Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б07-мм

Оптимизация алгоритма решения задачи достижимости с ограничениями в виде регулярных языков

Белянин Георгий Олегович

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
доцент кафедры системного программирования, к. ф.-м. н., Григорьев С. В.

Санкт-Петербург
2024

Оглавление

Введение

Графовые базы данных за последние годы становятся всё более популярным инструментом для анализа баз знаний [8], биологических данных [7], семантической паутины [5] и исследовании исходного кода [17]. Данные в них представляются графами с метками на узлах и рёбрах, при этом каждая вершина является отдельным объектом, а каждое ребро — отношением. Естественным образом при анализе таких данных появляется вопрос, какие объекты находятся с заданным в определённых отношениях. Для ответа на него необходимо искать пути между узлами-объектами, удовлетворяющие некоторым ограничениям. Такие ограничения часто классифицируют как формальные языки, разделяя их на регулярные, контекстно-свободные, контекстно-зависимые и неограниченные. Разновидности возможных меток на рёбрах при этом образуют алфавит. Каждый путь при этом образует слово, получающееся из меток рёбер содержащихся в нём. Класс регулярных ограничений можно воспринимать как класс языков. Он определяется рекурсивно и состоит из пустого языка, всех языков из одного односимвольного слова, конкатенации регулярных языков, их объединения и взятия звёздочки Клини, то есть языков, полученных при помощи применения конкатенации самим к себе произвольное число раз. На практике используется расширение регулярных ограничений — двусторонние регулярные ограничения [14, 12]. Они позволяют рассматривать псевдо-пути, содержащие движение в направлении противоположном существующим в графе рёбрам. Запросы с двусторонними регулярными ограничениями реализованы в языке SPARQL, разработанным W3C, и включены в ISO стандарт языков запросов к графам ISO/IEC 39075:2024[1], по реализации элементов которого публикуются работы [13, 19]. Несмотря на простоту, для большинства сценариев использования графовых баз данных запросы с такими ограничениями будут самыми частыми [20]. На данный момент не существует ни достаточно эффективного алгоритма для решения этой задачи, ни универсального способа представления графа, поэтому активно исследуются альтернативные методы [2, 19, 1].

Одним из перспективных подходов в графовых базах данных является представление данных разреженными матрицами смежности. Поскольку количество ненулевых элементов матриц смежности реальных графов сильно меньше числа нулей, их можно эффективно представить в памяти при помощи специальных структур данных и использовать высокопроизводительные параллельные алгоритмы вычисления матричных операций [10]. Недавно была опубликована работа, в которой исследуется реализация алгоритма достижимости с регулярными ограничениями с представлением графа разреженными матрицами смежности [1]. Однако исследуемая в ней реализация значительно проигрывает по производительности промышленным решениям. Кроме того, существует алгоритм, который, как и некоторые другие СУБД

[1]Стандарт ISO/IEC 39075:2024: https://www.iso.org/standard/76120.html (Дата доступа: 30.05.2024)

[4, 9], использует обход в ширину, но при этом выражает его операциями линейной алгебры над разреженными матрицами. Прототип этого подхода был разработан, но он не поддерживал двусторонние запросы, не был протестирован и адаптирован для применения на больших графах. Поэтому интересен анализ данного алгоритма, возможности модификации и уменьшения времени работы.

В данной работе предложены исправления, модификации и оптимизации алгоритма достижимости при помощи линейной алгебры на графах с регулярными ограничениями на метки в пути, а так же приводится анализ его быстродействия в сравнении с графовой СУБД MilleniumDB[2] [11].

[2]Репозиторий СУБД MilleniumDB: https://github.com/MillenniumDB/MillenniumDB (Дата доступа: 05.03.2024)

1. Постановка задачи

Целью работы является создание производительного алгоритма достижимости с регулярными ограничениями. Для достижения этой цели были поставлены следующие задачи.

1. Проверить и исправить существующую реализацию алгоритма решения задачи достижимости на графах с регулярными ограничениями на пути, основанного на представлении графа разреженными матрицами.
2. Реализовать поддержку двусторонних регулярных запросов.
3. Найти узкие места в алгоритме и уменьшить среднее время работы.
4. Сравнить время работы алгоритма со временем исполнения запросов графовой СУБД MilleniumDB.

2. Обзор

Для описания существующего алгоритма достижимости с регулярными ограничениями при помощи линейной алгебры необходимо ввести несколько определений и рассмотреть несколько утверждений. Для начала поймём, что мы имеем в виду под графом.

Определение 2.1. *Помеченный граф* (или просто *граф*) G — тройка $\langle V, E, L\rangle$, где:

- V — конечное множество вершин,
- E — конечное множество рёбер такое, что. $E \subseteq V \times L \times V$,
- L — конечное множество меток на рёбрах.

Введём функции, которые сопоставляют ребру стартовую вершину, конечную вершину и метку:

- $Start : E \mapsto V, Start((i, a, j)) = i$,
- $End : E \mapsto V, End((i, a, j)) = j$,
- $Label : E \mapsto L, Label((i, a, j)) = a$.

Определение 2.2. *Путём* P в графе $G = \langle V, E, L\rangle$ будем называть конечную (в том числе и пустую) последовательность рёбер $e_1, e_2, ..., e_n$ таких, что:

$$\forall i = 1..n - 1, End(e_i) = Start(e_{i+1}).$$

Один из способов задать граф — это задать его *матрицу смежности*.

Определение 2.3. Не умоляя общности, занумеруем конечное множество V натуральными числами от 1 до $|V|$. *Матрица смежности* $\mathcal{G}$ графа $G = \langle V, E, L\rangle$ — это квадратная матрица размером $|V| \times |V|$, для которой верно:

$$a \in \mathcal{G}_{ij} \iff \exists e \in E : Start(e) = i, End(e) = j, Label(e) = a.$$

Будем говорить, что матрица булева, если её элементы принадлежат множеству $\mathbb{B} = \{0, 1\}$, при этом определим сложение таких элементов как логическое ИЛИ, а умножение — как логическое И.

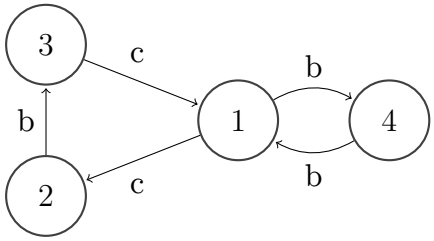
Замечание 2.1. Умножение матриц, элементами которой являются значения $\mathbb{B} = \{0, 1\}$ с операциями И, ИЛИ, ассоциативно. Так же умножение на диагональную матрицу коммутативно.

Замечание 2.2. Для представления графа $G = \langle V, E, L\rangle$ с матрицей смежности $\mathcal{G}$ булевыми матрицами матрицами смежности можно рассмотреть матрицы $\mathcal{G}^a, a \in L$ размера $|V| \times |V|$, где:

$$\mathcal{G}^a_{ij} = 1 \iff a \in \mathcal{G}_{ij}.$$

Такой набор матриц называется *булевой декомпозицией* матрицы смежности.

Пример 2.1. Будем рассматривать следующий граф G.



