Обработка текстов

Лекция 2

Регулярные выражения

Базовые задачи обработки текстов

Мотивация

- Обновить цену товара в прайс-листе:

  для конкретного товара за 1000р. сделать 999.99р.
- Заменить все вхождения одного слова в тексте на другое
  - для части слова (Википедия -> Энциклопедия)
  - с учетом контекста
- Найти сообщения о терроризме
- Фильтрация нецензурных высказываний на форумах

Регулярные выражения

- Regular Expressions (RegExp)
- Языки программирования (Python, Perl, Ruby, Java, .Net)
- Текстовые редакторы (Vim, EmEdit)
- Утилиты (grep, sed)

Регулярные выражения

- Регулярные выражения - алгебраическая нотация для записи множества строк
- Функции Python

```
import re
re.search("в","пиво").group(0) # в
re.sub("о","ко","пиво") # пивко
re.findall("cd","abcdcde") # [“cd”,”cd”]
```

Регулярные выражения

- Последовательность букв: *abcd*
- Чувствительны к регистру: *“Пиво”≠ “пиво”*
- Дизъюнкция: *[П|п]иво,[abc],[1234567890]*
- Интервал: *[A-Z], [0-9], [A-Za-z]*

```
for letter in re.findall("[а-о]","пиво"):
    print(letter, end=‘ ‘)
> и в о
```

- Знак ^: *[^a]* - все кроме *“a”*
- “.” - любой символ, кроме \n

Регулярные выражения

- ? - условие для 0 или 1 вхождения символа

```
re.findall("пивк?о", "пиво или пивко")
> [“пиво”,”пивко”]
```

- Как найти “Gooooogle”?
- Счетчики
  - Gooo*gle
  - Goo+gle

```
print(re.sub(“Goo+gle”,"Google","Gooooogle"))
> Google
```

Регулярные выражения

- Якоря

  ^ - начало строки

  $ - конец строки

```
re.search("^cat(1|2)","cat1 cat2").group(0)
> cat1
re.search("cat(1|2)$","cat1 cat2").group(0)
> cat 2
```

Регулярные выражения

- Память (Memory)

```
text="A conditional random field (CRF) is a ..."
print(re.search(“\([^\)]+\)”,text).group(0))
> (CRF)
print(re.search(“\(([^\)]+)\)”,text).group(1))
> CRF
```

```
text1 = "the faster they ran, the faster we ran"
text2 = "the faster they ran, the faster we ate"
re.search("the (.*)er they (.*), the \\1er we \\2", text1) # Match
re.search("the (.*)er they (.*), the \\1er we \\2", text2) # Not match
```

- Приоритет операций

Круглые скобки	()
Счетчики	* + ? {}
Последовательности и якоря	the ^my end$
Дизъюнкция	|

Практические примеры

- Найти все вхождения слова в тексте
  - слово the в тексте 6 раз

Практические примеры

- ELIZA

User: Men are all alike.
ELIZA: IN WHAT WAY?
User: They’re always bugging us about something other
ELIZA: CAN YOU THINK OF A SPECIFIC EXAMPLE?
User: My boyfriend made me come here
ELIZA: YOUR BOYFRIEND MADE YOU COME HERE
User: He says I’m depressed much of the time
ELIZA: I AM SORRY TO HEAR YOU ARE DEPRESSED

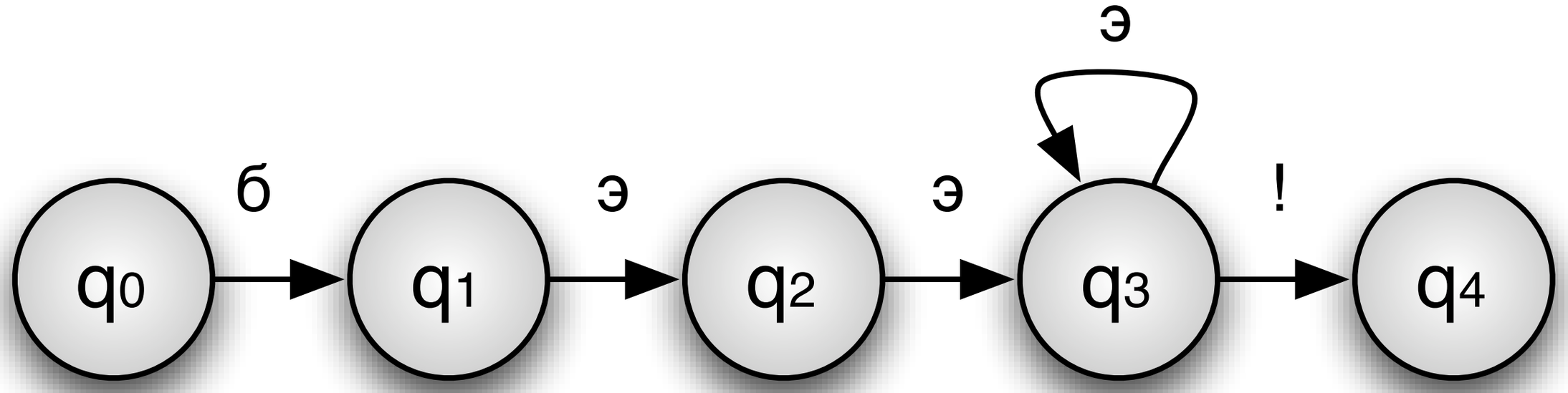
```
sentence=re.sub("I’m","YOU ARE", sentence)
...
sentence=re.sub(".* YOU ARE (depressed|sad) .*","I'M SORRY TO HEAR YOU ARE \\1",sentence)
sentence=re.sub(".* all .*","IN WHAT WAY?",sentence)
sentence=re.sub(".* always .*","CAN YOU THINK OF A SPECIFIC EXAMPLE?",sentence)
```

Конечные автоматы

- Finite-state automation (FSA)
- Один из важнейших инструментов для обработки текстов
- Могут быть использованы для реализации регулярных выражений

Использование КА для распознавания языка

- Научимся говорить с овцами
  - –бээ!
  - –бэээ!
  - –бээээ!
  - –бэээээ!
  - –...
- RE: “бээ+!”



Представление автоматов

- Текст: лента с ячейками

- Таблица переходов между состояниями

Состояние	Вход		
	б	э	!
0	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4	∅	∅	∅

Формальное определение

$Q = q_0 q_1 q_2 \ldots q_{N-1}$	конечное множество из N **состояний**
Σ	конечный **входной алфавит**
q_0	**начальное состояние**
F	множество **конечных состояний**
$\delta(q, i) : Q \times \Sigma \rightarrow Q$	**функция перехода** или матрица перехода между состояниями

