Санкт-Петербургский государственный электротехнический университет им. В.И. Ульянова (Ленина)

Разработка инструмента поиска плагиата в git-репозиториях

Выполнил:
Есиков Олег Игоревич, гр. 7304, oiesikov@stud.etu.ru
Руководитель:
Заславский Марк Маркович, к.т.н., доцент

Санкт-Петербург, 2021

Цель и задачи

Актуальность: ручная проверка на плагиат занимает много времени и не является эффективной

Цель: разработать web-приложение для поиска плагиата в студенческих работах по программированию на языке C в GitHub-репозитории

Задачи:

1. Изучить существующие методы
2. Провести сравнительный анализ методов
3. Разработать web-приложение
4. Исследовать время поиска, занимаемую память

Плагиат исходного кода

- Совпадения Ⅰ типа – полное совпадение двух фрагментов, допускающее добавление пробельных символов и комментариев
- Совпадения Ⅱ типа – Ⅰ тип + изменение названий переменных, функций, классов и т.д.
- Совпадения Ⅲ типа – Ⅱ тип + изменение, добавление и удаление операторов
- Совпадения Ⅳ типа – фрагменты синтаксически написаны по-разному, но выполняют одинаковые вычисления

Существующие методы выявления плагиата в исходном коде программ

Варианты представления исходного кода

Рассмотренные методы:

- Строковые
- Метрические
- Метод отпечатков
- Алгоритм Хескела
- Алгоритм жадного строкового замощения

Сравнительный анализ методов

	Выявление совпадений	**Сложность**	**Результат**
Строковые	Ⅰ тип – не всегда Ⅱ и Ⅲ тип – никогда	$O(n^2)$	совпадающие подстроки
Метрические	Ⅰ тип – всегда Ⅱ и Ⅲ тип – не всегда	O(n)	% сходства
Метод отпечатков	Ⅰ и Ⅱ тип – всегда Ⅲ тип – не всегда	O(n)	% сходства
Алгоритм Хескела	Ⅰ и Ⅱ тип – всегда Ⅲ тип – не всегда	O(n)	% сходства
Жадн. строк. замощение	Ⅰ и Ⅱ тип – всегда Ⅲ тип – не всегда	$O(n^3)$	совпадающие подстроки

Сравнение методов по критериям

Разработка токенизатора для языка C

- Токены выделяются с помощью регулярных выражений
- Особенности:
  - Игнорирование аргументов и параметров
  - Расстановка необязательных фигурных скобок
  - Токенизированное представление switch и тернарного оператора соответствует условным операторам