Матрица смежности $\mathcal{G}$ имеет следующий вид:

$$\mathcal{G} = \begin{pmatrix} . & \{c\} & . & \{b\} \\ . & . & \{b\} & . \\ \{c\} & . & . & . \\ \{b\} & . & . & . \end{pmatrix}$$

Её булева декомпозиция по каждому символу выглядит следующим образом:

$$\mathcal{G}^b = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \qquad \mathcal{G}^c = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Замечание 2.3. Для графа $G = \langle V, E, L \rangle$ с булевой декомпозиции матрицы смежности $\mathcal{G}^a$ и $v \in V$, строка $\mathcal{G}^a_v$ содержит значение 1 в столбцах, соответствующих вершинам, соединённых ребром, исходящим из v, то есть:

$$\mathcal{G}^a_{vj} = 1 \iff \exists e \in E, Start(e) = v, End(e) = j, Label(e) = a.$$

Утверждение 2.4. Рассмотрим граф $G = \langle V, E, L \rangle$ с булевой декомпозиции матрицы смежности $\mathcal{G}^a$. Зафиксируем множество вершин $V' \subset V$. Рассмотрим булеву матрицу $\mathcal{V}$ размером $1 \times |V|$ такую, что:

$$\mathcal{V}_{1v} = 1 \iff v \in V'$$

Тогда:

$$(\sum_{a \in L} \mathcal{V} \times \mathcal{G}^a)_{1v} = 1 \iff \exists e \in E : Start(e) \in V', End(e) = v.$$

Доказательство очевидно из замечания 2.3, поскольку такая матрица соответствует сумме строк с номерами ненулевых столбцов $\mathcal{V}$.

То есть при помощи такой конструкции из множества вершин графа можно получить множество всех вершин, соединённых с ними ребром.

Утверждение 2.5. Рассмотрим граф $G = \langle V, E, L \rangle$ с матрицей смежности $\mathcal{G}$. Для матрицы смежности $\mathcal{G}'$ графа $G' = \langle V, E', L \rangle$, где:

$$(i, a, j) \in E' \iff (j, a, i) \in E,$$

верно равенство:

$$\mathcal{G}' = \mathcal{G}^T.$$

Проще говоря, транспонирование матрицы смежности $\mathcal{G}$ «обращает» все рёбра графа G в противоположную сторону.

Доказательство очевидно и следует из определения матрицы смежности.

Так же для описания алгоритма необходимо разобраться с несколькими утверждениями из теории формальных языков. Введём несколько базовых определений.

Определение 2.4. *Алфавитом* Σ будем называть произвольное конечное множество, его элементы — *символами*.

Определение 2.5. *Словом* (или *строкой*) w над алфавитом Σ будем называть конечную (в том числе и пустую) последовательность символов $w = a_1 a_2 ... a_n, a_i \in \Sigma$.

Определение 2.6. *Языком* L над алфавитом Σ будем называть произвольное подмножество Σ^*, где Σ^* — множество всех слов (в том числе и пустого), над алфавитом Σ.

Определение 2.7. Будем называть *регулярными языками* над алфавитом Σ следующие множества:

- $\varnothing$, $\{\varepsilon\}$ и $\{t\}$, где $t \in \Sigma$, ε — пустое слово,
- $R_1 \cup R_2$, где R_1 и R_2 — регулярные языки,
- $R_1 \cdot R_2$, где R_1 и R_2 — регулярные языки, а $R_1 \cdot R_2 = \{r_1 r_2 \mid r_1 \in R_1, r_2 \in R_2\}$, где $r_1 r_2$ — конкатенация слов,
- R^*, где R — регулярный язык, а $R^* = \bigcup_{n=0}^{\infty} \underbrace{R \cdot R \cdot ... \cdot R}_{n \text{ раз}}$.

При этом никакие другие множества не являются регулярными языками.

Определение 2.8. *Регулярным выражением* над алфавитом Σ называются строки вида:

- ε, где ε — пустая строка,
- t, где $t \in \Sigma$,

- $(r_1 \mid r_2)$, где r_1 и r_2 — регулярные выражения,
- $(r_1 r_2)$, где r_1 и r_2 — регулярные выражения,
- (r^*), где r — регулярное выражение.

Определение 2.9. Введём отображение $J : \{r \mid r$ — регулярное выражение$\} \mapsto \{R \mid R$ — регулярный язык$\}$:

- $J(\varepsilon) = \{\varepsilon\}, J(t) = \{t\}$, где $t \in \Sigma$,
- $J((r_1 \mid r_2)) = J(r_1) \cup J(r_2)$,
- $J((r_1 r_2)) = J(r_1) \cdot J(r_2)$,
- $J((r^*)) = J(r)^*$.

Будем говорить, что регулярное выражение r *определяет язык* $R = J(r)$.

Теорема 2.6. Пересечение регулярных языков $R_1 \cap R_2$ — регулярный язык.

Доказательство данного факта доступно в литературе по формальным языкам [21].

Определение 2.10. *Детерминированный конечный автомат* (сокращённо *ДКА*) D — пятёрка $\langle Q, q_S, Q_F, \delta, \Sigma \rangle$, где:

- Q — конечное множество состояний,
- $q_S \in Q$ — стартовое состояние,
- $Q_F \subseteq Q$ — множество финальных состояний,
- $\delta : Q \times \Sigma \mapsto Q$ — функция переходов, при этом она может быть определена частично,
- Σ — конечный алфавит.

Определение 2.11. Детерминированный конечный автомат $D = \langle Q, q_S, Q_F, \delta, \Sigma \rangle$ *допускает* строку $w = a_1 a_2 ... a_n$, где $a_i \in \Sigma$, если существует последовательность состояний $q_0, q_1, ..., q_n$ такая, что:

- $q_0 = q_S$,
- $\delta(q_i, a_i) = q_{i+1}$ при $i = 0, 1, ..., n-1$,
- $q_n \in Q_F$.

Определение 2.12. Автомат D *задаёт* язык $R = \{w \mid D$ допускает $w\}$.

Существует важная теорема, которая, в некотором смысле говорит об эквивалентности множества регулярных языков и множества детерминированных конечных автоматов.

Теорема 2.7. Множество регулярных языков над алфавитом Σ и множество языков задаваемых некоторыми детерминированными конечными автоматами над алфавитом Σ совпадают. Иначе говоря:

$$\{R \mid R - \text{регулярный язык}\} = \{L \mid \exists \text{ ДКА D, распознающий } L\}.$$

Доказательство данного факта доступно в литературе по формальным языкам [21].

Замечание 2.8. Каждое регулярное выражение определяет язык и некоторый конечный автомат, распознающий его.

Существует несколько подходов преобразования регулярных выражений в одну из разновидностей конечных автоматов. Наиболее прямолинейным является подход, заключающийся в получении по регулярному выражению тривиальным образом некоторой более общей разновидности конечного автомата: недетерминированного конечного автомата с переходами по пустым строкам (ε-НКА) [6]. Из такого автомата сначала убираются переходы по пустым строкам. В конце концов происходит преобразование в детерминированный конечный автомат .

Есть алгоритмы, позволяющие по регулярному выражению получить сразу детерминированный конечный автомат, задающий соответствующий выражению язык [16].

Для реализации можно выбрать любой из существующих подходов. Важным для данной работы является только факт того, что по любому регулярному выражению можно построить детерминированный конечный автомат.

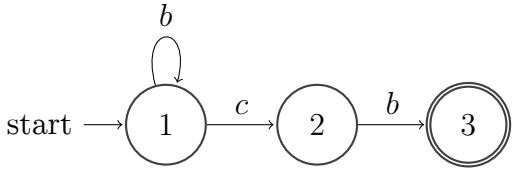
Разнообразие способов определить регулярный язык обусловлено тем, что регулярные выражения являются удобным для пользователя способом задать желаемый регулярный язык. А детерминированный конечный автомат является способом представить нужный язык в памяти компьютера путём хранения матриц смежности соответствующего ему графа.

Замечание 2.9. Детерминированный конечный автомат $D = \langle Q, q_S, Q_F, \delta, \Sigma \rangle$ может рассматриваться как тройка из стартовой вершины, множества конечных вершин и графа $G_D = \langle Q, E, \Sigma \rangle$, где:

$$(q_i, a, q_j) \in E \iff \delta(q_i, a) = q_j$$

Занумеруем состояния Q от 1 до $|Q|$. *Матрицей смежности конечного автомата* $\mathcal{D}$ будем называть матрицу смежности графа G_D.