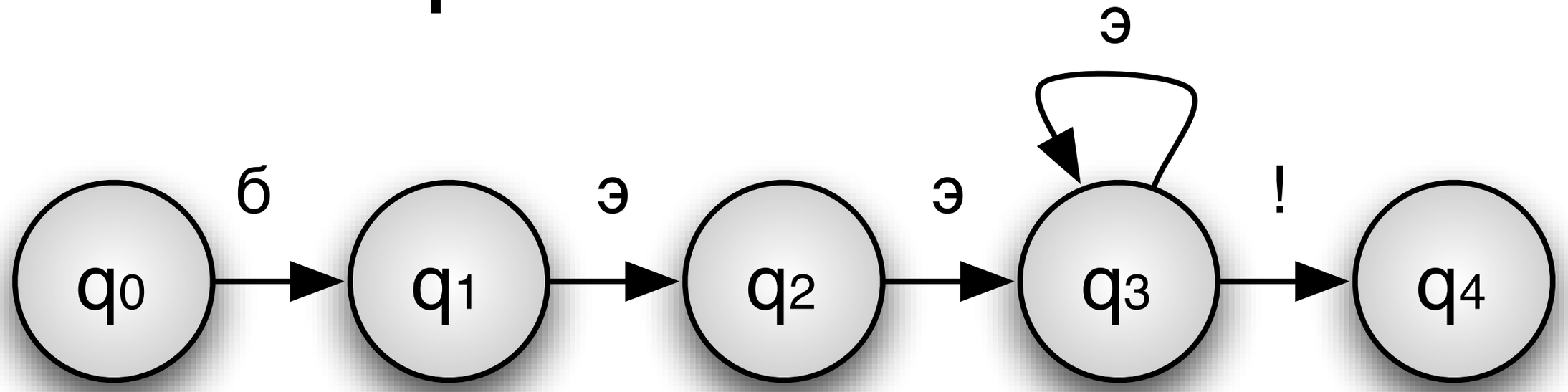
Алгоритм распознавания для детерминированного КА

```
#encoding=CP1251
def recognize(tape, machine, acceptStates):
    index = 0 # Beginning of tape
    currentState = 0 # Initial state of machine
    while True:
        if index == len(tape):
            if currentState in acceptStates:
                return True
            else:
                return False
        elif machine[currentState].has_key(tape[index]):
            currentState = machine[currentState][tape[index]]
            index+=1
        else:
            return False

machineSheep = {0:{"б":1}, 1:{"э":2}, 2:{"э":3}, 3:{"э":3,"!":4}, 4:{}}
print(recognize("бэээээээ!",machineSheep, [4]))
```

	Вход		
	б	э	!
0	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4	∅	∅	∅

Формальные языки



- **формальный язык** — это множество конечных слов (строк, цепочек) над конечным алфавитом

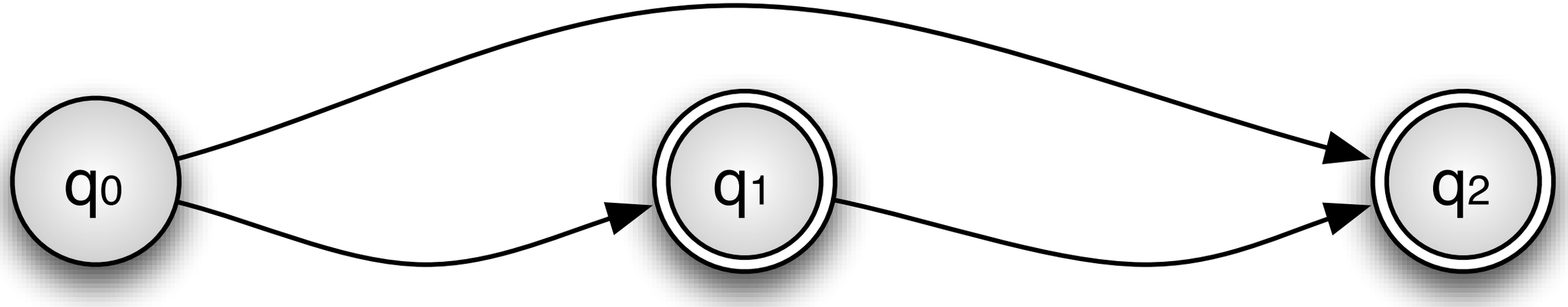
$$\Sigma = \{a, b, !\}$$
$$L(m) = \{baa!, baaa!, baaaa!, \ldots\}$$

Пример формального языка

один	шесть
два	семь
три	восемь
четыре	девять
пять	десять

одиннадцать
двенадцать
тринадцать
четырнадцать

пятнадцать
шестнадцать
семнадцать
восемнадцать
девятнадцать



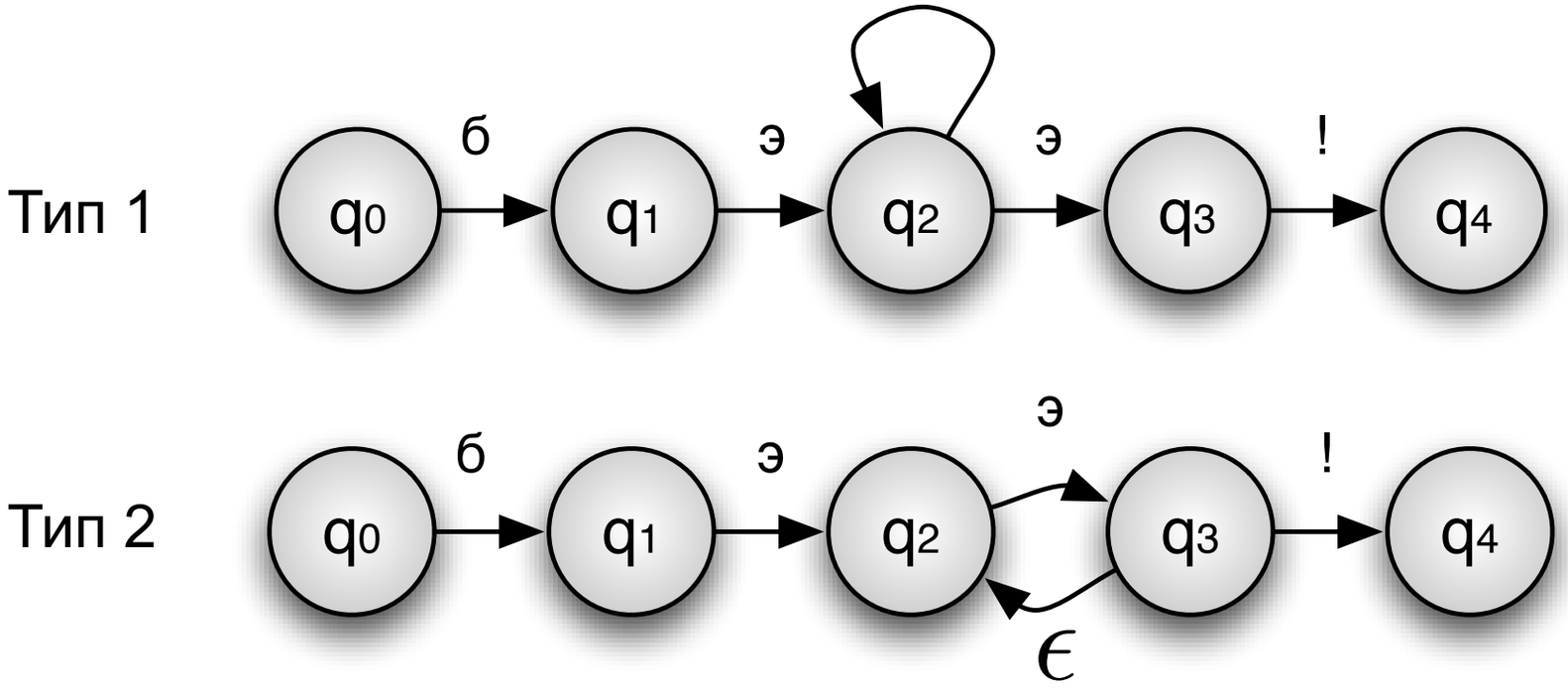
двадцать	шестьдесят
тридцать	семьдесят
сорок	восемьдесят
пятьдесят	девяносто

