

# Поиск списываний в контестах по программированию с помощью построения графов зависимостей программ

Анисимова Карина Витальевна

Научный руководитель: А.В. Садовников

Санкт-Петербургская школа физико-математических  
и компьютерных наук  
НИУ ВШЭ — Санкт-Петербург

19 января 2022 г.

Специфика контекстного плагиата:

- Одиночные файлы
- Небольшой размер файлов
- Одинаковые паттерны

# Задача поиска списывания

Основные модификации<sup>123</sup>:

```
#include <iostream>
using namespace std;

int main()
{
    int a = 5, b = 2, c = 0;
    for (int i = 0; i < a; i++) {
        c += i;
        cout << i;
    }
    cout << c + a + b;
    return 0;
}
```

- Добавление/удаление комментариев
- Добавление незначимых строк кода
- Переименование
- Перестановка операций
- Взаимозаменяемые конструкции
  - for/while
  - if/else

---

<sup>1</sup>GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis (2006)

<sup>2</sup>Finding Plagiarisms among a Set of Programs with JPlag (2003)

<sup>3</sup>Comparison and evaluation of code clone detection techniques and tools: A qualitative approach (2009)

# Задача поиска списывания

```
#include <iostream>
using namespace std;

int main()
/* comment */ {
    int a = 5, b = 2, c = 0;
    for (int i = 0; i < a; i++) {
        // comment
        c += i;
        cout << i;
    }
    cout << c + a + b;
    return 0;
}
```

- Добавление/удаление комментариев
- Добавление незначимых строк кода
- Переименование
- Перестановка операций
- Взаимозаменяемые конструкции
  - for/while
  - if/else

# Задача поиска списывания

```
#include <iostream>
using namespace std;

int main()
/* comment */ {
    int a, b, c, d;
    a = 5;
    b = 2; ←
    c = 0;
    d = 42; ←
    for (int i = 0; i < a; i++) {
        // comment
        c += i;
        cout << i;
    }
    int ans = c + a + b; ←
    cout << ans;
    return 0;
}
```

- Добавление/удаление комментариев
- Добавление незначимых строк кода
- Переименование
- Перестановка операций
- Взаимозаменяемые конструкции
  - for/while
  - if/else

# Задача поиска списывания

```
#include <iostream>
using namespace std;

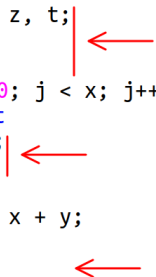
int main()
/* comment */ {
    int x, y, z, t;
    x = 5;
    y = 2;
    z = 0;
    t = 42;
    for (int j = 0; j < x; j++) {
        // comment
        z += j;
        cout << j;
    }
    int out = z + x + y;
    cout << out;
    return 0;
}
```

- Добавление/удаление комментариев
- Добавление незначимых строк кода
- Переименование
- Перестановка операций
- Взаимозаменяемые конструкции
  - for/while
  - if/else

# Задача поиска списывания

```
#include <iostream>
using namespace std;

int main()
/* comment */ {
    int x = 5, y, z, t;
    z = 0;
    y = 2;
    for (int j = 0; j < x; j++) {
        // comment
        cout << j;
        z += j;
    }
    int out = z + x + y;
    cout << out;
    t = 42;
    return 0;
}
```



- Добавление/удаление комментариев
- Добавление незначимых строк кода
- Переименование
- Перестановка операций
- Взаимозаменяемые конструкции
  - for/while
  - if/else

# Задача поиска списывания

```
#include <iostream>
using namespace std;

int main()
/* comment */ {
    int x = 5, y, z, t, j = 0;
    z = 0;
    y = 2;
    while (j < x) { ←
        // comment
        cout << j;
        z += j;
        j++; ←
    }
    int out = z + x + y;
    cout << out;
    t = 42;
    return 0;
}
```

- Добавление/удаление комментариев
- Добавление незначимых строк кода
- Переименование
- Перестановка операций
- Взаимозаменяемые конструкции
  - for/while
  - if/else



# Существующие решения и аналоги

- Sim<sup>4</sup>
  - Справляется с форматированием, переименованием и частично перестановкой инструкций
  - Поддерживает C, Java, Pascal
- MOSS<sup>5</sup>
  - Справляется с форматированием, переименованием и частично перестановкой инструкций
  - Поддерживает C/C++, C, Java, assembly
- GPLAG<sup>6</sup>
  - В основе сравнение Program Dependency Graph
  - Справляется со всеми основными модификациями
  - Поддерживает Java
  - Оценка качества поиска контекстного плагиата не проводилась
  - Отсутствует реализация алгоритма\*

