Pusan National University
Computer Science and Engineering
Technical Report 2022-22

2022년 전기 졸업과제 최종보고서

과제물 Python 코드의 상대적 품질 평가 시스템 개발
Developing Quality Evaluation System for Assignment Python Codes

저자 1 201724475 박 인 오
저자 2 201724528 이 범 수
저자 3 201741161 윤 석 현

지도교수 조 환 규

목 차

1. 서론

1.1 연구 동기

학생들은 프로그래밍 과제 수행 중 코드의 크기를 줄이기 위하여 변수명 또는 클래스명을 'a'와 같이 억지로 짧게 작성하고, 문제를 단순하게 해결하기 위해 조건문과 중첩되는 반복문을 남용하는 방법을 사용한다. 이 방식을 '우아하지 않다'고 한다. 우아하지 않게 작성된 코드는 가독성이 좋지 않게 된다. 가독성이 좋지 않은 코드를 작성하는 습관을 방치 할 경우 차후에 협업을 하거나 유지보수 작업을 수행 할 때, 우아한 코드보다 많은 소통이 필요하고 이해하는 데 시간을 더 크게 소모하게 된다. 또한, 지속적으로 우아하지 않은 코드를 작성할 경우, 코드 리팩토링에 어려움을 겪고, 코드 작성 습관을 고치기 위해 많은 시간을 들여야 한다.

위 문단에서 언급한 문제를 판단하는 방법으로, 모든 코드를 사람이 직접 보고 판단하는 방법과 기존에 개발된 도구를 활용하는 방법이 있다. 하지만 사람이 직접 보고 판단하는 방법은 인력과 시간이 많이 소모된다. 기존 개발된 도구를 사용하여 정량적인 점수를 계산하여 판단하는 방법은 사람이 판단하는 방법보다 효율적이다. 하지만 이 방법은 다른 목적으로 작성된 코드들에 대해서도 동일한 판단 기준을 적용한다. 과제물 코드들에 대해 적용하기에는 점수가 지나치게 낮거나 높은 결과가 나올 수 있고, 채점 기준에 부합하지 않을 수 있다.

개발 될 도구는 기존에 존재하는 도구들과 달리, 특정한 목적을 위해 작성된 여러 코드들에 대한 분석을 실시하고 이들을 상대적으로 평가한다. 단일 목적을 대상으로 하기에, 목적이 다른 코드를 동시에 입력 코드군으로 사용하지 않는다고 가정한다. 상대적으로 평가하기에 수치들은 표준화된다. 그 결과에 따라 우아한 코드와 일반적인 코드 우아하지 않은 코드로 분류하고, 사용자가 분류 결과를 납득할 수 있어야 한다.

1.2 기존 도구 및 문제점

flake8은 PEP8 스타일 가이드 위반을 탐지하는 모듈이다. 설치된 Python 버전에 맞추어 파싱한다. 옵션에 따라 특정 스타일 가이드를 무시하거나, 위반한 가이드 그룹만 선택하여 출력할 수 있다.

Radon은 단일 코드의 metric을 계산하는 모듈이다. 해당하는 metric들로는 raw metric과 cyclomatic complexity, Halstead metrics, maintainability index가 있다.

이름	유/무료	개발 목적	주요 방법	측정값	개발자
flake8	무료	PEP8 스타일 가이드 위반 탐지	Parse Tree	양의 정수	PyCQA
Radon	무료	단일 소스코드 metric 계산	Abstract Syntax Tree	[0-100] 실수	rubik

표 1. 기존 개발된 도구 목록

기존 개발된 도구들은 단일 소스코드에 대한 절대적인 코드 분석만을 실시한다. 특정 목적을 위해 작성된 여러 코드들의 상대적인 점수화 방법은 존재하지 않았다. 때문에 과제와 같이 특정 목표를 위하여 작성된 코드들에 대해서는 평가하기 힘들다는 문제점이 존재한다. 본 과제에서는 이를 해결하는 도구를 만들고자 한다.

1.3 관련 연구 및 사용되는 모듈

- Recommending Refactoring Solutions Based on Traceability and Code Metrics**[1]**

소스 코드 디자인과 추적 가능성을 개선하는 대체 리팩토링 방법을 선택한다. 추적 가능성과 소스 코드 디자인의 품질을 정량화하기 위해 엔트로피 기반과 고전적인 커플링, 응집 metric을 활용한다(To this end, we select among alternative refactoring solutions according to how they improve the traceability as well as source code design. To quantify the quality of traceability and source code design we leverage the use of entropy-based and traditional coupling and cohesion metrics respectively).