UML диаграмма классов разработанного web-приложения

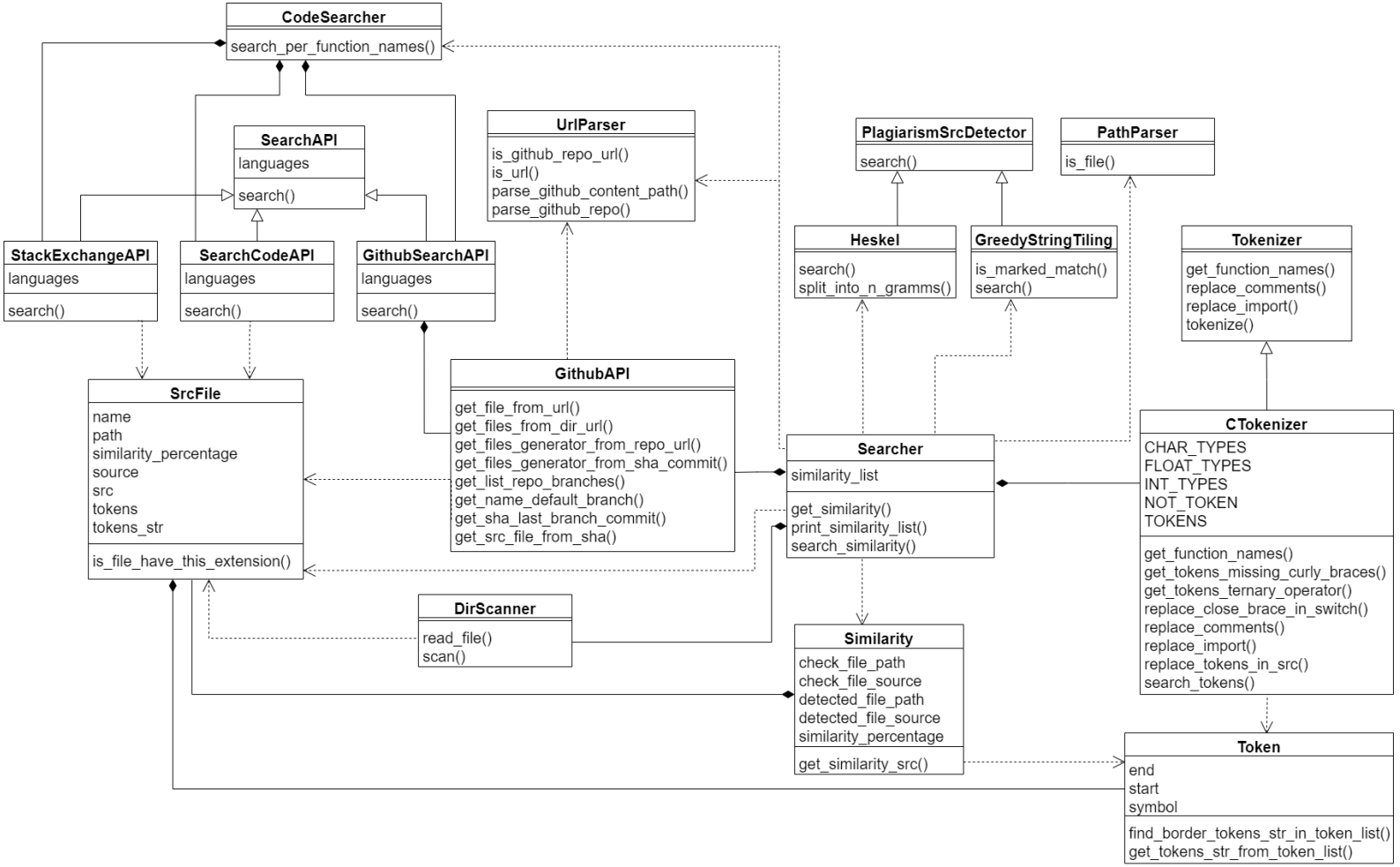