Пример 2.2. Зададим регулярное выражение b^*cb, определяющее язык вида $\{cb, bcb, bbcb, bbbcb, ...\}$. Ему соответствует следующий детерминированный конечный автомат D:



Матрица смежности этого ДКА выглядит следующим образом:

$$\mathcal{D} = \begin{pmatrix} \{b\} & \{c\} & . \\ . & . & \{b\} \\ . & . & . \end{pmatrix}$$

Нам будет необходима булева декомпозиция этой матрицы:

$$\mathcal{D}^b = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad \mathcal{D}^c = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Определение 2.13. Рассмотрим граф $G = \langle V, E, L \rangle$. Зададим регулярное выражение над алфавитом L и назовём его *регулярным ограничением.*

Будем говорить, что путь $e_1, e_2, ..., e_n$ в графе G *удовлетворяет регулярным ограничениям*, если образованное конкатенацией меток на пути слово принадлежит определяемому регулярным выражением языку.

Иначе говоря $w \in R$, где $w = a_1a_2...a_n, a_i = Label(e_i)$, а R — регулярный язык, соответствующий регулярному ограничению.

2.1. Описание алгоритма достижимости с ограничениями

Рассмотрим теперь изучаемый алгоритм достижимости с регулярными ограничениями. На вход он принимает граф, детерминированный конечный автомат, определяемый регулярным ограничением, и множество начальных вершин. Результатом его работы будут вершины, для которых существует хотя бы один удовлетворяющий регулярным ограничениям путь, начинающийся в одной из стартовых вершин.

Для наглядности сначала рассмотрим пример работы алгоритма на графе и выражении из примеров выше.

Пример 2.3. Обозначим граф из примера 2.1 $G = \langle V, E, L \rangle$, его матрицу смежности — $\mathcal{G}$. Зададим регулярное выражение $r = b^*cb$. Возьмем детерминированный конечный

автомат $D = \langle Q, q_S, Q_F, \delta, \Sigma \rangle$ с матрицей смежности $\mathcal{D}$ из примера 2.2. Зададим множество из нескольких стартовых вершин $V_{src} = \{1\}$.

Зададим матрицу M размером $|Q| \times (|Q| + |V|)$, такую, что $M = \begin{pmatrix} S & T \end{pmatrix}$:

1. S — матрица $|Q| \times |V|$ вида $S_{ij} = \begin{cases} 1 & i = j = q_S \\ 0 & \text{иначе} \end{cases}$

2. T — матрица $|Q| \times |Q|$ вида $T_{ij} = \begin{cases} 1 & i = q_S, j \in V_{src} \\ 0 & \text{иначе} \end{cases}$

Каждая строка такой матрицы на самом деле соответствует двум различным множествам вершин: в столбцах от 1 до $|Q|$ содержится вершина графа ДКА, а в столбцах $|Q| + 1$ до $|Q| + |V|$ — вершины графа. Таким образом, в матрице обхода в ненулевых строках состоянию ДКА сопоставляется множество вершин графа.

В начале работы алгоритма матрица M имеет следующий вид:

$$M = \begin{array}{|ccc|cccc|} \hline 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

Будем называть эту матрицу *матрицей обхода*.

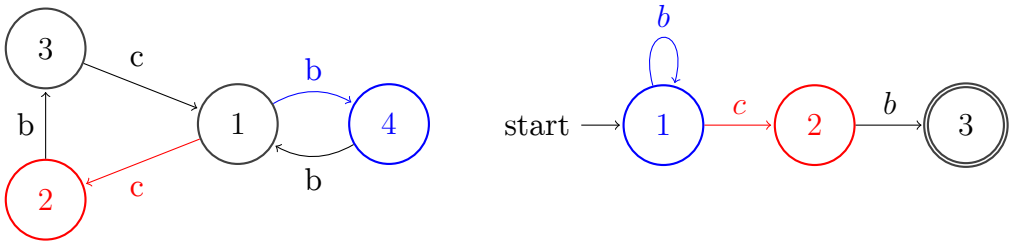
Умножим матрицу обхода M на матрицы $\mathcal{B}^a = \begin{pmatrix} \mathcal{D}^a & . \\ . & \mathcal{G}^a \end{pmatrix}$, $a \in L$. По замечанию 2.4 для каждой строки отдельно перейдём по существующим рёбрам в графе ДКА D и в графе G.

Для удобства будем отмечать строки матрицы M, которые кодируют разные состояния ДКА, разными цветами. Для каждой рассматриваемой операции $M \times \mathcal{D}^a, a \in L$ выберем разную раскраску строк.

$$b: \; M \times \mathcal{B}^b = \begin{array}{|ccc|cccc|} \hline 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} \times \left(\begin{array}{ccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array} \right) = \begin{array}{|ccc|cccc|} \hline 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

$$c:\ M \times \mathcal{B}^c = \left[\begin{array}{c|c} 1\,0\,0 & 1\,0\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \end{array}\right] \times \left(\begin{array}{ccc|cccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right) = \left[\begin{array}{c|c} 0\,1\,0 & 0\,1\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \end{array}\right]$$

Задействованы следующие рёбра и получаются отмеченные вершины в графе и автомате:



Сложим новые матрицы специальным образом. Нужно, чтобы множества вершин, соответствующие одинаковым состояниям автомата, совместились, а левая часть матрицы обхода M стала диагональной.

Чтобы это сделать, нужно трансформировать строчки так, чтобы сложить только те строки в правой части матрицы M, у которых в левой части единицы стоят на одинаковых позициях. После чего переставлять строчки в M так, чтобы её левая часть приняла диагональный вид.

В примере матрица обхода M для следующего шага выглядит так:

$$M = \left[\begin{array}{c|c} 1\,0\,0 & 0\,0\,0\,1 \\ \hline 0\,1\,0 & 0\,1\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \end{array}\right]$$

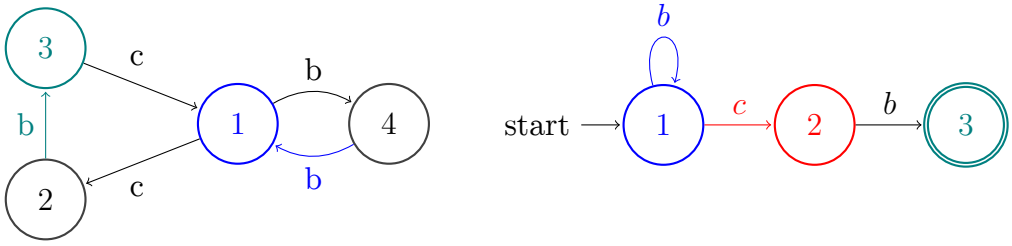
Видно, что во множества вершин графа, попали вершины 2 и 4. В вершину 2 мы попали в состоянии 2, в вершину 4 — в состоянии 1.

Сделаем еще один шаг алгоритма и придем к конечному состоянию в одном из фронтов, соответствующих обходу детерминированного конечного автомата:

$$b:\ M \times \mathcal{B}^b = \begin{array}{|c|c|}\hline 1\,0\,0 & 0\,0\,0\,1 \\ \hline 0\,1\,0 & 0\,1\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \\ \hline \end{array} \times \left(\begin{array}{ccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right) = \begin{array}{|c|c|}\hline 1\,0\,0 & 1\,0\,0\,0 \\ \hline 0\,0\,1 & 0\,0\,1\,0 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \\ \hline \end{array}$$

$$c:\ M \times \mathcal{B}^c = \begin{array}{|c|c|}\hline 1\,0\,0 & 0\,0\,0\,1 \\ \hline 0\,1\,0 & 0\,1\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \\ \hline \end{array} \times \left(\begin{array}{ccc|cccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right) = \begin{array}{|c|c|}\hline 0\,1\,0 & 0\,0\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \\ \hline \end{array}$$

В графах произойдут переходы.