---

<sup>4</sup>Sim: A Utility For Detecting Similarity in Computer Programs (1999)

<sup>5</sup>MOSS, A System for Detecting Software Plagiarism (2002)

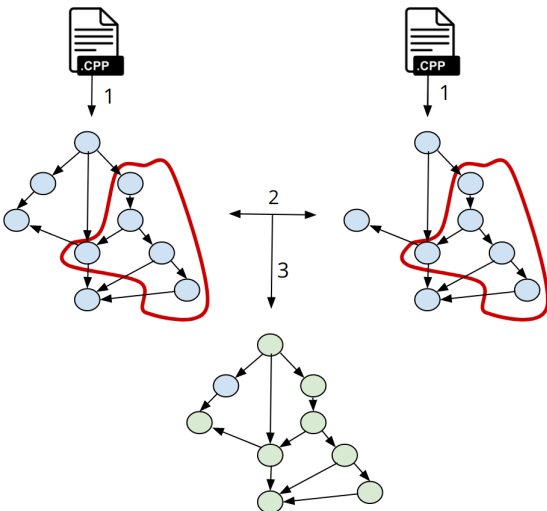
<sup>6</sup>GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis (2006)

**Цель:** Оценить применимость алгоритма GPLAG к решению задачи поиска контекстного плагиата

**Задачи:**

- Реализовать алгоритм GPLAG
- Собрать датасет для оценки применимости подхода к решению задачи поиска контекстного плагиата
- Провести тестирование и проанализировать работу полученного решения

# Реализация алгоритма. Пайплайн



- 1 Построение PDG с помощью Joern<sup>7</sup>
- 2 Сравнение графов
- 3 Оценка покрытия графа

---

<sup>7</sup>[Joern documentation](#)

# Реализация алгоритма. Сравнение графов

## Проблемы:

- 1 Подграфов в графах слишком много
- 2 Нужно учитывать типы вершин и ребер

## Решения:

- 1 Фиксируем размер подграфов: 9 вершин
- 2 Сужаем типы вершин до 60 основных

# Реализация алгоритма. Метрика похожести программ

- Похожесть =  $\frac{\text{Полученное покрытие}}{\text{Максимальное покрытие}}$
- При сравнении графов получаем покрытие
- Максимальное покрытие - покрытие полученное при сравнении программы с собой

**Итог:** реализован алгоритм GPLAG, поддержана возможность работы с разными языками программирования

**Мотивация:** Общедоступного датасета нет<sup>8</sup>

Датасет для оценки способности алгоритма находить плагиат и чувствительности к разным видам модификаций:

- Собрано 372 программы из 23 контекста с Codeforces
- С помощью инструмента gorshochek<sup>9</sup> построены модификации:
  - Добавление/удаление комментариев
  - Переименование
  - Замена взаимозаменяемых конструкций
- Добавлена возможность построения модификации вставки незначимых строк кода в gorshochek
- Для каждой программы построен файл с набором разных модификаций

---

<sup>8</sup>Academic Source Code Plagiarism Detection by Measuring Program Behavioral Similarity (2021)

<sup>9</sup>[github.com/JetBrains-Research/gorshochek](https://github.com/JetBrains-Research/gorshochek)

Датасет для оценки работы алгоритма в случаях отсутствия плагиата:

- 23 программы, решающих одну и ту же простую задачу (25 строк кода)
- 12 программ, решающих одну и ту же сложную задачу (60 строк кода)

# Тестирование. Результаты

	Средняя похожесть
Добавление/удаление комментариев	$1 \pm 0$
Вставка незначимых строк кода	$0.99 \pm 0.01$
Замена взаимозаменяемых конструкций	$0.94 \pm 0.02$
Переименование	$0.93 \pm 0.02$
Комбинация модификаций	$0.82 \pm 0.03$
Простая задача	$0.15 \pm 0.03$
Сложная задача	$0.05 \pm 0.03$

Вывод:

- Алгоритм справляется с выявлением контекстного плагиата
- Алгоритм корректно ведет себя в случаях отсутствия плагиата
- Алгоритм выдает неоднозначный результат на маленьких программах



- Реализован алгоритм из статьи GPLAG, поддержана возможность работы с разными языками программирования
- Собран датасет из 372 программ с 4 видами модификаций, 23 решений простой задачи и 12 решений сложной задачи
- Проведено исследование и показана применимость алгоритма GPLAG. По результатам тестирования алгоритм подходит для выявления контекстного плагиата, однако в случаях небольших программ возможны неточности.