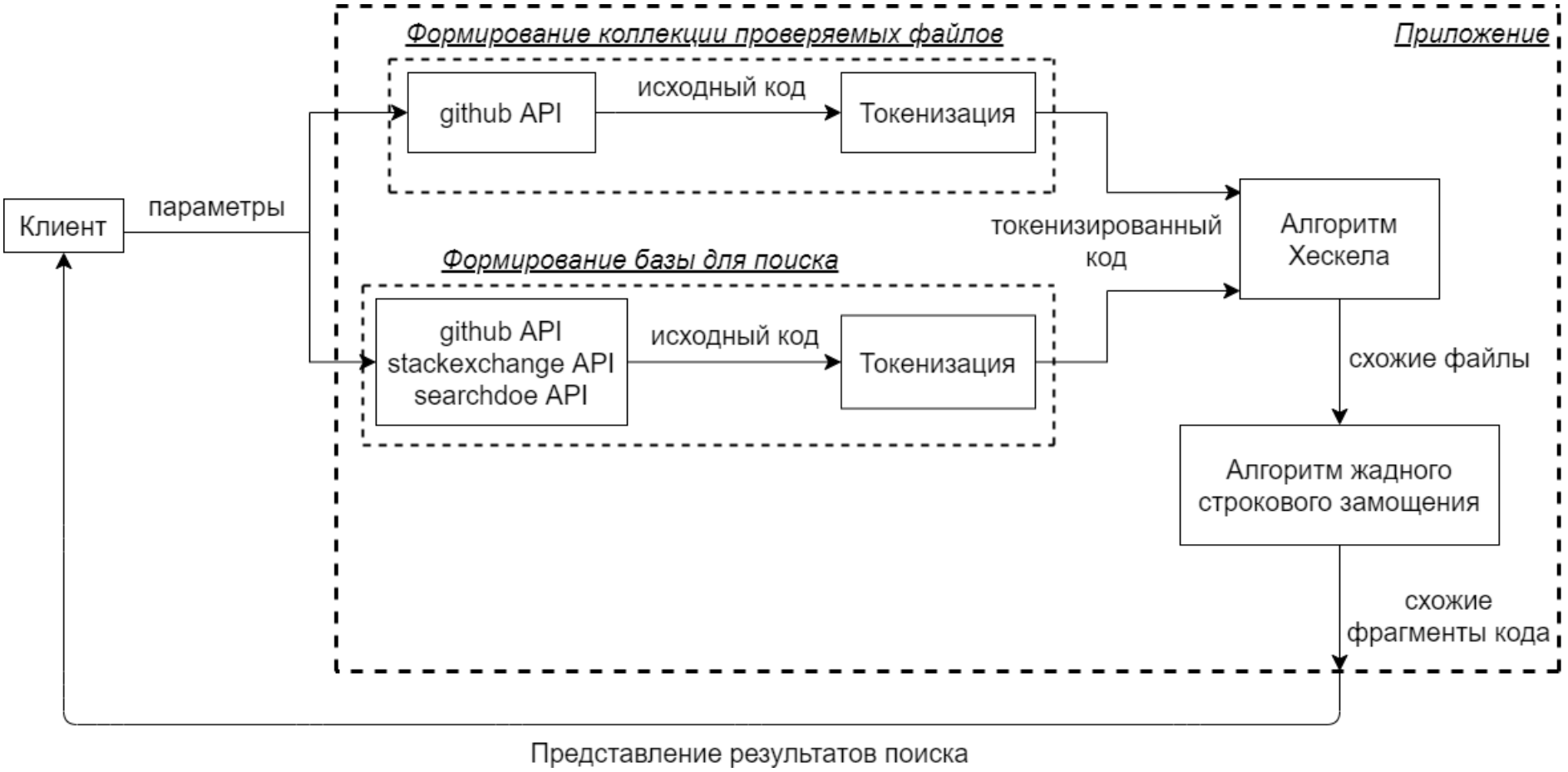
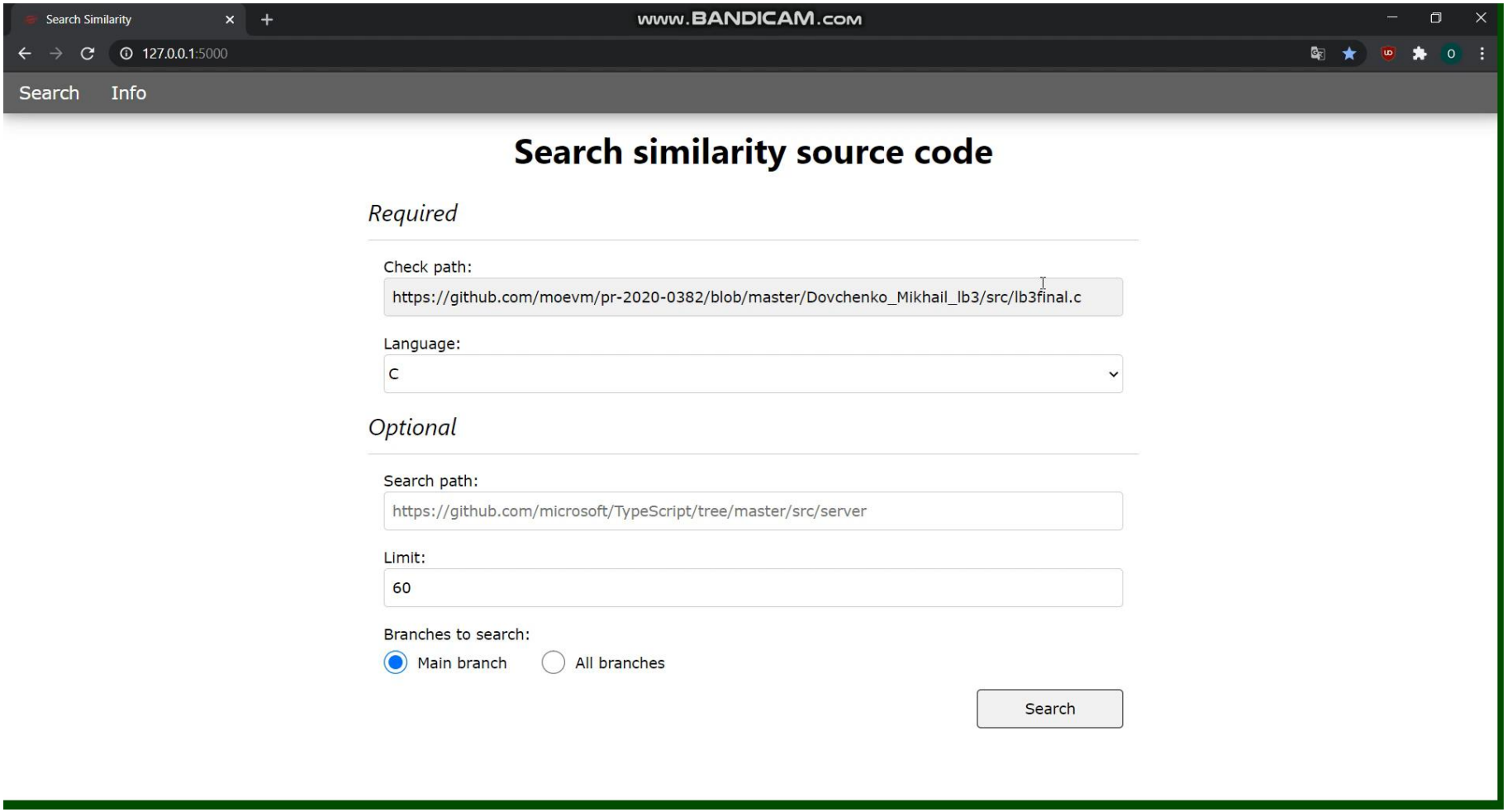


