2018 전기 졸업 과제 최종 보고서

소스 코드 표절 탐지 및 추적 시스템

‘명탐정 한이’

지도 교수: 조환규 교수님

소속: 전기컴퓨터공학부

팀 이름: 달려라 한이

작성자: 201524551 이하원

201524617 한성혜

목 차

1. 과제 소개 3

1.1 과제 개요 3

1.2 과제 목표 3

2. 대상 문제 분석 및 요구조건 분석 4

2.1 대상 문제 분석 4

2.2 요구 조건 분석 4

3. 과제 수행 환경 5

3.1 개발 환경 5

3.2 개발 자원 5

4. 수행 내용 6

4.1 시스템 흐름도 6

4.2 program DNA 생성 6

4.3 유사도 계산 8

4.4 유사도 결과 시각화 11

5. 수행 결과 14

5.1 개발 결과 14

5.2 타 시스템과 비교 19

6. 산업체 멘토링 결과 반영 내용 21

7. 과제 추진 체계 22

7.1 계획표 22

7.2 역할 분담 22

1 과제 소개

1.1 과제 개요

컴퓨터 공학 학생에게 코딩 과제는 개발자로서의 역량을 기르는 중요한 역할을 한다. 하지만 과제 검사 시 변수 이름조차 바꾸지 않은 표절한 코드를 쉽게 발견할 수 있다. 이러한 표절 행위는 개발자로서의 실력을 향상할 수 없게 만들 뿐만 아니라 노력에 맞는 정당한 점수 분배를 어렵게 한다. 본 논문에서는 이러한 상황을 개선하기 위해 함수 호출에 유의하여 토큰을 생성하는 실행 순서 추출 기법과 토큰의 등장 횟수를 고려하여 유사치를 내는 토큰 빈도 가중치를 적용한 소스 코드 표절 탐색 시스템을 제안하며 이름은 '명탐정 한이'라 한다.

1.2 과제 목표

- 학생들이 제출한 C/C++, JAVA 소스 코드를 받고, 소스 코드 쌍의 표절 유사도를 계산한다.
- 각 소스 코드 쌍의 유사도를 Phylogenetic Analysis 등의 방법을 통해 시각화 한다.

2 대상 문제 분석 및 요구조건 분석

2.1 대상 문제 분석

No.	공격 방법
A1	변수 이름이나 함수 이름을 변경한다.
A2	코드에 영향을 끼치지 않는 선에서 변수 선언 위치를 변경 한다.
A3	무의미한 연산을 추가하여 표절한 내용이 쉽게 눈에 띄지 않게 어지럽힌다.
A4	호출하지 않는 함수를 추가하여 마치 다른 프로그램인 것처럼 가장한다.
A5	하나의 함수를 두 개 이상의 함수로 분리 하거나 여러 함수를 하나의 함수로 결합하여 사용된 함수의 개수나 형태를 다르게 보이게 한다.
A6	for문을 while문으로 바꾸거나 제어 구조를 다르게 표시함으로써 다른 프로그램인 것처럼 가장 한다.
A7	프로그램 전체가 같은 기능이 되도록 새로 작성한다.
A8	프로그래밍 언어를 바꾼다. 예를 들어 c를 java로, java를 python으로 바꾸는 작업.

표 1. 표절 기법 유형 8가지

학생들이 사용하는 표절 기법을 8가지 유형으로 정리했다. A7,A8 방법을 사용하기 위해서는 코드에 대한 이해와 많은 시간을 필요로 한다. 따라서 이 두 수법을 표절한 것으로 보아야 할지는 확실 하지 않으며 표절 탐색 프로그램의 성능을 평가하기 위해서는 A7,A8을 제외한 A1~A6의 공격 방법을 잘 찾아내는지를 평가 한다.

2.2 요구조건 분석

1) C/C++, JAVA 소스 코드의 유사도 계산
2) 지속적인 표절 그룹 추적 시스템
3) 표절 탐색 결과 유사도 Phylogenetic Analysis 등을 통한 시각화
4) 사용자 (교수, 조교 등)들이 쉽게 사용할 수 있는 UI 구현

3 과제 수행 환경

3.1 개발환경

3.1.1. 대상 시스템

-OS : Ubuntu 16.04.3

-apache server

3.1.2. 개발 언어

Back end

-C (gcc 5.4.0)

-lexer

-C++

-python2.7.12

Front end

-html, css, ajax

-node js

3.2 개발 자원

3.2.1. 부산대학교 정보컴퓨터공학과 알고리즘 수업 과제물 (2016~2018 년도)

총 30개의 과제로 각 과제당 평균 길이는 130line이다. 오픈소스를 허락한 과제의 경우 평균 200line이 넘는 경우도 있었다. 2016, 2017년도는 C/C++ 코드가95%이상 이였으며 2018년도에는 python 코드가 25% 차지했다. 학생들이 제출한 이 소스 코드 데이터를 통해 표절 데이터를 제작해 시스템의 성능을 평가 한다.

3.2.2. SOCO data set

멕시코의 MAX(Metropolitan Autonomous University)에서 제작한 검증된 소스 코드 표절 데이터로, c와 c++ 코드 130개로 구성된다. 위의 알고리즘 과제 data set과 달리 하나의 data set은 같은 수행을 하는 코드가 아닌 다른 수행을 하는 다양한 크기의 코드로 이루어져 있다. 또한 코드가 단순 print하는 의미 없는 코드가 많기 때문에 이를 분리하는 작업이 필요하다.

