Міністерство освіти і науки України
Департамент науки і освіти Харківської облдержадміністрації
Харківське територіальне відділення МАН України

Відділення: комп'ютерних наук
Секція: комп'ютерні системи та мережі

РОЗРОБКА МЕТОДА ПОШУКУ ТА УСУНЕННЯ ПОВТОРЮВАНИХ ЧАСТИН У ПОЧАТКОВОМУ КОДІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Роботу виконав:
Човпан Ігор Сергійович,
учень 11 класу Харківського Навчально-виховного комплексу №45 «Академічна гімназія» Харківської міської ради Харківської області

Науковий керівник:
Руккас Кирило Маркович,
професор кафедри теоретичної та прикладної інформатики механіко-математичного факультету Харківського національного університету ім. В.Н. Каразіна, доктор технічних наук, доцент

Харків – 2020

Тези

...

ЗМІСТ

ВСТУП

Ваш ВСТУП

РОЗДІЛ 1.
ХАРАКТЕРИСТИКА ІСНУЮЧИХ МЕТОДІВ ЗНАХОДЖЕННЯ ДУБЛІКАТІВ У КОДІ

1.1. Визначення основних видів повторюваних частин

Як зазначено у роботах [7], [4] та [2], виділяється 4 головних типи повторюваних частин.

- I тип – повна копія без модифікацій, окрім пробілів та коментарів;

```
double xx = Math.cos(angle);
double yy = Math.sin(angle);
xx*=2;
yy*=2;
if (xx>PI)
  xx = 2*PI-xx;
if (yy>PI)
  yy = 2*PI-yy;
```

```
double xx = Math.cos(angle);
// of course using math package!!
double yy = Math.sin(angle);
xx *= 2;
yy *= 2;
if ( xx > PI)
//That is VERY important statement!
  xx = 2 * PI - xx;
if (yy > PI)
  yy = 2 * PI - yy;
```

Приклад копії I типу на мові Java

- II тип – синтаксично однакова копія, змінюються лише назви змінних, назв функцій, тощо;

```
void func(double angle) {
  double xx = Math.cos(angle);
  double yy = Math.sin(angle);
  xx*=2;
  yy*=2;
  if (xx>PI)
    xx = 2*PI-xx;
  if (yy>PI)
    yy = 2*PI-yy;
  write(xx);
}
```

```
void veryImportantFunc(double ang){
  double aa = Math.cos(ang);
  double bb = Math.sin(ang);
  aa*=2;
  bb*=2;
  if (xx>PI_CONST)
    aa = 2*PI_CONST-aa;
  if (yy>PI_CONST)
    bb = 2*PI_CONST-bb;
  writeToFile(aa);
}
```

Приклад копії II типу

- III тип – копія з подальшими змінами; доданими, зміненими або видаленими інструкціями;

```
void func(double angle) {
  double xx = Math.cos(angle);
  double yy = Math.sin(angle);
  double PI = Math.acos(-1);
  xx*=2;
  yy*=2;
  if (xx>PI)
    xx = 2*PI-xx;
  if (yy>PI)
    yy = 2*PI-yy;
  write(xx);
}
```

```
void doCalc(double ang){
  double bb = Math.sin(ang);
  double aa = Math.cos(ang);
  print("before"+aa);
  aa*=2;
  if (xx>PI_CONST)
    aa = 2*PI_CONST-aa;
  bb*=2;
  if (yy>PI_CONST)
    bb = 2*PI_CONST-bb;
  print(aa);
}
```

Приклад копії III типу

```
int fibonacci(int n) {
  int sum1=0, sum2=1;
  for (int i=2; i<=n; i++){
    int sum3 = sum3+sum2;
    sum1 = sum2;
    sum2 = sum3;
  }
  return sum2;
}
```

```
int[] mem = new int[...];
int fib(int n) {
  if (mem[n]!=0)
    return mem[n];
  if (n<2)
    return 1;
  mem[n] = fib(n-1)+fib(n-2);
  return mem[n];
}
```

Приклад копії IV типу

- IV тип – частина, що робить ідентичні обчислювання, але синтаксично імплементована інакше.