Матрица M примет новый вид.

$$M = \begin{array}{|c|c|}\hline 1\,0\,0 & 1\,0\,0\,0 \\ \hline 0\,1\,0 & 0\,0\,0\,0 \\ \hline 0\,0\,1 & 0\,0\,1\,0 \\ \hline \end{array}$$

Таким образом, вершина два оказалась в строке, соответствующей финальному состоянию 3, она и должна попасть в ответ.

На каждом шаге алгоритма будем суммировать с перестановкой умножения M на матрицы $\mathcal{G}^a$ для всех $a \in L$. Результат суммы присваивать в M. Так будем делать пока матрица M меняется или глубина обхода меньше $|V| \times |Q|$. Тогда в ответе окажутся все достижимые вершины, а алгоритм точно завершит работу.

Данный алгоритм достижимости с регулярными ограничениями на пути описыва-

ется псевдокодом 1. Поскольку получение ДКА из регулярного выражения — отдельная задача, выходящая за рамки работы, будем считать, что на вход алгоритма сразу принимается детерминированный конечный автомат. Для этого, например, можно воспользоваться упомянутым алгоритмом преобразования регулярного выражения в ДКА [16].

Algorithm 1 Алгоритм достижимости в графе с регулярными ограничениями на основе поиска в ширину, выраженный с помощью операций матричного умножения

procedure BFSBasedRPQ($D = \langle Q, q_S, Q_F, \delta, \Sigma\rangle, G = \langle V, E, L\rangle, V_{src}$)
  $\mathcal{D}^a \leftarrow$ Матрица смежности конечного автомата для метки $a \in L$
  $\mathcal{G}^a \leftarrow$ Матрица смежности графа для метки $a \in L$
  $\mathcal{B}^a \leftarrow \begin{pmatrix} \mathcal{D}^a & . \\ . & \mathcal{G}^a \end{pmatrix}$ ▷ Построение матриц $\mathcal{B}^l$
  $M \leftarrow$ Пустая матрица $|Q| \times (|Q| + |V|)$ ▷ Матрица фронта обхода
  $M' \leftarrow M$ ▷ Следующая матрица фронта обхода
  $M'_{q_S q_S} \leftarrow 1$
  $M'_{q_S(v+|Q|)} \leftarrow 1$ для каждого $v \in V_{src}$ ▷ Установим начальные вершины
  while Матрица M меняется и глубина обхода меньше $|Q| \times |V|$ **do**
    $M \leftarrow M'$
    for all $a \in (\Sigma \cap L)$ **do**
      $M'' \leftarrow M \times \mathcal{B}^a$ ▷ Матр. умножение в кольце операций ИЛИ, И
      $M' \leftarrow M'+$ диагонально переставить строки M''
    end for
    $\mathcal{P} \leftarrow$ Извлечь строки, соответствующие конечному состоянию из правой подматрицы M' размером $|Q| \times |V|$
  end while
  return $\mathcal{P}$
end procedure

Теперь докажем, что алгоритм действительно решает задачу. Докажем лемму о шаге алгоритма.

Лемма 2.10. После каждого шага алгоритма с номером k алгоритма в строке новой матрицы обхода M''_q находятся все вершины, между которыми и стартовыми вершинами из V_{src} в графе $G = \langle V, E, L\rangle$ есть путь длины k, соответствующий некоторому пути в детерминированном конечном автомате $D = \langle Q, q_S, Q_F, \delta, \Sigma\rangle$ от q_S до q. То есть:

$$\begin{cases} M''_{qq} = 1 \\ M''_{q(|Q|+v)} = 1 \end{cases} \iff \begin{cases} \exists \text{путь в графе ДКА } D\ \delta_1, \delta_2, ..., \delta_k : q_S = Start(\delta_1), q = End(\delta_k) \\ \exists \text{путь в графе } G\ e_1, e_2, ..., e_k : \\ Start(e_1) \in V_{src}, v = End(e_k), Label(e_i) = Label(\delta_i). \end{cases} \tag{1}$$

База индукции на 0 шаге тривиальна, поскольку изначально в M_{q_S} находятся стартовые вершины V_{src}.

Докажем переход. Выполним k-тый шаг алгоритма:

$$M'' = \sum_{a \in L \cap \Sigma} T(M \times \mathcal{B}^a),$$

где T — функция перестановки.

Зафиксируем некоторый $a \in L \cap \Sigma$. По индуктивному предположению для k выполняются равенства 1. Обозначим $M''^a = M \times \mathcal{B}^a$. По свойству 2.4 и блочной-диагональности $\mathcal{B}^a$ в правой части матрицы:

$$M''^a_{q(|Q|+v)} = 1 \iff \exists e \in E : M_{qStart(e)} = 1, End(e) = v, Label(e) = a.$$

А в левой, соответствующей состоянием ДКА:

$$M''^a_{qq'} = 1 \iff M_{qq} = 1,\ \delta(q, a) = q'.$$

Применим процедуру перестановки T и просуммируем M''^a для всех $a \in L \cap \Sigma$. Очевидно, что для результирующей M'' будут выполняться равенства 1 для $k + 1$. Что и требовалось доказать.

Докажем, что все нужные вершины попадут в ответ.

Теорема 2.11. Результатом работы алгоритма на входных данных
$D = \langle Q, q_S, Q_F, \delta, \Sigma \rangle, G = \langle V, E, L \rangle, V_{src}$ будет множество всех достижимых вершин.

Для доказательства воспользуемся леммой 2.10. На каждом шаге алгоритм помещает в ответ вершины, соответствующие некоторому конечному состоянию. Очевидно, что между каждой вершиной в ответе и некоторой из стартовой будет путь удовлетворяющий регулярным ограничениям.

Докажем в обратную сторону. Допустим, что существует вершина $v \in V$ и путь $e_1, e_2, ..., e_k, Start(e_1) \in V_{src}, v = End(e_k)$, удовлетворяющий регулярным ограничениям, который не попал в ответ. По лемме 2.10 вершина на шаге k окажется в матрице обхода. При этом она будет соответствовать конечному состоянию ДКА, так как путь удовлетворяет ограничениям. Получаем противоречие.

Замечание 2.12. Алгоритм применим для поиска всех пар вершин, между которыми есть хотя бы один путь, удовлетворяющий регулярным ограничениям. Чтобы найти найти такие пары, достаточно отметить все вершины стартовыми, то есть положить $V_{src} = V$.

2.2. Реализация алгоритма

Псевдокод изначальной реализации алгоритма представлен на листинге 2. Самой очевидной проблемой является то, что нигде не используется множество финальных состояний, что значит, что ответ на задачу не может быть верным.

