

**МИНОБРНАУКИ РОССИИ**

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

|  |
| --- |
| **РТУ МИРЭА** |
|  |
| **Институт кибербезопасности и цифровых технологий (ИКБ)** |
|  |
| КБ-2 «Информационно-аналитические системы кибербезопасности» |

**ОТЧЕТ О ВЫПОЛНЕНИИ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ №2**

**В РАМКАХ ДИСЦИПЛИНЫ «АВТОМАТИЗИРРОВАННЫЕ ТЕХНОЛОГИИ ОБРАБОТКИ БОЛЬШИХ ИФНОРМАЦИОННЫХ ПОТОКОВ»**

Выполнил:

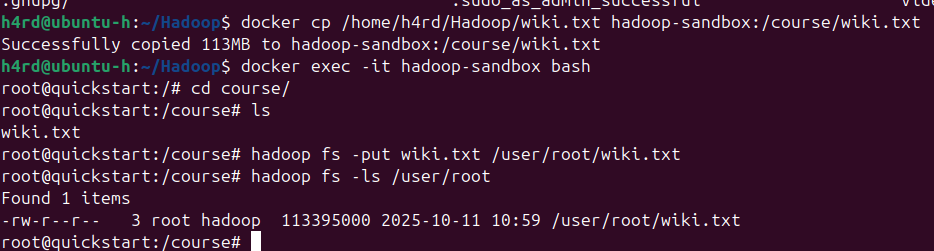
Студент 4-ого курса

Учебной группы БИСО-02-22

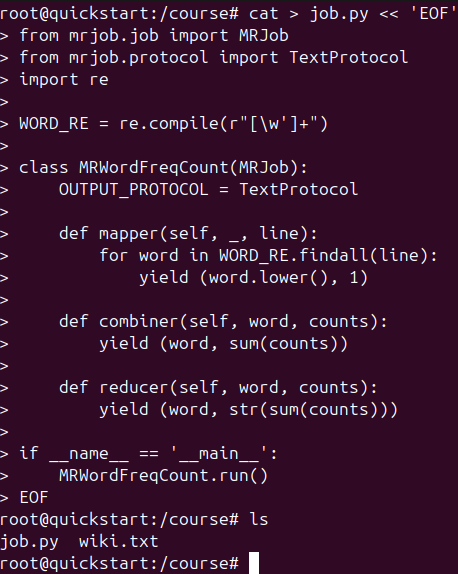
Зубарев В.С.