4. 설계 및 구현 상세

4.1 시스템 흐름도

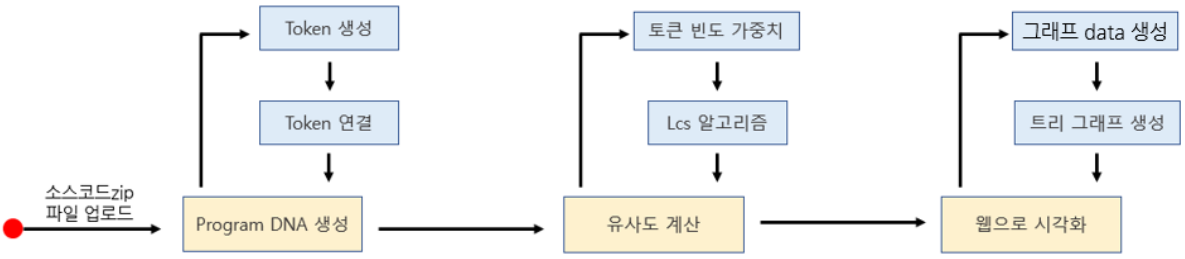


그림 1. [전체 시스템 흐름도]

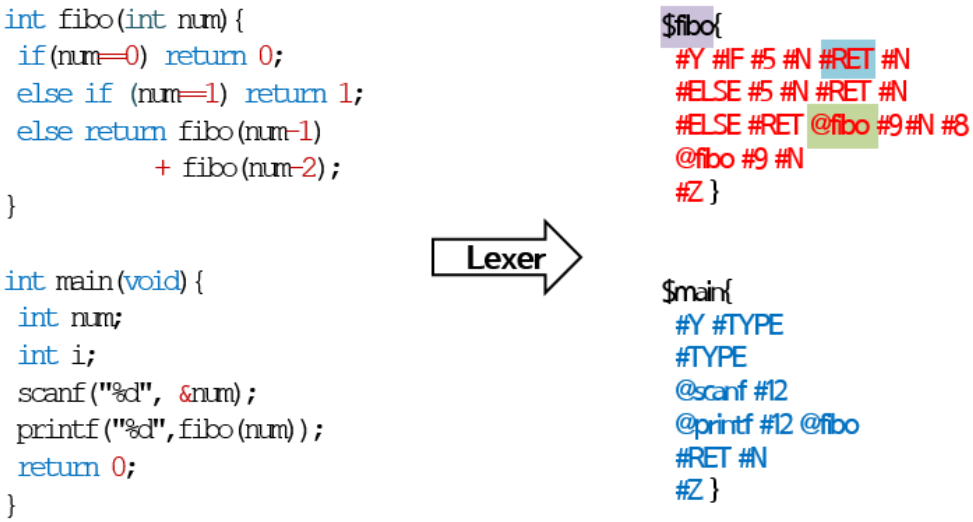
전체 시스템 흐름은 다음과 같다. 크게 소스코드를 요약하는 문자열인 program DNA를 생성한 후 DNA를 비교하여 유사도를 계산한 뒤, 생성된 유사도 결과를 통해 그래프로 시각화 한다.

4.2. program DNA 생성

User가 입력한 다량의 소스코드로 이루어진 zip파일을 받고 하나의 소스코드를 하나의 program DNA로 변환하는 작업을 한다. 이 작업은 'Lexer'를 통한 token 생성과 생성된 여러 토큰을 하나의 program DNA로 연결하는 작업으로 구성돼 있다.

4.2.1. Token 생성

사용자의 입력을 통해 받은 소스코드를 token으로 생성 및 분리한다. 여기서 token이란 소스코드에 있는 'int', ' void' ,'while'등의 문법적인 keyword들을 'INT' , 'VOID', 'WHILE'과 같이 우리가 지정해 놓은 단어들을 말한다. 구문분석에 많이 쓰이는 'Lexer"를 통해 제작했으며 정규식을 통해 소스 코드 속의 많은 keyword들을 token 으로 변형시킨다.



[그림 2. Lexer를 통한 token 생성]

그림 2은 c언어로 짜여진 소스 코드를 lexer를 통해 생성된 토큰이다. 토큰은 크게 '함수 생성', '함수 호출', '그 외' 세 개로 나눌 수 있으며 각각 토큰 앞에 '$', '@', '#'이 위치하여 이를 구분 할 수 있다. '그 외' 토큰인 #으로 시작하는 토큰에는 그림n에서 볼 수 있듯이 'Y', 'RET' , 숫자 등 다양한 토큰이 있다. 이 토큰들은 표n과 같은 기준에 의해 생성됐다. 토큰 들 중 일부를 표 n에 설명했다.

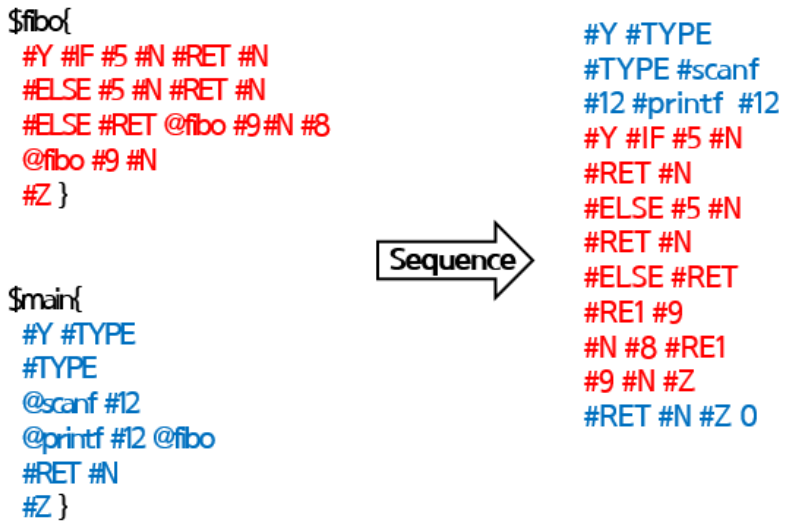
생선된 token	keyword	생선된 token	keyword
IF	If	1	++
Y	함수 시작	2	--
STATIC	static	4	Bit 연산자
RE1	재귀 함수	7	Assign 연산자
RE2	상호 재귀 함수	N	수(number)

[표 2. C/C++ 토큰 中 일부]

위와 같은 토큰들은 함수의 생성시 발생한다. 따라서 전역 변수나 구조체의 경우 토큰이 생성되지 않는다. Overriding? 함수의 경우 가장 처음 생성된 함수만 처리한다.

4.2.2 Token 연결

앞에서 생성된 token에 기반하여 소스코드를 하나의 문자열로 간추린 program DNA를 만든다. 이때 중요한 것은 코드의 함수 호출에 유의한다는 것이다.