Algorithm 2 Существующая реализация алгоритма достижимости в графе с регулярными ограничениями

procedure BFSBASEDRPQ($D = \langle Q, q_S, Q_F, \delta, \Sigma \rangle, G = \langle V, E, L \rangle, V_{src}$)
  $\mathcal{D}^a \leftarrow$ Матрица смежности ДКА для метки a
  $\mathcal{G}^a \leftarrow$ Матрица смежности графа для метки a
  $\mathcal{B}^a \leftarrow \begin{pmatrix} \mathcal{D}^a & . \\ . & \mathcal{G}^a \end{pmatrix}$
  $M \leftarrow$ Пустая матрица $|Q| \times |V|$
  $M' \leftarrow M$ ▷ Следующая матрица фронта обхода
  $M'_{q_S q_S} \leftarrow 1$
  $M'_{q_S(v+|Q|)} \leftarrow 1$ для каждого $v \in V_{src}$ ▷ Установим начальные вершины
  while Матрица M меняется и глубина обхода меньше $|Q| \times |V|$ **do**
    $M \leftarrow M'$
    for all $a \in (\Sigma \cap L)$ **do**
      $M'' \leftarrow M \times \mathcal{B}^a$ ▷ Матр. умножение в кольце операций ИЛИ, И
      $M' \leftarrow M'+$ Перестановка особым образом строк M''
    end for
    $V \leftarrow$ Извлечь и сложить строки, из правой части матрицы M'
    for $k \in 0 \dots |V_{src}| - 1$ **do**
      $W \leftarrow \mathcal{P}.getRow(k)$
      $\mathcal{P}.setRow(k, V + W)$
    end for
  end while
  return $\mathcal{P}$
end procedure

Алгоритм был реализован при помощи следующих инструментов и библиотек.

- Язык программирования — Python.
- SuiteSparse:GraphBLAS[3] — библиотека, реализующая стандарт API матричных операций GraphBLAS[4], оптимизированная для быстрого выполнения действий над разреженными матрицами в различных форматах [3, 10].
- pygraphblas[5] — обёртка библиотеки SuiteSparse:GraphBLAS для Python.
- pyformlang[6] — Python библиотека для работы с формальными грамматиками, использовавшаяся для преобразования регулярного выражения в конечный автомат [15].
- pytest[7] — библиотека для unit-тестирования Python кода.

[3]Библиотека SuiteSparse:GraphBLAS: https://github.com/DrTimothyAldenDavis/GraphBLAS (Дата доступа: 28.02.24)
[4]Стандарт API GraphBLAS: https://graphblas.org (Дата доступа: 28.02.24)
[5]Библиотека pygraphblas: https://github.com/Graphegon/pygraphblas (Дата доступа: 28.02.24)
[6]Библиотека pyformlang: https://github.com/Aunsiels/pyformlang (Дата доступа: 29.02.24)
[7]Библиотека pytest: https://pytest.org (Дата доступа: 29.02.24)

Несмотря на невысокую скорость интерпретации программ на Python, вследствие того, что всю содержательную работу выполняют библиотеки на C, медлительностью работы скриптового языка можно пренебречь.

3. Реализация

Теперь рассмотрим проделанные исправления, модификации и оптимизации существующей реализации алгоритма достижимости с регулярными ограничениями.

3.1. Исправление существующей реализации алгоритма

Первым делом поймём, как модифицировать алгоритм так, чтобы в ответе оказывались только те вершины графа $G = \langle V, E, L \rangle$, в которых мы оказываемся в финальном состоянии автомата $D = \langle Q, q_S, Q_F, \delta, \Sigma \rangle$.

Рассмотрим матрицу фронта после очередного шага обхода и перестановки строк:

$$M = \begin{array}{|c|c|}\hline 1\,0\,0 & 1\,0\,0\,0 \\ \hline 0\,1\,0 & 0\,0\,0\,0 \\ \hline 0\,0\,1 & 0\,0\,1\,0 \\ \hline \end{array}$$

Конечным состоянием является состояние, соответствующее третьей строке матрицы.

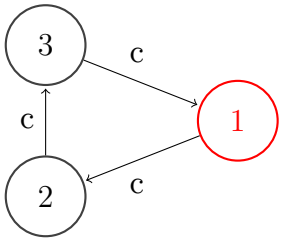
Рассмотрим вектор $\mathcal{F}$ такой, что $\mathcal{F}_i = \begin{cases} 1 & i \in Q_F \\ 0 & \text{иначе} \end{cases}$.

Рассмотрим $\mathcal{F}^T \times M$:

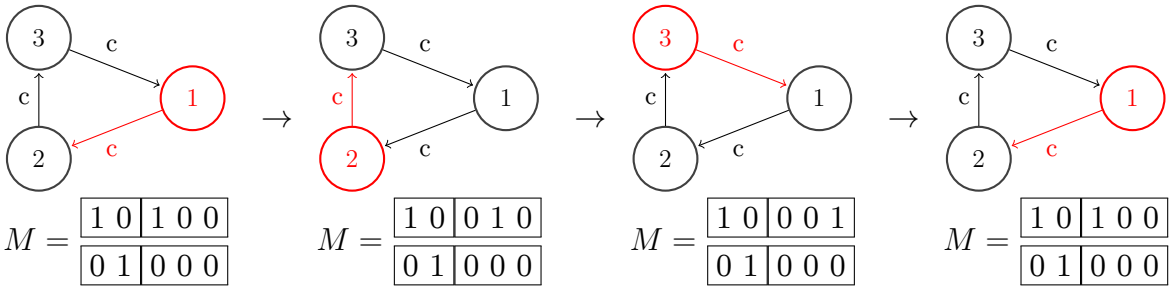
$$\mathcal{F}^T \times M = \boxed{0\,0\,1} \times \begin{array}{|c|c|}\hline 1\,0\,0 & 1\,0\,0\,0 \\ \hline 0\,1\,0 & 0\,0\,0\,0 \\ \hline 0\,0\,1 & 0\,0\,1\,0 \\ \hline \end{array} = \begin{array}{|c|c|}\hline 0\,0\,1 & 0\,0\,1\,0 \\ \hline \end{array}$$

Такое умножение даст нужные в ответе вершины, поскольку по определению матричного произведения это действие по замечанию аналогично извлечению строк, соответствующих конечным состоянием, и последующее их сложение.

Второй проблемой в изначальном алгоритме является повторение одних и тех же действий при работе на графах с циклами. Алгоритм работает пока «матрица обхода меняется» и «пока глубина обхода меньше $|Q| \times |V|$». Если оставить только первое условие и рассмотреть регулярное ограничение, выражаемое c^*b, взять граф, изображённый ниже, и стартовую вершину 1, проблема становится очевидна.



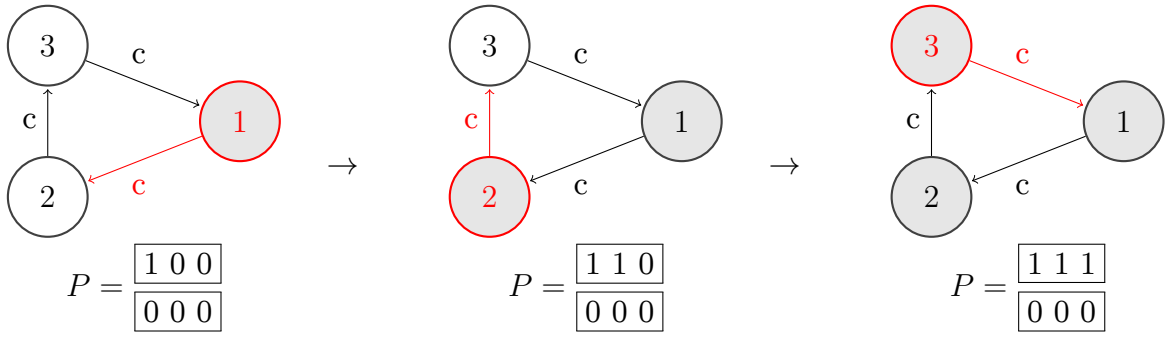
При этом так будет меняться фронт обхода:



Легко видеть, что на каждом шаге обхода, будет происходить переход по ребру с меткой c. Каждый раз матрица обхода M будет меняться, и условие будет выполнено. Это зацикливание можно исправить ограничением на максимальную длину пути. Но на большом графе время достижения верхней границы может быть очень большим.