один	шесть
два	семь
три	восемь
четыре	девять
пять	

Недетерминированные КА

- Обобщение ДКА
- Недетерминизм двух типов

э



Распознование для НКА

- Подходы к решению проблемы недетерминизма
  - –Сохранение состояний (backup)
    - поиск в глубину и ширину
  - –Просмотр будующих состояний (look-ahead)
  - –Параллелизм

Состояние	Вход			
	б	э	!	ϵ
0	1	∅	∅	∅
1	∅	2	∅	∅
2	∅	2,3	∅	∅
3	∅	∅	4	∅
4	∅	∅	∅	∅

ДКА и НКА

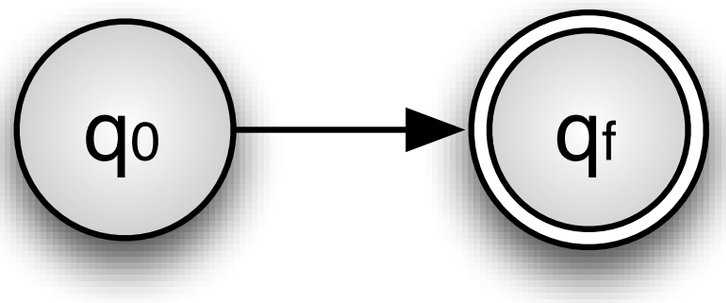
- ДКА и НКА эквивалентны
- Существует простой алгоритм для преобразования НКА в ДКА
- Идея:
  - взять все параллельные ветки НКА
  - в них взять все состояния, в которых одновременно может находиться НКА
  - объединить их в новое состояние ДКА
- В худшем случае НКА с N состояниями преобразуется в ДКА с 2^N состояниями

Регулярные языки и ДКА

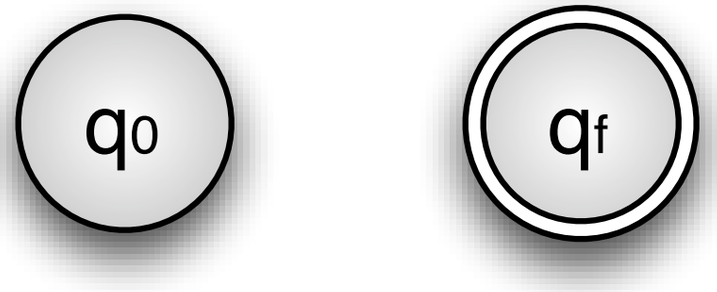
1. $\varnothing$ - регулярный язык
2. $\forall a \in \Sigma \cup \epsilon, \{a\}$ - регулярный язык
3. Для любых регулярных языков L_1 и L_2, такими также являются:
   (a) $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$, соединение L_1 и L_2
   (b) $L_1 \cup L_2$, объединение или дизъюнкция L_1 и L_2
   (c) L_1^*, замыкание (Клини) языка L_1

- регулярные языки также замкнуты относительно операций
  - пересечения
  - разности
  - дополнения
  - инверсии

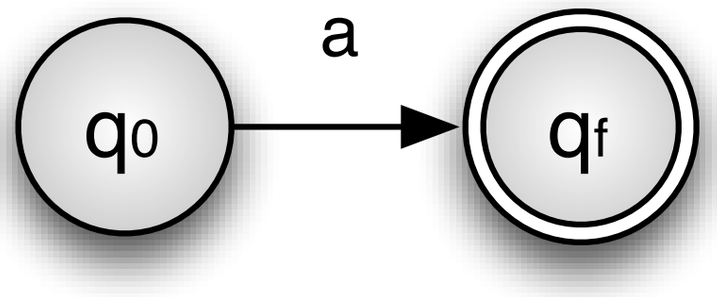
Построение автомата для регулярных выражений