Схема процесса работы web-приложения

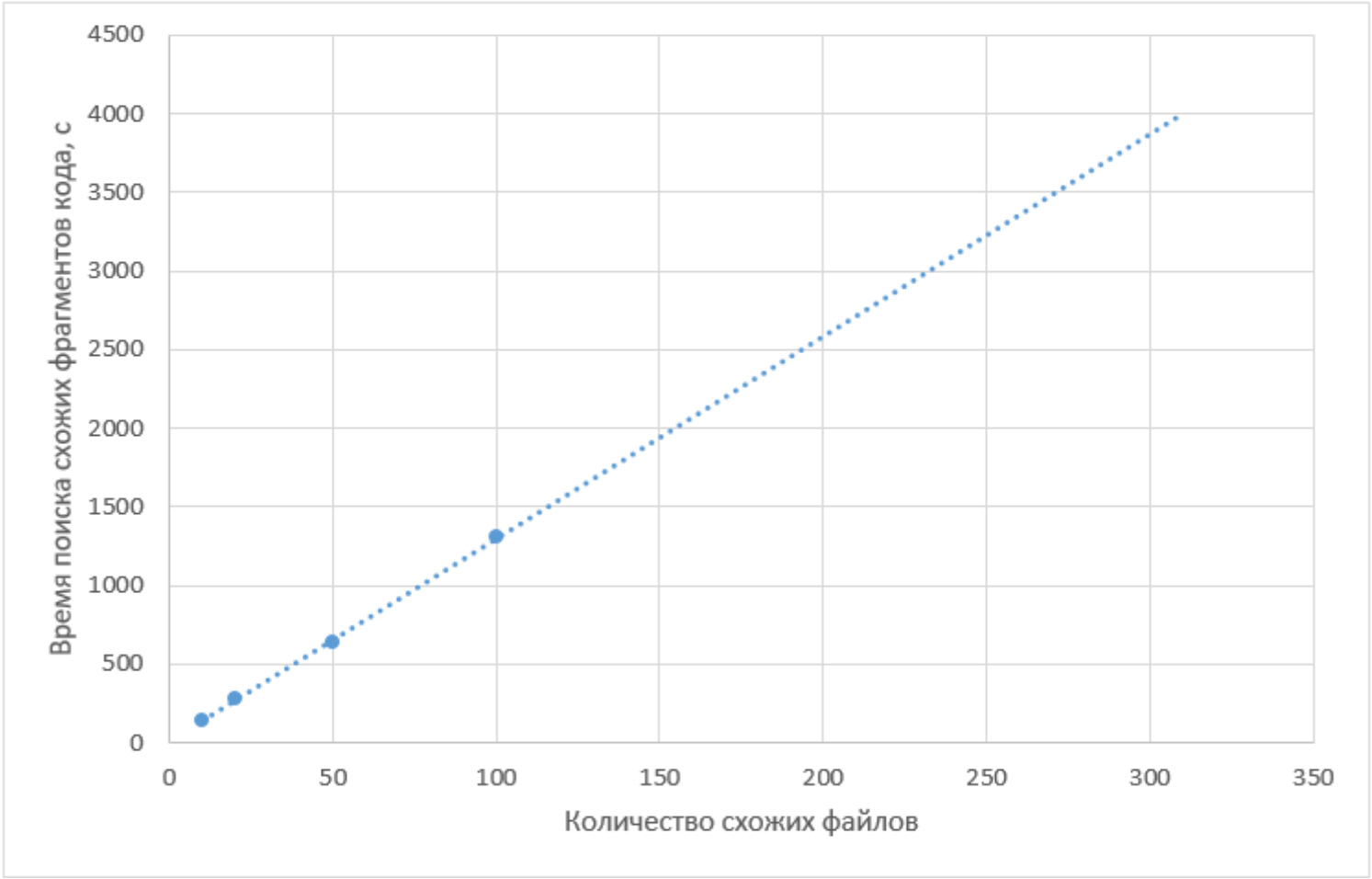


Представление результатов поиска

Демонстрация разработанного web-приложения

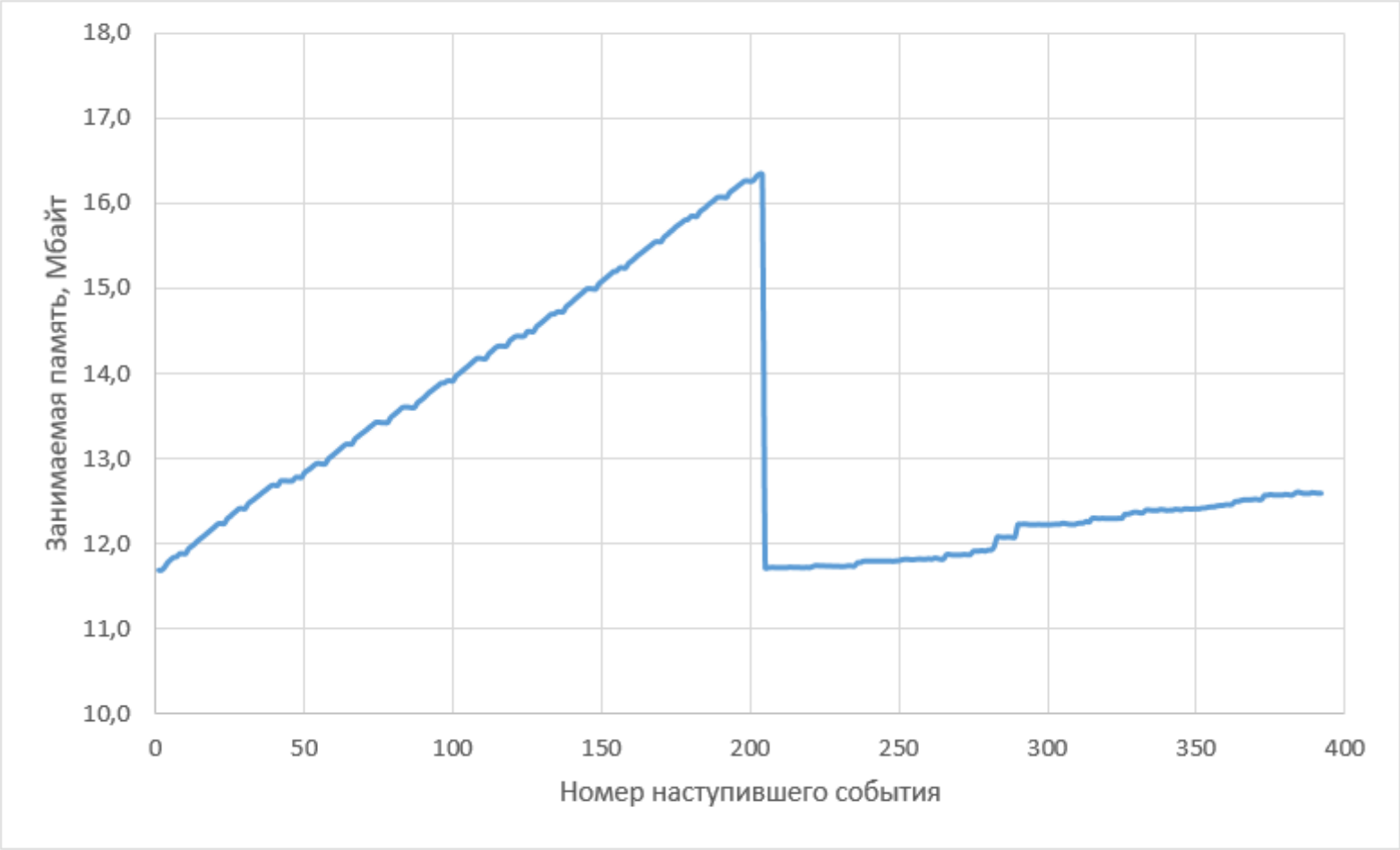


Исследование затрачиваемого времени на поиск



Зависимость времени поиска схожих фрагментов кода от количества схожих файлов

Исследование занимаемой памяти



Вид зависимости занимаемой памяти во время работы разработанного web-приложения

Заключение

- Проделанный обзор существующих методов показал, что необходимо использовать два токенизирующих метода – алгоритм Хескла и алгоритм жадного строкового замощения
- Спроектировано и опубликовано на GitHub web-приложение для поиска плагиата в студенческих работах по программированию на языке C в GitHub-репозитории
- Проведённое исследование показало, что web-приложение выйдет за допустимое время проверки 1 час при наличии в проверяемой коллекции 310 схожих файлов длиной 1000 строк
- При проверке репозиториев с работами студентов максимальное значение занимаемой памяти составило 17,3 Мб
- Дальнейшие направления улучшения web-приложения включают в себя добавление токенизаторов для других языков программирования, а также добавление базы данных проверенных файлов

Апробация работы

- «Разработка инструмента поиска плагиата в git-репозиториях» // Научно-практическая конференция с международным участием «Наука настоящего и будущего», 2021
- Репозиторий web-приложения https://github.com/moevm/bsc_esikov
- 

DEMO

Спасибо за внимание!

oiesikov@stud.etu.ru

Запасные слайды

Полностью совпадающие фрагменты I типа

```
1  #include <stdio.h>
2
3  // My first program
4  int main()
5  {
6      printf("Hello World");
