Оглавление

Введение

Актуальность

Графовые модели данных широко используются в различных областях, например, в биоинформатике [27], анализе социальных сетей [34, 9], графовых базах данных [22, 30] и разных видах статического анализа (**TODO**: ссылки).

Одной из важных задач в анализе графовых моделей данных является поиск путей с заданными ограничениями. Одним из способов задавать такие ограничения являются формальные языки: если на рёбрах графа написаны метки из фиксированного алфавита, то можно искать пути, конкатенация меток на которых принадлежит фиксированному языку [5]. Например, хорошо изучена задача поиска путей с ограничениями, заданными регулярными языками (**TODO**: link). В этой же работе мы остановимся на классе контекстно-свободных языков (**TODO**: link (?)), так как они позволяют решать более широкий класс задач.

Задача поиска путей с контекстно-свободными ограничениями, или, сокращённо, CFPQ[1] была сформулирована Михалисом Яннакакисом в 1990 году [30] в применении к запросам к декларативному языку Datalog [29, 7]. С тех пор было предложение множество алгоритмов для её решения, в основном, основанных на разных видах синтаксического анализа: алгоритм Репса [23], использующий метод, схожий с алгоритмом Кока-Янгера-Касами [32], алгоритм Хеллингса [16], использующий аннотированные грамматики и другие [26, 4, 22, 12, 11].

К сожалению, недавно Кёйперс и др. экспериментально показали [14], что текущие методы не достаточно эффективны для использования на практике. Что не удивительно, так как все они имеют асимптотику $\mathcal{O}(n^3)$ (где n — размер входного графа, а размер грамматики — константа), и лучшее ускорение, которого можно добиться, уменьшает время работы лишь в $\mathcal{O}(\log n)$ раз [10] (используя метод четырёх русских [24]). Более того, существует условная нижняя оценка [15, 8], согласно которой не существует комбинаторного[2] субкубического[3] алгоритма для задачи CFPQ.

Всё вышесказанное приводит к тому, что имеет смысл разрабатывать алгоритмы для частных случаев задачи, имеющие лучшее время работы.

Данная работа нацелена

Постановка задачи и ключевые термины

TODO: *Может, это подвинуть в Literature review?*

[1] Context-Free Path Querying

[2] Этот термин не вполне определен, но можно понимать его как “не алгебраический”. В частности, комбинаторные алгоритмы не должны использовать деление и вычитание, так те пользуются особенностями алгебраических структур (а именно, существованием обратного)

[3] С временем работы $\mathcal{O}(n^{3-\varepsilon})$

Для формальной постановки задачи потребуется ввести некоторые вспомогательные определения.

Определение 0.1. *Ориентированный помеченный граф* (или граф с метками) — это тройка $G = \langle V, E, \Sigma \rangle$, где V — множество вершин, Σ — множество меток, $E \subseteq V \times V \times \Sigma$ — множество рёбер.

Неформально, это обычный мультиграф, каждому ребру которого сопоставлена метка из алфавита Σ.

Определение 0.2. *Контекстно-свободная грамматика* — это четвёрка $\langle \Sigma, N, S, P \rangle$, где

- Σ — конечный алфавит
- N — конечное множество нетерминалов
- $S \in N$ — стартовый нетерминал
- P — конечное множество продукций (правил грамматики), имеющих вид $N_i \to \alpha$, где $N_i \in N, \alpha \in (\Sigma \cup N)^*$

Пример 0.1. **TODO***: пример*

Определение 0.3. *Контекстно-свободный язык* — это язык, распознаваемый контекстно-свободной грамматикой

Определение 0.4. Слово w *читается* на пути p, если конкатенация меток p образует w.

Теперь определим саму задачу.

Определение 0.5. Входной граф: G
Входная грамматика: $\mathcal{G}$
РКА входной грамматики: $\mathcal{R}$

Теперь будут введены некоторые понятие (в основном, из теории формальных языков), которые встретятся далее по тексту работы.

Определение 0.6 (Язык Дика)**.** Языком Дика на k типах скобок (D_k) называют контекстно-свободный язык над алфавитом $\Sigma_k = \{(_1,)_1, (_2,)_2 \dots (_k,)_k\}$, состоящий из правильных скобочных последовательностей на k типах скобок.

Задачу CFPQ для языка Дика называют также задачу Диковой достижимости (Dyck-reachability).

Определение 0.7 (Двунаправленный граф)**.** Помеченный граф $G = \langle V, E, \Sigma_k \rangle$ называется двунаправленным (bidirected), если в нём для каждого ребра $\langle u, v, (_i \rangle$ найдётся противоположное ребро $\langle v, u,)_i \rangle$ и наоборот.

Неформально, матрица смежности такого графа симметрична, и метки на симметричных рёбрах — это парные открывающая/закрывающая скобки.

Определение 0.8 (Система Непересекающихся Множеств (СНМ))**.**

Цель и задачи

Структура работы

1. Обзор литературы

Это всё будет красиво ужато в абзацы, а не останется в виде списка

1.1. Решения задачи в общем случае

1. $\mathcal{O}(n^3k^3)$ [23]

   Грамматика приводится к Нормальной форме Хомского, считается $dp_{i,j,c}$ — выводится ли путь $i \rightsquigarrow j$ из нетерминала s, при добавлении 1 ($dp_{i,j,A} = 1$) перебираются все соседние нетерминалы B, т.ч. $\exists C \to AB$ (или $C \to BA$) и $k \in V(G)$, и если $dp_{j,k,B}$ (или $dp_{k,i,B}$), то $dp_{i,k,C} = 1$ (или $dp_{k,j,C} = 1$) и (i, k, C) добавляется в рабочую очередь.

2. $\mathcal{O}(n^3k^3/\log n)$ [11]

   Алгоритм [23], к которому применили метод 4 русских

3.

TODO: Написать про BMM-сложность.

1.2. Решения задачи в частных случаях

Понятно, что для решения практических задач далеко не всегда нужна CFPQ в общем случае. Чаще всего для каждой конкретной задачи нужна конкретная КС грамматика, а иногда ещё и понятны ограничения на тип графа.

Пользуясь этой информацией (ограничениями на тип грамматики и графа) можно конструировать частные и потому более быстрые решения. Этим уже занимались, сейчас мы выпишем всё, что на текущий момент известно:

1. Язык Дика $\mathcal{O}(n^3k)$ [20]

   Просто применить алгоритм Репса [23] и нормально оценить время работы.

