее условие: если $d(I_1, x) \geq 1$, то $T(U, x) \geq R(I_1, \mathcal{F}_1, x)$.

Обозначим через \mathcal{U}_{I_1} множество таких графов над базовым множеством \mathcal{F}_1 , решающих задачу поиска подслова, что сложность каждого из них на любом запросе не превышает $R(I_1, \mathcal{F}_1, x)$.

Теорема 2. Для любой задачи $I_1 = \langle X, V, \rho_1 \rangle$ множество \mathcal{U}_{I_1} непусто.

Для каждой библиотеки из p слов выберем информационный граф наименьшего объема над базовым множеством \mathcal{F}_1 , решающий задачу поиска подслова для этой библиотеки, а затем среди всех полученных графов выберем граф наибольшего объема; его объем обозначим через $Q_1(p, n)$.

Теорема 3. Если $k \geq 3$, $p \leq (k-1)^{\lfloor n/3 \rfloor}$, то $pn^2/9 \lesssim Q_1(p, n) \lesssim (k+2)pn^2/2$ при $n \rightarrow \infty$, $p \rightarrow \infty$.

В главе 2 рассматривается задача поиска вхождений подслова в множестве слов при некоторых ограничениях на множество используемых при поиске функций. Приводится нижняя оценка времени работы алгоритма, осуществляющего такой поиск. Также приведена нижняя оценка объема памяти, требуемого для поиска за наименьшее по порядку время, и указан алгоритм, производящий поиск за минимально возможное по порядку время и требующий минимальной по порядку памяти.

Вхождением в слово будем называть пару, состоящую из слова и позиции в этом слове. Будем рассматривать задачу информационного поиска I_2 , состоящую в нахождении в словах из библиотеки каждой позиции, с которой начинается подслово, совпадающее с запросом. Назовем эту задачу задачей поиска вхождений подслова.

Понятие информационного графа вводится так же, как и для главы 1, с той разницей, что там, где для главы 1 рассматривалось множество слов из библиотеки, в настоящей главе будет рассматриваться множество вхождений в слова из библиотеки.

В качестве базового множества функций при решении задачи поиска вхождений подслова будем рассматривать множество \mathcal{F}_2 , состоящее из всех функций множества \mathcal{F}_1 , используемого в главе 1, а также множества функций $h_{v,i}(x)$. Функция $h_{v,i}(x)$ принимает значение 1, если подслово запроса x , начинающееся с его i -й позиции, является собственным префиксом слова v , либо длина запроса меньше i ; функция принимает значение 2, если v — префикс запроса; наконец, значение функции равно 3 во всех остальных случаях. Сложность вычисления этой функции пропорциональна количеству сравнений букв слова v с буквами запроса при вычислении ее значения. Множество всех информационных графов над базовым множеством \mathcal{F}_2 , решающих задачу поиска вхождений подслова, обозначим через $\mathcal{U}(I_2, \mathcal{F}_2)$.

Обозначим через $J_{I_2}(x)$ множество всех вхождений в слова из библиотеки, удовлетворяющих запросу x в задаче поиска вхождений подслова. Величину $\min(l(x)+1, n)+t \cdot (|J_{I_2}(x)|-1)$ будем обозначать как $R(I_2, \mathcal{F}_2, x)$. В теореме будет показано, что если информационный граф решает задачу поиска вхождений подслова, то сложность обработки им запроса x , являющегося подсловом хотя бы одного слова из библиотеки, не может быть ниже величины $R(I_2, \mathcal{F}_1, x)$.

Теорема 4. *При $k \geq 2$ для каждой задачи $I_2 = \langle X, V, \rho_2 \rangle$, для любых $U \in \mathcal{U}(I_2, \mathcal{F}_2)$ и $x \in X$ выполняется следующее условие: если $|J_{I_2}(x)| \geq 1$, то $T(U, x) \geq R(I_2, \mathcal{F}_2, x)$.*

Обозначим через \mathcal{U}_{fast} множество таких графов $U \in \mathcal{U}(I_2, \mathcal{F}_2)$, что для каждого из них при любых x из X выполняется условие $T(U, x) \leq 3R(I_2, \mathcal{F}_2, x)$.

Для каждой библиотеки из p слов выберем информационный граф наименьшего объема над базовым множеством \mathcal{F}_2 , решающий задачу поиска вхождений подслова для этой библиотеки, а затем среди всех полученных графов выберем граф наибольшего объема; его объем обозначим через $Q_2(p, n)$.

Далее, для каждой библиотеки из p слов среди информационных графов над базовым множеством \mathcal{F}_2 , решающих задачу поиска вхождений подслова для этой библиотеки, выберем графы, чья сложность на любом запросе x не превышает $3R(I_2, \mathcal{F}_2, x)$, а среди них — граф наименьшего объема. Затем среди всех полученных графов выберем граф наибольшего объема; его объем обозначим через $Q_{fast}(p, n)$. Если для какой-либо библиотеки сложность каждого

графа, решающего рассматриваемую задачу, будет превышать $3R(I_2, \mathcal{F}_2, x)$ на некотором запросе x , то положим $Q_{fast}(p, n) = \infty$.

Теорема 5. *Для любых натуральных $n \geq 2$, $k \geq 3$ и $p < (k - 1)^{n-1}$ выполнено $p(2n - \lfloor \log_{k-1} p \rfloor) - 1 \leq Q_2(p, n) \leq Q_{fast}(p, n) \leq (2k + 1)pn$.*

В главе 3, как и в главе 1, рассматривается задача поиска подслова в множестве слов при условии, что при поиске можно использовать лишь функции из определенного множества, но, в отличие от главы 1, предполагается, что слово длины s над алфавитом N_k представлено набором длины s , а длина слова подается алгоритму, осуществляющему поиск, вместе с самим словом. Приведены нижняя оценка времени работы алгоритма, осуществляющего такой поиск, и верхние оценки времени работы алгоритма при различных ограничениях на объем памяти алгоритма.

Будем рассматривать задачу информационного поиска I_3 , состоящую в перечислении для произвольного запроса x (слова длины не более n над алфавитом N_k) всех слов из некоторой библиотеки, содержащих x в качестве подслова, при условии, что сложность вычисления длины запроса равна 1. Назовем эту задачу задачей поиска подслова с известной длиной.

Понятие информационного графа вводится так же, как и для главы 1. Для построения информационных графов, решающих задачу поиска подслова с известной длиной, будем использовать функции из базового множества \mathcal{F}_3 . Это множество состоит из функции $l(x)$, множества функций \mathcal{F}_1 , описанного в главе 1, а также множества функций $g'_i(x)$, принимающих значение 1, если длина слова x больше a , и 2, если длина x меньше либо равна i . Сложность функции $l(x)$ и каждой функции $g'_i(x)$ полагается равной 1.

Сложностью информационного графа будем называть максимальную по всем запросам сложность обработки запроса этим графом.

Для некоторого числа q среди всех информационных графов объема не более q над базовым множеством \mathcal{F}_3 , решающих задачу поиска вхождений подслова с известной длиной, выберем граф, чья сложность минимальна, и обозначим его сложность через $T(I_3, \mathcal{F}_3, q)$. Если объем каждого графа, решающего рассматриваемую задачу, будет превышать q , то положим $T(I_3, \mathcal{F}_3, q) = \infty$. На-

зовем величину $T(I_3, \mathcal{F}_3, q)$ сложностью задачи I_3 при базовом множестве \mathcal{F}_3 и максимальном объеме q .

Обозначим через $d(I_3)$ наибольшее по всем запросам $x \in X$ количество слов из библиотеки, удовлетворяющих запросу.

Теорема 6. Пусть I_3 — задача поиска под слова с известной длиной, ϵ — произвольное положительное число, тогда для сложности $T(I_3, \mathcal{F}_3, q)$ задачи I_3 при базовом множестве \mathcal{F}_3 и максимальном объеме q справедливы оценки:

$$n + t \cdot (d(I_3) - 1) \leq T(I_3, \mathcal{F}_3, q);$$

$$T(I_3, \mathcal{F}_3, q) \leq 2n - 1 + t \cdot (d(I_3) - 1), \text{ если } q \geq (1 + \epsilon)pn^2(k + 3)/2;$$

$$T(I_3, \mathcal{F}_3, q) \leq n + 1 + t \cdot (d(I_3) - 1), \text{ если } q \geq (1 + \epsilon)pn^3k/6$$

при $n \rightarrow \infty, p \rightarrow \infty$.

Глава 4 посвящена построению модели юридического документа (закона, нормативно-правового акта) по тексту этого документа. Модель документа — это формальное представление совокупности схем вычисления значений объектов в соответствии с текстом документа. Построение модели документа осуществляется поэтапно.

Первый этап заключается в синтаксическом анализе текста, позволяющем по тексту каждого предложения получить его синтаксический граф. Вершинами этого графа являются слова, а ребрами — синтаксические отношения между словами. Граф, получившийся в результате синтаксического анализа, должен быть связным. В главе описан метод построения синтаксического графа.

На втором этапе происходит отождествление сущностей из различных предложений. В результате одну и ту же сущность во всех построенных на следующем этапе формулах будет выражать одна и та же переменная.

На третьем этапе по каждому предложению исходного текста и соответствующему синтаксическому графу строится формула логики предикатов. В результате связи между сущностями, о которых идет речь в предложении, оказываются выраженными с помощью математических операций.

На четвертом этапе по совокупности всех формул для каждого объекта строится модель вычисления этого объекта. Каждая такая модель фактически

является схемой вычисления значения объекта в соответствии с необходимыми и достаточными для этого данными. Совокупность всех таких моделей и будет представлять собой модель закона.

Для каждого этапа решения задачи приведены приемы, с помощью которых этот этап осуществляется. Приведенных приемов достаточно, чтобы построить модель положения ПБУ 6/01[28].

Приемы синтаксического анализа вынесены в приложение 1. Приложение 2 содержит пример синтаксического разбора предложения с помощью приемов из приложения 1.

Благодарности

Автор выражает искреннюю благодарность заведующему кафедрой д.ф.-м.н., профессору, академику В.Б. Кудрявцеву и всему коллективу кафедры за опыт и знания, полученные во время обучения. Особую признательность хочется высказать научному руководителю д.ф.-м.н., профессору Э.Э. Гасанову за постановку задачи и научное руководство.

Глава 1

Поиск подслова в множестве слов

В первой главе рассматривается задача поиска подслова в множестве слов. Описанные в данной главе результаты опубликованы в [29, 30].

§ 1 Обзор литературы

Широко известна задача, несколько отличающаяся от задачи, рассматриваемой в настоящей главе — задача поиска вхождений некоторого слова («образца») в другое, более длинное слово («текст»).

Алгоритмы, решающие задачу поиска вхождений образца в текст, можно условно разделить на 2 вида: осуществляющие предобработку текста и не осуществляющие ее. Характеристики алгоритмов этих двух видов существенно различаются.

Очевидный («наивный») алгоритм, решающий задачу без всякой предобработки, состоит в следующем. Начало образца помещается точно под началом текста, так что первая буква образца оказывается под первой буквой текста, вторая — под второй буквой текста, и т.д. Затем каждая буква образца, начиная с первой, сравнивается с той буквой текста, под которой она находится. Сравнение прекращается, если какая-либо буква текста отличается от соответствующей буквы образца, или если последняя буква образца совпадает с находящейся над ней буквой текста (т.е. образец является подсловом текста). В обоих случаях образец сдвигается на одну букву вправо относительно текста: первая буква образца оказывается под второй буквой текста. После этого каждая буква образца снова сравнивается с находящейся над ней буквой текста. Алгоритм прекращает работу, когда при очередном сдвиге на одну букву вправо под последней буквой текста оказывается предпоследняя буква образца.

В худшем случае для решения задачи с помощью наивного алгоритма потребуется время, пропорциональное произведению длины образца на разность длин текста и образца, к которой прибавили 1. Такая оценка может быть достигнута, например, когда как образец и текст представляют собой одну и ту же букву, повторенную много раз.

Алгоритм Кнута – Морриса – Пратта [2] (1977) также не производит преобработку текста, однако осуществляет поиск за время, в худшем случае пропорциональное сумме длин текста и образца. В начале работы алгоритма за время, пропорциональное длине образца, строится вспомогательная таблица, число элементов в которой равно длине образца. С помощью этой таблицы осуществляется модификация наивного алгоритма поиска: во многих случаях образец сдвигается относительно текста не на одну, а сразу на несколько позиций, и проверка после сдвига может начинаться не с первой буквы образца. В результате количество сравнений символов текста и образца не превышает удвоенную длину текста.

Другой известный алгоритм — Бойера – Мура [3] (1977) — в худшем случае имеет тот же порядок времени работы, что и наивный алгоритм, однако в среднем требует проверки числа символов, меньшего суммы длин текста и образца. Одно из отличительных черт этого алгоритма — проверка на совпадение образца с той частью текста, под которой он находится, происходит справа налево (от последнего символа образца к первому). Вот одно из преимуществ такого способа проверки: в случае, если над символом образца находится буква текста, которая в образце отсутствует, образец можно сдвинуть вправо так, чтобы его первая буква оказалась правее отсутствующей в образце буквы текста. Таким образом, некоторые буквы текста могут оказаться непроверенными, и это не приведет к ошибке.

Существует несколько известных модификаций алгоритма Бойера – Мура. Модификация, предложенная Галилем [4] в 1979 году, позволяет достичь времени работы, не превышающего (по порядку) длины образца. Модификация, предложенная Хорспулом [5] в 1980 году, упрощает алгоритм Бойера – Мура и при этом увеличивает его среднюю скорость, но уменьшает его скорость в худшем случае; порядок времени работы алгоритма Бойера – Мура – Хорспула,

однако, и в среднем, и в худшем случае такой же, как и у алгоритма Бойера – Мура.

Для данной работы, тем не менее, наибольший интерес представляют алгоритмы, осуществляющие предобработку текста. Эти алгоритмы удобно применять, когда происходит поиск разных образцов в одном и том же тексте (и поиск в словаре — именно этот случай). Время предобработки текста не учитывается при измерении времени поиска, так как предобработка осуществляется не при каждом поиске, а лишь один раз. В самых быстрых алгоритмах, осуществляющих предобработку текста, поиск происходит за время, в худшем случае пропорциональное сумме длины образца и количества вхождений образца в текст. К таким алгоритмам относятся суффиксные деревья и суффиксные массивы со словарями рангов.

П. Вейнер [6] предложил идею суффиксного дерева в 1973 году. Суффиксное дерево позволяло осуществлять поиск вхождений подслоа в слово, используя $O(n \log n)$ битов памяти, где n — длина текста. В последующие годы суффиксное дерево неоднократно улучшалось, главным образом, с целью уменьшения требуемой памяти. Суффиксный массив, предложенный Манбером и Майерсом [7] в 1993 году, позволял осуществлять поиск вхождений подслоа в слово и при этом использовал на практике в 3-5 раз меньше памяти, чем суффиксное дерево. Манро, Раман и Рао [8] в 1998 году описали представление суффиксного дерева, требующее $n \log n + O(n)$ битов памяти, где n — длина текста. Представление суффиксного дерева, предложенное Курцем [9] в 1999 году, требует в худшем случае 20 байтов на каждую букву текста, притом Курц учитывает и память, использованную при построении представления. Суффиксный массив со словарями рангов (Мякинен, Наварро [10], 2007) использует $n \log n + kn + o(kn)$ бит памяти, где k — количество символов в алфавите. Одним из наиболее значительных результатов является предложенный Гросси и Виттером [11] в 2000 году сжатый суффиксный массив: алгоритм на его основе использует $O(n \log k)$ битов памяти. Садакане [12] в 2003 году развил идею сжатого суффиксного массива: предложенный им алгоритм требует $n((1 + \epsilon')H_0(T)/\epsilon + 2 \log(1 + H_0(T)) + 3) + o(n)$ битов, где ϵ и ϵ' — любые константы такие, что $0 < \epsilon < 1$ и $\epsilon' > 0$. Функция $H_0(T)$ — это энтропия порядка

0 текста T , определенная как $-\sum_{i=1}^k (n_i/n) \log n_i/n$, где n_i — число вхождений буквы n_i в текст T . Алгоритм Садакане использует исходный текст при построении сжатого суффиксного массива, но не использует непосредственно при поиске. Ферраджина, Манцини, Мякинен и Наварро [13] в 2007 году улучшили результат Садакане для случаев $k = O(\text{polylog}(n))$, снизив объем используемой памяти до $nH_r(T) + o(n)$ битов, где $r \leq \alpha \log_k n$ для произвольной константы $\alpha, 0 < \alpha < 1$, а $H_r(T)$ — энтропия порядка r текста T , определяющаяся как $(1/n) \sum_{w \in \Sigma^r} |w_T| H_0(w_T)$, Σ — рассматриваемый алфавит, w_T — слово, получившееся в результате конкатенации символов, расположенных в тексте T сразу после вхождений подслова w . Надо заметить, что $H_r(T) \leq \log k$, а для некоторых текстов $H_r(T) = o(1)$.

Не все алгоритмы, осуществляющие поиск вхождений образца в текст, можно обобщить на случай, когда нужно найти все вхождения образца в несколько текстов. Тем не менее, для суффиксного дерева это можно сделать: на его основе может быть построено обобщенное суффиксное дерево (см., например, [1]), позволяющее искать образец в массиве текстов за время, пропорциональное сумме длины образца и количества вхождений образца в массив текстов.

В работах [8, 10, 11, 12, 13] используется модель вычислений word RAM [31]. В ней, в частности, предполагается, что каждая ячейка памяти состоит из w битов, причем арифметические и булевы операции над w -битными машинными словами осуществляются за константное время. Предполагается также, что длина текста меньше 2^w .

Интерес представляют не только сами алгоритмы поиска образца в тексте, но и нижние оценки памяти для алгоритмов, осуществляющие поиск за время, в худшем случае пропорциональное сумме длины образца и количества вхождений образца в текст. Э.Д. Демейн и А. Лопес-Ортис [14] в 2003 году показали, что такой алгоритм требует памяти, пропорциональной длине текста. В использованной ими модели предполагалось, что алгоритм имеет доступ к образцу, тексту и индексу — структуре данных, построенной для быстрого поиска образца в тексте. При вычислении времени поиска учитывались лишь запросы алгоритма к тексту, причем предполагалось, что целью каждого запро-

са алгоритма к тексту было получение одного бита текста. Галь и Мильтерсен [15] рассмотрели (в той же модели) случай, когда длина образца находится в промежутке между $2 \log_2 n + 5$ и $5 \log_2 n$, где n — длина текста, и получили, что в этом случае алгоритм требует не менее $n/(800t \log n) - 1$ памяти, где t — количество запросов алгоритма к тексту. Голынский [16] в 2009 году рассматривал модель, использующую ячейки памяти длиной w битов. Голынский показал, что для $w = \log n$ и образцов, имеющих длину $\log n / \log k$, алгоритм потребует $n \log k / \log n$ ячеек памяти. Тем не менее, общее число результатов, посвященных нижним оценкам объема памяти для задачи поиска под слова в слове, значительно меньше числа результатов, посвященных верхним оценкам объема памяти для той же задачи.

§ 2 Определения

Введем определения, необходимые для главы 1.

Обозначим $N_k = \{1, 2, \dots, k\}$. Пусть $W_{k,n} = \cup_{s=1}^n N_k^s \{0\}^{n-s}$ — множество слов длины не более n над алфавитом N_k . Слово длины s представлено набором длины n , первые s элементов которого — из множества N_k , и если $s < n$, то оставшиеся $n - s$ элементов — нули.

Для каждого слова $w \in W_{k,n}$ его длину будем обозначать через $l(w)$; i -й буквой слова $w \in W_{k,n}$, где $i \in \{1, \dots, n\}$, назовем i -й элемент соответствующего слову w набора. В частности, если $i > l(w)$, то i -я буква слова w равна 0. Будем обозначать i -ю букву слова w через $w[i]$. Через $w[a \dots b]$, где $a, b \in \mathbb{N}$, $1 \leq a \leq b \leq l(w)$, будем обозначать такое слово $v \in W_{k,n}$, что $l(v) = b - a + 1$ и $v[i] = w[a + i - 1]$ для всех натуральных i , не превышающих $l(v)$. Будем называть слово $w[a \dots b]$ подсловом слова w , начинающимся с его a -й буквы и заканчивающимся его b -й буквой. Будем говорить, что слова $w \in W_{k,n}$ и $v \in W_{k,n}$ равны или совпадают (и писать $w = v$), если $l(v) = l(w)$ и $v[1] = w[1], v[2] = w[2], \dots, v[l(v)] = w[l(w)]$ (очевидно, это условие эквивалентно условию $v[1] = w[1], v[2] = w[2], \dots, v[n] = w[n]$). Для слова w его подслово $w[1 \dots q]$, $q \in \mathbb{N}$, $q \leq l(w)$, будем называть началом длины q слова w , а подслово $w[l(w) - q + 1 \dots l(w)]$, $q \in \mathbb{N}$, $q \leq l(w)$ — концом длины q слова w . Скажем, что слово w лексикографически следует за словом v , а слово v лексикографически

предшествует слову w , если существует такое натуральное число i , не превышающее n , что $v[i] < w[i]$ и для всякого натурального числа j , меньшего i , j -я буква слова v равна j -й букве слова w . Будем записывать это следующим образом: $w > v$ (либо $v < w$).

Рассмотрим $X = W_{k,n}$ — множество запросов, то есть слов длины не более n над алфавитом N_k . Рассмотрим также $Y = N_k^n$ — множество записей. Каждая запись является словом длины n над алфавитом N_k . Введем бинарное отношение ρ_1 , которое позволит устанавливать, когда запись $y \in Y$ удовлетворяет запросу $x \in X$ (отношение поиска):

$$x\rho_1y \Leftrightarrow \text{существует } i \in \mathbb{N} : x = y[i \dots l(x) + i - 1], \quad x \in X, \quad y \in Y.$$

Будем рассматривать задачу информационного поиска (ЗИП) $I_1 = \langle X, V, \rho_1 \rangle$, где $V = \{v_1, v_2, \dots, v_p\}$, $V \subseteq Y$. Содержательно будем считать, что задача $I_1 = \langle X, V, \rho_1 \rangle$ состоит в перечислении для произвольно взятого запроса $x \in X$ всех тех и только тех записей y из V , которые содержат x в качестве подслова, то есть для которых выполнено $x\rho_1y$. Назовем эту задачу задачей поиска подслова. Множество V в дальнейшем будем называть библиотекой. Так как X и ρ_1 фиксированы, задача поиска подслова полностью определяется библиотекой.

Введем понятие информационного графа (ИГ) в соответствии с [17, 18]. В формальном определении понятия ИГ используется множество запросов X , множество записей Y , а также множество F предикатов, заданных на множестве X (предикаты — это функции, которые могут принимать только два значения: 0 или 1), и множество G переключателей, заданных на множестве X (переключатели — это функции, область значений которых является начальным отрезком натурального ряда). Пару $\mathcal{F} = \langle F, G \rangle$ будем называть базовым множеством. Рассмотрим произвольный ориентированный граф. Выделим в нем одну вершину. Назовем ее корнем. Выделим в графе какие-либо другие вершины. Назовем их листьями. Сопоставим каждому листу некоторую запись из множества Y . Это соответствие назовем нагрузкой листьев. Выделим в графе некоторые вершины (это могут быть в том числе корень и листья) и назовем их точками переключения. Если β — вершина графа, то через ψ_β обозначим полустепень исхода вершины β (то есть число исходящих из этой

вершины ребер). Каждой точке переключения β сопоставим некий символ из G . Это соответствие назовем нагрузкой точек переключения. Для каждой точки переключения β ребрам, из нее исходящим, поставим во взаимно однозначное соответствие числа из множества $\{1, 2, \dots, \psi_\beta\}$. Эти ребра назовем переключательными, а это соответствие — нагрузкой переключательных ребер. Ребра, не являющиеся переключательными, назовем предикатными. Каждому предикатному ребру графа сопоставим некоторый символ из множества F . Это соответствие назовем нагрузкой предикатных ребер. Полученную нагруженную сеть назовем информационным графом над базовым множеством $\mathcal{F} = \langle F, G \rangle$.

Определим функционирование ИГ. Скажем, что предикатное ребро проводит запрос $x \in X$, если предикат, приписанный этому ребру, принимает значение 1 на запросе x ; переключательное ребро, которому приписан номер r , проводит запрос $x \in X$, если переключатель, приписанный началу этого ребра, принимает значение r на запросе x ; ориентированная цепь ребер проводит запрос $x \in X$, если каждое ребро цепи проводит запрос x ; запрос $x \in X$ проходит в вершину β ИГ, если существует ориентированная цепь, ведущая из корня в вершину β , которая проводит запрос x ; запись y , приписанная листу α , попадает (включается) в ответ ИГ на запрос $x \in X$, если запрос x проходит в лист α . Ответом ИГ U на запрос x назовем множество записей, попавших в ответ ИГ на запрос x , и обозначим его $J_U(x)$. Эту функцию $J_U(x)$ будем считать результатом функционирования ИГ U и называть функцией ответа ИГ U . Тем самым понятие ИГ полностью определено.

Назовем каждую цепь, по которой запрос x проходит из корня в какой-либо лист, существенной цепью ИГ, соответствующей запросу x .

Скажем, что ИГ U решает ЗИП $I_1 = \{X, V, \rho_1\}$, если для любого запроса $x \in X$ ответ на этот запрос содержит все те и только те записи из V , которые удовлетворяют запросу x , то есть $J_U(x) = \{y \in V : x\rho_1y\}$.

Определим понятие сложности информационного графа на запросе. Сделаем это в более общем виде, чем в [17, 18]: будем считать, что время вычисления функции f от запроса x зависит не только от f , но и от x . Обозначим через $Pr(U, x)$ множество всех предикатных вершин ИГ U , в которые проходит запрос x , а через $Sw(U, x)$ — множество всех переключательных вершин ИГ U , в

которые проходит запрос x . Обозначим через $E(\beta)$ множество всех ребер, выходящих из вершины β . Тогда определим сложность вычисления ИГ U на запросе x следующим образом:

$$T(U, x) = \sum_{\beta \in Sw(U, x)} t(g_\beta, x) + \sum_{\beta \in Pr(U, x)} \sum_{e \in E(\beta)} t(f_e, x),$$

где g_β — переключатель, сопоставленный вершине β , f_e — предикат, сопоставленный ребру e , $t(f, x)$ — время вычисления функции f от запроса x . $T(U, x)$ характеризует время обработки запроса x .

Объемом $Q(U)$ ИГ U назовем число ребер в ИГ U . $Q(U)$ соответствует объему памяти, используемой информационным графом.

В качестве базового множества \mathcal{F}_1 будем рассматривать $\langle F, G_1 \rangle$, $F = \{f(x)\}$, $G_1 = \{g_i(x), i \in \mathbb{N}\}$, где $f(x) \equiv 1$, $g_i(x) = x[i]$, если $x[i] \neq 0$, и $g_i(x) = k + 1$, если $x[i] = 0$. С помощью предиката, тождественно равного 1, моделируется перечисление ответа на запрос. Вычисление этого предиката соответствует переходу по ссылке (например, к слову из ответа), в то время как вычисление любого переключателя $g_i(x)$ включает в себя выделение i -й буквы слова x и переход по ссылке. По этой причине сложность вычисления предиката, тождественно равного 1, считается положительной, но меньшей, чем сложность вычисления переключателя $g_i(x)$. Будем считать, что сложность вычисления любого переключателя $g_i(x)$ равна 1, а сложность вычисления предиката $f(x)$ равна t , $0 < t < 1$.

Введем следующие обозначения. $\mathcal{U}(I_1, \mathcal{F}_1)$ — множество всех ИГ над базовым множеством \mathcal{F}_1 , решающих ЗИП $I_1 = \langle X, V, \rho_1 \rangle$; $d(I_1, x) = |\{v \in V, x\rho_1 v\}|$; $e(x) = \min(l(x) + 1, n)$; $R(I_1, \mathcal{F}_1, x) = e(x) + t(d(I_1, x) - 1)$.

Пусть \mathcal{U}_{I_1} множество таких графов $U \in \mathcal{U}(I_1, \mathcal{F}_1)$, что для каждого из них при любых x из X выполняется условие $T(U, x) \leq R(I_1, \mathcal{F}_1, x)$. Функцию, равную $\max_{I: I = \langle X, V, \rho_1 \rangle, |V| = p} \min_{U \in \mathcal{U}_{I_1}} Q(U)$, обозначим через $Q_1(p, n)$.

Скажем, что функция $f(p, n) : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}_{>0}$ асимптотически не превосходит функцию $g(p, n) : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}_{>0}$ при $p \rightarrow \infty$, $n \rightarrow \infty$ и обозначим $f(p, n) \lesssim g(p, n)$, если существует такая функция $\alpha(p, n) : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$, $\alpha(p, n) \rightarrow 0$ при $n \rightarrow \infty$, $p \rightarrow \infty$ и числа $p_0 \in \mathbb{N}$, $n_0 \in \mathbb{N}$, что $f(p, n) \leq (1 + \alpha(p, n)) \cdot g(p, n)$, когда $p > p_0$, $n > n_0$.

§ 3 Нижняя оценка сложности поиска подслова в множестве слов

В данном параграфе получена нижняя оценка сложности поиска.

Лемма 1. Пусть $k \geq 3$ и в библиотеке $V \subseteq N_k^n$ из p слов, где $p \leq (k-1)^n$, каждая буква каждого слова отлична от k . Тогда для любого ИГ $U \in \mathcal{U}(I_1, \mathcal{F}_1)$, где $I_1 = \langle X, V, \rho_1 \rangle$, и для каждого запроса $x \in X$, которому удовлетворяет хотя бы одно слово из библиотеки, выполнено неравенство $T(U, x) \geq R(I_1, \mathcal{F}_1, x)$.

Доказательство. Рассмотрим произвольную библиотеку V , удовлетворяющую условиям леммы, и произвольный ИГ $U \in \mathcal{U}(I_1, \mathcal{F}_1)$, где $I_1 = \langle X, V, \rho_1 \rangle$. Покажем сначала, что если $d(I_1, x) \geq 1$, то $T(U, x) \geq l(x)$. Рассмотрим какую-либо цепь C , по которой запрос проходит из корня в лист с записью v , удовлетворяющей запросу (назовем этот лист γ), причем ни одна вершина этой цепи, кроме γ , не является листом. Пусть существует такое $i \in \mathbb{N}$, $1 \leq i \leq l(x)$, что ни одной из вершин C , исключая γ , не сопоставлен переключатель $g_i(x)$. Тогда заменим $x[i]$ на k . Получившийся запрос также будет проходить в лист с записью v . Мы имеем противоречие с тем, что ни в одном слове из V нет буквы k , а следовательно, и любого подслова, содержащего букву k . Получим, что для любого $i \in \mathbb{N}$, $1 \leq i \leq l(x)$ найдется вершина цепи, не совпадающая с γ и такая, что ей сопоставлен g_i . Значит, $T(U, x) \geq l(x)$.

Далее, пусть $l(x) < n$. Тогда по крайней мере одной из вершин C , не считая γ , должен быть сопоставлен переключатель $g_{l(x)+1}$. Если это не так, заменим $x[l(x)+1]$ на k и снова придем к противоречию. Следовательно, $T(U, x) \geq l(x) + 1$.

