HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

NGUYỄN THÀNH NAM

NGHIÊN CỨU PHƯƠNG PHÁP XÁC ĐỊNH MỨC ĐỘ TƯƠNG TỰ GIỮA CÁC MÃ NGUỒN DỰA VÀO CÂY CÚ PHÁP

Chuyên ngành: HỆ THỐNG THÔNG TIN
Mã số: 8.48.01.04

TÓM TẮT LUẬN VĂN THẠC SĨ

HÀ NỘI - NĂM 2022

Luận văn được hoàn thành tại:

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Người hướng dẫn khoa học: TS. NGUYỄN DUY PHƯƠNG

(Ghi rõ học hàm, học vị)

Phản biện 1: PGS.TS. Nguyễn Hữu Quỳnh

Phản biện 2: TS. Ngô Quốc Dũng

Luận văn được bảo vệ trước Hội đồng chấm luận văn thạc sĩ tại Học viện Công nghệ Bưu chính Viễn thông

Vào lúc: … giờ … ngày … tháng … năm 2022

Có thể tìm hiểu luận văn tại:

- Thư viện của Học viện Công nghệ Bưu chính Viễn thông.

MỞ ĐẦU

1. Lý do chọn đề tài

Theo thời gian, lượng tài liệu được tạo ra, lưu trữ và chia sẻ trên Internet ngày càng nhiều hơn. Tuy có nhiều thuận lợi cho việc tiếp cận kho tri thức mở, nhưng rõ ràng cũng đã nảy sinh nhiều vấn đề: các ý tưởng, các đoạn văn thậm chí toàn văn bản được sao chép một cách tràn lan mà không có chú thích nguồn hay tác giả. Điều này không chỉ ảnh hưởng đến quyền sở hữu trí tuệ, bản quyền của tác giả mà còn ảnh hưởng đến việc đánh giá chất lượng của các tài liệu báo cáo, tiểu luận, luận văn và trình độ của người tạo ra chúng, đặc biệt là trong môi trường giáo dục nói chung và đại học nói riêng.

Ngoài ra trong lĩnh vực giáo dục đào tạo, việc sao chép mã nguồn cũng có những ảnh hưởng nhất định tới chất lượng đào tạo nhân lực Công nghệ thông tin. Vấn đề này phổ biến đối với nhiều trường đại học trên thế giới, và các trường đại học tại Việt Nam cũng không phải là ngoại lệ. Các sinh viên sẽ sao chép mã nguồn của sinh viên khác sau đó sửa chữa đi một vài điểm để chuyển thành sản phẩm của mình.

So với việc sao chép các bài luận, các đề tài, nghiên cứu khoa học mà hình thức biểu hiện ở dưới dạng ngôn ngữ văn bản thông thường, việc sao chép mã nguồn lập trình có nhiều điểm khác biệt và đặc thù. Để giải quyết thực trạng này, trên thế giới đã có nhiều phần mềm được phát triển để đánh giá sự tương tự giữa các bộ mã nguồn. Việc nghiên cứu các phương pháp để tối ưu hơn việc phát hiện việc sao chép mã nguồn là một bài toán thực sự thú vị và có nhiều ý nghĩa cũng như ứng dụng trong thực tiễn.

Như vậy có thể thấy, thực trạng sao chép mã nguồn xuất hiện trong cả ngành công nghiệp phát triển phần mềm cũng như trong công tác đào tạo sinh viên ngành công nghệ thông tin. Phạm vi nghiên cứu của đề tài này sẽ tập trung sâu hơn vào vấn đề sao chép mã nguồn trong công tác đào tạo sinh viên ngành công nghệ thông tin tại giảng đường.

1 CHƯƠNG 1: TỔNG QUAN VỀ BÀI TOÁN ĐÁNH GIÁ MỨC ĐỘ TƯƠNG TỰ GIỮA CÁC MÃ NGUỒN

1.1. Tổng quan về sao chép và sử dụng lại mã nguồn

1.1.1 Vấn đề sao chép và sử dụng lại mã nguồn

Lập trình viên trong thời đại ngày nay được tiếp cận với một lượng tài nguyên khổng lồ và gần như không giới hạn. Nếu như trước kia, các tài liệu của các ngôn ngữ lập trình hay sự hỗ trợ từ cộng đồng là hạn chế, thì ngày nay, những cộng đồng lập trình trên toàn thế giới (stackoverflow, github…) cho phép lập trình viên nhanh chóng học hỏi và tận dụng lại tri thức từ những người đi trước. Đôi khi một bài toán trước đây cần nhiều thời gian suy nghĩ, thì ngày nay có thể đã được lập trình viên đi trước chia sẻ công khai, chỉ cần sao chép về và đưa vào dự án của mình.

Giờ đây việc tiếp cận các ngôn ngữ lập trình mới, các thư viện/framework có độ trưởng thành cao được cộng đồng chia sẻ, giúp cho sinh viên dù chưa có nhiều kinh nghiệm cũng có thể dễ dàng xây dựng nên một phần mềm tương đối hoàn chỉnh. Việc này rút ngắn thời gian tạo ra sản phẩm, tuy nhiên điều này sẽ làm giảm khả năng sáng tạo, đào sâu tìm tòi giải pháp của sinh viên. Trong khi đây là yếu tố rất quan trọng trong giai đoạn sinh viên công nghệ thông tin được đào tạo tại trường.

1.1.2 Những tác động của sao chép và sử dụng lại mã nguồn

Việc sao chép, sử dụng lại các mã nguồn cũng có những nhược điểm nhất định cần quan tâm.

Đối với công tác đào tạo:

- Giảm khả năng sáng tạo của sinh viên: từ đó giảm chất lượng của công tác đào tạo.
- Kết quả đào tạo không khách quan: từ đó giáo viên khó đánh giá chất lượng kết quả đào tạo để đưa ra phương hướng, điều chỉnh cho nội dung giảng dạy của mình.

Đối với quy trình phát triển phần mềm:

- Tăng khả năng lỗi tiềm ẩn trong những đoạn mã nguồn giống nhau, từ đó gây khó khăn cho công tác bảo trì, phát triển mở rộng.
- Gây khó khăn cho công tác tối ưu mã nguồn, các đoạn mã nguồn tương tự nhau lặp đi lặp lại trong ứng dụng.

1.1.3 Ý nghĩa của việc đánh giá mức độ tương tự giữa các mã nguồn

Đối với công tác đào tạo:

- **Phát hiện sao chép**.
- **Phân tích và đánh giá**

Đối với quy trình phát triển phần mềm:

- **Phân tích và cải tiến hiệu năng của những đoạn mã nguồn tương tự**
- **Tìm ra những đoạn mã nguồn không tốt**
- **Phát hiện sao chép mã nguồn mà không được phép**

1.1.4 Giới thiệu các kiểu sao chép mã nguồn phổ biến

Dưới đây là danh sách một số kiểu sao chép mã nguồn phổ biến, được liệt kê theo thứ

tự từ đơn giản đến phức tạp:

1. Thay đổi định dạng hoặc thêm/sửa các "comment code" (ghi chú).
2. Thay đổi tên của các hàm hoặc các biến.
3. Thay đổi thứ tự các toán hạng trong các biểu thức.
4. Thay đổi kiểu dữ liệu của các biến.
5. Thay một biểu thức bằng một biểu thức tương đương
6. Thêm các đoạn code dư thừa (deadcode)
7. Thay đổi thứ tự của các đoạn mã nguồn độc lập.
8. Thay đổi một vòng lặp bởi một vòng lặp khác
9. Thay đổi cấu trúc của câu lệnh lựa chọn
10. Thay thế việc gọi hàm mới bằng nội dung hàm
11. Kết hợp mã nguồn sao chép với mã nguồn của bản thân.

1.2. Đề xuất bộ tiêu chí đánh giá mức độ tương tự giữa các mã nguồn

1.2.1 Khái niệm ma trận nhầm lẫn (Confusion Matrix)

Confusion Matrix hay còn gọi là Ma trận nhầm lẫn, ma trận lỗi (Kohavi và Procot, 1998) là bố cục bảng cụ thể cho phép hình dung hiệu suất của một hệ thống phân loại. Hãy xem xét ví dụ thực tế dưới đây:

Vào một ngày nào đó, Bệnh viện A có 100 bệnh nhân đến khám bệnh, giả sử thực tế là trong 100 bệnh nhân có 60 người mắc bệnh, 40 người không có bệnh. Sau khi thăm khám, bệnh viện đưa ra kết quả:

- **Trong 60 người có bệnh thật gồm:**
  - 50 người chẩn đoán có bệnh
  - 10 người chẩn đoán không mắc bệnh.
- **Trong 40 người không có bệnh gồm:**
  - 25 người chẩn đoán không mắc bệnh
  - 15 người chẩn đoán là mắc bệnh.

Bảng 1.1: Kết quả ma trận nhầm lẫn sau khi chẩn đoán

	Dương tính (P)	**Âm tính (N)**
Dương tính	**TP (50)**	FP (15)
Âm tính	FN (10)	**TN (25)**

Từ ma trận cơ bản này, ta sẽ có một số thuật ngữ sau:

- Condition positive (P): Tổng số ca dương tính thực tế.
- Condition Negative (N: Tổng số ca âm tính thực tế.
- True positive (TP): Số các ca dự đoán dương tính đúng hay dương tính thật.
- True negative (TN): Số các ca dự đoán âm tính đúng hay âm tính thật.
- False positive (FP): Số các ca dự đoán dương tính sai hay dương tính giả.
- False negative (FN):: Số các ca dự đoán âm tính sai hay âm tính giả.