$$r = \epsilon$$

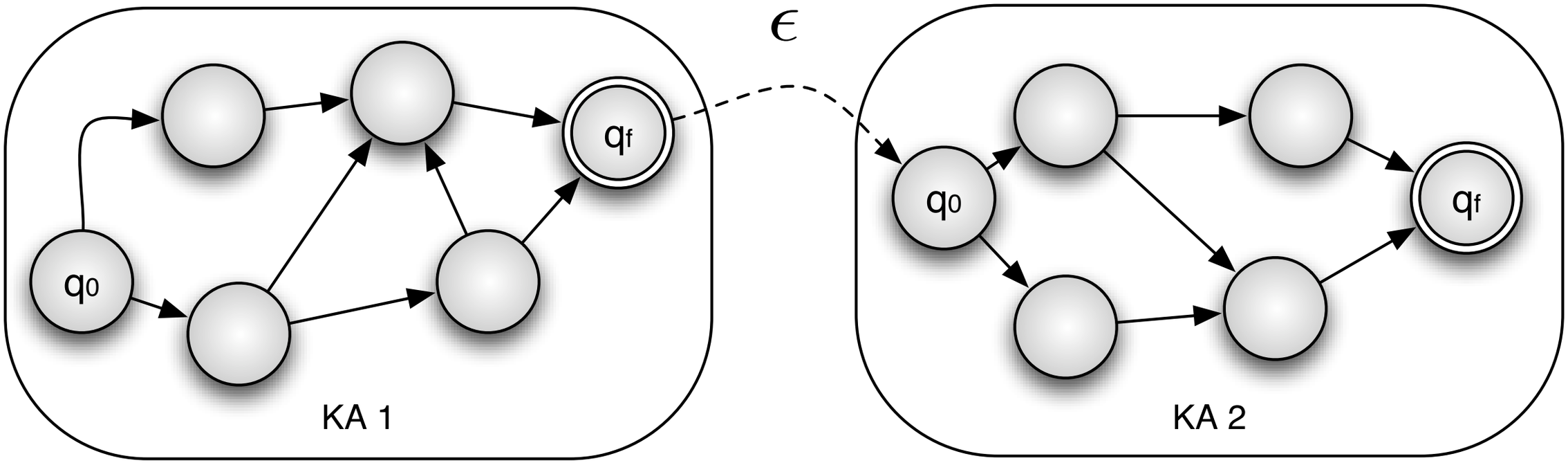


$$r = \varnothing$$



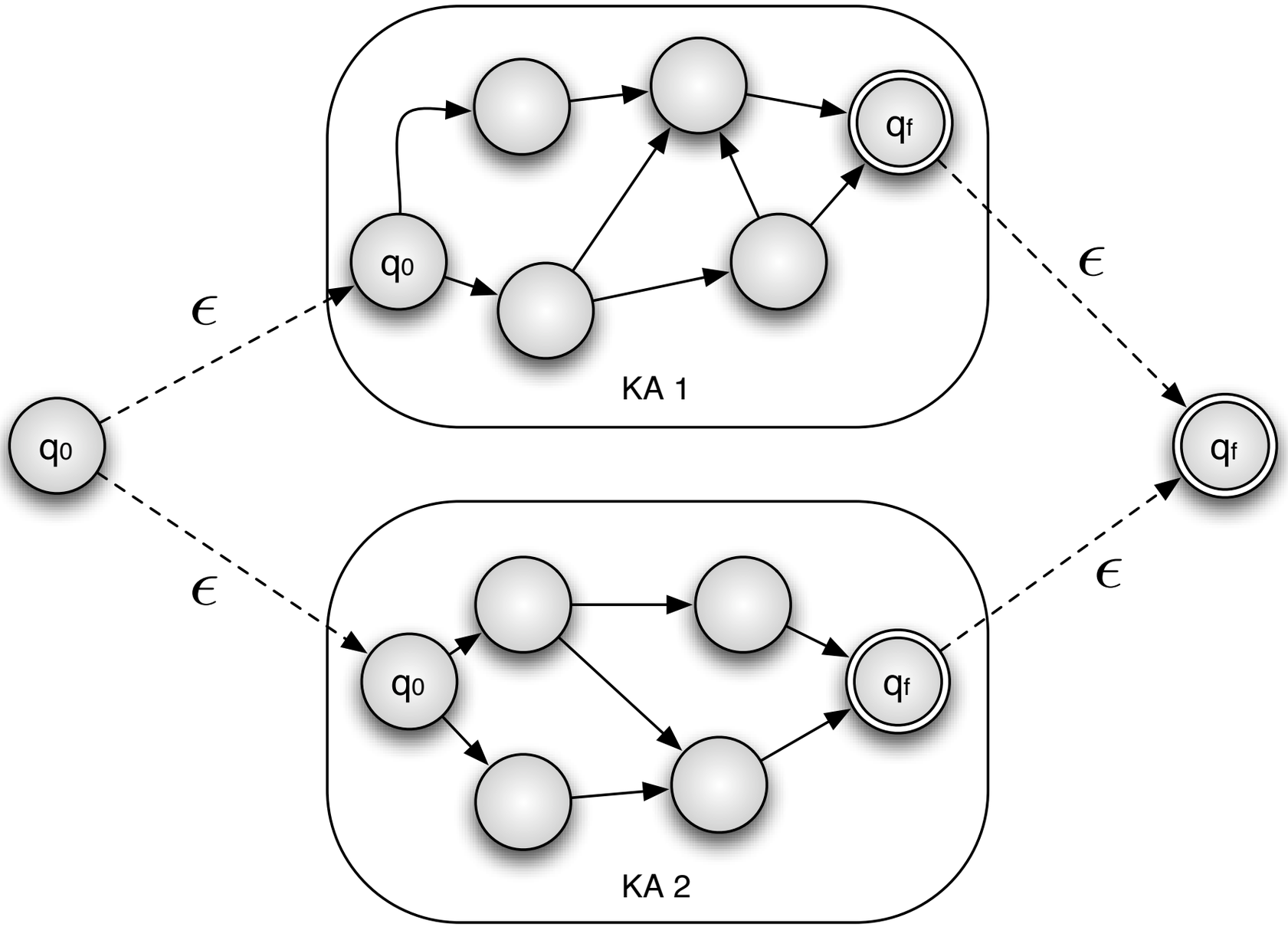
$$r = a$$

Построение автомата для регулярных выражений



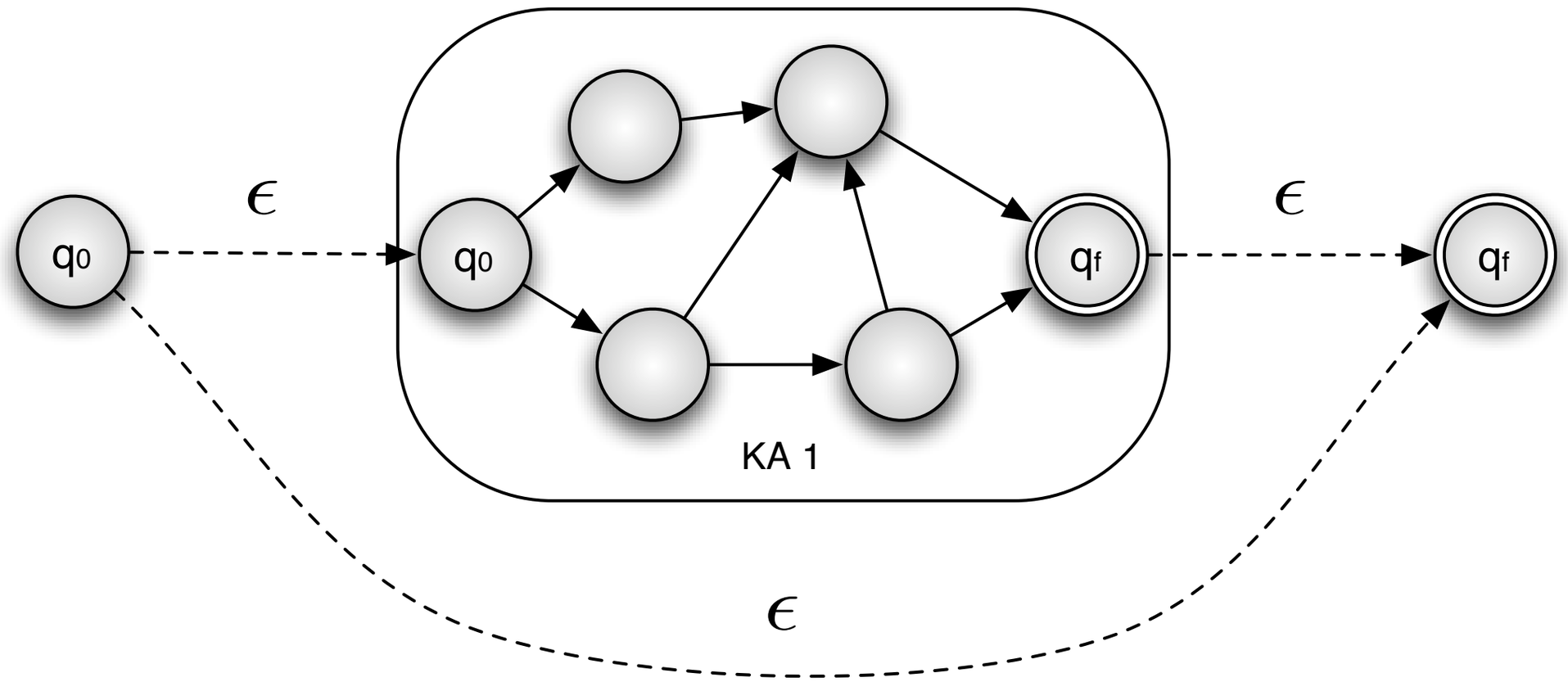
Последовательное соединение двух конечных автоматов

Построение автомата для регулярных выражений



Объединение двух конечных автоматов

Построение автомата для регулярных выражений



Замыкание конечного автомата

Базовые задачи

- Токенизация
- Стемминг и лемматизация
- Определение границ предложений
- Стоп-слова

