Использование алгоритма Охотина для синтаксического анализа графов

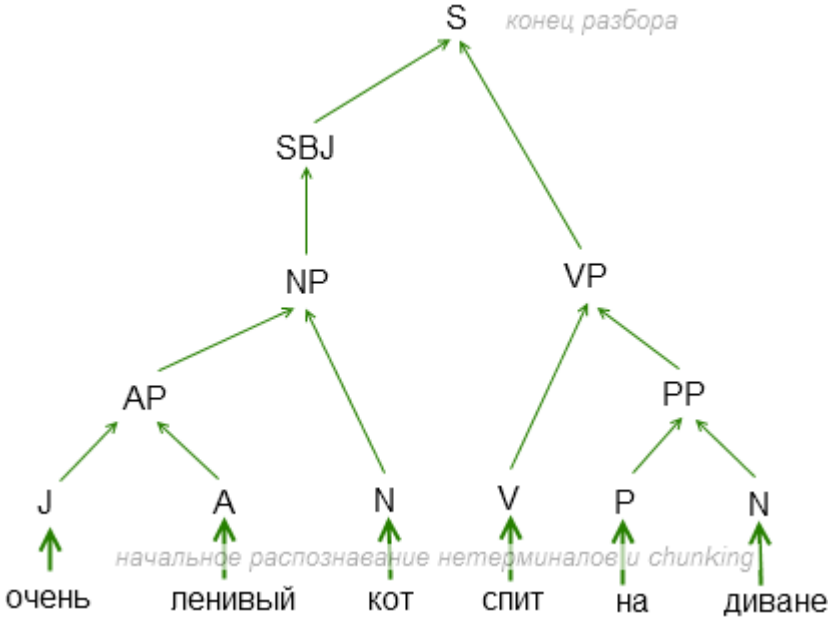
КУРАТОР: СЕМЁН ГРИГОРЬЕВ

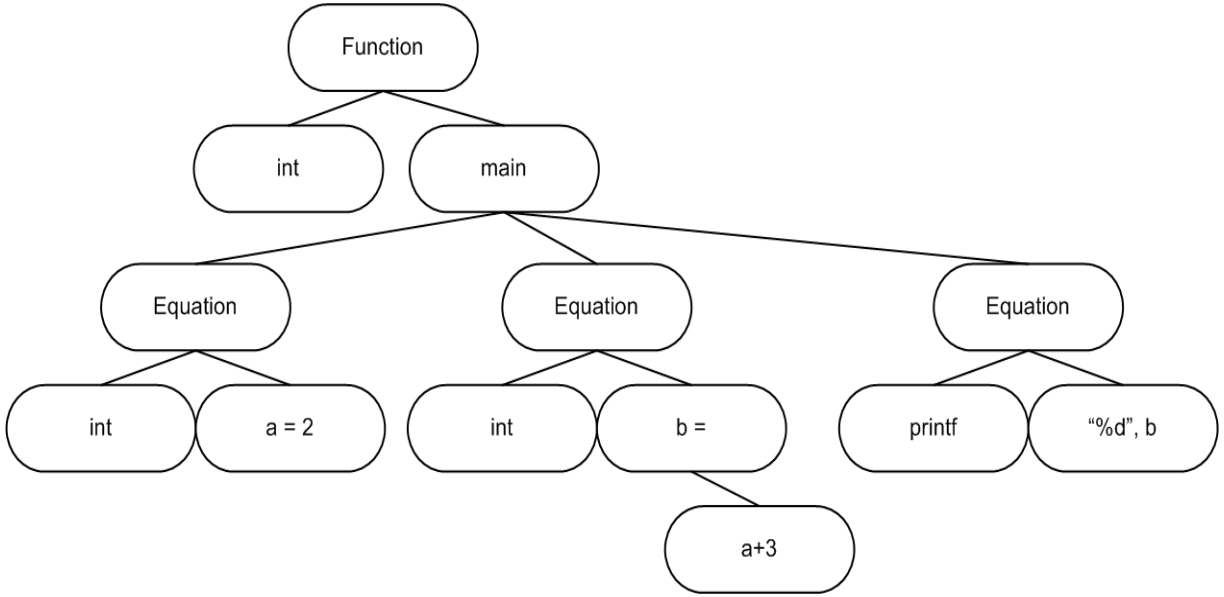
ВЫПОЛНИЛА: ЕКАТЕРИНА ШЕМЕТОВА

СПБАУ, 2018

Синтаксический анализ

- Синтаксический анализ – процесс сопоставления линейной последовательности лексем (слов, токенов) естественного языка с его формальной грамматикой.