Để đánh giá kết quả khám bệnh của của bệnh viện trên, chúng ta có các chỉ số đánh giá quan trọng sau:

- Độ chính xác ACC (Accuracy)

$$ACC = \frac{\text{Số dự đoán đúng}}{\text{Tổng số mẫu}} = \frac{TP + TN}{P + N} = \frac{50 + 25}{100} = 0{,}75$$

- Độ nhạy TPR (True Positive Rate - Tỷ lệ dương tính thực)

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} = \frac{50}{50 + 10} = 0{,}833$$

- FPR (False Positive Rate - Tỷ lệ dương tính giả)

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} = \frac{15}{40} = 0{,}375$$

- Độ đặc hiệu (Độ đặc trưng)

$$\text{Độ đặc hiệu} = 1 - \text{FPR} = 1 - 0{,}375 = 0{,}625$$

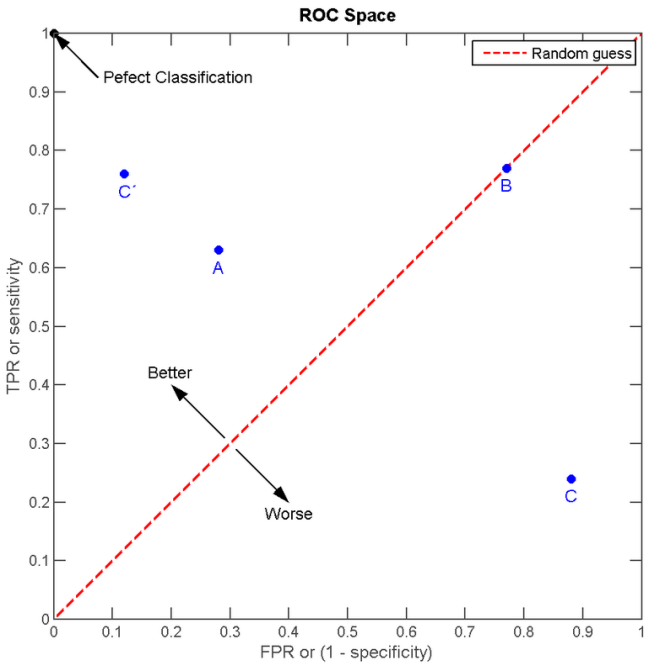
1.2.2 Biểu diễn đường cong ROC

Khi biểu diễn 2 chỉ số TPR và FPR trên đồ thị với trục tung là TPR và trục hoành là FPR, chúng ta có khái niệm không gian ROC.

Bảng 1.2: So sánh kết quả các phép đánh giá khi chẩn đoán

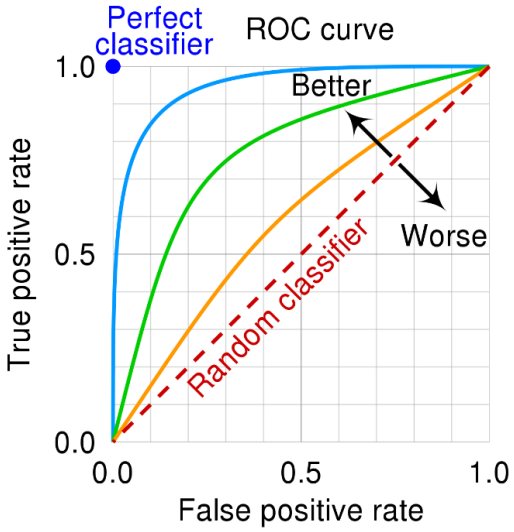
	A	**B**	**C**	**D**
TP	63	77	24	76
FP	28	77	88	12
FN	37	23	76	24
TN	72	23	12	88
TPR	**0,63**	**0,77**	**0,24**	**0,76**
FPR	**0,28**	**0,77**	**0,88**	**0,12**
ACC	**0,68**	**0,50**	**0,18**	**0,82**

Theo đó kết quả thu được như sau:



Hình 1.1: Không gian ROC

Những phép dự đoán cho kết quả nằm phía trên đường chéo được đánh giá là có kết quả tốt hơn so với các phép dự đoán cho kết quả nằm dưới đường chéo, phép đo có kết quả càng gần với đỉnh trên bên trái (0,1) thì càng tiệm cận tới phép đo "hoàn hảo".



Hình 1.2: Đường cong ROC

1.3. Đề xuất phương pháp đánh giá mức độ tương tự giữa các mã nguồn

1.3.1 Tổng quan về các công nghệ so sánh mã nguồn phổ biến

Dựa trên phân tích văn bản thuần túy (string-based): Mã nguồn được chuyển thành các chuỗi liên tiếp ký tự. Hai mã nguồn được coi là tương tự nhau nếu chúng bao gồm các chuỗi ký tự giống nhau. Thuật toán chính được sử dụng trong công nghệ này là thuật toán tìm chuỗi con dài nhất (LCS).

Dựa trên phân tích chuỗi ký tự đại diện (token-based): Công nghệ này được cải tiến từ việc phân tích mã nguồn dựa trên văn bản thuần túy trình bày phía trên. Mã nguồn chương trình lúc này được tách từ thành chuỗi các token, sau đó thực hiện quét và kiểm tra các chuỗi token trùng nhau để xác định khả năng xảy ra lặp, sao chép mã nguồn. Tuy nhiên các công cụ này đều có nhược điểm không thể phát hiện khi mã nguồn có sự thay đổi như thay đổi tên hàm, biến, thứ tự các đoạn code độc lập, hay chèn thêm các đoạn code vô nghĩa.

Dựa trên cấu trúc cây (tree-based): Đây là công nghệ mới được đề cập và phát triển nhanh trong những năm gần đây. Việc so sánh các mã nguồn dựa trên cấu trúc của nó là hướng đi rất hợp lý, khi mà nó có thể bằng cách này hay cách khác để loại bỏ sự ảnh hưởng bởi những thao tác như: đổi tên hàm, tên biến, thứ tự code…

Do ưu điểm của phương pháp tiếp cận này, nên các thuật toán trong luận văn này tập trung vào việc phân tích mã nguồn dựa trên cấu trúc cây.

Dựa trên biểu đồ phụ thuộc (Program Dependence Graph - PDG): Một số nghiên cứu chỉ ra rằng các thủ thuật sao chép mã nguồn không làm thay đổi nhiều đối với biểu đồ phụ thuộc của mã nguồn gốc.

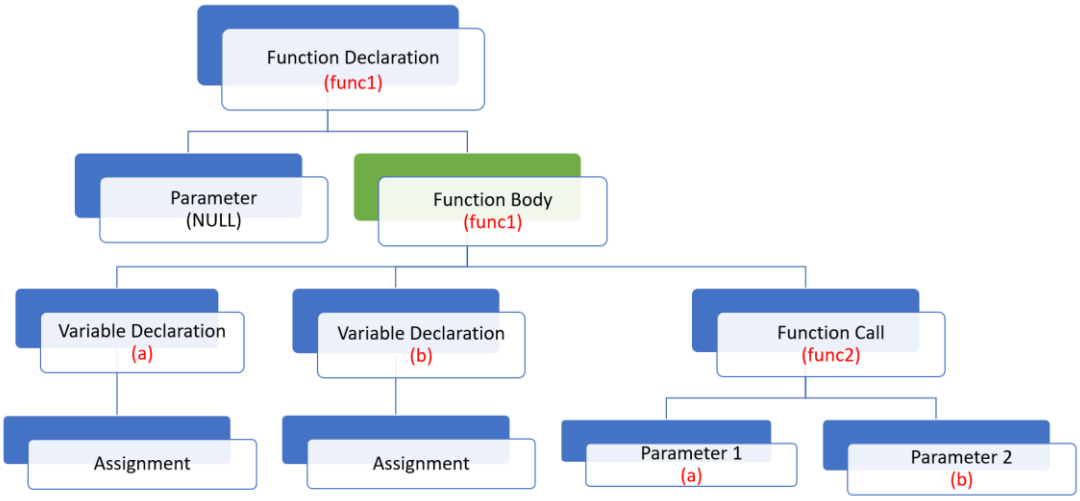
1.3.2 Khái niệm về cây cú pháp trừu tượng (AST, abstract syntax tree)

Trong ngành khoa học máy tính, cây cú pháp trừu tượng (AST, abstract syntax tree), là cấu trúc dữ liệu được sử dụng rộng rãi trong trình biên dịch, AST thường là kết quả của giai đoạn phân tích cú pháp của trình biên dịch.

Hãy cùng xem xét ví dụ đoạn mã nguồn sau (ngôn ngữ C++):

```
void fun1(){
     int a = 2;
     int b = 5;
     fun2(a, b);
}
```

Về cơ bản đoạn mã nguồn này có cây AST như sau:



Hình 1.3: Cây cú pháp trừu tượng của mã nguồn hàm func

1.3.3 Các phần mềm sinh ra cây cú pháp AST

Mỗi ngôn ngữ lập trình có những công cụ complier khác nhau (ví dụ Java có Java Complier Complier – JavaCC hoặc JJTree, Javascript có JIT, C++ có GCC hoặc Clang. Tương ứng với mỗi công cụ này, sẽ có các công cụ xây dựng cây cú pháp AST khác nhau.

Trong phạm vi luận văn này, do chỉ tập trung vào ngôn ngữ C++ nên chọn lựa trình biên dịch Clang trong bộ khung biên dịch LLVM để phục vụ cho việc xây dựng cây AST từ các mã nguồn viết bằng ngôn ngữ C++. Dưới đây là một số thông tin tổng quan về LLVM và Clang.

1.3.4 Tổng quan về LLVM (Low-Level Virtual Machine)

LLVM là một bộ khung biên dịch (compiler framework) được sử dụng để tối ưu hóa mã cho nhiều ngôn ngữ lập trình khác nhau. Nó cung cấp những công cụ mạnh mẽ để xây dựng phần front-end (parser, lexer) cũng như phần backend (phần chuyển phần code trung gian LLVM sang mã máy), cho các ngôn ngữ lập trình mới.

1.3.5 Tổng quan về Clang

Clang là một trình biên dịch (compiler front-end) sử dụng cơ sở hạ tầng trình biên dịch LLVM cho các ngôn ngữ lập trình C, C ++, Objective-C và Objective-C ++.