Выполнение практических заданий

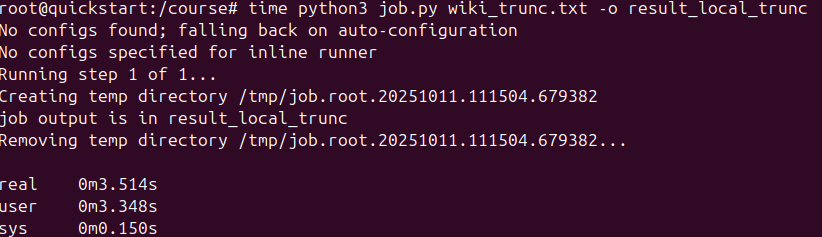
Загрузка файла wiki.txt в HDFS



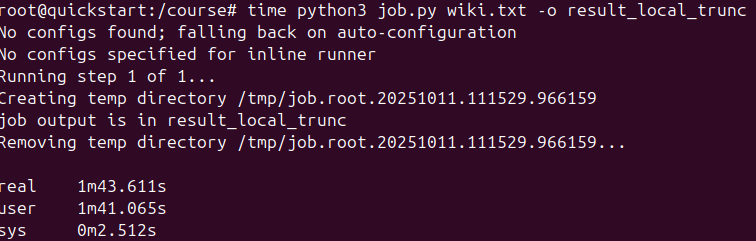
Создание файла job.py



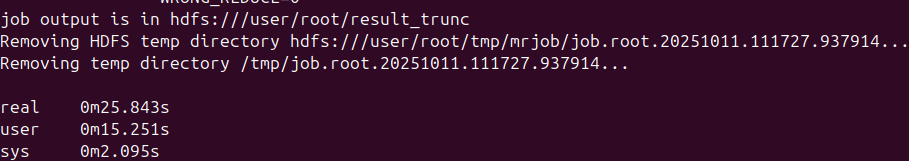
Время для усеченного файла



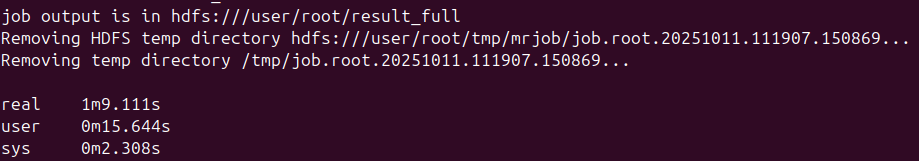
Время для полного файла



Время для усеченного файла через HDFS



Время для полного файла через HDFS



Выводы по заданию:

Для маленьких объёмов входных данных использование Hadoop не рационально. На больших объемах данных временная выгода от использования растет в зависимости от объема входных данных.

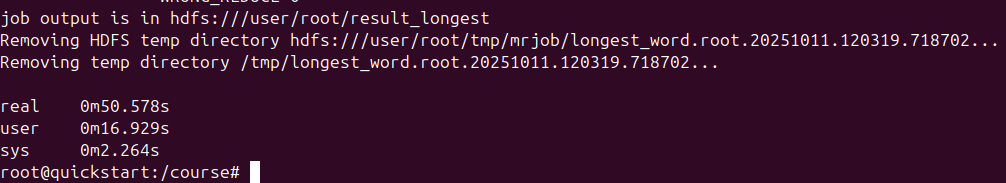
Задание 2

1. Напишите программу, которая находит самое длинное слово.

Код программы для нахождения самого длинного слова представлен ниже.

from mrjob.job import MRJob  
from mrjob.protocol import TextProtocol  
import re  
class MRLongestWord(MRJob):  
 OUTPUT\_PROTOCOL = TextProtocol  
 def mapper(self, \_, line):  
 for word in WORD\_RE.findall(line):  
 if len(word) > 1:  
 yield ("longest", word)  
 def combiner(self, key, words):  
 longest\_word = ""  
 for word in words:  
 if len(word) > len(longest\_word):  
 longest\_word = word  
 yield (key, longest\_word)  
 def reducer(self, key, words):  
 longest\_word = ""  
 for word in words:  
 if len(word) > len(longest\_word):  
 longest\_word = word  
 yield ("Longest word", longest\_word)  
WORD\_RE = re.compile(r"[\w']+")  
MRLongestWord.run()

Результат работы программы представлен ниже



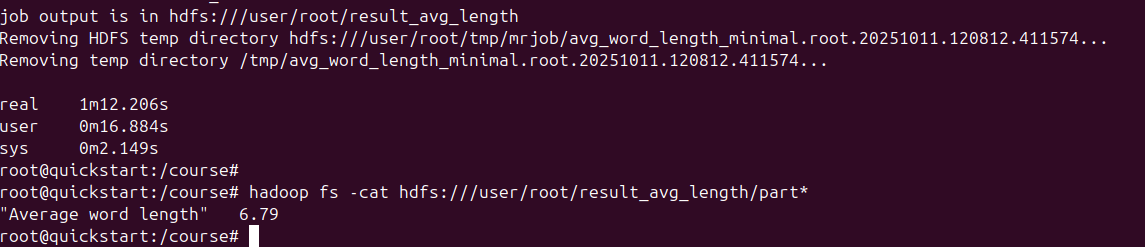


1. Напишите программу, которая находит среднюю длину слов.