Токенизация

- Токенизация - разбиение текста на осмысленные элементы (слова, фразы, символы), называемые **токенами**

```
>>> raw = """'When I'M a Duchess,' she said to herself, (not in a very hopeful tone
... though), 'I won't have any pepper in my kitchen AT ALL. Soup does very
... well without--Maybe it's always pepper that makes people hot-tempered,'..."""
```

```
>>> re.split(r' ', raw)
["'When", "I'M", 'a', "Duchess,'", 'she', 'said', 'to', 'herself,', '(not', 'in',
'a', 'very', 'hopeful', 'tone\nthough),', "'I", "won't", 'have', 'any', 'pepper',
'in', 'my', 'kitchen', 'AT', 'ALL.', 'Soup', 'does', 'very\nwell', 'without--Maybe',
"it's", 'always', 'pepper', 'that', 'makes', 'people', "hot-tempered,'..."]
>>> re.split(r'[ \t\n]+', raw)
["'When", "I'M", 'a', "Duchess,'", 'she', 'said', 'to', 'herself,', '(not', 'in',
'a', 'very', 'hopeful', 'tone', 'though),', "'I", "won't", 'have', 'any', 'pepper',
'in', 'my', 'kitchen', 'AT', 'ALL.', 'Soup', 'does', 'very', 'well', 'without--Maybe',
"it's", 'always', 'pepper', 'that', 'makes', 'people', "hot-tempered,'..."]
```

Токенизация

- Чтобы пунктуация не присоединялась к словам, можно попробовать оставить только символьные последовательности
- (W - эквивалент [^a-zA-Z0-9_])
- (w - эквивалент [a-zA-Z0-9_])

```
>>> re.split(r'\W+', raw)
['', 'When', 'I', 'M', 'a', 'Duchess', 'she', 'said', 'to', 'herself', 'not',
‘in', 'a', 'very', 'hopeful', 'tone', 'though', 'I', 'won', 't', 'have',
'any', ‘pepper', 'in', 'my', 'kitchen', 'AT', 'ALL', 'Soup', 'does', 'very',
'well', ‘without', 'Maybe', 'it', 's', 'always', 'pepper', 'that', 'makes',
'people', 'hot', ‘tempered', '']
```

- Но тогда появляются пустые токены

Токенизация

- Добавим границы
- \S - эквивалент [^ \t\r\n\f]

```
>>> re.findall(r'\w+|\S\w*', raw)
["'When", 'I', "'M", 'a', 'Duchess', ',', "'", 'she', 'said', 'to', 'herself', ',',
'(not', 'in', 'a', 'very', 'hopeful', 'tone', 'though', ')', ',', "'I", 'won', "'t",
'have', 'any', 'pepper', 'in', 'my', 'kitchen', 'AT', 'ALL', '.', 'Soup', 'does',
'very', 'well', 'without', '-', '-Maybe', 'it', "'s", 'always', 'pepper', 'that',
'makes', 'people', 'hot', '-tempered', ',', "'", '.', '.', '.']
```

- Теперь нужно не разбивать слова на токены

```
>>> re.findall(r"\w+(?:[-']\w+)*|'|[-.(]+|\S\w*", raw)
["'", 'When', "I'M", 'a', 'Duchess', ',', "'", 'she', 'said', 'to', 'herself', ',',
'(', 'not', 'in', 'a', 'very', 'hopeful', 'tone', 'though', ')', ',', "'", 'I',
"won't", 'have', 'any', 'pepper', 'in', 'my', 'kitchen', 'AT', 'ALL', '.', 'Soup',
'does', 'very', 'well', 'without', '--', 'Maybe', "it's", 'always', 'pepper',
'that', 'makes', 'people', 'hot-tempered', ',', "'", '...']
```

Токенизация

- В NLTK есть regexp_tokenizer

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(-\w+)*        # words with optional internal hyphens
...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
...   | \.\.\.            # ellipsis
...   | [][.,;"'?():-_`]  # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

- Если нет специфичных требований можно использовать WordPunctTokenizer

```
from nltk import WordPunctTokenizer
raw = """'When I'M a Duchess,' she said to herself, (not in a very hopeful tone
          though), 'I won't have any pepper in my kitchen AT ALL. Soup does very
          well without--Maybe it's always pepper that makes people hot-tempered,'..."""
tokens = WordPunctTokenizer().tokenize(raw)
print(tokens)

["'", 'When', 'I', "'", 'M', 'a', 'Duchess', ",'", 'she', 'said', 'to', 'herself', ',', '(',
 'not', 'in', 'a', 'very', 'hopeful', 'tone', 'though', '),', "'", 'I', 'won', "'", 't', 'have',
 'any', 'pepper', 'in', 'my', 'kitchen', 'AT', 'ALL', '.', 'Soup', 'does', 'very', 'well',
 'without', '--', 'Maybe', 'it', "'", 's', 'always', 'pepper', 'that', 'makes', 'people',
 'hot', '-', 'tempered', ",'..."]
```

Токенизация

- Многозначность определения токена
  - хэштеги (#текст)
  - “I’m” - один токен?
  - “won’t” - один токен?
    - Dr. - токен?
- Зависит от задачи

Токенизация

- В каком виде лучше представлять результат токенизации?
  - список токенов
    - простая модель
      - что, если нужно сразу несколько токенизаторов?
      - что если нужно понимать где в тексте оригинальное слово?
- Более общий способ представления результатов анализа текстов - модель аннотаций

Аннотации

- Аннотация - в общем случае тройка
  - начало
  - конец
  - значение (не обязательно)

- Пример токенизации

>>> [(0, 1), (1, 5), (6, 9), (10, 11), **(12, 19)**, (19, 20), (20, 21), (22, 25), (26, 30), (31, 33), (34, 41), (41, 42), (43, 44), (44, 47), (48, 50), (51, 52), (53, 57), (58, 65), (66, 70), (82, 88), (88, 89), (89, 90), (91, 92), (92, 93), (94, 96), (96, 99), (100, 104), (105, 108), (109, 115), (116, 118), (119, 121), (122, 129), (130, 132), (133, 137), (138, 142), (143, 147), (148, 152), (164, 168), (169, 183), (184, 186), (186, 188), (189, 195), (196, 202), (203, 207), (208, 213), (214, 220), (221, 233), (233, 234), (234, 238)]

```
print(raw[109:115])
```

>>> pepper

Аннотации

- Аннотации используются во многих проектах по обработке текстов
    - Apache UIMA
    - Texterra - ISPRAS API (https://api.ispras.ru)

```
from ispras import texterra
t = texterra.API('API KEY')
tokens = t.tokenizationAnnotate(raw)
print([(token['start'], token['end']) for token in tokens])
```

Сегментация

- В китайском языке слова не разделяются проблемными символами
    - 戴帽子的貓 -> Thecatinthehat
- Жадный алгоритм по словарю
- Многозначность
    - thetabledownthere
    - the table down there
    - theta bled own there
- Проблемы, если слова нет в словаре
- Но в целом, алгоритм неплохо работает для китайского языка, так как слова имеют схожую длину

Сегментация

- Обозначим сегментацию через бинарный вектор

```
text = "doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"
seg1 = “0000000000000001000000000100000000000000100000000000”
seg2 = “0100100100100001001001000010100100010010000100010010000"
```

