Задача поиска путей с контекстно-свободными ограничениями

Руководитель: Григорьев Семён Вячеславович
Студент: Шеметова Екатерина

Содержание задачи

Дан граф, каждое ребро помечено каким-либо словом или буквой (отношением)

Дана контекстно-свободная грамматика

Найти пути с контекстно-свободными ограничениями значит найти все пути в графе, такие, что слово, составленное из меток рёбер, входящих в этот путь, выводится из нетерминалов данной грамматики

Пример

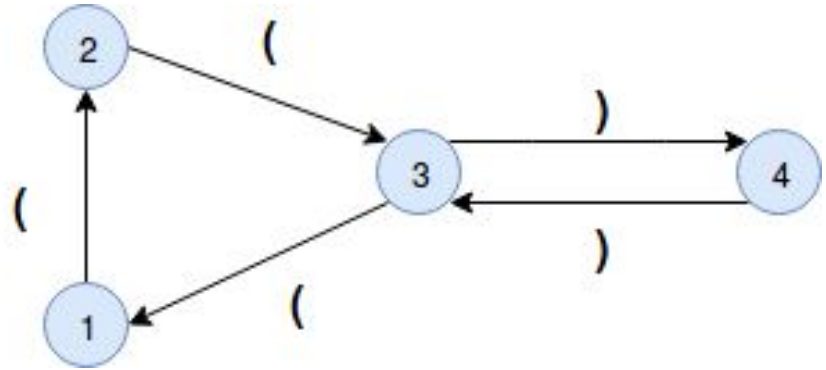
Дана грамматика языка правильных скобочных последовательностей и граф, метки рёбер которого - скобки.

Нужно найти пути в графе, соответствующие правильным скобочным последовательностям.

Ответ:

$2 \rightarrow 4$ - ( ) , $1 \rightarrow 3$ - (( ))

$3 \rightarrow 3$ - (((((( )))))) и т.д.



Применение на практике

- **Статический анализ кода**

  Анализ графа потока управления, межпроцедурный анализ и др.

- **Запросы к графовым базам данных**

  Социальные сети, биоинформатика

  Граф - база данных, грамматика - запрос

Существующие решения

1. **Context-free recognizer for graphs**
   - Rustam Azimov, Semyon Grigorev. 2017. Context-Free Path Querying by Matrix Multiplication
   - Работает на любом виде графов
   - Любая контекстно-свободная грамматика
   - Время работы:

$$O(|V|^2|N|^3 BMM(|V|) + BMU(|V|)$$

*|N| - количество нетерминалов, |V| - число вершин в графе, BMM - время умножения булевых матриц, BMM - время логического объединения булевых матриц

Существующие решения

2. Алгоритм Бредфорда

- Phillip G. Bradford. 2018. Efficient Exact Paths For Dyck and semi-Dyck Labeled Path Reachability.
- Работает на любом виде графов
- Только для языков Дика на одном типе скобок - языков правильных скобочных последовательностей
- Время работы:
  $O(BMM(|V|)polylog|V|)$

*|V| - число вершин в графе, BMM - время умножения булевых матриц

Существующие решения.

3. Параллельные алгоритмы

- Нет готовых алгоритмов
- Задача как минимум такая же сложная, как задача принадлежности строки контекстно-свободной грамматике
- Для определения принадлежности строки параллельных алгоритмов много, есть и для отдельных подклассов грамматик

Существующие решения.

Выводы:

- Существующие решения медленные
- Практически не изучены возможности построить более эффективные алгоритмы для отдельных подклассов графов/грамматик, а для задач подобного рода польза таких алгоритмов на практике существенна
- Параллельные алгоритмы отсутствуют
- Неизвестно, можно ли вообще эффективно распараллелить задачу
- В качестве модели параллельных вычислений удобно рассмотреть модель логических схем, так как для строкового входа логических схем много, что-то можно модифицировать для графового входа

