

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Информационной безопасности

ОТЧЕТ
по лабораторной работе № 2
по дисциплине «Распределенные системы обработки данных»
Тема: Реализация алгоритма MOSS с использованием PySpark для
обнаружения плагиата в исходном коде

Студентка гр. 1361

Токарева У.В.

Преподаватель

Троценко В.В.

Санкт-Петербург

2024

Цель работы.

Реализовать алгоритм MOSS с использованием PySpark для обнаружения плагиата в исходном коде.

Основные теоретические положения.

MOSS был разработан в 1994 году профессором Стэнфордского университета Алексом Аикеном (Alex Aiken). Основная цель создания MOSS — помочь преподавателям и учебным заведениям выявлять случаи плагиата в программных кодах студентов.

MOSS состоит из нескольких этапов:

- **Предобработка кода.** Цель предобработки — свести к минимуму незначительные изменения в коде (переименование переменных, форматирование), чтобы сравнивать программы по логике, а не по внешнему виду.
- **Разбиение кодов на последовательности токенов.** Исходный код программы рассматривается как строка символов. MOSS разбивает его на перекрывающиеся подстроки фиксированной длины n (например, 5 символов).
- **Хеширование последовательностей.** Выбор хэш-функции важен для эффективного и быстрого сравнения программ. Основные критерии: скорость, коллизии.
- **Поиск общих хешей.** Для каждой подстроки программы вычисляются хэши. Сравнивать обычно все хэши между программами неэффективно: слишком много данных для анализа. Для уменьшения их количества используют Winnowing - выбирают только представительные хэши (минимальные в окне), уменьшая объем сравниваемых данных без потери точности. Это позволяет игнорировать малозначительные изменения кода и ускоряет сравнение. В теории даже при изменении порядка или добавлении небольших фрагментов кода, минимальные хэши остаются стабильными.
- **Вычисление коэффициента сходства.**

UDF (User-Defined Function) — это функция, определённая пользователем, которая может быть использована для обработки данных в Spark SQL и DataFrame API. UDF позволяет расширить возможности стандартных функций Spark, добавив кастомную логику для обработки данных. Используется для обработки данных, которые нельзя легко выразить с помощью встроенных функций Spark.

Ход работы.

1. Предобработка кода.

Предобработка кода заключается в удалении пробелов в исходном коде. Предобработка данных производится следующим образом:

```
cleaned_df = raw_df.withColumn('content',  
clean_data_udf(col('content'))).
```

Далее к входным данным применяется udf:

```
clean_data_udf = udf(clean_data, StringType()).
```

Данный код вызывает функцию:

```
def clean_data(inp: str) -> str:  
    return "".join(inp.split()).
```

Очистка данных строится таким образом, потому что необходимо применение функции, которая была описана самостоятельно. Для вызова функций, описанных пользователем, используется udf.

2. Разбиение кода на последовательности токенов.

Необходимо разбиение на подстроки одинаковой длины. Разбиение на подстроки:

```
substrings_df = cleaned_df.withColumn(  
    "substrings",  
    array_distinct(split_bytes_udf(lit(SUBSTRING_LENGTH),  
col('content')))  
) .withColumn("size", array_size(col("substrings"))).
```

Тут к каждому значению применяется `split_bytes_udf` для деления на фиксированные отрезки длины `SUBSTRING_LENGTH`, которая задается отдельно.

Далее `split_bytes_udf` вызывает пользовательскую функцию:

```
split_bytes_udf = udf(lambda n, data: split_str(n, data), ArrayType(StringType())).
```

Функция `split_str`:

```
def split_str(n: int, data: str) -> list:
    return [data[i:n + i] for i in range(0, len(data) - n + 1)] if n <= len(data) else [data]
```

Функция возвращает список подстрок фиксированной длины. Если строка меньше заданного значения, то она возвращается целиком. Эта функция необходима для получения перекрывающихся подстрок фиксированной длины, в нашем случае это значение равно семи.

3. Хеширование последовательностей

Для хеширования в данной работе был использован алгоритм MD5, так как он отлично подходит для надежного сравнения. Сначала описываем `udf` хеширования:

```
md5_udf = udf (lambda x: md5(x.encode()).hexdigest(),
StringType()).
```

Далее применяем хеширование к полученным ранее подстрокам:

```
exploded_df = substrings_df.select("author",
explode(col("substrings")).alias("substring"))
hashed_df = exploded_df.withColumn("hash",
md5_udf(col("substring"))).
```

На рисунке 1 представлены хеши, полученные для подстрок.