[그림 3. 함수 호출을 고려한 program DNA 생성]

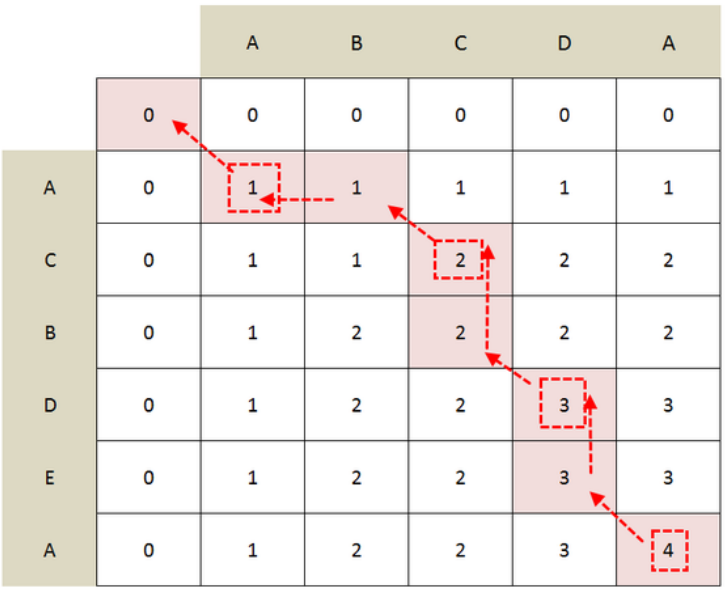
그림 3은 앞에서 만든 fibo.c. 코드의 token을 program DNA 결과 값을 보여준다. 코드의 시작인 main 함수의 token들을 시작으로 '@'로 시작하는 함수 호출 token이 있을 시 해당 함수의 토큰은 DNA에 붙인다. 결과 적으로 main함수의 token에 해당하는 파란색 토큰 사이에 'fibo'함수에 해당하는 빨간색 token이 위치한 것을 알 수 있다. 이를 통해서 학생들이 표절 탐지에 걸리지 않기 위해 함수의 위치를 바꾸거나, 여러 개의 함수를 하나 등의 함수를 합치기, 반대로 함수를 쪼개는 등의 수법에 통하지 않고 유사한 program DNA를 생성함으로써 표절을 알아챌 수 있다.

4.3 유사도 계산

앞의 과정에서 생산된 n개의 program DNA끼리 알고리즘을 통해 최종 유사도를 계산한다. 이때 사용된 알고리즘은 공통 문자열 추출에 많이 사용되는 LCS(Longest Common Subsequence)알고리즘을 변형한 최대 유사치 결정 알고리즘을 사용 했다. 여기서 토큰이 같을 때 마다 해당 program DNA에서 토큰의 빈도수를 고려하여 점수를 주는 '토큰 빈도 가중치' 기법을 적용함으로써 정확도를 높였다.

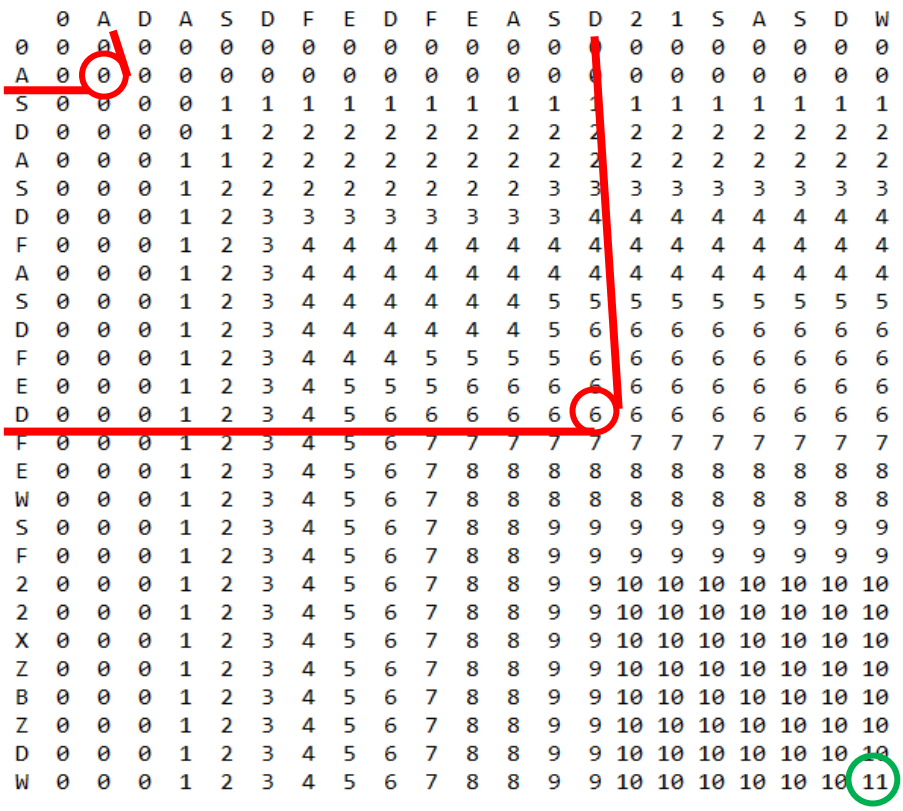
4.3.1. 최대 유사치 결정 알고리즘

최대 유사치 결정 알고리즘은 LCS 알고리즘의 변형이다. LCS알고리즘은 두 문자열 사이에서 각 문자(char)의 순서를 고려했을 때 가장 긴 문자를 추출하는데 사용되는 알고리즘이다.



[그림 4. lcs 알고리즘 예시]

그림 4에서 알 수 있듯이 비교하는 두 문자열의 문자가 같을 시 '1'의 점수가 더해 진다. 그리고 맨 끝부터 역추적하여 "ACDA"라는 최장 공통 문자열을 찾아낸다. '명탐정 한이'의 경의 문자열이 아닌 최대 유사 점수(위의 경우 4)만 필요하기 때문에 역추적은 하지 않는다.