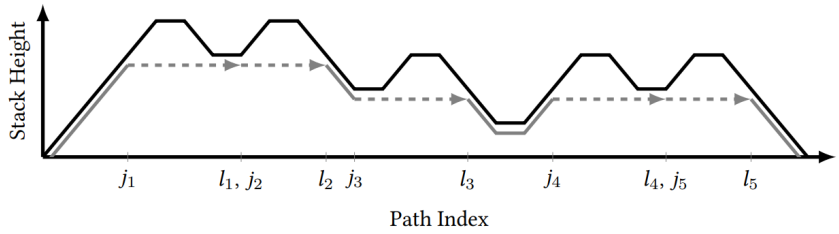
2. Язык Дика (почти) $\mathcal{O}(n^3)$ [25]

   Ещё более точный анализ алгоритм Репса [23], учитывающий, что построенный (для конкретного анализа) граф содержит константное число скобок

3. Язык Дика на одном типе скобок D_1 $\mathcal{O}(n^\omega \log^2 n)$ [21]

   Определение 1.1 (Bell-shaped путь)**.** Путь, который понимается строго вверх, потом сколько-то идёт ровно (ε-рёбра), потом спускается строго вниз

Ищем bell-shaped пути: удваиваем рёбра, ищем пути с серединкой из bell-shaped пути поменьше (так $\log n^2$ раз).



Сжимаем bell-shaped пути в ε-рёбра. Снова ищем и снова сжимаем. После каждого сжимания мы убираем все локальные максимумы. Чем больше был максимум, тем длиннее ε-ребро. Хуже всего, когда все новые рёбра длины 2. В любом случае путь становится короче хотя бы в 2 раза, так что таких итераций потребуется не более $\log n^2$ (есть лемма, что найдётся путь длины не более $\mathcal{O}(n^2)$).

4. Двунаправленные графы и язык Дика

   Существует несколько частных решений для задачи Диковой достижимости на двунаправленных графах:

   - Деревья [33]

     $\mathcal{O}(n \log n \log k)$ — центроиды + внутри что-то идейное

   - Общий случай [8]

     Решение основано на двух фактах. Первый: в двунаправленном графе формируются компоненты Диковой достижимости. Второй: если есть две вершины u, v и компонента Диковой достижимости C, такие что $u \xrightarrow{\alpha_i} C$ и $v \xrightarrow{\alpha_i} C$, то u и v тоже лежат в одной компоненте Диковой достижимости.

     Пользуясь этими фактами, алгоритм с помощью СНМ'а поддерживает компоненты Диковой достижимости и исходящие из них рёбра, чтобы быстро искать новые пары вершин, принадлежащих одной компоненте.

     Итоговая асимптотика алгоритма $\mathcal{O}(m + n\alpha(n))$.

   - Interleaved Dyck reachability

     Алгоритм за $\mathcal{O}(n^7)$ для $D_1 \bigodot D_1$ достижимости на bidirected графах: `https://helloqirun.github.io/papers/popl21_yuanbo.pdf`

     Было ещё про это (там, вроде, про один из языков сказали, что он bounded, поэтому можно пересекать с регулярным): `https://dl.acm.org/doi/pdf/10.1145/3296979.3192378`

5. Граф-цепочка $\mathcal{O}(n^\omega)$ [28]

   CFPQ на графе-цепочке — просто задача КС-распознавания (CF-recognition). А она решается за перемножение булевых матриц [28]

6. Ацикличный граф $\mathcal{O}(n^\omega)$ [30]

   Ацикличный граф — это почти бамбук (= цепочка), нужно только его потопсорить (и где-то ещё быть аккуратным, я не совсем помню сведение)

7. Bounded-stack RSM $\mathcal{O}(n^3k^3/\log^2)$ [11]

   RSM, который не уходит в рекурсию (т.е. есть из конца ребра $\xrightarrow{S}$ не достижимо никакое ребро $\xrightarrow{S}$)

   Тут применяется какое-то более хитрое (я ещё не разбиралась) итеративное транзитивное замыкание (что-то с dfs'ом, а потом ещё 4 русских сверху, кажется)

8. Hierarchical FSM $\mathcal{O}(n^\omega k^\omega)$ [11]

   RSM, в котором боксы упорядочены (топсорт) и бокс с меньшим номером содержит рёбра только с вызовами боксов с большим номером. Задают регулярный язык, но размер FSM может быть экспоненциальным относительно размера RSM.

   Алгоритм идёт в порядке, обратном топсорту, и считает транзитивное замыкание внутри бокса, чтобы провести все рёбра, которые ему соответствуют.

1.3. Выводы и результаты по главе

2. Алгоритм, основанный на пересечении грамматик

Основные определения (пререквизиты)

Определение 2.1 (Конечный автомат (?))**.** НКА и ДКА

Определение 2.2 (Рекурсивный конечный автомат (РКА))**.** *Для простоты тут будет немного не такое определение, как в [3]*

Это набор компонент $M_1, M_2, \ldots, M_k$, где каждая компонента M_i — это пятёрка $\langle Q_i, \Sigma_i, En_i, Ex_1, \delta_i \rangle$, где

- Q_i — конечное множество состояний
- Σ_i — конечный алфавит
- $En_i \subset Q_i$ — множество начальных состояний
- $Ex_i \subset Q_i$ — множество конечных состояний
- $\delta_i \colon Q_i \times (\Sigma_i \cup \bigcup\limits_{j=1}^{k} En_i \times Ex_i) \to Q_i$ — функция перехода. У δ_i есть два типа переходов: *внутренние*, которые работают как обычные переходы в НКА и *рекурсивные*, которые делают вызов другой компоненты (при этом обозначая начальную и конечную вершину в ней).

Неформально, это набор компонент, каждая из которых представляет собой ДКА, на рёбрах которого могут быть "рекурсивные вызовы" других компонент.

TODO: картинка с примером

Определение 2.3 (Прямое произведение автоматов)**.**

Определение 2.4 (Транзитивное замыкание)**.**

Определение 2.5 (Инкрементальное транзитивное замыкание)**.**

2.1. Сведение к задаче достижимости для РКА

В данном разделе будет подробно описан алгоритм, предложенный в [12], в модификации которого будет состоять дальнейшая работа.

Главное идеей алгоритма является следующее замечание: любой помеченный граф можно рассматривать как НКА, в котором не обозначены начальное и конечные состояния. При этом, если зафиксировать конкретные вершины s и t как стартовое и конечное состояние, то полученный автомат будет задавать язык слов w, таких что существует путь из s в t, на котором читается w.