1.2. Основні методи пошуку повторюваних частин

Існує багато прийомів, що використовуються для пошуку повторюваних частин у початковому коді програмного забезпечення.

Перелічим основні методи пошуку:

- пошук збігу рядків початкового коду;
- використання токенів;
- метод порівняння функцій;
- застосування графа програмних залежностей;
- метод порівняння дерев.

Далі визначимо усі переваги і недоліки кожного з методів.

1.2.1 Пошук збігу рядків початкового коду

Обчислюється ступінь схожості для кожної пари рядків за допомогою відстані Левенштейна. Емпірично встановлюється мінімальна величина, за якої вважається, що 2 рядки є копіями одна одну.

Переваги цього метода:

- добре знаходить копії I типу;
- невеликий час виконання порівняно з іншими методами;
- підтримка будь-якої мови программування.

Недоліки метода:

- велика кількість хибнонегативних результатів;
- нестійкість до різних ”шумів”: коментарів, змінених назв функцій або змінних, тобто неможливість знайти дублікати II та III типу.
- Не враховуються особливості мови програмування.

Прикладом використання є програма PMD.

1.2.2 Використання токенів

Метод використовує токенізатор. У ньому початковий код розбивається на токени, при пошуці порівнюються послідовності токенів. Головною перевагою цього методу є стійкість до переформатування початкового коду, зміні назв змінних. Недоліком є те, що токенізатори враховують тільки базові особливості мови програмування. Через те, що токени усе одно порівнюються як рядки, присутня досить велика похибка при розпізнаванні копій. Наприклад, наступна пара токенів може вважатися як копії один одному:

`reallySoLongNameThatYouCouldNotStandIt1`

та

`reallySoLongNameThatYouCouldNotStandIt2.`

Прикладом використання є програма CCFnderX.

1.2.3 Метод порівняння функцій

За допомогою парсера мови програмування знаходять усі функції у початковому коді. Далі усі ці функції порівнюються між собою або за допомогою спеціально обраної ”поганої”геш-функції, або за допомогою обчислення коефіцієнту схожості (наприклад, коеф. Жаккара). Метод гарно знаходить збіги між різними функціями, розпізнаються копії I-III типу, проте він не може знайти повторювани частини всередині коду. Прикладом використання є [9].

1.2.4 Застосування графа програмних залежностей

Згідно з [6], граф програмних залежностей (далі просто граф) – представлення програми як графа, у якому кожна вершина - інструкція у програмі, а також зв’язані з цією інструкцією оператори та операнди; ребрами у такому графі є дані, від яких залежить виконання цієї інструкції та умови, за яких ця інструкція виконається. Дві частини програми вважаються ідентичними, якщо їх графи ізоморфні. Головною перевагою є те, що цей граф не залежить від переставлення інструкції програми; зміни назв функцій, змінних, тощо; не залежить від аспектів реалізації. Недоліки методу:

- Дуже довгий час роботи, оскільки завдання пошуку ізоморфних підграфів є NP-повною, і може бути вирішена за поліноміальний час тільки для планарних графів, що не обов’язково виконається для графа програмних залежностей;
- Такий метод не зможе знайти дублікати у коді, який не виконується у загальному випадку, оскільки у граф додаються лише виконані інструкції. Приклад використання: [8].

1.2.5 Метод порівняння дерев

У цьому методі використовуються абстрактні синтаксичні дерева (АСД). Згідно з [10], абстрактне синтаксичне дерево – позначене і орієнтоване дерево, в якому внутрішні вершини співставлені з відповідними операторами мови програмування, а листя з відповідними операндами. Щоб визначити, чи є частина коду копією іншої частини, знаходять відповідні їм піддерева, а далі ці піддерева порівнюються між собою.