toktalk@LAPTOP-I92UI0JP: ~/leti-spark-course-2024

author	substring	hash
template	frompys	883cc45190d0a9c1b...
template	rompysp	f04dddbff3660a998...
template	ompyspa	a540449912d5f2fd4...
template	mpyspar	1835a228d2d764391...
template	pyspark	77bb59dcd89559748...
template	yspark.	c40a00784f60c6165...
template	spark.s	48c7b36403ff3bf5d...
template	park.sh	033b2919aef833786...
template	ark.she	5d811a35ac2165d83...
template	rk.shel	02d4c14499eb871ad...
template	k.shell	77dad670f20b79d8d...
template	.shelli	33e024ee5f9d8ef1f...
template	shellim	869557ed67c219a40...
template	hellimp	ceb738a9b9d85e5a7...
template	ellimpo	0459078f37b88e1b6...
template	llimpor	ccdd81f18151bb7e3...
template	limport	e56e3bcab8f1c86f9...
template	imports	a6878bf5d85836764...
template	mportsp	0a2b54dab8d818d76...

Рисунок 1 – Хеши, полученные для подстрок.

4. Поиск общих хешей

Необходимо найти все различные пары авторов, которые имеют одинаковый хеш. Данный алгоритм выполняет следующий код:

```
comparison_df = hashed_df.alias("lhs").join(
    hashed_df.alias("rhs"),
    (col("lhs.author") != col("rhs.author")) &
    (col("lhs.hash") == col("rhs.hash")),
    how="inner"
).select(
    col("lhs.author").alias("lhs_author"),
    col("rhs.author").alias("rhs_author"),
    col("lhs.hash")
).
```

В результате на рисунке 2 представлены пары авторов, имеющие совпадающий хеши.

toktalk@LAPTOP-I92UI0JP: ~/leti-spark-course-2024

lhs_author	rhs_author	hash
template	9090_ivanovii	ceb738a9b9d85e5a7...
template	9090_ivanovii	fd0ebfe96f3f9d514...
template	9090_ivanovii	1a3fdfb6729f72f7a...
template	9090_ivanovii	a75ffe6c07ee382ab...
9090_ivanovii	template	ceb738a9b9d85e5a7...
9090_ivanovii	template	fd0ebfe96f3f9d514...
9090_ivanovii	template	1a3fdfb6729f72f7a...
9090_ivanovii	template	a75ffe6c07ee382ab...
template	9091_ivanovii	c40a00784f60c6165...
template	9090_ivanovii	c40a00784f60c6165...
template	almost_none	c40a00784f60c6165...
template	9090_ivanovii	23130a4f1eaeed904...
template	9090_ivanovii	9c28d32df23403777...
template	9090_ivanovii	fc2bdaa63995632af...
almost_none	9091_ivanovii	c40a00784f60c6165...
almost_none	9090_ivanovii	c40a00784f60c6165...
almost_none	template	c40a00784f60c6165...
9090_ivanovii	9091_ivanovii	c40a00784f60c6165...
9090_ivanovii	almost_none	c40a00784f60c6165...
9090_ivanovii	template	c40a00784f60c6165...

Рисунок 2 – Пары авторов, имеющие совпадающие хеши.

Далее необходимо вычислить количество совпадений для каждой пары:

```
grouped_df = comparison_df.groupBy("lhs_author",
"rhs_author").agg(count("hash").alias("match_count")).
```

Количество совпадающих хешей для различных авторов представлено на рисунке 3.

toktalk@LAPTOP-I92UI0JP: ~/leti-spark-course-2024

24/11/26 04:47:53 WARN NativeCodeLoader: Unable

lhs_author	rhs_author	match_count
9090_ivanovii	almost_none	24
template	almost_none	24
almost_none	template	24
template	9090_ivanovii	937
template	9091_ivanovii	109
almost_none	9091_ivanovii	25
9091_ivanovii	template	109
9091_ivanovii	almost_none	25
9090_ivanovii	template	937
almost_none	9090_ivanovii	24
none	almost_none	32
almost_none	none	32
9091_ivanovii	9090_ivanovii	150
9090_ivanovii	9091_ivanovii	150