TODO: картинка с примером

Утверждение 2.1. *[18]*

Автомат A (НКА/ДКА), построенный как прямое произведение автоматов A_1 и A_2 ($A = A_1 \otimes A_2$), распознаёт язык, равный пересечению языков A_1 и A_2 ($\mathcal{L}(A) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$)

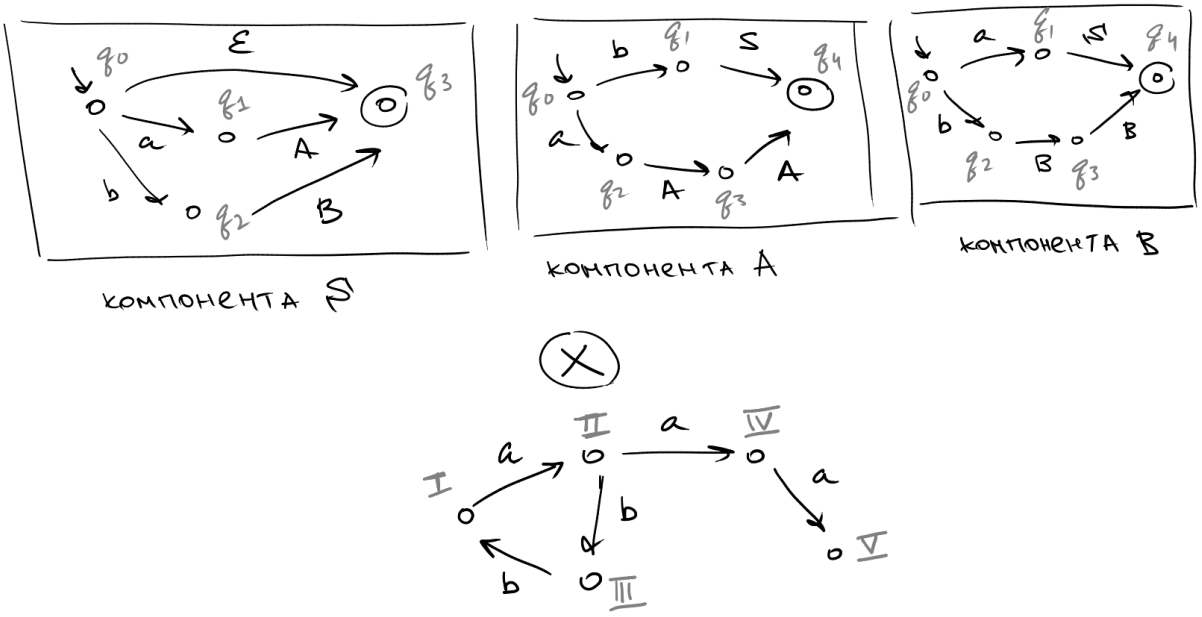
Данное утверждение остаётся верным, если один из языков задан РКА [6].

Пример 2.1 (Построение пересечения РКА и помеченного графа (НКА))**.** *РКА для грамматики, задающей язык слов, содержащих равное число букв a и b. Может быть задана следующими продукциями:*

$S \to \varepsilon \mid aA \mid bB$

$A \to bS \mid aAA$

$B \to aS \mid bBB$



TODO*: дорисовать пример (а потом перерисовать)*

TODO: Доказательство корректности сведения (?)

TODO: сделать на него ссылки везде (не только на него)

Все алгоритмы для CFPQ, которые будут описаны в этой работе имеют следующую схему:

1. Построим прямое произведение входной грамматики $\mathcal{R}$ и входного графа G: $\mathcal{P} = \mathcal{R} \otimes G$.

2. *Решим задачу достижимости для полученного РКА $\mathcal{P}$*

3. Из вершины u в вершину v входного графа существует путь, выводимой входной грамматикой $\mathcal{G} \Leftrightarrow$ в $\mathcal{P}$ есть путь из стартового состояния (q_0, u) в конечное состояние (q_f, v)

Рассмотрим внимательнее второй пункт — задачу достижимости для РКА. В случае обычного автомата эта задача эквивалентна задаче построения транзитивного замыкания [30]. В случае же РКА задача осложняется наличием рекурсивных вызовов, которые разрешаются итеративно. (??)

2.2. Алгоритм П

В листинге 1 приведён псевдокод Алгоритма П.

TODO: что-то написать про епс-переходы

Listing 1 Алгоритм достижимости для РКА

```
1:  function RSMReachability(R)
2:      A ← Adjacency matrix for R
3:      while A is changing do
4:          A' ← transitiveClosure(A)            ▷ Построение транзитивного замыкания
5:          for i ∈ 1..k do
6:              for u ∈ En_i do
7:                  for v ∈ Ex_i do
8:                      if A'_{u,v} ∧ ¬A_{u,v} then
9:                          A' ← A' ∪ getEdges(i, u, v)        ▷ Добавление новых рёбер
10:         A → A'
11:     return A
```

Работа происходит над матрицей смежности $\mathcal{R}$ — изначально туда записываются все “внутренние” (нерекурсивные) рёбра.

Далее, внешний цикл повторяется, пока матрица смежности A меняется (т.е. пока добавляются новые рёбра). На каждой итерации считается A' — транзитивное замыкание A. После этого находятся все новые пути вида $\langle$стартовое состояние$\rangle \rightsquigarrow \langle$конечное состояние$\rangle$ — те рёбра между стартовой и конечной вершинами компоненты, которых не было в A, но которые есть в A' — и добавляются соответствующие этим путям рёбра: для нового пути $(u \in En_i) \rightsquigarrow (v \in Ex_i)$ проводятся все рёбра, соответствующие рекурсивным вызовам i-ой компоненты с начальной вершиной u и конечной вершиной v.

2.2.1. Время работы

Время работы — $k \cdot T(n)$, где k — число итераций внешнего цикла, $T(n)$ — время работы одной итерации.

Оценим $T(n)$. Внутренняя часть цикла состоит из двух частей: нахождения транзитивного замыкания (строка 4) и прохода по матрице для выявления новых рёбер (строки 5-9).

Задача поиска транзитивного замыкания эквивалентна задаче перемножения булевых матриц [1] и может быть решена сведением к быстрому перемножению (обычных) матриц за $\mathcal{O}(n^\omega)$, где $2 < \omega < 2.273$ [2].