Репозиторий: [github.com/Karina5005/Plagiarism](https://github.com/Karina5005/Plagiarism)

- Оценить точность других алгоритмов поиска плагата в применении к задаче поиска списываний в контекстах
- Заменить в системе построение PDG по абстрактному синтаксису на построение PDG по assembler и сравнить эти два подхода
- Придумать и применить эвристики для ускорения поиска подграфов и сокращения их количества без сильной потери точности работы алгоритма
- Решить проблему нахождения плагиата в случаях популярных паттернов

# Program Dependency Graph

## Definition

### Program Dependency Graph (PDG) – представление

программы в виде графа.

**Вершинами** являются базовые выражения.

**Ребра зависимости по данным** соединяют вершины, в которых используются одинаковые данные.

**Ребра передачи управления** соединяют две вершины, если контролирующая вершина определяет, будет ли выполняться выражение в зависимой вершине.

**program**

`x := 5; y := 5; a := 1;`

**while** `a < 5` **do**

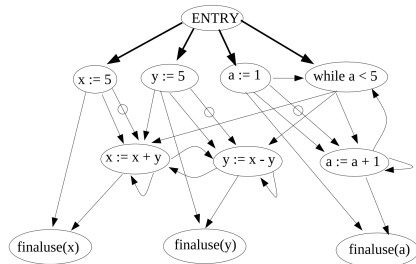
`x := x + y;`

`y := x - y;`

`a := a + 1;`

**end**

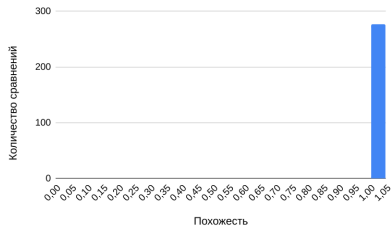
**end**(x, y, a)



# Тестирование. Результаты

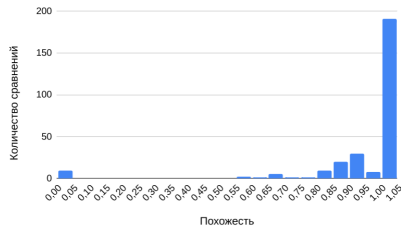
## Добавление/удаление комментариев

Среднее время работы: 96 секунд



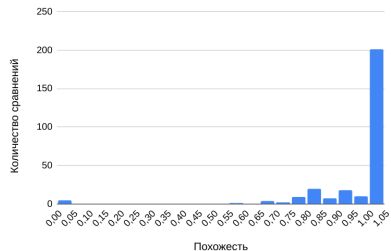
## Переименование

Среднее время работы: 141 секунда



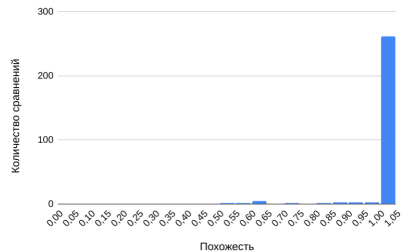
## Взаимозаменяемые конструкции

Среднее время работы: 141 секунда



## Вставка незначимых строк кода

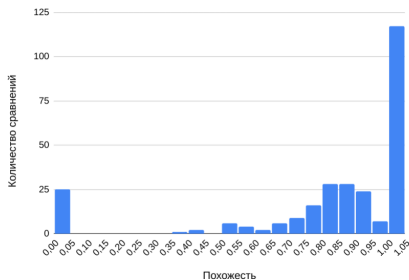
Среднее время работы: 127 секунд



# Тестирование. Результаты

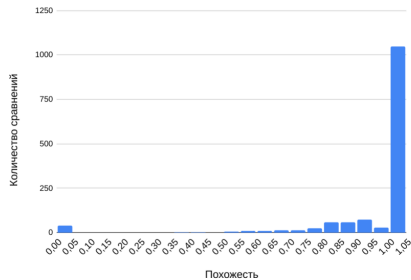
## Все модификации

Среднее время работы: 115 секунд



## Все сравнения

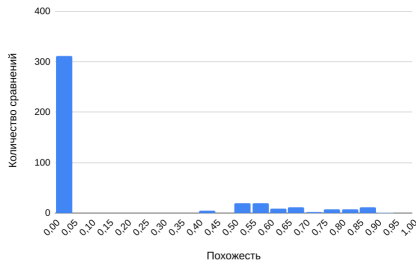
Среднее время работы: 123 секунды



# Тестирование. Результаты

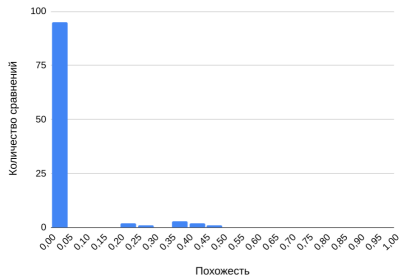
## Простая задача

Среднее время работы: 121 секунда



## Сложная задача

Среднее время работы: 1217 секунд



- Сложно автоматизировать
- Идеи автоматизации основаны на использовании графа зависимостей программ<sup>10</sup>