Đặc điểm và mục tiêu của Clang:

- Đối với Người dùng cuối:
    - Biên dịch nhanh và sử dụng ít bộ nhớ

- Chẩn đoán nhanh (ví dụ)
- GCC tương thích

- Đối với Tiện ích và Ứng dụng:
  - Kiến trúc mô đun hóa
  - Hỗ trợ đa dạng "client" (tái cấu trúc, phân tích tĩnh, tạo mã, v.v.)
  - Cho phép tích hợp chặt chẽ với các IDE
  - Sử dụng Giấy phép LLVM Apache 2
- Thiết kế nội bộ và Triển khai:
  - Trình biên dịch chất lượng áp dụng được áp dụng vào thực tiễn
  - Một cơ sở mã nguồn đơn giản và dễ hiểu, dễ tùy biến
  - Một trình phân tích cú pháp hợp nhất cho các ngôn ngữ C, Objective C, C ++ và Objective C ++
  - Tuân thủ C / C ++ / ObjC và các biến thể của chúng

Hình 1.4: Các pha của trình biên dịch CLang

Các cách sử dụng Clang

- Cài đặt và sử dụng trực tiếp Clang thông qua bộ thư viện mà Clang công khai trên trang chủ tại địa chỉ: https://clang.llvm.org/get_started.html. Lưu ý việc triển khai này phụ thuộc môi trường cài đặt của máy tính.
- Cài đặt và sử dụng gián tiếp thông qua API của bộ thư viện tích hợp LibClang trên các ngôn ngữ lập trình phổ biến (Python). Đây là cách được triển khai trong nội dung luận văn này.

1.3.6 Sử dụng Clang với Python

Python là một ngôn ngữ lập trình phổ biến và được cộng đồng hỗ trợ nhiều trong các năm gần đây.

Thông qua các API của thư viện LibClang trên Python, chúng ta có thể sử dụng một cách dễ dàng nhưng đảm bảo tương đối đầy đủ các thao tác đối với cây AST.

Hãy cùng xem xét cây AST tạo ra tương ứng với đoạn mã nguồn trong **phần 1.3.2** bởi LibClang, được thể hiện dưới dạng cây như sau:

```
C:\Users\Admin\AppData\Local\Programs\Python\Python37\python.exe D:/CaoHoc/LuanVan/source_code/CompareAST/ast_viewer.py
├──TRANSLATION_UNIT test_data_2/test_6.cpp  [line=0, col=0]
   └──FUNCTION_DECL fun1  [line=1, col=6]
      └──COMPOUND_STMT   [line=1, col=13]
         ├──DECL_STMT   [line=2, col=1]
         │  └──VAR_DECL a  [line=2, col=5]
         │     └──INTEGER_LITERAL   [line=2, col=9]
         ├──DECL_STMT   [line=3, col=1]
         │  └──VAR_DECL b  [line=3, col=5]
         │     └──INTEGER_LITERAL   [line=3, col=9]
         └──UNEXPOSED_EXPR   [line=4, col=1]
            ├──DECL_REF_EXPR   [line=4, col=1]
            │  └──OVERLOADED_DECL_REF fun2  [line=4, col=1]
            ├──DECL_REF_EXPR a  [line=4, col=6]
            └──DECL_REF_EXPR b  [line=4, col=9]
```

Hình 1.5: Hiển thị một cây AST sinh ra bởi mã nguồn

1.4. So sánh cây AST khi áp dụng các thủ thuật sao chép khác nhau

Mã nguồn gốc để sử dụng cho mục này là bài tập xây dựng hàm tính chuỗi Fibonacci:

```
#include <stdio.h>
#include <conio.h>
int Fibonacci(int n){
  if (n == 1 || n == 2)
     return 1;
  return Fibonacci(n - 1) + Fibonacci(n - 2);
}
```

1.4.1 Thay đổi định dạng hoặc thêm/sửa các "comment code"

Do đặc tính của cây cú pháp trừu tượng AST, các khoảng trắng và ghi chú (comment code) bị loại bỏ khi trình biên dịch xây dựng cây AST, vì vậy việc thay đổi định dạng hoặc ghi chú sẽ không làm ảnh hưởng tới cấu trúc của AST.

1.4.2 Đổi tên các định danh (hàm, tham số và biến)

Với thủ thuật sao chép này, khi so sánh cấu trúc AST, chúng ta chỉ cần bỏ qua giá trị tại các node, chỉ sử dụng loại node và cấu trúc AST để so sánh. Khi đó cấu trúc AST mã nguồn gốc và mã nguồn đã sửa đổi là tương đương nhau.

1.4.3 Thay đổi thứ tự các toán hạng trong biểu thức

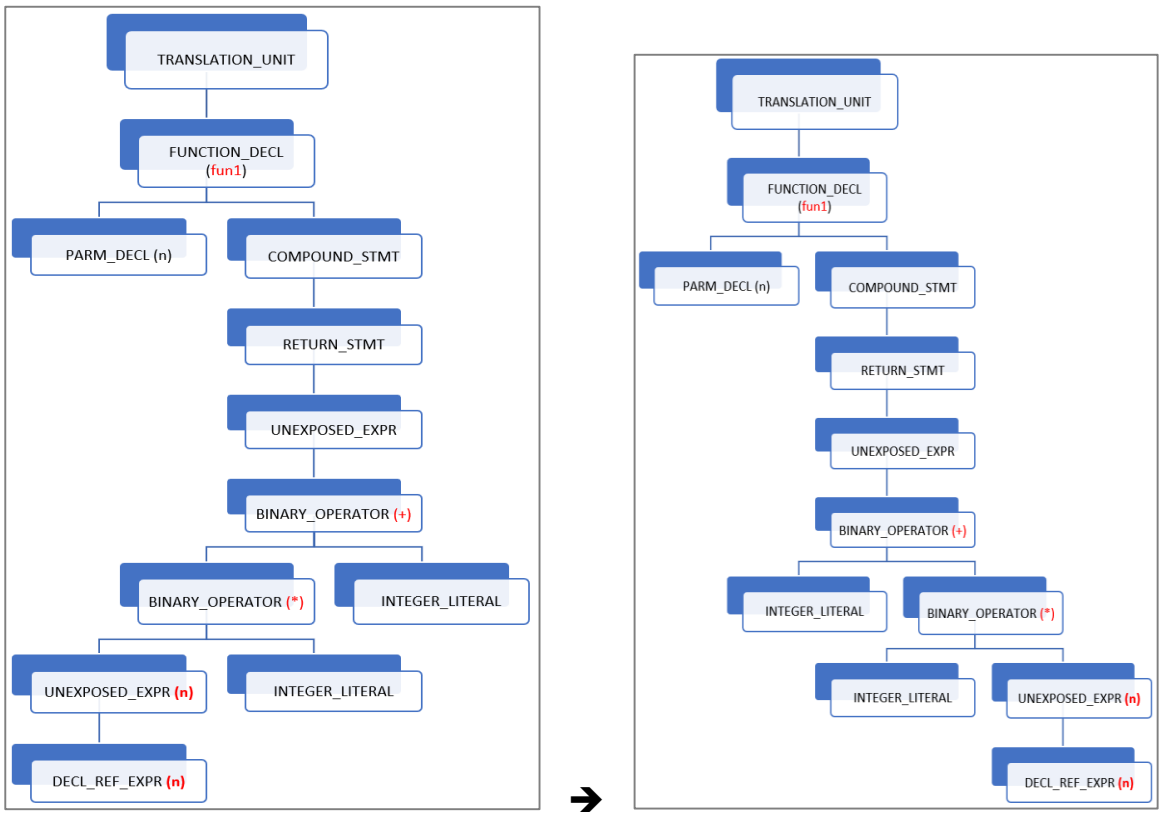
Một thủ thuật khác cũng tương đối dễ thực hiện và đảm bảo không làm thay đổi tính đúng đắn của chương trình là thay đổi thứ tự các toán hạng trong một biểu thức. Ví dụ thay đổi biểu thức trong mã nguồn sau:

```
void fun1(int n){
    return n * 8 + 3;
}
```

chuyển thành

```
void fun1(int n){
    return 3 + 8 * n;
}
```

Chúng ta thu được 2 cây AST tương ứng như sau:



Hình 1.6: Cây AST sinh ra sau khi thay đổi thứ tự các toán hạng

Có thể thấy rằng con của hai cây AST bắt nguồn từ node BINARY_OPERATOR (+) đã được đổi vị trí cho nhau (node trái → node phải, node phải → node trái). Tương tự như vậy với node BINARY_OPERATOR (*).

1.4.4 Thay đổi kiểu dữ liệu (data types)

Một thủ thuật khác thường được áp dụng là thay đổi các kiểu dữ liệu trong cùng nhóm. Việc thay đổi này hoàn toàn không làm thay đổi cấu trúc và type của node cây AST.

1.4.5 Thay thế giữa các biểu thức tương đương

Trong cây cú pháp AST, các loại toán tử này được biểu diễn tương ứng các loại node trong cây AST, các toán tử logic, toán tử quy chiếu,… được quy về chung loại BINARY_OPERATOR.