7      return 0;
8  }
```

```
1      #include <stdio.h>
2
3          int
4                          main    ()
5              {
6  printf     ("Hello World")
7  ;
8
9                  return
10 0;}
11 // end
```

Полностью совпадающие фрагменты Ⅱ типа

```c
#include <stdio.h>

int sum(int a, int b)
{
    return a + b;
}

int main()
{
    int result = sum(10, 5);
    return 0;
}
```

```c
#include <stdio.h>

int func(int x, int y) {
    return y + x;
}

int main() {
    int a = func(10, 5);
    return 0;
}
```

Полностью совпадающие фрагменты Ⅲ типа

```c
#include <stdio.h>

int sum(int a, int b) {
    return a + b;
}

int main()
{
    int result = sum(10, 5);
    return 0;
}
```

```c
#include <stdio.h>

int func(int x, int y) {
    return y + x;
}

int main() {
    const int checkSum = 123;
    int a = func(10, 5);
    return 0;
}
```

Полностью совпадающие фрагменты Ⅳ типа

```c
#include <stdio.h>

unsigned long long fact(unsigned int n)
{
    unsigned long long result = 1;
    for(int i = 2; i <= n; i++)
        result *= i;
    return result;
}

int main() {
    unsigned long long result = fact(10);
    printf("%llu", result);
    return 0;
}
```

```c
#include <stdio.h>

unsigned long long fact(unsigned int n )
{
    if (n < 2)
        return 1;
    else
        return n * fact(n - 1);
}

int main() {
    unsigned long long result = fact(10);
    printf("%llu", result);
    return 0;
}
```

Регулярные выражения для выделения токенов

- Определение функции:

```
\w+((\s*\*\s*)+|\s+)\w+\s*\([^{]*\)\s*(?={)
```

- Определение вещественной переменной:

```
(long double|double|float)\s+\w+[\[\]\d:]*
(\s*,\s*\w+[\[\]\d]*\s*)*\s*(?=[;=])
```