Проход по матрице (строки 5-7) работает за $\mathcal{O}(n^2)$, что доминируется временем построения транзитивного замыкания. Добавление новых рёбер (строки 8-9) отработает суммарно за $\mathcal{O}(n^2)$ (т.к. каждое ребро будет добавлено не более одного раза).

Итого, время работы алгоритма $\mathcal{O}(k \cdot n^\omega)$.

2.3. Алгоритм П2

Можно заметить, что не очень осмысленно на каждой итерации заново считать транзитивное замыкание, достаточно искать только пути, проходящие через рёбра, добавленные непосредственно на предыдущей итерации. То есть достаточно решать задачу **инкрементального** транзитивного замыкания.

В листинге 2 приведён псевдокод Алгоритма П2 (основанного на инкрементальном ТЗ)

Listing 2 Алгоритм достижимости для РКА (2)

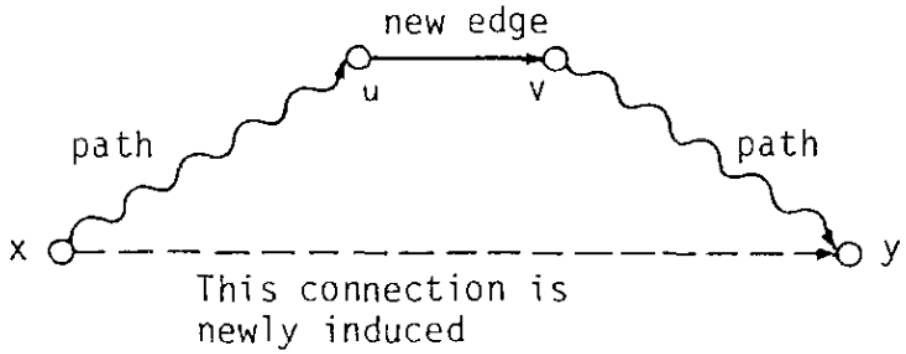
```
1:  function RSMReachability2(R)
2:      A ← Empty adjacency matrix
3:      Q ← Empty Queue
4:      for i ∈ 1..k do
5:          for u -c-> v ∈ δ_i do
6:              Q.Push(⟨u, v, i⟩)
7:      while Q is not Empty do
8:          ⟨u, v, i⟩ ← Q.Pop()
9:          if u ∈ En_i ∧ v ∈ En_i then                ▷ Нашли новый путь
10:             A ← A ∪ getEdges(i, u, v)
11:             Q.PushAll(getEdges(i, u, v))           ▷ Добавляем новые рёбра
12:         for x ∈ Q_i do
13:             if A_{x,u} ∧ ¬A_{x,v} then
14:                 for y ∈ Q_i do
15:                     if A_{v,y} ∧ ¬A_{x,y} then
16:                         A ← A ∪ ⟨x, y⟩
17:                         Q.Push(⟨x, y, i⟩)          ▷ Обновлем транзитивное замыкание
18:     return A
```

В алгоритме используется (более менее) стандартная реализация инкрементального транзитивного замыкания [19]. Для этого в ходе работы алгоритма поддерживается

рабочая очередь Q рёбер транзитивного замыкания, которые были найдены, но ещё не обработаны.

При обработке очередного *(потому что оно из очереди ахахах)* ребра, ищутся новые пути, которые проходя через него. А именно, пусть было добавлено ребро $u \to v$. Тогда далее перебирается вершина x, такая что из неё была достижима вершина u ($x \rightsquigarrow u$), но не была достижима вершина v ($x \not\rightsquigarrow v$). Из такой вершины x становятся достижимы все вершины y, которые были достижимы из v ($v \rightsquigarrow y$).

Также, как и в Алгоритме П, если ребро ТЗ (= путь в графе) соединяет начальную и конечную вершину, в очередь добавляются также все соответствующие ему рекурсивные рёбра.



TODO: норм картинка

2.3.1. Время работы

Внешний цикл итерируется по всем рёбрам, так что работает за $\mathcal{O}(m^*)$. Добавление новых рёбер как и в алгоритме П рассмотрит каждое ребро не более одного раза, так что тоже работает за $O(m^*)$. Нужно только оценить работу по поддержанию транзитивного замыкания.

Цикл на строке 12 перебирает все вершины, так что отработает суммарно за $\mathcal{O}(m^*n)$. Внутренний цикл (строка 14) тоже перебирает все вершины, но после его выполнения будет добавлено хотя бы одно новое ребро ($x \to v$), так что суммарное время работы этих циклов также можно оценить как $\mathcal{O}(m^*n)$.

Итого, суммарное время работы составляет $\mathcal{O}(nm^*)$, что в плотных графах будет равно $\Theta(n^3)$.

Замечание. Время работы итеративного транзитивного замыкания можно ускорить в $\log n$ раз [11], воспользовавшись методом четырёх русских [24] (так как работа происходит над булевыми векторами).

2.4. Выводы и результаты по главе

3. Результаты (1)

Основой для получения частных решений будут модификации Алгоритмов П и П2.

Модифицируем Алгоритм П2

Как уже было сказано, узким местом Алгоритма П2 является построение инкрементального транзитивного замыкания, которое в общем случае нельзя (скорее всего) решить быстрее, чем за кубическое время.

Следовательно, чтобы получить более быстрый алгоритм в частном случае, нужно рассматривать такие частные случаи, для которых задачу инкрементального транзитивного замыкания можно решать быстрее.

Рассмотрим сначала всякие несложные случаи:

- Графы с ограниченной степенью

  В [31] представлен алгоритм для инкрементального транзитивного замыкания на графах с ограниченной исходящей степенью. Асимптотика алгоритма: $\mathcal{O}(dm^*)$, где d — ограничение сверху на исходящую степень графа, а m^* — число рёбер в транзитивном замыкании.

- Планарные (?)

  TODO: Разобраться, что там написала Александра

- Неориентированные графы

  Вот тут получаем нетривиальные результаты, про них в следующем разделе

Для неориентированных графов отношение достижимости симметрично и на самом деле это отношение “принадлежать одной компоненте связности”. Поддерживать добавление рёбер и проверку связности в неориентированном графе может СНМ.

3.1. Алгоритм, основанный на неориентированном транзитивном замыкании

Я буду называть его Алгоритм НП

В листинге 3 приведён псевдокод Алгоритма НП.