Пусть $d(I_1, x) > 1$. Это возможно только тогда, когда $l(x) < n$. Для каждого отличного от γ листа, в который проходит запрос x , выберем ребро, по которому x входит в этот лист. Так как никакому листу не может быть сопоставлено более одной записи, все эти ребра разные, причем если некоторые из этих ребер исходят из одной вершины, то это предикатные ребра. Кроме того, ни одно из этих ребер не принадлежит C . Значит, для прохода запроса по таким ребрам будет вычислено не менее $d(I_1, x) - 1$ функций. Кроме того, при

обработке запроса будет вычислено не менее $l(x) + 1$ указанных ранее переключателей. Значит, $T(U, x) \geq l(x) + 1 + t \cdot (d(I_1, x) - 1)$. Итак, мы показали, что если $d(I_1, x) \geq 1$, то $T(U, x) \geq R(I_1, \mathcal{F}_1, x)$. □

Из леммы 1 непосредственно вытекает следующая теорема.

Теорема 1. *При $k \geq 3$ среди всех библиотек $V \subseteq N_k^n$ из p слов, где $p \leq (k-1)^n$, найдется такая библиотека V , что для задачи I_1 для любых $U \in \mathcal{U}(I_1, \mathcal{F}_1)$ и $x \in X$ выполняется следующее условие: если $d(I_1, x) \geq 1$, то $T(U, x) \geq R(I_1, \mathcal{F}_1, x)$.*

§ 4 Информационный граф, осуществляющий поиск с минимальной временной сложностью

Покажем, что для любой библиотеки V можно построить такой ИГ $U_1(I_1, \mathcal{F}_1)$, решающий задачу $I_1 = \langle X, V, \rho_1 \rangle$ над базовым множеством \mathcal{F}_1 , что для всех $x \in X$ выполняется $T(U_1, x) \leq R(I_1, \mathcal{F}_1, x)$. Далее часто вместо $U_1(I_1, \mathcal{F}_1)$ будем писать U_1 , предполагая, что I_1 и \mathcal{F}_1 зафиксированы.

Пусть $V = \{v_1, \dots, v_p\}$. Через V' обозначим множество всех подслов всех слов из V . Будем строить информационный граф U_1 индукцией по длине s слов из V' .

Базис индукции: $s = 1$. Выберем произвольную вершину, объявим ее корнем информационного графа. Выберем еще $k + 1$ вершин, и в каждую из них направим из корня ребро. Пронумеруем эти ребра числами от 1 до $k + 1$. Сопоставим корню переключатель $g_1(x)$. Рассмотрим все слова длины 1 из V' . Пусть это слова $v_1^1, v_2^1, \dots, v_{p_1}^1$. Заметим, что так как $v_i^1 \neq v_j^1, i \neq j$, то $v_i^1[1] \neq v_j^1[1], i \neq j$. Для каждого $i \in \{1, 2, \dots, p_1\}$ сделаем следующее. Найдем вершину, в которую из корня ведет ребро под номером $v_i^1[1]$, назовем ее $\beta_{v_i^1}$. Выберем $k + 1$ новых вершин и в каждую из них направим из $\beta_{v_i^1}$ ребро. Пронумеруем эти ребра числами от 1 до $k + 1$. Вершину, в которую ведет $(k + 1)$ -е исходящее из $\beta_{v_i^1}$ ребро, назовем $\gamma_{v_i^1}$. Сопоставим вершине $\beta_{v_i^1}$ переключатель $g_2(x)$. Рассмотрим теперь все слова из V , в которых v_i^1 содержится в качестве подслова (хотя бы одно такое слово найдется по построению V'). Пусть это слова $v_{j_1, i, 1}, \dots, v_{j_1, i, p_1, i}$,

где $j_{1,i,a} < j_{1,i,b}$, когда $a < b$. Слово $v_{j_{1,i,1}}$ сопоставим вершине $\gamma_{v_i^1}$. Каждое из оставшихся слов из V , содержащих v_i^1 в качестве подслова, сопоставим новой вершине. Для каждого из слов $v_{j_{1,i,a}}$, $a \in \{1, \dots, p_{1,i} - 1\}$, выпустим из вершины, которой это слово приписано, в вершину, которой приписано слово $v_{j_{1,i,a+1}}$, ребро с предикатом $f(x)$.

Шаг индукции: пусть $2 \leq s \leq n$, и мы уже рассмотрели все слова из V' , имеющие длину $1, 2, \dots, s - 1$, причем для каждого такого слова v' в графе найдется вершина $\beta_{v'}$. Рассмотрим все слова длины s из V' . Пусть это слова $v_1^s, v_2^s, \dots, v_{p_s}^s$. Для каждого $i \in \{1, 2, \dots, p_s\}$ сделаем следующее. Для каждого v_i^s его подслово $v_i^s[1 \dots s - 1]$, очевидно, принадлежит V' , значит, по предположению индукции в ИГ есть вершина $\beta_{v_i^s[1 \dots s - 1]}$. Заметим, что так как $v_i^s \neq v_j^s$, когда $i \neq j$, то либо $v_i^s[1 \dots s - 1] \neq v_j^s[1 \dots s - 1]$, либо $v_i^s[s] \neq v_j^s[s]$, когда $i \neq j$. Возможны 2 случая:

1) $s < n$. Рассмотрим вершину, в которую из $\beta_{v_i^s[1 \dots s - 1]}$ ведет ребро под номером $v_i^s[s]$. Назовем ее $\beta_{v_i^s}$. Выберем $k + 1$ новых вершин, и в каждую из них направим из $\beta_{v_i^s}$ ребро. Пронумеруем эти ребра числами от 1 до $k + 1$. Вершину, в которую ведет $(k + 1)$ -е исходящее из $\beta_{v_i^s}$ ребро, назовем $\gamma_{v_i^s}$. Сопоставим вершине $\beta_{v_i^s}$ переключатель $g_{s+1}(x)$.

2) $s = n$. Рассмотрим вершину, в которую из $\beta_{v_i^s[1 \dots s - 1]}$ ведет ребро под номером $v_i^s[s]$. Назовем ее $\gamma_{v_i^s}$.

Рассмотрим теперь все слова из V , в которых v_i^s содержится в качестве подслова (хотя бы одно такое слово найдется по построению V'). Пусть это слова $v_{j_{s,i,1}}, \dots, v_{j_{s,i,p_{s,i}}}$, где $j_{s,i,a} < j_{s,i,b}$, когда $a < b$. Слово $v_{j_{s,i,1}}$ сопоставим вершине $\gamma_{v_i^s}$. Каждое из оставшихся слов из V , содержащих v_i^s в качестве подслова, сопоставим новой вершине. Для каждого из слов $v_{j_{s,i,a}}$, где $a \in \{1, \dots, p_{s,i} - 1\}$, выпустим из вершины, которой это слово приписано, в вершину, которой приписано слово $v_{j_{s,i,a+1}}$, ребро с предикатом $f(x)$. Шаг индукции на этом закончен.

Тем самым построение ИГ U_1 завершено.

Лемма 2. Информационный граф $U_1(I_1, \mathcal{F}_1)$ решает задачу $I_1 = \langle X, V, \rho_1 \rangle$ над базовым множеством \mathcal{F}_1 .

Доказательство. Заметим сначала, что так как из каждой предикатной вершины в графе выходит не более одного ребра, то из каждой вершины, куда

прошел запрос, он может перейти не более чем по одному ребру. Кроме того, граф является деревом (поскольку при построении графа новые ребра всякий раз направлялись в новые вершины).

Будем доказывать лемму индукцией по длине запроса.

Базис индукции. Рассмотрим какой-либо запрос x длины 1. Пусть найдутся такие $i \in \{1, \dots, p\}$ и $j \in \{1, \dots, n\}$, что $x = v_i[j \dots j]$. Тогда $x \in V'$, то есть существует $q \in \{1, \dots, p_1\}$ такое, что $x = v_q^1$ и в графе найдется вершина β_x , притом запрос x проходит из корня в эту вершину после вычисления переключателя $g_1(x)$. Затем запрос проходит в вершину γ_x ($g_2(x) = k + 1$, так как $x[2] = 0$). Этой вершине сопоставлено одно из слов, удовлетворяющих запросу x . Далее на каждом шаге запрос переходит по предикатному ребру с предикатом $f(x)$, тождественно равном единице, в вершину, которой приписано следующее слово, удовлетворяющее запросу, пока не попадет в вершину со словом $v_{j_1, q, p_1, q}$. Из этой вершины не выходит ни одно ребро. Таким образом, в ответ на запрос попадут все записи, удовлетворяющие x , и никакие другие. Пусть теперь для всех $i \in \{1, \dots, p\}$ и $j \in \{1, \dots, n\}$ выполнено условие $x \neq v_i[j \dots j]$. Тогда в графе нет вершины β_x , и вершине, в которую запрос пройдет из корня, не приписано ни одно слово, притом из этой вершины не выходит ни одно ребро. Значит, ни одна запись не попадет в ответ на запрос x . Итак, действительно, информационный граф решает задачу для запросов длины 1.

Предположим, что ИГ решает задачу для всех запросов длины не более $s - 1$, где $s \in \{2, \dots, n\}$, причем: а) каждый такой запрос w пройдет в вершину β_w тогда и только тогда, когда в библиотеке найдется хотя бы одно слово, удовлетворяющее этому запросу; б) если запрос w проходит в вершину β_w , то в вершинах цепи, по которой он туда проходит, не считая самой β_w , последовательно вычисляются переключатели $g_1, \dots, g_{l(w)}$, и только они. Мы уже показали, что для слов длины 1 условия а) и б) выполнены. Рассмотрим произвольное слово x длины s . Пусть существуют такие $i \in \{1, \dots, p\}$ и $j \in \{1, \dots, n\}$, что $x = v_i[j \dots j + s - 1]$. Тогда $x \in V'$ и найдется такое q , что $x = v_q^s$. Для слова $x' = x[1 \dots s - 1]$ в графе найдется вершина $\beta_{x'}$. По предположению индукции в вершинах (за исключением $\beta_{x'}$) пути, по которому запрос x' проходит в $\beta_{x'}$, последовательно вычисляются переключатели g_1, \dots, g_{s-1} , и только они. Значит,

запрос x также проходит в вершину $\beta_{x'}$. В этой вершине вычисляется переключатель g_s . Если $s = n$, то запрос x проходит в вершину γ_x ; если же $s < n$, то x проходит в вершину β_x , а уже из нее, после вычисления переключателя g_{s+1} , который на запросе x примет значение $k + 1$, запрос пройдет в γ_x . Вершине γ_x сопоставлено одно из слов, удовлетворяющих запросу x . Далее на каждом шаге запрос переходит по предикатному ребру с предикатом f , тождественно равном единице, в вершину, которой приписано следующее слово, удовлетворяющее запросу, пока не попадет в вершину со словом $v_{j_s, q, p_s, q}$. Из этой вершины не выходит ни одно ребро. Таким образом, в ответ на запрос попадут все записи, удовлетворяющие x , и никакие другие. Пусть теперь для всех $i \in \{1, \dots, p\}$ и $j \in \{1, \dots, n\}$ выполнено условие $x \neq v_i[j \dots j + s - 1]$. Возможны 2 случая:

1) Для всех $v' \in V'$ выполнено $x[1 \dots 1] \neq v'$. Тогда вершине, в которую запрос пройдет из корня, не приписано ни одно слово, притом из этой вершины не выходит ни одно ребро. Значит, ни одна запись не попадет в ответ на запрос x .

2) Выполнено условие $x[1 \dots 1] \in V'$. Рассмотрим $l = \max_{x[1 \dots l'] \in V'} l'$. Очевидно, $1 \leq l \leq s - 1$. Тогда в графе есть вершина $\beta_{x[1 \dots l]}$, причем запрос x проходит из корня в эту вершину. Из $\beta_{x[1 \dots l]}$ запрос пройдет в вершину, которой не приписано ни одно слово, притом такую, что из нее не выходит ни одно ребро. Значит, ни одна запись не попадет в ответ на запрос x .

Итак, действительно, построенный информационный граф U_1 решает задачу I_1 . □

Лемма 3. Для любого запроса $x \in X$ выполнено $T(U_1(I_1, \mathcal{F}_1), x) \leq R(I_1, \mathcal{F}_1, x)$.

Доказательство. Оценим сложность U_1 на произвольном запросе x . Возможны следующие случаи.

1) Ни одно слово из V не содержит x в качестве подслова. Тогда найдется такое $s \in \{1, \dots, l(x)\}$, что при обработке запроса будут последовательно вычислены переключатели g_1, g_2, \dots, g_s , причем в результате вычисления g_s запрос пройдет в вершину, которой не приписано ни одно слово и из которой не выходит ни одно ребро. Значит, сложность обработки такого запроса x не превысит $l(x)$.

2) $l(x) = n$ и в библиотеке V есть слово, содержащее x в качестве под- слова (значит, $x \in V'$, так как длина каждой записи из библиотеки равна n). Тогда при обработке запроса будут последовательно вычислены переключатели g_1, g_2, \dots, g_n , причем в результате вычисления g_n запрос пройдет в вершину γ_x , которой сопоставлено слово x . Из этой вершины не выходит ни одно ребро, так как других слов, кроме x , удовлетворяющих запросу, в библиотеке быть не может. Следовательно, сложность обработки такого запроса x равна $l(x)$.

3) $l(x) < n$ и в библиотеке V есть слово, содержащее x в качестве под- слова. Тогда при обработке запроса будут последовательно вычислены переключатели $g_1, g_2, \dots, g_{l(x)+1}$, причем в результате вычисления $g_{l(x)+1}$ запрос пройдет в вершину γ_x , которой сопоставлено какое-либо слово из V , удовлетворяющее x . Далее запрос последовательно (по ребрам с предикатами $f(x)$) проходит в вер- шину, которым сопоставлены все остальные слова, удовлетворяющие запросу. Значит, сложность обработки такого запроса x равна $l(x) + 1 + t \cdot (d(I_1, x) - 1)$.

Итак, мы показали, что $U_1 \in \mathcal{U}(I_1, \mathcal{F}_1)$ и для любого x выполнено $T(U_1, x) \leq R(I_1, \mathcal{F}_1, x)$. Следовательно, $U_1 \in \mathcal{U}_{I_1}$, то есть, $\mathcal{U}_{I_1} \neq \emptyset$. \square

С помощью лемм 2 и 3 докажем следующую теорему.

Теорема 2. *Для любой задачи $I_1 = \langle X, V, \rho_1 \rangle$ множество \mathcal{U}_{I_1} непусто.*

Доказательство. Согласно лемме 2, при любой библиотеке $V \subseteq Y$ ИГ $U_1(I_1, \mathcal{F}_1)$, где $I_1 = \langle X, V, \rho_1 \rangle$, решает задачу I_1 над базовым множеством \mathcal{F}_1 . Согласно лемме 3, для любого запроса $x \in X$ выполнено $T(U_1(I_1, \mathcal{F}_1), x) \leq R(I_1, \mathcal{F}_1, x)$. Значит, $U_1 \in \mathcal{U}_{I_1}$. Теорема 2 доказана. \square

§ 5 Оценка порядка объемной сложности информационного графа, имеющего минимальную временную сложность

Лемма 4. *Если $I_1 = \langle X, V, \rho_1 \rangle$, $|V| = p$, то $Q(U_1(I_1, \mathcal{F}_1)) \leq (k + 2)p(n + 2)(n - 1)/2 + p + k + 1$.*

Доказательство. ИГ U_1 содержит 3 группы ребер.

1) Каждому слову $v' \in V'$ сопоставлена вершина $\gamma_{v'}$, причем в эту вершину проходит только запрос v' . Значит, каждому слову v' можно поставить в соответствие переключательное ребро, ведущее в $\gamma_{v'}$, а также каждое ребро, по которому запрос v' пройдет после попадания в $\gamma_{v'}$. Значит, каждому v' будет сопоставлено столько ребер, сколько слов из V содержат v' в качестве подслова. Суммарно для всех слов из V' таких ребер будет не более общего числа подслов всех слов из библиотеки V , то есть не более $p \cdot (1 + 2 + \dots + n) = pn(n + 1)/2$.

2) Кроме того, каждому слову v' , имеющему длину не более $n - 1$, сопоставлена вершина $\beta_{v'}$. Следовательно, каждому такому v' можно поставить в соответствие все ребра, выходящие из $\beta_{v'}$, за исключением ребер, ведущих в какие-либо вершины $\gamma_{v''}$, то есть не более $k + 1$ ребер. Так как в V' не более $pn(n + 1)/2$ слов, мы получим, что ребер, рассматриваемых в данном пункте, в графе не больше $(k + 1)p(n(n + 1)/2 - 1) = (k + 1)p(n + 2)(n - 1)/2$.

3) Кроме ребер, рассмотренных в пунктах 1) и 2), в графе есть еще только ребра, выходящие из корня. Их $k + 1$.

Итак, мы получили, что $Q(U_1) \leq (k + 2)p(n + 2)(n - 1)/2 + p + k + 1$. \square

Построим такую библиотеку V , что если $U \in \mathcal{U}_{I_1}$, где $I_1 = \langle X, V, \rho_1 \rangle$, то $Q(U) \gtrsim pn^2/9, n \rightarrow \infty, p \rightarrow \infty$.

Пусть $k \geq 3, n \geq 3$. Обозначим $r = \lfloor n/3 \rfloor$.

Рассмотрим все слова $w_1, \dots, w_{(k-1)^r}$ длины r , не содержащие букву k , причем $w_i < w_j$, если $i < j$. Определим каждое слово $v_i, i \in N_{k-1}^r$, следующим образом:

$$\begin{cases} v_i[r + 1 \dots 2r] = w_i \\ v_i[r] = v_i[2r + 1] = 1 \\ v_i[r - j] = (j - 1) \bmod (k - 2) + 2, & \text{если } j \in \mathbb{N}, j \leq r - 1 \\ v_i[2r + j + 1] = (j - 1) \bmod (k - 2) + 2, & \text{если } j \in \mathbb{N}, j \leq n - 2r - 1 \end{cases} \quad (1.1)$$

Пример: пусть $n = 14, k = 5$. Тогда $r = 4, w_5 = 1121, v_5 = 43211121123423$.

Рассмотрим слова $v_{i,a,b} = v_i[a \dots b]$, где $a \in \mathbb{N}, b \in \mathbb{N}, a \leq r, 2r + 1 \leq b \leq n, i \in \{1, \dots, (k - 1)^r\}$.

Лемма 5. $v_{i_1, a_1, b_1} = v_{i_2, a_2, b_2}$ тогда и только тогда, когда $i_1 = i_2, a_1 = a_2, b_1 = b_2$.

Доказательство. Достаточность, очевидно, следует из определения слов $v_{i,a,b}$. Докажем необходимость. Пусть $v_{i_1,a_1,b_1} = v_{i_2,a_2,b_2}$. По определению $v_{i,a,b}$ имеем: $v_{i_1,a_1,b_1}[r - a_1 + 1] = v_{i_1}[r] = 1, v_{i_1,a_1,b_1}[j] \neq 1, j \leq r - a_1$. Аналогично $v_{i_2,a_2,b_2}[r - a_2 + 1] = 1, v_{i_2,a_2,b_2}[j] \neq 1, j \leq r - a_2$. Значит, $r - a_1 + 1 = r - a_2 + 1$, откуда $a_1 = a_2$. Так как $b_1 - a_1 + 1 = l(v_{i_1,a_1,b_1}) = l(v_{i_2,a_2,b_2}) = b_2 - a_2 + 1$, то $b_1 = b_2$. Далее, имеем $v_{i_1,a_1,b_1}[r - a_1 + 2 \dots 2r - a_1 + 1] = v_{i_1}[r + 1 \dots 2r] = w_{i_1}, v_{i_2,a_2,b_2}[r - a_1 + 2 \dots 2r - a_1 + 1] = v_{i_2,a_2,b_2}[r - a_2 + 2 \dots 2r - a_2 + 1] = v_{i_2}[r + 1 \dots 2r] = w_{i_2}$, то есть $w_{i_1} = w_{i_2}$, откуда $i_1 = i_2$. Необходимость доказана. \square

Итак, любые 2 слова вида $v_{i,a,b}$, у которых не совпадает хотя бы одно из значений i, a, b , разные.

Лемма 6. Пусть $k \geq 3$ и в библиотеке $V \subseteq N_k^n$ из p слов, где $p \leq (k - 1)^n$, каждая буква каждого слова отлична от k , пусть также $U \in \mathcal{U}(I_1, \mathcal{F}_1)$, где $I_1 = \langle X, V, \rho_1 \rangle$, и для каждого запроса $x \in X$, которому удовлетворяет хотя бы одно слово из библиотеки, выполнено $T(U, x) = R(I_1, \mathcal{F}_1, x)$. Тогда:

1) каждой из первых $e(x)$ вершин (считая от корня) любой существенной цепи информационного графа, соответствующей x , сопоставлен один из переключателей g_i , где $i \in \{1, \dots, e(x)\}$, причем каждый переключатель g_i сопоставлен ровно одной вершине;

2) первые $e(x) + 1$ вершин (считая от корня) всех существенных цепей информационного графа, соответствующих x , общие;

3) $(e(x) + 1)$ -я и все последующие вершины (считая от корня) любой существенной цепи информационного графа, соответствующей x , являются листьями;

4) каждое выходящее из листа, в который проходит запрос x , ребро любой существенной цепи информационного графа, соответствующей x , является предикатным.

Доказательство. Предположим, что для каждого запроса x , для которого в библиотеке найдется хотя бы одно удовлетворяющее ему слово, выполнено $T(U, x) = R(I_1, \mathcal{F}_1, x)$. Как показано в доказательстве леммы 1, при обработке запроса хотя бы по одному разу должен быть вычислен каждый из переключателей $g_1(x), \dots, g_{e(x)}(x)$. Кроме того, для попадания запроса в каждый

лист должно быть вычислено еще не менее $d(I_1, x) - 1$ функций. Значит, если $T(U, x) = R(I_1, \mathcal{F}_1, x)$, то каждый из переключателей $g_1(x), \dots, g_{e(x)}(x)$ при обработке запроса должен быть вычислен ровно по одному разу, а остальных функций должно быть вычислено ровно $d(I_1, x) - 1$. Так как $t < 1$, эти функции являются предикатами. Соответствующие этим предикатам ребра должны входить в листья. Следовательно, каждый предикат при обработке запроса вычисляется уже после того, как вычислены все переключатели $g_1(x), \dots, g_{e(x)}(x)$. Значит, в любой существенной цепи рассматриваемого ИГ, соответствующей x , все эти переключатели сопоставлены первым $e(x)$ вершинам (считая от корня). Таким образом, п.1 обоснован.

Далее, заметим, что из каждой из первых $e(x)$ вершин, в которые проходит запрос x , данный запрос переходит ровно по одному ребру, так как все эти вершины — переключательные. Значит, первые $e(x) + 1$ вершин (считая от корня) всех существенных цепей информационного графа, соответствующих x , общие. П.2 обоснован.

Так как слов, удовлетворяющих запросу x , всего $d(I_1, x)$, а предикатных ребер в графе $d(I_1, x) - 1$, то найдется лист, в который запрос x проходит по переключательному ребру. Значит, этот лист является $(e(x) + 1)$ -й вершиной (считая от корня) каждой существенной цепи информационного графа, соответствующей x . Далее, в рассматриваемом графе всего $e(x)$ переключательных вершин, притом в каждой существенной цепи информационного графа, соответствующей x , это вершины с номерами $1, 2, \dots, e(x)$, считая от корня. Значит, все ребра, выходящие из остальных вершин каждой такой цепи, являются предикатными. В рассматриваемом ИГ каждое предикатное ребро входит в лист, следовательно, в каждой существенной цепи информационного графа, соответствующей x , вершины с номерами, большими $e(x)$ (считая от корня), являются листьями. П.3 обоснован.

Как уже показано, каждый лист, в который проходит запрос x , должен иметь номер не меньше $e(x) + 1$, считая от корня, в какой-либо существенной цепи ИГ, соответствующей запросу x . Значит, каждое выходящее из этого листа ребро является предикатным, то есть п.4 также обоснован.

□

Лемма 7. Пусть $k \geq 3$ и в библиотеке $V \subseteq N_k^n$ из p слов, где $p \leq (k-1)^n$, каждая буква каждого слова отлична от k , пусть также $U \in \mathcal{U}(I_1, \mathcal{F}_1)$, где $I_1 = \langle X, V, \rho_1 \rangle$, и для каждого запроса $x \in X$, которому удовлетворяет хотя бы одно слово из библиотеки, выполнено $T(U, x) = R(I_1, \mathcal{F}_1, x)$. Тогда в каждой существенной цепи информационного графа, соответствующей запросу x , переключатель $g_i, i \in \{2, \dots, e(x)\}$, сопоставлен вершине с номером $i-1$ или i (считая от корня), а переключатель g_1 может быть сопоставлен вершине с любым номером от 1 до $e(x)$.

Доказательство. Заметим, что согласно пунктам 1 и 2 леммы 6, у каждой существенной цепи информационного графа, соответствующей запросу x , первые $e(x)$ вершин (считая от корня) общие, причем этим вершинам сопоставлены переключатели $g_i, i \in \{1, \dots, e(x)\}$. Обозначим часть любой из этих существенных цепей от корня до $e(x)$ -й вершины через $D = D(x)$. Докажем лемму индукцией по $l(x)$.

Базис индукции. Пусть $l(x) = 1$. Тогда в D две вершины: 1-я и 2-я. Им должны быть приписаны переключатели g_1 и g_2 , притом они могут идти в любом порядке. Как видно, условие леммы для базиса индукции выполнено.

Шаг индукции. Пусть теперь условие леммы выполняется для всех запросов длины меньше s . Докажем, что оно выполнено для всех запросов длины s . Рассмотрим запрос $x, l(x) = s$. Обозначим $x' = x[1 \dots l(x) - 1]$. Рассмотрим цепь $D' = D(x')$. Заметим, что результат вычисления всех переключателей, кроме $g_{l(x)}$, на запросах x и x' одинаков, а результат вычисления $g_{l(x)}$ на этих запросах разный (соответственно $x[l(x)]$ и $k+1$). Значит, часть цепи D' от корня до вершины β с переключателем $g_{l(x)}$ является частью цепи D . Вершина β по предположению индукции имеет в цепи D' номер $l(x) - 1$ либо $l(x)$. Все вершины части цепи D' от корня до β сохраняют свои номера и в цепи D . Рассмотрим оба случая.

1) β имеет номер $l(x) - 1$. Тогда в части цепи D от корня до β имеются вершины с переключателями $g_2, \dots, g_{l(x)}$, но нет вершин с переключателями g_1 и $g_{l(x)+1}$. Если $l(x) < n$, то у D $l(x) + 1$ вершина, и вершинам с номерами $l(x)$ и $l(x) + 1$ должны быть приписаны переключатели g_1 и $g_{l(x)+1}$, притом они могут идти в любом порядке. Если $l(x) = n$, то у D $l(x)$ вершин, и вершине с номером

$l(x)$ должен быть сопоставлен переключатель g_1 .

2) β имеет номер $l(x)$. Тогда D' — часть D . В части цепи D от корня до β имеются вершины с переключателями $g_1, \dots, g_{l(x)}$. Если $l(x) = n$, то у D $l(x)$ вершин, поэтому D и D' совпадают. Если же $l(x) < n$, то у D имеется $l(x) + 1$ вершин, и вершине с номером $l(x) + 1$ должен быть приписан переключатель $g_{l(x)+1}$.

Этим исчерпываются все возможные случаи. \square

Перейдем к доказательству основной теоремы данной главы.

Теорема 3. Если $k \geq 3$, $p \leq (k - 1)^{\lfloor n/3 \rfloor}$, то $pn^2/9 \lesssim Q_1(p, n) \lesssim (k + 2)pn^2/2$ при $n \rightarrow \infty$, $p \rightarrow \infty$.

Доказательство. Рассмотрим библиотеку $V = \{v_i, i \in \{1, \dots, p\}\}$, каждая запись v_i которой определена согласно соотношению (2.3), и произвольный граф $U \in \mathcal{U}_{I_1}$, где $I_1 = \langle X, V, \rho_1 \rangle$. Заметим, что библиотека V удовлетворяет условиям лемм 1, 6 и 7.

Рассмотрим цепь C_1 , по которой какой-либо запрос x_1 с $d(I_1, x_1) > 0$ проходит из корня в лист (назовем этот лист γ_1) и которая не содержит никакого другого листа. Рассмотрим также цепь C_2 , по которой какой-либо другой запрос x_2 с $d(I_1, x_2) > 0$ проходит из корня в лист (назовем этот лист γ_2) и которая не содержит никакого другого листа. Пусть в графе найдутся 2 цепи A_1 и A_2 , по которым x_1 и x_2 проходят из корня в одну и ту же переключательную вершину α , причем ребра, по которым эти запросы проходят непосредственно в α , не совпадают. Пусть в A_1 имеется l_1 вершин, считая α , в A_2 имеется l_2 вершин, считая α . Не ограничивая общности, считаем, что $l_1 \leq l_2$. Заметим, что $l_2 \geq l_1 > 1$. По лемме 7 имеем: вершине α сопоставлен, с одной стороны, один из переключателей g_1, g_{l_1}, g_{l_1+1} , а с другой стороны — переключатель из множества $\{g_1, g_{l_2}, g_{l_2+1}\}$. Возможны следующие случаи.

1) Вершине α сопоставлен переключатель g_{l_1+1} , совпадающий с переключателем g_{l_2} . То есть $l_2 = l_1 + 1$. Из леммы 7 вытекает, что для каждого $i \in \{1, \dots, l_2 - 1\}$ среди первых (считая от корня) $l_2 - 1$ вершин цепи A_2 найдется вершина с переключателем g_i . Рассмотрим запрос $x_3 = x_2[1 \dots l_2 - 1]$. Так как $d(x_2) > 0$, то $d(x_3) > 0$. Запрос x_3 пройдет в вершину α , а из нее по ребру $k + 1$ — в лист β , которому приписано слово v , удовлетворяющее запросу x_3 . Далее, из

леммы 7 вытекает, что для любого $i \in \{2, \dots, l_1\}$ среди первых $l_1 - 1$ вершин цепи A_1 найдется вершина с переключателем g_i , а вершин с переключателем g_1 нет. Рассмотрим запрос x_4 , полученный из $x_1[1 \dots l_1]$ заменой первой буквы на k . Тогда этот запрос пройдет по цепи A_1 в вершину α , а из нее по ребру $k + 1$ — в β . Следовательно, x_4 пройдет в лист, которому приписано слово v , чего быть не должно, так как в v нет буквы k . Значит, случай 1) невозможен.