1.4.6 Bổ sung các đoạn mã nguồn không có giá trị (dead-code)

```
void func3(n){
    if ( n == 0 ) {
        return;
    }
}
```

chuyển thành

```
boolean debug = false ;
void func4(n){
    if ( n == 0 ) {
        if(debug){
            cout << "ERROR: ...";
        }
        return;
    }
}
```

```
├──TRANSLATION_UNIT test_data_2/test_8.cpp  [line=0, col=0]
   └──FUNCTION_DECL func3 void (int) [line=1, col=6]
      ├──PARM_DECL  int [line=1, col=13]
      └──COMPOUND_STMT   [line=1, col=14]
         └──IF_STMT   [line=2, col=5]
            └──COMPOUND_STMT   [line=2, col=19]
               └──RETURN_STMT   [line=3, col=9]
```

Hình 1.7: Cây AST trước khi thay thế biểu thức tương đương

```
├──TRANSLATION_UNIT test_data_2/test_8.cpp  [line=0, col=0]
   ├──VAR_DECL debug int [line=8, col=9]
   └──FUNCTION_DECL func4 void (int) [line=9, col=6]
      ├──PARM_DECL  int [line=9, col=13]
      └──COMPOUND_STMT   [line=9, col=14]
         └──IF_STMT   [line=10, col=5]
            └──COMPOUND_STMT   [line=10, col=19]
               ├──IF_STMT   [line=11, col=9]
               │  └──COMPOUND_STMT   [line=11, col=18]
               └──RETURN_STMT   [line=14, col=9]
```

Hình 1.8: Cây AST sau khi thay thế biểu thức tương đương

1.4.7 Thay đổi thứ tự của những đoạn mã nguồn độc lập

Hãy xem ví dụ dưới đây, chúng ta thay đổi thứ tự một thuộc tính và phương thức của lớp "Car" như sau:

```
class Rectangle {
    public:
        double width;
        double height;
        double getArea(){
            return width * height;
        }
        double getPerimeter(){
            return (width + height) * 2;
        }
};
```

Sau khi thay đổi thứ tự các thuộc tính và phương thức, đoạn mã chuyển thành:

```
class Rectangle {
    public:
        double height;
        double width;
        double getPerimeter(){
            return (width + height) * 2;
        }
        double getArea(){
            return width * height;
        }
};
```

Trong ví dụ trên, chỉ thứ tự của một số trường khai báo được thay đổi, nếu chúng ta thay đổi thứ tự của khai báo lớp hoặc khai báo phương thức, thì sự khác biệt giữa hai cây sẽ lớn hơn nhiều. Sự khác biệt giữa hai cây tùy thuộc vào loại cấu trúc được thay đổi vị trí.

1.4.8 Thay thế một câu lệnh lặp bằng một câu lệnh tương đương

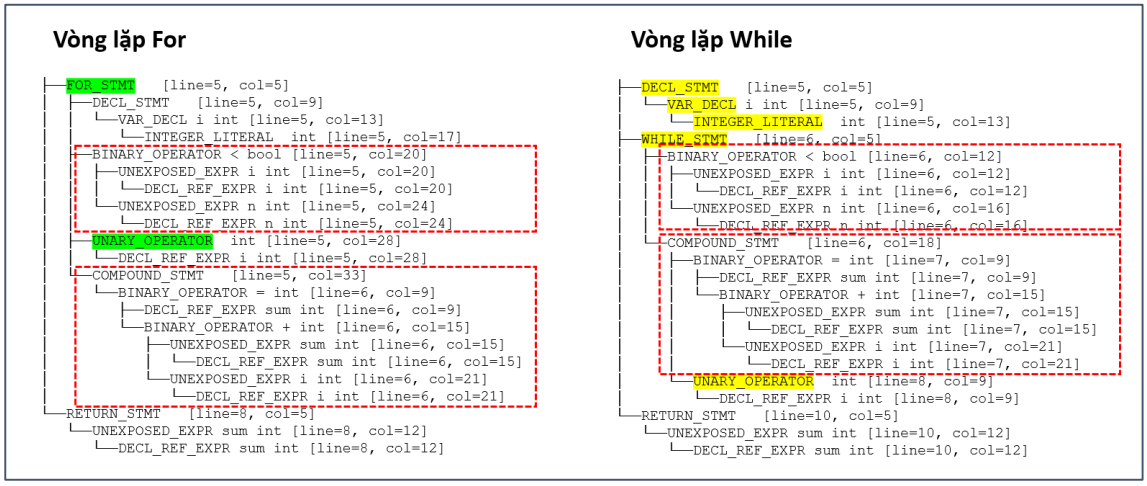
Vòng lặp *for* có thể được thay thế bằng vòng lặp *while (do-while hoặc while-do)* và ngược lại. Hãy xem xét mã trong ví dụ dưới đây:

```
int func1(int n){
    int sum = 0;
    for(int i = 0; i < n ; i++) {
        sum = sum + i;
    }
    return sum;
}
```

Thay thế vòng lặp *for* bằng vòng lặp *while* ta có:

```
int func1(int n){
    int sum = 0;
    int i = 0;
    while (i < n){
        sum = sum + i;
        i++;
    }
    return sum;
}
```

2 cây AST được sinh ra thông qua thư viện ast_viewer.py như sau:



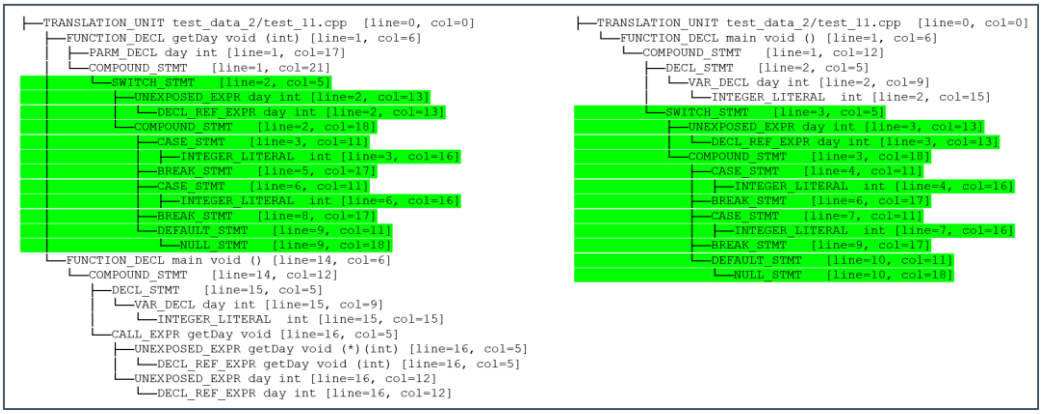
Hình 1.9: Cây AST sau khi thay thế vòng lặp for bằng vòng lặp while

1.4.9 Thay đổi các câu lệnh rẽ nhánh tương đương

Các câu lệnh *if* lồng nhau đôi khi có thể được thay thế bằng một chuỗi các câu lệnh *if*. Chúng ta cũng có thể sử dụng câu lệnh *if* thay vì câu lệnh *switch* hoặc ngược lại. Việc này không làm thay đổi cấu trúc cây.

1.4.10 Thay thế các lệnh gọi hàm bằng chính nội dung của hàm

Trong ví dụ ở trường hợp này, phần nội dung của hàm không quá lớn, nhưng đối với các mã nguồn khác, có thể có phần nội dung của hàm lớn hơn nhiều.



Hình 1.10: Cây AST sau khi thay thế lời gọi hàm bằng nội dung hàm

1.4.11 Kết hợp đoạn mã nguồn sao chép với mã nguồn tự viết

Việc sinh viên sử dụng lại mã nguồn một cách chủ động có vận dụng kiến thức của bản thân cũng giúp cho sinh viên hiểu bài hơn. Do đó, việc đưa ra thuật toán để đánh giá mức độ tương tự của 2 mã nguồn khi sử dụng thủ thuật này là không cần thiết.

1.5. Đánh giá sự khác nhau giữa các cây AST với mỗi thủ thuật sao chép

Sự khác biệt giữa 2 cây AST	**Thủ thuật**
Không thay đổi	- Thay đổi định dạng hoặc thêm/sửa các "comment code" - Đổi tên các định danh (hàm, tham số và biến)
Thay đổi nhỏ	- Thay đổi thứ tự các toán hạng trong biểu thức - Thay đổi kiểu dữ liệu (data types) - Thay thế giữa các biểu thức tương đương
Có thể thay đổi nhiều	- Bổ sung các đoạn code không có giá trị (deadcode) - Thay đổi thứ tự của những đoạn mã nguồn độc lập - Thay thế một câu lệnh lặp bằng một câu lệnh tương đương - Thay đổi các câu lệnh rẽ nhánh tương đương - Thay thế các lệnh gọi hàm bằng chính nội dung của hàm - Kết hợp đoạn mã nguồn sao chép với mã nguồn của bản thân

Bảng 1.3: Sự khác biệt giữa 2 cây AST tương ứng với các thủ thuật sao chép khác nhau

1.6. Kết luận chương 1

Kết thúc chương 1, luận văn đã nghiên cứu, giới thiệu tổng quan về vấn đề sao chép và sử dụng lại mã nguồn, các phương pháp đánh giá mức độ tương tự giữa các mã nguồn, trình bày tổng quan về cây cú pháp trừu tượng AST và ứng dụng thư viện Clang với ngôn ngữ Python để tạo ra cây AST từ mã nguồn viết bằng ngôn ngữ C/C++.

Trong chương này, luận văn cũng đã phân tích và đánh giá sự thay đổi trên cây AST tương ứng với các cách sao chép khác nhau. Sự thay đổi giữa các cây AST đi từ ít đến nhiều, đơn giản đến phức tạp khi mà phương thức sao chép mã nguồn trở nên phức tạp hơn.

Các vấn đề nghiên cứu ở chương 1, gợi ý về việc cần có bước tiền xử lý cây AST để tinh gọn và tối ưu trước khi được sử dụng. Sau bước tiền xử lý này, việc đánh giá độ tương đồng giữa 2 mã nguồn tương đương với việc đánh giá độ tương đồng giữa 2 cây AST.

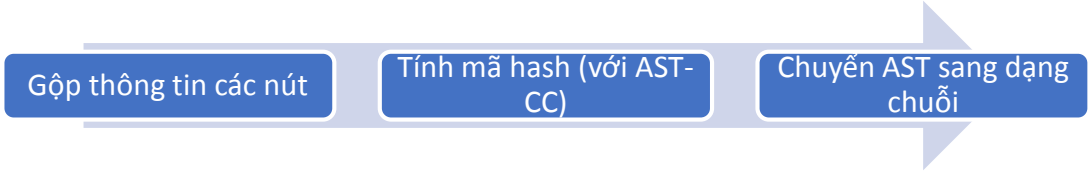
Trong chương tiếp theo, luận văn sẽ nghiên cứu các phương pháp đánh giá mức độ tương tự giữa các mã nguồn dựa trên dữ liệu từ cây AST.