Код программы представлен ниже

from mrjob.job import MRJob  
import re  
WORD\_RE = re.compile(r"\w+")  
class AvgWordLength(MRJob):  
 def mapper(self, \_, line):  
 for word in WORD\_RE.findall(line):  
 if len(word) > 1:  
 yield ("stats", (len(word), 1))  
 def reducer(self, key, values):  
 total\_len, total\_count = 0, 0  
 for length, count in values:  
 total\_len += length  
 total\_count += count  
 avg = total\_len / total\_count if total\_count > 0 else 0  
 yield ("Average word length", round(avg, 2))  
AvgWordLength.run()

Результат выполнения



1. Напишите программу, которая находит самое частотопределяемое слово, состоящее из латинских букв.

Код программы представлен ниже

from mrjob.job import MRJob  
import re  
LATIN\_RE = re.compile(r'[a-zA-Z]+')  
class MostFrequentLatin(MRJob):  
 def mapper(self, \_, line):  
 for word in LATIN\_RE.findall(line):  
 yield (word.lower(), 1)  
  
 def reducer(self, word, counts):  
 total = sum(counts)  
 if total > 10:  
 yield (word, total)  
MostFrequentLatin.run()

Результат выполнения программы

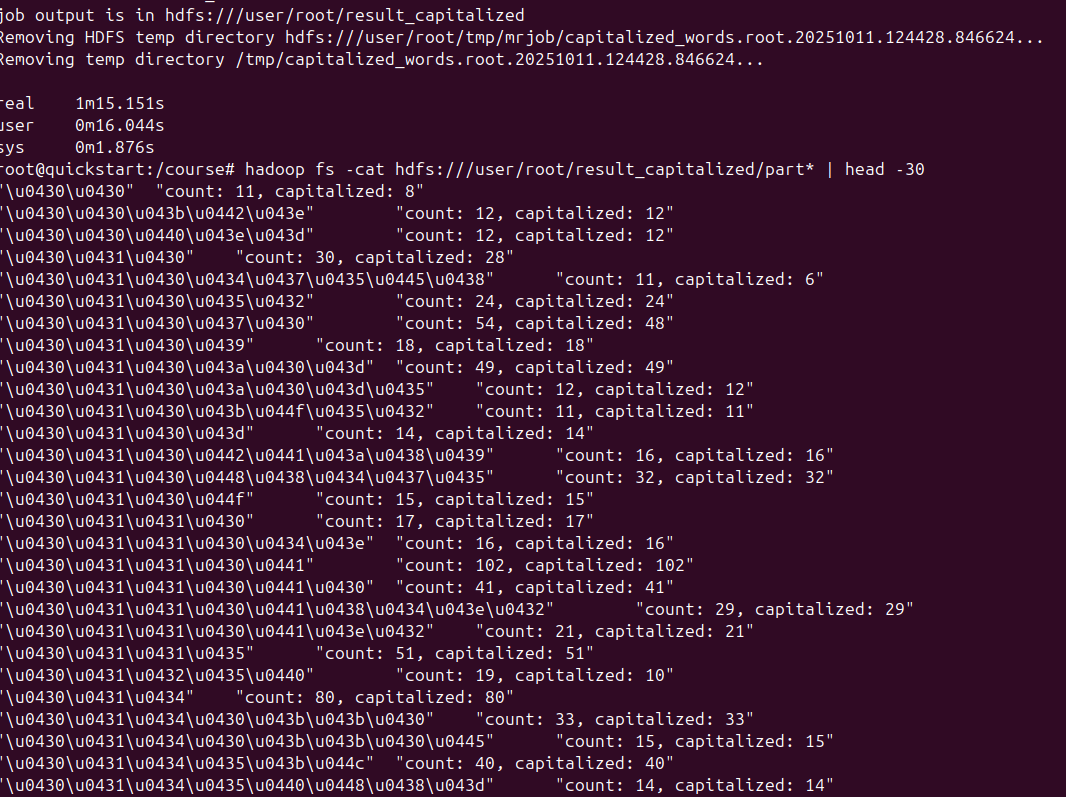


1. Все слова, которые более чем в половине случаев начинаются с большой буквы и встречаются больше 10 раз.