Code Metric을 분석한 후 Code Refactoring에 대한 추천 Solution을 제공함으로써 우아한 코드에 좀 더 다가가는 것으로 보이지만 이러한 Code Metric은 Refactoring 가이드라인에 기준한 절대적 분석이다. 협업을 통한 소프트웨어 개발과 같은 상황에서는 옳은 방법이지만, Coding Test와 같이 다수의 사람이 특정한 한 가지 목표를 보고 소스코드들을 작성하는 상황에서는 이러한 분석을 통해서 우아함을 측정하는 것은 옳다고 보기 힘들 수 있다고 보았다.

즉, 상황에 따라서는 절대평가가 아닌 상대평가를 통해 코드의 우아함을 측정 할 필요가 있다는 것이다.

- flake8**[2]**

PEP8 스타일 가이드 위반을 탐지하는 모듈이다. 설치된 Python 버전에 맞추어 파싱한다. 옵션에 따라 특정 스타일 가이드를 무시하거나, 위반한 가이드 그룹만 선택하여 출력할 수 있다.

- Radon**[3]**

단일 코드의 metric을 계산하는 모듈이다. 해당하는 metric들로는 raw metric과 cyclomatic complexity, Halstead metrics, maintainability index가 있다.

- PLY**[4]**

Python Lex-Yacc의 약자이며, 순수하게 Python으로 작성된 구문 분석 도구이다. C언어에 존재하는 Lex와 Yacc을 Python으로 이식한 것이며, 원본과 동일하게 LALR parsing 기술을 사용하고 다양한 디버깅 및 오류탐지 도구가 존재한다.

기존 Python에서 제공하는 Tokenizer의 경우 토큰의 종류가 적기 때문에 더 정확한 구문 분석을 위해 직접 Token의 종류를 설정 할 수 있는 PLY를 사용하게 되었다.

- React**[5]**

HTML을 컴포넌트 단위로 나눠 작성할 수 있어서 재사용성이 뛰어나고 가상의 DOM을 조작하기 때문에 DOM을 직접 조작하는 jQuery에 비해 빠르다.

- React-table**[6]**

유사도를 표로 나타내는데 사용하는 라이브러리이다. 정렬과 검색기능을 지원하고 tr, td 태그를 직접 작성해야 하는 작업을 mapping을 활용해 편리하다.

- Chart.js**[7]**

우아함의 정도를 그래프로 나타내는데 사용되는 라이브러리이다. 다양한 형태의 차트를 지원한다. 본 과제에서는 scatterChart를 사용한다.

- Express**[8]**

Node.js의 웹 프레임워크로 서버를 만드는데 Javascript로 통일하는 것이 좋을 것이라고 생각하여 Node.js의 표준 웹 서버 프레임워크로 불리는 Express를 사용하게 되었다.

1.4 연구 목표

개발하는 도구는 입력받은 코드들을 분석하여 판단 기준에 따라 점수를 계산하고, 사용자가 계산된 결과를 알기 쉽게 하는 것이 목표이다. 입력받는 코드들은 모두 하나의 특정 목표를 위해 작성된 코드라고 가정하며, 우아함의 등급은 절대적 점수가 아닌 상대적인 점수로 부여한다. 도구의 실행 결과를 사용자가 납득할 수 있도록 하기 위해, 도구를 검증할 때 사람이 직접 확인하여 임의의 점수를 부여하고 실행한다. 그 결과에 따라 각 판단 기준에 대한 가중치를 조정한다.

분류 기준은 코드 블록의 깊이와 반복문의 깊이, 반복문의 수, 한 함수의 다른 함수 호출 수, 재귀 여부 등을 고려한다. 코드 블록의 깊이와 내부 반복문의 깊이, 반복문의 수에 따라 가중치를 다르게 하여 계산하고, 실행 결과에 따라 가중치를 조정한다. 수치들은 과제 목표와 같이, 특정 목적에 따라 수치들의 추세가 바뀌므로, 표준화시켜 분포를 나타낸다. 또한 코드 metric 도구를 이용하여 계산된 수치도 고려한다. 기존 도구에 의해 계산된 수치들은 단일 코드를 분석하므로, 그 결과 또한 표준화한다.

PLY는 build를 수행할 때 코드 내부에서 위에 존재하는 함수에 선언된 토큰, 아래에 존재하는 함수에 선언된 토큰, 변수로 선언된 토큰 순서로 우선순위가 부여된다. 우선순위로는 복소수와 같이 복잡한 식을 가진 토큰들이 높은 우선순위를 갖도록 규칙을 정의하였다. 그 후 test, test_blank 함수를 실행하면 위의 우선순위에 따라 토큰이 분석된다.