2 CHƯƠNG 2: PHƯƠNG PHÁP ĐÁNH GIÁ ĐỘ TƯƠNG TỰ GIỮA CÁC MÃ NGUỒN

2.1. Tiền xử lý cây AST trước khi thực hiện đánh giá

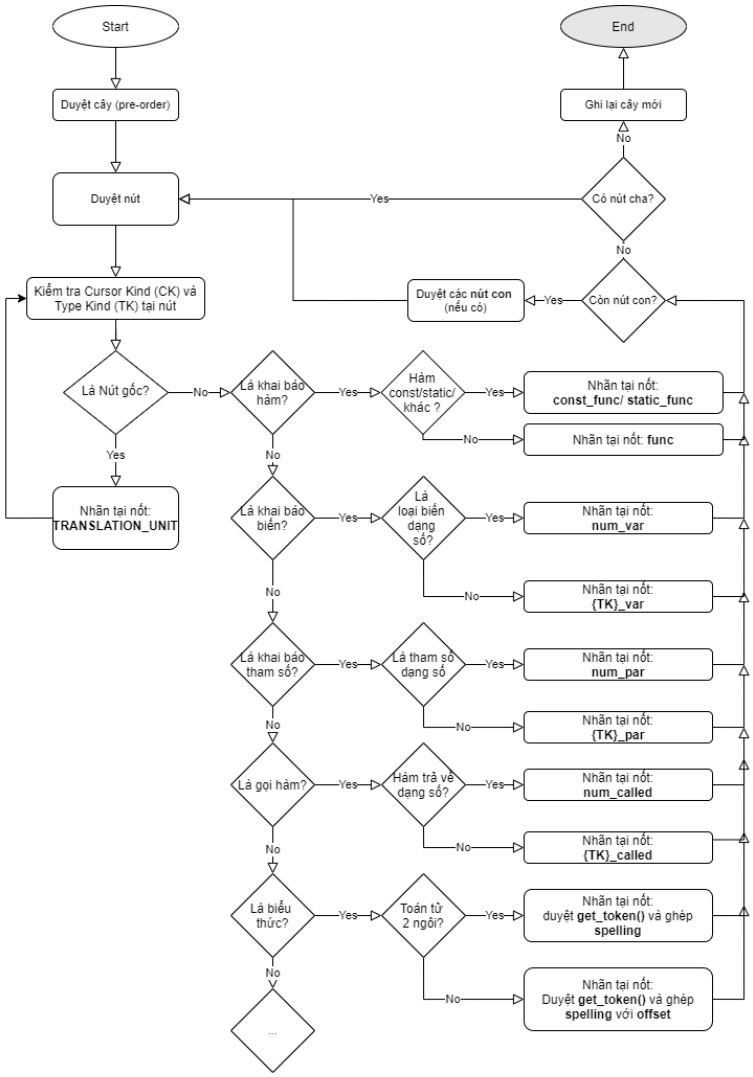
Hình 2.1: Các bước thực hiện so sánh giữa 2 mã nguồn

Công đoạn tiền xử lý cây AST cơ bản gồm 3 bước:



Hình 2.2: Các bước thực hiện tiền xử lý cây AST

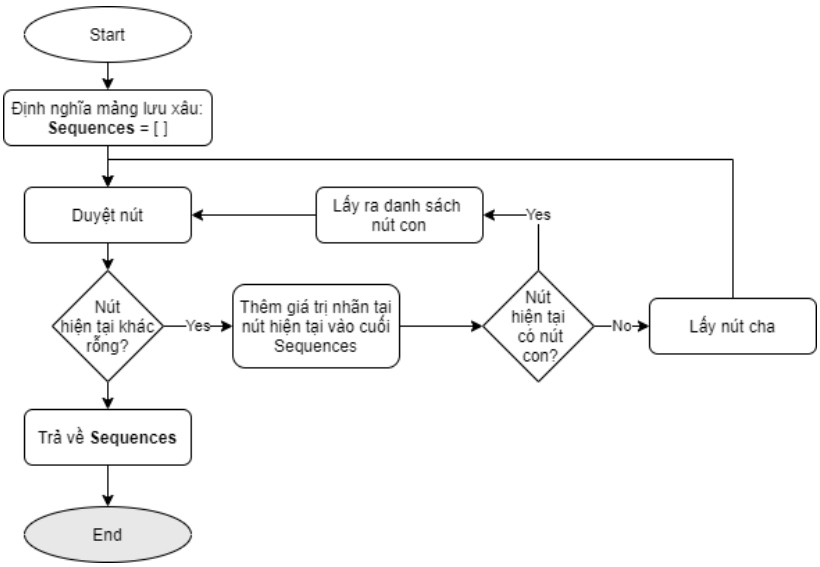
2.1.1 Gộp thông tin tại các nút



Hình 2.3: Tóm tắt giải thuật tiền xử lý cây AST

2.1.2 Chuyển AST sang dạng chuỗi (xâu)

Cây AST sau khi được tiền xử lý được duyệt đệ quy để ghép vào một mảng 1 chiều lưu các chuỗi giá trị tại các nút.



Hình 2.4: Giải thuật chuyển cây AST thành chuỗi

2.2. Các phương pháp đánh giá mức độ tương tự giữa các mã nguồn

2.2.1 Tìm xâu con chung dài nhất (LCS - Longest common subsequence)

2.2.1.1 Lý thuyết về LCS

Bài toán tìm xâu con chung dài nhất (LCS) (Cormen và cộng sự, 2001) được sử dụng khi chúng ta muốn tìm xâu con dài nhất giữa hai hoặc nhiều xâu (trong phạm vi nghiên cứu này, chỉ xét đến LCS giữa hai chuỗi).

2.2.1.2 Giải thuật phương pháp LCS

Mô tả: Bắt đầu từ vị trí L[n][m] và dừng lại khi L[i][j] == 0.

- Bắt đầu duyệt từ $i = n, j = m$. Phần tử thứ i của xâu a sẽ là $a[i-1]$, thứ j của xâu b sẽ là $b[j-1]$
- Nếu $a[i-1] == b[j-1]$ ta sẽ lưu lại con chung $a[i-1]$ và giảm i và j đi 1 đơn vị
- Nếu $a[i-1]! = b[j-1]$ có 2 trường hợp:
  - Nếu $\boldsymbol{L[i-1][j] \geq L[i][j-1]}$ thì giảm i đi 1 đơn vị
  - Ngược lại giảm **j** đi 1 đơn vị

2.2.1.3 Ứng dụng LCS để so sánh 2 cây AST

Gọi 2 cây AST lần lượt là $T_1 = (V_1, E_1)$ và $T_2 = (V_2, E_2)$ tương ứng với 2 cặp mã nguồn cần so sánh là p và q. Khi đó hàm $d(p,q)$ để đo mức độ tương tự giữa p và q được định nghĩa như sau:

$$d(p,q) = \frac{|V_1| + |V_2| - 2 * LCS(p,q)}{|V_1| + |V_2|}$$

Trong đó:

- $d(p,q)$ là hàm đo mức độ tương tự, có giá trị từ 0 → 1
- $|V_1|$ là số lượng đỉnh của cây T_1, $|V_2|$ là số lượng đỉnh của cây T_2
- $LCS(p,q)$ là độ dài xâu con dài nhất của 2 cây AST

2.2.1.4 Đánh giá hiệu quả sử dụng LCS

Ưu điểm chính của LCS là nó có thể tìm thấy điểm tương đồng giữa các cây con mà nhãn của các nút gốc khác nhau, chẳng hạn như cây con của các vòng lặp khác nhau hoặc cây con của các câu lệnh lựa chọn khác nhau.

2.2.2 TF-IDF và Độ tương tự Cosin

2.2.2.1 Lý thuyết về TF-IDF

TF-IDF (viết tắt của Term Frequency – Inverse Document Frequency)

TF là gì?

TF: Term Frequency (Tần suất xuất hiện của từ) là số lần từ xuất hiện trong văn bản. Công thức TF:

$$tf(t,d) = \frac{f(t,d)}{\max\{f(w,d)\ \forall w\ \in d\}}$$

Trong đó:

- $tf(t,d)$ tần suất xuất hiện của từ t trong văn bản ***d***
- $f(t,d)$ Số lần xuất hiện của từ t trong văn bản ***d***
- $\max\{f(w,d)\ \forall w\ \in d\}$ Số lần xuất hiện của từ có số lần xuất hiện nhiều nhất trong văn bản ***d***

IDF là gì?

IDF: Inverse Document Frequency (Nghịch đảo tần suất của văn bản), giúp đánh giá tầm quan trọng của một từ . Công thức tính IDF như sau:

$$idf(t,D) = log\frac{|D|}{|\{d \in D : t \in D\}|}$$

Trong đó:

- $idf(t,d)$ giá trị idf của từ t trong tập văn bản
- |D| tổng số văn bản trong tập D
- |{d ∈ D : t ∈ d}| thể hiện số văn bản trong tập D có chứa từ t.

$$tfidf(t,d,D) = tf(t,d) * idf(t,D)$$

2.2.2.2 Lập trình theo phương pháp TF-IDF

```
In [94]: def computeTF(wordDict, bow):
             tfDict = {}
             bowCount = len(bow)
             for word, count in wordDict.items():
                 tfDict[word] = count/float(bowCount)
             return tfDict
```

```
In [98]: def computeIDF(docList):
             import math
             idfDict = {}
             N = len(docList)

             idfDict = dict.fromkeys(docList[0].keys(), 0)
             for doc in docList:
                 for word, val in doc.items():
                     if val > 0:
                         idfDict[word] += 1

             for word, val in idfDict.items():
                 idfDict[word] = math.log10(N / float(val))

             return idfDict
```

```
In [100]: def computeTFIDF(tfBow, idfs):
              tfidf = {}
              for word, val in tfBow.items():
                  tfidf[word] = val*idfs[word]
              return tfidf
```

2.2.2.3 Lý thuyết về Mô hình không gian vector (Vector space model)

Mô hình không gian vector là một mô hình đại số biểu diễn các văn bản dưới dạng véc-tơ, các phần tử của véc-tơ này thể hiện mức độ quan trọng của một từ và cả sự xuất hiện hay không xuất hiện của nó trong một tài liệu.