2) Вершине α сопоставлен переключатель g_{l_1} , совпадающий с переключателем g_{l_2} . То есть $l_2 = l_1$. Как и в предыдущем случае, запрос $x_4 = x_2[1 \dots l_2 - 1]$ пройдет в вершину α . В эту же вершину пройдет слово $x_3 = x_1[1 \dots l_1 - 1]$. Заметим, что $x_4 \neq x_3$, так как иначе эти запросы попали бы в α по одной и той же цепи. Рассмотрим все слова $v \in V$. Положим $q_{1,v}$ равным $\min\{q : v[q \dots q + l_1 - 2] = x_3\}$, если подслово x_3 содержится в v , и $n + 1$ в противном случае. Аналогично положим $q_{2,v} = \min\{q : v[q \dots q + l_1 - 2] = x_4\}$, если подслово x_4 содержится в v , и $n + 1$ в противном случае. Пусть $q' = \min_{v \in V} \min(q_{1,v}, q_{2,v})$. Тогда $q' < n + 1$, так как и для запроса x_3 , и для x_4 найдется в V хотя бы одно слово, удовлетворяющее этому запросу. Не ограничивая общности, будем считать, что слово $w \in V$ таково, что $w[q' \dots q' + l_1 - 2] = x_3$ и цепь C , по которой запрос $x_5 = w[q' \dots n]$ проходит из корня в лист γ со словом w , не содержит других листьев с записями, удовлетворяющими запросу x_5 . Тогда цепь A_1 является началом цепи C . Следовательно, на отрезке цепи C от α до γ есть вершины с переключателями $g_i, i \in \{l_1, \dots, \min(n - q' + 2, n)\}$, и нет вершин с другими переключателями. Рассмотрим запрос x_6 , полученный из x_5 заменой первых $l_1 - 1$ букв на соответствующие буквы слова x_4 . Запрос x_6 пройдет по цепи A_2 в вершину α , а затем по отрезку цепи C от α до γ — в вершину γ . Таким образом, x_6 является подсловом слова w . Так как $l(x_6) = n - q' + 1$ и $x_6 \neq x_5$, то должно найтись такое число $q'' < q'$, что $w[q'' \dots q'' - q' + n] = x_6$. Но, учитывая, что x_3 — начало x_6 , это противоречит выбору q' . Значит, случай 2) невозможен.

3) Вершине α сопоставлен переключатель g_{l_1+1} , совпадающий с переключателем g_{l_2+1} . То есть $l_2 = l_1 < n$. Из леммы 7 вытекает, что для любого $i \in \{2, \dots, l_1\}$ среди первых $l_1 - 1$ вершин цепи A_1 найдется вершина с переключателем g_i , а вершин с переключателем g_1 нет. Это же верно и для A_2 .

Запрос $x_3 = x_1[1 \dots l_1]$ пройдет в вершину α , а из нее по ребру $k + 1$ — в какую-либо вершину β , причем β должен быть сопоставлен переключатель g_1 . Рассмотрим запрос x_4 , полученный из $x_2[1 \dots l_1]$ заменой первой буквы на $x_1[1]$. Дальнейшие рассуждения повторяют рассуждения из предыдущего пункта с небольшими изменениями. Заметим, что $x_3 \neq x_4$, так как иначе эти запросы должны были бы пройти в α по одному и тому же пути, чего не происходит. Для любого слова $v \in V$ положим $q_{1,v}$ равным $\min\{q : v[q \dots q + l_1 - 2] = x_3\}$, если подслово x_3 содержится в v , и $n + 1$ в противном случае. Аналогично положим $q_{2,v} = \min\{q : v[q \dots q + l_1 - 2] = x_4\}$, если подслово x_4 содержится в v , и $n + 1$ в противном случае. Пусть $q' = \min_{v \in V} \min(q_{1,v}, q_{2,v})$. Тогда $q' < n + 1$, так как для запроса x_3 найдется в V хотя бы одно слово, удовлетворяющее этому запросу. Не ограничивая общности, будем считать, что слово $w \in V$ таково, что $w[q' \dots q' + l_1 - 1] = x_3$ и цепь C , по которой запрос $x_5 = w[q' \dots n]$ проходит из корня в лист γ со словом w , не содержит других листьев с записями, удовлетворяющими запросу x_5 . Тогда цепь A_1 является началом цепи C . Следовательно, на отрезке цепи C от α до γ есть вершины с переключателями $g_i, i \in \{l_1 + 1, \dots, \min(n - q' + 2, n)\}$, а также вершина с переключателем g_1 , и нет вершин с другими переключателями. Рассмотрим запрос x_6 , полученный из x_5 заменой первых l_1 букв на соответствующие буквы слова x_4 . Запрос x_6 пройдет по цепи A_2 в вершину α , а затем по отрезку цепи C от α до γ — в вершину γ . Таким образом, x_6 является подсловом слова w . Так как $l(x_6) = n - q' + 1$ и $x_6 \neq x_5$, то должно найтись такое число $q'' < q'$, что $w[q'' \dots q'' - q' + n] = x_6$. Но, поскольку x_4 — начало x_6 , это противоречит выбору q' . Значит, случай 3) невозможен.

4) Вершине α сопоставлен переключатель g_1 . Согласно лемме 7, для любого $i \in \{2, \dots, l_1\}$ среди первых $l_1 - 1$ вершин цепи A_1 найдется вершина с переключателем g_i и для любого $i \in \{2, \dots, l_2\}$ среди первых $l_2 - 1$ вершин цепи A_2 найдется вершина с переключателем g_i . Рассмотрим цепь C_i , по которой запрос $x_i, i \in \{1, 2\}$, проходит из корня в какой-либо лист γ_i и которая не содержит других листьев. При этом цепи C_1 и C_2 содержат соответственно цепи A_1 и A_2 и поэтому не совпадают. Скажем, что вершина обладает O -свойством относительно цепи C_i , если ей сопоставлен переключатель g_1 и она является ближайшей к

γ_i переключательной вершиной цепи C_i . Случай 4) можно разбить на несколько подслучаев.

4.1) Вершина α не обладает O -свойством ни относительно цепи C_1 , ни относительно цепи C_2 . Пусть запрос $x_3 = x_1[1 \dots l_1]$ проходит из α в переключательную вершину β . Из леммы 7 вытекает, что этой вершине может быть сопоставлен только переключатель g_{l_2+1} . Рассмотрим запрос x_4 , полученный из запроса $x_2[1 \dots l_2]$ заменой первой буквы на $x_1[1]$. Этот запрос пройдет из корня в α по цепи A_2 , а далее — в β по ребру $x_1[1]$. Поэтому вершине β должен быть сопоставлен переключатель g_{l_1+1} , откуда получаем, что $l_1 = l_2$. Далее практически без изменений повторяются рассуждения, приведенные в п.3. Заметим, что $x_3 \neq x_4$, так как иначе эти запросы должны были бы пройти в α по одному и тому же пути, чего не происходит. Для любого слова $v \in V$ положим $q_{1,v}$ равным $\min\{q : v[q \dots q + l_1 - 1] = x_3\}$, если подслово x_3 содержится в v , и $n + 1$ в противном случае. Аналогично положим $q_{2,v} = \min\{q : v[q \dots q + l_1 - 1] = x_4\}$, если подслово x_4 содержится в v , и $n + 1$ в противном случае. Пусть $q' = \min_{v \in V} \min(q_{1,v}, q_{2,v})$. Тогда $q' < n + 1$, так как для запроса x_3 найдется в V хотя бы одно слово, удовлетворяющее этому запросу. Не ограничивая общности, будем считать, что слово $w \in V$ таково, что $w[q' \dots q' + l_1 - 1] = x_3$ и цепь C , по которой запрос $x_5 = w[q' \dots n]$ проходит из корня в лист γ со словом w , не содержит других листьев с записями, удовлетворяющими запросу x_5 . Тогда цепь A_1 является началом цепи C . Следовательно, на отрезке цепи C от β до γ есть вершины с переключателями $g_i, i \in \{l_1 + 1, \dots, \min(n - q' + 2, n)\}$, и нет вершин с другими переключателями. Рассмотрим запрос x_6 , полученный из x_5 заменой первых l_1 букв на соответствующие буквы слова x_4 . Запрос x_6 пройдет по цепи A_2 в вершину α , потом — в β по ребру $x_2[1]$, а затем по отрезку цепи C от β до γ — в вершину γ . Таким образом, x_6 является подсловом слова w . Так как $l(x_6) = n - q' + 1$ и $x_6 \neq x_5$, то должно найтись такое число $q'' < q'$, что $w[q'' \dots q'' - q' + n] = x_6$. Но, поскольку x_4 — начало x_6 , это противоречит выбору q' . Значит, случай 4.1) невозможен.

4.2) Вершина α обладает O -свойством относительно цепи C_i , но не относительно C_j , где $i, j \in \{1, 2\}, i \neq j$. Тогда, в частности, $l_j < n$. Запрос x_i проходит из α непосредственно в γ_i . Рассмотрим такой запрос x_3 , что $x_3[1] = x_i[1]$,

$x_3[2 \dots l_j] = x_j[2 \dots l_j]$, $x_3[l_j + 1] = k$. Этот запрос пройдет по цепи A_j в вершину α , оттуда по ребру $x_i[1]$ — в вершину γ_i . Значит, слово, сопоставленное γ_i , должно удовлетворять этому запросу, чего быть не может, так как оно не содержит буквы $x_3[l_j + 1] = k$.

4.3) Вершина α обладает O -свойством и относительно цепи C_1 , и относительно C_2 . Такую вершину будем называть вершиной слияния.

Итак, из всех возможных вариантов мы не исключили только 4.3.

Ранее мы показали, что все слова $v_{i,a,b} = v_i[a \dots b]$, $1 \leq a \leq r$, $2r + 1 \leq b \leq n$, $1 \leq i \leq p$, разные. Значит, все слова $v_{i,a,b}$, у которых $a < r$, также разные; обозначим через V' множество таких слов. Как нетрудно проверить, $|V'| = (r - 1) \cdot (n - 2r) \cdot p$. Для каждого запроса x_i из V' , $i \in \{1, \dots, |V'|\}$, рассмотрим цепь C_i , по которой запрос x_i проходит из корня в какой-либо лист γ_i со словом, удовлетворяющим запросу, и которая не содержит других листьев с записями, удовлетворяющими этому запросу. Выделим для каждого x_i такое ребро, что никакой запрос $x_j \in V'$ не проходит по этому ребру, если $j \neq i$.

Если C_i не содержит вершин слияния, то в качестве такого ребра можно выбрать последнее (считая от корня) переключательное ребро c_i цепи C_i . Действительно, пусть это ребро выходит из вершины α . Тогда для любого $i \in \{1, \dots, \min(l(x_i + 1), n)\}$ найдется вершина цепи C_i от корня до α с переключателем g_i , причем в каждую вершину этого отрезка цепи C_i , кроме корня, входит одно ребро (в корень — 0 ребер). Значит, для того, чтобы произвольный запрос x прошел по ребру c_i , должны выполняться равенства $x[1] = x_i[1], \dots, x[l(x_i)] = x_i[l(x_i)]$, а также, если $l(x_i) < n$, равенство $x[l(x_i) + 1] = x_i[l(x_i) + 1] = 0$. Следовательно, $x = x_i$.

Пусть C_i содержит вершину слияния. Очевидно, ни в какой цепи не может быть более одной такой вершины. Назовем эту вершину α . Тогда в качестве нужного ребра можно выбрать ребро e_i цепи C_i , входящее в α . Действительно, для того, чтобы произвольный запрос x прошел по ребру e_i , должны выполняться равенства $x[2] = x_i[2], \dots, x[l(x_i)] = x_i[l(x_i)]$, а также, если $l(x_i) < n$, равенство $x[l(x_i) + 1] = x_i[l(x_i) + 1] = 0$. Но слова из V' подобраны так, что из равенства вторых букв любых двух слов из V' следует равенство их первых букв. Значит, снова получаем $x = x_i$.

Итак, для каждого слова из V' мы выделили в ИГ U ребро, соответствующее этому слову и не соответствующее никаким другим словам из V' . Значит, ребер в U не меньше $|V'| = (r - 1) \cdot (n - 2r) \cdot p$. Таким образом, $pn^2/9 \lesssim Q_1(p, n), n \rightarrow \infty, p \rightarrow \infty$.

Согласно лемме 2 при любой библиотеке $V \subseteq Y$ ИГ $U_1(I_1, \mathcal{F}_1)$, где $I_1 = \langle X, V, \rho_1 \rangle$, решает задачу I_1 над базовым множеством \mathcal{F}_1 . Согласно лемме 3 для любого запроса x выполнено неравенство $T(U_1(I_1, \mathcal{F}_1), x) \leq R(I_1, \mathcal{F}_1, x)$. Значит, $U_1 \in \mathcal{U}_{I_1}$. Тогда $Q(U_1) \leq (k + 2)p(n + 2)(n - 1)/2 + p + k + 1$ согласно лемме 4. Следовательно, $Q_1(p, n) \lesssim (k + 2)pn^2/2, n \rightarrow \infty, p \rightarrow \infty$.

Имеем $pn^2/9 \lesssim Q_1(p, n) \lesssim (k + 2)pn^2/2, n \rightarrow \infty, p \rightarrow \infty$, что и требовалось. □

Глава 2

Поиск вхождений подслова в множестве слов

В главе 2 рассматривается задача поиска вхождений подслова в множестве слов. Изложенные в настоящей главе результаты опубликованы в [32].

§ 1 Определения

Пару (w, i) , где $w \in W_{k,n}$ и $i \in N_{l(w)}$, назовем вхождением в слово w . Множество всех вхождений в слова из некоторого множества M будем обозначать как $Oc(M)$. Определим отношение поиска ρ_2 следующим образом:

$$x \rho_2 (v, i) \Leftrightarrow x = v[i \dots l(x) + i - 1],$$

где $x \in X$ и $(v, i) \in Oc(Y)$. Будем рассматривать задачу информационного поиска $I_2 = \langle X, V, \rho_2 \rangle$, где $V = \{v_1, v_2, \dots, v_p\}$, $V \subseteq Y$. Будем считать, что задача $I_2 = \langle X, V, \rho_2 \rangle$ состоит в перечислении для произвольно взятого запроса $x \in X$ всех тех и только тех вхождений $\omega \in Oc(V)$, для которых выполнено $x \rho_2 \omega$. Назовем эту задачу задачей поиска вхождений подслова.

Понятие информационного графа вводится так же, как и для главы 1, с той разницей, что там, где для главы 1 рассматривалось множество Y , в настоящей главе будет рассматриваться множество $Oc(Y)$.

Будем говорить, что вершина α информационного графа схемно достижима из вершины β , если существует ориентированная цепь из β в α .

Скажем, что ИГ U решает ЗИП $I_2 = \{X, V, \rho_2\}$, если для любого запроса $x \in X$ ответ ИГ U на этот запрос содержит все те и только те вхождения из $Oc(V)$, которые удовлетворяют запросу x , то есть $J_U(x) = \{\omega \in Oc(V) : x \rho_2 \omega\} \stackrel{def}{=} J_{I_2}(x)$.

Обозначим $J_V = \{J_{I_2}(x) : x \in X\}$ — множество ответов, соответствующих библиотеке V . Назовем ответом любое множество, являющееся элементом J_V . Т.е., ответ — это множество всех вхождений, удовлетворяющих одному и тому же запросу. Если задача I_2 зафиксирована, ответ на запрос x будем обозначать как $J(x)$.

В качестве базового множества \mathcal{F}_2 будем рассматривать $\langle F, G_2 \rangle$, $G_2 = G_1 \cup H$, где F и G_1 — множества функций, описанные для главы 1, а функции из множества $H = \{h_{v,i}(x), v \in W_{k,n} \setminus N_k^n, i \in N_{l(v)}\}$ определяются следующим образом:

$$h_{v,i}(x) = \begin{cases} 1, & \text{если } i > l(x) \text{ либо } x[i \dots l(x)] \text{ — собственный префикс } v, \\ 2, & \text{если } i \leq l(x) \text{ и } v \text{ — префикс } x[i \dots l(x)], \\ 3, & \text{во всех остальных случаях.} \end{cases} \quad (2.1)$$

Функция на языке C , реализующая алгоритм вычисления переключателя $h_{v,i}(x)$, может выглядеть, например, так:

```
int h(v,i,x)
{
  int j=0;
  do
  {
    if(x[j+i]!=v[j+1])
    {
      if(x[j+i]==0) return 1;
      return 3;
    }
    j++;
  }
  while(v[j+1]!=0);
  return 2;
}
```

Определим сложность вычисления переключателя $h_{v,i}$ на запросе x . Будем считать, что сложность каждого шага цикла равна 2 (так как производится ровно 2 сравнения), а сложность перехода по любой из ссылок 1, 2 и 3 равна 1. Тогда

$$t(h_{v,i}, x) = \begin{cases} 1 + 2 \cdot \max(l(x) - i + 2, 1), & \text{если } h_{v,i}(x) = 1 \\ 1 + 2 \cdot l(v), & \text{если } h_{v,i}(x) = 2, \\ 3 + 2 \cdot \max_{j: x[i\dots i+j-1]=v[1\dots j]} j, & \text{если } h_{v,i}(x) = 3. \end{cases} \quad (2.2)$$

Введем следующие обозначения: $\mathcal{U}(I_2, \mathcal{F}_2)$ — множество всех ИГ над базовым множеством \mathcal{F}_2 , решающих ЗИП $I_2 = \langle X, V, \rho_2 \rangle$; $R(I_2, \mathcal{F}_2, x) = e(x) + t \cdot (|J_{I_2}(x)| - 1)$; $Q_2(p, n) = \max_{I: I = \langle X, V, \rho_2 \rangle, |V|=p} \min_{U \in \mathcal{U}(I_2, \mathcal{F}_2)} Q(U)$; $T(I_2, x) = \min_{U \in \mathcal{U}(I_2, \mathcal{F}_2)} T(U, x)$.

Пусть \mathcal{U}_{fast} — множество таких графов $U \in \mathcal{U}(I_2, \mathcal{F}_2)$, что для каждого из них при любых x из X выполняется условие $T(U, x) \leq 3R(I_2, \mathcal{F}_2, x)$. Функцию, равную $\max_{I_2: I_2 = \langle X, V, \rho_2 \rangle, |V|=p} \min_{U \in \mathcal{U}_{fast}} Q(U)$, если множество \mathcal{U}_{fast} непусто, будем обозначать через $Q_{fast}(p, n)$; если $\mathcal{U}_{fast} = \emptyset$, положим $Q_{fast}(p, n) = \infty$.

§ 2 Нижняя оценка сложности поиска вхождений подслова в множестве слов

В данном параграфе получена нижняя оценка сложности поиска.

Теорема 4. *При $k \geq 2$ для каждой задачи $I_2 = \langle X, V, \rho_2 \rangle$, для любых $U \in \mathcal{U}(I_2, \mathcal{F}_2)$ и $x \in X$ выполняется следующее условие: если $|J_{I_2}(x)| \geq 1$, то $T(U, x) \geq R(I_2, \mathcal{F}_2, x)$.*

Доказательство этой теоремы в целом аналогично доказательству теоремы 1, приведенной в главе 1. Однако, есть и существенные отличия, связанные с тем, что в главе 1 осуществлялся поиск слов, а не вхождений в них; кроме того, в базовом множестве отсутствовало семейство функций $h_{v,i}(x)$.

Доказательство. Рассмотрим произвольную библиотеку $V \subseteq N_k^n$. Рассмотрим произвольный ИГ $U \in \mathcal{U}(I_2, \mathcal{F}_2)$, где $I_2 = \langle X, V, \rho_2 \rangle$. Пусть C — какая-либо цепь, по которой запрос проходит из корня в лист с вхождением (w, j) , удовлетворяющим запросу (обозначим этот лист γ), причем ни одна вершина этой

цепи, кроме γ , не является листом. Обозначим через C' цепь, полученную из C удалением вершины γ и входящего в нее ребра. Рассмотрим произвольное $i \in \mathbb{N}$, $1 \leq i \leq e(x)$. Пусть всем переключательным вершинам цепи C' сопоставлены функции, не зависящие от $x[i]$ существенно. Заменяем $x[i]$ на 1, если $j + i - 1 \leq n$ и $w[j + i - 1] \neq 1$, и на 2 в противном случае. Обозначим получившийся запрос как x' . Этот запрос также будет проходить в лист с записью (w, j) . Мы получим, что либо $j + l(x') - 1 > n$, либо $w[j + i - 1] \neq x'[i]$, т.е. в любом случае вхождение (w, j) не удовлетворяет запросу x' . Имеем противоречие с тем, что граф U решает задачу поиска вхождений под слова. Получим, что для любого $i \in \mathbb{N}$, $1 \leq i \leq e(x)$ найдется вершина цепи, не совпадающая с γ и такая, что ей сопоставлен переключатель, существенно зависящий от $x[i]$. Это либо переключатель $g_i(x)$, либо переключатель $h_{v,r}(x)$, для которого $l(v) \geq i - r$ и $x[r \dots i - 1] = v[1 \dots i - r]$. В последнем случае в процессе вычисления $h_{v,r}(x)$ $x[i]$ сравнивается с $v[i - r + 1]$, после чего происходит еще не менее одного сравнения. Таким образом, минимальный вклад, вносимый обработкой буквы $x[i]$ в сложность вычисления ИГ U на запросе x , равен 1. Значит, суммарный вклад в $T(U, x)$ переключателей, сопоставленных вершинам цепи C' — не меньше $e(x)$.

Пусть $|J_{I_2}(x)| > 1$. Для каждого отличного от γ листа, в который проходит запрос x , выберем ребро, по которому x входит в этот лист. Так как никакому листу не может быть сопоставлено более одного вхождения, все эти ребра разные, причем если некоторые из этих ребер исходят из одной и той же вершины, то это предикатные ребра. Кроме того, ни одно из этих ребер не принадлежит C . Значит, для прохода запроса по таким ребрам будет вычислено не менее $|J_{I_2}(x)| - 1$ функций, отличных от переключателей, сопоставленных вершинам цепи C' . Так как $t < 1$, получим, что $T(U, x) \geq e(x) + t \cdot (|J_{I_2}(x)| - 1) = R(I_2, \mathcal{F}_2, x)$. \square

§ 3 Нижняя оценка объема памяти алгоритма, решающего задачу поиска вхождений под слова

Лемма 8. Для любых натуральных $n \geq 2$, $k \geq 3$ и $p < (k - 1)^{n-1}$ выполнено $Q_2(p, n) \geq p(2n - \lceil \log_{k-1} p \rceil) - 1$.

Доказательство. Рассмотрим все слова $w_1, \dots, w_{(k-1)^r}$ длины $r = \lceil \log_{k-1} p \rceil$, не

содержащие букву 1. Определим каждое слово $v_i, i \in N_p, v_i \in N_k^n$, следующим образом:

$$\begin{cases} v_i[j] = 1, & \text{если } j \in \mathbb{N}, j \leq n - r \\ v_i[n - r + 1 \dots n] = w_i \end{cases} \quad (2.3)$$

Будем рассматривать библиотеку $V_1 = \{v_1, v_2, \dots, v_p\}$. Рассмотрим произвольный информационный граф, решающий задачу $I_2 = \langle X, V, \rho_2 \rangle$, где $|V| = p$. Для каждого вхождения (v_i, j) найдется запрос, которому это вхождение удовлетворяет, например, таковым будет запрос $v_i[j \dots n]$. Поэтому каждое вхождение (v_i, j) должно быть сопоставлено хотя бы одному листу, и в этот лист должна вести из корня ориентированная цепь ребер.

Рассмотрим множество вхождений в слова из библиотеки V_1 , сопоставленных ровно одному листу, причем такому, что в него входит ровно одно ребро. Обозначим это множество через $Oc_1(V_1)$. Рассмотрим далее подмножество $Oc_2(V_1)$ множества $Oc_1(V_1)$, состоящее из всех вхождений (v_i, j) , для которых $j < n - r$. Множество $Oc_1(V_1) \setminus Oc_2(V_1)$, состоящее из всех вхождений (v_i, j) , для которых $j \geq n - r$, обозначим через $Oc_3(V_1)$.

Рассмотрим вхождение $(v, j) \in Oc_2(V_1)$. Пусть α — вершина, из которой ведет ребро в лист γ с (v, j) .

Предположим, что из α ведет ребро в лист γ' , которому сопоставлено вхождение $(v', j') \in Oc_2(V_1)$, отличное от (v, j) . Так как в вершину γ , как и в вершину γ' , входит только одно ребро, то все запросы, попадающие в эти вершины, проходят через вершину α . Следовательно, вершине α не может быть сопоставлен переключатель, так как запрос 1 должен пройти и в вершину γ , и в вершину γ' . Значит, ребрам, выходящим из вершины α , сопоставлен предикат, тождественно равный 1, т.е. в вершины γ и γ' проходят одни и те же запросы. В частности, в обе вершины проходят запросы $v[j \dots n]$ и $v'[j' \dots n]$. Так как в лист γ проходят только префиксы слова $v[j \dots n]$, а в лист γ' — только префиксы слова $v'[j' \dots n]$, то $v[j \dots n] = v'[j' \dots n]$. Из этого следует, что $v[n - r + 1 \dots n] = v'[n - r + 1 \dots n]$, откуда по построению библиотеки V_1 получаем, что $v = v'$. Кроме того, $j = j'$, так как $v[j \dots n] = v'[j' \dots n]$. Следовательно, вхождения (v, j) и (v', j') совпадают, что противоречит выбору вхождения (v', j') . Итак, мы показали, что из α не ведет ребро ни в какой лист с вхождением из $Oc_2(V_1)$, кроме γ .

Предположим, что самой вершине α сопоставлено некоторое вхождение $(v', j') \in Oc(V_1)$, отличное от (v, j) . Тогда все слова, проходящие в вершину γ , являются префиксами слова $v'[j' \dots n]$. В частности, таким префиксом является слово $v[j \dots n]$, следовательно, слово $v[n - r + 1 \dots n]$ — подслово слова $v'[j' \dots n]$. Отсюда по построению библиотеки V_1 получаем, что $v = v'$, так как $v[n - r + 1 \dots n]$ является подсловом лишь одного слова из V_1 . Так как в слове v подслово $v[n - r + 1 \dots n]$ содержится лишь в качестве суффикса, то слово $v[j \dots n]$ одновременно и префикс, и суффикс слова $v[j' \dots n]$, при том, что $v[j \dots n]$ не может встречаться в слове $v[j' \dots n]$ более одного раза. Следовательно, $v[j' \dots n] = v[j \dots n]$, откуда $j = j'$, и (v, j) и (v', j') совпадают, что противоречит выбору вхождения (v', j') . Итак, мы показали, что α не сопоставлено никакое вхождение (само вхождение (v, j) не может быть ей сопоставлено, так как $(v, j) \in Oc_2(V_1)$).

Если существует ребро, по которому запрос $v[j \dots n]$ проходит в вершину α (т.е. если α — не корень информационного графа), сопоставим это ребро вхождению (v, j) . Прделаем так для всех вхождений из $Oc_2(V_1)$, для которых такое ребро существует. Так как ни из какой вершины (в том числе, из корня) не может вести более одного ребра в вершины с вхождениями из $Oc_2(V_1)$, то вершинам из $Oc_2(V_1)$ будет сопоставлено суммарно не менее $|Oc_2(V_1)| - 1$ различных ребер; обозначим это множество через E_1 .

Рассмотрим также все ребра, входящие в листья, которым сопоставлены вхождения из $Oc(V_1)$. Обозначим это множество через E_2 . В этом множестве не менее $2|Oc(V_1) \setminus Oc_1(V_1)| + |Oc_1(V_1)|$ ребер. Ранее было показано, что каждой вершине, в которую входит ребро из множества E_1 , не сопоставлено никакое вхождение, следовательно, множества E_1 и E_2 не пересекаются.

Оценим число ребер в множестве $E_1 \cup E_2$: $|E_1 \cup E_2| \geq 2|Oc(V_1) \setminus Oc_1(V_1)| + |Oc_1(V_1)| + |Oc_2(V_1)| - 1 = 2|Oc(V_1) \setminus Oc_1(V_1)| + |Oc_3(V_1)| + 2|Oc_2(V_1)| - 1 = 2|Oc(V_1)| - |Oc_3(V_1)| - 1$.

Так как $|Oc(V_1)| = pn$, а $|Oc_3(V_1)| = pr = p \lceil \log_{k-1} p \rceil$, получим, что $Q_2(p, n) \geq |E_1 \cup E_2| \geq p(2n - p \lceil \log_{k-1} p \rceil) - 1$. \square

§ 4 Решающий задачу поиска вхождений подслова информационный граф с минимальной по порядку объемной и временной сложностью

Сформулируем и докажем основную теорему данной главы.

Теорема 5. *Для любых натуральных $n \geq 2$, $k \geq 3$ и $p < (k - 1)^{n-1}$ выполнено $p(2n - \lceil \log_{k-1} p \rceil) - 1 \leq Q_2(p, n) \leq Q_{fast}(p, n) \leq (2k + 11)pn$.*

Прежде чем перейти к непосредственному доказательству теоремы 5, докажем три вспомогательные леммы.

Лемма 9. *Пусть $I_2 = \langle X, V, \rho_2 \rangle$ — задача поиска вхождений подслова в множестве слов, причем V состоит из p слов. Тогда число различных ответов, соответствующих V , не превосходит $2pn$, и найдется ИГ U' без листьев (т.е. вершин, которым сопоставлены вхождения) над базовым множеством \mathcal{F}_2 , обладающий следующими свойствами:*

- 1) $Q(U') < 2(k + 4)pn$;

- 2) Для каждого запроса $x \in X$ выполнено $T(U', x) \leq 3e(x)$;

- 3) для каждого ответа J , соответствующего V , найдется вершина $\kappa(J)$, из которой не выходит ни одно ребро и в которую попадают все те и только те запросы x , ответ на которые совпадает с J .

Доказательство. Рассмотрим все вхождения из $Os(V)$. Для каждого вхождения (v_i, j) рассмотрим $x = v_i[j \dots n]$ (напомним, что каждое слово длины s представляется набором длины n , первые s элементов которого — буквы этого слова, а каждый из оставшихся $n - s$ элементов равен $k + 1$). Выберем $e(x) + 1$ вершин, пронумеруем их от 1 до $e(x) + 1$, а затем для каждого $r \in \{1, \dots, e(x)\}$ проведем из вершины под номером r в вершину под номером $r + 1$ ребро, присвоив этому ребру номер $x[r]$. Вершину с номером $e(x) + 1$ дополнительно назовем $\lambda(x)$.

Теперь отождествим все вершины, имеющие номер 1. Получившуюся вершину объявим корнем. Далее последовательно для каждого натурального r от 1 до $n - 1$ сделаем следующее. Рассмотрим каждую вершину, имеющую номер r . Отождествим все ребра с одинаковым номером, выходящие из этой вершины,

и вершины, в которые входят эти ребра (по построению, все отождествленные вершины будут иметь номер $r + 1$). Заметим, что по построению из вершины с номером n не могут выходить два или более ребра с одним и тем же номером, так как это означало бы наличие в библиотеке одинаковых слов.

Полученный граф является ориентированным деревом, так как в корень не входит ни одного ребра, а в каждую вершину, кроме корня, входит одно ребро. Каждой вершине α , в которую ведет ребро с номером из N_k , можно сопоставить слово $x(\alpha)$, являющееся конкатенацией номеров ребер, по которым можно пройти из корня в эту вершину. По построению графа, никаким двум различным вершинам не будет сопоставлено одно и то же слово (в противном случае эти вершины ранее должны были быть отождествлены). Кроме того, по построению графа, каждое слово $x(\alpha)$ является префиксом какого-либо слова $v_i[j \dots n]$, т.е. подсловом какого-либо слова из V , и обратно, каждое подслово какого-либо слова из V является $x(\alpha)$ для какой-либо вершины α . Пример полученного графа см. на рисунке 2.1.