Цель работы

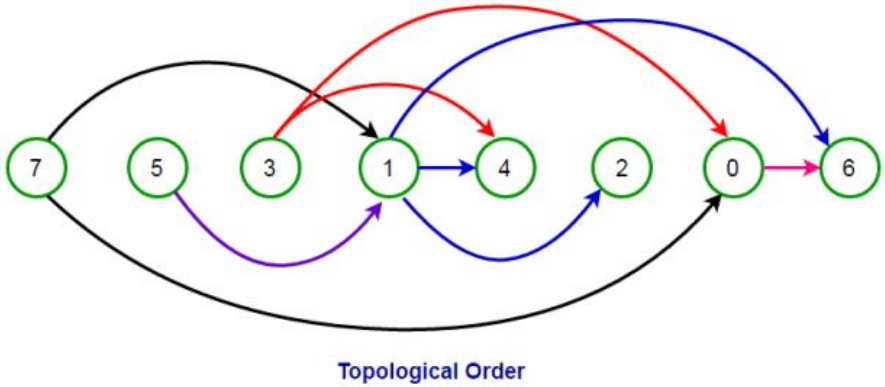
Цель работы: разработать алгоритм, параллельный или последовательный, решающий задачу эффективнее существующих решений либо в общем случае, либо на отдельных подклассах графов/грамматик.

Задачи

1. **Разработка эффективного алгоритма для подкласса графов**
   - Изучить, на каких подклассах графов возможно построить более быстрый алгоритм
   - Построить алгоритм и оценить его время работы
   -
2. **Разработка параллельного алгоритма**
   - Построить логическую схему и оценить её параметры
   - Показать, для каких классов грамматик/видов графов задачу можно эффективно распараллелить.

Эффективный алгоритм для подкласса графов

- Классические эффективные алгоритмы синтаксического анализа являются алгоритмами динамическог программирования
- Для того, чтобы алгоритм работал корректно, должен быть задан порядок на подстроках
- Для графа в качестве входа таких алгоритмов тоже необходим порядок
- **Топологическая сортировка** позволяет задать такой порядок на вершинах графа



Topological Order

Эффективный алгоритм для подкласса графов. Выводы.

- **Ациклические графы (DAG-и) позволяют топологическую сортировку, а значит необходимый порядок**
- Можем модифицировать и применять динамические алгоритмы синтаксического анализа
- Модифицирован алгоритм Охотина (Alexander Okhotin. 2014. Parsing by matrix multiplication generalized to Boolean grammars.) и доказана его корректность
- Время работы: $O(|G|BMM(|V|)log|V|)$ - аналогично времени работы на строках *|G| - размер грамматики, |V| - число вершин в графе, BMM - время умножения булевых матриц

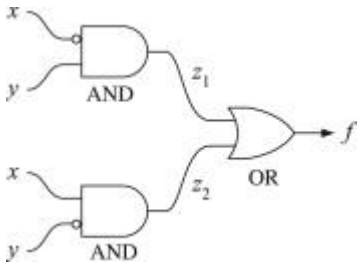
Параллельный алгоритм. Логическая схема

Логическая схема - ациклический граф с помеченными вершинами, выделенными входными и выходными элементами. Промежуточные элементы называют гейтами и они представляют собой булевы функции: дизъюнкцию, конъюнкцию или отрицание.

Важными характеристиками схемы являются её размер (количество элементов) и глубина (самый длинный путь от входа до выхода)

Эффективной считаем схему логарифмической

или полилогарифмической от размера входа глубины.



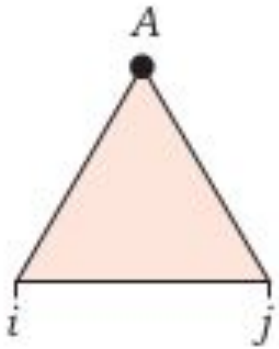
Параллельный алгоритм. Логическая схема

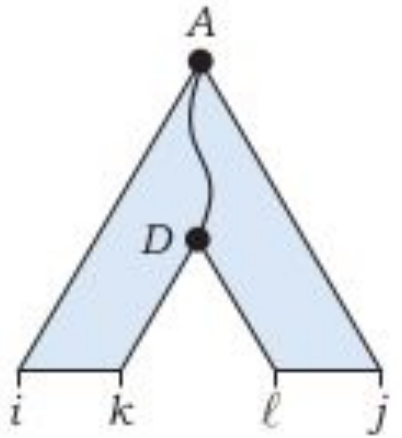
Была построена следующая логическая схема.