Listing 3 Алгоритм достижимости для РКА, основанный на неориентированном ТЗ

```
1: function UndirectedRSMReachability(R)
2:     A ← Empty adjacency matrix
3:     Q ← Empty Queue
4:     D ← DSU(| ⋃_{i=1}^{k} Q_i |)
5:     for i ∈ 1..k do
6:         for u -c-> v ∈ δ_i do
7:             Q.Push(⟨u, v, i⟩)
8:     while Q is not Empty do
9:         ⟨u, v, i⟩ ← Q.Pop()
10:        if u ∈ En_i ∧ v ∈ En_i then                ▷ Нашли новый путь
11:            A ← A ∪ getEdges(i, u, v)
12:            Q.PushAll(getEdges(i, u, v))
13:            D.Union(u, v)                          ▷ Добавляем новые рёбра
14:    return A
```

Для реализации алгоритма используются две вспомогательные структуры данных: очередь Q, хранящая рёбра, которые были добавлены в граф, но ещё не обработаны (как и в оригинальном алгоритме П2), и СНМ D, поддерживающая компоненты связности и поиск новых путей $\langle$стартовое состояние$\rightarrow$конечное состояние$\rangle$.

Опишем подробно структуру используемого СНМ (в листинге 4 приведён псевдокод.

Listing 4 Система Непересекающихся Множеств

```
1:  Structure DisjointSets
2:      function DisjointSets(V)
3:          for v ∈ V do
4:              P[v] ← v                                          ▷ Предкок
5:              R[v] ← 0                                          ▷ Ранг
6:          for v ∈ En(V) do
7:              En[v] ← {v}                  ▷ Список стартовых вершин поддерева
8:          for v ∈ Ex(V) do
9:              Ex[v] ← {v}                  ▷ Список конечных вершин поддерева
10:     function Find(v)
11:         if P[v] = v then return v
            return P[v] = Find(P[v])                  ▷ Эвристика сжатие путей
12:     function Union(u, v)
13:         u ← Find(u)
14:         v ← Find(v)
15:         if u = v then return
16:         if R[u] > R[v] then
17:             Swap(u, v)                                ▷ Ранговая эвристика
18:         Q.PushAll({⟨en_u, ex_v⟩ | en_u ∈ En[u], ex_v ∈ Ex[v]})
19:         Q.PushAll({⟨en_v, ex_u⟩ | en_v ∈ En[v], ex_u ∈ Ex[u]})   ▷ Добавление новых
    рёбер
20:         En[v] ← En[v] ∪ En[u]
21:         Ex[v] ← Ex[v] ∪ Ex[u]
22:         R[v] ← max(R[v], R[u] + 1)
23:         P[u] = v                                          ▷ Объединение компонент
24:
```

За основу взята стандартная реализация [17] на подвешенные деревья, использующая обе эвристики: сжатие путей и ранговую.

Дополнительно в корнях хранятся списки всех начальных и конечных состояний компоненты. При добавлении ребра в операции *Join* перебираются все пары начальная/конечная вершина из двух компонент и соответствующие им рёбра добавляются в рабочую очередь Q.

TODO: (подумать) можно ли добавлять сразу много рёбер и сжимать их дфсом (как Борувка)?

3.2. Время работы

Теорема 3.1. *На РКА из n состояний и m^* рёбрах в транзитивном замыкании, Алгоритм НП отработает за время $\mathcal{O}(n + m^*\alpha(m^* + n, n))$*

Доказательство.
TODO: *m^* джойнов, а проходы по спискам не долгие, так как каждый раз генерируем новое ребро* □

3.3. Корректность для неориентированных графов и некоторых классов грамматик

TODO: может, в мусорку этот subsection?

3.4. Корректность для двунаправленных графов и языка Дика

Напоминание, что для языка Дика контекстно-свободная достижимость → Дикова достижимость

Для доказательства потребуется следующее вспомогательное утверждение

Лемма 3.1. *Для вершин двунаправленных графов отношении Диковой достижимости является отношением эквивалентности.*

Доказательство.
TODO: ну тут тупо

□

Замечание. Для данного алгоритма будем использовать следующий вид грамматики для языка Дика:

$S \to \varepsilon \mid SS \mid ({}_1S)_1 \mid \ldots \mid ({}_kS)_k$

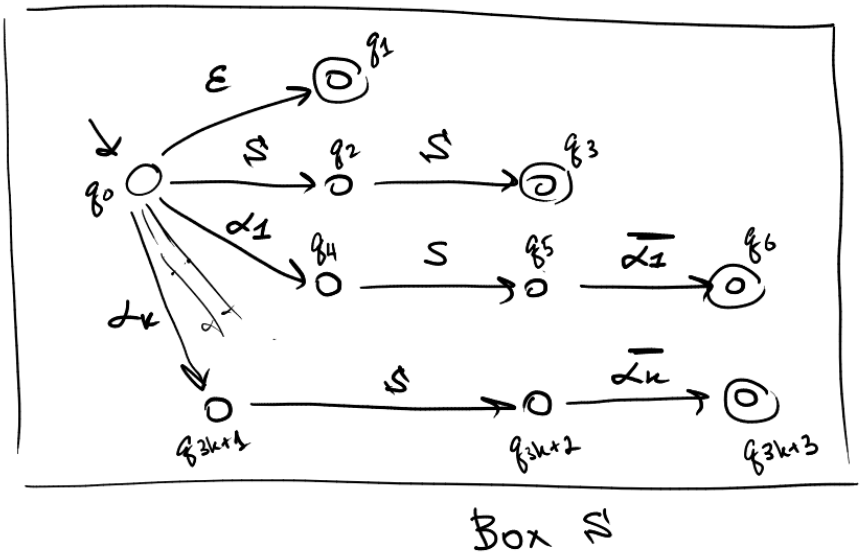


Рис. 1: РКА для языка Дика

На рисунке 1 приведена РКА для данной грамматики. Заметим, что он содержит всего одну компоненту (S).

Теорема 3.2. *Решение для CFPQ, использующее Алгоритм НП работает корректно на двунаправленных графах и языке Дика.*

Доказательство. Достаточно доказать, что для любой пары состояний $u \in En_i, v \in Ex_i$ существование неориентированного пути эквивалентно существованию ориентированного.

$\Leftarrow$ (ориентированный $\Rightarrow$ неориентированный)

Очевидно, если есть ориентированный путь $u \rightsquigarrow v$, то ровно если убрать ориентацию этот путь никуда не денется.

$\Rightarrow$ (неориентированный $\Rightarrow$ ориентированный)

At first, note that the Kronecker product $G \otimes \mathcal{G}$ forms some kind of a layered structure — i-th layer consists of vertices (q_i, v), where q_i is i-th RSM state. Because RSM is topologically sorted (TODO), every edge $(q_i, u) \to (q_j, v)$ goes forward.

