Модели информационного поиска

Модели поиска

- Математическая модель – для осуществления процесса поиска
  - Предположения о релевантности – в математической форме

- Модели поиска
  - булева модель - основная модель поиска 60-80-е
  - векторная модель

Булев поиск

- Два возможных результата для сопоставления запроса и документа
  - TRUE и FALSE
  - Поиск по полному совпадению
  - Простейшая форма ранжирования
- Запрос может специфицироваться посредством Булевых операторов
  - AND, OR, NOT
  - Могут быть использоваться операторы близости (proximity)

Булев поиск: пример

Какие пьесы Шекспира содержат слова
Brutus *AND* ***Caesar*** but *NOT* ***Calpurnia***?

Просмотр и сканирование пьес, чтобы найти ***Brutus*** and ***Caesar,*** и вычеркнуть те пьесы, где есть ***Calpurnia***?

Ответ: нет, так нельзя. Почему?
Очень медленно (для большой коллекции)

Чтобы избежать онлайнового просмотра документов,
нужно их заранее проиндексировать

Ответы к запросу

Antony and Cleopatra, Act III, Scene ii

Agrippa [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,

When Antony found Julius ***Caesar*** dead,

He cried almost to roaring; and he wept

When at Philippi he found ***Brutus*** slain.

Hamlet, Act III, Scene ii

Lord Polonius: I did enact Julius ***Caesar*** I was kill

Capitol; ***Brutus*** killed me.

Большие коллекции

- Коллекция *N* = 1 миллион документов, каждый документ – около 100 слов
- Например, *M* = 500K отдельных термов
- Матрица терм – документ: 500K x 1M
  - Занимала бы сверхбольшие объемы
  - Очень разреженная

  - Поэтому другая форма представления

Матрица терм-документ

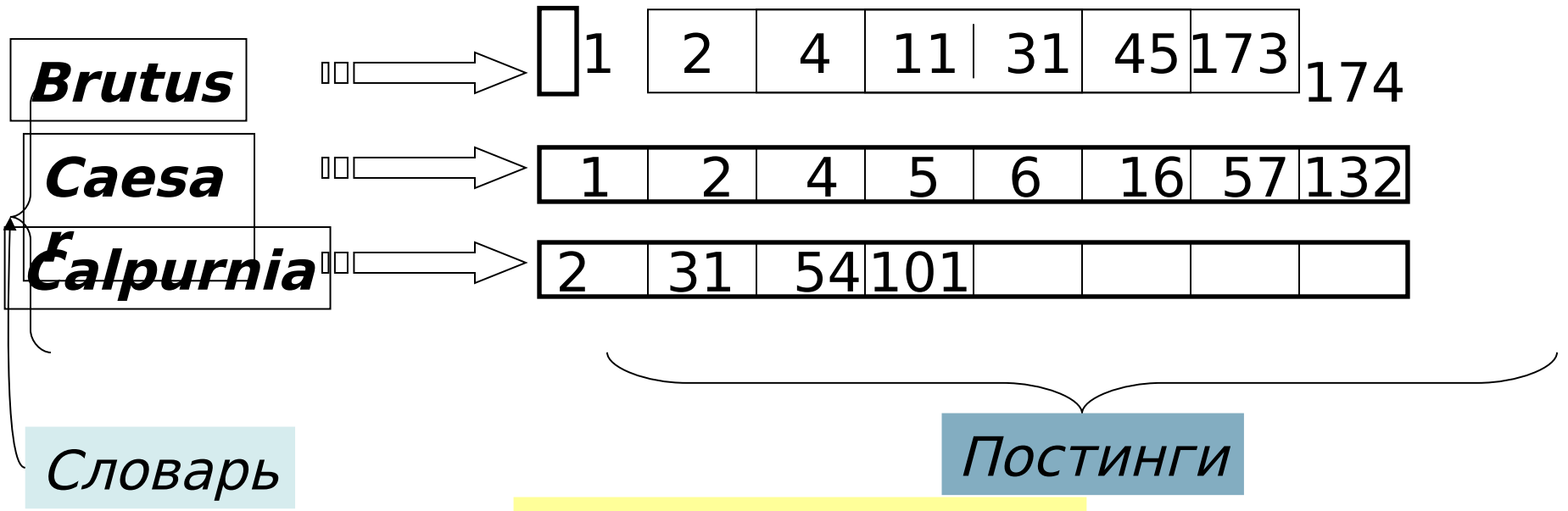
	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Brutus *AND* ***Caesar*** *BUT NOT* ***Calpurnia***