2. 요구조건 및 제약사항 분석에 따른 수정사항

2.1 기존 요구조건 및 수정사항

2.1.1 Python 소스코드의 토큰화

Python 소스코드의 우아함을 확인하기 위해서는 우선적으로 해당 소스코드의 구조를 파악할 수 있는 토큰화 과정이 필요하다. 2022년 7월 중순 기준으로 Python 버전 3.10.5에서는 Python의 기본 모듈로 tokenize.py가 존재하지만, Token의 종류가 적다. 그렇기 때문에 세부적인 토큰화를 위해 Token의 종류를 개발자가 정할 수 있는 Lex를 이용하여 토큰화를 진행하려 하였다. 그러나, 소스코드의 우아함 측정이 모두 Python을 통해 작성되고, 기존 Lex가 더 이상 사후지원되지 않아 팀원과의 연계를 위해 PLY(Python Lex-Yacc)를 이용하는 것으로 요구조건을 일부 변경하였다.

기존 요구조건	Python 소스코드를 입력받아 해당 소스코드를 토큰화한다. - Lex를 이용하여 해당 작업을 수행 한다. - 입력받는 소스코드는 .py 파일의 형태로 입력받는다. - 토큰화 이후 출력은 결과 값을 포함한 .txt 파일로 출력한다.
수정 요구조건	- PLY를 이용하여 해당 작업을 수행한다. - 입력받는 소스코드는 Python 문자열(Raw String)의 형태로 입력받는다. - 토큰화 이후 출력은 PLY의 Token Object List로 출력한다. - 소스코드 내부에서 정의된 함수의 이름들은 별개의 토큰 종류로 분류한다. - 위의 모든 기능을 클래스 내부에 구현하여 다른 작업에서도 자유롭게 사용할 수 있도록 만든다.

표 2. Python 소스코드의 Tokenize의 요구조건

2.1.2 토큰화된 Python 소스코드 구조분석

Python 소스코드의 우아함을 측정하기 위해서는 토큰화 이후 해당 소스코드에 사용된 반복문, 조건문, 함수에 대해서 Tree 자료구조를 이용해서 구조화하는 Token의 가공 과정이 필요하다 생각하였다. 또한, Tree로 저장한 소스코드 구조를 이용해서 반복문의 깊이, 조건문의 넓이, 함수가 외부 함수를 호출한 횟수 등을 구해서 2차원 리스트로 출력하여 우아함을 계산하는 팀원에게 전달하는 것으로 결정하였다.

하지만, Python의 경우 반복문, 조건문, 함수 이외에도 class, with 구문 등이 Indent를 변화시키는 점이 문제가 되어 반복문, 조건문, 함수만을 Tree에 저장하는 것이 아닌, Indent를 변화시키는 모든 Token을 Tree에 저장하는 것으로 Python 소스코드 Tree를 더욱 정교화 시키는 것으로 요구조건을 일부 변경하였다.

또한 함수와 관련된 정보를 저장하는 Tree를 별개로 제작하였다.

기존 요구조건	- 제작된 Python Tokenizer가 출력한 Token List를 입력으로 받는다. - anytree Library를 이용한다. - 반복문, 조건문, 함수를 한 트리에 저장하여 구조화한다. - 구조화된 트리를 이용하여 우아함 수치화에 필요한 자료를 출력한다.
수정 요구조건	- Python 소스코드의 Indent를 변화시킬 수 있는 Token들 전

	부를 Tree에 집어넣는다. - 함수로만 구성되는 Tree를 따로 제작한다. - Python 소스코드의 Indent에 변화를 주는 Token들로 이루어진 Tree는 반복문의 깊이, 조건문의 넓이를 구하는데 사용한다. - 함수로만 구성되는 Tree는 함수내부에서 자기 이외의 함수 호출횟수, 재귀회출 여부에 대해서 구하는데 사용한다.

표 3. 토큰화된 Python 소스코드 구조분석의 요구조건

2.1.3 소스코드의 우아함 척도

초기에 계획하였던 우아함의 척도로는 변수명이 사전에 등록된 단어인지와, 변수의 타입 변경 여부가 있었다. 변수명이 사전에 등록된 단어인지 파악하기 위해서는 토큰화를 거친 후 타입에 따라 판별해야 하는데, 적절하게 축약된 단어는 자주 사용하고, 파일 크기를 줄이는 데 도움이 된다고 판단하여 변수명 판단은 기준에서 제거되었다. 또한, 변수의 타입 변경 여부는 동적 검사 시에만 가능하기에 계획에서 제거되었다.