- Условные конструкции:

```
\b(if|else\s*if)\s*\([^{;]+?\)\s*
(?=[{\w.])|\belse\b
```

Все используемые токены для языка С

Имя	Токен	Значение	Имя	Токен	Значение
int	N	Целое число	return	R	Возврат из функции
double	D	Дробное число	if	I	Условие
char	B	Байт	cycle	S	Цикл
ptr	P	Указатель	compare	E	Сравнение
call	C	Вызов функции	logic	L	Логический оператор
assign	A	Присваивание	shift	U	Побитовая операция
func	F	Определение функции	control	G	Управляющий оператор
cast	T	Приведение типа	struct	V	Структура
math	M	Математический оператор	{ }	{ }	Фигурные скобки

Токенизация функции

```
#include <stdio.h>

int func(int number) {
    return number + 10;
}
```

Токен	Код
FUNC	int func(int number)
{	{
RETURN	return number + 10;
MATH	number + 10;
}	}

Токенизация цикла

```
#include<stdio.h>

int main()
{
    do
        printf("Hello!");
    while (0 > 1);
    return 0;
}
```

Токен	Код
FUNC	int main()
{	{
CYCLE	do
{	
CALL	printf("Hello!");
}	
RETURN	return 0;
}	}

Токенизация тернарного оператора

1.

```
z = (x > y) ? func1(): func2();
```

2.

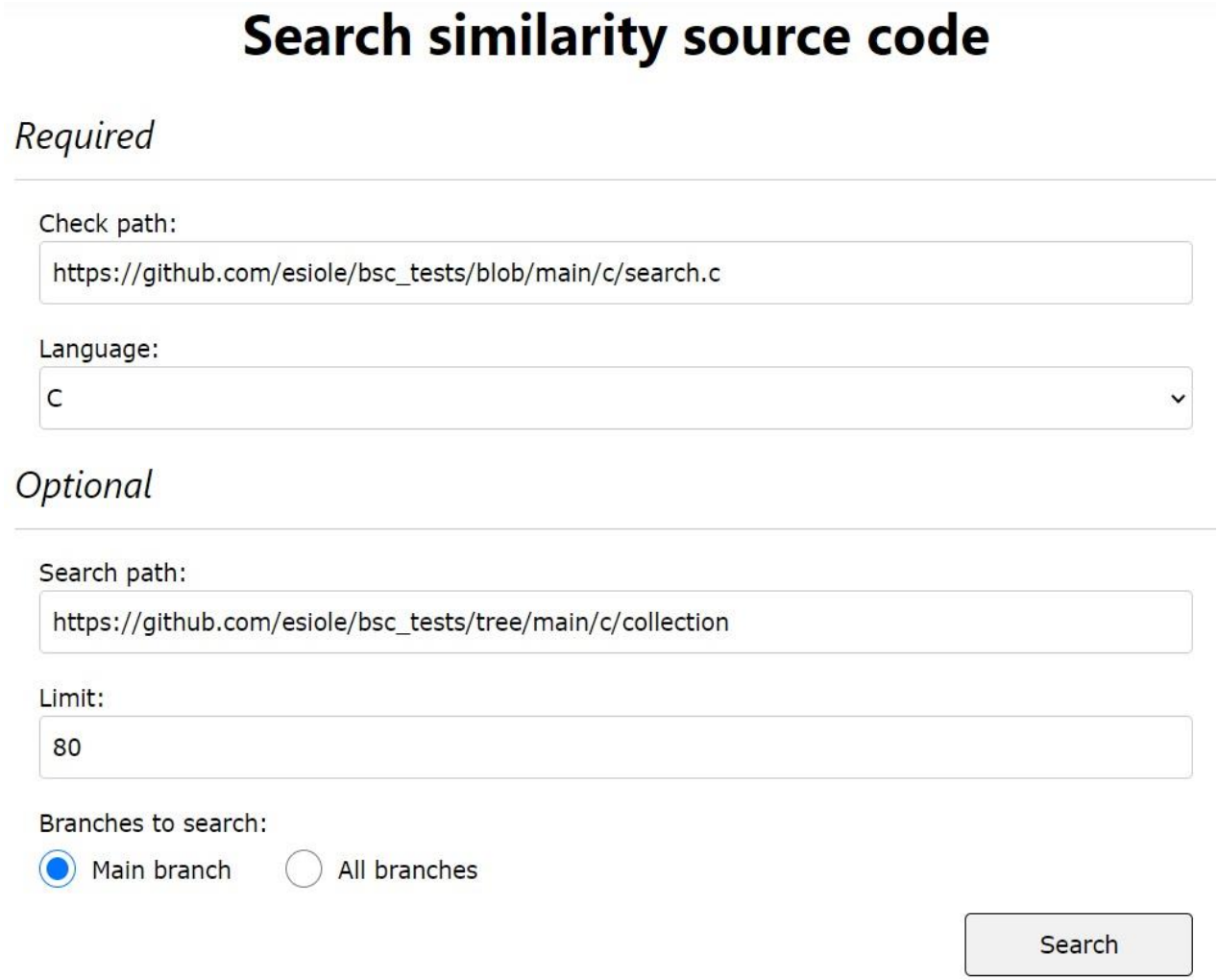
```
if (x > y)
    z = func1();
else
    z = func2();
```