1 если текст содержит слово, 0 если не

Инвертированный индекс

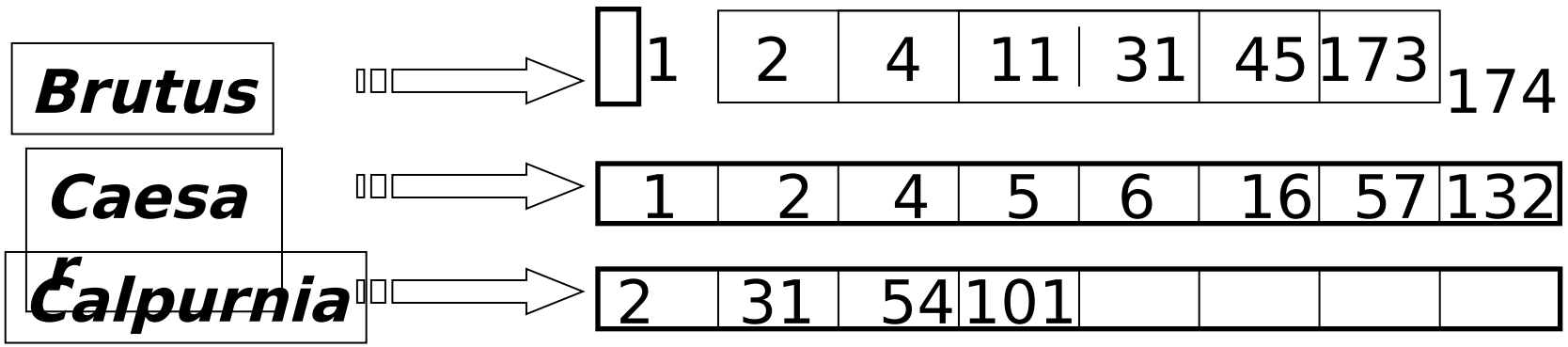
- Для каждого терма *t нужно хранить список документов, которые содержат t*.
  - Каждый документ идентифицируется номером документа - **docID**