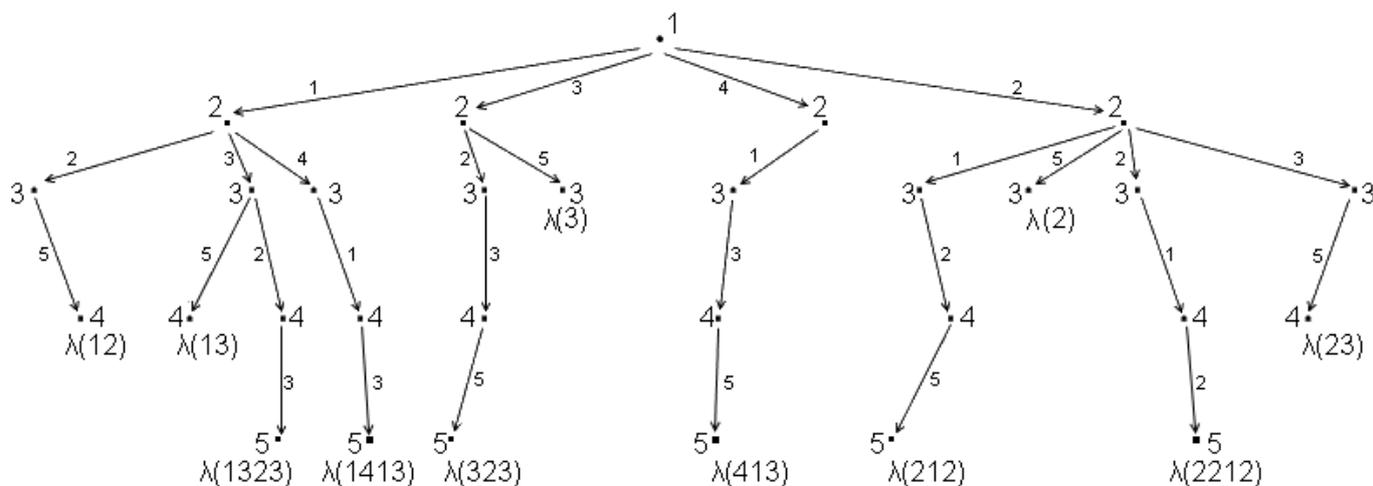


Рис. 2.1. Граф, построенный в лемме 9 на этапе отождествления вершин для $k = 4, n = 4, V = \{1323, 1413, 2212\}$

Сопоставим каждой вершине с номером i , где $i \in N_n$, переключатель $g_i(x)$. Получим, что в каждую вершину α , в которую ведет ребро с номером из N_k , пройдут все те и только те запросы, префикс которых совпадает с $x(\alpha)$; в вершину, в которую из некоторой вершины α ведет ребро с номером $k + 1$, пройдет

только запрос $x(\alpha)$. Заметим, что концевыми вершинами дерева являются все те и только те вершины, которые обозначены $\lambda(v_i[j \dots n])$ для некоторых i и j , причем в каждую концевую вершину $\lambda(x)$ проходит только запрос x . Для каждой концевой вершины $\lambda(x)$ обозначим через $M(x)$ множество всех таких вхождений (v_i, j) , что $x = v_i[j \dots n]$. Очевидно, множества $M(x)$, соответствующие разным концевым вершинам, не пересекаются. При этом, из каждой вершины α схемно достижимы все те и только те вершины $\lambda(x)$, для которых $x(\alpha)$ является префиксом слова x . Так как ответ $J(x(\alpha))$ состоит из всех тех и только тех вхождений (v_i, j) , для которых $x(\alpha)$ является префиксом слова $v_i[j \dots n]$, то этот ответ представляет собой объединение множеств $M(x)$, соответствующих всем концевым вершинам, схемно достижимым из α .

Оценим число различных непустых ответов, соответствующих библиотеке. Каждому такому ответу J можно сопоставить вершину α , для которой $J(x(\alpha)) = J$, и либо α — концевая, либо из нее выходит не менее двух ребер. Действительно, выберем среди вершин β , для которых $J(x(\beta)) = J$, ту, номер которой максимален, и обозначим ее через α . Предположим, что из нее выходит ровно одно ребро. Пусть оно входит в некоторую вершину γ . Тогда $J(x(\gamma)) = J$, так как множества концевых вершин, схемно достижимых из α и γ , совпадают. При этом номер вершины γ больше номера вершины α , что противоречит выбору вершины α . Следовательно, число различных непустых ответов не превышает суммы числа концевых вершин дерева и числа его внутренних вершин, имеющих полустепень исхода не менее двух. Эта сумма, в свою очередь, не превышает $2pn - 1$, так как число концевых вершин дерева по построению не превышает pn . С учетом пустого ответа, библиотеке соответствует не более $2pn$ ответов.

Добавим в граф еще одну вершину, назовем ее $\kappa(\emptyset)$.

Для каждой пары (i_1, i_2) , $i_1, i_2 \in N_{n+1}$, $2 \leq i_1 < i_2$, рассмотрим каждую цепочку вершин, имеющих номера i_1, \dots, i_2 , обладающую следующими свойствами: а) из каждой вершины этой цепочки, за исключением вершины с номером i_2 , выходит только одно ребро, номер этого ребра не равен $k+1$, и это ребро входит в вершину, которая также принадлежит этой цепочке; б) добавить к этой цепочке какие-либо вершины так, чтобы свойство а) сохранилось, невозможно.

Обозначим каждую принадлежащую цепочке вершину через α_r , где r — номер этой вершины. Заметим, что $J(x(\alpha_{i_1})) = J(x(\alpha_{i_1+1})) = \dots = J(x(\alpha_{i_2}))$, так как из всех $\alpha_r, r \in \{i_1, i_1 + 1, \dots, i_2\}$ схемно достижимы одни и те же концевые вершины.

Обозначим $J(x(\alpha_{i_1}))$ как J . Рассмотрим произвольный запрос x , прошедший в вершину α_{i_1} , т.е. $x[1 \dots i_1 - 1] = x(\alpha_{i_1})$. Возможно несколько случаев:

1) x проходит в вершину α_{i_2} , т.е. слово $x(\alpha_{i_2})$ является префиксом слова x ;
 2) x является собственным префиксом слова $x(\alpha_{i_2})$, тогда, т.к. $x = x(\alpha_i)$ для некоторого $i \in \{i_1, \dots, i_2 - 1\}$, ответом на запрос x является J ;

3) x не проходит в вершину α_{i_2} и не совпадает с $x(\alpha_i)$ ни для какого $i \in \{i_1, \dots, i_2 - 1\}$. Тогда по построению графа получаем, что $J(x) = \emptyset$.

Удалим теперь вершины $\alpha_{i_1+1}, \dots, \alpha_{i_2-1}$ и выходящие из них ребра, а ребро, выходящее из α_{i_1} , направим в α_{i_2} , и присвоим ему номер 2. Если вершина $\lambda(x(\alpha_{i_2}))$ существует (это либо α_{i_2} , либо в нее из α_{i_2} входит ребро номер $k + 1$), дополнительно назовем ее $\kappa(J)$, иначе выберем новую вершину, назовем ее $\kappa(J)$ и проведем в нее из α_{i_2} ребро с номером $k + 1$. Проведем в $\kappa(J)$ из α_{i_1} ребро с номером 1. Кроме того, проведем из α_{i_1} в $\kappa(\emptyset)$ ребро номер 3. Сопоставим вершине α_{i_1} переключатель h_{v, i_1} , где $v = x(\alpha_{i_2})[i_1 \dots i_2 - 1]$. Получим, что в случае 1) x по-прежнему проходит в вершину α_{i_2} , а если x совпадает с $x(\alpha_{i_2})$ — то и в $\kappa(J)$; в случае 2) x проходит в вершину $\kappa(J)$, а в случае 3) — в вершину $\kappa(\emptyset)$.

Пусть все цепочки рассмотрены. Все вершины $\lambda(x)$, не обозначенные дополнительно как $\kappa(J)$ для некоторого ответа J , обозначим дополнительно как $\kappa(J(x))$. Из каждой вершины α , которой сопоставлен переключатель $g_i(x), i \in \{2, \dots, n\}$, и из которой не выходит ребро под номером $k + 1$, выпустим это ребро, и направим его в новую вершину, которую назовем $\kappa(J(x(\alpha)))$. Получим, что каждый запрос x , имеющий непустой ответ $J(x)$, пройдет в вершину $\kappa(J(x))$, и не пройдет ни в какую вершину $\kappa(J(x'))$, где $J(x) \neq J(x')$.

Выпустим из каждой вершины, которой сопоставлен переключатель $g_i(x), i \in \{1, \dots, n\}$, несколько ребер, чтобы из вершины выходило всего $k + 1$ ребер. Пронумеруем эти ребра так, чтобы из вершины выходило ребро с каждым номером из $\{1, \dots, k + 1\}$. Направим все новые ребра в вершину $\kappa(\emptyset)$.

Кроме того, направим в эту вершину из корня ребро номер $k + 1$.

Построенный граф не имеет ориентированных циклов, так как в каждую вершину, имеющую ненулевую полустепень исхода, входит не более одного ребра, причем в корень не входит ни одно ребро. Так как при построении ИГ не использовались предикаты, каждый запрос проходит ровно в одну из вершин графа, из которой не выходит ни одно ребро. Каждая такая вершина названа $\kappa(J)$ для некоторого ответа J , и уже показано, что если J непуст, то в $\kappa(J)$ проходят все запросы, имеющие ответ J , и только они. Получим, что все запросы, ответ на которые пуст (и только они), проходят в $\kappa(\emptyset)$. Таким образом, третье утверждение леммы доказано. Пример построенного графа см. на рисунке 2.2.

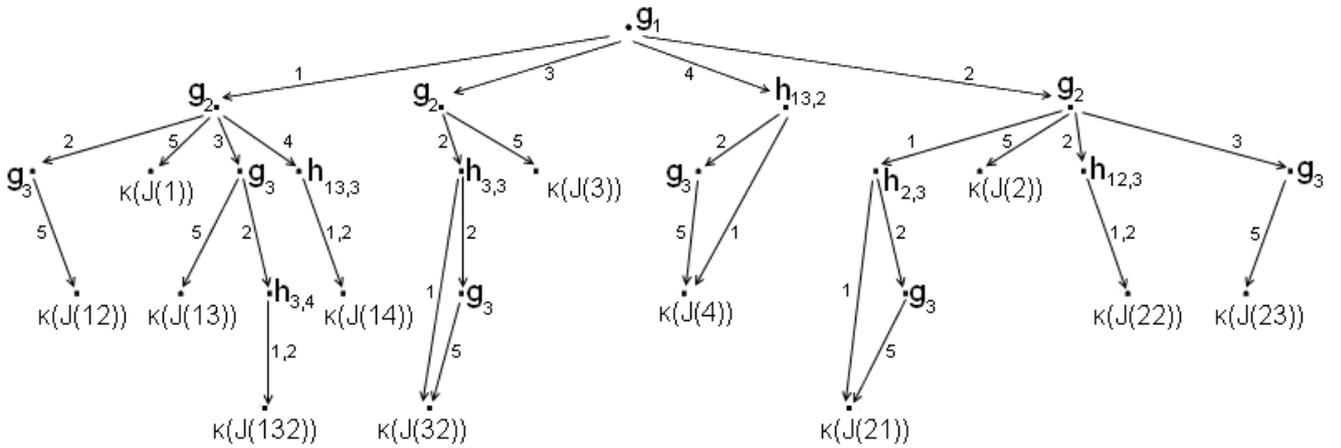


Рис. 2.2. Граф, построенный по лемме 9 для $k = 4, n = 4, V = \{1323, 1413, 2212\}$. Ребра, ведущие в вершину $\kappa(\emptyset)$, и сама эта вершина на рисунке не приведены

Обозначим построенный ИГ U' . Легко заметить, что при обработке каждого запроса x каждая его буква обрабатывается не более одного раза, и если $l(x) < n$, то еще не более одного раза обрабатывается буква $x[l(x) + 1]$. Каждая буква обрабатывается либо в шаге цикла функции $h_{v,i}(x)$, либо в функции $g_i(x)$, т.е. сложность обработки буквы не превышает 3. Поэтому время обработки запроса, т.е. время, за которое запрос попадет из корня в некоторую вершину $\kappa(J)$, не превышает $3e(x)$. Тем самым, второе утверждение леммы доказано.

Оценим объем графа U' . Заметим, что в U' из каждой вершины α , кроме корня и вершин с нулевой полустепенью исхода, выходит хотя бы одно

ребро, входящее в вершину $\kappa(J(x(\alpha)))$. С другой стороны, в каждую вершину $\kappa(J)$, $J \neq \emptyset$ входят ребра из не более чем двух вершин, причем если в $\kappa(J)$ входят ребра из двух вершин, то одной из этих вершин сопоставлен переключатель из множества G_1 , а другой — переключатель из множества G_2 . Сопоставив непустому ответу J все вершины, из которых в $\kappa(J)$ ведет ребро, получим, что из этих вершин выходит суммарно не более $k + 4$ ребер. Все вершины графа, которые мы не можем таким образом сопоставить какому-либо непустому ответу — корень и вершины с нулевой полустепенью исхода. Следовательно, $Q(U') \leq (k + 4)(2pn - 1) + k + 1 < (k + 4)2pn$. Лемма доказана. \square

Лемма 10. Пусть $I_2 = \langle X, V, \rho_2 \rangle$ — задача поиска вхождений под слова в множестве слов, библиотеке V из p слов соответствует s различных ответов и ИГ U' без листьев над базовым множеством \mathcal{F}_2 обладает тем свойством, что для каждого ответа J найдется вершина $\kappa(J)$, из которой не выходит ни одно ребро и в которую попадают все те и только те запросы x , ответ на которые совпадает с J . Тогда граф U' можно, добавив не более $pn + s - 1$ ребер, достроить до ИГ U над базовым множеством \mathcal{F}_2 , решающего задачу поиска вхождений под слова в множестве слов, причем время обработки запроса x после попадания в вершину $\kappa(J(x))$ не будет превышать $2(|J(x)| - 1)t$.

Доказательство. Для каждого слова x , для которого ответ $J(x)$ непуст, сделаем следующее. Если $l(x) < n$, то для каждого $r \in N_k$, для которого $0 < |J(xr)| < |J(x)|$, выпустим из $\kappa(J(x))$ в $\kappa(J(xr))$ ребро с предикатом $f(x)$, если это ребро еще не проведено. Если найдется хотя бы одно вхождение (v_i, j) такое, что $x = v_i[j \dots n]$, то одно из этих вхождений сопоставим вершине $\kappa(J(x))$, а остальные — каким-либо новым вершинам, и в каждую из этих новых вершин проведем из $\kappa(J(x))$ ребро с предикатом $f(x)$.

Построенный граф назовем графом U .

Покажем, что подграф U_1 графа U , состоящий из всех добавленных вершин и ребер, а также всех вершин $\kappa(J)$, кроме $\kappa(\emptyset)$, представляет собой ориентированный лес. Пусть в некоторую вершину $\kappa(J)$ ведет ребро из вершины $\kappa(J_1)$ и из вершины $\kappa(J_2)$, $J_1 \neq J_2$. это значит, что найдутся такие запросы x_1 и x_2 , а также

буквы i_1 и i_2 , что $J_1 = J(x_1)$, $J_2 = J(x_2)$ и $J = J(x_1i_1) = J(x_2i_2)$. Рассмотрим какое-либо вхождение (v_i, j) из J . Получим, что как x_1i_1 , так и x_2i_2 являются префиксами слова $v_i[j \dots n]$. Не ограничивая общности, будем считать, что x_1i_1 — префикс x_2i_2 . Если $x_1i_1 = x_2i_2$, то $x_1 = x_2$, откуда $J_1 = J_2$, что противоречит определению J_1 и J_2 . Следовательно, x_1i_1 является префиксом x_2 . Очевидно, $|J(x_1i_1)| \geq |J(x_2)|$, т.е. $|J| \geq |J(x_2)|$, что противоречит построению графа. Мы получили, что в каждую вершину $\kappa(J)$ ведет не более одного ребра. В каждую добавленную вершину мы (по построению) провели ровно одно ребро. Кроме того, в этом графе нет циклов, так как все добавленные вершины являются конечными, и если из некоторой вершины $\kappa(J(x_1))$ идет ребро в какую-либо вершину $\kappa(J(x_1r))$, то по построению $|J(x_1r)| < |J(x_1)|$. Итак, граф является ациклическим, и в каждую его вершину входит не более одного ребра, поэтому этот граф — ориентированный лес.

Везде далее в список вершин, в которые запрос пройдет из $\kappa(J(x))$, будем включать и саму вершину $\kappa(J(x))$. Покажем, что запрос x пройдет из $\kappa(J(x))$ во все те и только те листья, которым сопоставлены удовлетворяющие этому запросу вхождения. Для запросов, которым не удовлетворяет ни одно вхождение, это очевидно, т.к. вершине $\kappa(\emptyset)$ не сопоставлено ни одно вхождение, и ни одно ребро не выходит из этой вершины. Докажем теперь это для всех запросов, которым удовлетворяет хотя бы одно вхождение. Проведем доказательство по индукции по длине запроса x .

Замечание 1. Так как граф U_1 — ориентированный лес, причем всем его ребрам сопоставлены предикаты, тождественно равные 1, то любые два запроса x_1 и x_2 , прошедшие в какую-либо вершину $\kappa(J)$, пройдут из нее во все вершины, схемно достижимые из $\kappa(J)$.

Замечание 2. Запросу x удовлетворяет каждое вхождение (v_i, j) , для которого $x = v_i[j \dots n]$, а также каждое вхождение, которое удовлетворяет запросу xi , где $i \in \{1, \dots, k\}$. Никакие другие вхождения запросу x не удовлетворяют.

База индукции: $l(x) = n$. Так как слову x может удовлетворять лишь вхождение $(x, 1)$, то это вхождение при построении U было приписано вершине $\kappa(J(x))$. Кроме того, из $\kappa(J(x))$ не были проведены никакие ребра, т.е. запрос не пройдет из листа $\kappa(J(x))$ ни в какие другие листья.

Пусть мы уже доказали для всех x длины не менее $q + 1$. Докажем для всех x длины q , где $q \geq 1$. Возможно два случая:

1) Найдется слово x' такое, что $l(x') > q$ и $J(x') = J(x)$. Тогда по предположению индукции запрос x' пройдет из $\kappa(J(x'))$ во все те и только те листья, которым сопоставлены удовлетворяющие этому запросу вхождения. Учитывая замечание 1 и то, что $J(x) = J(x')$, получаем, что предположение индукции верно и для x .

2) Для каждого запроса x' длины, превышающей q , $J(x) \neq J(x')$. Отсюда, в частности, следует, что для любого r из N_k выполнено $|J(x)| > |J(xr)|$. При построении из вершины $\kappa(J(x))$ были проведены ребра с предикатами, тождественно равными 1, ко всем вершинам $\kappa(J(xr))$, для которых $0 < |J(xr)| < |J(x)|$. Кроме того, из $\kappa(J(x))$ были проведены ребра с теми же предикатами ко всем, кроме одной, вершинам с записями (v_i, j) , для которых $x = v_i[j \dots n]$. Единственная такая вершина, к которой не было проведено ребро — сама $\kappa(J(x))$. По предположению индукции, запрос xr пройдет из $\kappa(J(xr))$ во все те и только те листья, которым сопоставлены удовлетворяющие этому запросу вхождения. Учитывая замечание 1, получим, что запрос x пройдет во все листья, которым сопоставлены вхождения, удовлетворяющие запросам xr , и во все листья, которым сопоставлены записи (v_i, j) , для которых $x = v_i[j \dots n]$; ни в какие другие листья запрос x не пройдет. Учитывая замечание 2, получаем, что для x верно предположение индукции.

Итак, мы доказали, что U решает задачу поиска вхождений подслова.

Вычислим число ребер в графе U_1 . Так как U_1 — лес, это число равно числу вершин за вычетом числа компонент связности. В этом графе $s - 1$ либо s вершин $\kappa(J)$, соответствующих непустым ответам J (в зависимости от того, есть ли среди s ответов, соответствующих библиотеке, пустой ответ). Кроме этих вершин, в U_1 есть новые вершины, каждой из которых сопоставлено некоторое вхождение. Одно и то же вхождение (v_i, j) не может быть сопоставлено более чем одной вершине, т.к. для него единственным образом определяется вершина $\kappa(J(v_i[j \dots n]))$, которой либо сопоставлено это вхождение, либо из которой идет ребро в новую вершину с этим вхождением. Следовательно, новых вершин не больше, чем различных вхождений, соответствующих библиотеке, т.е. pn . Итак,

мы получили, что в U_1 не более чем $pn + s - 1$ ребер.

Оценим сложность обработки запроса x после попадания запроса в вершину $\kappa(J(x))$, когда $J(x) \neq \emptyset$. Так как граф U_1 — ориентированный лес, то все вершины, в которые запрос пройдет из $\kappa(J(x))$, и выходящие из них ребра образуют ориентированное дерево, корнем которого является $\kappa(J(x))$. Назовем это дерево $U_1(J(x))$. Покажем, что каждая вершина этого дерева, которой не сопоставлено никакое вхождение, является внутренней, причем из нее выходит не менее двух ребер. Действительно, по построению это может быть только некоторая вершина $\kappa(J)$, причем ответ J непуст, так как в графе U_1 нет вершин, из которых в $\kappa(\emptyset)$ идет ребро. Пусть x' имеет максимальную длину среди всех запросов, ответом на которые является J . Получим, что нет ни одного соответствующего библиотеке вхождения (v_i, j) , для которого выполнено равенство $x' = v_i[j \dots n]$, так как одно из таких вхождений было бы сопоставлено $\kappa(J)$. Следовательно, согласно замечанию 2, J является объединением нескольких непустых ответов $\kappa(J(x'r))$, где $r \in N_k$. По определению x' ни один из этих ответов не совпадает с J , поэтому таких ответов не менее двух. По построению из $\kappa(J)$ в каждую из вершин $\kappa(J(x'r))$, где $i \in N_k$ и ответ $J(x'r)$ непуст, идет ребро. Итак, из $\kappa(J)$ выходит не менее двух ребер. Так как в ориентированном дереве, в котором полустепень исхода каждой внутренней вершины не меньше двух, количество внутренних вершин меньше количества концевых вершин, то в дереве $U_1(J(x))$ вершин, которым не сопоставлено никакое вхождение, меньше, чем вершин, которым сопоставлено какое-либо вхождение. Так как U решает задачу поиска вхождений подслова и каждое соответствующее библиотеке вхождение сопоставлено ровно одной вершине, в дереве $U_1(J(x))$ вхождение сопоставлено $|J(x)|$ вершинам, а всего в нем не более $2|J(x)| - 1$ вершин и $2|J(x)| - 2$ ребер. Итак, общая сложность обработки запроса x после его попадания в $\kappa(J(x))$ не превосходит $2(|J(x)| - 1)t$. \square

Лемма 11. Пусть $I_2 = \langle X, V, \rho_2 \rangle$, где $|V| = p$ — задача поиска вхождений подслова. Тогда найдется ИГ $U \in \mathcal{U}(I_2, \mathcal{F}_2)$ такой, что $Q(U) \leq (2k + 11)pn$ и для каждого $x \in X$ выполнено $T(U, x) \leq 3R(I_2, \mathcal{F}_2, x)$.

Доказательство. Согласно лемме 9, количество ответов, соответствующих V , не превышает $2pn$. Рассмотрим граф U' , построенный по лемме 9. По лемме 10

его можно достроить до ИГ U , решающего задачу $I_2 = \langle X, V, \rho_2 \rangle$. Объем полученного графа будет равен сумме объема графа U' и количества добавленных по лемме 10 ребер, т.е. $Q(U) \leq (k + 4)2pn + pn + 2pn - 1 < (2k + 11)pn$.

Время обработки произвольного запроса x графом U складывается из времени прохождения запроса из корня в вершину $\kappa(J(x))$ и времени обработки запроса x после попадания в эту вершину, т.е. $T(U, x) \leq 3e(x) + 2(|J(x)| - 1)t \leq 3R(I_2, \mathcal{F}_2, x)$. Лемма доказана. \square

Приведем пример графа, построенного по лемме 11. Пусть $k = 4, n = 4, V = \{1323, 1413, 2212\}$. На рисунке 2.3 изображен ИГ, решающий задачу $I_2 = \langle X, V, \rho_2 \rangle$.

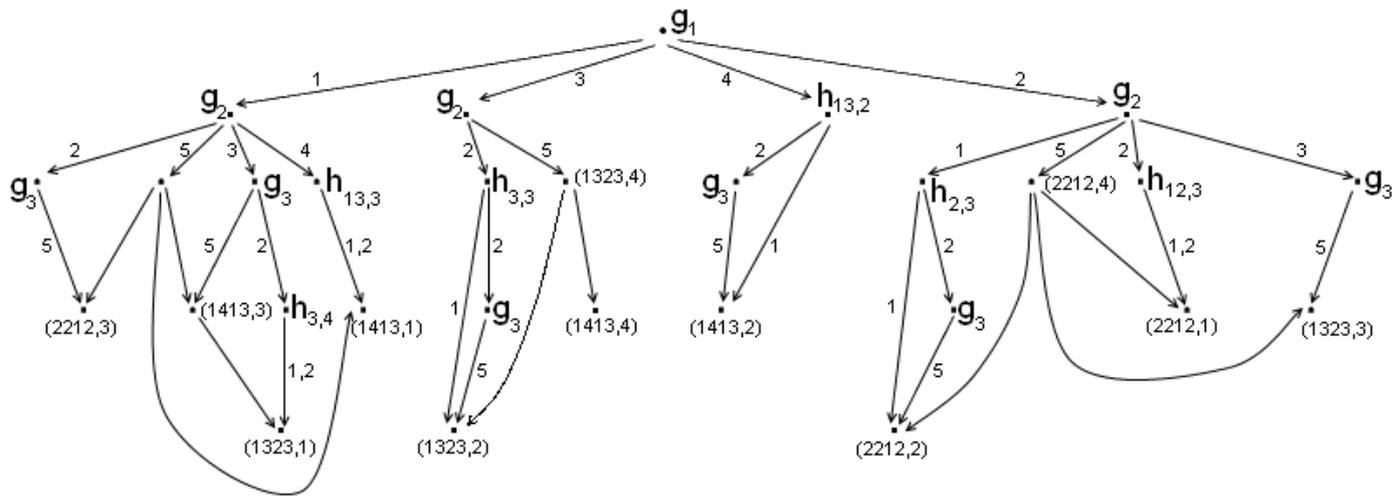


Рис. 2.3. Граф, решающий задачу поиска вхождений подслова в множестве слов для $k = 4, n = 4, V = \{1323, 1413, 2212\}$. Вершина $\kappa(\emptyset)$ и входящие в нее ребра на рисунке не приведены. Все ребра, у которых на рисунке нет номера, соответствуют предикату $f(x) \equiv 1$.

Перейдем к непосредственному доказательству теоремы 5.

Из определений $Q_2(p, n)$ и $Q_{fast}(p, n)$ следует, что $Q_2(p, n) \leq Q_{fast}(p, n)$. Согласно лемме 11, для любой библиотеки из p слов найдется ИГ $U \in \mathcal{U}_{fast}$ такой, что $Q(U) \leq (2k + 11)pn$. Из этого следует, что $Q_{fast}(p, n) \leq (2k + 11)pn$. Отсюда и из леммы 8 вытекает доказательство теоремы.

Глава 3

Поиск подслова с известной длиной в множестве слов

В главе 3 рассматривается задача поиска подслова в множестве слов при условии, что сложность вычисления длины запроса равна 1. Описанные в данной главе результаты опубликованы в [33].

§ 1 Определения

Введем определения, необходимые для настоящей главы. Пусть $W'_{k,n} = \bigcup_{s=1}^n N_k^s$ — множество слов длины не более n над алфавитом N_k ; в отличие от множества $W_{k,n}$, определенного в главе 1, в множестве $W'_{k,n}$ слово длины s представлено набором длины s .

В качестве множества запросов X' будем рассматривать множество $W'_{k,n}$, а в качестве множества записей — то же множество $Y = N_k^n$ слов длины n над алфавитом N_k , что и в главе 1. Отношение поиска введем аналогично тому, как это было сделано в главе 1, с учетом того, что в качестве множества запросов теперь выступает множество X' , а не X : $x\rho_1y \Leftrightarrow$ существует $i \in \mathbb{N} : x = y[i \dots l(x) + i - 1]$, $x \in X'$, $y \in Y$.

Будем рассматривать задачу информационного поиска $I_3 = \langle X', V, \rho \rangle$, где $V = \{v_1, v_2, \dots, v_p\}$, $V \subseteq Y$. Как и в предыдущих главах, множество V будем называть библиотекой. Назовем задачу I_3 задачей поиска подслова с известной длиной.

В настоящей главе будем использовать определение информационного графа, данное в главе 1. В качестве базового множества функций, используемых для построения информационных графов, решающих задачу поиска подслова

с известной длиной, будем использовать множество $\mathcal{F}_3 = \langle F_3, G_3 \rangle$. Множество предикатов F_3 состоит из предикатов, тождественно равных 1. Множество переключателей G_3 состоит из функции $l(x)$, функций $g'_i(x), i \in \mathbb{N}$, принимающих значение 1, если длина слова x больше i , и 2, если длина x меньше либо равна i , а также функций $g_i(x) = x[i], i \in \mathbb{N}$, причем если $i > l(x)$, то значение функции $g_i(x)$ не определено. Предикат, тождественно равный 1, и функции $g_i(x)$ использовались для построения информационных графов в главе 1 (с той разницей, что в главе 1 значение функции $g_i(x)$ при $l(x) < i$ было равно $k + 1$). Как и в главе 1, сложность вычисления каждой функции $g_i(x)$ полагается равной 1, а сложность вычисления предиката, тождественно равного 1, полагается равной некоторому числу t , причем $0 < t < 1$. Сложность вычисления функции $l(x)$ и функций $g'_i(x)$ полагается равной 1, т.е. неявно предполагается, что длина запроса подается на вход алгоритма, осуществляющему поиск, вместе с самим запросом, а потому не требуется перебирать буквы запроса, чтобы выяснить его длину.

Множество всех информационных графов над базовым множеством \mathcal{F}_3 , решающих задачу поиска вхождений подслова с известной длиной, обозначим через $\mathcal{U}(I_3, \mathcal{F}_3)$. Сложностью задачи I_3 при базовом множестве \mathcal{F}_3 и заданном объеме q назовем число $T(I, \mathcal{F}, q) = \min\{T(U) : U \in \mathcal{U}(I_3, \mathcal{F}_3), Q(U) \leq q\}$, если $\exists U \in \mathcal{U}(I_3, \mathcal{F}_3) : Q(U) \leq q$; в противном случае положим $T(I, \mathcal{F}, q) = \infty$.