Формальная грамматика

- Грамматика – четвёрка (N, Σ, P, S)
- Контекстно-свободная грамматика:

  $N \rightarrow A$

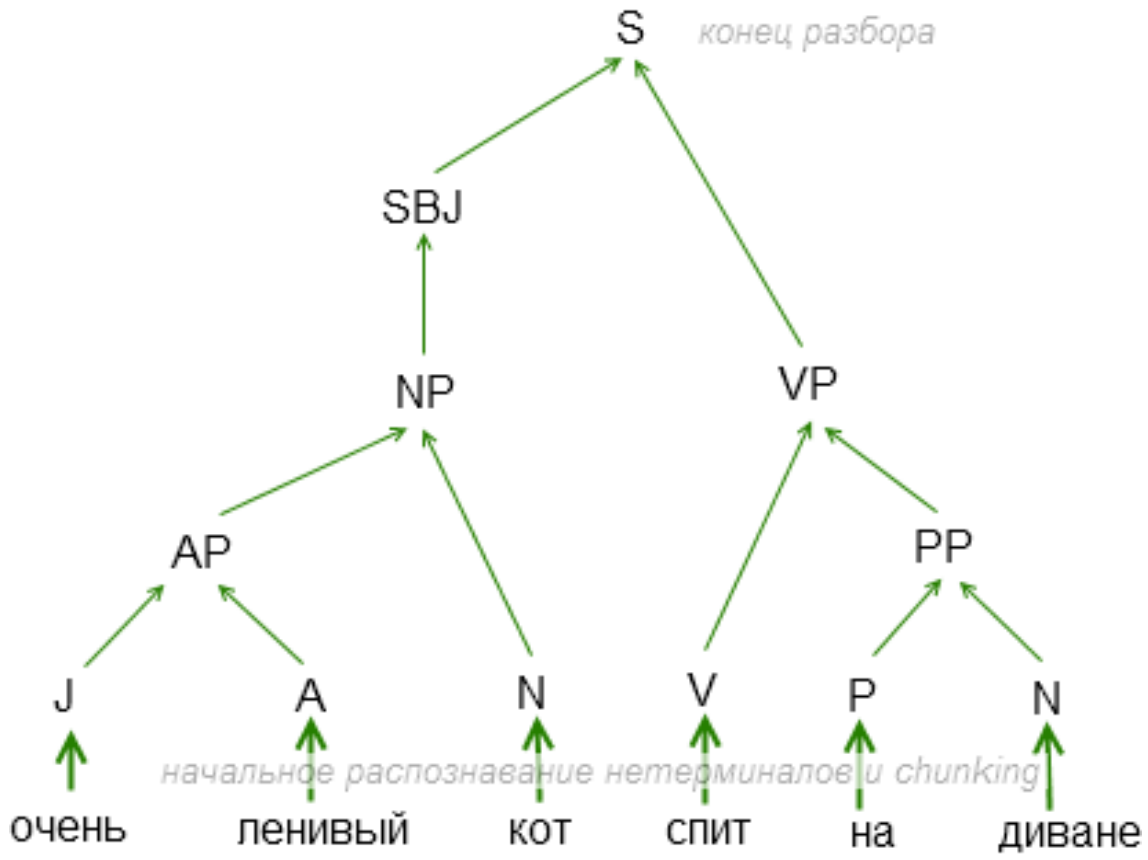
  $N \rightarrow A|B$
- Булева грамматика:

  $N \rightarrow A$

  $N \rightarrow A|B$

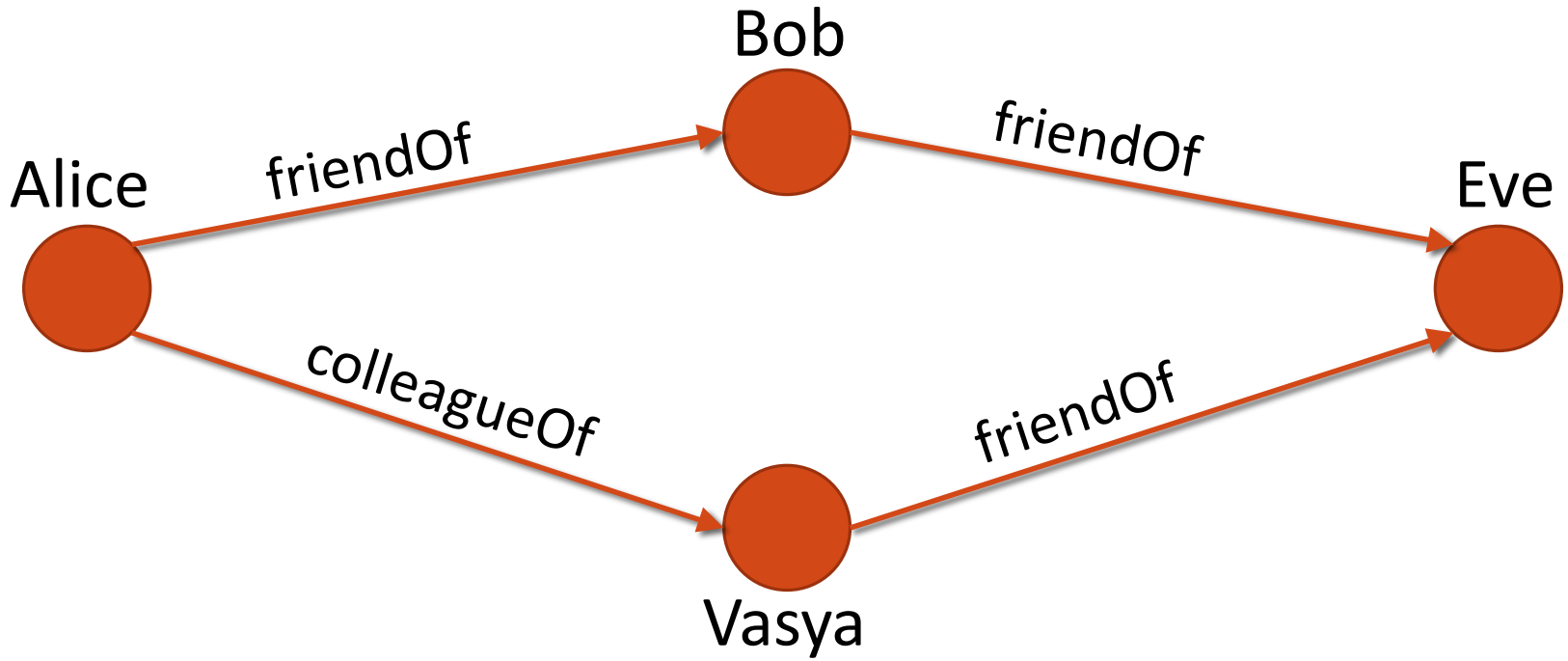
  $N \rightarrow A\&B$

  $N \rightarrow \neg A$



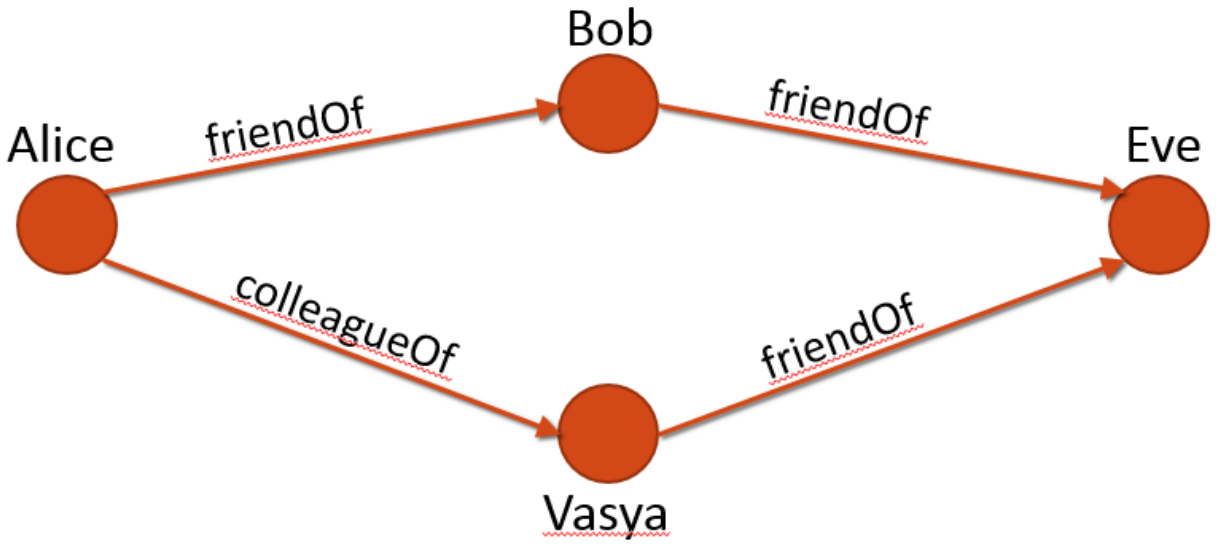
Зачем анализировать графы?