2.2.2.4 Lý thuyết về Độ tương tự Cosin (Cosine similarity)

Độ tương tự Cosin là một cách đo ***độ tương tự*** giữa 2 vectơ khác 0 trong một không gian véc-tơ đã được chuẩn hóa. Độ tương tự này được định nghĩa bằng giá trị cosine của góc giữa hai véc-tơ, và cũng là tích vô hướng của cùng các véc-tơ đơn vị để cả hai đều có chiều dài 1. Giá trị cosine của 0° là 1, và bé hơn 1 với bất kỳ góc nào trong khoảng các radian (0, π].

Sau khi đã biểu diễn 2 văn bản dưới dạng véc-tơ thì có thể sử dụng phép đo Độ tương tự Cosin để đánh giá mức tương đồng giữa 2 văn bản này.

2.2.2.5 Ứng dụng TF-IDF và Độ tương tự Cosin để so sánh 2 cây AST

Sau bước tiền xử lý ở mục 1.3.4, cây AST của mã nguồn đã được chuyển thành dạng chuỗi, tức là 2 văn bản, với nội dung là nhãn của các node của cây AST. Để so sánh độ tương đồng giữa 2 văn bản này, chúng ta sử dụng độ tương tự TF-IDF và Độ tương tự Cosin để so sánh mức độ tương đồng. Nếu độ tương tự cosin cho kết quả càng gần 1, càng cho thấy 2 cây AST có mức tương đồng cao.

2.2.2.6 Đánh giá hiệu quả sử dụng TF-IDF và Độ tương tự Cosin

Khi áp dụng TF-IDF và độ tương tự Cosin vào đánh giá độ tương tự giữa 2 cây AST, có một số đặc điểm như sau:

- Đánh giá độ tương tự dựa trên tần suất và độ quan trọng của giá trị tại các nút
- Ít bị ảnh hưởng bởi sự thay đổi thứ tự các đoạn mã nguồn (trong khi phương pháp LCS trình bày ở mục 1.3.5 bị ảnh hưởng nhiều)
- Có thể gây ra tỉ lệ dương tính giả cao nếu đặt n-gram không hợp lý.

2.2.3 AST-CC (AST Code Comparison)

2.2.3.1 Tổng quan về AST-CC

Vào năm 2015, tại Hội nghị quốc tế lần thứ 10 về Máy tính, Truyền thông và Ứng dụng Băng thông rộng và Không dây, nhóm tác giả Jingling Zhao, Kunfeng Xia, Yilun Fu đến từ Đại học Bưu chính Viễn thông Bắc Kinh, Bắc Kinh; Phòng thí nghiệm Kỹ thuật Quốc gia về An ninh Mạng Di động; Viện nghiên cứu điện lực Trung Quốc, Bắc Kinh; đã giới thiệu một thuật toán phát hiện sao chép mã nguồn đáng tin cậy dựa trên cây AST.

Thuật toán được nhóm tác giả đặt tên là AST-CC (AST Code Comparison).

Ý tưởng của thuật toán như sau:

Sau khi có được cấu trúc cây cú pháp của mã nguồn, việc cần làm là so sánh cây cú pháp. Tuy nhiên trong thực tế, các cây cú pháp đều khá phức tạp và có kích thước lớn nên việc so sánh trực tiếp giữa các cây cú pháp là khó và hiệu quả sẽ thấp. Vì vậy, nhóm tác giả sử dụng giá trị băm của cây cú pháp và tính toán giá trị dựa trên loại của mỗi nút và các cây

con của cây AST. Sau đó so sánh các giá trị băm này. Nhờ đó, độ khó so sánh được giảm bớt và thông tin của cây cú pháp không bị mất.

2.2.3.2 Giải thuật theo phương pháp AST-CC

Giải thuật AST-CC có 3 bước chính:

- Đọc thông tin của mã nguồn cần kiểm tra từ HashListArray tại vị trí có chỉ số bằng với ngưỡng (tức là số lượng nút con bằng giá trị ngưỡng, vì lưu trữ thông tin của cây con với n nút con trong HashListArray[n]), và sau đó so sánh với mã gốc.
- Duyệt qua các mảng của SuspectHashListArray và OriginalHashListArray lưu trữ thông tin của mã cần kiểm tra và mã gốc tương ứng. Sau đó, sắp xếp hai mảng này và đảm bảo rằng chuỗi của các nút được lưu trữ theo số nút và giá trị băm.
- So sánh giá trị băm của các cây con có cùng số nút con. Nếu chúng chia sẻ cùng một giá trị băm, thì hãy lưu trữ vị trí của chúng trong các tệp nguồn vào cơ sở dữ liệu.

2.3. Kết luận chương 2

Kết thúc chương 2, luận văn đã trình bày được về tổng quan về sự thay đổi giữa các cây AST khi áp dụng các thủ thuật sao chép mã nguồn khác nhau, áp dụng các phương pháp LCS, TF-IDF, AST_CC trong việc so sánh đánh giá các mã nguồn. Trong chương tiếp theo luận văn tập trung xây dựng, thử nghiệm hệ thống đánh giá mức độ tương tự giữa các mã nguồn theo 3 giải thuật đã trình bày ở chương 2.

3 CHƯƠNG 3: THỰC NGHIỆM VÀ ĐÁNH GIÁ

3.1. Cài đặt hệ thống để so sánh độ tương tự giữa các mã nguồn

3.1.1 Cài đặt Python

Bước 1: Tải về bộ cài đặt Python 3 cho hệ điều hành windows

Với đặc thù của bài toán, chúng ta chọn phiên bản Python được đánh giá ổn định là Python 3.7

Bước 2: Cài đặt Python 3 trên Hệ điều hành Windows

Bước 3: Kiểm tra phiên bản Python đã cài đặt thành công

Bước 4: Kiểm tra thư viện pip (Trình quản lý các gói thư viện của Python) đã được cài đặt hay chưa.

3.1.2 Cài đặt thư viện Clang

Tài liệu hướng dẫn cài đặt thư viện Clang trên nền ngôn ngữ lập trình Python tại đây: https://pypi.org/project/clang/

Bước 1: Mở terminal trên máy cần cài đặt

Bước 2: Gõ lệnh cài đặt clang:

```
pip install clang
```

Bước 3: Gõ lệnh cài đặt libclang

```
pip install libclang
```

Bước 4: Kiểm tra kết quả cài đặt:

```
pip freeze | grep clang
```

3.2. Xây dựng công cụ so sánh mã nguồn với 3 phương pháp LCS, TF-IDF, AST-CC

Công cụ so sánh mã nguồn gồm các script thực hiện các nhiệm vụ độc lập như sau:

Bảng 3.1: Bảng danh sách các script thực hiện so sánh mức tương tự giữa các mã nguồn

STT	Tên file	Ý nghĩa
1	ast_viewer.py	Hiển thị cây AST của một mã nguồn trên terminal
2	reduceAST.py	Tiền xử lý AST của một mã nguồn trước khi so sánh
3	similarity.py	Hàm so sánh 2 trực tiếp 2 mã nguồn và trả về mức độ tương tự (lựa chọn giữa 3 giải thuật LCS, TF-IDF, AST-CC)
3	compare_ two_code.py	Giao diện cho phép so sánh trực tiếp 2 mã nguồn, hiển thị chi tiết mức độ tương tự, các phần mã nguồn được xác định tương tự (lựa chọn giữa 3 giải thuật LCS, TF-IDF, AST-CC)
4	main.py	Hàm chính chạy so sánh giữa 1 tập dữ liệu là N file mã nguồn trong thư mục test_data. Kết quả trả về là một bảng NxN, trong đó cho thấy mức độ tương tự giữa các cặp file mã nguồn.

3.3. Thực nghiệm hệ thống với dữ liệu đầu vào đã gán nhãn (phân loại)

Dữ liệu đầu vào đã phân loại gồm 16 mã nguồn gồm 10 mã nguồn đã được gán nhãn (phân loại) theo thủ thuật sao chép và 6 mã nguồn không liên quan để đảm bảo tránh trường hợp dương tính giả (False Positive)

Name	Date modified	Type	Size
00-original.cpp	3/27/2022 4:06 PM	C++ source file	3 KB
01-variables-and-types-changed.cpp	3/27/2022 4:06 PM	C++ source file	4 KB
02-order-changed.cpp	4/16/2022 4:34 PM	C++ source file	3 KB
03-dead-code.cpp	3/27/2022 4:06 PM	C++ source file	3 KB
04-moderately-plagiarized.cpp	3/27/2022 4:06 PM	C++ source file	2 KB
05-heavily-plagiarized.cpp	3/27/2022 4:06 PM	C++ source file	3 KB
06-different-approach.cpp	3/27/2022 4:06 PM	C++ source file	2 KB
07-another-approach.cpp	3/27/2022 4:06 PM	C++ source file	2 KB
08-somewhat-related-dfs.cpp	3/27/2022 4:06 PM	C++ source file	2 KB
09-somewhat-related-bfs.cpp	3/27/2022 4:06 PM	C++ source file	2 KB
10-unrelated-quicksort.cpp	3/27/2022 4:06 PM	C++ source file	2 KB
11-unrelated-mergesort.cpp	3/27/2022 4:06 PM	C++ source file	2 KB
12-unrelated-fibonacci-recursive.cpp	3/27/2022 4:06 PM	C++ source file	1 KB
13-unrelated-fibonacci-matrix.cpp	3/27/2022 4:06 PM	C++ source file	2 KB
14-unrelated-fibonacci-memoization.cpp	3/27/2022 4:06 PM	C++ source file	1 KB
15-unrelated-fibonacci-dynamic.cpp	3/27/2022 4:06 PM	C++ source file	1 KB