Обозначим через $d(I_3, x)$ количество слов из библиотеки, содержащих x как подслово. Через $d(I_3)$ будем обозначать величину $\max_{x \in X'} d(I_3, x)$.

§ 2 Оценки сложности поиска подслова с известной длиной в множестве слов при ограничениях на объем памяти

Лемма 12. Пусть $I_3 = \langle X, V, \rho_3 \rangle$, где $|V| = p$ — задача поиска вхождений подслова с известной длиной в множестве слов, и r — такое натуральное число, что $k^{r-1}/(n - r + 1) < p \leq k^r/(n - r)$, если $p \leq k^{n-1}$, и $r = n$, если $p > k^{n-1}$. Тогда найдется информационный граф $U_1 \in \mathcal{U}(I_3, \mathcal{F}_3)$ такой, что $Q(U_1) < n + pn(n + 1)/2 + k^3p(n - r + 1)/(k - 1)^2 + k^2p(n - r + 1)(n - r)/(k - 1) + kp(n - r)(n - r + 1)(n - r + 2)/6$, и для каждого $x \in X$ выполнено $T(U_1, x) \leq$

$$l(x) + 1 + t(d(I_3, x) - 1).$$

Доказательство. Пусть $V = \{v_1, v_2, \dots, v_p\}$. Построим информационный граф U_1 , решающий задачу поиска подслова с известной длиной в множестве слов.

Пусть $s \in N_n$. Для каждого слова $v \in V$ рассмотрим каждое его подслово $w = v[a \dots b]$ длины s . Выберем $s + 1$ вершин, пронумеруем их от 1 до $s + 1$, а затем для каждого $r \in N_s$ проведем из вершины под номером r в вершину под номером $r + 1$ ребро, присвоив этому ребру номер $w[r]$. Вершину с номером $s + 1$ назовем $\lambda(w)$.

Теперь отождествим все вершины, имеющие номер 1. Далее последовательно для каждого натурального r от 1 до s сделаем следующее. Рассмотрим каждую вершину, имеющую номер r . Отождествим все ребра с одинаковым номером, выходящие из этой вершины, и вершины, в которые входят эти ребра (по построению, все отождествленные вершины будут иметь номер $r + 1$). Каждой вершине построенного графа, имеющей номер r , не превышающий s , сопоставим переключатель $g_r(x)$. Назовем полученный граф $U_{1,s}$. Вершину графа $U_{1,s}$, полученную отождествлением всех вершин с номером 1, обозначим $\beta_{1,s}$.

Построим граф U_1 , используя графы $U_{1,1}, \dots, U_{1,n}$. Выберем некоторую новую вершину, объявим ее корнем графа U_1 , и для каждого $i \in N_n$ проведем из нее ребро с номером i в вершину графа $U_{1,s}$.

Для каждой вершины $\lambda(w)$, если $d(I_3, w)$ (количество слов в библиотеке, содержащих подслово w) не меньше 2, выберем $d(I_3, w) - 1$ новых вершин и проведем в каждую из них ребро из вершины $\lambda(w)$. Далее каждое слово из библиотеки, содержащее подслово x , сопоставим вершине $\lambda(w)$ либо вершине, в которую из $\lambda(w)$ ведет ребро.

Выберем новую вершину. Обозначим ее γ .

Рассмотрим каждую вершину, которой сопоставлен переключатель $g_i(x)$. Для каждого $j \in N_k$, если из этой вершины не выходит ребро с номером j , проведем такое ребро из этой вершины в вершину γ .

Сопоставим корню полученного графа переключатель $l(x)$, а каждому ребру, выходящему из вершин $\lambda(w)$ — предикат, тождественно равный 1. На этом построение графа U_1 завершено.

Полученный граф является ориентированным деревом, так как в его ко-

рень не входит ни одного ребра, а в каждую из остальных вершин входит одно ребро.

Пример полученного графа см. на рисунке 3.1.

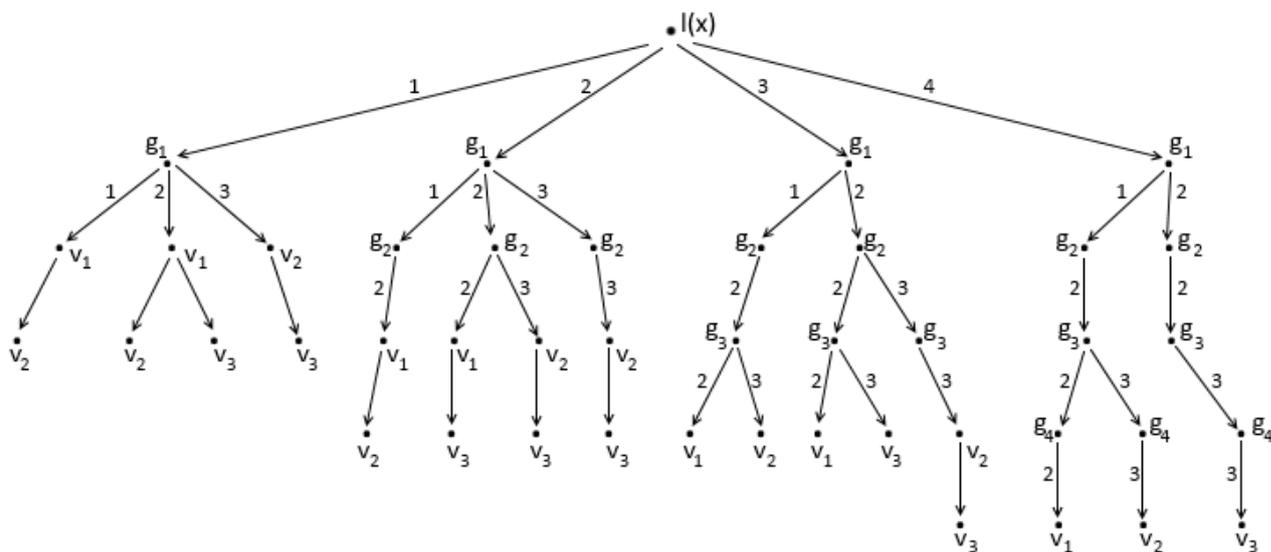


Рис. 3.1. Граф U_1 , построенный при $k = 4$ по библиотеке, состоящей из слов $v_1 = 1222$, $v_2 = 1233$, $v_3 = 2233$. На рисунке не изображены ребра, входящие в вершину γ , и сама эта вершина; также не изображены пометки у ребер, соответствующих предикату, тождественно равному 1.

Покажем, что информационный граф U_1 решает задачу поиска подслова известной длины в множестве слов.

Пусть запрос представляет собой слово x . Из корня графа U_1 он проходит в вершину $\beta_{1,l(x)}$. Заметим, что по построению графа $U_{1,l(x)}$ из вершины $\beta_{1,l(x)}$ запрос не может пройти ни в одну вершину $\lambda(w)$, для которой $l(w) \neq l(x)$. Кроме того, конкатенация номеров ребер цепи, ведущей из $\beta_{1,l(x)}$ в $\lambda(w)$, совпадает с w , а вершинам цепи последовательно сопоставлены переключатели $g_1(x), \dots, g_{l(w)}(x)$. Таким образом, запрос пройдет в вершину $\lambda(w)$ в том и только в том случае, если $x[i] = w[i]$ для всех $i \in N_{l(x)}$, т.е. если $x = w$. Вершине $\lambda(w)$ сопоставлено слово из библиотеки, содержащее подслово w , а все остальные слова, содержащее подслово w , сопоставлены вершинам, куда пройдет из вершины $\lambda(w)$ любой попавший в нее запрос. Следовательно, для любого слова из библиотеки имеем, что запрос пройдет в вершину, которой сопоставлено это слово, в том и только в том случае, если это слово удовлетворяет запросу. Таким образом, информационный граф U_1 решает задачу поиска подслова известной

длины в множестве слов.

Оценим сложность информационного графа U_1 .

Пусть x — запрос. Рассмотрим число $d(I_3, x)$ слов из библиотеки, удовлетворяющих этому запросу. Если $d(I_3, x) = 0$, т.е. в библиотеке V нет слов, удовлетворяющих запросу w , то при обработке этого запроса сначала произойдет одно вычисление переключателя $l(x)$, а затем — не более $\min(l(x), n)$ вычислений переключателей $g_i(x)$. Если же $d(I_3, x) > 0$, то при обработке запроса произойдет одно вычисление переключателя $l(x)$, $l(x)$ вычислений переключателей $g_i(x)$, и еще $d(I_3, x) - 1$ вычислений предиката, тождественно равного 1. Таким образом, $T(U_1, x) \leq l(x) + 1 + t(d(I_3, x) - 1)$.

Оценим объем информационного графа U_1 .

Ребра в графе U_1 выходят из корня, из вершин $\lambda(w)$, а также из вершин каждого графа $U_{1,s}$, $s \in N_n$, имеющих номера, не превышающие s . Из корня графа выходит n ребер, а число ребер, выходящих из вершин $\lambda(w)$, не превышает числа всех подслов слов из библиотеки V . Так как в слове длины n ровно $n - i + 1$ подслов (не обязательно различных) длины i , $i \in N_n$, то суммарно в одном слове $n(n + 1)/2$ подслов, а во всех словах из библиотеки — $pn(n + 1)/2$. Из каждой вершины графа $U_{1,s}$, $s \in N_n$, имеющей номера, не превышающий s , выходит k ребер.

Рассмотрим граф $U_{1,s}$ для произвольного $s \in N_n$. Обозначим через $N(j, s)$ число вершин в этом графе, имеющих номер $j \in N_s$. По построению графа $U_{1,s}$ это число не может превышать $p(n - s + 1)$ — количества подслов (не обязательно различных) длины s слов из библиотеки. В то же время, $N(1, s) = 1$, и $N(j, s) \leq kN(j - 1, s)$, $j \in N_s$, $j \geq 2$, так как в каждую вершину графа $U_{1,s}$ с номером $j \geq 2$ входит одно ребро из вершины с номером $j - 1$, а из вершины с номером $j - 1$ выходит k ребер; следовательно, $N(j, s) \leq k^{j-1}$.

Пусть $p \leq k^{n-1}$. Согласно условию леммы, в этом случае $r \in N_{n_1}$ — такое число, что $k^{r-1}/(n - r + 1) < p \leq k^r/(n - r)$.

Если $s \leq r$, то $p(n - s + 1) \geq p(n - r + 1) > k^{r-1}$, т.е. $k^{j-1} < p(n - s + 1)$, так как $j \leq s \leq r$.

Если $s > r$, то $p(n - s + 1) \leq p(n - r) \leq k^r$, т.е. $p(n - s + 1) \leq k^{j-1}$, если $j \geq r + 1$.

Тогда $N(j, s) \leq k^{j-1}$, $1 \leq j \leq r$, и $N(j, s) \leq p(n - s + 1)$, $j > r$.

Следовательно,

$$\begin{aligned} \sum_{s=1}^n \sum_{j=1}^s N(j, s) &\leq \sum_{s=1}^r \sum_{j=1}^s k^{j-1} + \sum_{s=r+1}^n \sum_{j=1}^r k^{j-1} + p \sum_{s=r+1}^n \sum_{j=r+1}^s (n - s + 1) = \\ &= \sum_{s=1}^r \frac{k^s - 1}{k - 1} + \sum_{s=r+1}^n \frac{k^r - 1}{k - 1} + p \sum_{s=r+1}^n (s - r)(n - s + 1) = \\ &= \frac{k(k^r - 1)}{(k - 1)^2} + \frac{k^r(n - r) - n}{k - 1} + \frac{p(n - r)(n - r + 1)(n - r + 2)}{6} < \\ &< \frac{k^2 p(n - r + 1)}{(k - 1)^2} + \frac{kp(n - r + 1)(n - r)}{k - 1} + \frac{p(n - r)(n - r + 1)(n - r + 2)}{6}. \end{aligned}$$

Пусть теперь $p > k^{n-1}$. Тогда $k^{j-1} < p(n - s + 1)$ для любых $s \in N_n, j \in N_s$,

и

$$\sum_{s=1}^n \sum_{j=1}^s N(j, s) \leq \sum_{s=1}^n \sum_{j=1}^s k^{j-1} = \frac{k(k^n - 1)}{(k - 1)^2} - \frac{n}{k - 1} < \frac{k^2 p}{(k - 1)^2}.$$

Таким образом, во всех случаях для объема графа U_1 справедливо

$$\begin{aligned} Q(U_1) &< n + \frac{pn(n + 1)}{2} + \frac{k^3 p(n - r + 1)}{(k - 1)^2} + \frac{k^2 p(n - r + 1)(n - r)}{k - 1} + \\ &+ \frac{kp(n - r)(n - r + 1)(n - r + 2)}{6}. \end{aligned}$$

Лемма доказана. □

Лемма 13. Пусть $I_3 = \langle X, V, \rho_3 \rangle$, где $|V| = p$ — задача поиска вхождений под слова с известной длиной в множестве слов, и $r \in N_{n-1} \cup \{0\}$ — такое число, что $k^r / (n - r + 1) < p \leq k^{r+1} / (n - r)$. Тогда найдется информационный граф $U_2 \in \mathcal{U}(I_3, \mathcal{F}_3)$ такой, что $Q(U_2) < pn(n + 1)/2 + k(k + 2)p(n - r + 1)/(k - 1) + (k + 2)p(n - r + 2)(n - r - 1)/2 + k$, и для каждого $x \in X$ выполнено $T(U_2, x) \leq \min(l(x), n - 1) + l(x) + t(d(I_3, x) - 1)$.

Доказательство. Построим информационный граф U_2 , решающий задачу поиска подслова с известной длиной в множестве слов.

Для каждого слова $v \in V$ рассмотрим каждый его суффикс $w = v[a \dots n]$. Обозначим длину этого суффикса через s . Обозначим через q число $2s + 1$, если $s < n$, и число $2s$, если $s = n$. Выберем q вершин и пронумеруем их от 1 до q , а затем для каждого $r \in N_s$ проведем ребро из вершины под номером $2r - 1$ в вершину под номером $2r$, присвоив этому ребру номер $w[r]$. Кроме того, для каждого $r \in N_{s-1}$ проведем ребро из вершины под номером $2r$ в вершину под номером $2r + 1$, присвоив этому ребру номер 1. Если $s < n$, то проведем ребро из вершины под номером $2s$ в вершину под номером $2s + 1$, также присвоив этому ребру номер 1.

Отождествим все вершины, имеющие номер 1. Объявим получившуюся вершину корнем информационного графа. Последовательно для каждого натурального r от 1 до q проделаем следующее. Рассмотрим каждую вершину, имеющую номер r . Отождествим все ребра с одинаковым номером, выходящие из этой вершины, и вершины, в которые входят эти ребра (по построению, все отождествленные вершины будут иметь номер $r + 1$).

Выберем новую вершину и обозначим ее через γ . Рассмотрим каждую вершину с номером $2i - 1$, $i \in N_n$, из которой выходит хотя бы одно ребро. Сопоставим этой вершине переключатель $g_i(x)$. Для каждого $j \in N_k$, если из рассматриваемой вершины не выходит ребро с номером j , проведем такое ребро из этой вершины в вершину γ .

Каждой вершине с номером $2i$, $i \in N_{n-1}$, сопоставим переключатель $g'_i(x)$. Проведем из нее ребро с номером 2 в какую-либо новую вершину. Обозначим эту вершину $\lambda(w)$, где w — конкатенация номеров ребер, выходящих из нечетных вершин цепи, ведущей из корня графа в эту вершину; по построению графа получаем, что w — подслово некоторого слова из библиотеки. Каждую вершину с номером $2n$ также обозначим через $\lambda(w)$. Для каждого подслова w вершина $\lambda(w)$ найдется, так как подслово $w = v[a \dots b]$ является префиксом слова $v[a \dots n]$, рассмотренного при построении информационного графа, и поэтому в графе есть такая вершина с переключателем $g'_i(x)$, что конкатенация номеров ребер, выходящих из нечетных вершин цепи, ведущей из корня графа в эту

вершину, совпадает с w . С другой стороны, слову w не может соответствовать более одной такой вершины с переключателем $g'_i(x)$, так как при построении графа эти вершины были отождествлены.

Для каждой вершины $\lambda(w)$, если $d(I_3, w)$ (количество слов в библиотеке, содержащих подслово w) не меньше 2, выберем $d(I_3, w) - 1$ новых вершин и проведем в каждую из них ребро из вершины $\lambda(w)$. Далее каждое слово из библиотеки, содержащее подслово w , сопоставим вершине $\lambda(w)$ либо вершине, в которую из $\lambda(w)$ ведет ребро. Каждому ребру, выходящему из вершин $\lambda(w)$, сопоставим предикат, тождественно равный 1. На этом построение графа U_2 завершено.

Граф U_2 является ориентированным деревом, так как в его корень не входит ни одного ребра, а в каждую из остальных вершин входит одно ребро.

Пример графа U_2 см. на рисунке 3.2.

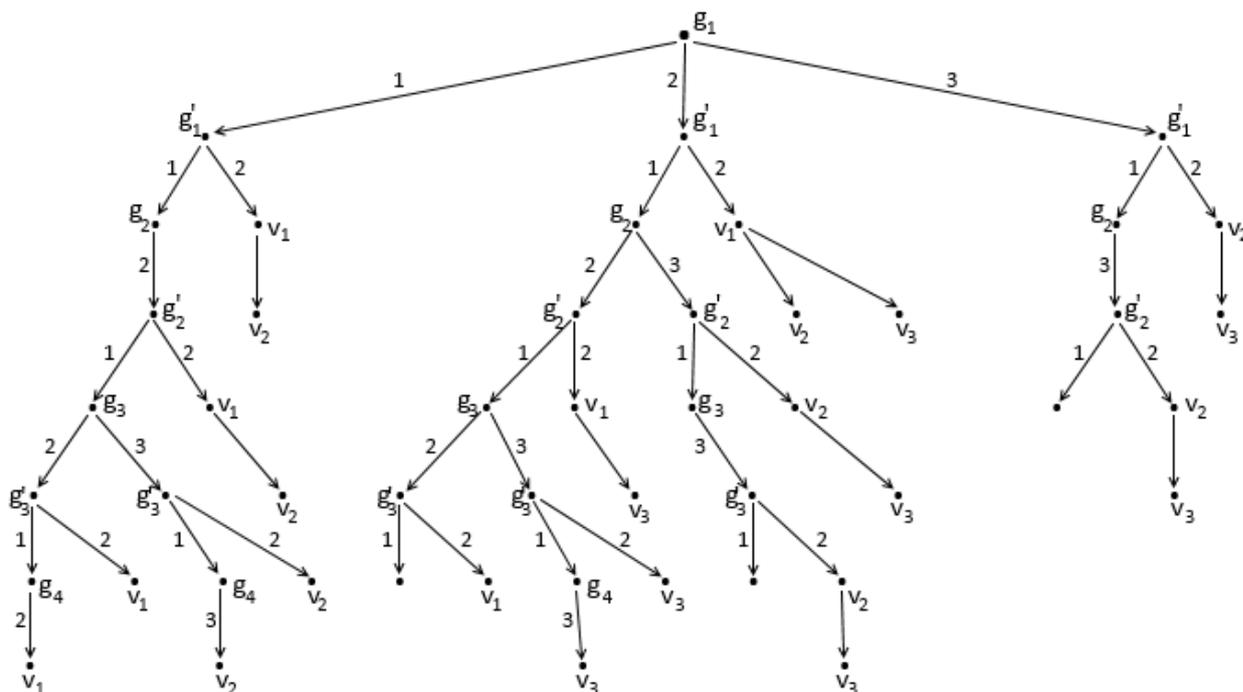


Рис. 3.2. Граф U_2 , построенный при $k = 4$ по библиотеке, состоящей из слов $v_1 = 1222$, $v_2 = 1233$, $v_3 = 2233$. На рисунке не изображены ребра, входящие в вершину γ , и сама эта вершина; также не изображены пометки у ребер, соответствующих предикату, тождественно равному 1.

Покажем, что информационный граф U_2 , решает задачу информационного поиска I_3 .

Рассмотрим цепь, ведущую из корня графа в какую-либо вершину $\lambda(w)$. Нечетным вершинам этой цепи сопоставлены переключатели $g_1(x), \dots, g_{l(w)}(x)$, а четным — переключатели $g'_1(x), \dots, g'_{l(w)-1}(x)$, а также переключатель $g'_{l(w)}(x)$, если $l(w) < n$. Ребро этой цепи, выходящее из вершины с переключателем $g_i(x)$, имеет номер $w[i]$, а ребро, выходящее из вершины с переключателем $g'_i(x)$ — номер 1, если $i < l(w)$, и номер 2, если $i = l(w)$. Таким образом, запрос x пройдет в вершину $\lambda(w)$ в том и только в том случае, если $l(x) > l(w) - 1$, $l(x) \leq l(w)$ и $x[i] = w[i] \forall i \in N_{l(w)}$, т.е. если $x = w$. Вершине $\lambda(w)$ сопоставлено слово из библиотеки, содержащее подслово w , а все остальные слова, содержащее подслово w , сопоставлены вершинам, куда пройдет из вершины $\lambda(w)$ любой попавший в нее запрос. Следовательно, для любого слова из библиотеки имеем, что запрос пройдет в вершину, которой сопоставлено это слово, в том и только в том случае, если это слово удовлетворяет запросу. Таким образом, информационный граф U_1 решает задачу поиска подслова известной длины в множестве слов.

Оценим сложность информационного графа U_2 .

Пусть x — запрос. Если $d(I_3, x) = 0$, т.е. в библиотеке V нет слов, удовлетворяющих запросу x , то этот запрос пройдет либо в вершину γ , либо в вершину с нечетным номером, из которой не выходит ни одно ребро. В первом случае, так как в вершину γ запрос может попасть только из вершины с номером $2s - 1$, $s \in N_n$, при обработке запроса будут вычислены переключатели $g_1(x), \dots, g_s(x)$ и $g'_1(x), \dots, g'_{s-1}(x)$, при этом, $g'_{s-1}(x) = 1$, т.е. $l(x) > s - 1$. Во втором случае для некоторого $s \in N_{n-1}$ будут вычислены переключатели $g_1(x), \dots, g_s(x)$ и $g'_1(x), \dots, g'_s(x)$, причем $g'_s(x) = 1$, т.е. $l(x) > s$. Таким образом, всего будет вычислено не более $l(x)$ переключателей $g_i(x)$ и не более $l(x) - 1$ переключателей $g'_i(x)$, т.е. если $d(I_3, x) = 0$, то $T(U_1, x) \leq 2l(x) - 1$.

Если $d(I_3, x) > 0$, то в графе есть вершина $\lambda(x)$, и запрос пройдет в эту вершину. Перед попаданием запроса в вершину $\lambda(x)$ будут вычислены переключатели $g_1(x), \dots, g_{l(x)}(x)$ и $g'_1(x), \dots, g'_{l(x)-1}(x)$. Если $l(x) < n$, то, кроме того, будет вычислен переключатель $g'_{l(x)}(x)$. После попадания запроса в вершину $\lambda(x)$ произойдет еще $d(I_3, x) - 1$ вычислений предиката, тождественно равного 1. Таким образом, если $d(I_3, x) > 0$, то $T(U_1, x) = \min(l(x), n - 1) + l(x) + t(d(I_3, x) - 1)$.

Оценим объем информационного графа U_2 .

Ребра в графе U_2 выходят из вершин $\lambda(w)$, из вершин, которым сопоставлены переключатели $g_i(x), i \in N_n$, а также из вершин, которым сопоставлены переключатели $g'_i(x), i \in N_{n-1}$. Суммарное число ребер, выходящих из вершин $\lambda(w)$, не превышает числа всех подслов слов из библиотеки V . Так как в слове длины n ровно $n - i + 1$ подслов (не обязательно различных) длины $i, i \in N_n$, то суммарно в одном слове $n(n + 1)/2$ подслов, а во всех словах из библиотеки — $pn(n + 1)/2$ подслов. Из каждой вершины с переключателем $g_i(x)$ выходит k ребер, а из каждой вершины с переключателем $g'_i(x)$ — 2 ребра.

Обозначим через $N(j)$ число вершин в графе U_2 , которым сопоставлен переключатель $g_j(x), j \in N_n$. Кроме того, через $N'(j)$ обозначим число вершин в графе, которым сопоставлен переключатель $g'_j(x), j \in N_n$. Заметим, что $N(j + 1) \leq N'(j)$ при $j \in N_{n-1}$, так как в каждую вершину с переключателем $g_{j+1}(x), j \in N_{n-1}$, входит одно ребро из вершины с переключателем $g'_j(x)$, и из каждой вершины с переключателем $g'_j(x), j \in N_{n-1}$ ведет не более одного ребра в вершину с переключателем $g_{j+1}(x)$. Кроме того, $N'(j) \leq kN(j), j \in N_{n-1}$, так как в каждую вершину с переключателем $g'_j(x), j \in N_{n-1}$ входит одно ребро из вершины с переключателем $g_j(x)$, и из каждой вершины с переключателем $g_j(x), j \in N_{n-1}$ ведет не более k ребер в различные вершины с переключателем $g_{j+1}(x)$. Таким образом, $N'(j + 1) \leq kN'(j), j \in N_{n-1}$. Так как $N'(1) \leq k$, то $N(j) \leq k^j, j \in N_n$.

Из каждой вершины с переключателем $g'_j(x), j \in N_{n-1}$ ведет ребро номер 2 в некоторую вершину $\lambda(w)$, где w — подслово длины j некоторого слова из библиотеки, причем в вершину $\lambda(w)$ не входят никакие другие ребра. Следовательно, $N(j)$ не может превышать суммарного количества подслов длины j (не обязательно различных) слов из библиотеки. У каждого слова из библиотеки $n - j + 1$ подслов длины j , поэтому $N'(j) \leq p(n - j + 1)$.

Пусть $r \in N_{n-1} \cup \{0\}$ — такое число, что $k^r / (n - r + 1) < p \leq k^{r+1} / (n - r)$.

Если $j \leq r$, то $p(n - j + 1) \geq p(n - r + 1) > k^r \geq k^j$. Если $j > r$, то $p(n - j + 1) \leq p(n - r) \leq k^{r+1} \leq k^j$. Тогда $\min(p(n - j + 1), k^j) = k^j, 1 \leq j \leq r$, и $\min(p(n - j + 1), k^j) = p(n - j + 1), r < j \leq n - 1$.

Следовательно,

$$\begin{aligned}
\sum_{j=1}^{n-1} N'(j) &\leq \sum_{j=1}^r k^j + \sum_{j=r+1}^{n-1} p(n-j+1) = \\
&= \frac{k^{r+1} - k}{k-1} + \frac{p(n-r+2)(n-r-1)}{2} < \\
&< \frac{kp(n-r+1)}{k-1} + \frac{p(n-r+2)(n-r-1)}{2}.
\end{aligned}$$

Тогда для объема графа U_2 справедливо

$$\begin{aligned}
Q(U_2) &\leq \frac{pn(n+1)}{2} + 2 \sum_{j=1}^{n-1} N'(j) + k \sum_{j=1}^n N(j) < \\
&< \frac{pn(n+1)}{2} + \frac{k(k+2)p(n-r+1)}{k-1} + \frac{(k+2)p(n-r+2)(n-r-1)}{2} + k.
\end{aligned}$$

Таким образом, доказательство леммы завершено. \square

Лемма 14. Пусть $k \geq 3$ и в библиотеке $V \subseteq N_k^n$ из p слов, где $p \leq (k-1)^n$, каждая буква каждого слова отлична от k . Тогда для любого ИГ $U \in \mathcal{U}(I_3, \mathcal{F}_3)$, где $I_3 = \langle X, V, \rho_3 \rangle$, и для каждого запроса $x \in X$, которому удовлетворяет хотя бы одно слово из библиотеки, выполнено неравенство $T(U, x) \geq \min(l(x) + 1, n) + t \cdot (d(I_3, x) - 1)$.

Доказательство данной леммы, в целом, аналогично доказательству леммы 1 из главы 1. Тем не менее, так как задача поиска подслова с известной длиной в множестве слов отличается от задачи поиска подслова в множестве слов, и так как базовые множества функций, рассматриваемые в лемме 14 из настоящей главы и в лемме 1 из главы 1, также отличаются, приведем доказательство леммы 14 полностью.

Доказательство. Рассмотрим произвольную библиотеку V , удовлетворяющую условиям леммы, и произвольный ИГ $U \in \mathcal{U}(I_3, \mathcal{F}_3)$, где $I_3 = \langle X, V, \rho_1 \rangle$. Покажем сначала, что если $d(I_3, x) \geq 1$, то $T(U, x) \geq l(x)$. Рассмотрим какую-либо цепь C , по которой запрос проходит из корня в лист с записью v , удовлетворяющей запросу (назовем этот лист γ), причем ни одна вершина этой цепи, кроме

γ , не является листом. Пусть существует такое $i \in \mathbb{N}$, $1 \leq i \leq l(x)$, что ни одной из вершин C , исключая γ , не сопоставлен переключатель $g_i(x)$. Тогда заменим $x[i]$ на k . Получившийся запрос также будет проходить в лист с записью v . Мы имеем противоречие с тем, что ни в одном слове из V нет буквы k , а следовательно, и любого подслова, содержащего букву k . Получим, что для любого $i \in N_{l(x)}$ найдется вершина цепи, не совпадающая с γ и такая, что ей сопоставлен $g_i(x)$. Значит, $T(U, x) \geq l(x)$.

Далее, пусть $l(x) < n$. Предположим, что вершинам цепи C , кроме γ , сопоставлены только переключатели $g_i(x), i \in N_{l(x)}$. Рассмотрим запрос xk . Этот запрос пройдет в вершину γ , и мы снова получим противоречие с тем, что ни в одном слове из V нет буквы k . Итак, найдется вершина цепи C , которой сопоставлен переключатель $l(x)$ либо переключатель $g'_j(x)$, где j — некоторое натуральное число, не превышающее $n - 1$. Следовательно, $T(U, x) \geq l(x) + 1$.

Пусть $d(I_1, x) > 1$. Это возможно только тогда, когда $l(x) < n$. Для каждого отличного от γ листа, в который проходит запрос x , выберем ребро, по которому x входит в этот лист. Так как никакому листу не может быть сопоставлено более одной записи, все эти ребра разные, причем если некоторые из этих ребер исходят из одной вершины, то это предикатные ребра. Кроме того, ни одно из этих ребер не принадлежит C . Значит, для прохода запроса по таким ребрам будет вычислено не менее $d(I_1, x) - 1$ функций. Кроме того, при обработке запроса будет вычислено не менее $l(x) + 1$ указанных ранее переключателей. Значит, $T(U, x) \geq l(x) + 1 + t \cdot (d(I_1, x) - 1)$. Итак, мы показали, что если $d(I_1, x) \geq 1$, то $T(U, x) \geq \min(l(x) + 1, n) + t \cdot (d(I_3, x) - 1)$, что и требовалось. \square

Перейдем к непосредственному доказательству теоремы 6.