	0	A	D	A	S	D	F	E	D	F	E	A	S	D	2	1	S	A	S	D	W
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
D	0	0	0	0	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
A	0	0	0	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
S	0	0	0	1	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	3
D	0	0	0	1	2	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4
F	0	0	0	1	2	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
A	0	0	0	1	2	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
S	0	0	0	1	2	3	4	4	4	4	4	4	5	5	5	5	5	5	5	5	5
D	0	0	0	1	2	3	4	4	4	4	4	4	5	6	6	6	6	6	6	6	6
F	0	0	0	1	2	3	4	4	4	5	5	5	5	6	6	6	6	6	6	6	6
E	0	0	0	1	2	3	4	5	5	5	6	6	6	6	6	6	6	6	6	6	6
D	0	0	0	1	2	3	4	5	6	6	6	6	6	6	6	6	6	6	6	6	6
F	0	0	0	1	2	3	4	5	6	7	7	7	7	7	7	7	7	7	7	7	7
E	0	0	0	1	2	3	4	5	6	7	8	8	8	8	8	8	8	8	8	8	8
W	0	0	0	1	2	3	4	5	6	7	8	8	8	8	8	8	8	8	8	8	8
S	0	0	0	1	2	3	4	5	6	7	8	8	9	9	9	9	9	9	9	9	9
F	0	0	0	1	2	3	4	5	6	7	8	8	9	9	9	9	9	9	9	9	9
2	0	0	0	1	2	3	4	5	6	7	8	8	9	9	10	10	10	10	10	10	10
2	0	0	0	1	2	3	4	5	6	7	8	8	9	9	10	10	10	10	10	10	10
X	0	0	0	1	2	3	4	5	6	7	8	8	9	9	10	10	10	10	10	10	10
Z	0	0	0	1	2	3	4	5	6	7	8	8	9	9	10	10	10	10	10	10	10
B	0	0	0	1	2	3	4	5	6	7	8	8	9	9	10	10	10	10	10	10	10
Z	0	0	0	1	2	3	4	5	6	7	8	8	9	9	10	10	10	10	10	10	10
D	0	0	0	1	2	3	4	5	6	7	8	8	9	9	10	10	10	10	10	10	10
W	0	0	0	1	2	3	4	5	6	7	8	8	9	9	10	10	10	10	10	10	11

[그림 5. 최대 유사치 결정 알고리즘의 적용]

또한 문자가 같다고 무조건 점수를 주는 것이 아닌 앞에서 얼마나 많은 토큰이 유사한지를 고려하여 점수를 준다. 그림 5에서 'A'는 공통 문자 이지만 점수를 주지 않았다. 앞에서 같은 문자가 없었기 때문이다. 문자를 비교하면서 일치하는 문자가 증가할지 일치 문자열 점수를 증가 시키며 이 점수가 특정 점수(15를 적용함)를 넘을 시 가산을 허용한다. 중간의 'D'에도 적용된 것을 확인할 수 있으며 이는 'D'앞의 문자들이 일치하지 않아서 일치 문자열 점수가 감소했기 때문이다. 최종적으로 최대 유사치는 '11'이 된다.

4.3.2. 토큰 빈도 가중치

LCS알고리즘에서 보통 가중치는 '1'로 동일한 점수를 주거나, 사용자가 지정해 높은 토큰별 점수를 적용한다. 하지만 소스코드마다 그 가중치는 달라질 수 있으며 코드를 짠 학생마다 'for'문 보다 'while'문을 사용하는 학생, 'register' 변수 등과 같이 많은 학생들이 사용하지 않는 문법적 요소를 사용하는 학생 등, 특정 토큰의 가중치를 임의로 매기는 것은 소스코드 표절 검사에 위험한 방법이라 할 수 있다. 따라서 우리는 소스코드에서 토큰의 빈도를 고려한 '토큰 빈도 가중치' 기법을 적용했다. 즉 빈번하게 등장하는 토큰에는 낮은 가중치를, 드물게 나오는 토큰에는 높은 가중치를 주는 것이다.

$$Adaptive\_W(t) = \frac{DNA길이}{C(t)*5 + 3} + 1$$

$$C(t) = 해당\ 토큰\ 등장\ 횟수$$

[그림 6. 빈도 가중치 함수]

그림 6은 실제 가중치를 구하는데 사용된 함수로 이 값을 lcs알고리즘에서 가중치로 적용하여 최종 유사도를 구한다. 이때 DNA길이나 C(t)의 경우 program DNA마다 달라지게 된다. 따라서 DNA1 vs DNA2의 결과와 DNA2 vs DNA1는 다른 결과를 보여준다. 함수에 사용된 상수 '5'와 '3'의 경우 2016~2017년도 알고리즘 과제물 중 표절 데이터라 의심되는 소스코드로 실험한 결과 유사도가 높게 측정되는 경우의 상수 결과이다.

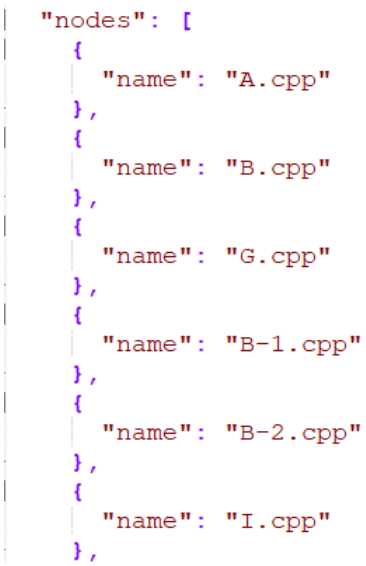
최대 유사치 결정 알고리즘과 토큰 빈도 가중치 기법을 통해 보다 정확하게 소스 코드 표절을 탐지 할 수 있다.

4.4 유사도 결과 시각화

위의 4.2~4.3의 과정으로 도출된 학생들 간의 유사도 결과를 분석하기 쉽게 시각화 한다. 4.1의 과정에서 소스 코드 유사도 결과를 json 파일로 전체 학생들의 이름과 각각의 유사도 결과를 정리한다. 한 눈에 알아보기 어려운 표절의 흐름을 알기 쉽도록 phylogenetic tree 로 출력할 수 있도록 json 파일을 작업한다. Phylogenetic tree 란 유사도 관계가 있는 개체끼리 연결하고, 유사한 관계가 있는 개체가 어떻게 변형되는지 그 흐름을 볼 수 있는 그래프이다.

4.2.1 json 파일 처리

4.1 과정을 마친 후 소스코드의 유사도 결과는 아래의 그림과 같이 nodes와 links 두가지 항목으로 json을 구성한다. nodes는 학생이 소스코드를 제출한 순서에 따라 나열한 것이고, links는 각각의 소스 코드 쌍의 유사도를 계산한 결과를 weight의 값으로 나열한 것이다.