Hình 3.1: Bộ mã nguồn đã gán nhãn

3.3.1 Sử dụng phương pháp LCS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	0.76	0.86	0.61	0.91	0.26	0.31	0.39	0.38	0.43	0.4	0.22	0.27	0.26	0.25
2	1	1	0.76	0.86	0.61	0.91	0.26	0.31	0.39	0.38	0.43	0.4	0.22	0.27	0.26	0.25
3	0.76	0.76	1	0.67	0.52	0.74	0.26	0.3	0.39	0.35	0.41	0.4	0.2	0.28	0.27	0.25
4	0.86	0.86	0.67	1	0.55	0.78	0.21	0.26	0.34	0.33	0.39	0.41	0.27	0.26	0.24	0.21
5	0.61	0.61	0.52	0.55	1	0.61	0.26	0.34	0.35	0.32	0.43	0.4	0.22	0.22	0.29	0.31
6	0.91	0.91	0.74	0.78	0.61	1	0.3	0.34	0.42	0.39	0.45	0.41	0.22	0.26	0.28	0.28
7	0.26	0.26	0.26	0.21	0.26	0.3	1	0.53	0.34	0.39	0.23	0.17	0.39	0.13	0.26	0.4
8	0.31	0.31	0.3	0.26	0.34	0.34	0.53	1	0.38	0.35	0.29	0.25	0.37	0.18	0.33	0.34
9	0.39	0.39	0.39	0.34	0.35	0.42	0.34	0.38	1	0.82	0.37	0.33	0.27	0.23	0.3	0.35
10	0.38	0.38	0.35	0.33	0.32	0.39	0.39	0.35	0.82	1	0.34	0.3	0.25	0.23	0.29	0.34
11	0.43	0.43	0.41	0.39	0.43	0.45	0.23	0.29	0.37	0.34	1	0.58	0.31	0.26	0.36	0.32
12	0.4	0.4	0.4	0.41	0.4	0.41	0.17	0.25	0.33	0.3	0.58	1	0.2	0.28	0.3	0.31
13	0.22	0.22	0.2	0.27	0.22	0.22	0.39	0.37	0.27	0.25	0.31	0.2	1	0.2	0.41	0.4
14	0.27	0.27	0.28	0.26	0.22	0.26	0.13	0.18	0.23	0.23	0.26	0.28	0.2	1	0.29	0.23
15	0.26	0.26	0.27	0.24	0.29	0.28	0.26	0.33	0.3	0.29	0.36	0.3	0.41	0.29	1	0.51
16	0.25	0.25	0.25	0.21	0.31	0.28	0.4	0.34	0.35	0.34	0.32	0.31	0.4	0.23	0.51	1

Hình 3.2: Kết quả phương pháp LCS so sánh giữa 16 mã nguồn đã gán nhãn

3.3.2 Sử dụng phương pháp TF-IDF

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	0.94	0.87	0.69	0.97	0.38	0.32	0.43	0.42	0.42	0.45	0.23	0.3	0.22	0.26
2	1	1	0.94	0.87	0.69	0.97	0.38	0.32	0.43	0.42	0.42	0.45	0.23	0.3	0.22	0.26
3	0.94	0.94	1	0.85	0.68	0.92	0.38	0.33	0.44	0.42	0.42	0.44	0.23	0.3	0.22	0.25
4	0.87	0.87	0.85	1	0.65	0.85	0.37	0.32	0.42	0.39	0.53	0.47	0.43	0.33	0.31	0.3
5	0.69	0.69	0.68	0.65	1	0.68	0.37	0.33	0.42	0.37	0.34	0.42	0.23	0.23	0.21	0.34
6	0.97	0.97	0.92	0.85	0.68	1	0.39	0.33	0.43	0.41	0.41	0.45	0.22	0.31	0.22	0.26
7	0.38	0.38	0.38	0.37	0.37	0.39	1	0.63	0.34	0.35	0.26	0.3	0.32	0.19	0.19	0.25
8	0.32	0.32	0.33	0.32	0.33	0.33	0.63	1	0.29	0.29	0.26	0.31	0.29	0.17	0.19	0.26
9	0.43	0.43	0.44	0.42	0.42	0.43	0.34	0.29	1	0.8	0.33	0.34	0.2	0.26	0.19	0.29
10	0.42	0.42	0.42	0.39	0.37	0.41	0.35	0.29	0.8	1	0.3	0.31	0.23	0.27	0.19	0.31
11	0.42	0.42	0.42	0.53	0.34	0.41	0.26	0.26	0.33	0.3	1	0.66	0.36	0.36	0.37	0.33
12	0.45	0.45	0.44	0.47	0.42	0.45	0.3	0.31	0.34	0.31	0.66	1	0.34	0.37	0.34	0.45
13	0.23	0.23	0.23	0.43	0.23	0.22	0.32	0.29	0.2	0.23	0.36	0.34	1	0.23	0.37	0.29
14	0.3	0.3	0.3	0.33	0.23	0.31	0.19	0.17	0.26	0.27	0.36	0.37	0.23	1	0.39	0.39
15	0.22	0.22	0.22	0.31	0.21	0.22	0.19	0.19	0.19	0.19	0.37	0.34	0.37	0.39	1	0.41
16	0.26	0.26	0.25	0.3	0.34	0.26	0.25	0.26	0.29	0.31	0.33	0.45	0.29	0.39	0.41	1

Hình 3.3: Kết quả phương pháp TF-IDF so sánh giữa 16 mã nguồn đã gán nhãn

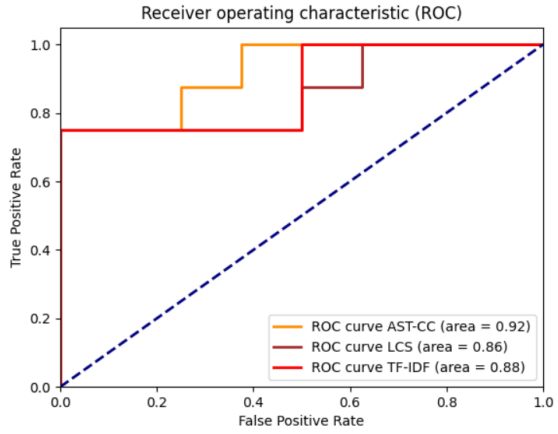
3.3.3 Sử dụng phương pháp AST-CC

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	1	0.82	0.29	0.44	0.1	0.13	0.12	0.1	0.18	0.22	0.08	0.05	0.05	0.05
2	1	1	1	0.82	0.29	0.44	0.1	0.13	0.12	0.1	0.18	0.22	0.08	0.05	0.05	0.05
3	1	1	1	0.82	0.29	0.44	0.1	0.13	0.12	0.1	0.18	0.22	0.08	0.05	0.05	0.05
4	0.82	0.82	0.82	1	0.23	0.37	0.08	0.1	0.1	0.08	0.15	0.24	0.21	0.08	0.09	0.1
5	0.29	0.29	0.29	0.23	1	0.28	0.11	0.14	0.13	0.11	0.17	0.21	0.09	0.03	0.03	0.08
6	0.44	0.44	0.44	0.37	0.28	1	0.14	0.14	0.13	0.11	0.17	0.22	0.09	0.06	0.06	0.06
7	0.1	0.1	0.1	0.08	0.11	0.14	1	0.23	0.13	0.11	0.08	0.09	0.23	0.05	0.07	0.14
8	0.13	0.13	0.13	0.1	0.14	0.14	0.23	1	0.13	0.09	0.12	0.12	0.17	0.04	0.06	0.1
9	0.12	0.12	0.12	0.1	0.13	0.13	0.13	0.13	1	0.33	0.08	0.09	0.09	0.04	0.05	0.08
10	0.1	0.1	0.1	0.08	0.11	0.11	0.11	0.09	0.33	1	0.04	0.04	0.1	0.04	0.05	0.09
11	0.18	0.18	0.18	0.15	0.17	0.17	0.08	0.12	0.08	0.04	1	0.31	0.14	0.07	0.16	0.16
12	0.22	0.22	0.22	0.24	0.21	0.22	0.09	0.12	0.09	0.04	0.31	1	0.09	0.06	0.09	0.14
13	0.08	0.08	0.08	0.21	0.09	0.09	0.23	0.17	0.09	0.1	0.14	0.09	1	0.1	0.2	0.24
14	0.05	0.05	0.05	0.08	0.03	0.06	0.05	0.04	0.04	0.04	0.07	0.06	0.1	1	0.05	0.12
15	0.05	0.05	0.05	0.09	0.03	0.06	0.07	0.06	0.05	0.05	0.16	0.09	0.2	0.05	1	0.15
16	0.05	0.05	0.05	0.1	0.08	0.06	0.14	0.1	0.08	0.09	0.16	0.14	0.24	0.12	0.15	1

Hình 3.4: Kết quả phương pháp AST-CC so sánh giữa 16 mã nguồn đã gán nhãn

3.3.4 Nhận xét, đánh giá

Dựa trên kết quả dữ liệu đã gán nhãn đầu vào, đường cong ROC được vẽ bởi 3 phương pháp LCS, TF-IDF và AST-CC được biểu diễn như sau:



Hình 3.5: Kết quả phương pháp LCS so sánh giữa mã nguồn chưa phân loại

Kết quả đánh giá từ phần thực nghiệm từ dữ liệu đã phân loại (đã gán nhãn) cho thấy một số nhận xét như sau:

- Với các thủ thuật thông thường như thay đổi format mã nguồn, thay đổi tên hàm, tên biến, thay đổi kiểu biến. Cả 3 giải thuật hoạt động tốt.
- Với thủ thuật thay đổi thứ tự mã nguồn, bổ sung mã nguồn dư thừa.
    - LCS không còn nhạy, trả về độ tương tự thấp (âm tính giả).
    - TF-IDF và ASC-CC vẫn thể hiện độ nhạy cao, tính đáp ứng tương đối tốt, khi trả về độ tương tự cao (trên 0,8)
- Với việc triển khai mã nguồn theo hướng tiếp cận mới:
    - LCS và TF-IDF trả về độ tương tự cao
    - AST-CC trong khi đó trả về độ tương tự thấp
- Khi so sánh với nhóm mã nguồn hoàn toàn không liên quan từ 10 cho đến 16:
    - LCS và TF-IDF cho thấy khả năng gây ra dương tính giả cao.