Сортировка по

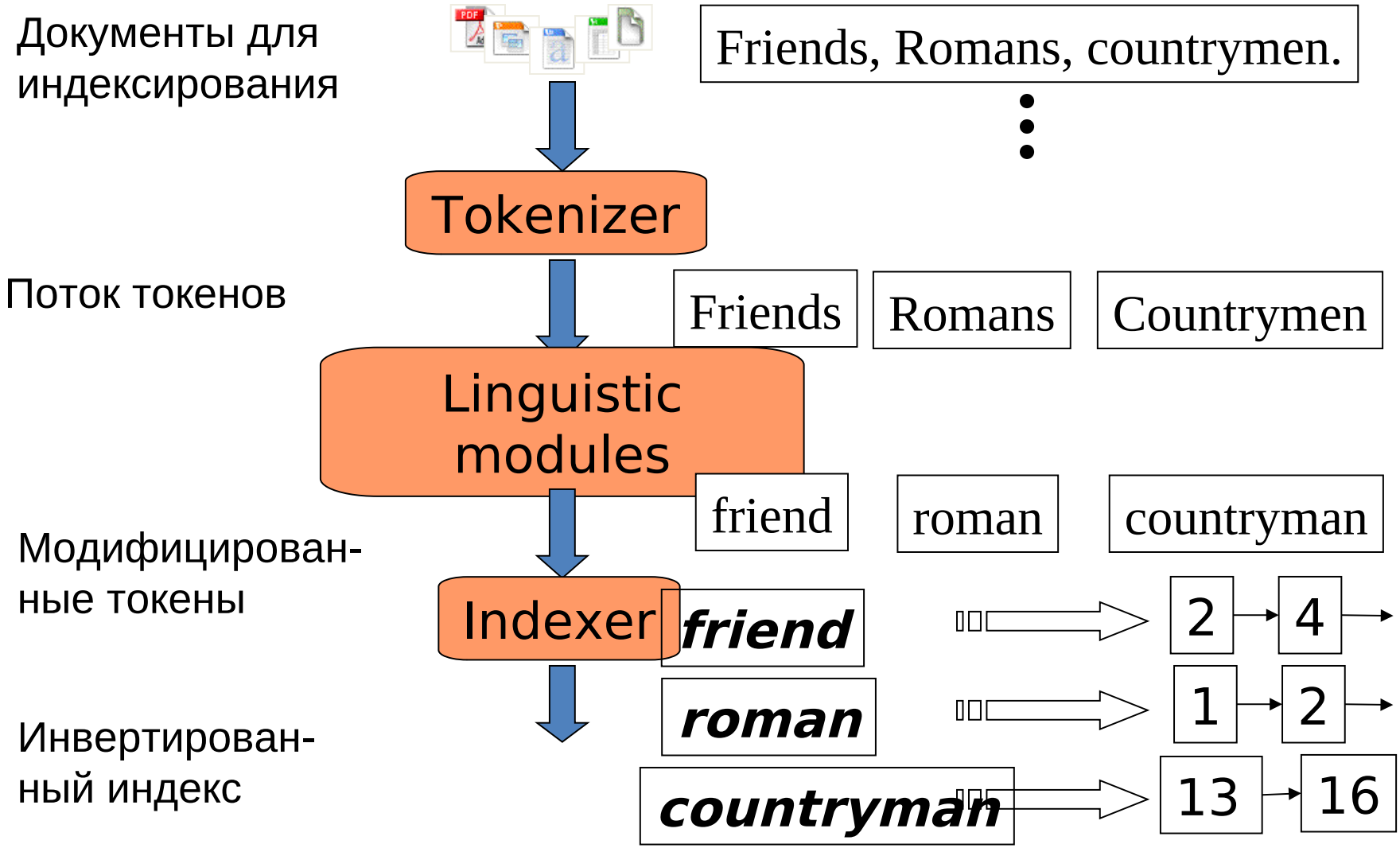
Инвертированный индекс

- Для каждого терма *t нужно хранить список документов, которые содержат t*.
  - Каждый документ идентифицируется номером документа - **docID**



Добавление слова ***Caesar*** к документу 14?

Создание инвертированного индекса



Шаги индексатора: Последовательность токенов

- Последовательность пар (Токен, DocID)

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Шаги индексатора: Сортировка

- Сортировка по токенам и затем DocId

Основной шаг индексирования

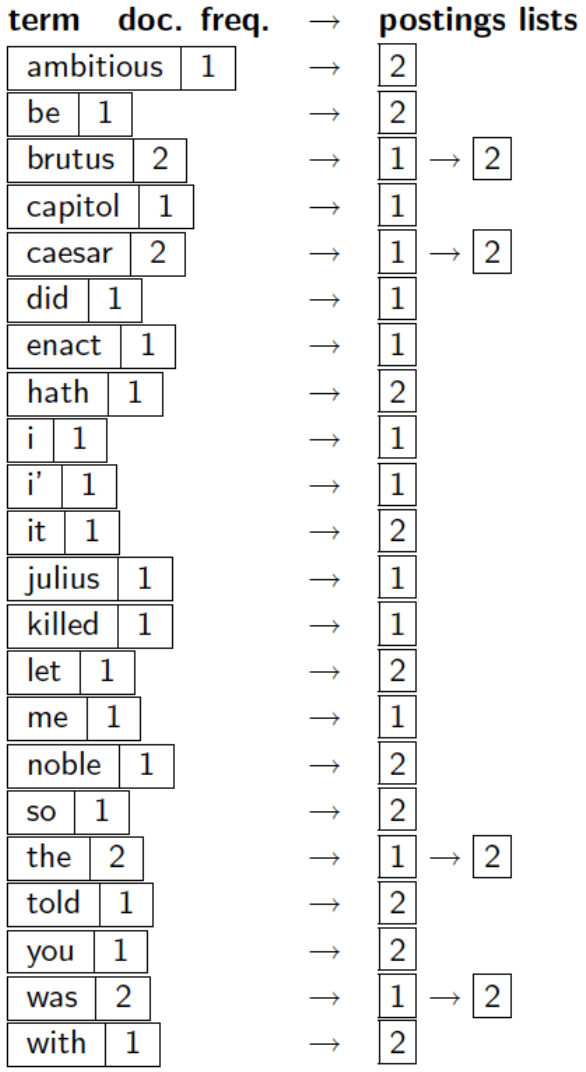
Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Шаги индексатора: Словарь & записи

- Вхождения токена в документе склеиваются
- Расчленяются на словарь и записи
- Добавляется информация о подокументной частотности.