Код программы

from mrjob.job import MRJob  
import re  
class CapitalizedWords(MRJob):  
 def mapper(self, \_, line):  
 try:  
 if isinstance(line, bytes):  
 clean\_line = line.decode('utf-8', errors='ignore')  
 else:  
 clean\_line = line  
 words = re.findall(r'[a-zA-Zа-яА-ЯёЁ]+', clean\_line)  
 for word in words:  
 if len(word) > 1:  
 is\_capitalized = 1 if word[0].isupper() else 0  
 yield (word.lower(), (is\_capitalized, 1))  
 except Exception as e:  
 pass  
 def reducer(self, word, counts):  
 total\_capitalized = 0  
 total\_count = 0  
 for cap, count in counts:  
 total\_capitalized += cap  
 total\_count += count  
 if total\_count > 10 and total\_capitalized / total\_count > 0.5:  
 yield (word, "count: {}, capitalized: {}".format(total\_count, total\_capitalized))  
CapitalizedWords.run()

Результат работы программы

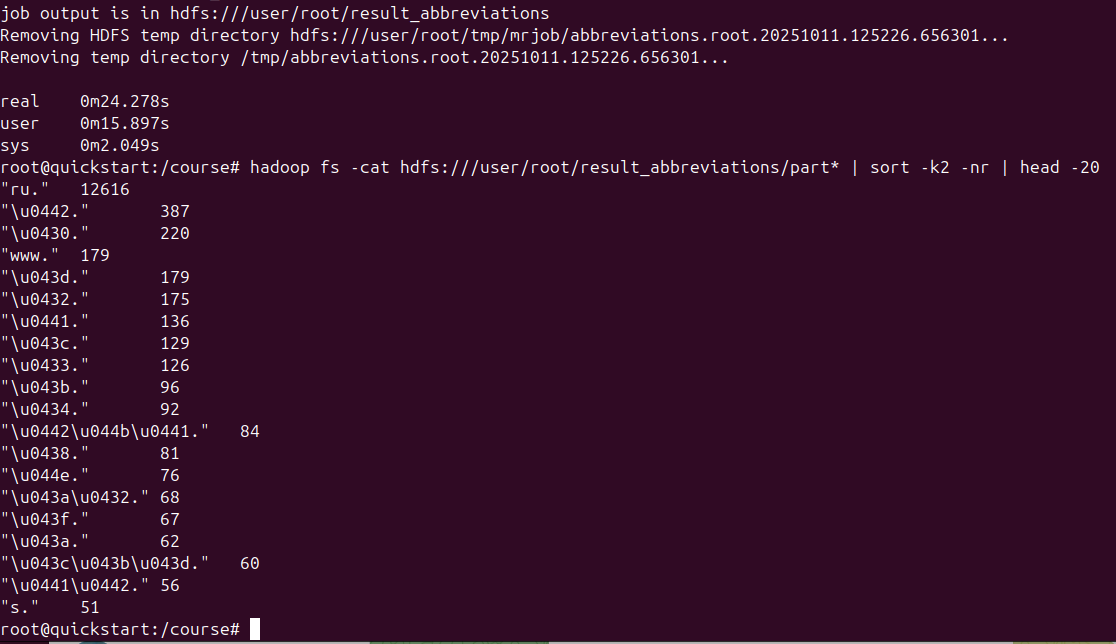


1. Напишите программу, которая с помощью статистики определяет устойчивые сокращения вида «пр.», «др.».

Код программы

from mrjob.job import MRJob  
import re  
class Abbreviations(MRJob):  
 def mapper(self, \_, line):  
 try:  
 if isinstance(line, bytes):  
 clean\_line = line.decode('utf-8', errors='ignore')  
 else:  
 clean\_line = line  
 abbreviations = re.findall(r'\b[a-zA-Zа-яА-ЯёЁ]{1,3}\.\b', clean\_line)  
 for abbrev in abbreviations:  
 yield (abbrev.lower(), 1)  
 except Exception:  
 pass  
 def reducer(self, abbrev, counts):  
 total = sum(counts)  
 if total > 5:  
 yield (abbrev, total)  
Abbreviations.run()