- Графовые модели данных – биоинформатика, социальные сети и многое другое.



Зачем анализировать графы?

- Исполнение запроса к графовой БД – синтаксический анализ отношений в графе с использованием правил контекстно-свободной грамматики
- Нужно найти все возможные пути в графе запроса, соответствующие правилам грамматики. Путь – слово, составленное из меток рёбер.



Варианты решения

- Для big data эффективно использовать алгоритмы, основанные на умножении матриц

Context-free recognizer for graphs (Rustam Azimov, Semyon Grigorev)

- На вход подается ациклический граф
- Время работы: O($|V|^2 |N|^3$ (BMM($|V|$) + BMU($|V|$)))
- Только для контекстно-свободной грамматики

Есть алгоритм Охотина

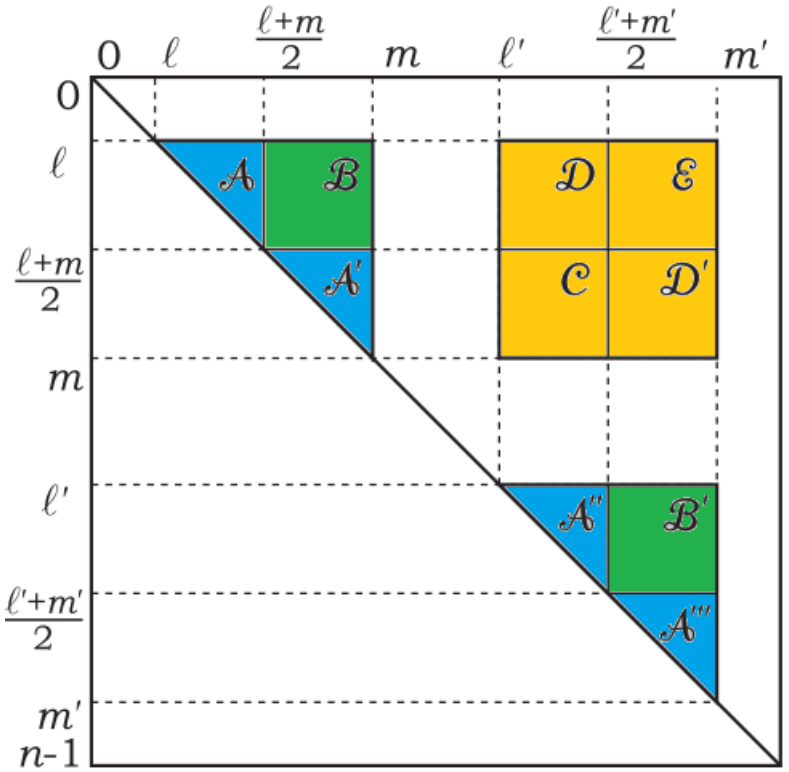
- Подходит для булевых грамматик
- Время работы: O(|G|BMM(n)log*n*)

Алгоритм Охотина быстрее и подходит для более широкого класса грамматик.

Алгоритм Охотина

- Основан на семантике единственного решения для булевых грамматик – принадлежность строки правилам грамматики зависит от принадлежности её подстрок
- Идея – разбиваем строку на всевозможные подстроки, вычисляем нетерминалы, порождающие их, постепенно увеличиваем размер подстроки, пока не найдем ответ для исходной строки
- Две процедуры – ***compute и complete***

Но как его применить к графу?