По лемме 14 для произвольного информационного графа $U \in \mathcal{U}(I_3, \mathcal{F}_3)$ получим, что если запросу x удовлетворяет хотя бы одно слово из библиотеки, то $T(U, x) \geq \min(l(x) + 1, n) + t \cdot (d(I_3, x) - 1)$. Следовательно, $T(I_3, \mathcal{F}_3) \geq n + t \cdot (d(I_3) - 1)$.

По лемме 12 существует такой информационный граф $U_1 \in \mathcal{U}(I_3, \mathcal{F}_3)$, что для каждого запроса x выполнено $T(U_1, x) \leq l(x) + 1 + t(d(I_3, x) - 1)$. Следовательно, $T(U_3) \leq n + 1 + t \cdot (d(I_3) - 1)$. Согласно лемме 12, $Q(U_1) < n + pn(n+1)/2 +$

$k^3p(n-r+1)/(k-1)^2+k^2p(n-r+1)(n-r)/(k-1)+kp(n-r)(n-r+1)(n-r+2)/6$, где r — такое натуральное число, что $k^{r-1}/(n-r+1) < p \leq k^r/(n-r)$, если $p \leq k^{n-1}$, и $r = n$, если $p > k^{n-1}$. Для произвольного $\epsilon > 0$ при $n \rightarrow \infty$, $p \rightarrow \infty$ получим, что $Q(U_1) < (1 + \epsilon)pn^3k/6$. Таким образом, $T(I_3, \mathcal{F}_3, q) \leq n + 1 + t \cdot (d(I_3) - 1)$, если $q \geq (1 + \epsilon)pn^3k/6$.

По лемме 13 существует такой информационный граф $U_2 \in \mathcal{U}(I_3, \mathcal{F}_3)$, что для каждого запроса x выполнено $T(U_2, x) \leq \min(l(x), n-1) + l(x) + t(d(I_3, x) - 1)$. Следовательно, $T(U_3) \leq 2n - 1 + t \cdot (d(I_3) - 1)$. Согласно лемме 13, $Q(U_2) < pn(n+1)/2 + k(k+2)p(n-s+1)/(k-1) + (k+2)p(n-s+2)(n-s-1)/2 + k$, где s — такое натуральное число, что $k^s/(n-s+1) < p \leq k^{s+1}/(n-s)$. Для произвольного $\epsilon > 0$ при $n \rightarrow \infty$, $p \rightarrow \infty$ получим, что $Q(U_2) < (1 + \epsilon)pn^2(k+3)/2$. Таким образом, $T(I_3, \mathcal{F}_3, q) \leq 2n - 1 + t \cdot (d(I_3) - 1)$, если $q \geq (1 + \epsilon)pn^2(k+3)/2$.

Теорема 6 доказана.

Глава 4

Построение модели нормативных актов

В настоящей главе рассматривается задача построения модели нормативно-правового акта по его тексту. Описаны этапы построения модели нормативно-правового акта. Результаты, изложенные в данной главе, опубликованы в [34, 35, 36, 37].

§ 1 Постановка задачи

Рассмотрим ориентированный граф, каждой вершине которого сопоставлена некоторая процедура одного из следующих видов:

1) получение значения из конкретного поля в памяти. Считается, что это значение передается по всем ребрам, выходящим из такой вершины;

2) запись значения в конкретное поле в памяти. В такую вершину должно вести единственное ребро. Считается, что записываемое значение поступает в данную вершину по этому ребру;

3) вычисление некоторой арифметической функции одного или двух аргументов, логической функции одного или двух аргументов либо унарного или бинарного отношения. В такую вершину должно входить столько ребер, сколько аргументов у функции или отношения. По каждому из этих ребер в вершину поступает значение соответствующего аргумента функции (отношения). Значение функции (отношения) передается по всем ребрам, выходящим из этой вершины.

4) выбор одного значения из нескольких. В эту вершину для некоторого натурального числа $m \geq 2$ должно вести $m + 1$ ребро. Эти ребра должны быть пронумерованы числами от 0 до m . По ребру с номером m в вершину поступает число $i \in Z, 0 \leq i \leq m - 1$. По каждому ребру, выходящему из

вершины, передается значение, поступившее в вершину по ребру с номером i .

Будем относить вершину к i -му виду вершин, $i \in \{1, 2, 3, 4\}$, если ей сопоставлена процедура i -го вида.

Из вершин графа 2-го вида выделим одну.

Этому графу сопоставляется процедура его вычисления: для выделенной вершины вычисляется значение, поступающее в нее по единственному входящему ребру. Это значение вычисляется с помощью процедуры, сопоставленной вершине, из которой выходит соответствующее ребро.

Для любой вершины 1-го, 3-го, 4-го вида вычисление значения, передаваемого по ребрам, выходящим из этой вершины, происходит следующим образом.

Для вершины 1-го вида это значение берется из соответствующего поля в памяти.

Для вершины 3-го вида сначала вычисляется значение, поступающее в вершину по одному из входящих в нее ребер. Если этого недостаточно для вычисления значения функции (отношения), то вычисляется значение, поступающее в вершину по другому ребру. В любом случае, дальше вычисляется значение функции (отношения) на полученных значениях аргументов, это значение и передается по выходящим из вершины ребрам.

Для вершины 4-го вида сначала вычисляется число i , поступающее в вершину по входящему в него ребру с максимальным номером, затем вычисляется значение, поступающее в вершину по ребру номер i . Это значение и передается по выходящему из вершины ребру.

Если результат процедуры вычисления графа при любых исходных значениях объектов совпадает с результатом вычисления соответствующего графу объекта согласно тексту закона, назовем этот граф моделью вычисления объекта. Граф, содержащий в себе в качестве подграфа модель вычисления любого объекта из текста закона, назовем моделью закона.

Будем считать, что вершине может соответствовать некоторое слово или фраза, объясняющие значение этой вершины.

Таким образом, задачу построения алгоритма вычисления любого описанного в законе объекта можно свести к задаче построения модели закона.

Решение этой задачи осуществляется в несколько этапов.

Первым этапом является создание по каждому предложению рассматриваемого текста его синтаксического дерева (дерева зависимостей) — дерева связей между словами в предложении. Определение дерева зависимостей см., например, на сайте Национального корпуса русского языка [22].

Большая часть парсеров — программ, осуществляющих синтаксический анализ — основывается либо на машинном обучении на синтаксически размеченных корпусах [19], либо на использовании строго определенных правил, позволяющих находить связи между словами [23].

Для программ первого типа необходима большая база синтаксически размеченных текстов (то есть, предложений, для которых синтаксический граф уже построен). Для русского языка можно выделить две базы синтаксически размеченных текстов — это Национальный корпус русского языка (доступ к этому корпусу ограничен) и проект Universal Dependencies [21]. Проект содержит корпус для более чем 60 языков; цель проекта — разработка единой модели синтаксической разметки для разных языков [20]. Корпус русского языка проекта Universal Dependencies содержит более 66000 предложений, и его можно свободно использовать для обучения парсеров.

Что касается программ, работающих на основе правил, то они не требуют базы данных синтаксически размеченных текстов. Тем не менее, создание набора правил, который позволял бы проводить синтаксический анализ произвольных предложений — сложная задача. Одной из моделей языка, используемых программами, осуществляющими обработку естественного языка на основе правил, является модель «Смысл \Leftrightarrow Текст» (Мельчук [24]). Вводя эту модель, Мельчук постулирует, что естественный язык — это преобразователь из текста в смысл и обратно. Мельчук строит формальную модель языка, состоящую из нескольких уровней (в т.ч., семантического и синтаксического), и вводит наборы правил, с помощью которых должен совершаться переход обрабатываемого представления текста на другой уровень. В идеале эти наборы правил должны обеспечивать переход от текста к его смыслу и обратно. Строго описанные правила оказалось удобно использовать в различных программах, осуществляющих обработку естественного языка (не обязательно русского), например, для автоматической генерации текста [25].

Вероятно, наиболее развитой системой, осуществляющей обработку русского языка, в настоящее время является Abbyy Compeno. Она проводит синтаксический и семантический анализ текста [26] и используется в программах, извлекающих информацию из неструктурированного текста, и программах, проводящих классификацию документов. Доступ к этой системе ограничен.

У задачи автоматического построения по тексту закона схем вычисления значений упомянутых в законе объектов есть особенности, затрудняющие использование парсеров, имеющих в свободном доступе. Первая из этих особенностей состоит в том, что при переводе с естественного языка на формальный язык логики предикатов достаточно одной неправильно определенной связи между словами в каком-либо предложении, чтобы формула, вычисляющая значение некоторого объекта, оказалась неверной. Другая особенность — наличие в тексте большого количества длинных сложных предложений, на которых парсеры делают значительно больше ошибок, чем на простых предложениях.

Таким образом, возникает необходимость в надстройке, которая исправляла бы ошибки парсера. Эта надстройка должна добавлять связи между словами, не обнаруженные парсером, и удалять неверно построенные связи. Число ошибок, совершаемых парсерами при разборе текста положения по бухгалтерскому учету ПБУ 6/01 [28], оказалось достаточно велико, чтобы надстройка представляла собой практически полноценный парсер. В результате автором работы было принято решение отказаться от использования парсеров, имеющих в свободном доступе, и создать собственный парсер. Он построен на основе правил, так как такой парсер значительно проще контролировать, чем парсер, созданный с помощью машинного обучения. При создании парсера учитывался тот факт, что в работе рассматриваются не произвольные тексты, а тексты нормативно-правовых актов, касающихся бухгалтерского учета, что значительно упростило создание необходимого набора правил.

После синтаксического разбора по каждому синтаксическому графу строится формула логики предикатов, которая определяет условия, накладываемые на рассматриваемые в тексте объекты. Наконец, по всем формулам вместе создается модель закона. Как и синтаксический анализ, построение формул логики предикатов и модели закона происходит на основе строго определенных

правил.

§ 2 Построение связного синтаксического графа предложения

Результаты, изложенные в данном параграфе, опубликованы в [34, 35].

Синтаксическое отношение — это отношение между парой слов предложения, причем одно из слов является главным в этом отношении, а другое — зависимым. Каждое синтаксическое отношение в зависимости от частей речи участвующих в отношении слов, их морфологических характеристик и т.д., может быть отнесено к определенному классу синтаксических отношений.

На вход алгоритму, осуществляющему синтаксический анализ, поступает последовательность *лексем*. Лексема, в свою очередь, представляет собой последовательность символов. Это может быть слово, число, знак препинания. Текст предложения разбивается на последовательность лексем в процессе *лексического анализа*.

Помимо последовательности лексем, на вход синтаксическому анализу поступает последовательность *токенов*. Токен создается для каждого слова предложения в процессе морфологического анализа и представляет собой тройку, в которую входят: само слово; *лемма* — каноническая форма слова (например, для существительного это будет то же слово, но в именительном падеже и единственном числе); набор *морфологических характеристик* (для существительного это род, падеж, число и т.д., для глагола это вид, время и т.д.).

Выходом синтаксического анализа является *синтаксическое дерево*, также называемое *деревом зависимостей*. Это ориентированное дерево. Каждой его вершине сопоставлен токен. Дуга в синтаксическом дереве ведет из вершины А в вершину В тогда и только тогда, когда сопоставленные этим вершинам слова связаны в русском языке синтаксическим отношением, причем главным в этом отношении является слово, соответствующее вершине А. Дуге при этом сопоставлено название отношения.

В данной работе для синтаксического анализа предлагается использовать (возможно, в несколько упрощенном виде) подход, применяемый А.С. Подколзиным для автоматического решения различных математических задач [38].

В применении к синтаксическому анализу этот подход состоит в следующем. Рассматривается список *правил*, которые позволяют находить синтаксические связи между словами. Для каждого токена и каждого правила проверяется, применимо ли это правило к данному токену; если да, то создается продиктованное этим правилом синтаксическое отношение.

Каждое правило состоит из трех частей. Первая часть проверяет, подходит ли слово для этого правила: обладает ли оно нужным набором морфологических характеристик. В большинстве случаев значение леммы не проверяется, однако есть и правила, которые работают с конкретными леммами. В тех случаях, когда целью правила является построения синтаксического отношения, в котором рассматриваемое слово было бы зависимым, проверяется также, что слово еще не является зависимым ни в каком построенном синтаксическом отношении. Объясняется эта проверка просто: каждое слово может входить в какое угодно число синтаксических отношений в качестве главного, но лишь в одно — в качестве зависимого.

Вторая часть заключается в поиске слова, которое может входить в синтаксическое отношение с рассматриваемым словом. В некоторых правилах ищется не одно слово, а несколько, притом таких, что каждое из них могло бы образовывать синтаксические отношения либо с другим найденным словом, либо с рассматриваемым словом.

Наконец, третья часть строит синтаксические отношения между рассматриваемым словом и найденными словами. В том случае, если слово прошло проверку в первой части правила, и для него были найдены подходящие слова во второй части правила, будем говорить, что правило *применимо* к слову. Таким образом, если правило применимо к слову, то в результате применения правила к этому слову строится одно или несколько новых синтаксических отношений.

Надо заметить, что описанный в работе алгоритм включает в себя элементы семантического анализа — это видно по названию некоторых отношений, по использованию списков существительных, формируемых исходя из смыслового значения этих существительных, и по нескольким другим признакам. Дело в том, что с учетом конечной цели создаваемого алгоритма, частью которого

является алгоритм синтаксического анализа, определенную работу по семантическому анализу удобно выполнять одновременно с синтаксическим анализом.

Для большей части создаваемых алгоритмом синтаксических отношений обозначения классов, к которым относятся эти отношения, взяты из [23], но для некоторых отношений используются специальные названия, принятые только в данной работе: это позволяет облегчить работу с синтаксически разобранным предложением на следующих этапах.

В главе подробно описаны правила, с помощью которых алгоритм создает синтаксические отношения.

Прежде чем перейти к описанию правил, заметим, что на слова, которые могли бы образовывать синтаксические отношения друг с другом, накладываются определенные ограничения. Если некоторая часть предложения заключена в круглые скобки, то никакое слово из этой части не может быть главным в синтаксическом отношении, в котором зависимое слово находится вне этих скобок. Применению основного алгоритма построения синтаксического дерева предшествует определение глубины вложенности каждого слова предложения в круглые скобки (в дальнейшем будем коротко именовать ее глубиной). Эта глубина вычисляется как разность числа открывающих и числа закрывающих круглых скобок перед словом.

Подробнее остановимся на том, как осуществляется вторая часть правила. Происходит последовательный перебор всех слов предложения, осуществляемый либо в сторону начала предложения, либо в сторону его конца. Перебор начинается со слова, следующего после рассматриваемого слова в том направлении, в котором осуществляется перебор. Если рассматриваемое слово должно быть главным в синтаксическом отношении и имеет большую глубину, чем слово, до которого дошел перебор, то перебор завершается неудачей. Если рассматриваемое слово должно быть зависимым в синтаксическом отношении и имеет меньшую глубину, чем слово, до которого дошел перебор, то перебор также завершается неудачей. Если перебор дошел до начала либо конца предложения, а в некоторых случаях — до начала либо конца *клаузы* (простого предложения в составе сложного), в которой содержалось рассматриваемое слово, и нужное слово не найдено, перебор снова заканчивается неудачей. Если же нужное сло-

во найдено, перебор успешно завершается, и найденное слово используется в следующей части правила для построения синтаксического отношения. В тех случаях, когда ищется несколько слов, может быть осуществлено несколько переборов.

Будем называть слово *допустимым*, если его глубина не является препятствием для включения этого слова в синтаксическое отношение.

Перед основным алгоритмом построения синтаксических отношений, совершается несколько подготовительных действий.

1. Для каждого тире в предложении создается токен, который в качестве леммы содержит само тире, а его набор морфологических характеристик состоит из одного элемента — указания на то, что тире следует рассматривать как глагол. Действительно, в большинстве случаев тире заменяет глагол «есть» в какой-либо форме.
2. Предложение разбивается на сегменты по некоторым знакам препинания — запятым, точкам с запятой, двоеточиям, притом рассматриваются только те из них, что имеют нулевую глубину. Во многих случаях сегмент совпадает с клаузой, но не всегда, так как запятые не только разделяют простые предложения в составе сложного, но и выполняют другие роли (например, отделяют друг от друга однородные члены предложения).

В основном алгоритме синтаксического анализа, помимо попыток применения к слову правил построения синтаксических отношений, для этого слова производятся некоторые вычисления, которые могут быть в дальнейшем использованы при рассмотрении не только этого слова, но и последующих слов. Например, таким образом находится и запоминается последний встретившийся в рассмотренной части предложения глагол, а также последний встретившийся глагол перед последним встретившимся двоеточием.

Некоторые части речи в процессе применения правил приравниваются к другим частям речи. В большинстве случаев это явно оговаривается. Исключение — местоимения: местоимения-существительные во всех правилах приравниваются к существительным, а местоимения-прилагательные — к прилагательным, если находятся непосредственно перед прилагательным или существительным, и к существительным в противном случае. В правилах это явно

не оговаривается, но, например, когда речь в правиле идет о существительном, предполагается, что это может быть также и местоимение-существительное, и местоимение-прилагательное, если оно приравнено к существительному.

Правила синтаксического анализа приведены в приложении 1. Правила расположены в приложении в том же порядке, в каком их рассматривает алгоритм. Если несколько правил независимо друг от друга пытаются создать разные отношения, в которых одно и то же слово было бы зависимым, работает только то из правил, которое было рассмотрено раньше. Таким образом, при изменении порядка применения правил результат может измениться.

Помимо правил, касающихся построения новых синтаксических отношений, имеются три правила, изменяющих уже созданные синтаксические отношения. Все эти правила так или иначе касаются построения множественного актанта — отношения между однородными членами предложения. Эти правила также приведены в приложении 1.

Приведем пример предложения, для которого описанный алгоритм построил дерево зависимостей: «При способе уменьшаемого остатка годовая сумма амортизационных отчислений определяется исходя из остаточной стоимости объекта основных средств на начало отчетного года и нормы амортизации, исчисленной исходя из срока полезного использования этого объекта и коэффициента не выше 3, установленного организацией». Это предложение — несколько измененная часть пункта 19 ПБУ 6/01 [28]. Построенный по этому предложению синтаксический граф изображен на рисунке 4.1. Процесс построения синтаксического графа по этому предложению описан в приложении 2.

§ 3 Упрощение синтаксического графа предложения

Описанные в настоящем параграфе результаты опубликованы в [36, 37].

После синтаксического разбора каждая вершина в графе, за исключением вершин, добавленных в граф по правилу 26, будет соответствовать некоторому токenu. Однако некоторой сущности в предложении может соответствовать не одно слово, а сочетание слов.

Рассмотрим произвольное поддерево синтаксического графа, состоящее из двух или более вершин. Пусть его вершины — $\alpha_1, \alpha_2, \dots, \alpha_n$, причем α_1 — его

в какую-либо вершину $\alpha_j, j \leq n$, проводится ребро из вершины α , ведущее в ту же вершину, что и e , и обладающее той же меткой;

4) вершине α сопоставляется набор всех токенов, сопоставленных вершинам $\alpha_1, \alpha_2, \dots, \alpha_n$.

5) вершины $\alpha_1, \alpha_2, \dots, \alpha_n$ и все входящие в них и выходящие из них ребра удаляются.

Граф будем называть *упрощенным синтаксическим графом*, если он получен из синтаксического графа предложения путем нескольких (в т.ч., 0) применений операции объединения вершин. Мы будем строить упрощенный граф таким образом, чтобы в нем каждой вершине соответствовала либо одна сущность, либо одно или несколько служебных слов (предлогов, союзов и т.д.).

На объединяемые вершины наложены определенные ограничения:

а) вершину α запрещено объединять с любой вершиной, в которую из α ведет ребро, если вершине α сопоставлены слова (сочетания слов) «за исключением», «кроме», «числитель», «знаменатель», «соотношение», «а также», «исходя», «один из», «в течение», «по», «при»; если вершине α сопоставлено количественное отношение («больше», «свыше» и т.д.);

б) вершину α запрещено объединять с вершиной, из которой в α ведет ребро, если вершина α соответствует множественному актанту; если ребру, ведущему в вершину α , сопоставлено одно из следующих синтаксических отношений: «ПРИДАТ_ОПР», «В_СЛУЧАЕ», «В_ОТНОШЕНИИ», «ОТР», «УСЛ», «ЕСЛИ», «КОЛИЧ»; если хотя бы одному ребру, выходящему из вершины α , сопоставлено синтаксическое отношение «ПРИДАТ_ОПР».

Цель данных ограничений - избежать объединения вершин, которым приписаны слова, обозначающие логические функции и отношения. В некоторых случаях эти ограничения могут игнорироваться.

Упрощение графа производится с помощью перечисленных ниже приемов. Во всех случаях указано, игнорирует ли прием ограничения а и б.

1) Пусть «способ» — одно из слов, соответствующих некоторой вершине α , и пусть из α в некоторую вершину β ведет ребро, соответствующее синтаксическому отношению «ГЕНИТ_ИГ». Тогда объединяются вершины α, β и все вершины, в которые можно перейти из β . Ограничения а и б игнорируются.

2) Пусть из некоторой вершины α в некоторую вершину β ведет ребро, соответствующее синтаксическому отношению «НЕДЕЛИМ». Тогда объединяются вершины α и β . Ограничения а и б игнорируются.

3) Пусть некоторой вершине α сопоставлен один токен, являющийся предлогом (но не «по», «при», «кроме»), и из α выходит лишь одно ребро. Тогда α объединяется с вершиной, в которую из α ведет ребро. Ограничение а учитывается, ограничение б - игнорируется.

4) Пусть из некоторой вершины α в некоторую вершину β ведет ребро, соответствующее синтаксическому отношению «ПГ». Тогда объединяются вершины α и β . Ограничения а и б учитываются.

5) Путь несколько идущих подряд слов предложения составляют некоторую сущность из хранящегося в отдельном файле списка сущностей (например, «объект основных средств», «физическое лицо» и т.д). Тогда объединяются все вершины, соответствующие этим словам. Ограничения а и б игнорируются.

6) Пусть из вершины α в вершину β ведет ребро. Тогда α и β объединяются. Ограничения а и б учитываются.

Приемы используются в следующем порядке: сначала происходит попытка применить прием 1 там, где это возможно; затем то же делается с приемом 2. Далее в цикле по всем вершинам происходит попытка применения приемов 3 и 4, после чего в цикле по всем словам предложения происходит попытка использовать прием 5. Наконец, везде, где возможно, используется прием 6.

Нетрудно заметить, что прием 6 фактически является основным, тогда как почти все остальные приемы нужны лишь для случаев, когда необходимо проигнорировать хотя бы одно из ограничений а и б.

В результате применения указанных приемов к синтаксическому графу, изображенному на рисунке 4.1, мы получим упрощенный синтаксический граф, изображенный на рисунке 4.2.

§ 4 Отождествление сущностей

Результаты, описанные в данном параграфе и в следующих параграфах настоящей главы, опубликованы в [36, 37].

Пусть по каждому предложению текста закона уже построена формула.

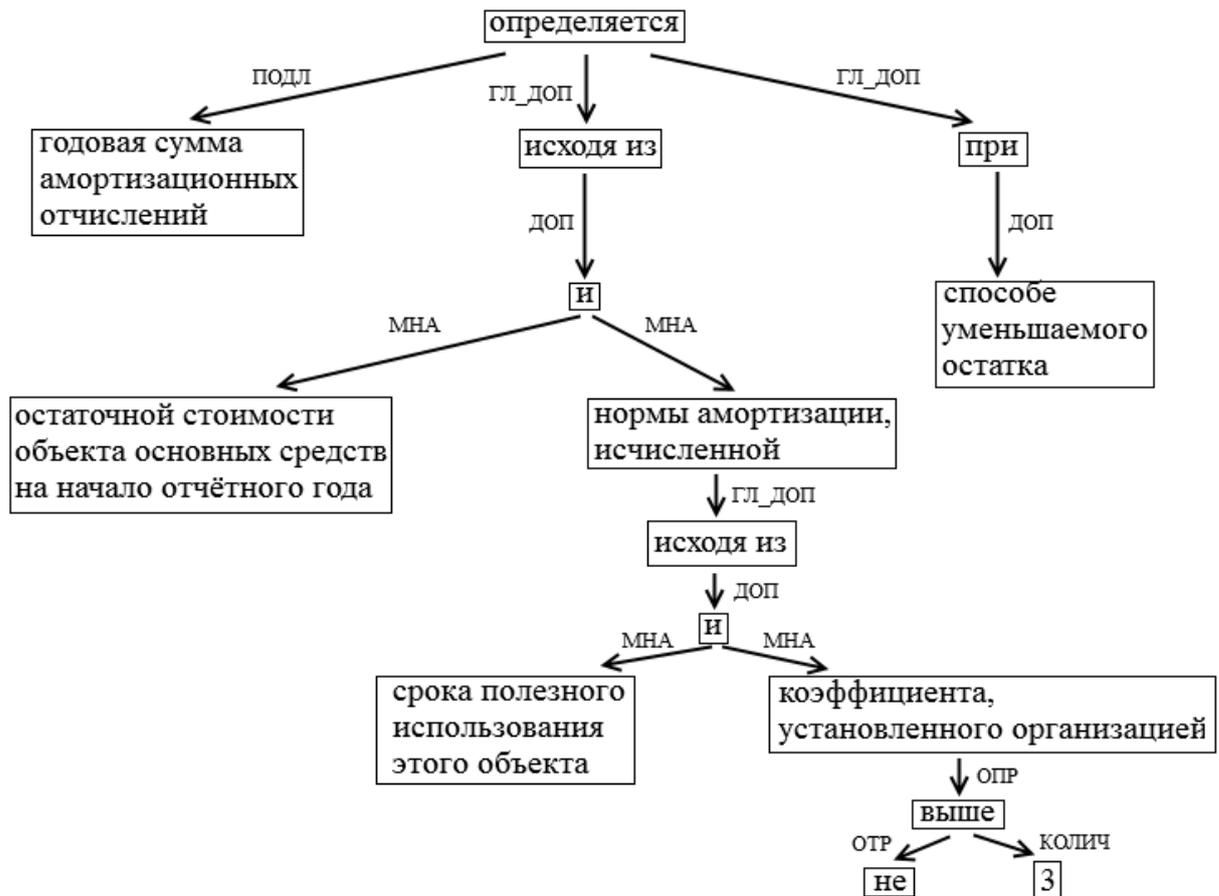


Рис. 4.2. Упрощенный синтаксический граф

Переменные этих формул соответствуют определенным текстовым фрагментам.

Перед построением формулы каждой сущности необходимо сопоставить переменную. Для того, чтобы одной и той же сущности в разных предложениях была сопоставлена одна и та же переменная, проводится процедура отождествления сущностей. Часто сущность полностью определяется текстовым фрагментом (предполагается, что если слова в двух фрагментах текста отличаются только своей формой, например, находятся в разных падежах, то эти два фрагмента определяют одну и ту же сущность). В некоторых случаях этого, однако, не происходит. Для представления множества всех сущностей удобно использовать ориентированное дерево, каждой вершине которого (кроме корня) сопоставлена некоторая сущность и некоторый текст. Если в этом дереве из вершины, которой сопоставлена сущность A , ведет ребро в вершину, которой сопоставлена сущность B , то это означает, что B — атрибут (т.е. свойство или

действие) A .

Перед построением дерева сущностей каждое личное местоимение («он», «она», «его» и др.) заменяется на слово, на которое это местоимение указывает. Для этого ищется ближайшее к личному местоимению в сторону начала предложения слово, согласованное с этим местоимением в роде и числе.

Дерево сущностей строится по упрощенному синтаксическому графу предложения. Этот граф представляет собой ориентированное дерево. Алгоритм вначале рассматривает его корень, затем — все вершины, которые из корня ведет ребро, и т.д. Вначале дерево сущностей состоит из двух вершин: корня α_0 , которому не сопоставлена никакая сущность, и вершины α_1 , в которую из корня ведет ребро и которой сопоставлена сущность «объект основных средств». Обозначим рассматриваемую вершину упрощенного синтаксического графа через α .

Если вершине α соответствует только одно или несколько служебных слов, то этой вершине не соответствует никакая сущность, и рассмотрение этой вершины сразу же прекращается. Каждой из остальных вершин упрощенного синтаксического графа будет соответствовать одна вершина в дереве сущностей.

Рассмотрим в упрощенном синтаксическом графе цепь, ведущую из корня в вершину α . Обозначим ближайшую к α (но отличную от нее) вершину этой цепи, которой соответствует некоторая вершина в строящемся дереве сущностей, через β (если таковая вершина найдется). Вершину цепи, в которую из β ведет ребро, обозначим через δ (эта вершина может совпадать с α). Пусть вершине β в строящемся дереве сущностей соответствует вершина β' .

Рассмотрение вершины α заключается в попытке использования следующих приемов.

1. Если из α в некоторую вершину γ ведет ребро, соответствующее синтаксическому отношению «ПОДЛ», то для целей построения дерева сущностей считается, что в упрощенном синтаксическом графе есть ребро из γ в α , но нет ребра из α в γ , и что ребро, ведущее (в действительности) в α (если оно есть), ведет в γ . Таким образом, мы считаем, что сказуемое является атрибутом подлежащего, а не наоборот.
2. Пусть в тексте, соответствующем вершине α , есть слово «объект», причем

из соответствующей этому слову вершины первоначального (т.е. неупрошенного) синтаксического графа не выходит ребро, которому сопоставлено синтаксическое отношение «ГЕНИТ_ИГ»; либо пусть в тексте, соответствующем α , есть слова «объект основных средств». Тогда вершине α соответствует вершина дерева сущностей, в которую ведет ребро из α_1 , и которой сопоставлена сущность, соответствующая α , и сопоставленный α текст без слова «объект» (во втором случае — без слов «объект основных средств»).

3. Если вершина β отсутствует, то вершине α соответствует вершина дерева сущностей, в которую ведет ребро из корня, и которой сопоставлены текст и сущность, соответствующие вершине α .
4. Пусть ребру из β в δ сопоставлено синтаксическое отношение «ДОП», «ГЛ_ДОП» либо «КОЛИЧ», а вершине β сопоставлены слова «в случае», «кроме», «за исключением», «в отношении», «исходя из», «при», «по», «для», «пониматься», «являться», либо слова, обозначающие количественное отношение («больше», «равный» и т.д.). Тогда вершине α соответствует вершина дерева сущностей, в которую ведет ребро из корня, и которой сопоставляются текст и сущность, соответствующие вершине α .
5. Пусть ребру из β в δ сопоставлено синтаксическое отношение «УСЛ», а из α не выходит ребро, соответствующее синтаксическому отношению «ПОДЛ». Тогда вершине α соответствует вершина дерева сущностей, в которую ведет ребро из корня, и которой сопоставляются текст и сущность, соответствующие вершине α .
6. Вершине α в строящемся дереве сущностей будет соответствовать вершина, в которую ведет ребро из β' , и которой сопоставлены те же текст и сущность, что и вершине α .

Порядок применения приемов следующий: сначала используется прием 1; далее происходит попытка применения приемов 2–6 (в порядке возрастания номера приема). Если попытка использования одного из приемов 2–6 успешна, то рассмотрение вершины α сразу же прекращается. Нетрудно заметить, что

ситуация, когда все приемы 2–6 неприменимы, невозможна, так как ровно один из приемов 3 и 6 всегда применим.