Токен
IF
{
ASSIGN
CALL
}
IF
{
ASSIGN
CALL
}

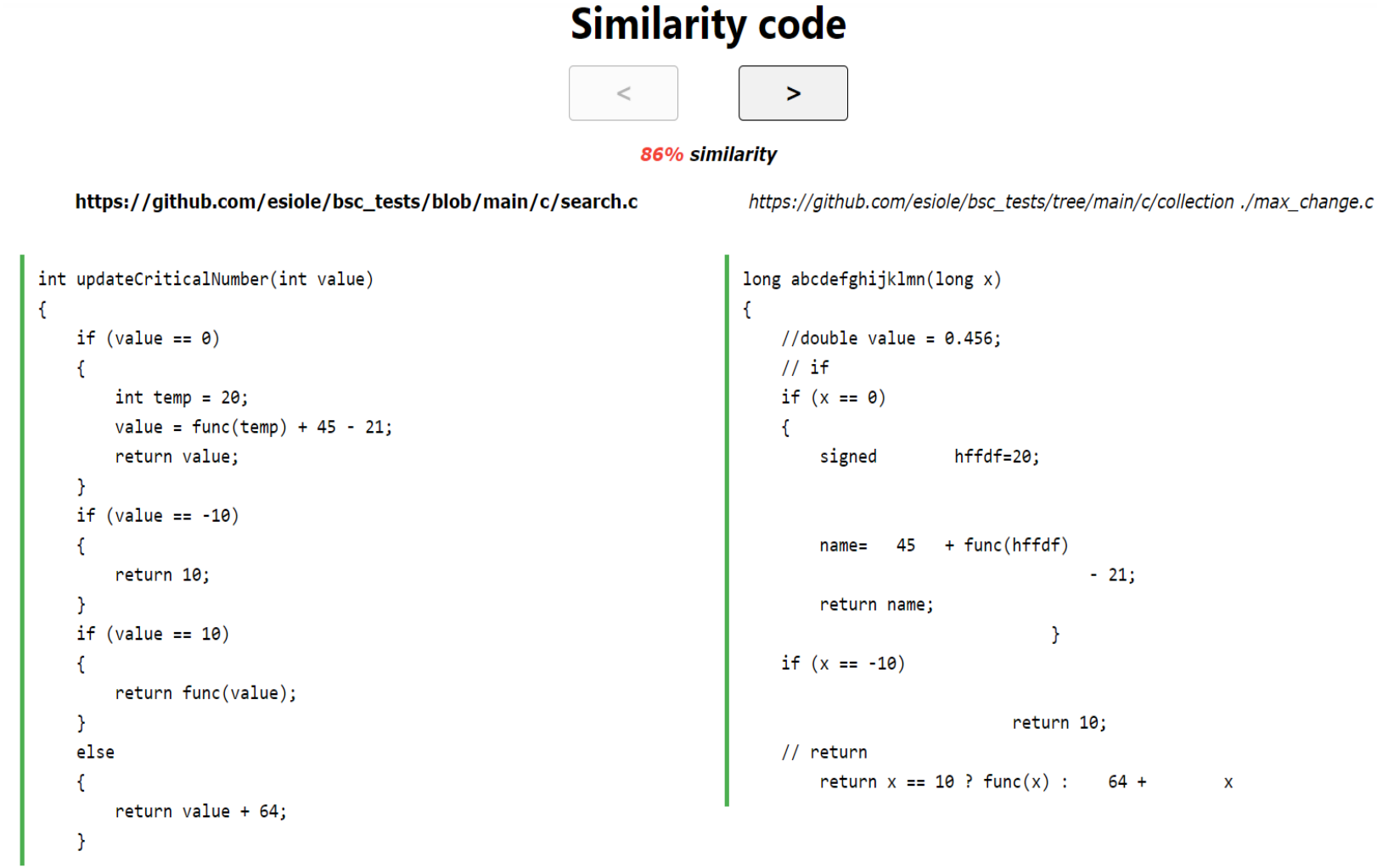
Запросы для взаимодействия с другими API

- GET https://searchcode.com/api/codesearch_I
- GET https://api.stackexchange.com/2.2/search/advanced
- GET https://api.github.com/search/cod
- GET https://api.github.com/repos/{owner}/{repo}
- GET https://api.github.com/repos/{owner}/{repo}/branches
- GET https://api.github.com/repos/{owner}/{repo}/branches/{brn}
- GET https://api.github.com/repos/{owner}/{repo}/contents/{path}
- GET https://api.github.com/repos/{owner}/{repo}/git/blobs/{sha}
- GET https://api.github.com/repos/{owner}/{repo}/git/trees/{sha}