Исправить это можно следующим образом: будем аккумулировать булеву матрицу $\mathcal{P}$ размером $|Q| \times |V|$. $\mathcal{P}_{qv}$ будет хранить информацию, были ли мы в вершине v в состоянии q. При обходе разрешим переход в состояние (q, v) только если $\mathcal{P}_{qv} = 0$ и после присвоим $\mathcal{P}_{qv} = 1$. Таким образом пропадёт часть лишних вычислений одних и тех же переходов, а алгоритм будет работать до тех пор, пока матрица обхода M не станет пустой. Условие цикла при этом примет вид: «пока матрица обход ненулевая».

Рассмотрим работу алгоритма с данной модификацией на графе с циклом. Вершины, посещённые в состоянии ДКА 0 будем отмечать серым фоном. Алгоритм будет работать следующим образом:



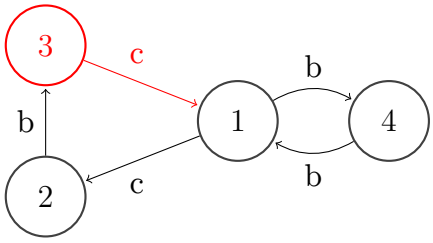
Таким образом зацикливание пропадёт. Алгоритм завершит работу до достижения границы $|V| \times |Q|$.

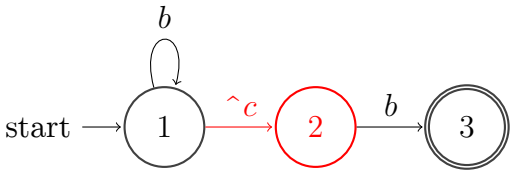
3.2. Поддержка двусторонних запросов

Теперь рассмотрим, как при помощи данного алгоритма выполнять запросы с регулярными ограничениями. Расширим определение конечного автомата, добавив в алфавит символы $\hat{} \cdot L$. Будем иметь в виду под переходом $\hat{}b$ переход по ребру типа b в обратную сторону.

По утверждению 2.5 легко понять, что для того чтобы поддержать такой тип запросов в алгоритме необходимо умножать матрицу обхода на матрицу $\mathcal{B}'^a = \begin{pmatrix} \mathcal{D}^a & . \\ . & (\mathcal{G}^a)^T \end{pmatrix}$, поскольку такая матрица соответствует графу с обращенными в противоположную сторону рёбрами.

Так, например, будет выглядеть первый шаг для меток типа `a` при работе алгоритма на регулярном выражении $b^*\hat{}cb$





$$c: \; M \times \mathcal{B}'^c = \begin{array}{|ccc|cccc|}\hline 1&0&0&1&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ \hline \end{array} \times \left(\begin{array}{ccc|cccc} 0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ \hline 0&0&0&0&0&1&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0 \end{array}\right) = \begin{array}{|ccc|cccc|}\hline 0&1&0&0&0&1&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ \hline \end{array}$$

3.3. Уменьшение среднего времени работы алгоритма

Далее рассмотрим процедуру построчной перестановки строк. Можно обратить внимание, что на каждом шаге алгоритма после умножения матрицы на $\mathcal{B}^a$ левая подматрица из матрицы обхода умножается на одну и ту же матрицу $\mathcal{D}^a$. Так это выглядит на данных из примера:

$$\text{Шаг 1}: \begin{array}{|ccc|cccc|}\hline 1&0&0&1&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ \hline \end{array} \xrightarrow{\times\mathcal{B}^b} \begin{array}{|ccc|cccc|}\hline 1&0&0&0&0&0&1\\ 0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0\\ \hline \end{array}$$

$$\text{Шаг 2}: \begin{array}{|ccc|cccc|}\hline 1&0&0&0&0&0&1\\ 0&1&0&0&1&0&0\\ 0&0&0&0&0&0&0\\ \hline \end{array} \xrightarrow{\times\mathcal{B}^b} \begin{array}{|ccc|cccc|}\hline 1&0&0&1&0&0&0\\ 0&0&1&0&0&1&0\\ 0&0&0&0&0&0&0\\ \hline \end{array}$$

Обозначим результат умножения на некотором шаге алгоритма $M''^a = M \times \mathcal{B}^a$. Тогда:

$$M''^a_{ij} = 1 \Rightarrow \mathcal{D}^a_{ij} = 1 \; \forall i, j = 1..|Q|$$

Это происходит из-за того, что левая часть матрицы обхода всегда диагональная по определению процедуры перестановки. Каждый раз столбцы, соответствующие состояниям ДКА, умножаются одним и тем же образом для каждого $a \in L$.

Наблюдение 3.1. В каждой строчке матрицы $\mathcal{D}^a$ не более одной 1, это следует из определения детерминированного конечного автомата.

Поймём, как можно выразить процедуру нужной перестановки и сложения строк при помощи операций линейной алгебры. А так же избавимся от лишних произведений левой части матрицы обхода на одну и ту же матрицу смежности ДКА.

Утверждение 3.1. Перестановку строк нужным образом для каждого символа $a \in L$ можно осуществить, если умножить $(\mathcal{D}^a)^T$ на результат $M \times \mathcal{B}^a$.

Рассмотрим произведение $M'' = (\mathcal{D}^a)^T \times M'$. Зафиксируем номер строки i. По определению матричного произведения строка матрицы

$$M''_i = \sum_{i=1..k} M'_{j_i}, \tag{2}$$

где $j_1, j_2, ..., j_k$ — номера ненулевых столбцов в строке $(\mathcal{D}^l)^T_i$.

Поймём, что

$$(\mathcal{D}^a)^T_{ij} = 1 \iff (\mathcal{D}^a)_{ji} = 1,$$

что соответствует переходу в конечном автомате D по ребру с меткой a из состояния j в состояние i.

То есть сумма 2 складывает все строки, переходящие после умножения $M \times \mathcal{B}^a$ в состояние i. Из наблюдения 3.1 следует, что

$$M''_{ii} = 1 \iff \exists m = 1..k, M'_{j_m i} = 1$$
$$\forall p = 1..|Q|, p \neq i, M''_{ip} = 0, \text{ так как } \forall m = 1..k, M'_{j_m p} = 0$$

Что и нужно по требованию процедуры перестановки.

Примеры таких перестановок для графа и автомата из примера и метки b:

$$\text{Шаг 1 : } \left[\begin{array}{c|c|c} 1\,0\,0 & & \\ \hline 0\,0\,0 & & \\ \hline 0\,1\,0 & & \end{array}\right] \times \left[\begin{array}{c|c} 1\,0\,0 & 0\,0\,0\,1 \\ \hline 0\,0\,1 & 0\,0\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \end{array}\right] = \left[\begin{array}{c|c} 1\,0\,0 & 0\,0\,0\,1 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \\ \hline 0\,0\,1 & 0\,0\,0\,0 \end{array}\right]$$

$$\text{Шаг 2}: \begin{array}{|c|}\hline 1\,0\,0 \\ \hline 0\,0\,0 \\ \hline 0\,1\,0 \\ \hline \end{array} \times \begin{array}{|c|c|}\hline 1\,0\,0 & 1\,0\,0\,0 \\ \hline 0\,0\,1 & 0\,0\,1\,0 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \\ \hline \end{array} = \begin{array}{|c|c|}\hline 1\,0\,0 & 1\,0\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,0\,0 \\ \hline 0\,0\,1 & 0\,0\,1\,0 \\ \hline \end{array}$$

Псевдокод итоговой реализации алгоритма представлен в алгоритме 3.