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



Обработка запроса: AND

- Запрос:

  Brutus *AND* ***Caesar***

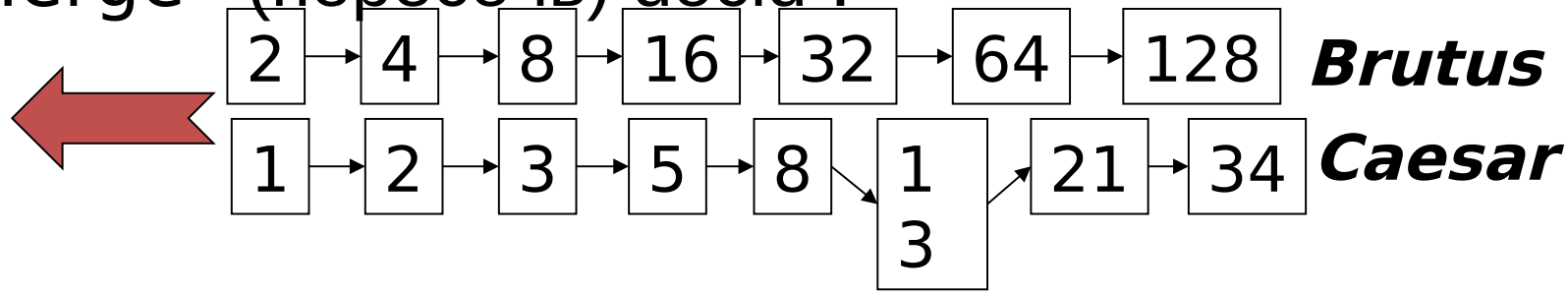
  – Найти ***Brutus*** в словаре;

  •Извлечь все его docid.

  – Найти *Caesar* в словаре;

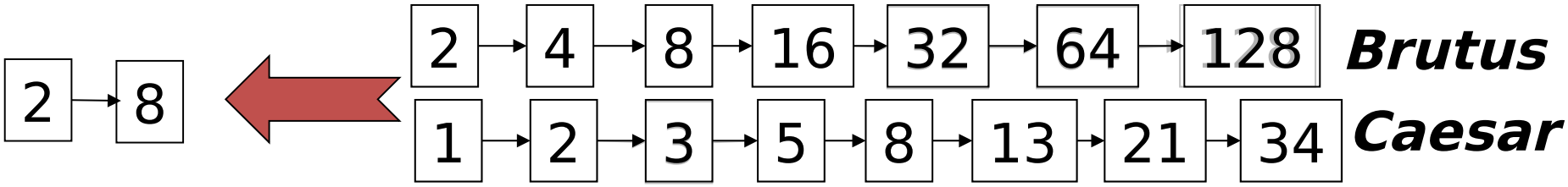
  •Извлечь все его docid.

  – “Merge” (пересечь) docid :



Пересечение списков

- Одновременный проход по спискам документов по времени линейный от количества элементов в списках



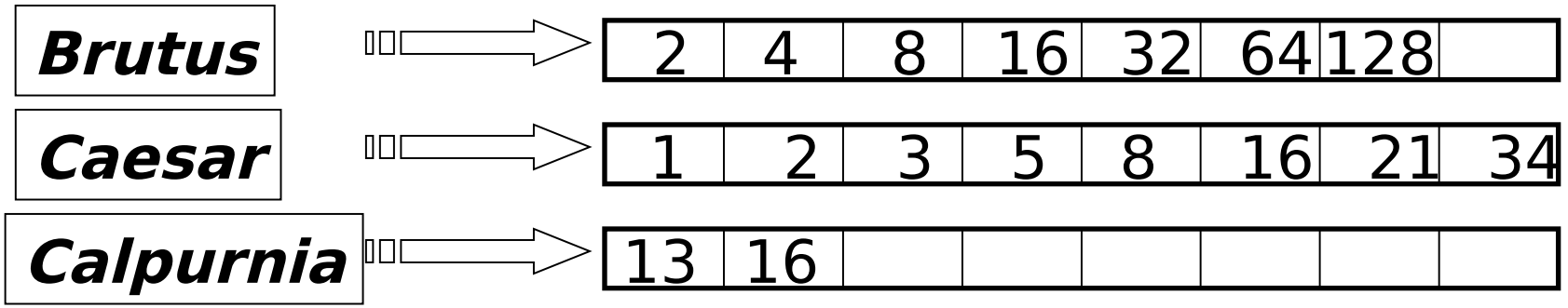
Если длины списков x и y, то пересечение $O(x+y)$ операций
Необходимо: записи в списке должны быть отсортированы по docId