В процессе построения дерева сущностей каждой сущности сопоставляется переменная.

После того, как дерево сущностей построено, происходит небольшое изменение текста некоторых вершин этого дерева: из текста удаляется слово «исчисленный», если в упрощенном синтаксическом графе из соответствующей этому тексту вершины ведет ребро в вершину со словами «исходя из». Кроме того, из текста удаляются указательные местоимения («этот», «такой» и т.д.).

На рисунке 4.3 изображено дерево сущностей, построенное по упрощенному синтаксическому графу, изображенному на рисунке 4.2.

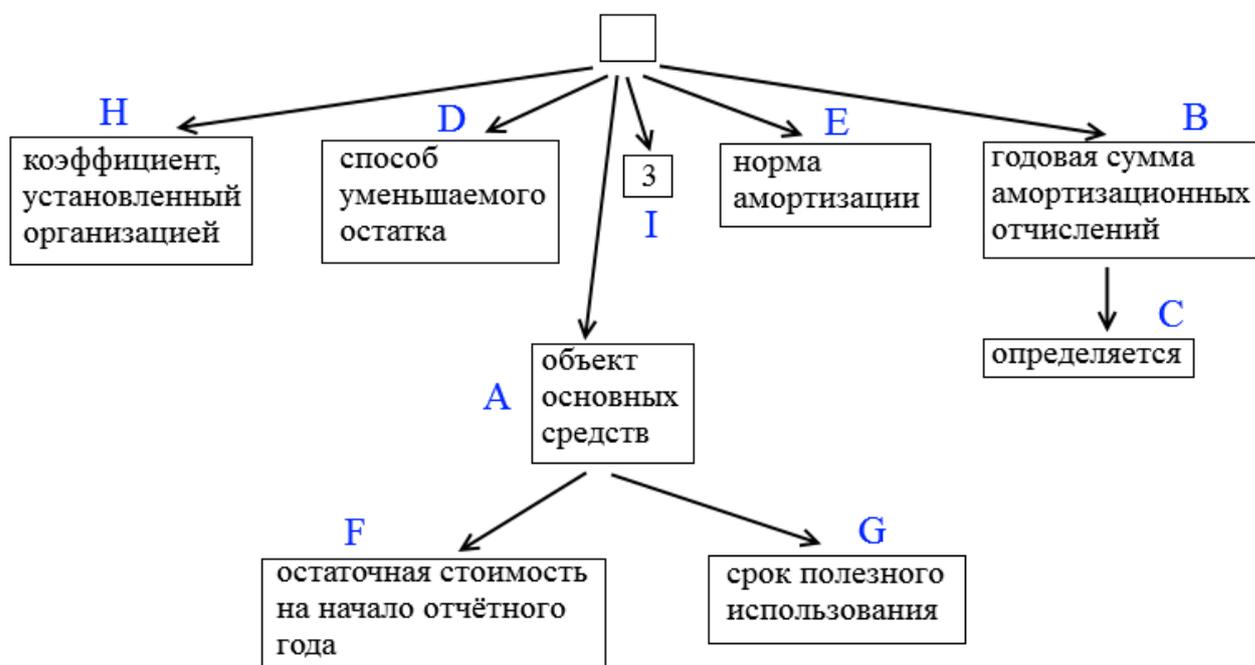


Рис. 4.3. Дерево сущностей

§ 5 Построение логической формулы

Для каждого предложения по его упрощенному синтаксическому графу (с использованием дерева сущностей) строится логическая формула. При этом ис-

пользуются следующие приемы.

1) Если в предложении присутствует одно из следующих слов или сочетаний слов: «по», «в течение», «в случаях», «при», «в отношении», «если», то управляемая этим словом или сочетанием слов часть предложения определяет область применения статьи закона либо некоторое условие. Пусть в соответствующую этому слову вершину γ ведет ребро из некоторой вершины α . Вершину, из которой в α ведет ребро (если таковая найдется) обозначим как β . Вершину, в которую из γ ведет ребро (если таковая найдется; таких вершин не может быть более одной) обозначим как δ . Хотя бы одна из вершин β и δ будет существовать. Возможны три случая:

а) вершина δ существует Пусть A — формула, соответствующая графу, состоящему из всех вершин, достижимых из δ (будем считать, что каждая вершина достижима из себя), и соединяющих их ребер, B — формула, соответствующая графу, состоящему из всех остальных вершин (кроме γ) и соединяющих их ребер. Тогда итоговая формула будет иметь вид $(A \rightarrow B)$;

б) вершина δ не существует, а β не соответствует множественному актанту. Пусть A — формула, соответствующая графу, состоящему из всех вершин, достижимых из α , и соединяющих их ребер, B — формула, соответствующая графу, состоящему из всех остальных вершин (кроме γ) и соединяющих их ребер. Тогда итоговая формула будет иметь вид $(A \rightarrow B)$;

в) вершина δ не существует, а β соответствует множественному актанту. Это означает, что слова «по», «в течение», «в случаях», «при», «в отношении», «если» относятся ко всему множественному актанту. Пусть B — формула, соответствующая графу, состоящему из всех вершин, достижимых из β , и соединяющих их ребер, C — формула, соответствующая графу, состоящему из всех остальных вершин и соединяющих их ребер. Тогда итоговая формула будет иметь вид $(B \rightarrow C)$.

2) Если в предложении некоторая его часть подчинена действию с помощью слова «кроме», то это означает, что действие выполняется тогда и только тогда, когда не выполняется условие, выраженное частью предложения, управляемой словом «кроме». Пусть это слово сопоставлено вершине α . Вершину, из которой в α ведет ребро, обозначим β . Пусть A — формула, соответствующая подграфу,

состоящему из всех вершин, достижимых из β , B — формула, соответствующая графу, состоящему из всех остальных вершин (кроме α) и соединяющих их ребер. Итоговая формула будет иметь вид $(\neg A \sim B)$.

3) «не» означает логическое отрицание. Пусть это слово сопоставлено вершине α , в которую входит ребро, выходящее из вершины β . Если по вершине β построено логическое выражение A , то после применения приема оно превращается в $(\neg A)$.

4) «Либо», «или» при перечислении однородных членов предложения, если речь не идет о перечислении аргументов некоторой функции, означает логическое «или» либо функцию, истинную, когда истинно значение ровно одного ее аргумента. Пусть вершина β — это множественный актанта, которому соответствует несколько однородных членов предложения, а сами эти однородные члены предложения сопоставлены вершинам $\alpha_1, \dots, \alpha_r$, в каждое из которых ведет ребро из β . Пусть также в вершину β не ведет ребро из вершины, которой сопоставлены слова «исходя из», и хотя бы одной из вершин $\beta, \alpha_1, \dots, \alpha_r$, либо вершине, в которую из $\beta, \alpha_1, \dots, \alpha_r$ ведет ребро, сопоставлено слово «или» («либо»). Возможны 2 случая:

4.1) среди рассматриваемых вершин $\alpha_j, j \in \{1, \dots, r\}$, найдется вершина α_i такая, что из нее выходит ребро, соответствующее синтаксическому отношению «УСЛ». Пусть это ребро ведет в вершину γ . Обозначим через A формулу, построенную по графу, состоящему из всех вершин, достижимых из γ , и соединяющих их ребер; через B — формулу, построенную из всех остальных вершин, достижимых из α_i , и соединяющих их ребер. Если осталась лишь одна вершина $\alpha_j, i \neq j$, то обозначим через формулу, построенную по графу, состоящему из всех вершин, достижимых из α_j , и соединяющих их ребер. Если же таких вершин α_j , что $j \neq i$, больше одной, то обозначим через формулу, полученную применением 4.1 или 4.2 ко всем рассматриваемым вершинам $\alpha_s, s \in \{1, \dots, r\}$, кроме α_i , которую мы исключим из рассмотрения. Тогда результатом применения приема будет формула $(A \& B \vee \neg A \& C)$;

4.2) среди рассматриваемых вершин $\alpha_j, j \in \{1, \dots, r\}$, нет ни одной вершины α_i такой, что из нее выходит ребро, соответствующее синтаксическому отношению «УСЛ». Тогда, если обозначить через $A_j, j \in \{1, \dots, r\}$, фор-

мулу, построенную по графу, состоящему из всех вершин, достижимых из α_j , то в результате применения приема будет создана формула, являющаяся дизъюнкцией всех формул A_j , соответствующих рассматриваемым вершинам $\alpha_j, j \in \{1, \dots, r\}$;

5) Союз «и» или отсутствие союзов при перечислении однородных членов предложения, если речь не идет о перечислении аргументов некоторой функции, означает логическое «и» либо «или». Пусть вершина β — это множественный актанта, которому соответствует несколько однородных членов предложения, а сами эти однородные члены предложения сопоставлены вершинам $\alpha_1, \dots, \alpha_r$, в каждое из которых ведет ребро из β . Пусть также в вершину β не ведет ребро из вершины, которой сопоставлены слова «исходя из», и никакой из вершин $\beta, \alpha_1, \dots, \alpha_r$ и вершин, в которую из $\beta, \alpha_1, \dots, \alpha_r$ ведет ребро, не сопоставлены слова «или», «либо», «а также». Если однородные члены предложения — глаголы, в результате применения приема будет создана формула $(A_1 \& A_2 \& \dots \& A_r)$. В противном случае результатом применения приема будет формула $(A_1 \vee A_2 \vee \dots \vee A_r)$.

6) Слова «исходя из» означают некоторую функцию. Пусть эти слова сопоставлены вершине γ , β' — вершина, из которой в γ идет ребро. Если вершине β' соответствует глагол, обозначим через β вершину, в которую из β' ведет ребро с меткой «ПОДЛ»; иначе обозначим через β саму вершину β' . Обозначим через α вершину, в которую ведет ребро из γ (найдется лишь одна такая вершина α). Пусть B — переменная, соответствующая вершине β . Возможно несколько случаев:

6.1) Вершине α не сопоставлен множественный актанта. Пусть C — переменная, обозначающая объект, определенный текстовым фрагментом, соответствующим вершине α . Тогда в результате применения приема графу, состоящему из всех вершин, достижимых из β' , и соединяющих их ребер будет поставлена в соответствие формула $((B = f_\beta(C)) \& G)$, где f_β — некоторая функция, в тексте закона не определенная явно, G — формула, полученная путем применения приема 6 или приема 8 к вершине, в которую из α ведет ребро (если найдется такая вершина, удовлетворяющая условиям применения хотя бы одного из этих приемов), и тождественная истина иначе.

6.2) Вершине α сопоставлен множественный актанта. Пусть соответствующие ему однородные члены предложения сопоставлены вершинам $\alpha_1, \dots, \alpha_r$, в каждое из которых ведет ребро из α . Пусть никакой из вершин $\alpha, \alpha_1, \dots, \alpha_r$ и вершин, в которую из $\alpha, \alpha_1, \dots, \alpha_r$ ведет ребро, не сопоставлены слова «или», «либо», «а также». Пусть $A_j, j \in \{1, \dots, r\}$ — переменная, соответствующая вершине α_j , либо поставленная в соответствие вершине α_j вспомогательная переменная, если α_j соответствует множественный актанта. Тогда в результате применения приема графу, состоящему из всех вершин, достижимых из β' , и соединяющих их ребер будет поставлена в соответствие формула $((B = f_\beta(A_1, A_2, \dots, A_r)) \& G_1 \& \dots \& G_r)$, где f_β — некоторая функция, в тексте закона не определенная явно, а G_j — формула, полученная путем применения приема 6 или приема 8 к вершине, в которую из α_j ведет ребро (если найдется такая вершина, удовлетворяющая условиям применения хотя бы одного из этих приемов), и тождественная истина иначе.

6.3) Вершине α сопоставлен множественный актанта. Пусть соответствующие ему однородные члены предложения сопоставлены вершинам $\alpha_1, \dots, \alpha_r$, в каждое из которых ведет ребро из α , и хотя бы одной из вершин $\alpha, \alpha_1, \dots, \alpha_r$, либо вершине, в которую из $\alpha, \alpha_1, \dots, \alpha_r$ ведет ребро, сопоставлено слово «или» («либо»). Тогда в результате применения приема графу, состоящему из всех вершин, достижимых из β' , и соединяющих их ребер будет поставлена в соответствие формула $D \& G$, где f_β — некоторая функция, в тексте закона не определенная явно, C — поставленная в соответствие вершине α вспомогательная переменная, D — формула, строящаяся с помощью приема из пункта 7, примененного к вершине α . G — формула, полученная путем применения приема 6 или приема 8 к вершине α , если она удовлетворяет условиям применения хотя бы одного из этих приемов, и тождественная истина иначе.

7) Пусть вершине α сопоставлен множественный актанта, и из вершины γ , которой сопоставлены слова «исходя из», ведет ребро либо в α , либо в вершину, из которой в α можно попасть, переходя по ребрам с меткой «МНА» (в соответствии с их направлением). Пусть вершины β' и β ищутся для вершины γ так же, как в пункте 6. Пусть также соответствующие α однородные члены предложения сопоставлены вершинам $\alpha_1, \dots, \alpha_r$, в каждое из которых

ведет ребро из α , и хотя бы одной из вершин $\alpha, \alpha_1, \dots, \alpha_r$, либо вершине, в которую из $\alpha, \alpha_1, \dots, \alpha_r$ ведет ребро, сопоставлено слово «или» («либо»). Пусть $D_j, j \in \{1, \dots, r\}$ — это $f_\beta(A_j)$, если вершине α_j не сопоставлен множественный актант; в противном случае это соответствующая множественному актанту формула, строящаяся с помощью приема из пункта 7, примененного к вершине α_j . Пусть также G_j — формула, полученная путем применения приема 6 или приема 8 к вершине, в которую из α_j ведет ребро (если найдется такая вершина, удовлетворяющая условиям применения хотя бы одного из этих приемов), и тождественная истине иначе. Тогда в результате применения приема будет создана формула $D_1 \& G_1 \vee \dots \vee D_r \& G_r$.

8) Для слов, обозначающих количественные отношения («больше», «меньше», «равный» и т.д.), в формуле будет присутствовать соответствующий предикат. Пусть β — вершина, которой сопоставлено одно из этих слов, γ — вершина, из которой в β ведет ребро. Пусть выходящее из β ребро, которому соответствует синтаксическое отношение «КОЛИЧ», ведет в вершину α . Пусть A — переменная, соответствующая вершине α , C — переменная, соответствующая вершине γ . Тогда графу, состоящему из вершин α, β, γ и соединяющих их ребер, будет соответствовать формула $pr_\beta(C, A)$, где $pr_\beta(C, A)$ — соответствующий вершине β предикат.

Применение приема означает проверку для вершины всех условий, указанных в тексте приема, и построение некоторой формулы, если эти условия выполнены. Предполагается, что один и тот же прием не может быть применен к одной и той же вершине (обозначенной во всех приемах как β) более одного раза, за исключением случаев, когда прием обращается рекурсивно к себе.

Алгоритм построения формулы по графу, состоящему из всех вершин, достижимых из данной, и соединяющих их ребер, состоит в следующем. Проверяется, является ли данная вершина вершиной β из приемов 2-6 или 8. Если да, то применяется все приемы, которые можно применить (в порядке возрастания номеров), притом применяемые приемы могут вызвать этот же алгоритм для некоторых вершин, к которым из данной идет ребро. Если ни один из приемов 2-6, 8 нельзя применить, то данной вершине сопоставляется переменная, обозначающая объект, определенный текстовым фрагментом, соответствующим этой

вершине. Производится конъюнкция этой переменной и всех формул, построенных с помощью этого же алгоритма по вершинам, в которые из данной вершины идет ребро.

Итоговый алгоритм построения формулы в соответствии с описанными приемами следующий.

Если условия приема 1 выполнены для какой-либо вершины (не должно быть более одной такой вершины), то в результате применения этого приема исходный граф разбивается на 2 подграфа. В каждом из этих подграфов выбирается корень, и к ним поочередно применяется описанный выше алгоритм построения формулы по графу, состоящему из всех вершин, достижимых из данной.

Если условия приема 1 не выполнены ни для какой вершины, то в графе выбирается корень, и к нему применяется описанный выше алгоритм построения формулы по графу, состоящему из всех вершин, достижимых из данной.

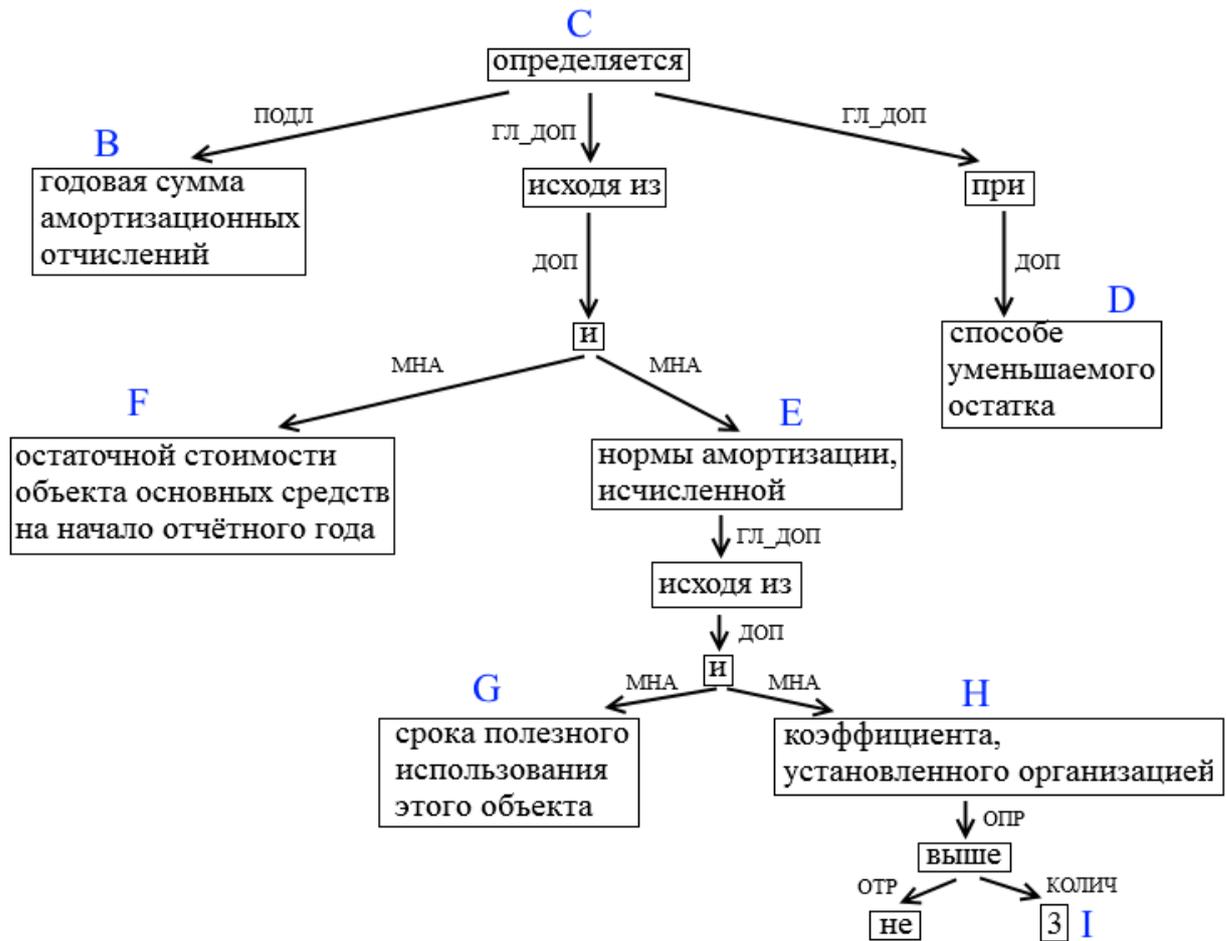
На рисунке 4.4 изображены упрощенный синтаксический граф вместе с сопоставленными его вершинам переменными, а также формула, построенная по упрощенному синтаксическому графу.

§ 6 Построение модели закона по логическим формулам

Когда по каждому предложению закона построена логическая формула, можно перейти непосредственно к построению модели. Как и на всех предыдущих этапах, это можно сделать с помощью применения определенных приемов к имеющимся формулам.

1) Пусть формула имеет вид $A \rightarrow B$, где в B имеется выражение вида $C = D$, но нет выражения вида $E = C$. Найдем в уже построенном фрагменте графа модели закона вершину второго вида, в которой происходит запись значения в поле памяти, соответствующее C из рассматриваемой формулы. Возможны 2 варианта.

1.1) Такая вершина есть. Обозначим ее α . Рассмотрим вершину, из которой в α ведет ребро. Обозначим ее β . Далее пройдем из β по цепочке ребер с номером 0 против их направления, пока эта цепочка не закончится. Обозначим вершину, в которую мы попадем, как γ . Далее выполним действия, указанные в пункте



$$D \rightarrow ((B=f(F,E)) \& (E=g(G,H)) \& !(H>I))$$

Рис. 4.4. Формула, построенная по упрощенному синтаксическому графу (f,g — еще не определенные функции)

1.3.

1.2) Такой вершины нет. Тогда выберем какую-либо не рассмотренную ранее вершину и будем считать ее вершиной второго вида, соответствующей C из рассматриваемой формулы. Обозначим эту вершину α . Выберем еще одну не рассмотренную ранее вершину, обозначим ее γ . Выпустим из нее ребро в вершину α . Далее выполним действия, указанные в пункте 1.3.

1.3) Будем считать γ вершиной четвертого вида. Рассмотрим какие-либо три новые вершины, выпустим из них в γ по одному ребру, пронумеруем эти

ребра числами 0, 1 и 2. Тогда вершине, из которой в γ ведет ребро номер 2, будет соответствовать вычисление логического выражения A . К вершине, из которой в γ ведет ребро номер 1, следует применить прием 3 относительно переменной C . Вершина, из которой в γ ведет ребро номер 0, будет, возможно, рассмотрена в результате применения 1.1 к одной из оставшихся формул. Если этого не произойдет, значит, процедура вычисления модели никогда не затронет эту вершину.

2) Пусть в формуле имеется выражение вида $A = B$, нет выражения вида $C = A$, и прием 1 к формуле неприменим. Тогда выберем какую-либо не рассмотренную ранее вершину и будем считать ее вершиной второго вида, соответствующей A из рассматриваемой формулы. Обозначим эту вершину α . Выберем еще одну не рассмотренную ранее вершину, обозначим ее γ . Выпустим из нее ребро в вершину α . Вершине γ будет соответствовать вычисление функции B (при помощи приема 3).

3) По фрагменту формулы, соответствующему вычислению значения переменной B , строится один из следующих фрагментов модели (прием 3.2 используется, если 3.1 неприменим):

3.1) Пусть в формуле есть подформула вида $A \& (B = C) \vee \& D$, причем в D входит выражение вида $B = E$. Выберем из всех таких подформул данной формулы (для фиксированного B) ту, что не содержится в другой такой подформуле. Пусть α — вершина, соответствующая вычислению значения B . Будем считать, что это — вершина четвертого вида. Выберем какие-либо три новые вершины, выпустим из них в α по одному ребру, пронумеруем эти ребра числами 0, 1 и 2. Тогда вершине, из которой в α ведет ребро номер 2, будет соответствовать вычисление логического выражения A . Вершине, из которой в α ведет ребро номер 1, будет соответствовать вычисление функции C (смотрите прием 3.2). Если D имеет вид $(B = E)$, вершине, из которой в α ведет ребро номер 0, будет соответствовать вычисление функции E (также с помощью приема 3.2); в противном случае следует использовать прием 3.1, рассматривая эту вершину в качестве α , а D в качестве выбранной подформулы.

3.2) Если в формуле есть подформула $B = A$, где A — некоторая переменная, и нет выражений вида $A = C$, где C — некоторая переменная или

функция, то будем считать, что рассматриваемая вершина — первого вида, и в ней происходит получение значения A из соответствующего поля памяти. В противном случае речь идет о вычислении некоторой арифметической функции от нескольких аргументов $A_1, \dots, A_k, k \in N$. Если эта функция в тексте закона не задана явно, то можно, например, потребовать у пользователя ее задания. Если функция уже задана, то ей можно сопоставить несколько вершин третьего типа, моделирующих вычисление этой функции (так, чтобы последний этап вычисления функции проходил в вершине α). Ребра, передающие значения аргументов A_1, \dots, A_k , будут входить в некоторые из этих вершин. К каждой вершине, из которой эти ребра выходят, применяется прием 3 относительно соответствующей переменной.

4) Пусть переменная A — аргумент некоторой функции, и на эту переменную наложено некоторое условие (в виде бинарного или унарного отношения, которому она должна удовлетворять). Тогда выполнение этого условия проверяется в той вершине, в которой вычисляется значение A . В случае невыполнения условия пользователю потребуется изменить значение переменных, с помощью которых вычисляется A (как правило, это собственно переменная A).

5) Пусть формула имеет вид $A \rightarrow B_1 \& \dots \& B_k, \in N$, где B_1, \dots, B_k — переменные. Тогда такая формула преобразуется в конъюнкцию формул $A \rightarrow B_1, \dots, A \rightarrow B_k$, к которым можно применять прием 6.

6) Пусть формула имеет вид $A \rightarrow B$, где B — переменная, A — подформула, не содержащая никаких переменных, кроме булевых. Пусть B' — переменная B (точнее, соответствующая ей сущность), определенная с помощью всех остальных формул ($B' = 0$, если B не определяется из оставшихся формул). Тогда B определяется как $A \vee B'$.

7) Пусть формула имеет вид $A \rightarrow (B \sim C)$ либо $A \rightarrow (C \sim B)$, где B — переменная, A, C — подформулы, не содержащие никаких переменных, кроме булевых. Если C — тоже переменная, то для того, чтобы отличить B от C , можно использовать семантический граф: в нем соответствующие этим переменным текстовые фрагменты должны быть связаны каким-либо семантическим отношением. За B принимается переменная, чей текстовый фрагмент является главным в этом семантическом отношении. Пусть B' — переменная B

(точнее, соответствующая ей сущность), определенная с помощью всех остальных формул ($B' = 0$, если B не определяется из оставшихся формул). Тогда B определяется как $A \& C \vee B'$.

8) Если функция, определяющая некоторую булеву переменную, не содержит никаких переменных, кроме булевых, то она моделируется с помощью несколько вершин третьего типа. В некоторые из этих вершин будут входить ребра, соответствующие переменным.

Алгоритм построения модели закона с помощью этих приемов следующий. Сначала для каждой формулы, не содержащей никаких переменных, кроме булевых, применяется прием 5; затем ко всем таким формулам применяются приемы 6 и 7; наконец, к каждой такой формуле применяется восьмой прием. Ко всем остальным формулам применяется прием 1 или 2, а затем — прием 4.

На рисунке 4.5 изображен фрагмент модели закона, соответствующий предложению «При способе уменьшаемого остатка годовая сумма амортизационных отчислений определяется исходя из остаточной стоимости объекта основных средств на начало отчетного года и нормы амортизации, исчисленной исходя из срока полезного использования этого объекта и коэффициента не выше 3, установленного организацией». Этот фрагмент построен по формуле, приведенной на рисунке 4.4, с помощью дерева сущностей, откуда были получены соответствующие переменным формулы сущности.

Предполагается, что выяснить, какая именно функция имеется в виду в законе, если эта функция в тексте закона не задана явно, можно, предоставив этот выбор пользователю, либо получив эти данные из каких-либо других нормативно-правовых актов. Последнее решение сложнее реализуется и не всегда может дать результат; в алгоритме в настоящее время реализовано первое решение.

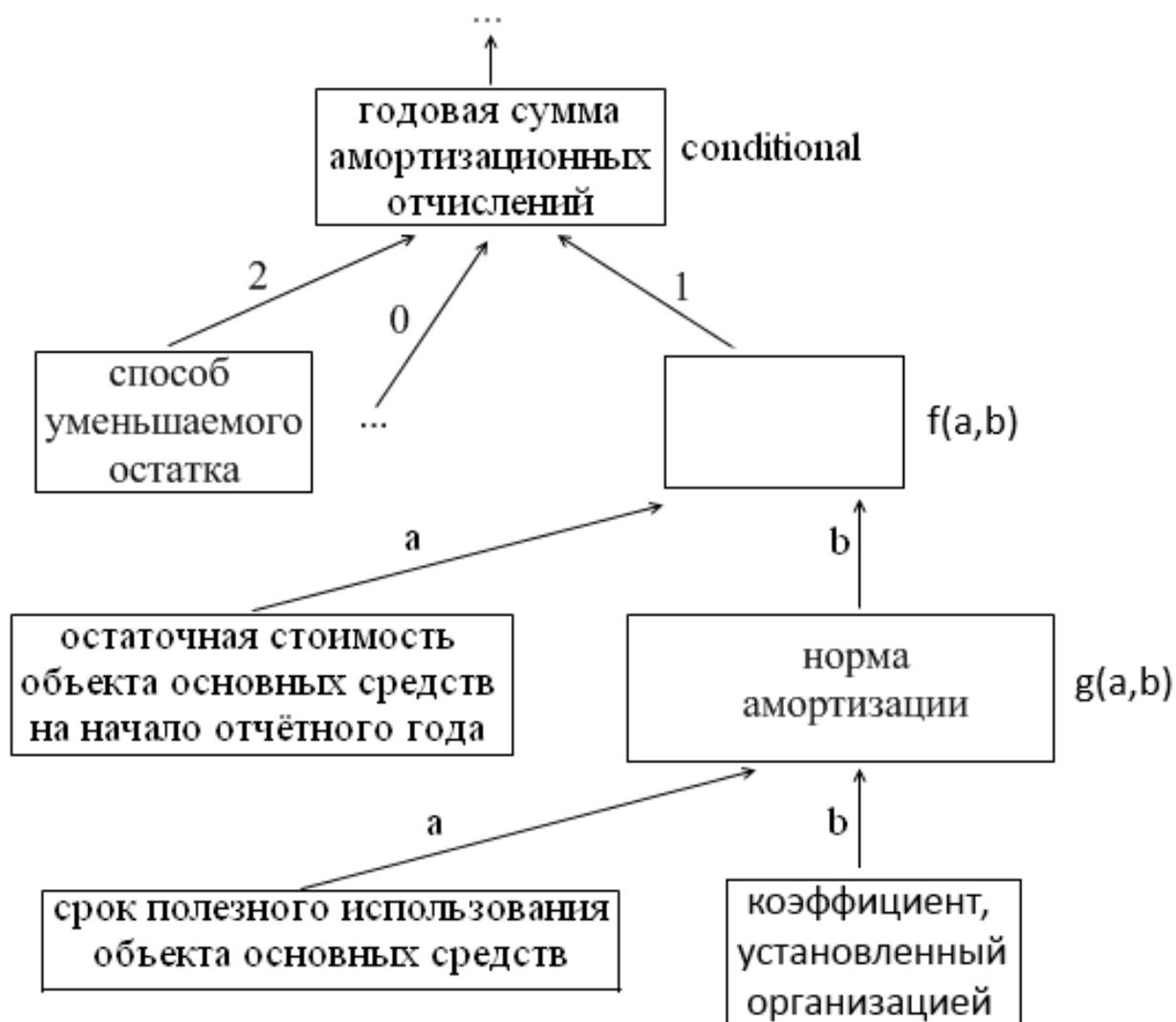


Рис. 4.5. Фрагмент модели закона.