We will call path *simple* if it visits every layer no more than once.

We prove the claim by induction on the l (path length).

Clearly the result is true for $l \leqslant 3$, because the only way to achive final vertex in $1, 2$ or 3 edges is by a simple vertical path (which exists in the original graph too).

Otherwise (if $l \geqslant 4$), path is not simple.

Consider the first flex point of the path, that is the vertex (q_i, v) such that edges $(q_j, u) \to (q_i, v)$ and $(q_i, v) \to (q_k, w)$ are in the path and $j, k \leqslant i$ (so, the path is convex at this point).

Looking at the grammar graph we can notice, that every state has indegree $\leqslant 1$. So at the flex point there are actually two same-labeled edges (that is, $j = k$).

There can be three different types of labels on those edges:

- α_l-label

  path: $(q_0, u) \to (q_i, v) \to (q_0, w) \to \cdots \to (q_f, z)$.

  Since α_l-labeled edges could only be added on the initialization stage, graph $\mathcal{G}$ contains edges $u \xrightarrow{\alpha_l} v$ and $w \xrightarrow{\alpha_l} v$. Notice, that cause $\mathcal{G}$ is bidirected, it also has to contain edges $v \xrightarrow{\overline{\alpha_l}} u$ and $v \xrightarrow{\overline{\alpha_l}} w$.

  Now we can notice, that w is Dyck-reachable (by the path $\alpha_l\overline{\alpha_l}$) from u, so there is an S-labeled edge from u to w. We can also conclude, that (by induction) there is a directed path from (q_0, w) to (q_f, z) (there z is the end of the path and q_f is some final state of G), so there is an S-labeled edge from w to z.

  Using this two observation we can construct a directed path from u to z: $u \xrightarrow{S} w \xrightarrow{S} z$.

- S-label

  path: $(q_0, a) \to \cdots \to (q_j, u) \to (q_i, v) \to (q_j, w) \to \cdots \to (q_f, z)$.

  $\mathcal{G}$ contains S-labeled edges $u \xrightarrow{S} v$ and $w \xrightarrow{S} v$. Since $\mathcal{G}$ is bidirected, then by **??** $v \xrightarrow{S} u$ and $v \xrightarrow{S} w$. Combining $u \xrightarrow{S} v$ and $v \xrightarrow{S} w$ we get that $u \xrightarrow{S} w$.

  No we want to sort of contract this edge, joining u and w (on the j-th level). Then we can get (by induction hypothesis) the directed path from $(q_0, a) \rightsquigarrow (q_f, z)$. If new path does not contain joined uw vertex, then that's the answer. Otherwise we can split this vertex back, inserting between u and w the S-labeled path (that one, from $u \xrightarrow{S} w$ edge). We can do it, because the both of these paths form correctly matched parenthesis (TODO: we can prove this using stack-based checking algorithm).

  Two other cases can be proved the same way, but I find it a little dishonest

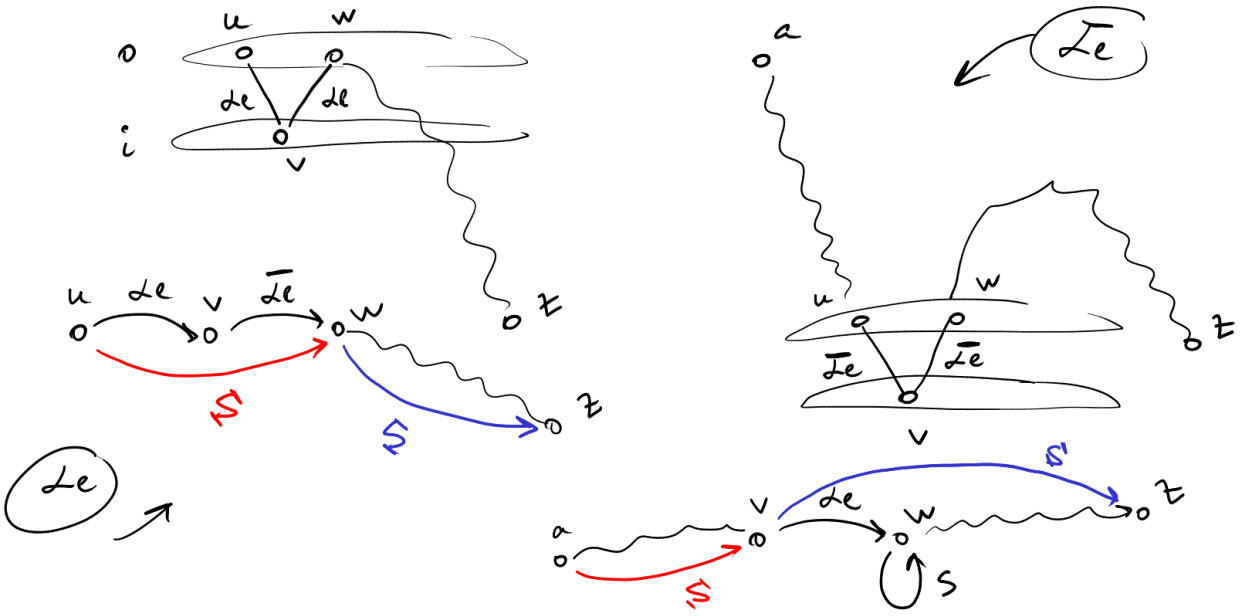
- $\overline{\alpha_l}$-label

  path: $(q_0, a) \to \cdots \to (q_j, u) \to (q_f, v) \to (q_j, w) \to \cdots \to (q_f, z)$.

  Since α_l-labeled edges could only be added on the initialization stage, graph $\mathcal{G}$ contains edges $u \xrightarrow{\overline{\alpha_l}} v$ and $w \xrightarrow{\overline{\alpha_l}} v$. Notice, that cause $\mathcal{G}$ is bidirected, it also has to contain edges $v \xrightarrow{\alpha_l} u$ and $v \xrightarrow{\alpha_l} w$.

  By induction, we get that $a \xrightarrow{S} v$. Now we will construct a second part of the path: $(q_0, v) \xrightarrow{\alpha_l} (q_{j-1}, w) \xrightarrow{S} (q_j, w) \rightsquigarrow (q_f, z)$ $(q_{j-1}, w) \xrightarrow{S} (q_j, w)$ — initial S-loop). By induction, we have directed simple version of this path, so $v \xrightarrow{S} z$.