Пересечение двух списков docId (a “merge” algorithm)

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

Оптимизация обработки запросов

- Лучший порядок для обработки запросов?
- Например, запрос имеет вид *AND* *n* термов.
- Для каждого из *n* термов, получить его список документов, и пересечь их.

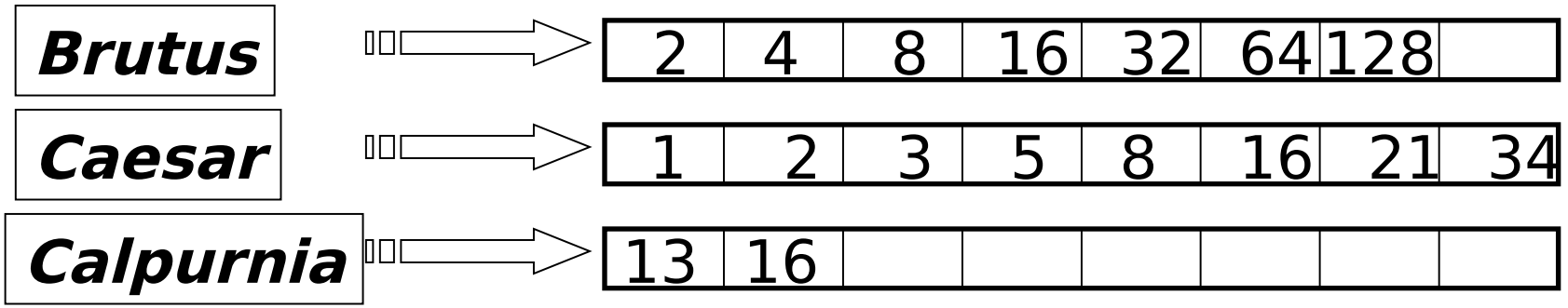


Query: ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***

Пример оптимизации обработки запросов

- Обработка должна происходить по мере увеличения частот:
  - *Начинаем с минимальных списков*.

Для этого нужно хранить количество документов в словаре



Выполняем запрос как (***Calpurnia*** *AND* ***Brutus)*** *AND* ***Caes***

Более общий случай оптимизации

- *(ВМК OR ВМиК) AND*
- *(студент OR студентка OR студенческий)*
- *AND (билет OR студбилет)*

- Получить doc. freq для всех термов.
- Оценить размер каждого *OR* используя сумму его документных частот.
- Обрабатывать в процессе увеличения количества документов в OR.

Булевские запросы: точное сопоставление

Булевская модель рассматривает запрос как булевское выражение

Булевские запросы используют *AND, OR* и *NOT связки между термами запроса*

- Каждый документ - набор термов
- Каждый запрос – по умолчанию операция И
- Точная выдача: Документ либо подходит к условию, либо не подходит.

Многие поисковые системы еще булевские:

Поиск по электронной почте, поиск в каталоге библиотеки

Пример: WestLaw http://www.westlaw.com/

Коммерческая система поиска по законодательству (платные подписчики): (функционирует с 1975; ранжирование добавлено в 1992)

Терабайты данных

Большинство пользователей используют булевсские запросы

Пример запроса:

What is the statute of limitations in cases involving the federal tort claims act?

LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM

/3 = within 3 words, /S = in same sentence

Булевский поиск

- Преимущества
  - Результаты предсказуемы, их легко объяснить
  - Могут быть встроены многие различные признаки
  - Эффективная обработка

- Недостатки
  - Качество выдачи зависит исключительно от пользователя
  - Простые запросы дают слишком много документов (нет упорядочения)
  - Длинные запросы сложно составить

Поиск, ведомый числом документов

- Последовательность запросов, направляемая числом документов
  - “lincoln” в новостных статьях
  - president AND lincoln
  - president AND lincoln AND NOT (automobile OR car)
  - president AND lincoln AND biography AND life AND birthplace AND gettysburg AND NOT (automobile OR car)
  - president AND lincoln AND (biography OR life OR birthplace OR gettysburg) AND NOT (automobile OR car)

Векторная модель

Ранжированный поиск

В булевском поиске документы либо подходят, либо не подходят.

Булевский поиск хорош для экспертов с точным пониманием потребности и коллекции

Не подходит для большинства пользователей.

Многие пользователи не могут или не хотят писать булевские запросы с операторами

Многие пользователи не хотят просматривать тысчи результатов.

И это особенно существенно для веб-поиска

Ранжированные модели поиска

Система должна не только выдавать документы, удовлетворяющие запросу, на и упорядочивать их так, чтобы лучшие документы оказались в начале списка

Обычно свободные текстовые запросы: теперь уже не предполагается, что пользователь будет применять булевские операторы – пользовательские запросы формулируются просто на естественном языке

Современные поисковые системы:

ранжированный поиск

+булевские операторы,

но ранжированный поиск ассоциируется именно со свободными текстовыми запросами

Ранжированные модели vs. Булевские модели

Когда система выдает ранжированную выдачу, то большое множество результатов, становится значительно меньшей проблемой

Мы можем просмотреть первые 10 результатов

При условии, что ранжирующий алгоритм работает

Для упорядочения документов наша модель поиска должна вырабатывать числовой вес, например [0, 1], который отражает степень соответствия между запросом и документом

Сравнение запроса и документа

Если взять запрос, состоящий из одного слова

- то какие документы будут лучше соответствовать запросу?

Если в запросе несколько слов

- то здесь не обязательно полное вхождение слов запроса в документ

- какие документы соответствуют запросу лучше?

Сравнение запроса и документа

Если взять запрос, состоящий из одного слова

- то более высокая частота слова в документе лучше

Если в запросе несколько слов

- то хорошо бы учесть долю пересечения слов запроса с документом

-

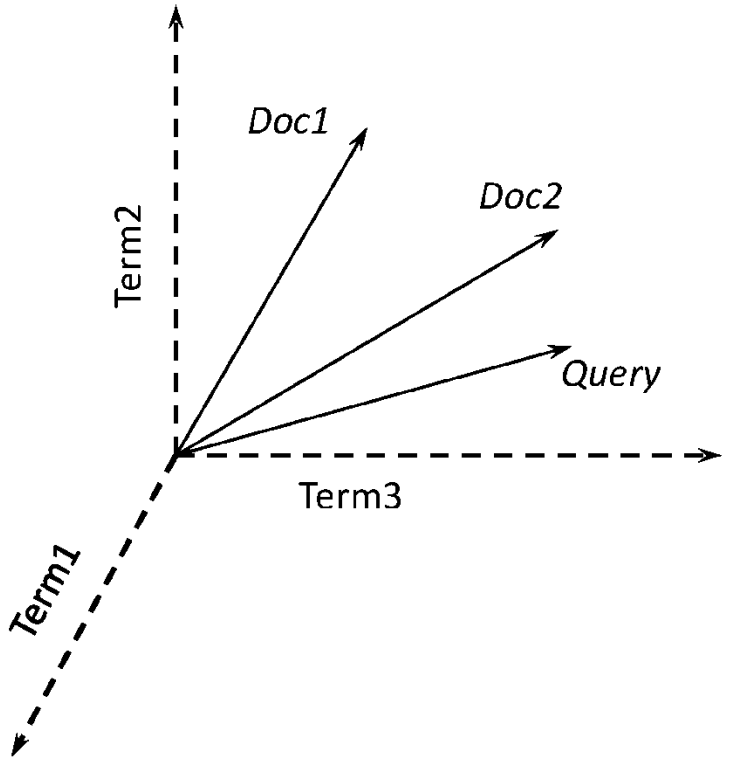
Матрица частоты употребления терма в документе

Рассмотрим число вхождений терма в документ (=частота=frequency)