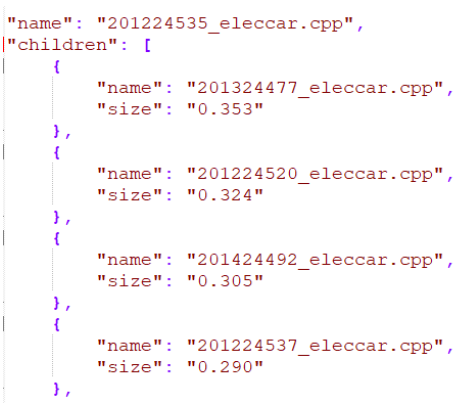
```
"nodes": [
  {
    "name": "A.cpp"
  },
  {
    "name": "B.cpp"
  },
  {
    "name": "G.cpp"
  },
  {
    "name": "B-1.cpp"
  },
  {
    "name": "B-2.cpp"
  },
  {
    "name": "I.cpp"
  },
```


[그림 7. 제출 순서를 알려주는 nodes]

```
"links": [
  {
    "source": "201224444_eleccar.cpp",
    "target": "201224419_eleccar.cpp",
    "weight": "1.000"
  },
  {
    "source": "201124427_eleccar2.java",
    "target": "201124427_eleccar.java",
    "weight": "1.000"
  },
  {
    "source": "201324562_eleccar.cpp",
    "target": "201424492_eleccar.cpp",
    "weight": "0.814"
  },
  {
    "source": "201124496_eleccar.cpp",
    "target": "201224432_eleccar.cpp",
    "weight": "0.693"
  },
```


[그림 8. 소스 코드 유사도 결과를 알려주는 links]

nodes의 순서를 기준으로 각 node에 해당하는 표절 결과를 리스트에 정리한다. 이 리스트에서는 각각의 소스 코드와 일정 수준의 이상 유사한 다른 소스 코드가 무엇이 있는지 둘 사이의 유사도 결과값은 어떠한 수준인지 모두 알 수 있다.



```
"name": "201224535_eleccar.cpp",
"children": [
    {
        "name": "201324477_eleccar.cpp",
        "size": "0.353"
    },
    {
        "name": "201224520_eleccar.cpp",
        "size": "0.324"
    },
    {
        "name": "201424492_eleccar.cpp",
        "size": "0.305"
    },
    {
        "name": "201224537_eleccar.cpp",
        "size": "0.290"
    },
```


[그림 9.]

하지만 이 리스트에서는 A와 B의 소스코드가 유사한 경우, A와 B 각각의 리스트에 서로의 유사도 결과값이 중복되어 나타나기 때문에 중복되는 내용이 많고, 제출 순서인 nodes 값을 고려했다고 할 수 없다. 따라서 A와 B 소스코드가 유사한 경우, 먼저 제출한 A의 리스트에만 A-B 유사도 결과값이 나타날 수 있도록 한다.

각각의 리스트에서 가장 유사도가 높은 소스 코드 쌍을 이용해서 tree를 그리는 작업을 한다. 모든 순서의 기준은 제출 순서 정보를 담고 있는 nodes 값이다. 리스트에서 가장 유사도가 높은 소스코드 쌍을 모아서 비교하며 서로 중복되는 소스 코드가 있는지 확인하며 자식 노드, 부모 노드를 형성한다. 예를 들어, 제출 순서는 A, A.1, A.1.1 순서라고 가정해보자. A – A.1, A.1 – A.1.1 쌍이 있을 때, 상위 노드로 A가 지정된다. 그의 자식으로는 A.1이 오고, 또, A.1의 자식으로 A.1.1이 연달아 tree 형태의 json을 이루도록 한다.


```
"name": "A.cpp",
"children": [
    {
        "name": "A-1.cpp",
        "rule": "86.72",
        "children": []
    },
    {
        "name": "A-2.cpp",
        "rule": "69.63",
        "children": []
    }
]
},
{
"name": "B.cpp",
"children": [
    {
        "name": "B-1.cpp",
        "rule": "93.79",
        "children": []
    },
```


[그림 10. phylogenetic tree를 그리는 최종 json]

4.2.2 phylogenetic tree 생성

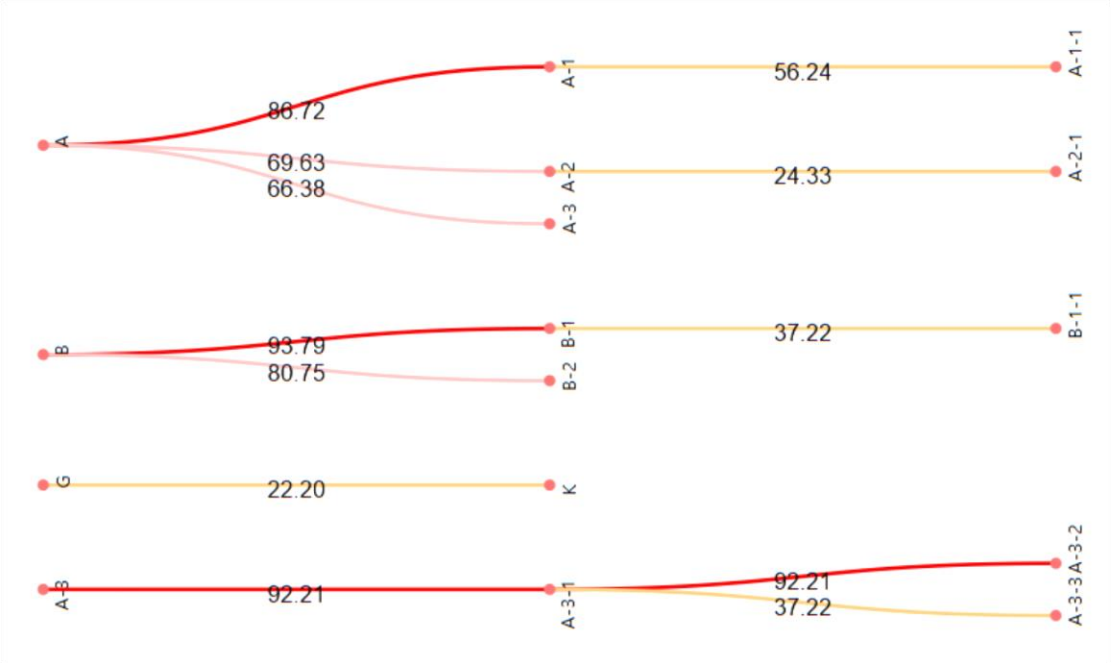
Phylogenetic tree는 d3 라이브러리를 사용해 구현했다. 각각의 소스코드를 표현하는 node와 소스코드 사이의 관계와 유사도 결과를 보여주는 link로 구성되어있다. 기존의 d3 라이브러리에서 제공하는 그래프는 가로이지만 프로그램의 특성상 그래프를 세로로 표현하는 것이 소스 코드 표절의 흐름이 잘 나타날 것이라 생각해 각각의 node와 link를 그룹으로 묶여 회전하였다.

node는 각각의 소스코드를 표현한다. 소스코드의 이름이 긴 경우 텍스트가 옆의 노드와 겹치는 것이 문제점이었다. 이를 해결하기 위해서 텍스트의 길이가 일정 길이를 넘는 경우 랩 함수를 사용하여서 그 다음 줄로 넘어갈 수 있도록 구현했다. 또한 자식 node를 클릭하면 유사하다고 결과가 나온 소스 코드 쌍을 직접 비교해서 눈으로 볼 수 있도록

했다.

link는 유사도 결과값을 나타낼 수 있도록 텍스트를 추가했다. 또, 유사도 결과에 따라 유사도가 85점이 넘는 경우 빨간색, 85점과 60점 사이는 분홍색, 그 밑으로는 노란색으로 표시해서 표절의 정도를 한 눈에 볼 수 있다.