Algorithm 3 Итоговая версия алгоритма достижимости в графе с регулярными ограничениями на основе поиска в ширину, выраженного с помощью операций матричного умножения

procedure BFSBasedRPQ($D = \langle Q, q_S, Q_F, \delta, \Sigma \rangle, G = \langle V, E, L \rangle, V_{src}$)
  $\mathcal{D}^a \leftarrow$ Матрица смежности ДКА для символа a
  $\mathcal{G}^a \leftarrow$ Матрица смежности графа для символа a
  $\mathcal{F} \leftarrow$ Вектор финальных состояний автомата D
  $M \leftarrow$ Матрица $|Q| \times |V|$ ▷ Матрица обхода
  $P \leftarrow$ Матрица $|Q| \times |V|$ ▷ Матрица с посещёнными состояниями
  $M' \leftarrow$ Матрица $|Q| \times |V|$ с вершинами из V_{src}, соответствующими q_S
  while M' ненулевая **do**
    $M \leftarrow M'$
    $M' \leftarrow$ Пустая матрица
    for all $a \in (\Sigma \cap L)$ **do**
      if a начинается не с ^ **then**
        $M' \leftarrow M' + (\mathcal{D}^a)^T \times (M \times \langle \neg P \rangle \mathcal{G}^a)$ ▷ Умножение с маской
      else
        $M' \leftarrow M' + (\mathcal{D}^a)^T \times (M \times \langle \neg P \rangle (\mathcal{G}^a)^T)$
      end if
    end for
    $P \leftarrow P + M'$ ▷ Аккумулирование достигнутых состояний
    $\mathcal{P} \leftarrow \mathcal{P} + \mathcal{F} \times M'$ ▷ Поиск финальных состояний
  end while
  return $\mathcal{P}$
end procedure

Иных оптимизаций найти не удалось. Предполагалось, что изменение формата матриц может ускорить работу алгоритма, но экспериментально было выявлено, что изначальный формат представления разреженных матриц является самым оптимальным для данной задачи.

4. Сравнение времени работы алгоритма со временем выполнения запроса MillenniumDB

Для анализа производительности алгоритма было произведено сравнение времени работы с изначальной версией алгоритма и с СУБД MillenniumDB [11] (сокращённо — MDB) с открытым исходным кодом. MillenniumDB не является промышленным решением, поскольку она нацелена на исследование различных алгоритмов, но при этом она демонстрирует лучшее время на некоторых бенчмарках, в сравнении с популярными графовыми СУБД Blazegraph, Virtuoso, Neo4J, Jena [11].

В качестве данных была выбрана база знаний Wikidata[8] [18], как один из самых разнообразных и больших графов доступных публично. Вместе с ним доступны наборы реальных регулярных запросов, которые использовались для измерения времени работы алгоритма.

Стоит отметить, что MillenniumDB — полноценная СУБД. Загрузка ею всей базы данных Wikidata занимает около 2 часов, поскольку при этом строятся индексы. При сравнении алгоритма, работающим с графом в оперативной памяти, и СУБД нас интересует не будем ли слишком сильно проигрывать по времени выполнению запросов с индексами. Если проигрыш будет составлять всего несколько раз, а не целые порядки, можно считать, что алгоритм достаточно эффективен.

4.1. Условия эксперимента

Эксперименты проводились на тестовом стенде со следующей конфигурацией:

- **CPU**: AMD Ryzen 9 7900X, 12 физических, 24 логических ядра, max 5.733 GHz
- **RAM**: 128 Gb, 3600 MHz, DDR5
- **OS**: Ubuntu 22.04

Тестирование проводилось на 21 запросе. Запросы выбирались так, чтобы результаты и регулярные ограничения были разнообразны. В исходном наборе много запросов, содержащих метку только одного типа и звезду Клини, поэтому они не настолько интересны. В большинстве запросов длина ответа — не более 50, поэтому один из запросов выбран на основании большого числа вершин в ответе. Часть запросов взята из-за того, что они двусторонние.

Измерялось только время работы алгоритма или исполнения запроса. Загрузка графа в оперативную память не учитывалась. При этом использовалась утилита

[8]База знаний WikiData: https://www.wikidata.org/wiki/Wikidata:Database_download (Дата доступа 05.04.24)

pytest-benchmark[9]. Этот инструмент используется для анализа производительности Python кода, он автоматически регулирует размер выборки, вычисляет стандартное отклонение и другие статистические показатели.

Библиотека SuiteSparse:GraphBLAS при этом была собрана из исходного кода с флагами оптимизации `-O3`, `-march=native` и поддержкой OpenMP, профилирование Python скрипта показало, что время интерпретации скрипта пренебрежимо мало в сравнении со временем исполнения C библиотек.

Измерения времени исполнения запроса MillenniumDB проводились на основании внутреннего таймера СУБД, временем запроса считалась сумма времени работы оптимизатора и исполнения запроса. При этом обработка регулярного выражения и загрузка графов в оперативную память не учитывалась ни в алгоритме, ни в MillenniumDB. Выборка была размером 20 для каждого запроса. При этом стандартное отклонение получилось большим из-за специфики работы СУБД, заключающейся в оптимизации популярных запросов.

[9]Утилита pytest-benchmark: https://github.com/ionelmc/pytest-benchmark (Дата доступа: 07.03.24)

4.2. Результаты измерений

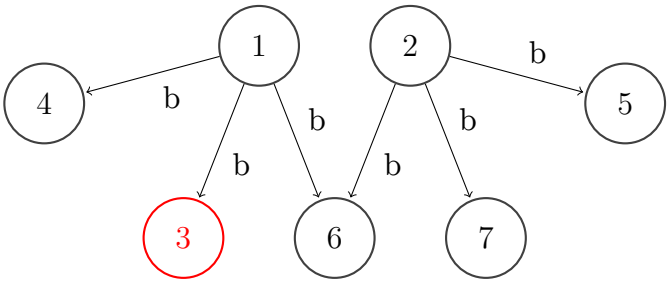
	RPQ		OLD			MDB		
№	Среднее	σ	Среднее	σ	Ускорение	Среднее	σ	Замедление
1	266	±12	2136	±6	8.0	97	±12	2.7
2	263	±3	2166	±6	8.2	120	±20	2.2
3	693	±4	2169	±6	3.1	80	±20	8.7
4	203	±2	2183	±8	10.8	80	±20	2.5
5	521	±3	2196	±12	3.7	170	±20	3.1
6	266	±2	2207	±11	8.3	70	±15	3.8
7	264	±3	—			66	±13	4.0
8	283	±2	—			25	±3	11.3
9	197	±2	—			45	±10	4.4
10	516,600	±1100	—			176,000	±10000	2.9
11	282	±4	—			55	±5	5.1
12	282	±2	—			36	±12	7.8
13	284	±3	—			31	±8	9.2
14	326	±3	—			50	±5	6.5
15	238	±2	—			31	±8	7.7
16	237	±2	—			29	±8	8.2
17	754	±4	—			200	±40	3.8
18	237	±2	—			30	±5	7.9
19	32,500	±600	—			52,300	±1000	0.6
20	1,727,000	±11000	—			175,000	±10000	9.9
21	1,780,000	±30000	—			216,000	±11000	8.2

OLD — изначальный алгоритм, RPQ — обновлённый алгоритм, MDB — СУБД MillenniumDB. Время указано в миллисекундах. Указано среднее время исполнения запроса и стандартное отклонение. Замедление и ускорение рассчитывались как отношение средних. Для алгоритма OLD указаны не все запросы, поскольку время некоторых замеров было слишком велико, а некоторые завершались аварийно из-за нехватки оперативной памяти.

Видно, что по сравнению с изначальной реализацией время работы сильно уменьшилось, при этом по сравнению с MillenniumDB проигрыш на большинстве запросов достаточно большой. Исследуемый алгоритм примерно в $3 - 10$ раз медленнее, чем СУБД MillenniumDB.