Каждый документ – это вектор частот $\mathbb{N}^{|V|}$

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Векторная модель



Модель «мешок слов»

Векторное представление не учитывает порядок слов в тексте, не учитывает никакие связи между словами (синтаксические, семантические)

Банк купил компанию и *Компания купила банк* - это те же самые векторы

Модель мешок слов - bag of words

Второй фактор: подокументная частотность

Частотные слова менее информативны, чем редкие

- самые частотные слова в документе - служебные

Частотные слова: *высокий, низкий, линия*

Сверхчастотные слова типа: предлоги, союзы, которые есть во многих документах вообще иногда рассматриваются как стоп-слова и выбрасываются из документа

Чтобы учесть эту распространенность – вводится фактор, df – количество документов, в которых употреблялось это слово – подокументная частотность

Вес idf

- df_t – подокументная частотность t: количество документов, содержащих t
  - df_t – это обратная мера информативности t
  - $df_t \leq N$

- idf (обратная подокументная частота) t

$$idf_t = \log_{10}\left(N / df_t\right)$$

Логарифм используется, чтобы «смягчить» разницу в употреблении более частотных и менее частотных слова.

Пример: пусть $N = 1$ миллион документов

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1,000	
fly	10,000	2
under	100,000	1
the	1,000,000	0

$$\mathrm{idf}_t = \log_{10}\left(N/\mathrm{df}_t\right)$$

величины idf value для каждого слова в коллекции

Общая частота vs. Подокументная частота

Общая частота в коллекции – это количество упоминаний *t* в документах коллекции

Прим

Слово	Частота в коллекции	Подокументная частота
insurance	10440	3997
try	10422	8760

Какое слово имеет более высокую тематическую ценность?

tf-idf взвешивание

- Вес tf-idf это произведение tf на idf
- Могут использоваться всякие модификации tf и idf:
- 1) $w_{t,d}$=tf*idf, tf=count
- 2) $w_{t,d} = \log(1 + tf_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$

- Tf- это количество вхождений терма в документ
- Заметим, что “-” tf-idf – это не минус
  - Альтернативные имена: tf.idf, tf x idf
- Увеличивается при росте количества упоминаний в документе
- Увеличивается для редких терминов в коллекции

Матрица весов для формулы

$$w_{t,d} = \log(1 + \mathrm{tf}_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$$

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5,25	3,18	0	0	0	0,35
Brutus	1,21	6,1	0	1	0	0
Caesar	8,59	2,54	0	1,51	0,25	0
Calpurnia	0	1,54	0	0	0	0
Cleopatra	2,85	0	0	0	0	0
mercy	1,51	0	1,9	0,12	5,25	0,88
worser	1,37	0	0,11	4,15	0,25	1,95

Теперь каждый документ представлен вещественным вектором размерностью число разных слов в коллекции: tf-idf веса $\in \mathrm{R}^{|V|}$

Документы как векторы

|V|-мерное векторное пространство

Слова – это отдельные позиции (оси) в векторах

Документы – точки или вектора в пространстве

Очень большой размерности: десятки миллионов для интернета

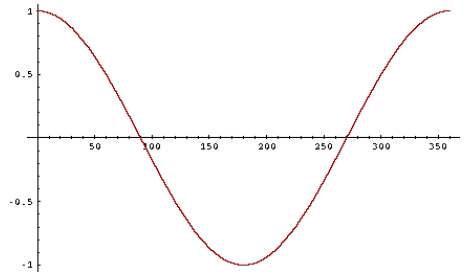
Вектора документов – разреженные.

Очень много нулей

Формализация близости векторов

- Вектора запроса и документов – точки в многомерном пространстве
- Как найти близость между векторами
  - Расстояние не подходит
  - Сравним вектор документа и вектор удвоенного документа

- Предположение: чем меньше угол, тем более похожи вектора
  - Косинусная мера: чем больше косинус угла между векторами, тем более похожи документы
  - Как вычислять косинус?



Косинус (q,d)

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\cdot\frac{\vec{d}}{|\vec{d}|} = \frac{\sum q_i d_i}{\sqrt{\sum q_i^2}\sqrt{\sum d_i^2}}$$

q_i – это tf-idf вес слова i в запросе

d_i - это tf-idf вес терма i в документе

q – может быть не только запрос, но и другой документ

$\cos(\vec{q},\vec{d})$ – это косинусное сходство между $\vec{q}$ и $\vec{d}$ или эквивалентно, косинус угла между q и d.