각각 개별적인 소스코드 A,B,G,K로 이루어진 데이터 셋이 있다. 그 데이터셋에서 각각의 데이터를 표절한 소스코드를 A-1과 같이 "-숫자"로 이름을 붙였다. 소스코드 파일의 이름으로 표절의 흐름을 표시했고, 유사도 계산과 phylogenetic tree가 표절 흐름을 잘 표시하는지 알아보는 척도로 사용했다. 그 결과 아래와 같이 소스코드의 표절 흐름에 따라 그래프가 정확히 나타냈다는 것을 알 수 있다.

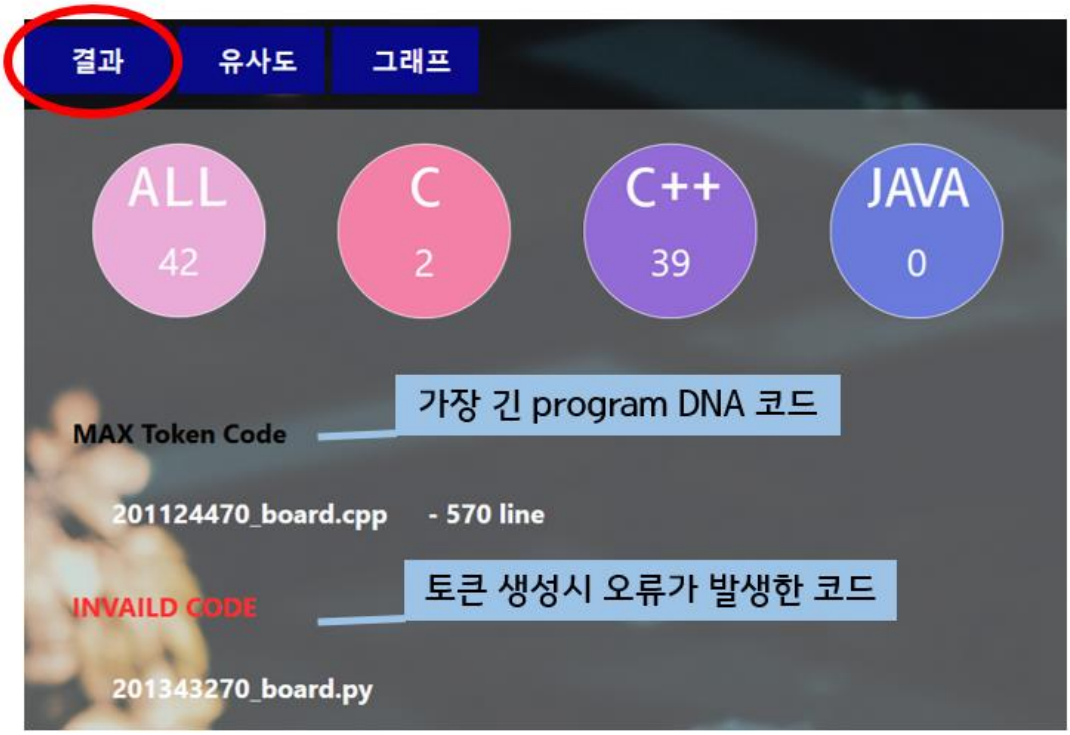


[그림 11. phylogenetic tree]