Цели и задачи работы

- **Цель работы:** Исследовать возможность использования алгоритма Охотина для синтаксического анализа графов

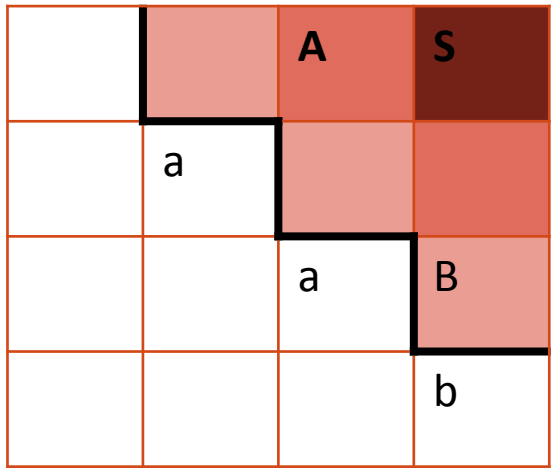
Задачи:

- Изучить алгоритм Охотина
- Найти критерии его применимости
- Понять, могут ли графы соответствовать этим критериям? Если да, то любые графы или какая-то отдельная группа?
- Каким образом представлять входные данные?
- Не потеряем ли мы асимптотику, если будем работать с графами?

Решение для графов

- Таблица парсинга у алгоритма Охотина – верхнетреугольная
- Строка состоит из подстрок
- Тогда для графа аналогом подстрок являются промежуточные пути
- Один путь можно разбить на несколько путей меньшей длины
- Тривиальный путь – ребро графа. Он соответствует слову (токену) строки

Но подстроки в основной строке идут по порядку, как мы получим такой порядок для графа?