Пример: вычисление сходства

- Рассмотрим документы *D1, D2* и запрос *Q*
  - *D1* = (0.5, 0.8, 0.3), *D2* = (0.9, 0.4, 0.2), *Q* = (1.5, 1.0, 0)

$$Cosine(D_1, Q) = \frac{(0.5 \times 1.5) + (0.8 \times 1.0)}{\sqrt{(0.5^2 + 0.8^2 + 0.3^2)(1.5^2 + 1.0^2)}}$$

$$= \frac{1.55}{\sqrt{(0.98 \times 3.25)}} = 0.87$$

$$Cosine(D_2, Q) = \frac{(0.9 \times 1.5) + (0.4 \times 1.0)}{\sqrt{(0.9^2 + 0.4^2 + 0.2^2)(1.5^2 + 1.0^2)}}$$

$$= \frac{1.75}{\sqrt{(1.01 \times 3.25)}} = 0.97$$

Варианты весов

- Базовый – это сколько раз слово встретилось в документе (count),
- Логарифмирование count

$$w_{t,d} = \log(1 + tf_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$$

- Нормализация первого сомножителя
  - 1) tf/tf_{max} или (α+ (1-α) tf/tf_{max})
  - 2) tf/|d|, где |d| - количество слов в документе (вариант, описанный в Википедии)

Варианты для tf-idf

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

Введены сокращения для различных схем вычисления

Схема весов может различаться в запросах и документах

Многие поисковые системы позволяют использовать различное взвешивание для запросов и документов

SMART нотация: обозначает реальную комбинацию весов, которая используется в системе: *ddd.qqq,* где using the acronyms from the previous table

Стандартная схема взвешивания: lnc.ltc

Документ: логарифимический tf (l в первом столбце), нет idf и косинусная нормализация

Запрос: логарифимический tf, idf (t во втором столбце), Косинусная нормализация…

Заключение

Представление запроса как взвешенного tf-idf вектора

Представление документа как взвешенного tf-idf вектора

Вычисление косинусной меры между вектором запроса и вектором документа – вес для ранжирования

Ранжирование документов по мере снижения веса

Выдача первых K (e.g., $K = 10$) документов пользователю

!! Векторная модель может использоваться для сопоставления документов, фрагментов документов, предложений.

Задание на дом (N3) (след.раз)

Term	df	idf	d1	d2	d3
car	18165	1.65	27	4	24
auto	6723	2.08	3	33	0
Insu-rance	19241	1.62	0	33	29
best	25235	1.5	14	0	17

Задача-продолжение

- Запрос
  - Car insurance
- Вычислить вес каждого документа
  - Представить запрос как вектор
  - Представить документ как вектор
  - Вычислить сходство запроса и документа как скалярное произведение
    - Tf – **1)** число вхождений (count) **2)** log (1+count)
    - Idf – дано в третьем столбце
    - Нормализация векторов
  - Показать, какие веса у документов по отношению к запросу и как упорядочатся документы. Есть ли различия между вариантами

Задание на дом (на 3 недели - 7 октября) (N4)

- Запросы – это проанализированные вами факты из Википедии
- Коллекция собирается из всех упомянутых статьи из всех фактов -
- Документы – это предложения из статей Википедии, указанных в этих фактах, т.е. коллекция – это объединенная коллекция предложений статей всех фактов
  - Все должно быть обработано морфологическим анализатором
  - Для разделения на предложения используйте библиотеку razdel
  - https://github.com/natasha/razdel
- Задача (см. след. Слайд)

Задание на дом (на 3 недели - 7 октября) (N4)

- Нужно найти наиболее релевантные предложения
  - По tf.idf
    - df в данном случае – это количество предложений, в которых встречалось слово
    - Tf (слайд 45) д.б. реализованы два варианта=
      - 1) (natural) это количество упоминаний слова в предложении (count) и
      - 2) (augmented) 0.5+0.5 (count/count_max) при count>0, =0 при count=0
  - Нормализация запроса и предложения
  - Программа должна выстроить все предложения из всех статей по мере сходства с запросом по 2 векторным моделям.
  - В отчете должны быть показаны веса выдаваемых предложений