Одним из интересных результатов является заметно меньшее время алгоритма на одном из запросов. По сравнению с остальными, он интересен тем, что ищет пути вида $(\hat{}bb)+$. Такой запрос на каждом шаге ищет все «родственные» найденным вершинам по отношению b. На таком графе результатом такого запроса со стартовой вершиной

3 будут вершины $3, 4, 5, 6, 7$.



Причины, по которым такой запрос выполняется быстрее, выяснить не удалось в связи с тем, что других запросов такого типа нет. Запросы $20 - 21$ так же были двусторонними, но на них алгоритм проигрывает по времени исполнения.

Профилирование алгоритма показало, что почти всё время работы составляет перемножение матриц $M \times \mathcal{G}^a$, что соответствует содержательному поиску удовлетворяющих регулярным ограничениям путей, что является косвенным свидетельством того, что алгоритм не делает производит лишних вычислений.

Вместе с измерением производительности сравнивалось число найденных СУБД и алгоритмом пар вершин, между которыми существовал путь удовлетворяющий заданным ограничениям. Во всех случаях ответы совпадали, что так же является косвенным подтверждением корректности реализации алгоритма.

Заключение

В рамках работы над алгоритмом достижимости с регулярными ограничениями на пути при помощи линейной алгебры были достигнуты следующие результаты.

- Исправлены ошибки в реализации алгоритма поиска достижимых вершин с регулярными ограничениями, из-за которых результат работы алгоритма был некорректен.
- Добавлена поддержка двусторонних регулярных запросов.
- Улучшена производительность алгоритма за счёт выражения всех действий в более естественных и производительных матричных операциях и изменения условия окончания работы алгоритма.
- Экспериментальное сравнение показало, что алгоритм значительно быстрее прошлой реализации, медленнее полноценной СУБД, использующей кэши и индексы, в несколько раз.

Исходный код решения доступен в репозитории https://github.com/georgiy-belyanin/CFPQ_PyAlgo.

Дальнейшие направления исследований

В дальнейшем работа над алгоритмом достижимости с регулярными ограничениями на пути может быть продолжена в следующих направлениях.

- Поскольку MilleniumDB не является промышленным решением, интересно сравнение алгоритма с другими графовыми СУБД, в частности с RedisGraph, поскольку она может использовать схожее хранение всей базы данных в оперативной памяти.
- Графовые СУБД поддерживают различные семантики запросов. В некоторых случаях интересен конкретный путь, по которому вершина достижима. При этом нужны могут быть все удовлетворяющие регулярным ограничениям пути, только кратчайшие или любые. Так же отдельно представляют интерес случаи с одним источником, несколькими и поиск всех пар вершин. Можно попробовать применить модификации описанного алгоритма для выполнения этих задач.
- Интересны причины выигрыша в случае выполнения запроса на поиск всех вершин «родственников» «родственникам». Возможно, что существуют другие типы запросов, не попавшие в данные Wikidata, на которых алгоритм работает быстрее аналогов.

- В исследовании нуждаются возможности применения свойств операций линейной алгебры для дальнейшего уменьшения времени работы алгоритма.

Список литературы

[1] Arroyuelo Diego, Gómez-Brandón Adrián, Navarro Gonzalo. Evaluating Regular Path Queries on Compressed Adjacency Matrices. — 2024. — 2307.14930.

[2] Casel Katrin, Schmid Markus L. Fine-Grained Complexity of Regular Path Queries // Logical Methods in Computer Science. — 2023. — . — Vol. Volume 19, Issue 4. — URL: http://dx.doi.org/10.46298/lmcs-19(4:15)2023.

[3] Davis Timothy A. Algorithm 1037: SuiteSparse:GraphBLAS: Parallel Graph Algorithms in the Language of Sparse Linear Algebra // ACM Trans. Math. Softw. — 2023. — sep. — Vol. 49, no. 3. — 30 p. — URL: https://doi.org/10.1145/3577195.

[4] Farías Benjamín, Rojas Carlos, Vrgoč Domagoj. Evaluating Regular Path Queries in GQL and SQL/PGQ: How Far Can The Classical Algorithms Take Us? — 2023. — 06.

[5] Foundations of Modern Query Languages for Graph Databases / Renzo Angles, Marcelo Arenas, Pablo Barceló et al. // ACM Comput. Surv. — 2017. — sep. — Vol. 50, no. 5. — 40 p. — URL: https://doi.org/10.1145/3104031.

[6] Hagenah Christian, Muscholl Anca. Computing ε-free NFA from regular expressions in O(n log2(N)) time. — 1970. — 01. — Vol. 34. — P. 277–285. — ISBN: 978-3-540-64827-7.

[7] Highly accurate protein structure prediction with AlphaFold / John Jumper, Richard Evans, Alexander Pritzel et al. // Nature. — 2021. — Aug. — Vol. 596, no. 7873. — P. 583–589. — URL: https://doi.org/10.1038/s41586-021-03819-2.

[8] Knowledge Graphs / Aidan Hogan, Eva Blomqvist, Michael Cochez et al. // ACM Comput. Surv. — 2021. — jul. — Vol. 54, no. 4. — 37 p. — URL: https://doi.org/10.1145/3447772.

[9] Koschmieder Andre, Leser Ulf. Regular Path Queries on Large Graphs. — Vol. 7338. — 2012. — 06.

[10] Mathematical foundations of the GraphBLAS / Jeremy Kepner, Peter Aaltonen, David Bader et al. // 2016 IEEE High Performance Extreme Computing Conference (HPEC). — 2016. — P. 1–9.

[11] Vrgoc Domagoj, Rojas Carlos, Angles Renzo et al. MillenniumDB: A Persistent, Open-Source, Graph Database. — 2021. — 2111.01540.

[12] Bienvenu Meghyn, Calvanese Diego, Ortiz Magdalena, Simkus Mantas. Nested Regular Path Queries in Description Logics. — 2014. — 1402.7122.

[13] Farías Benjamín, Martens Wim, Rojas Carlos, Vrgoč Domagoj. PathFinder: A unified approach for handling paths in graph query languages. — 2024. — 2306.02194.

[14] Reasoning on regular path queries / Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Moshe Vardi // ACM SIGMOD Record. — 2003. — 12. — Vol. 32. — P. 83–92.

[15] Romero Julien. Pyformlang: An Educational Library for Formal Language Manipulation // Proceedings of the 52nd ACM Technical Symposium on Computer Science Education. — 2021. — URL: https://api.semanticscholar.org/CorpusID:232126450.

[16] Sanjay Bhargava, Purohit G.N. Construction of a Minimal Deterministic Finite Automaton from a Regular Expression // International Journal of Computer Applications. — 2011. — 02. — Vol. 15.

[17] Urma Raoul-Gabriel, Mycroft Alan. Source-code queries with graph databases—with application to programming language usage and evolution // Science of Computer Programming. — 2015. — Vol. 97. — P. 127–134. — Special Issue on New Ideas and Emerging Results in Understanding Software. URL: https://www.sciencedirect.com/science/article/pii/S0167642313002943.

[18] Vrandečić Denny, Krötzsch Markus. Wikidata: a free collaborative knowledgebase // Commun. ACM. — 2014. — sep. — Vol. 57, no. 10. — P. 78–85. — URL: https://doi.org/10.1145/2629489.

[19] Vrgoč Domagoj. Evaluating regular path queries under the all-shortest paths semantics. — 2023. — 2204.11137.

[20] Wood Peter. Query Languages for Graph Databases // Sigmod Record. — 2012. — 04. — Vol. 41. — P. 50–60.

[21] Мартыненко Б. К. Языки и трансляции. — Издательство Санкт-Петербургского Университета, 2013.