  Combining this two paths ($a \xrightarrow{S} v$ and $v \xrightarrow{S} z$) we get $a \xrightarrow{SS} z \Rightarrow a \xrightarrow{S} z$ — desired path.


u
w
i
v
z
a
u
v
w
z
s
s'
s

□

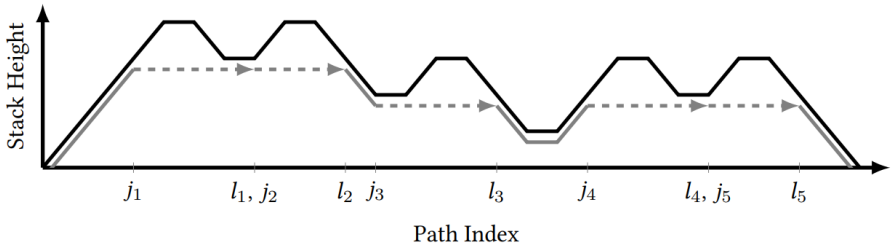
4. Результаты (2)

4.1. Алгоритм для языка Дика на одном типе скобок

Алгоритм из статьи [21]

Определение 4.1 (Bell-shaped путь)**.** Путь, который понимается строго вверх, потом сколько-то идёт ровно (ε-рёбра), потом спускается строго вниз

Ищем bell-shaped пути: удваиваем рёбра, ищем пути с серединкой из bell-shaped пути поменьше (так $\log n^2$ раз).



Сжимаем bell-shaped пути в ε-рёбра. Снова ищем и снова сжимаем. После каждого сжимания мы убираем все локальные максимумы. Чем больше был максимум, тем длиннее ε-ребро. Хуже всего, когда все новые рёбра длины 2. В любом случае путь становится короче хотя бы в 2 раза, так что таких итераций потребуется не более $\log n^2$ (есть лемма, что найдётся путь длины не более $\mathcal{O}(n^2)$).

4.1.1. Алгоритм

Для построения алгоритма воспользуемся следующим результатом:

Лемма 4.1. *[13]*

Для языка L определим $p_L(n)$ [4] — максимальная длина кратчайшего слова в $L \cap K$ по всем регулярным языкам K, задаваемым НКА с $\leqslant n$ состояниями.

Тогда для языка Дика D_1 на одном типе скобок $p_{D_1}(n) = \mathcal{O}(n^2)$

Следствие 4.0.1. *Для любой пары вершин $u, v \in V(G)$, если есть Диков путь $u \rightsquigarrow v$, то существует и Диков путь $u \rightsquigarrow v$, длина которого $\mathcal{O}(n^2)$.*

Доказательство. Слова, читаемые на путях $u \rightsquigarrow v$ задаются НКА на n вершинах — графом G, в котором u и v выбраны за начальное и конечное состояния соответственно. □

Пользуясь этим фактом, можно соорудить наивный алгоритм — достаточно лишь заметить, что раз длина искомого пути всегда ограничена, то можно задать такие

[4] Формально, $p_L(n) = \max\{\min\{|w|\colon w \in L \cap K\}, K \in Rat_n, L \cap K \neq \varnothing\}$,
где $Rat_n(X)$ — регулярные языки над алфавитом X, распознаваемые НКА с $\leqslant n$ состояниями.

пути с помощью ДКА (язык D_1 задаётся автоматом с одним счётчиком, значение счётчика не превышает $\mathcal{O}(n^2)$, так что его можно закодировать в состояние) на $\mathcal{O}(n^2)$ состояниях.

TODO: дописать красиво

Ну тут мотивация простая, регулярный язык у нас большой получается, зато КС-грамматика маленькая

TODO: написать красивые слова

КС-грамматика, задающая

4.1.2. Время работы

TODO: Улучшить обычную оценку (ищем ТЗ отдельно в каждой компоненте РКА)

Замечание. Этот результат не обобщается на языки Дика с большим типом скобок.

Мотивация примерно такая: во1, они сложнее (язык Дика на $\geqslant 2$ типах скобок — такой же мощный как и произвольный КС-язык (теорема Хомского-Шутценбергера, тогда как язык Дика на одном типе скобок вроде как попроще. *Ещё Дик на $\geqslant 2$ типах скобок генерирует full AFL (Abstract Families of Languages) (что бы это не значило)*), во2, сейчас мы играли на том, что состояние — примерно одна чиселка (ну опять-таки, язык Дика на одном типе скобок распознаётся автоматом с одним счётчиком), а для большего типа скобок нужен весь стек.

Список литературы

[1] Aho Alfred V., Hopcroft John E. The Design and Analysis of Computer Algorithms. — 1st edition. — USA : Addison-Wesley Longman Publishing Co., Inc., 1974. — ISBN: 0201000296.

[2] Alman Josh, Williams Virginia Vassilevska. A Refined Laser Method and Faster Matrix Multiplication. — 2020. — 2010.05846.

[3] Analysis of Recursive State Machines / Rajeev Alur, Michael Benedikt, Kousha Etessami et al. — 2005. — Jul. — Vol. 27, no. 4. — P. 786–818. — Access mode: https://doi.org/10.1145/1075382.1075387.

[4] Azimov Rustam, Grigorev Semyon. Context-Free Path Querying by Matrix Multiplication // Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences amp; Systems (GRADES) and Network Data Analytics (NDA). — GRADES-NDA '18. — New York, NY, USA : Association for Computing Machinery, 2018. — Access mode: https://doi.org/10.1145/3210259.3210264.

[5] Barrett Chris, Jacob Riko, Marathe Madhav. Formal-Language-Constrained Path Problems // SIAM J. Comput. — 2000. — 01. — Vol. 30. — P. 809–837.

[6] Beigel Richard, Gasarch William. A Proof that if L = L1 ∩ L1 where L1 is CFL and L2 is Regular then L is Context Free Which Does Not use PDA's.

[7] Ceri Stefano, Gottlob Georg, Tanca Letizia. What you Always Wanted to Know About Datalog (And Never Dared to Ask). // Knowledge and Data Engineering, IEEE Transactions on. — 1989. — 04. — Vol. 1. — P. 146 – 166.

[8] Chatterjee Krishnendu, Choudhary Bhavya, Pavlogiannis Andreas. Optimal Dyck Reachability for Data-Dependence and Alias Analysis // Proc. ACM Program. Lang. — 2017. — Dec. — Vol. 2, no. POPL. — Access mode: https://doi.org/10.1145/3158118.