5. 수행 결과

5.1 개발 결과

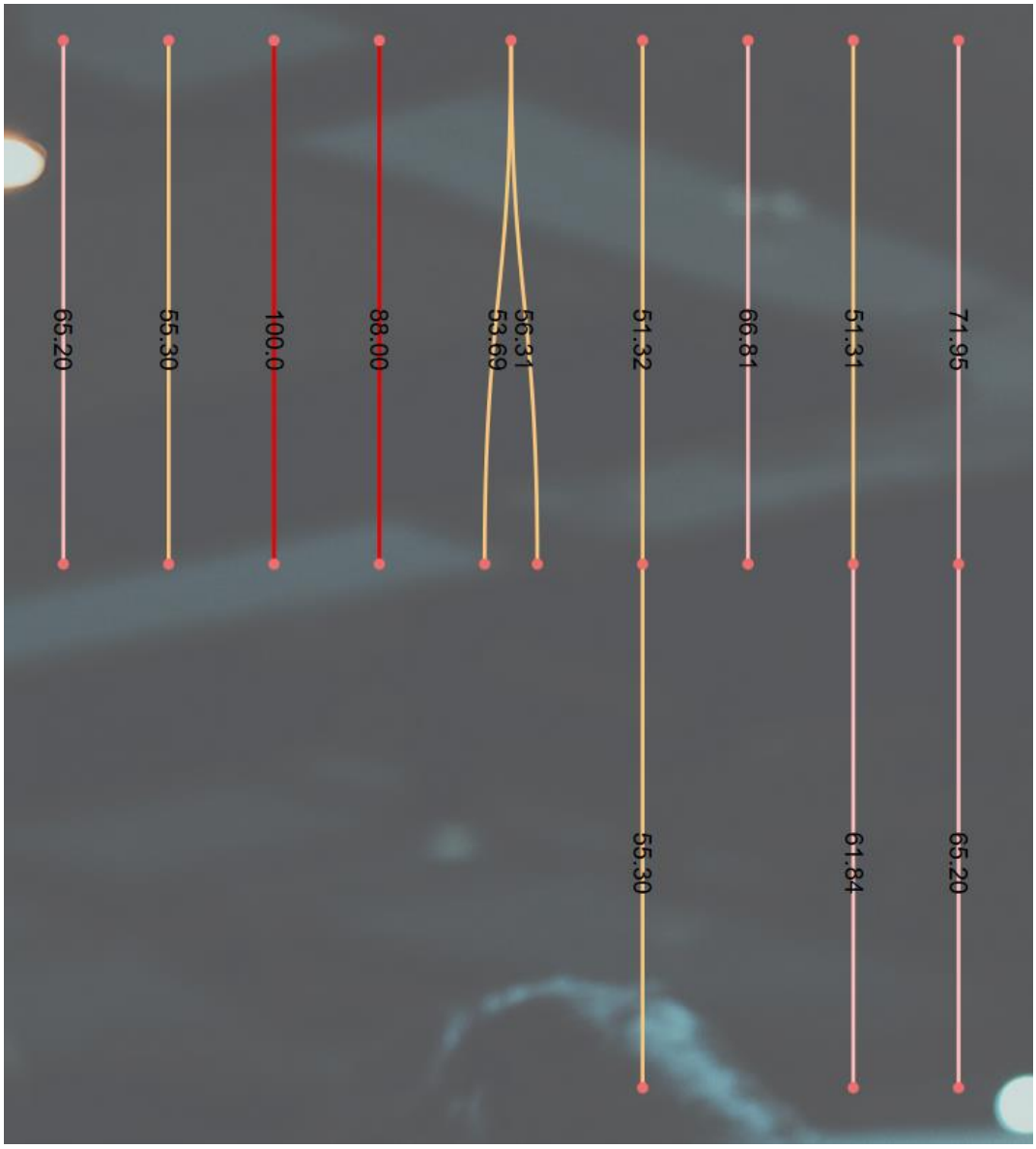
웹의 첫 화면에서는 표절을 확인할 소스코드가 담긴 zip파일을 업로드 한다. 이때 토큰의 생성 단계를 'rich', 'small'중 선택할 수 있으며 토큰 빈도 가중치의 적용 여부를 결정 할 수 있다. 'rich' 토큰이란 'small' 보다 풍부한 토큰을 가지고 있다. 예를 들어 'small'토큰은 'int', 'double' ,'char'등의 type에 해당하는 keyword들을 'TYPE'이라는 하나의 토큰으로 묶었으나, 'rich'의 경우 각각의 type 토큰을 하나하나의 토큰으로 분리했다. '빈도 가중치 off'의 경우 '토큰 빈도 가중치'를 적용하지 비교하는 두개의 토큰이 일치했을 시 '1'로 같은 가중치를 주는 경우를 말한다.



zip파일을 업로드 하면 다음과 같은 결과를 확인할 수 있다. 첫째줄에서는 해당 zip파일을 구성하고 있는 소스코드의 개수를 언어별로 보여주고 있다. 현재 우리가 지원하는 c, c++, java 언어에 한한다.

'invaild code'의 경우 파이썬과 같은 지원하지 않는 언어나 프로그램의 오류로 인해 DNA가 생성이 되지 않은 코드나 너무 길이가 짧은 경우에 해당한다.

알고리즘을 통해 생성된 유사도 결과를 d3.js 라이브러리를 사용하여 막대그래프로 보여준다.



알고리즘을 통해 생성된 유사도 결과를 d3.js 라이브러리를 사용하여 phylogenetic tree로 보여준다. 소스 코드의 표절 흐름 방향을 알 수 있다. 또한 유사도 결과 값에 따라서 85점 이상의 경우 빨간색, 60점 이상은 분홍색 그 밑으로는 노란색으로 표현했다.

5.2 타 시스템과의 비교

```
int Floyd(....){
  BODY 1
}

void Floyd_hall(....){
  BODY 2
}

int main(){
  string NodeTbla;
  int Length;

  Floyd(....);
  Floyd_hall(....);

}
```

2013245XX_candle.cpp

```
int main(){
  string nodetable;
  int length;

  for (int StartNode = 1;
    for (int EndNode = 1;
      if( StartNode ==
                ....
  }

  for (int ViaNode = 1 ;
    for (int StartNode =
      for (int EndNode =
            ....
  }
}
```

2012245XX_candle.cpp

[그림 12. '명탐정 한이'가 찾아낸 표절 의심 코드]

그림 12은 우수한 표절 소스코드 탐지 프로그램으로 평가 받은 'Jplag'와 'Moss'는 탐지하지 못했지만 우리의 '명탐정 한이'는 찾아낸 2016년 알고리즘 과제 중 표절 의심 코드이다. 좌측의 코드는 두개의 함수를 main함수에서 호출하여 제출했으며, 우측의 코드는 좌측의 두개의 함수와 비슷한 코드를 main함수안에서 확인할 수 있다. Jplag와 Moss의 경우 소스코드에서 함수의 구조를 고려하지 않고 알고리즘에 의존한 채 유사도를 계산한다. 따라서 위와 같이 함수를 쪼개거나, 합치기, 그리고 함수의 순서를 바꾸는 단순한 표절 수법에도 유사도를 낮추고 만다. 하지만 '명탐정 한이'는 소스 코드에서 함수를 인식하고 함수 별로 token을 생성하기 때문에 위와 같은 수법에 걸리지 않고 표절을 탐지 할 수 있다.

2016년, 2017년 과제 중 하나인 'eleccar'와 'food' 소스 코드를 변형하여 만든 표절 데이터와 공인된 표절 소스 코드 데이터인 'soco'를 사용하여 '명탐정 한이'를 타 시스템인 'jplag', 'moss'와 비교하여 평가 했다. 정답율은 (탐지한 표절 쌍 개수)/(총 표절 쌍 개수)이다.

	Jplag	Moss	명탐정 한이	토큰 빈도 가중치 미적용
Eleccar (30개*)	90%	70%	94%	90%
Food (18개)	94%	88%	98%	92%
Soco1 (36개)	41%	58%	83%	75%
Soco2 (56개)	71%	78%	71%	64%
Soco3 (58개)	89%	100%	100%	100%

(*) 소스 코드 개수

[표 3. Data set 시험 결과]

표 3과 같이, 타 시스템보다 '명탐정 한이'가 높은 정답율은 보였으며 빈도 가중치를 적용했을 때 더 높은 정답율을 보였다.

학생들이 제출한 과제 소스 코드를 통해 많은 테스트를 해 본 결과 2.1에서 언급한 학생들의 표절 수법 중 'unreachable 함수 추가', '함수 쪼개기/ 결합하기' 수법에 '명탐정 한이'가 속지 않고 높은 표절율을 측정하는 것을 확인 할 수 있다.