기존에 계획되지 않은 다른 척도들이 추가되었다. 추가된 척도로는, 순환 복잡도(cyclomatic complexity, CC)와 조건문의 수, 반복문의 수, 함수의 수, 함수의 깊이가 있다. 이 수치들은 채점자의 기준에서 명확한 수치가 도움이 될 것이라 판단하여 표준화하지 않는다. 추가로, 최대 깊이와 평균 깊이, 총 합 깊이를 계산한 후 표준화하여 나타낸다. 우아함 검사 후 결과를 JSON을 요구함에 따라 출력에 대한 요구사항을 수정하였다.

기존 요구조건	- 우아함의 척도로 변수명을 검사한다. - 변수의 타입이 변경되는지 확인한다.
수정 요구조건	- 순환 복잡도(cyclomatic complexity)를 검사한다. - 조건문과 반복문, 함수의 개수와 함수의 깊이를 검사한다. - 최대 깊이와 평균 깊이, 총 합 깊이를 표준화하여 나타낸다. - 우아함 검사의 결과를 JSON 형식으로 한다.

표 4. 소스코드의 우아함의 척도의 요구조건

2.1.4 소스코드 분류 결과

초기에 계획하였던 소스코드 분류는 elegant와 plain, ugly였다. 하지만 이 분류는 상대평가에 사용되는 도구라는 목적을 크게 만족시키지 못한다고 판

단했다. 상대평가라는 목적을 만족시키기 위해 개별 점수를 부여하여 순위를 정하기 쉽도록 한다. 개별 점수는 판단 기준들에 따라 계산된다.

기존 요구조건	- 소스코드를 분류하여 elegant와 plain, ugly로 분류한다.
수정 요구조건	- 판단 기준에 따라 소스코드에 점수를 부여한다.

표 5. 소스코드 분류 결과의 요구조건

2.2 기존 제약사항 및 수정사항

2.2.1 기존 제약사항

제약 사항	① 소스코드의 우아함을 측정할 테스트용 코드가 부족할 수 있다. ② 개발된 도구가 입력받는 코드는 정상 동작을 보장하지 않는다. ③ 짧은 길이의 코드 또는 구현이 간단한 코드인 경우 유사도가 높을 수 있다.
해결 방안	① 임의로 코드를 작성하여 사용한다. ② Python Interpreter를 거쳐 코드의 정상 동작 여부를 판단한 후 도구를 사용한다. ③ 수치화된 우아함만을 출력하지 않고, 평균과 표준편차를 구하여 점수를 표준화한다.

표 5. 기존 제약 사항과 해결 방안

2.2.2 수정/추가 제약사항

제약 사항	① 적은 사람이 임의로 코드를 작성하여 만드는 소스코드들로는 테스트에 한계가 존재한다. 더 많은 사람이 특정 목표(ex. 과제)를 위해 작성한 소스코드들이 필요하다. ② 유사도 검사와 우아함 검사의 결과 형식을 통일하여 웹사이트에 표시하기 용이하도록 한다.
해결 방안	① 지도교수님께서 강의하시는 컴퓨터프로그래밍입문 수업의 과제로 제출된 학생들의 소스코드들을, 지도교수님의 허가 하에 우아함 측정 테스트에 사용한다. ② 우아함 검사의 결과 형식을 JSON으로 통일한다.

표 6. 수정 및 추가된 제약 사항과 해결 방안

3. 연구 내용

3.1 Python 소스코드의 토큰화

3.1.1 reserved word 및 built in function Token 설정

Python 3.10.5 기준 reserved word, built in function을 reserved dictionary를 이용해 정규표현식을 이용하지 않고 토큰화에 대한 정보를 설정한다. **(부록, 코드 1)**

3.1.2 [3.1.1]에서 정의한 token들을 제외한 나머지 token들을 정의

앞서 reserved dictionary로 따로 정의한 token들을 제외한 정규표현식으로 구현할 token들을 tokens list에 정의한다. **(부록, 코드 2)**

3.1.3 정규표현식을 이용해 token 선별 규칙을 구현

tokens list에 정의된 token들을 t_TOKEN 형태의 변수나 함수로 정의하고 내부에 정규표현식을 입력하여 해당 token을 선별하는 규칙을 구현한다. 특히, 함수로 구현된 token 규칙들은 token적용에 대한 우선순위를 구현할 수 있다. 예를들어 3.1이라는 실수는 INTEGER, DOT, INTEGER라는 token으로 표현할 수도 있지만, FLOATNUM 이라는 token으로 표현할 수도 있다. 이를 결정하는 것은 INTEGER token과 FLOATNUM token 사이의 우선순위를 어떻게 결정하냐에 따라 달라지며, 해당 프로그램에서는 FLOATNUM token의 우선순위를 더 높게 설정해 3.1이라는 실수가 입력될 경우 FLOATNUM이라는 token이 출력된다. **(부록, 코드 3)**