[9] Chaudhary Anoop, Faisal Abdul. Role of graph databases in social networks. — 2016. — 06.

[10] CFL-reachability in subcubic time : Rep. / Technical report, IBM Research Report RC24126 ; Executor: Swarat Chaudhuri : 2006.

[11] Chaudhuri Swarat. Subcubic Algorithms for Recursive State Machines. — POPL '08. — New York, NY, USA : Association for Computing Machinery, 2008. — P. 159–169. — Access mode: https://doi.org/10.1145/1328438.1328460.

[12] Context-Free Path Querying by Kronecker Product / Egor Orachev, Ilya Epelbaum, Rustam Azimov, Semyon Grigorev // Advances in Databases and Information Systems / Ed. by Jérôme Darmont, Boris Novikov, Robert Wrembel. — Cham : Springer International Publishing, 2020. — P. 49–59.

[13] Deleage Jean-Luc, Pierre Laurent. The Rational Index of the Dyck Language $D_1'^*$ // Theor. Comput. Sci. — 1986. — Nov. — Vol. 47, no. 3. — P. 335–343.

[14] An Experimental Study of Context-Free Path Query Evaluation Methods / Jochem Kuijpers, George Fletcher, Nikolay Yakovets, Tobias Lindaaker. — SSDBM '19. — New York, NY, USA : Association for Computing Machinery, 2019. — P. 121–132. — Access mode: https://doi.org/10.1145/3335783.3335791.

[15] Heintze N., McAllester D. On the cubic bottleneck in subtyping and flow analysis // Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science. — 1997. — P. 342–351.

[16] Hellings Jelle. Path Results for Context-free Grammar Queries on Graphs // CoRR. — 2015. — Vol. abs/1502.02242. — 1502.02242.

[17] Hopcroft John E., Ullman Jeffrey D. Set merging algorithms // SIAM Journal on Computing. — 1973. — Vol. 2, no. 4. — P. 294–303.

[18] Hopcroft John E, Ullman Jeffrey D. An introduction to automata theory, languages, and computation. — Upper Saddle River, NJ : Pearson, 1979.

[19] Ibaraki T., Katoh N. On-line computation of transitive closures of graphs // Information Processing Letters. — 1983. — Vol. 16, no. 2. — P. 95–97. — Access mode: https://www.sciencedirect.com/science/article/pii/0020019083900339.

[20] Kodumal John, Aiken Alex. The Set Constraint/CFL Reachability Connection in Practice. — PLDI '04. — New York, NY, USA : Association for Computing Machinery, 2004. — P. 207–218. — Access mode: https://doi.org/10.1145/996841.996867.

[21] Mathiasen Anders Alnor, Pavlogiannis Andreas. The Fine-Grained and Parallel Complexity of Andersen's Pointer Analysis // Proc. ACM Program. Lang. — 2021. — Jan. — Vol. 5, no. POPL. — Access mode: https://doi.org/10.1145/3434315.

[22] Medeiros Ciro M., Musicante Martin A., Costa Umberto S. Efficient Evaluation of Context-Free Path Queries for Graph Databases // Proceedings of the 33rd Annual ACM Symposium on Applied Computing. — SAC '18. — New York, NY, USA : Association for Computing Machinery, 2018. — P. 1230–1237. — Access mode: https://doi.org/10.1145/3167132.3167265.

[23] Melski David, Reps Thomas. Interconvertibility of a class of set constraints and context-free-language reachability // Theoretical Computer Science. — 2000. — Vol. 248, no. 1. — P. 29–98. — PEPM'97. Access mode: https://www.sciencedirect.com/science/article/pii/S0304397500000499.

[24] On economical construction of the transitive closure of an oriented graph / Vladimir L'vovich Arlazarov, Yefim A Dinitz, MA Kronrod, IgorAleksandrovich Faradzhev // Doklady Akademii Nauk / Russian Academy of Sciences. — 1970.

[25] Rehof Jakob, Fähndrich Manuel. Type-Base Flow Analysis: From Polymorphic Subtyping to CFL-Reachability // Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — POPL '01. — New York, NY, USA : Association for Computing Machinery, 2001. — P. 54–66. — Access mode: https://doi.org/10.1145/360204.360208.

[26] Santos Fred C., Costa Umberto S., Musicante Martin A. A Bottom-Up Algorithm for Answering Context-Free Path Queries in Graph Databases // Web Engineering / Ed. by Tommi Mikkonen, Ralf Klamma, Juan Hernández. — Cham : Springer International Publishing, 2018. — P. 225–233.

[27] Sevon Petteri, Eronen Lauri. Subgraph Queries by Context-free Grammars // Journal of Integrative Bioinformatics. — 2008. — Vol. 5, no. 2. — P. 157–172. — Access mode: https://doi.org/10.1515/jib-2008-100.

[28] Valiant Leslie G. General context-free recognition in less than cubic time // Journal of computer and system sciences. — 1975. — Vol. 10, no. 2. — P. 308–315.

[29] Wikipedia contributors. Datalog — Wikipedia, The Free Encyclopedia. — 2021. — [Online; accessed 6-May-2021]. Access mode: https://en.wikipedia.org/w/index.php?title=Datalog&oldid=1015453010.

[30] Yannakakis Mihalis. Graph-Theoretic Methods in Database Theory // Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. — PODS '90. — New York, NY, USA : Association for Computing Machinery, 1990. — P. 230–242. — Access mode: https://doi.org/10.1145/298514.298576.

[31] Yellin Daniel M. Speeding up Dynamic Transitive Closure for Bounded Degree Graphs // Acta Inf. — 1993. — Apr. — Vol. 30, no. 4. — P. 369–384. — Access mode: https://doi.org/10.1007/BF01209711.

[32] Younger Daniel H. Recognition and parsing of context-free languages in time n^3 // Information and Control. — 1967. — Vol. 10, no. 2. — P. 189–208. — Access mode: https://www.sciencedirect.com/science/article/pii/S001999586780007X.

[33] Yuan Hao, Eugster Patrick. An Efficient Algorithm for Solving the Dyck-CFL Reachability Problem on Trees // Programming Languages and Systems / Ed. by Giuseppe Castagna. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2009. — P. 175–189.

[34] Zarrinkalam Fattane, Kahani Mohsen, Paydar Samad. Using graph database for file recommendation in PAD social network // 7'th International Symposium on Telecommunications (IST'2014). — 2014. — P. 470–475.