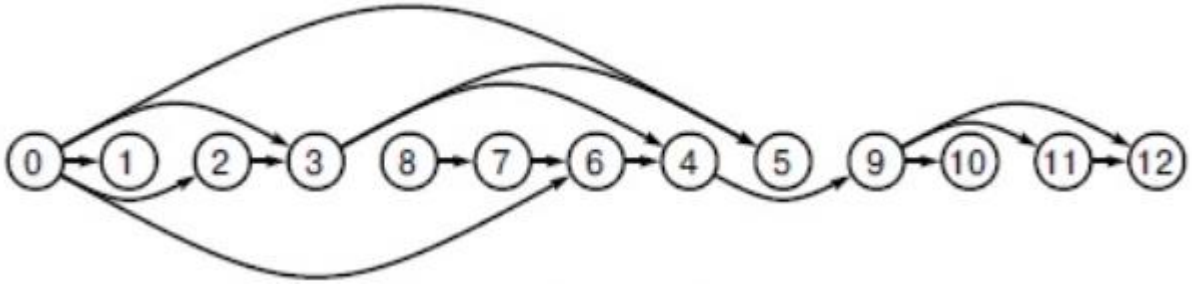


S -> AB

A -> aa

B -> b

Sorted DAG

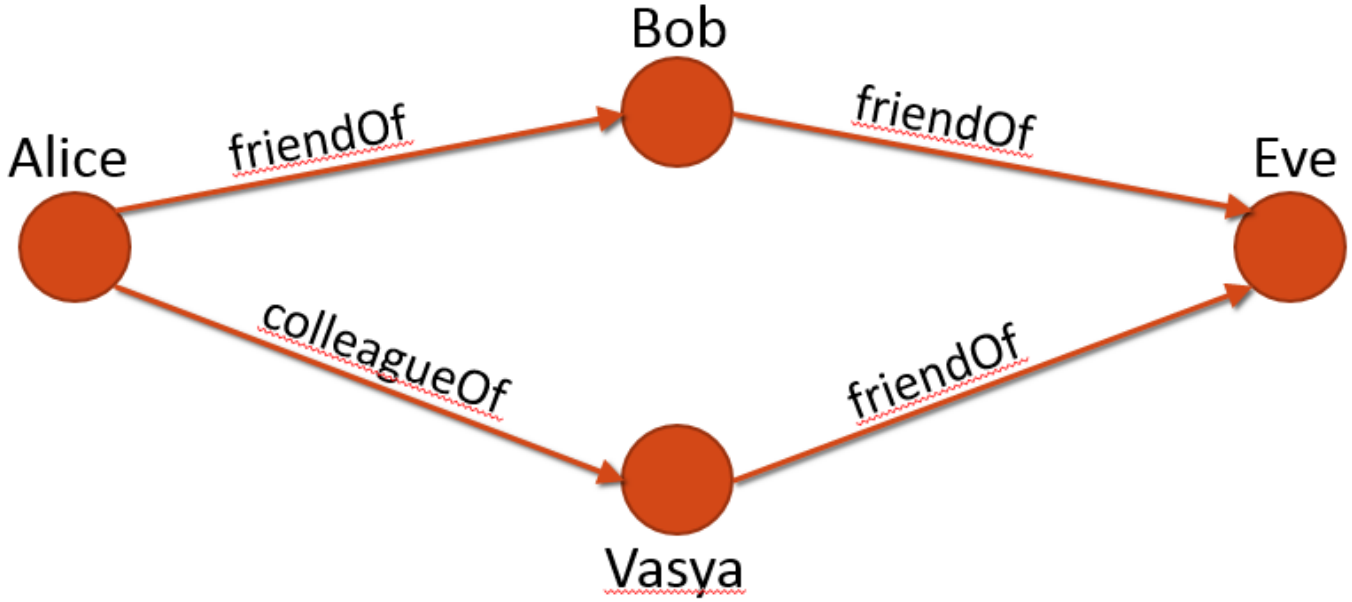


- Топологическая сортировка!
- К любому DAG-у применима топологическая сортировка
- Мы получим верхнетреугольную матрицу смежности (все тривиальные пути-ребра будут обработаны алгоритмом)
- После сортировки вершины располагаются в таком порядке, что рёбра идут от вершин с меньшим номером к большему
- Теперь порядок определен – для каждой вершины соответствующая подстрока состоит из рёбер, идущих из неё в другие вершины
- $T_{i,j}$ - путь из i-ой вершины в i-ую, и оно же – слово, составленное из меток соответствующих рёбер, начинающихся с i-ой позиции по j-ую

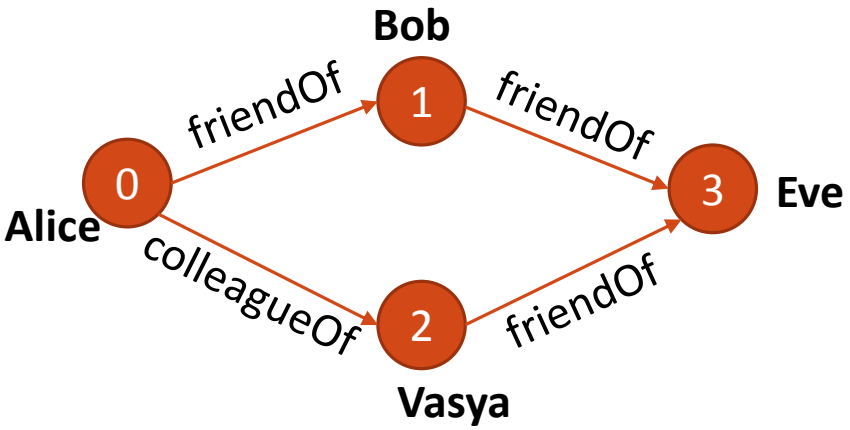
Пример

Правило грамматики - Запрос

- **A -> friendOf** - «Друг»
- **B -> colleagueOf** - «Коллега»
- **S -> AS|A** - Все дружеские связи, друзья, друзья друзей и так далее
- **C -> ¬S –** Все, кроме друзей, друзей друзей и т.д.



Пример



A -> friendOf

B -> colleagueOf

C -> ¬S

S -> AS|A

	0 (Alice)	1 (Bob)	2 (Vasya)	3 (Eve)
0 (Alice)		**A**	**B, C** = B U $T_{0,1} \times T_{1,2}$	**S** = $T_{0,1} \times T_{1,3}$ U $T_{0,2} \times T_{2,3}$
1 (Bob)			Ø	**A** = A U $T_{1,2} \times T_{1,2}$
2 (Vasya)				**A**
3 (Eve)				

Выводы

- Алгоритм Охотина можно применять для анализа объектов с заданным них порядком
- Для графов такой порядок гарантирует топологическая сортировка
- Алгоритм Охотина применим для ациклических ориентированных графов (DAG-ов) с топологической сортировкой
- Ассоциируя пути в графе с подстроками, можем производить вычисления путей «bottom up» - не теряем в асимптотике