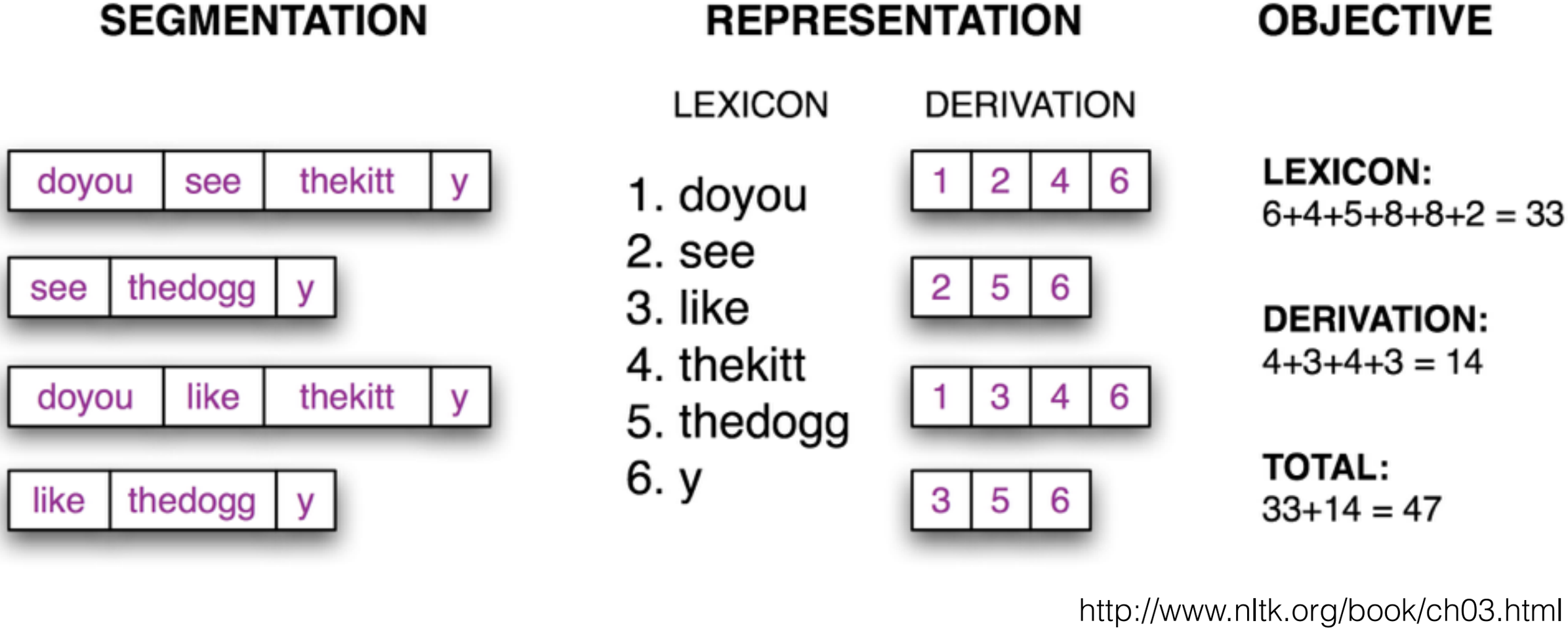
```
def segment(text, segs):
    words = []
    last = 0
    for i in range(len(segs)):
        if segs[i] == '1':
            words.append(text[last:i+1])
            last = i+1
    words.append(text[last:])
    return words
```

```
print(segment(text, seg2))
>>> ['do', 'you', 'see', 'the', 'kitty', 'see', 'the', 'doggy', 'do', 'you', 'like',
'the', 'kitty', 'like', 'the', 'doggy']
```

*NLTK Book

Сегментация

- Придумаем функцию оценки качества сегментации
  - размер лексикона (длина слов плюс разделительный символ для каждого слова)
  - количество информации, необходимое для реконструкции исходного текста из лексикона



http://www.nltk.org/book/ch03.html

Сегментация

- Придумаем функцию оценки качества сегментации
  - размер лексикона (длина слов плюс разделительный символ для каждого слова)
  - количество информации, необходимое для реконструкции исходного текста из лексикона

```
text = "doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"
seg1 = “0000000000000010000000001000000000000001000000000”
seg2 = “0100100100100001001001000010100100010010000100010010000"
```

```
def evaluate(text, segs):
    words = segment(text, segs)
    text_size = len(words)
    lexicon_size = sum(len(word) + 1 for word in set(words))
    return text_size + lexicon_size
```

```
print(evaluate(text, seg1))
>>> 64
print(evaluate(text, seg2))
>>> 48
```

Сегментация

- Найдем минимум функции алгоритмом имитации отжига

```
from random import randint

def flip(segs, pos):
    return segs[:pos] + str(1-int(segs[pos])) + segs[pos+1:]

def flip_n(segs, n):
    for i in range(n):
        segs = flip(segs, randint(0, len(segs)-1))
    return segs

def anneal(text, segs, iterations, cooling_rate):
    temperature = float(len(segs))
    while temperature > 0.5:
        best_segs, best = segs, evaluate(text, segs)
        for i in range(iterations):
            guess = flip_n(segs, int(round(temperature)))
            score = evaluate(text, guess)
            if score < best:
                best, best_segs = score, guess
        score, segs = best, best_segs
        temperature = temperature / cooling_rate
        print(evaluate(text, segs), segment(text, segs))
    return segs

anneal(text, seg1, 5000, 1.2)
```

Сегментация

- Результат работы

```
(64, ['doyouseethekitty', 'seethedoggy', 'doyoulikethekitty', 'likethedoggy'])
(64, ['doyouseethekitty', 'seethedoggy', 'doyoulikethekitty', 'likethedoggy'])
(64, ['doyouseethekitty', 'seethedoggy', 'doyoulikethekitty', 'likethedoggy'])
(64, ['doyouseethekitty', 'seethedoggy', 'doyoulikethekitty', 'likethedoggy'])
(64, ['doyouseethekitty', 'seethedoggy', 'doyoulikethekitty', 'likethedoggy'])
(63, ['doyouse', 'et', 'hekitty', 'seethedoggydoyoulik', 'et', 'hekitty', 'likethe', 'd', 'oggy'])
(62, ['doyouse', 'ethekitty', 'seethe', 'doggydoyoulik', 'ethekitty', 'l', 'ikethed', 'oggy'])
(60, ['do', 'youse', 'ethekitty', 'seethedoggydoyoulik', 'ethekitty', 'l', 'ikethedoggy'])
(60, ['do', 'youse', 'ethekitty', 'seethedoggydoyoulik', 'ethekitty', 'l', 'ikethedoggy'])
(57, ['do', 'youse', 'ethekitty', 'see', 'thedoggy', 'doyoulik', 'ethekitty', 'l', 'ike', 'thedoggy'])
(53, ['doyouse', 'ethekitty', 'see', 'thedoggy', 'doyoulik', 'ethekitty', 'like', 'thedoggy'])
(51, ['doyou', 'se', 'ethekitty', 's', 'ee', 'thedoggy', 'doyou', 'lik', 'ethekitty', 'lik', 'e', 'thedoggy'])
(49, ['doyou', 'se', 'ethekitty', 'see', 'thedoggy', 'doyou', 'lik', 'ethekitty', 'lik', 'e', 'thedoggy'])
(49, ['doyou', 'se', 'ethekitty', 'see', 'thedoggy', 'doyou', 'lik', 'ethekitty', 'lik', 'e', 'thedoggy'])
(46, ['doyou', 'se', 'ethekitty', 'se', 'e', 'thedoggy', 'doyou', 'lik', 'ethekitty', 'lik', 'e', 'thedoggy'])
(46, ['doyou', 'se', 'ethekitty', 'se', 'e', 'thedoggy', 'doyou', 'lik', 'ethekitty', 'lik', 'e', 'thedoggy'])
(46, ['doyou', 'se', 'ethekitty', 'se', 'e', 'thedoggy', 'doyou', 'lik', 'ethekitty', 'lik', 'e', 'thedoggy'])
```

