Санкт-Петербургский государственный университет

Кафедра системного программирования

Оптимизация алгоритма решения задачи достижимости с ограничениями в виде регулярных языков

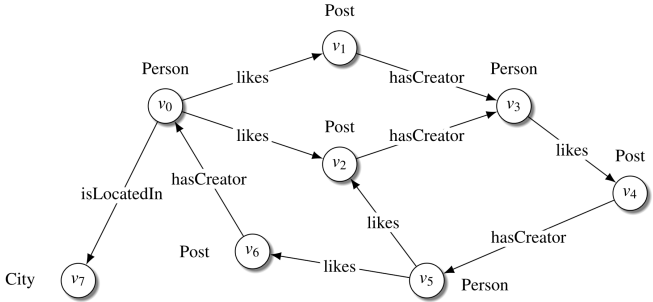
Белянин Георгий Олегович, группа 22.Б07-мм

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

Санкт-Петербург
2024

Запросы с регулярными ограничениями в графовых БД

- Графовые базы используются в биологии, анализе кода, семантической сети
- Регулярные запросы (*RPQ*) реализованы в языке SPARQL, описаны в стандарте ISO/IEC 39075:2024[1]



Поиск потенциально интересных авторов: (*likes hasCreator*)* [2]

[1] https://www.iso.org/standard/76120.html (Дата доступа: 30.05.2024)

[2] Tetzel, F., Lehner, W. & Kasperovics, R. Efficient Compilation of Regular Path Queries. Datenbank Spektrum 20 — 2020. https://doi.org/10.1007/s13222-020-00353-9

Выполнение запросов с регулярными ограничениями

- Существующие алгоритмы выполнения запросов недостаточно эффективны
- Алгоритм «Regular Path Queries on Compressed Adjacency Matrices» с использованием разреженных матриц смежности сильно проигрывает по производительности[3]
- Линейная алгебра — эффективный способ обработки графов[4]
- Алгоритм, разработанный на кафедре системного программирования, не доработан

[3] Arroyuelo D., Gómez-Brandón A., Navarro, G. Evaluating Regular Path Queries on Compressed Adjacency Matrices. — 2024. — 2307.14930 — https://arxiv.org/abs/2307.14930 (Дата доступа: 11.03.2024)

[4] Davis Timothy A. SuiteSparse:GraphBLAS: Parallel Graph Algorithms in the Language of Sparse Linear Algebra // ACM Trans. Math. Softw. — https://doi.org/10.1145/3577195 (Дата доступа: 01.03.2024)

Постановка задачи

Целью работы является создание производительного алгоритма достижимости с регулярными ограничениями.

Задачи:

1. Исправить существующую реализацию алгоритма
2. Поддержать двусторонние регулярные запросы
3. Найти узкие места в данном алгоритме и уменьшить среднее время работы
4. Сравнить время работы алгоритма со временем исполнения запросов графовой СУБД MillenniumDB[5]

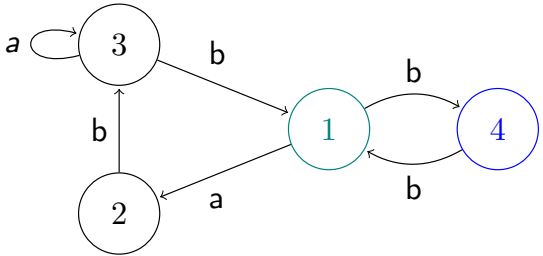
[5] https://github.com/MillenniumDB/MillenniumDB (Дата доступа: 05.03.2024)

Реализация алгоритма

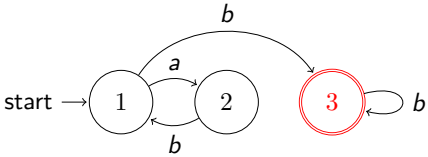
- Python
- SuiteSparse:GraphBLAS — линейная алгебра над разреженными матрицами
- pygraphblas — обёртка SuiteSparse:GraphBLAS для Python
- pyformlang — работа с формальными языками
- pytest — unit-тестирование