Результат выполнения

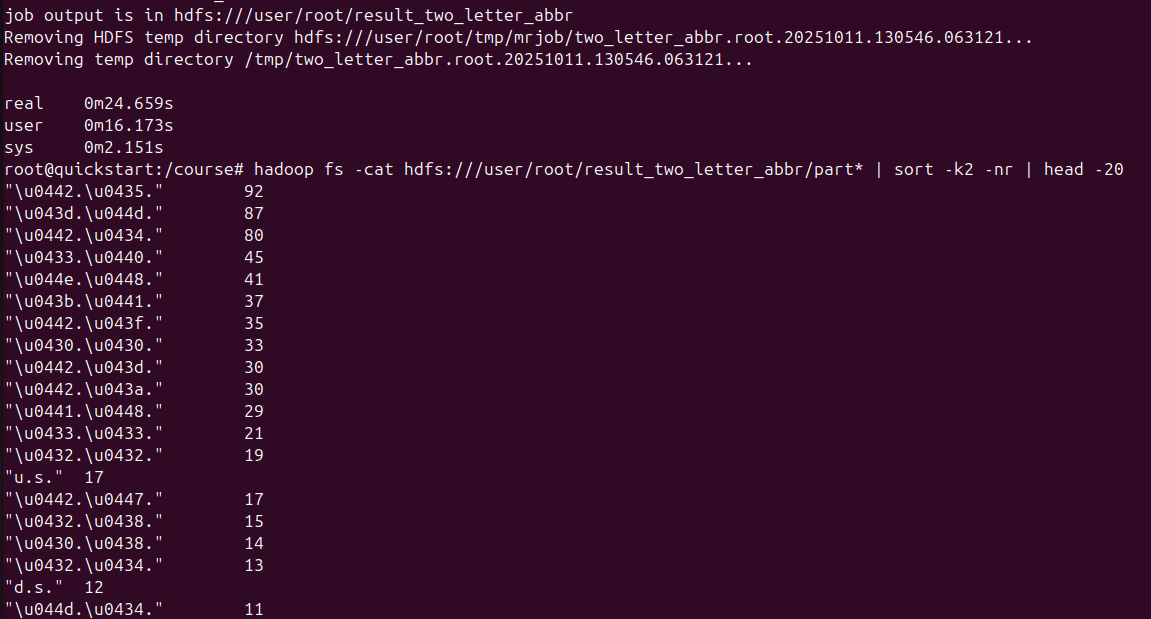


1. Напишите программу, которая с помощью статистики определяет устойчивые сокращения вида «т.п.», «н.э.»,

Код программы

from mrjob.job import MRJob  
import re  
class TwoLetterAbbreviations(MRJob):  
 def mapper(self, \_, line):  
 try:  
 pattern = r'\b[a-zA-Zа-яА-Я]\.{1}[a-zA-Zа-яА-Я]\.{1}'  
 abbreviations = re.findall(pattern, line)  
  
 for abbr in abbreviations:  
 yield (abbr.lower(), 1)  
 except Exception:  
 pass  
 def combiner(self, abbr, counts):  
 yield (abbr, sum(counts))  
  
 def reducer(self, abbr, counts):  
 total = sum(counts)  
 if total > 10:  
 yield (abbr, total)  
TwoLetterAbbreviations.run()

Результат работы



Задание 3

Есть большой набор данных с текстом, нужно подсчитать статистику взаимо встречаемости слов в одном предложении. На вход подаются тексты, на выходе - пары вида ((apple, eat), 42). Это означает, что слово «apple» встретилось со словом «eat» в 42 предложениях. Подумайте, как можно оптимально реализовать подобный подсчет с помощью Надоор. Какие могут быть подводные камни и варианты реализации.

Пример реализации

1. Разбиение текста на предложения (учитывая сокращения инициалов и слов.)
2. Сгенерировать пары слов путем игнорирования союзов предлогов и других вспомогательных частей речи.
3. Создать список часто встречающихся слов и не добавлять в массив анализ слова не входящие в данный список
4. Проанализировать полученный массивы