- AST-CC vẫn hoạt động tốt khi cho mức độ tương tự thấp.

3.4. Thực nghiệm hệ thống với dữ liệu đầu vào chưa phân loại

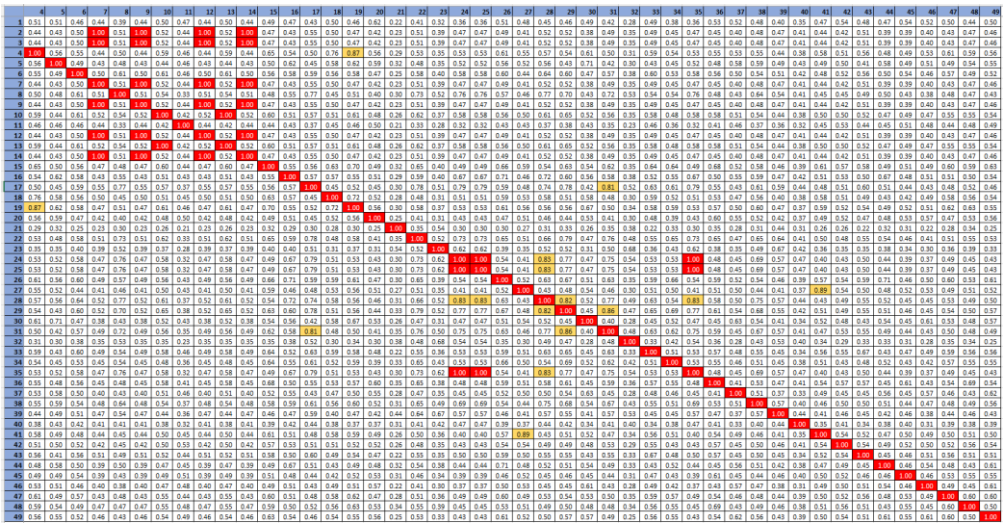
Dữ liệu đầu vào chưa phân loại gồm 49 mã nguồn là các bài làm của sinh viên trên hệ thống *code.ptit.edu.vn* cho bài tập sau:

Một xâu nhị phân độ dài *n* được gọi là thuận nghịch hay đối xứng nếu đảo ngược xâu nhị phân đó ta vẫn nhận được chính nó. Cho số tự nhiên *n* (n nhập từ bàn phím). Hãy viết chương trình liệt kê tất cả các xâu nhị phân thuận nghịch có độ dài *n*. Hai phần tử khác nhau của xâu thuận nghịch được ghi cách nhau một khoảng trống. Ví dụ với n = 4 ta tìm được 4 xâu nhị phân thuận nghịch như dưới đây. **Ví dụ**

Input	Output
4	0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 1

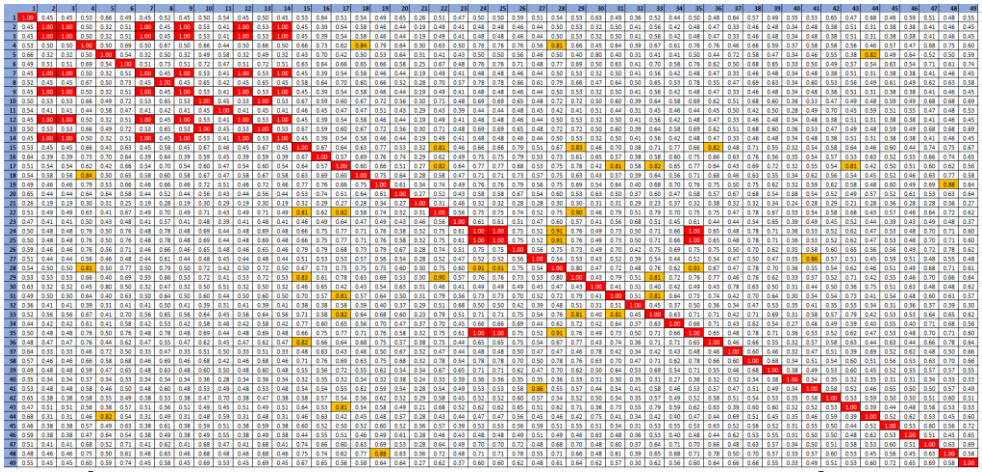
Bài tập thực hành Sinh nhị phân trên

3.4.1 Sử dụng phương pháp LCS



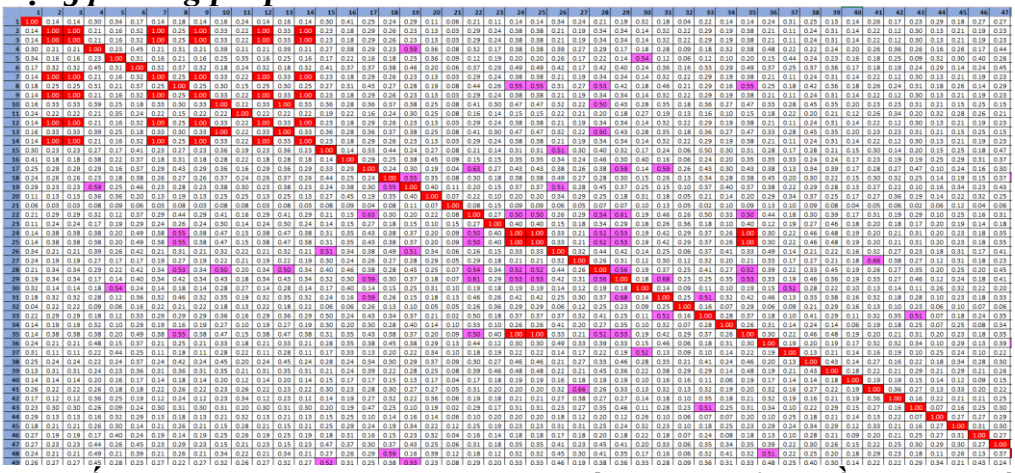
Hình 3.6: Kết quả phương pháp LCS so sánh giữa mã nguồn chưa phân loại

3.4.2 Sử dụng phương pháp TF-IDF



Hình 3.7: Kết quả phương pháp TF-IDF so sánh giữa mã nguồn chưa phân loại

3.4.3 *Sử dụng phương pháp AST-CC*



Hình 3.8: Kết quả phương pháp AST-CC so sánh giữa mã nguồn chưa phân loại

3.4.4 *Nhận xét, đánh giá*

Kết hợp kết quả đánh giá từ phần thực nghiệm từ dữ liệu chưa phân loại và việc kiểm tra nội dung các cặp file mã nguồn có độ tương tự cao (từ 0.8 trở lên) cho thấy như sau:

Nói chung 3 giải thuật cho thấy các ưu điểm và nhược điểm khác nhau, đối với bài toán tính toán mức độ trùng lặp giữa các bài tập cho sinh viên thực hiện trên hệ thống *code.ptit.edu.vn*, việc cần hạn chế dương tính giả là phù hợp do tính chất nhạy cảm của việc đánh giá. Do đó đề xuất áp dụng giải thuật AST-CC để đánh giá mức độ tương đồng giữa các giải thuật của sinh viên nộp trên hệ thống, cụ thể như sau:

Hình 3.9: Đề xuất áp dụng AST-CC trên hệ thống code.ptit.edu.vn

3.5. Kết luận chương 3

Kết thúc chương 3, luận văn đã thực hiện được một số nội dung sau:

- Cài đặt Python và cài đặt thư viện Clang
- Viết mã nguồn sử dụng các giải thuật LCS, TF-IDF, AST-CC
- Thực nghiệm trên dữ liệu đầu vào đã gãn nhán
- Thực nghiệm trên dữ liệu đầu vào chưa gãn nhán
- So sánh - đánh giá kết quả giữa các giải thuật
- Đề xuất phương án phù hợp để áp dụng trên hệ thống *code.ptit.edu.vn*

KẾT LUẬN

Luận văn này đã nghiên cứu, thực hiện và đạt được một số kết quả như sau:

- Nghiên cứu tổng quan về vấn đề sao chép mã nguồn, trình bày các thủ thuật sao chép mã nguồn phổ biến, các phương pháp đánh giá mức độ tương tự giữa các mã nguồn.

- Nghiên cứu tổng quan về vấn đề cây cú pháp trừu tượng AST và ứng dụng thư viện CLang kết hợp với ngôn ngữ lập trình Python để phân tích mã nguồn viết trên ngôn ngữ C/C++.

- Nghiên cứu và đánh giá những điểm khác và giống nhau giữa các cây AST trên mã nguồn gốc và mã nguồn cần đánh giá tương ứng với các thủ thuật sao chép khác nhau.

- Nghiên cứu và đề xuất phương pháp tiền xử lý cây AST để giảm độ phức tạp, tăng hiệu năng trước khi thực hiện so sánh mã nguồn.

- Nghiên cứu và ứng dụng phương pháp tìm chuỗi con chung dài nhất (LCS) vào việc so sánh độ tương tự giữa hai mã nguồn.

- Nghiên cứu và ứng dụng phương pháp TF-IDF vào việc so sánh độ tương tự giữa hai mã nguồn.

- Nghiên cứu và ứng dụng phương pháp AST-CC vào việc so sánh độ tương tự giữa hai mã nguồn.

- Nghiên cứu, xây dựng, thử nghiệm và đánh giá khi áp dụng các phương pháp LCS, TF-IDF, AST-CC đối với dữ liệu trích xuất từ bài làm của các sinh viên trên hệ thống *code.ptit.edu.vn*;

Trong tương lai, luận văn có thể được tiếp tục nghiên cứu theo hướng xử lý loại bỏ các đoạn mã nguồn dư thừa trên cây AST trước khi thực hiện việc so sánh để tăng độ chính xác. Ngoài ra có thể xây dựng công cụ đóng gói và cho phép tích hợp trực tiếp trên hệ thống code.ptit.edu.vn và các hệ thống Lab thực hành để đưa ra khuyến nghị, cảnh báo cho Giảng viên và Sinh viên khi sử dụng để ôn tập, kiểm tra.