Підходів порівняння піддерев досить багато. Так, наприклад, у роботі [5] усі піддерева, що відповідають класам у початковому коді, порівнюються кожен з кожним за допомогою обчислення відстані між деревами. Автор відмічає, що цей метод є найточнішим порівняно з іншими, але має дуже довгий час роботи. Наприклад, код плагінів `org.eclipse.compare-plug-in`, що складався зі 114 класів перевірявся на копії більше ніж годину.
У роботі [3] теж стверджується, що алгоритм знахождення між деревами має занадто велику складність обчислення, тому автором було запропоновано інший підхід. Усі піддерева гешуються за допомогою вибраної ”поганої”геш-функції та розподіляються у B бакетів, де $B \approx N/10$. Далі кожне піддерево порівнюється лише з піддеревами з цього ж бакету за формулою:

$$\text{Схожість} = \frac{2 * S}{2 * S + L + R}.$$

S – кількість однакових вершин у обох піддеревах, L – кількість вершин, що присутні лише у першому піддереві, R – кількість вершин, що є тільки у другому піддереві.
Отже, головними перешкодами до використання цього методу є:

- Великі час роботи та алгоритмічна складність; [1]
- Досить низький відсоток знайдених копій через використання додаткових евристик. [4]

Далі буде запропоновано новий підхід, що значно зменшить час роботи, необхідний для знаходження копій, та, у той же час, збільшить ефективність знаходження копій.

РОЗДІЛ 2.
НОВИЙ АЛГОРИТМ

...

РОЗДІЛ 3.
ПОРІВНЯННЯ З ІНШИМИ МЕТОДАМИ

...

РОЗДІЛ 4.
ВИСНОВКИ

...

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Systematic Review on Code Clone Detection / Q. U. Ain [та ін.] // IEEE Access. — 2019. — Т. 7. — С. 86121—86144. — DOI: 10.1109/ACCESS.2019.2918202.
2. An Empirical Study on the Maintenance of Source Code Clones / S. Thummalapenta [та ін.] // Empirical Software Engineering. — 2010. — Лют. — Т. 15. — С. 1—34. — DOI: 10.1007/s10664-009-9108-x.
3. Clone detection using abstract syntax trees / I. D. Baxter [та ін.] // Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272). — 1998. — С. 368—377. — DOI: 10.1109/ICSM.1998.738528.
4. *Dang S.* Performance Evaluation of Clone Detection Tools // International Journal of Science and Research (IJSR). — 2015. — Квіт. — Т. 4. — С. 1903—1906.
5. Detecting similar Java classes using tree algorithms / T. Sager [та ін.] //. — 01.2006. — С. 65—71. — DOI: 10.1145/1137983.1138000.
6. *Ferrante J.*, *Ottenstein K.*, *Warren J.* The Program Dependence Graph and Its Use in Optimization. // ACM Transactions on Programming Languages and Systems. — 1987. — Лип. — Т. 9. — С. 319—349. — DOI: 10.1145/24039.24041.
7. *Gautam P.*, *Saini H.* Various Code Clone Detection Techniques and Tools: A Comprehensive Survey // Smart Trends in Information Technology and Computer Communications / за ред. A. Unal [та ін.]. — Singapore : Springer Singapore, 2016. — С. 655—667. — ISBN 978-981-10-3433-6.
8. GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis / C. Liu [та ін.] // Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. — Philadelphia, PA, USA : Association for Computing Machinery, 2006. — С. 872—881. — (KDD '06). — ISBN 1595933395. — DOI: 10.1145/1150402.1150522. — URL: https://doi.org/10.1145/1150402.1150522.
9. Structural Function Based Code Clone Detection Using a New Hybrid Technique / Y. Yang [та ін.] // 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC). Т. 01. — 2018. — С. 286—291. — DOI: 10.1109/COMPSAC.2018.00045.

10. *Вікіпедія*. Абстрактне синтаксичне дерево — Вікіпедія, — 2020. — URL: https://uk.wikipedia.org/w/index.php?title=%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D0%B5_%D1%81%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%BD%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE&oldid=27659104 ; [Онлайн; цитовано 5-листопад-2020].