Форма для ввода параметров поиска



Представление результатов поиска



Теоретическая оценка времени работы

$$T = N_c \cdot (T_t + N_d \cdot T_h + N_s \cdot T_g),$$

где T – общее время поиска плагиата

Nc – количество проверяемых файлов

Tt – время токенизации файла

Nd – количество файлов в базе поиска

Th – время поиска схожих файлов

Ns – количество схожих с проверяемым файлом в базе

Tg – время поиска схожих фрагментов исходного кода

Характеристики используемого для исследования устройства

- Система Windows 10 Home 19041.867 x64
- Процессор Intel® Core™ i5-8300H CPU @ 2.30GHz
- Оперативная память 8192 Мб, DDR4, 2133 МГц

Время токенизации файла

Число строк	Время токенизации, с
100	0,020
300	0,036
700	0,050
1000	0,094
2000	0,178
4000	0,415
8000	1,035
10000	1,708

Примерное соответствие между числом строк и количеством символов в файле

Число строк	Количество символов
100	2801
300	11164
700	19219
1000	29768
2000	59431
4000	118757
8000	237409
10000	295484

Время поиска схожих файлов

Количество строк в файлах в базе	Количество файлов в базе	Время поиска схожих файлов, с
100	10	0,0003
100	20	0,0009
100	50	0,0023
100	100	0,0050
1000	10	0,0049
1000	20	0,0083
1000	50	0,0202
1000	100	0,0406
10000	10	0,0639
10000	20	0,0709
10000	50	0,1946
10000	100	0,3596

Время поиска схожих фрагментов исходного кода для файла в 100 строк кода

Количество строк в файлах в базе	Количество схожих файлов	Время поиска схожих файлов, с
100	10	0,2913
100	20	0,5449
100	50	1,3861
100	100	2,7980
1000	10	5,3945
1000	20	10,3691
1000	50	23,3818
1000	100	46,5506

Время поиска схожих фрагментов исходного кода для файла в 1000 строк кода

Количество строк в файлах в базе	Количество схожих файлов	Время поиска схожих файлов, с
100	10	4,1135
100	20	7,2886
100	50	19,4511
100	100	38,0526
1000	10	139,0077
1000	20	275,5242
1000	50	632,2365
1000	100	1302,1430

Результаты проверки репозиториев с работами студентов

Семестр	Количество проверяемых лабораторных	Количество проверенных работ	Количество пар схожих работ		
			87-100% сходства	76-86% сходства	60-76% сходства
Осень 2019	2 + курсовая	415	31	13	29
Весна 2020	2	217	20	6	35
Осень 2020	1 + курсовая	229	48	12	11
Весна 2021	2	145	8	3	2

Количество студентов по семестрам

Семестр	Количество студентов, сдавших хотя бы 1 работу
Осень 2019	161
Весна 2020	130
Осень 2020	142
Весна 2021	109

Обнаруженный схожий код студентов

```
="Dragon flew away!";
p[0]='\0';
while(1)
        {
         for(i=strlen(p);i<n1;i++)
                {
                p[i]=getchar();
                // УБИРАЕМ ПРОБЕЛЫ=================================================
                 if (((p[i]==' ')||(p[i]=='\t')||(p[i]=='\n'))&&((i==0)||(p[i-1]
                        {
                        while ((p[i]==' ')||(p[i]=='\t')||(p[i]=='\n'))
                                        p[i]=getchar();
                        continue;
                        }

                if ((p[i]=='.')||(p[i]==';')||(p[i]=='?')||(p[i]=='!'))
                                break;

                }
                //ПРОВЕРКА НА ПОСЛЕДНЕЕ ПРЕДЛОЖЕНИЕ==============================
                if (p[i]=='!')
                        {
                        for (j=0;j<17;j++)
                                if (ch[j]!=p[(strlen(p)-17)+j])
                                        break;
                        if(j==17)
                                {
                                p[i+1]='\0';
                                break;
                                }
                        }
                else
                        {
                        p[i+1]='\0';
                        n1+=strlen(p);
```

```
=0;//// Flag, counter, sentences
    buffer[0]='\0';
    while(1) {
        for(i=strlen(buffer); i<n; i++) {
            buffer[i]=getchar();// remove the space
            if (     ((buffer[i] == ' ') || (buffer[i] == '\t') || (buffer[i] ==
                     && ((i==0) || (buffer[i-1] == '.') || (buffer[i-1] == ';')
                while ((buffer[i] == ' ') || (buffer[i] == '\t') || (buffer[i] =
                    buffer[i]=getchar();
                continue;
            }

            if ((buffer[i] == '.') || (buffer[i] == ';') || (buffer[i] == '?') |
                break;


        }
        if (buffer[i] == '!') {// check for end
            for (j=0; j<17; j++)
                if (dragon[j] != buffer[(strlen(buffer)-17)+j])
                    break;
            if(j == 17) {
                buffer[i+1]='\0';
                break;
            }
        } else {
            buffer[i+1]='\0';
            n+=strlen(buffer);
            buffer=(char*)realloc(buffer,n*sizeof(char));
        }
    }
    exit=(char*)malloc((strlen(buffer)+1)*sizeof(char));// overwrite

    for(i=0,offer=0,n=0; buffer[i]!='\0'; i++) {
        flag=0;
        if(buffer[i] == '7'
```

Сложность абстрактных синтаксических деревьев

```
int f(int a, int b)
{
    int c;
    c = a + 2;
    print(b, c);
}
```