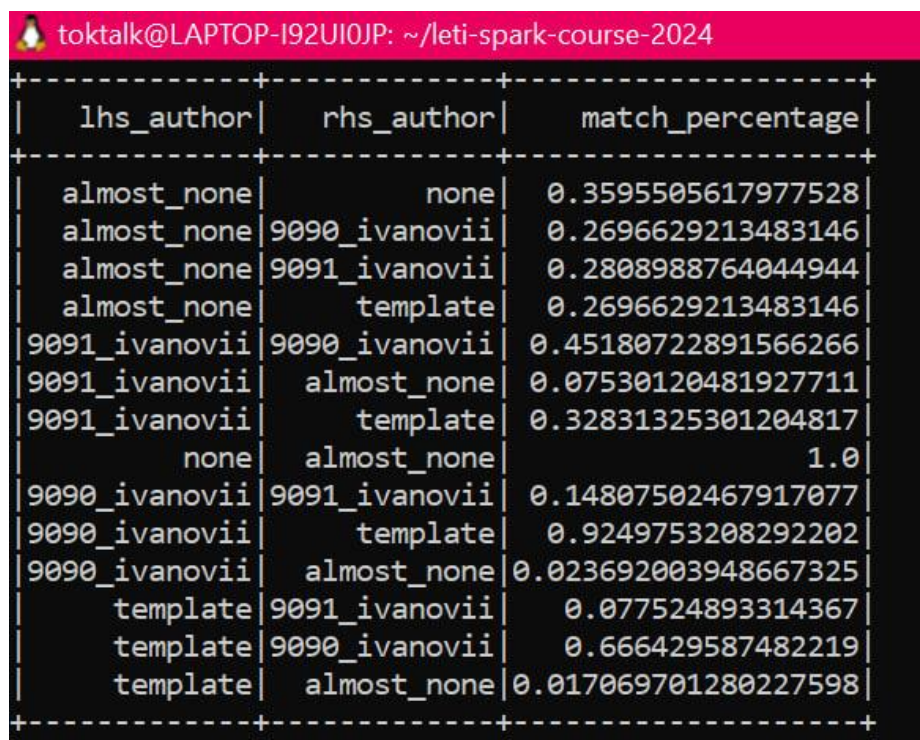
Рисунок 3 – Количество совпадающих хешей для различных авторов.

5. Вычисление коэффициента сходства

Далее необходимо вычислить процентное соотношение совпадений подстрок различных авторов. Эту операцию выполняет следующий код:

```
result_df = grouped_df.join(
    substrings_df.select(col("author").alias("lhs_author"),
        col("size")),
        on="lhs_author"
    ).select(
        "lhs_author", "rhs_author", (col("match_count") /
col("size")).alias("match_percentage")
    )
```

В итоге на рисунке 4 представлен результат сравнения.



lhs_author	rhs_author	match_percentage
almost_none	none	0.3595505617977528
almost_none	9090_ivanovii	0.2696629213483146
almost_none	9091_ivanovii	0.2808988764044944
almost_none	template	0.2696629213483146
9091_ivanovii	9090_ivanovii	0.45180722891566266
9091_ivanovii	almost_none	0.07530120481927711
9091_ivanovii	template	0.32831325301204817
none	almost_none	1.0
9090_ivanovii	9091_ivanovii	0.14807502467917077
9090_ivanovii	template	0.9249753208292202
9090_ivanovii	almost_none	0.023692003948667325
template	9091_ivanovii	0.077524893314367
template	9090_ivanovii	0.666429587482219
template	almost_none	0.017069701280227598

Рисунок 4 – Результат сравнения.

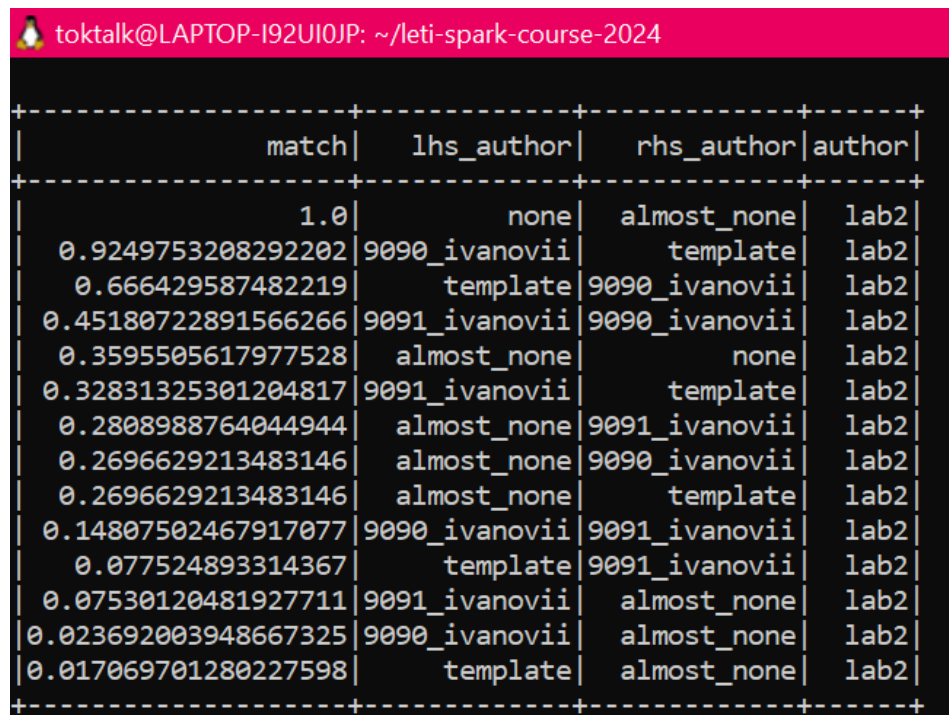
Выводы.