Исправления изначального алгоритма

- Добавлен ранний выход из цикла, убран повторный расчёт одних и тех же состояний
- Добавлена фильтрация вершин с использованием маски



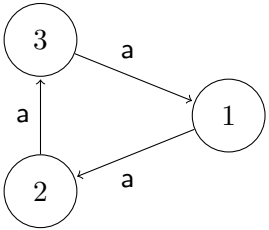
Конечный автомат выражения $(ab)^*bb^*$



1 0 0	0 0 1 0
0 0 0	0 0 0 0
0 0 1	1 0 0 1

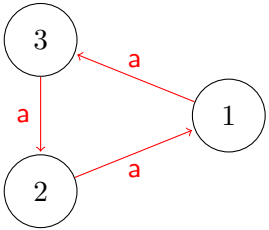
Поддержка двусторонних запросов

- Добавлена поддержка двусторонних запросов с использованием свойств матриц смежности



$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Граф, матрица смежности

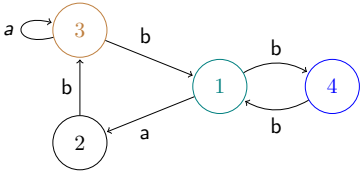


$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Транспонируем матрицу смежности

Ускорение алгоритма

- Ручная перестановка строк и столбцов заменена векторно-матричными операциями



Конечный автомат выражения $(ab)^*bb^*$

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

Перестановка нужным образом реализуется умножением на предподсчитанную матрицу, новая левая часть — диагональна

Сравнение с СУБД MillenniumDB

- База знаний Wikidata[6] — граф размером 270 миллионов вершин, 15 миллиардов рёбер, 200 различных реальных регулярных запросов
- pytest-benchmark[7] для замеров времени работы Python кода
- Время исполнения MillenniumDB измерялось с использованием внутреннего таймера

[6] https://www.wikidata.org/ (Дата доступа 05.04.24)
[7] https://github.com/ionelmc/pytest-benchmark (Дата доступа: 07.03.24)

Результаты сравнения с MillenniumDB

RPQ		OLD			MDB		
среднее	σ	среднее	σ	ускорение	среднее	σ	замедление
266	$\pm$ 12	2136	± 6	8.0	97	$\pm$ 12	2.7
693	$\pm$ 4	2169	± 6	3.1	80	$\pm$ 20	8.7
264	$\pm$ 3	—			66	$\pm$ 13	4.0
516 600	$\pm$ 1 100	—			176 000	$\pm$ 10 000	2.9
284	$\pm$ 3	—			31	$\pm$ 8	9.2
32 500	$\pm$ 600	—			52 300	$\pm$ 1 000	**0.6**
1 727 000	$\pm$ 11 000	—			175 000	$\pm$ 10 000	9.9

Таблица: Время выполнения и стандартное отклонение разных алгоритмов на разных запросах, мс; «—» — измерение не проводилось

- СУБД использует кэши, поэтому проигрыш до 10 раз приемлем
- Один из запросов на поиск сиблингов оказался быстрее, чем в СУБД
- Значительное уменьшение времени работы, по сравнению со старой версией

Результаты

- Исправлены ошибки в реализации алгоритма поиска достижимых вершин с регулярными ограничениями
- 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

Псевдокод итогового алгоритма

Algorithm 1 RPQ

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 $\hat{}$ **then**
        $M' \leftarrow M' + (\mathcal{D}^a)^T \times (M \times \langle \neg P \rangle \mathcal{G}^a)$ ▷ Умножение с маской
      else
        $M' \leftarrow M' + (\mathcal{D}^a)^T \times (M \times \langle \neg P \rangle (\mathcal{G}^a)^T)$
    $P \leftarrow P + M'$ ▷ Аккумулирование достигнутых состояний
    $\mathcal{P} \leftarrow \mathcal{P} + \mathcal{F} \times M'$ ▷ Поиск финальных состояний
  return $\mathcal{P}$