Стемминг и лемматизация

- Часто необходимо обрабатывать разные формы слова одинаково.
- Например, при поиске: по запросам “кошками” и “кошкам” ожидаются одинаковые ответы
- **Стемминг** - это процесс нахождения основы слова, которая не обязательно совпадает с корнем слова
- **Лемматизация** - приведение слова к словарной форме

Стемминг

- **Стемминг** - это процесс нахождения основы слова, которая не обязательно совпадает с корнем слова
- Стемминг отбрасывает суффиксы и окончания до неизменяемой формы слова
- Примеры:
  - кошка -> кошк
  - кошками -> кошк
  - пылесосы -> пылесос

Стемминг

- Наиболее распространенный стеммер - Snowball из проекта Apache Lucene
- Работает для нескольких языков, включая русский

```
#coding: utf-8

from nltk import SnowballStemmer

word = "пылесосы".decode("utf-8")
stem = SnowballStemmer("russian").stem(word)
print(stem)
```

http://snowball.tartarus.org/algorithms/russian/stemmer.html

Лемматизация

- У разных слов часто совпадает основа
  - **пол** : полу , пола , поле , полю , поля , пол , полем , полях , полям
  - **лев** : левый, левая, лев
- Увеличивается многозначность и ухудшаются результаты работы приложений

- **Лемматизация** - приведение слова к словарной форме
- Примеры:
  - Кошки -> кошка
  - Кошками -> кошка

Лемматизация

- Для английского языка можно использовать nltk.WordNetLemmatizer()
- Для русского языка:
    - Илья Сегалович, Михаил Маслов. *Русский морфологический анализ и синтез с генерацией моделей словоизменения для не описанных в словаре слов.*
- Используется словарь словоформ А.А. Зализняка
- Находит нормальную форму даже для не словарных слов
- Алгоритм реализован в системе mystem

Лемматизация

- Вариант русского лемматайзера реализован в ISPRAS API
- lemmatizationAnnotate(text)

Так говорила в июле 1805 года известная Анна Павловна Шерер, фрейлина и приближенная императрицы Марии Феодоровны, встречая важного и чиновного князя Василия, первого приехавшего на ее вечер. Анна Павловна кашляла несколько дней, у нее был грипп, как она говорила (грипп был тогда новое слово, употреблявшееся только редкими). В записочках, разосланных утром с красным лакеем, было написано без различия во всех:

так говорить в июль 1805 год известный анна павловна шерер фрейлин и приближенный императрица мария феодоровна встречать важный и чиновной князь василий первый приехавший на она вечер анна павловна кашлял несколько день у она быть грипп как она говорить грипп быть тогда новый слово употребляться только редкий в записочка разосылать утро с красный лакей быть писать без различие в весь

Определение границ предложений

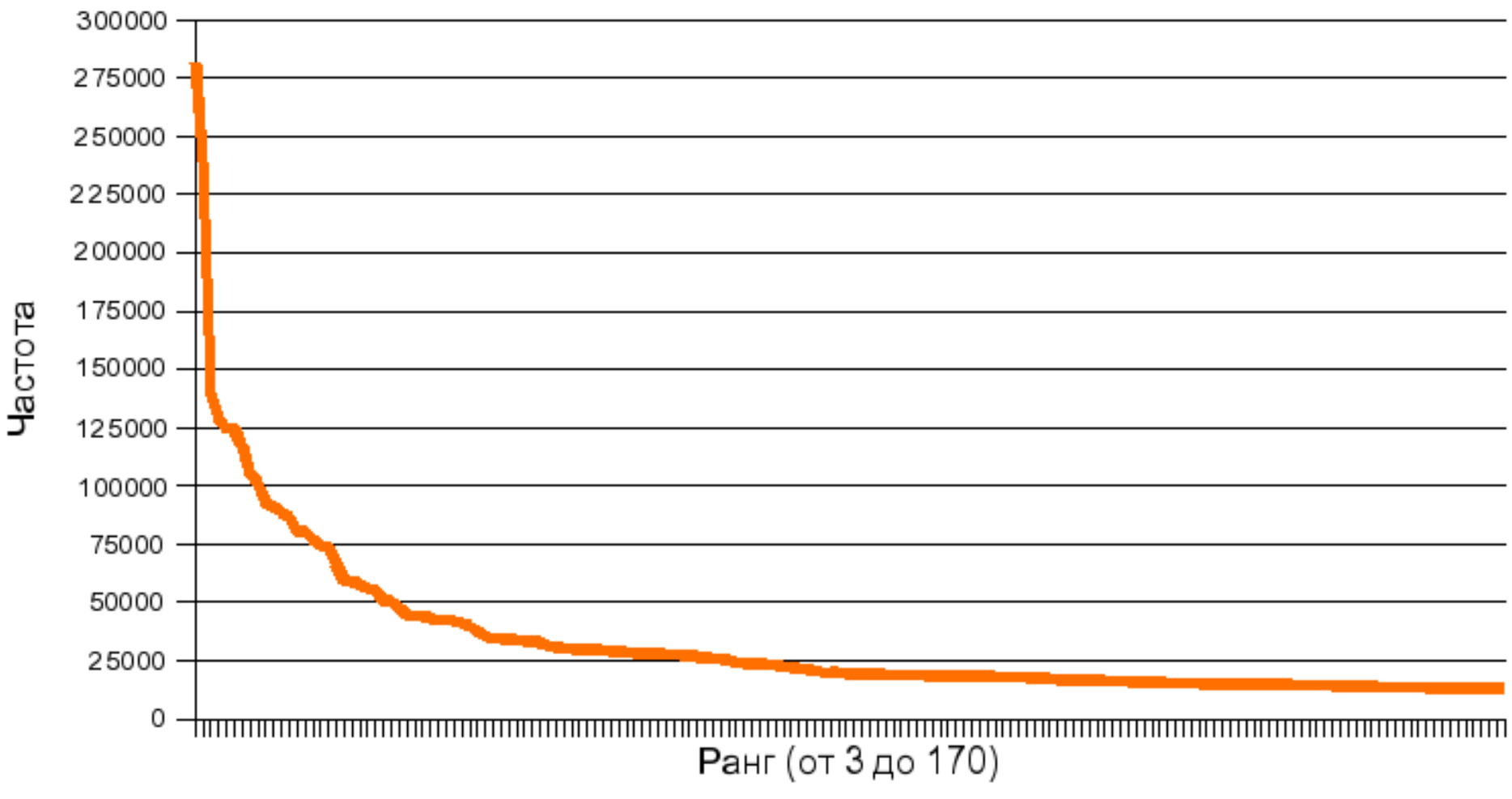
- Поиск терминов необходимо производить внутри предложений
- Как автоматически определять границы предложений?
  - Обычно определяются по точке
  - Точка - имеет много значений
    - граница предложения
    - сокращение: “Dr.”, “U.S.A.”
    - Разделитель в числах 3.14
    - …