No.	공격 명	Jplag	Moss	명탐정 한이
A1	이름 변경	○	○	○
A2	변수 선언 위치 변경	△	△	×
A3	무의미한 연산 추가	×	×	×
A4	unreachable 함수 추가	×	×	○
A5	함수 쪼개기/ 결합하기	△	△	○
A6	동일 연산 바꾸기	×	×	×

○ 표절 수법에 속지 않음
△ 보통
× 표절 수법에 속음

[표 4. 표절 수법에 대한 타시스템과의 비교]

모든 표절 수법에 본 시스템이 우수한 성능을 내는 것은 아니다. 다만 함수의 호출 순서를 고려한 token 생성관 토큰 빈도 가중치를 사용하여 위와 같은 경우에 타 시스템들은 잡지 못하는 표절 수법들을 잡아내고 그것을 시각화 함으로써 보다 편리한 시스템을 제공한다.

6. 산업체 멘토링 결과 반영 내용

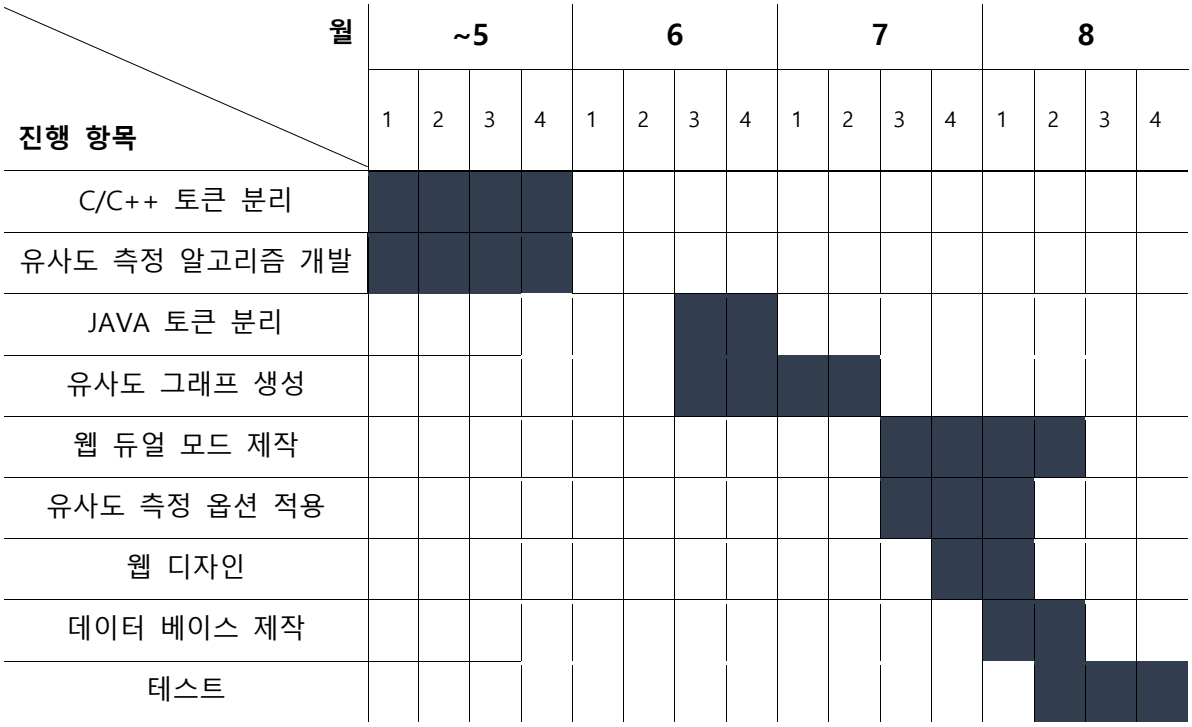
구분	내용
지도 내용	- 유사도 측정 개발 시스템의 소스코드 표절 탐지 기법의 정확도를 확인하고, 이를 설명하기 위한 부분을 추가할 필요가 있어 보임. 예로, 현재 이용할 수 있는 소스코드 유사도 탐지 프로그램(ex. MOSS, JPLG 등)을 활용하여 개발 시스템과 기존 프로그램 간의 정확도 차이를 확인하고 필요에 따라 개선을 통해 완성도를 높이는 방향으로 프로젝트를 발전시킬 수 있음. - Phylogenetic tree 향후 개발 목표인 다 계층 유사도 표현은 유용한 기능으로 생각됨.
반영 내용	- 유사도 측정 5.2에 타 소스 코드 유사도 탐지 프로그램과의 비교를 서술했고, 4.1~4.3의 유사도 계산하는 과정을 서술함을 통해서 계산 결과의 정확도에 대한 설명을 하였다. - Phylogenetic tree 중간 보고서 당시에는 1층의 형태로 단순 소스 코드 쌍의 유사도 관계만 확인할 수 있었으나, 이 후 중복되는 내용을 제외하고 소스 코드의 흐름을 나타낼 수 있도록 구현하였다.

7. 과제 추진 결과

7.1 계획표

구성원	역할
한성혜	- C/C++, JAVA 토큰 분리, 토큰 분리 - 유사도 측정 알고리즘 및 토큰 빈도 가중치 구현
이하원	- 유사도 그래프 생성 - 전체 WEB UI
공통	- 보고서 및 PPT 작성 - 표절 데이터 제작 - 테스트

7.2 역할 분담



참 고 문 헌

[1] 강은미, 황미녕, 조환규, "유전체 서열의 정렬 기번을 이용한 소스코드 표절 검사" 정보과학회논문지, Vol. 9, No 3, pp.352-354, 2003

[2] 이기화, 김연어, 우균, "데이터 구조를 고려한 소스코드 표절 검사 기법", 정보처리학회논문지. Vol 3, No6, pp.192-195, 2013

[3] Lutz Prechelt, Guido Malpohl, and Michael Philippsen, "Finding plagiarisms among a set of programs with JPlag", Journal of Universal Computer Science, Vol 8, No 11, pp 1016-1038, 2002.

[4] Thomas Lancaster, Fintain Culwin, "Classifications of Plagiarism detection Engines", ITALICS, 2005

[5] 프로그램 심의 조정 위원회, "컴퓨터 소프트웨어 감정관련 국내.외 동향 조사 및 분석", 2002