В данной работе с помощью PySpark был реализован алгоритм MOSS. Для хеширования использовалась хеш-функция MD5. В данной работе был

использован udf для использования функций, прописанных отдельно, а также Spark Api для обработки больших данных.

Результат работы программы.

В результате работы была получена таблица, визуализирующая процент плагиата в исходном коде. На рисунке 5 представлен результат работы программы.



match	lhs_author	rhs_author	author
1.0	none	almost_none	lab2
0.9249753208292202	9090_ivanovii	template	lab2
0.666429587482219	template	9090_ivanovii	lab2
0.45180722891566266	9091_ivanovii	9090_ivanovii	lab2
0.3595505617977528	almost_none	none	lab2
0.32831325301204817	9091_ivanovii	template	lab2
0.2808988764044944	almost_none	9091_ivanovii	lab2
0.2696629213483146	almost_none	9090_ivanovii	lab2
0.2696629213483146	almost_none	template	lab2
0.14807502467917077	9090_ivanovii	9091_ivanovii	lab2
0.077524893314367	template	9091_ivanovii	lab2
0.07530120481927711	9091_ivanovii	almost_none	lab2
0.023692003948667325	9090_ivanovii	almost_none	lab2
0.017069701280227598	template	almost_none	lab2

Рисунок 5 – Результат работы программы.

Исходный код программы.

```
# Импорт всех необходимых пакетов
from pyspark.sql import DataFrame
from pyspark.sql.functions import col, lit, udf,
array_size, array_distinct, explode, count, min,
coalesce
from pyspark.sql.types import StringType, ArrayType
from hashlib import md5
from lab2.common import SparkContextCommon

# Константа для деления строк
```



```

SUBSTRING_LENGTH = 7

# Удаление пробелов
def clean_data(inp: str) -> str:
    return "".join(inp.split())

# Разбиение на подстроки
def split_str(n: int, data: str) -> list:
    return [data[i:n + i] for i in range(0, len(data) -
n + 1)] if n <= len(data) else [data]

# Определение udf
clean_data_udf = udf(clean_data, StringType())
split_bytes_udf = udf(lambda n, data: split_str(n,
data), ArrayType(StringType()))
md5_udf = udf (lambda x: md5(x.encode()).hexdigest(),
StringType())

# Основная часть
def solve(common: SparkContextCommon) -> DataFrame:

    # Чтение и очистка данных
    raw_df = common.read_data()
    cleaned_df = raw_df.withColumn('content',
clean_data_udf(col('content')))

    # Разбиение на подстроки
    substrings_df = cleaned_df.withColumn(

```

```

        "substrings",
array_distinct(split_bytes_udf(lit(SUBSTRING_LENGTH),
col('content'))))
    ).withColumn("size", array_size(col("substrings")))

# Хеширование
exploded_df      =      substrings_df.select("author",
explode(col("substrings")).alias("substring"))
hashed_df        =      exploded_df.withColumn("hash",
md5_udf(col("substring")))

# Сравнение
comparison_df = hashed_df.alias("lhs").join(
    hashed_df.alias("rhs"),
    (col("lhs.author") != col("rhs.author")) &
(col("lhs.hash") == col("rhs.hash")),
    how="inner"
).select(
    col("lhs.author").alias("lhs_author"),
    col("rhs.author").alias("rhs_author"),
    col("lhs.hash")
)

# Группировка
grouped_df      =      comparison_df.groupBy("lhs_author",
"rhs_author").agg(count("hash").alias("match_count"))

# Новый датафрейм с совпадениями
result_df = grouped_df.join(

```

```

substrings_df.select(col("author").alias("lhs_author"),
col("size")),
    on="lhs_author"
).select(
    "lhs_author", "rhs_author", (col("match_count")
/ col("size")).alias("match_percentage")
)

# Результат
common.view("result", result_df)

# SQL запрос
return common.spark.sql("""
    SELECT  CAST(match_percentage AS Double) AS
match,
            CAST(lhs_author AS String) AS
lhs_author,
            CAST(rhs_author AS String) AS rhs_author
    FROM result
""")

```