Определение границ предложений

- Необходимы алгоритмы разрешения многозначности точки
- Задача сводится к классификации точки на два класса: конец предложения или нет
- Например, можно написать список правил
  - перед точкой и после нее стоят цифры
  - слово перед точкой есть в словаре сокращений
- Правил может быть много и хочется выводить их комбинировать автоматически
- Используется машинное обучение

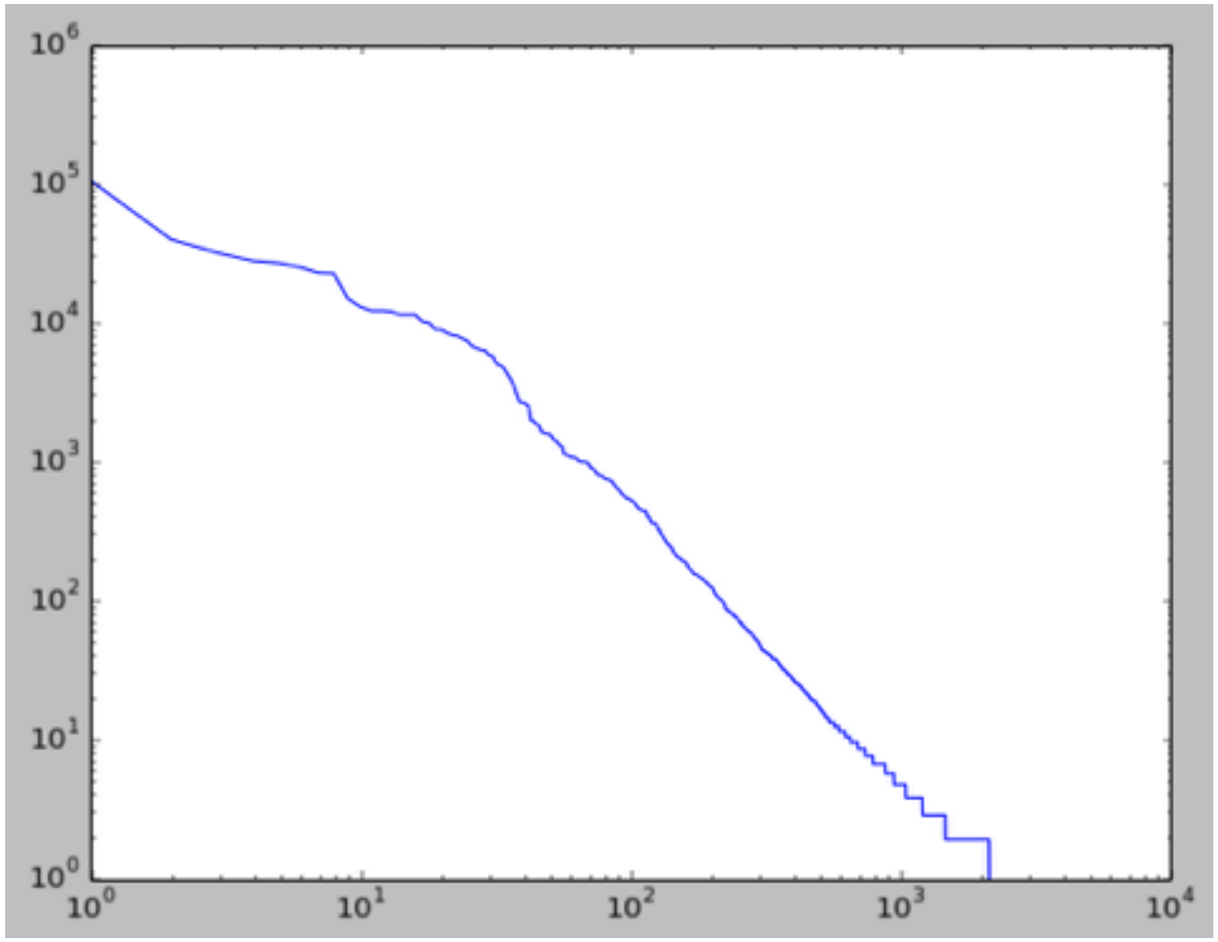
Закон Ципфа

Закон Ципфа — эмпирическая закономерность распределения частоты слов естественного языка: если все слова языка (или просто достаточно длинного текста) упорядочить по убыванию частоты их использования, то частота *n*-го слова в таком списке окажется приблизительно обратно пропорциональной его порядковому номеру *n*. *(Википедия)*



Закон Ципфа

- Распределение частоты слов в первом томе “Война и мир” (логарифмическая шкала)




```
from urllib import urlopen
from bs4 import BeautifulSoup as bs
from nltk import WordPunctTokenizer
from collections import Counter
import matplotlib.pyplot as plt

f = urlopen("http://az.lib.ru/t/
tolstoj_lew_nikolaewich/text_0040.shtml")
data = f.read().decode("cp1251")
f.close()

text = bs(data).get_text()
tokens = WordPunctTokenizer().tokenize(data)
cnt = Counter(tokens).most_common()

# draw plot
X = range(len(cnt))
Y = [y[1] for y in cnt]
plt.loglog(X, Y)
plt.show()
```

Стоп-слова

- Во многих задачах использование наиболее частотных слов создает шум
- Например, при полнотекстовом поиске, система может вернуть почти все документы, если в запросе были предлоги
- Поэтому часто наиболее частотные слова фильтруют и не используют при анализе

	большой	бы
быть	в	весь
вот	все	всей
вы	говорить	год
да	для	до
еще	же	знать
и	из	к
как	который	мочь
мы	на	наш
не	него	нее
нет	них	но
о	один	она
они	оно	оный
от	ото	по
с	свой	себя
сказать	та	такой
только	тот	ты
у	что	это
этот	я	

http://www.artlebedev.ru/everything/yandex/site/saved/stopword.html

Резюме

- Изучены регулярные выражения
  - –регулярные выражения - мощный инструмент для обработки текстов
  - –любое регулярное выражение может быть реализовано с помощью КА (кроме памяти)
  - –автомат неявно определяет формальный язык
  - –для любого НКА существует ДКА
- Рассмотрены базовые задачи обработки текстов
  - –Токенизация и сегментация
  - –Стемминг и лемматизация
  - –Определение границ предложений
  - –Фильтрация стоп-слов
- Для представления результатов работы алгоритмов удобно использовать аннотации

Задания для тренировки

- Написать аналог ELIZA
- Реализовать конечный автомат для распознавания всех русских числительных
- Спроектировать КА для дат: *March 12, the 22nd of November, Christmas*
- Расширить предыдущий автомат относительными датами: *yesterday, tomorrow, a week from tomorrow, the day before yesterday, three weeks from Saturday, next Monday, ...*

Следующая лекция

- Языковые модели
- Задача определения частей речи слов