Приложение 1. Перечень правил синтаксического анализа

В настоящем приложении перечислены правила синтаксического анализа, используемые в настоящей работе. Правила 1–23 создают новые синтаксические отношения, тогда как правила 24–26 изменяют уже созданные синтаксические отношения, причем правила 25–26 применяются тогда, когда для предложения уже построено дерево зависимостей, к каждой подходящей вершине дерева.

1. Правило, создающее для тире синтаксическое отношение «ВАРИАНТ» между этим тире и последним глаголом, находящимся перед последним двоеточием перед тире. Если такой глагол существует, то он был найден ранее. В создаваемом отношении этот глагол является главным элементом, а тире — зависимым.
2. Правило, которое для тире проверяет, является ли союз первым элементом сегмента, в котором это тире находится. Если является, то происходит перебор сегментов от предыдущего к первому в предложении, пока первым элементом сегмента является тот же союз и не достигнуто начало предложения. Если у сегмента, на котором перебор прекратился, вторым словом является тот же союз, а первым — глагол или причастие, то между этим глаголом (причастием) и тире создается синтаксическое отношение «ВАРИАНТ», в котором глагол (причастие) — главный элемент, а тире — зависимый.
3. Правило, проверяющее, не образует ли рассматриваемое слово вместе с несколькими идущими за ним предлог, состоящий из нескольких слов (к таким предлогам относятся, например, «исходя из», «в течение», «в отношении» и др; для проверки алгоритм обращается к списку таких предло-

гов). Если образует, то каждая пара идущих подряд слов из этого предлога соединяется отношением «НЕДЕЛИМ», где главным объявляется то из двух слов, которое находится ближе к началу предложения. Кроме того, после применения этого правила все слова, входящие в предлог, считаются предлогами.

4. Правило, которое для слова «не» ищет ближайшее находящееся после него допустимое слово, не являющееся предлогом или союзом, и создает синтаксическое отношение «ОТР» («отрицание»), в котором найденное слово является главным элементом, а «не» — зависимым.
5. Правило, которое для существительного ищет ближайшее находящееся после него допустимое слово, не являющееся прилагательным в родительном падеже. Если такое слово найдено, и это — существительное в родительном падеже, то создается синтаксическое отношение «ГЕНИТ-ИГ» («генитивная именная группа») с рассматриваемым словом в качестве главного элемента и найденным существительным в родительном падеже в качестве зависимого элемента.
6. Правило, ищущее для существительного, не находящегося в именительном падеже, ближайшее к нему среди находящихся перед ним слов сегмента допустимое слово, являющееся глаголом, существительным, причастием (не согласованным с существительным), кратким причастием, предлогом либо союзом (но не союзом «или», «и», «а»). Если такого слова нет и сегмент отделен от предыдущего точкой с запятой, то в качестве искомого слова рассматривается последний глагол в первом сегменте предложения, а при наличии предлога непосредственно после этого глагола — этот предлог. Создается синтаксическое отношение «ДОП» («дополнение»), в котором главным является найденное слово, а зависимым — рассматриваемое существительное.
7. Правило, ищущее для не находящегося в именительном падеже существительного, чей сегмент отделен от предыдущего запятой, первое в предыдущем сегменте (считая от его начала) существительное в том же падеже. Если такое существительное найдено, и оно является зависимым в каком-либо

синтаксическом отношении, то добавляется такое же отношение, в котором зависимым словом является рассматриваемое правилом существительное.

8. Правило, ищущее для предлога «на» ближайшее к нему среди находящихся перед ним слов сегмента допустимое слово из списка существительных, обозначающих величину или значение («стоимость», «цена», «размер» и т.д.). Если между этим существительным и рассматриваемым предлогом «на» не расположены глагол или тире, создается синтаксическое отношение «ОПР» («опредетельное») с найденным существительным в качестве главного слова и предлогом «на» в качестве зависимого слова.
9. Правило, которое ищет для предлога «о» ближайшее к нему среди находящихся перед ним слов сегмента допустимое слово из списка существительных, обозначающих информацию («законодательство», «положение» «приказ» и т.д.). Если между этим существительным и рассматриваемым предлогом «о» не расположены глагол или тире, создается синтаксическое отношение «ОПР», где главным словом является найденное существительное, а зависимым — предлог «о».
10. Правило, проверяющее для существительного в именительном падеже, есть ли глагол в сегменте, где это существительное находится. Пусть глагола там нет, и непосредственно перед последним двоеточием, встретившимся до этого сегмента, находится то же самое существительное (возможно, в другом падеже). Тогда создается отношение «А_ИМЕННО», где главным словом является найденное перед двоеточием существительное, а зависимым — рассматриваемое слово.
11. Правило, которое для предлога, а также для слов «исходя» (если следующее за ним слово – «из») и «как», ищет допустимый глагол, чтобы связать их синтаксическим отношением «ГЛ_ДОП» («глагол – дополнение»), в котором глагол будет главным словом, а рассматриваемое правилом слово — зависимым. Глагол ищется в том же сегменте, где находится рассматриваемое слово, сначала в направлении от этого слова к концу сегмента, а в случае неудачи — в направлении от этого слова к началу сегмента. В данном правиле к глаголу приравниваются отглагольные существительные и

причастия. Если нужного слова найти не удалось, а рассматриваемое правилом слово — «по» или «при», ищется в направлении от него к концу предложения ближайший допустимый глагол, не находящийся в клаузе, начинающейся со слова «который». В случае, когда нужное слово все еще не найдено, в качестве него берется последний глагол, находящийся перед последним (до рассматриваемого правилом слова) двоеточием, если такой существует.

12. Правило, ищущее для существительного в именительном падеже допустимый глагол, чтобы связать их синтаксическим отношением «ПОДЛ» («подлежащее»), в котором глагол будет главным словом, а существительное — зависимым. Глагол ищется так же, как и в предыдущем правиле, за тем исключением, что отглагольное существительное в данном случае к глаголу не приравнивается.
13. Правило, которое ищет для наречия допустимый глагол, чтобы связать их синтаксическим отношением «ГЛ_ОБСТ» («глагол – обстоятельство»), где глагол будет главным словом, а наречие — зависимым. Глагол ищется так же, как и в правиле 11.
14. Правило, ищущее для слова «если» допустимый глагол, чтобы связать их синтаксическим отношением «ЕСЛИ», в котором глагол будет главным словом, а существительное — зависимым. Глагол ищется так же, как и в правиле 11.
15. Правило, которое для слова «если», являющегося зависимым в каком-либо синтаксическом отношении, ищет ближайший к нему допустимый глагол, находящийся в предложении до сегмента, в котором находится рассматриваемое слово. Если такое слово найдено, то создается отношение «УСЛ» («условие»), где найденный глагол будет главным словом, а зависимым будет слово, являющееся главным в отношении, где «если» является зависимым.
16. Правило, которое для полного причастия, находящегося непосредственно после запятой, ищет ближайшее к нему в сторону начала предложения существительное, имеющее тот же падеж и число, причем если это число

— единственное, то и тот же род. Если такое существительное нашлось, создается синтаксическое отношение «ПРИЧ_СУЩ» («причастие – существительное»), в котором главным словом является существительное, а зависимым — причастие.

17. Правило, которое для прилагательного или полного причастия ищет ближайшее к нему в сторону конца предложения существительное, имеющее тот же падеж и число, причем если это число — единственное, то и тот же род. Если такое существительное нашлось, создается синтаксическое отношение «ПРИЛ_СУЩ» («прилагательное – существительное») (если рассматриваемое слово — прилагательное) или «ПРИЧ_СУЩ» (если рассматриваемое слово — причастие), где главным словом является существительное, а зависимым — рассматриваемое слово.
18. Правило, обрабатывающее союзы «и», «или», «а», не находящиеся в начале сегмента. Сначала ищется слово w_1 — допустимое слово, ближайшее к рассматриваемому союзу по направлению к началу предложения. Кроме того, ищется слово w_2 — допустимое слово, ближайшее к рассматриваемому союзу по направлению к концу предложения. Если w_2 — полное причастие, краткое причастие, глагол либо отглагольное существительное, то от рассматриваемого союза в сторону начала предложения ищется слово w_3 — ближайшее допустимое слово, удовлетворяющее следующим условиям: если w_2 — полное причастие, то w_3 должно быть полным причастием либо прилагательным, имеющим тот же падеж, что w_2 ; если w_2 — глагол либо краткое причастие, то найденное слово также должно быть глаголом либо кратким причастием и иметь то же число, что и w_2 ; если w_2 — отглагольное существительное, то w_3 должно быть существительным в том же падеже. Если w_2 не является полным причастием, кратким причастием, глаголом либо отглагольным существительным, ищется ближайшее к союзу по направлению к концу предложения допустимое слово w_4 , удовлетворяющее следующим условиям: если w_1 — прилагательное либо полное причастие, то найденное слово также должно быть прилагательным либо полным причастием и иметь тот же падеж, что и w_1 ; если w_1 — существительное, то найденное слово должно быть существительным и иметь тот же падеж, что

и w_1 ; если w_1 — глагол либо краткое причастие, то найденное слово также должно быть глаголом либо кратким причастием и иметь то же число, что и w_1 . Заметим, что w_1 может совпадать с w_3 , а w_2 — с w_4 .

Если подходящее слово найдено, то создается два синтаксических отношения «ОДНОР_ИГ» («однородные именные группы»), которые содержит рассматриваемый союз в качестве главного слова и найденные слова w_1 и w_4 (либо w_2 и w_3 , если w_2 — полное причастие) в качестве зависимых. Если слово w_1 (w_3) до применения данного правила участвовало в каком-либо синтаксическом отношении в качестве зависимого, то оно заменяется в этом отношении на рассматриваемый союз.

19. Правило, обрабатывающее союзы «и», «или», «а», находящиеся в начале сегмента. Правило в основном аналогично предыдущему, отличие заключается в том, что в качестве w_2 выбирается не любое допустимое слово, ближайшее к рассматриваемому союзу по направлению к концу предложения, а только глагол или краткое причастие. Затем так же, как это делается в предыдущем правиле, для w_2 ищется слово w_3 , и с участием этих слов и рассматриваемого союза создаются синтаксические отношения «ОДНОР_ИГ».
20. Правило, проверяющее, является ли последнее допустимое слово предложения, идущее в предложении до количественного числительного в именительном падеже, существительным. Если да, то создается отношение «НОМЕР», в котором найденное существительное — главное слово, а числительное — зависимое.
21. Правило, проверяющее, является ли последнее допустимое слово предложения, идущее в предложении до числительного, словом, которое может обозначать количественное отношение («больше», «ниже», «равное» и т.д.). Если это так, то строится синтаксическое отношение «КОЛИЧ» («количественное»), где найденное слово является главным, а числительное — зависимым словом.
22. Правило, проверяющее, является ли существительным последнее отличное от частицы «не» допустимое слово предложения, идущее в предложении до

слова, обозначающего количественное отношение. Если это так, то строится синтаксическое отношение «ОПР» («определяющее»), где найденное существительное — главный элемент, а слово, обозначающее количественное отношение — зависимый.

23. Правило, обрабатывающее слово, чья лемма — «который». Сначала в предыдущем сегменте ищется ближайшее к рассматриваемому слову допустимое существительное, имеющее то же число, что и рассматриваемое слово. Затем ищется ближайшее к рассматриваемому слову по направлению к концу предложения допустимое слово, являющееся глаголом, кратким причастием или тире. Если нужные слова найдены, создается синтаксическое отношение «ПРИДАТ_ОПР» («придаточное определяющее»), в котором главным словом является найденное существительное, а зависимым — найденный глагол, краткое причастие или тире.
24. Правило, которое для каждого союза «и», «или», «а» просматривает все отношения «ОДНОР_ИГ», где этот союз является главным словом. Если среди зависимых слов есть слова $w_1, w_2, \dots, w_k, k \in N$ из списка существительных, обозначающих величину или значение, и ровно одно слово не из этого списка, то отношения, в которых рассматриваемый союз — главный, разрываются, а в том отношении, где этот союз является зависимым, он заменяется на единственное найденное слово не из списка существительных, обозначающих величину или значение. Далее происходит перебор вершин ориентированного дерева от вершины, которой соответствует рассматриваемый союз, по направлению к корню дерева. Пусть вершине, получаемых при этом переборе, соответствует слово w из списка существительных, обозначающих величину или значение. Тогда перебор прекращается; найденное слово в том отношении, где оно является зависимым, заменяется на рассматриваемый союз; создаются отношения «ОДНОР_ИГ», в которых главным словом является рассматриваемый союз, а зависимыми — слова w, w_1, w_2, \dots, w_k .

Данное правило введено по следующей причине: оказывается полезным считать, что существительные, обозначающие значение, могут быть однородными только с существительными, обозначающими значение. Это пра-

вило применяется сразу после применения правила 18 или 19.

25. Правило, создающее множественный актант из слов, являющихся зависимыми от одного и того же слова в отношении «ОДНОР_ИГ». Множественный актант удобно представлять как вершину, из которой исходит ребро с меткой «МНА» к каждой вершине, соответствующей слову из множественного актанта. Основная работа по созданию такого множественного актанта уже проделана при применении правил 18 и 19. Теперь же достаточно указать, что каждая вершина, из которой исходит хотя бы одно ребро с меткой «ОДНОР_ИГ» (в действительности, как видно из построения отношений «ОДНОР_ИГ», не бывает вершин, из которых выходило бы ровно одно такое ребро), соответствует множественному актанту, а метку ребра поменять на «МНА».
26. Правило, создающее множественный актант из всех слов w_1, w_2, \dots, w_k , являющихся зависимыми от одного и того же слова w в отношении с одним и тем же названием v (но не «ОДНОР_ИГ», так как этот случай рассматривается отдельно, и не «ГЛ_ДОП», так как такие слова не являются однородными). В дерево зависимостей добавляется новая вершина w_0 , куда из вершины, соответствующей слову w , опускается ребро с меткой v . Все ребра, ведущие в вершины, соответствующие словам w_1, w_2, \dots, w_k , удаляются, и в каждую из этих вершин из w_0 проводится ребро с меткой «МНА». Наконец, вершина w_0 помечается как соответствующая множественному актанту.

Приложение 2. Пример построения синтаксического графа предложения

Будем считать, что на вход алгоритма, осуществляющего синтаксический анализ, поступила последовательность лексем, на которые было разбито предложение, а также следующие токены (они получены благодаря морфологическому разбору, произведенному программой, созданной на проекте АОР [23]):

1. При ПРИ ПРЕДЛ,
2. способе СПОСОБ С,но,мр,пр,ед,
3. уменьшаемого УМЕНЬШАТЬ ПРИЧАСТИЕ,стр,пе,нс,но,од,нст,мр,рд,ед,
4. остатка ОСТАТОК С,но,мр,рд,ед,
5. годовая ГОДОВОЙ П,но,од,жр,им,ед,
6. сумма СУММА С,но,жр,им,ед,
7. амортизационных АМОРТИЗАЦИОННЫЙ П,но,од,рд,мн,
8. отчислений ОТЧИСЛЕНИЕ С,но,ср,рд,мн,
9. определяется ОПРЕДЕЛЯТЬСЯ Г,дст,нп,нс,Зл,нст,ед,
10. исходя ИСХОДИТЬ нп,нс,
11. из ИЗ
12. остаточной ОСТАТОЧНЫЙ П,кач,но,од,жр,рд,ед,
13. стоимости СТОИМОСТЬ С,но,жр,рд,ед,
14. объекта ОБЪЕКТ С,но,мр,рд,ед,

15. основных ОСНОВНОЙ П,но,од,рд,мн,
16. средств СРЕДСТВО С,но,ср,рд,мн,
17. на НА ПРЕДЛ,
18. начало НАЧАЛО С,но,ср,вн,ед,
19. отчетного ОТЧЕТНЫЙ П,кач,но,од,мр,рд,ед,
20. года ГОД С,но,мр,рд,ед,
21. и И СОЮЗ,
22. нормы НОРМА С,но,жр,вн,рд,им,ед,мн,
23. амортизации АМОРТИЗАЦИЯ С,но,жр,рд,ед,
24. , ,
25. исчисленной ИСЧИСЛИТЬ ПРИЧАСТИЕ,стр,пе,св,но,од,прш,жр,пр,тв,
дт,рд,ед,
26. исходя ИСХОДИТЬ нп,нс,
27. из ИЗ
28. срока СРОК С,но,мр,рд,ед,
29. полезного ПОЛЕЗНЫЙ П,кач,но,од,ср,рд,ед,
30. использования ИСПОЛЬЗОВАНИЕ С,но,ср,рд,ед,
31. этого ЭТОТ МС-П,но,од,мр,рд,ед,
32. объекта ОБЪЕКТ С,но,мр,рд,ед,
33. и И СОЮЗ,
34. коэффициента КОЭФФИЦИЕНТ С,но,мр,рд,ед,
35. не НЕ ЧАСТ,
36. выше ВЫШЕ Н,

37. 3 3 ЧИСЛ-П,но,од,жр,тв,ед,

38. , ,

39. установленного УСТАНОВИТЬ ПРИЧАСТИЕ,стр,пе,св,но,од,прш,ср,мр,
вн,рд,ед,

40. организацией ОРГАНИЗАЦИЯ С,но,жр,тв,ед,

41. . .

В результате применения описанного алгоритма без использования правил 25 и 26 создаются следующие синтаксические отношения (у отношения сначала указывается главное слово, а потом зависимое):

1. ГЛ_ДОП(определяется, при) по правилу 11;
2. ГЕНИТ_ИГ(способе, остатка) по правилу 5;
3. ДОП(при, способе) по правилу 6;
4. ПРИЧ_СУЩ(остатка, уменьшаемого) по правилу 17;
5. ПРИЛ_СУЩ(сумма, годовая) по правилу 17;
6. ГЕНИТ_ИГ(сумма, отчислений) по правилу 5;
7. ПОДЛ(определяется, сумма) по правилу 12;
8. ПРИЛ_СУЩ(отчислений, амортизационных) по правилу 17;
9. ГЛ_ДОП(определяется, исходя) по правилу 11;
10. НЕДЕЛИМ(исходя, из) по правилу 3 (таких отношений создается два, так как «исходя из» встречается в предложении дважды);
11. ПРИЛ_СУЩ(стоимости, остаточной) по правилу 17;
12. ДОП(из, стоимости) по правилу 6 (однако затем по правилу 18 это отношение заменяется на отношение
13. ДОП(из, и));

14. ГЕНИТ_ИГ(стоимости, объекта) по правилу 5;
15. ГЕНИТ_ИГ(объекта, средств) по правилу 5;
16. ПРИЛ_СУЩ(средств, основных) по правилу 17;
17. ОПР(стоимости, на) по правилу 8;
18. ГЕНИТ_ИГ(начало, года) по правилу 5;
19. ДОП(на, начало) по правилу 6;
20. ПРИЛ_СУЩ(года, отчетного) по правилу 17;
21. ОДНОР_ИГ(и, стоимости) по правилу 18;
22. ОДНОР_ИГ(и, нормы) по правилу 18;
23. ГЕНИТ_ИГ(нормы, амортизации) по правилу 5;
24. ПРИЧ_СУЩ(амортизации, исчисленной) по правилу 16;
25. ГЛ_ДОП(исчисленной, исходя) по правилу 11;
26. ГЕНИТ_ИГ(срока, использования) по правилу 5;
27. ДОП(из, срока) по правилу 6 (однако затем по правилу 18 это отношение заменяется на отношение
28. ДОП(из, и));
29. ПРИЛ_СУЩ(использования, полезного) по правилу 17;
30. ГЕНИТ_ИГ(использования, объекта) по правилу 5;
31. ПРИЛ_СУЩ(объекта, этого) по правилу 17;
32. ОДНОР_ИГ(и, срока) по правилу 18;
33. ОДНОР_ИГ(и, коэффициента) по правилу 18;
34. ОТР(выше, не) по правилу 4;
35. ОПР(коэффициента, выше) по правилу 22;

36. КОЛИЧ(выше, 3) по правилу 21;
37. ПРИЧ_СУЩ(коэффициента, установленного) по правилу 16;
38. ДОП(установленного, организацией) по правилу 6.

В случае применения правил 25 и 26 к вершинам построенного дерева зависимостей, метки ребер, соответствующих отношениям 21, 22, 32 и 33, поменяются с «ОДНОР_ИГ» на «МНА». Кроме того, вершины, из которых эти ребра выходят, будут помечены как соответствующие множественному актанту. Как можно заметить, правило 25 будет применено дважды, а правило 26 — ни разу.

Заключение

В диссертации получены следующие основные результаты.

Получено значение минимального времени поиска подслова в базе данных, представляющей собой множество слов, при определенных ограничениях на используемые при поиске функции. Именно, используются только функции, каждая из которых для некоторого числа i возвращает i -ю букву запроса к базе данных, и функция, тождественно равная 1. Для алгоритмов, осуществляющих поиск за минимальное время, найдена нижняя оценка памяти. Кроме того, описан алгоритм, производящий поиск за минимальное время и требующий памяти, минимальной по порядку для алгоритмов, осуществляющих поиск за наименьшее время.

Найдено значение минимального времени поиска вхождений подслова в базе данных, представляющей собой множество слов, при некоторых ограничениях на множество используемых функций. Ограничения состоят в следующем: разрешено использовать функцию, тождественно равную 1, а также функции, каждая из которых для некоторого числа i возвращает i -ю букву запроса к базе данных. Кроме перечисленных, разрешены только функции, которые для некоторого слова и некоторого числа i проверяют, содержится ли префикс этого слова в запросе к базе данных, начиная с i -й буквы запроса, причем предполагается, что время вычисления такой функции пропорционально количеству произведенных при ее вычислении операций сравнения букв. Построен алгоритм, осуществляющий поиск за минимальное по порядку время и требующий минимальной по порядку памяти.

Для случая, когда длина подслова подается на вход алгоритма вместе с самим подсловом, получено значение минимального времени поиска при определенных ограничениях на используемые при поиске функции. Именно, разрешено использовать функцию, тождественно равную 1, и функции, для некоторого

числа i возвращающие i -ю букву запроса к базе данных; кроме того, можно использовать функцию, возвращающую длину запроса, и функции, проверяющие для некоторого числа, не превышает ли длина запроса это число. Найдены верхние оценки сложности поиска при ограничениях на объем используемой памяти.

Описан метод, позволяющий по тексту нормативно-правового акта, касающегося бухгалтерского учета, автоматически строить схемы вычисления значений объектов, упомянутых в данном нормативно-правовом акте. В частности, описаны приемы синтаксического анализа; упрощения синтаксического графа; отождествления сущностей; представления упрощенного синтаксического графа в виде формулы; построения по всем формулам схем вычисления значений объектов. С помощью данного метода созданы схемы вычисления значений объектов, описанных в положении по бухгалтерскому учету ПБУ 6/01.

В качестве перспектив дальнейшей разработки темы диссертации можно предложить распространение метода автоматического построения схем, вычисляющих значения объектов, на более широкий класс нормативных актов.

Список литературы

- [1] Gusfield D. Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. — N. Y.: Cambridge University Press, 1997. — 556 p.
- [2] Knuth D.E., Morris J.H., Pratt V.B. Fast pattern matching in strings // SIAM J. Comput. — 1977. — Vol. 6, N 2. — P. 323—350.
- [3] Boyer R.S., Moore J.S. A fast string searching algorithm // Commun ACM. — 1977. — Vol. 20, N 10. — P. 762—772.
- [4] Galil Z. On improving the worst case running time of the Boyer-Moore string searching algorithm // Comm. ACM. — 1979. — Vol. 22, N 9. — P. 505—508.
- [5] Horspool N. Practical fast searching in strings // Software Practice and Experience. — 1980. — Vol. 10. — P. 501—506.
- [6] Weiner P. Linear pattern matching algorithm // Proceedings of the 14th IEEE Annual Symposium on Switching and Automata Theory. — 1973. — P. 1—11.
- [7] Manber U., Myers G. Suffix Arrays: A New Method for On-Line String Searches // SIAM J. Comput. — 1993. — Vol. 22, N 5. — P. 935—948.
- [8] Munro J.I., Raman V., Rao S.S. Space Efficient Suffix Trees // Proceedings of the 18th Foundations of Software Technology and Theoretical Computer Science Conference, LNCS. — N. Y.; Berlin: Springer-Verlag, 1998. — P. 186—196.
- [9] Kurtz S. Reducing the Space Requirement of Suffix Trees // Softw. Pract. Exper. — 1999. — Vol. 29, N 13. — P. 1149—1171.
- [10] Navarro G., Mäkinen V. Compressed fulltext indexes // ACM Comput. Surv. — 2007. — Vol. 39, N 1. — Article 2.

- [11] Grossi R., Vitter J.S. Compressed suffix arrays and suffix trees with applications to text indexing and string matching // Proc. of 32nd annual ACM symposium on Theory of computing, STOC'00. — 2000. — P. 397–406.
- [12] Sadakane K. New text indexing functionalities of the compressed suffix arrays // Journal of Algorithms. — 2003. — Vol. 48, N 2. — P. 294–313.
- [13] Ferragina P., Manzini G., Mäkinen V., Navarro G. Compressed representations of sequences and full-text indexes // ACM Transactions on Algorithms. — 2007. — Vol. 3, N 2. — Article 20.
- [14] Demaine E.D., Lopez-Ortiz A. A linear lower bound on index size for text retrieval // Journal of Algorithms - Special issue: Twelfth annual ACM-SIAM symposium on discrete algorithms. — 2003. — Vol. 48, N 1. — P. 2–15.
- [15] Gal A., Miltersen P.B. The Cell Probe Complexity of Succinct Data Structures // Proc. of 30th International Colloquium on Automata, Languages and Programming, ICALP'03. — 2003. — P. 332–344.
- [16] Golynski A. Cell Probe Lower Bounds For Succinct Data Structures // Proc. of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms. — 2009. — P. 625–634.
- [17] Гасанов Э.Э., Кудрявцев В.Б. Теория хранения и поиска информации. — М.: Физматлит, 2002. — 288 с.
- [18] Кудрявцев В.Б., Гасанов Э.Э., Подколзин А.С. Основы теории интеллектуальных систем. — М.: МАКС Пресс, 2017. — 612 с.
- [19] Chen D., Manning C.D. A Fast and Accurate Dependency Parser using Neural Networks // Proc. of EMNLP 2014. — 2014. — P. 740–750.
- [20] de Marneffe M., Dozat T., Silveira N., Haverinen K., Ginter F., Nivre J., Manning C.D. Universal Stanford Dependencies: A cross-linguistic typology // In Proc. of the 9th Conference on Language Resources and Evaluation (LREC). — 2014. — P. 4585–4592.
- [21] Universal Dependencies [Электронный ресурс]. — Режим доступа: <http://universaldependencies.org>

- [22] Национальный корпус русского языка. Синтаксически размеченный корпус русского языка: информация для пользователей [Электронный ресурс]. — Режим доступа: <http://www.ruscorpora.ru/instruction-syntax.html>
- [23] Сокирко А. Семантические словари в автоматической обработке текста (по материалам системы ДИАЛИНГ) [Электронный ресурс]. — Режим доступа: <http://www.aot.ru/docs/sokirko>
- [24] Мельчук И. А. Опыт теории лингвистических моделей «Смысл \Leftrightarrow Текст». — М.: Школа «Языки русской культуры», 1999. — 368 с.
- [25] Iordanskaja L., Kittredge R., Polguere A. Lexical Selection and Paraphrase in a Meaning-Text Generation Model // Natural Language Generation in Artificial Intelligence and Computational Linguistics. SECS. — Boston: Springer, 1991. — Vol. 119, P. 293—312.
- [26] Anisimovich K.V., Druzhkin K.Ju., Minlos F.R., Petrova M.A., Selegey V.P., Zuev K.A. Syntactic and semantic parser based on ABBYY Compreno linguistic technologies // Международная конференция по компьютерной лингвистике «Диалог-2012». — 2012. — Vol. 2, P. 91—103.
- [27] Ben-Or M. Lower Bounds For Algebraic Computation Trees // Proc. 15th ACM Annu. Symp. Theory Comput. — 1983. — P. 80—86.
- [28] Приказ Министерства финансов Российской Федерации от 30.03.2001 N 26н «Об утверждении Положения по бухгалтерскому учету «Учет основных средств» ПБУ 6/01» [Электронный ресурс]. — Режим доступа: <http://base.consultant.ru/cons/cgi/online.cgi?req=doc;base=LAW;n=111056>.
- [29] Перпер Е.М. Нижние оценки временной и объемной сложности задачи поиска подстроки // Тезисы докладов XX Международной научной конференции студентов, аспирантов и молодых ученых «Ломоносов-2013». — М., 2013.
- [30] Перпер Е.М. Нижние оценки временной и объемной сложности задачи поиска подслово // Дискретная математика. — 2014. — Т. 26, вып. 2. — С. 58—70.

- [31] Hagerup T. Sorting and Searching on the Word RAM // STACS 98, LNCS. — Berlin; Heidelberg: Springer, 1998. — Vol. 1373, P. 366—398.
- [32] Перпер Е.М. Порядок сложности задачи поиска в множестве слов вхождений под слова // Интеллектуальные системы. Теория и приложения (ранее: Интеллектуальные системы по 2014, вып. 2, ISSN 2075-9460). — 2015. — Т. 19, вып. 1. — С. 99—116.
- [33] Перпер Е.М. О сложности поиска подстроки // Интеллектуальные системы. Теория и приложения (ранее: Интеллектуальные системы по 2014, вып. 2, ISSN 2075-9460). — 2012. — Т. 16, вып. 1–4. — С. 299—320.
- [34] Перпер Е.М. О синтаксическом анализе нормативных актов // Материалы XII Международного семинара «Дискретная математика и ее приложения» имени академика О. Б. Лупанова (Москва, МГУ, 20–25 июня 2016). — 2016. — М.: Изд-во механико-математического факультета МГУ. — С. 344—346.
- [35] Перпер Е.М. О синтаксическом анализе юридических текстов // Интеллектуальные системы. Теория и приложения (ранее: Интеллектуальные системы по 2014, вып. 2, ISSN 2075-9460). — 2016. — Т. 20, вып. 2. — С. 31—49.
- [36] Перпер Е.М. Автоматическое построение модели нормативно-правового акта по его тексту // Тезисы докладов XXI Международной научной конференции студентов, аспирантов и молодых ученых «Ломоносов–2014». — М., 2014.
- [37] Кудрявцев В.Б., Гасанов Э.Э., Перпер Е.М. Автоматическая генерация компьютерной программы, моделирующей нормативно-правовой акт // Интеллектуальные системы. Теория и приложения (ранее: Интеллектуальные системы по 2014, вып. 2, ISSN 2075-9460). — 2014. — Т. 18, вып. 2. — С. 133—156.
- [38] Подколзин А.С. Самообучение интеллектуальной системы // Интеллектуальные системы. Теория и приложения (ранее: Интеллектуальные системы по 2014, вып. 2, ISSN 2075-9460). — 2014. — Т. 18, вып. 2. — С. 197—266.