- Используем идею из алгоритма Брента-Гольдшлягера-Риттера
- Представим синтаксический анализ как логическую систему

с утверждениями вида **A(i,j)** и условными утверждениями вида “если **D(k,l)**, то **A(i,j)**“.

A(i, j) - дерево разбора с корнем в нетерминале A, i,j - вершины начала и конца пути соответственно





Параллельный алгоритм. Логическая схема

- **Вход схемы:** рёбра графа - существует ли ребро из i в j, помеченное символом a?
- **Выход:** тройки A(i,j)
- **Гейты двух типов:**

  - x(A,i,j), в котором вычисляется значение A(i, j)

  - y(A,i,j,D,k,l) для условных утверждений

Логическая схема. Оценка

Доказаны следующие оценки:

- Количество элементов схемы аналогично строковому входу - $O(n^6)$
- Глубина схемы - $\log^2(t)$
- **t** - означает максимальную длину строки, являющейся ответом на задачу
- Для каждой пары вершин графа найдем минимальные пути, слова из меток которых выводятся нетерминалами грамматики (интересуют все такие нетерминалы)
- Максимальный из таких путей среди всех пар вершин и нетерминалов и есть **t**.

Логическая схема. Оценка

- В худшем случае **t** экспоненциальна от количества вершин в графе (n)

$$O(2^{Nn^2})$$

- Получаем глубину схемы, полиномиальную от входа
- Это говорит о неэффективности распараллеливания в худшем случае
- Но значение **t** отличается для разных подклассов грамматик, и может быть полиномиальным для некоторых

Логическая схема. Выводы

- Была построена логическая схема, решающая задачу
- Оценены глубина схемы и размер
- Глубина схемы зависит от особого параметра - максимальной длины строки, являющейся ответом на задачу
- Этот параметр различается у разных классов грамматик
- В общем случае глубина схемы полиномиальна от входа - распараллеливание неэффективно

Логическая схема. Оценки для подклассов грамматик

- Дерево разбора для максимальной строки имеет высоту не более $O(Nn^2)$
  n - число вершин в графе, N - число нетерминалов
- Для **линейных грамматик** (грамматика, которая имеет не более одного нетерминала в правой части каждого правила) можно построить дерево с такой высотой, количество листьев в нём не превысит $O(Nn^2)$ (из-за формы правил грамматики). Получаем, что **t** полиномиальна от числа вершин в графе.
- Если **граф на входе схемы ациклический**, то несложно заметить, что **t** также полиномиальна, так как максимальная длина пути никогда не превысит число вершин в графе

Логическая схема. Оценки для подклассов грамматик

- **Input-driven грамматики -** грамматики, сбалансированных скобок (пример: XML, JSON, языки Дика (правильные скобочные последовательности)) - **t** порядка $O(n^2)$ если в грамматике один тип скобок, и экспоненциальная, если типов скобок больше, чем один
- **LL-грамматики** (позволяют парсинг сверху вниз, на правила наложены ограничения) - ограничения на правила не влияют на рассматриваемый показатель в общем случае, поэтому **t** экспоненциальная.

Логическая схема. Оценки для подклассов грамматик. Выводы

- Для линейных грамматик, грамматик языка Дика на одном типе скобок, а также для любой контекстно-свободной грамматики и ациклического графа на входе высота схемы $O(\log^2 n)$, значит данные задачи можно решить эффективно с помощью параллельного алгоритма
- Существование более эффективной схемы для подкласса грамматики по сравнению с общем случаем для строкового входа не обязательно значит существование таковой для графового входа

Что сделано

Представлен алгоритм, эффективно решающий задачу на ациклических графах с возможностью в качестве запросов использовать булевы грамматики

Построена логическая схема, решающая задачу

Даны оценки эффективности этой схемы в общем случае

Даны оценки эффективности для отдельных подклассов грамматик и графов

Выводы

Используя топологическую сортировку, возможно построить эффективный последовательный алгоритм для ациклических графов

Для решения задачи возможно построить логическую схему, решающую её

Построенная схема неэффективна в общем случае, но эффективна для ациклических графов и некоторых классов грамматик

Логическая схема эффективна для тех классов грамматик и графов, для которых величина максимальной строки полиномиальна от количества вершин в графе

СПАСИБО ЗА ВНИМАНИЕ!