---

<sup>10</sup>The Program Dependence Graph and Its Use in Optimization (1987) ◀ ▶ ◀ ▶ ◀ ▶

Примеры сужения типов:

(<operator>.assignment, i = 1)       $\longrightarrow$     assignment

(<operator>.postIncrement, i++)       $\longrightarrow$     increment

(<operator>.lessEqualsThan, j <= n)  $\longrightarrow$     check



# Некорректная модификация

```
#include<bits/stdc++.h>
using namespace std;
#define all(v) v.begin(),v.end()

void solve(){
    string s;
    cin>>s;
    int c=count(all(s),'a');
    ...
}
```

```
#include<bits/stdc++.h>
using namespace std;
#define all(v) v.begin(),v.end()

void lx_rs(){
    string g;
    cin>>g;
    int w=count(all(s),'a');
    ...
}
```

# Похожие программы

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    long long int t,n,min=0,max=0,op;
    cin>>t;
    while(t--){
        cin>>n;
        int arr[n];
        for (int i=0; i<n; i++){
            cin>>arr[i];
        }

        sort(arr, arr+n);
        min=arr[0];
        max=arr[n-1];
        op=max-min;
        cout<<op<<endl;
    }

    return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int t;
    cin>>t;
    int max, min;
    while(t--){
        int n;
        cin>>n;
        deque <int> a(n);
        for(int i=0; i<n; i++){
            cin>>a[i];
        }
        sort(a.begin(), a.end());
        cout<<a.back()-a.front()<<"\n";
    }

    return 0;
}
```

# Решение сложной задачи

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int t;
    cin>>t;
    while(t-->0)
    {
        int i,n,j,a=1,b,c;
        cin>>n;
        vector<int> v(n+1);
        unordered_map<int,int> m;
        v[0]=0;
        for(i=1;i<=n;i++)
        {
            cin>>v[i];
            if(v[i]<=n)
                m[v[i]]++;
        }
        for(i=1;i<=n;i++)
        {
            if(m[v[i]]!=1)
            {
                int temp=v[i];
                while(temp>0)
                {
                    temp/=2;
                    if(temp<=n)
                    {
                        if(m[temp]==0)
                        {
                            m[temp]=1;
                            m[v[i]]--;
                            v[i]=temp;
                            break;
                        }
                    }
                }
                if(temp==0)
                {
                    cout<<"NO\n";
                    a=0;
                    break;
                }
            }
        }
        if(a)
            cout<<"YES\n";
    }
}
```

- PROGEX<sup>11</sup>

- Парсит только Java, но можно добавлять и свои грамматики
- На практике добавление новой грамматики проблематично

- TinyPDG<sup>12</sup>

- Умеет парсить только Java код
- Добавление новой грамматики не предусмотрено

- Joern

- Умеет парсить только C/C++ и Java код
- Результат в формате \*.dot

---

<sup>11</sup>[github.com/ghaffarian/progex](https://github.com/ghaffarian/progex)

<sup>12</sup>[github.com/YoshikiHigo/TinyPDG](https://github.com/YoshikiHigo/TinyPDG)

Существует две реализации:

- `vfrunza\GPLAG-Plagerism-Detection`<sup>13</sup>
  - Полностью самостоятельная реализация
  - Упрощенная версия графа зависимостей программ
  - Поддерживает только C
  - Нет тестов отдельных частей
  - Тестирование проводилось на большом датасете модификаций элементарной программы (9 строк)
- `sarahfoss\GPLAG`<sup>14</sup>
  - Полностью самостоятельная реализация
  - Поддерживает только Java
  - Нет тестов отдельных частей
  - Тестирование проводилось на 4 тестовых файлах

---

<sup>13</sup> [github.com/vfrunza/GPLAG-Plagerism-Detection](https://github.com/vfrunza/GPLAG-Plagerism-Detection)

<sup>14</sup> [github.com/sarahfoss/GPLAG](https://github.com/sarahfoss/GPLAG)