3.1.4 Lex build와 tokenize 확인을 위한 test 함수구현

위의 모든 과정을 거친 후 Lex를 build하여 tokenize 준비를 끝낸다. 그리고 test와 test_blank 함수를 이용해 Python 소스코드를 String으로 입력받아 Token Object 형태로 출력한다. test_blank 함수의 경우 test 함수와 다르게 들여쓰기를 파악하기 위해 추가적인 작업을 수행한다. **(부록, 코드 4)**

3.2 토큰화된 Python 소스코드 구조분석

3.2.1 Python 소스코드 Token Tree 생성

[3.1]의 PythonTokenizer를 사용하여 소스코드를 Token Object List로 변형하고, 이를 if, elif, else, for, while등의 토큰과 Indent의 변화를 토대로 Python 구조 트리를 제작한다.

또는 소스코드 내부의 함수와 Indent의 변화를 토대로 Python 함수 트리를 제작한다. **(부록, 코드 5)**

3.2.2 소스코드 구조 분석 및 결과 출력

[3.2.1]에서 제작된 Python 구조 또는 함수 트리를 이용해서 소스코드의 반복문의 깊이, 조건문의 넓이, 함수의 타 함수 호출 횟수 및 재귀 여부를 파악한 후 이를 2차원 List로 출력한다. **(부록, 코드 6)**

3.3 소스코드의 우아함 척도

3.3.1 반복문과 조건문, 함수의 개수와 깊이

코드 블록의 깊이와 반복문의 깊이, 반복문의 수, 조건문의 수와 함수당 호출 수를 각각 구한다. 구한 수치들로 최대 깊이와 평균 깊이, 총 합 깊이를 구한다. 수치들은 과제 목표와 같이, 과제 목표에 따라 수치들의 추세가 바뀌므로 표준화하여 분포를 나타낸다. **(부록, 코드 7)**

3.3.2 PEP8 스타일 가이드 위반

오픈소스 라이브러리 flake8을 이용한다. 클래스를 인스턴스화할 때, PEP8 스타일 가이드 기준을 설정한다. PEP8 스타일 가이드 기준은 권장 기준을 선택하거나, 사용자가 웹에서 별도로 선택할 수 있도록 한다.

작성된 코드가 위반한 PEP8 스타일 가이드를 찾는다. 현재 위반 사항 중 '파일 끝의 라인 없음'과 같은 일부 가이드는 무시(ignore 옵션)하고 있으나 사용자가 선택할 수 있도록 한다. **(부록, 코드 8)**

3.3.3 순환 복잡도(cyclomatic complexity)

순환 복잡도(이하 CC)는 코드 블록의 decision 횟수에 비례한다. CC에 영향을 주는 decision으로는 if와 elif, for, while, except, with, assert, 여러 comprehension, bool 연산자가 있다. else와 finally는 판단하지 않고 종속되기에 영향을 주지 않는다. CC의 수치가 낮을수록 코드의 안정성이 높다.

$$CC = decisions + 1$$

식 1. cyclomatic complexity

코드에서 계산된 CC의 리스트를 만든다. 그 합과 최대 CC, 평균 CC를 이용하여 복잡도를 계산한다. 이 수치는 표준화하여 나타낸다. **(부록, 코드 9)**

3.3.4 상대 점수 계산

상대 점수는 표준화된 수치들을 이용하여 계산한 점수이다. 반복문과 조건문, 함수의 개수와 최대 깊이, 총 깊이, PEP8 스타일 가이드 위반 수치들을 표준화하여 0에서 뺀다. 이 수치가 높을수록 상대적으로 잘 만들어진 코드로 판단한다. **(부록, 코드 10)**

3.4 실행 결과

계산된 모든 값들을 json 오브젝트로 변환하고, output.json으로 내보낸다. json 파일은 코드 파일명을 key로, 계산된 값들을 value로 한다. 서버 측에서 메인 코드를 실행하여 결과를 계산하고, 생성된 json 파일의 값들을 이용하여 프론트에 표시하여 사용자가 확인할 수 있도록 한다.

그림 1. 시작 화면

사용자는 접속했을 때 그림 1과 같은 페이지에 접속하게 된다. 검사할 코드들을 끌어 넣거나 버튼을 클릭하여 업로드 대상으로 지정할 수 있고, 업로드 대상이 된 파일들의 개수가 좌측에 표시된다. 이후 CODE SUMBIT 버튼을 클릭하여 업로드할 수 있다.

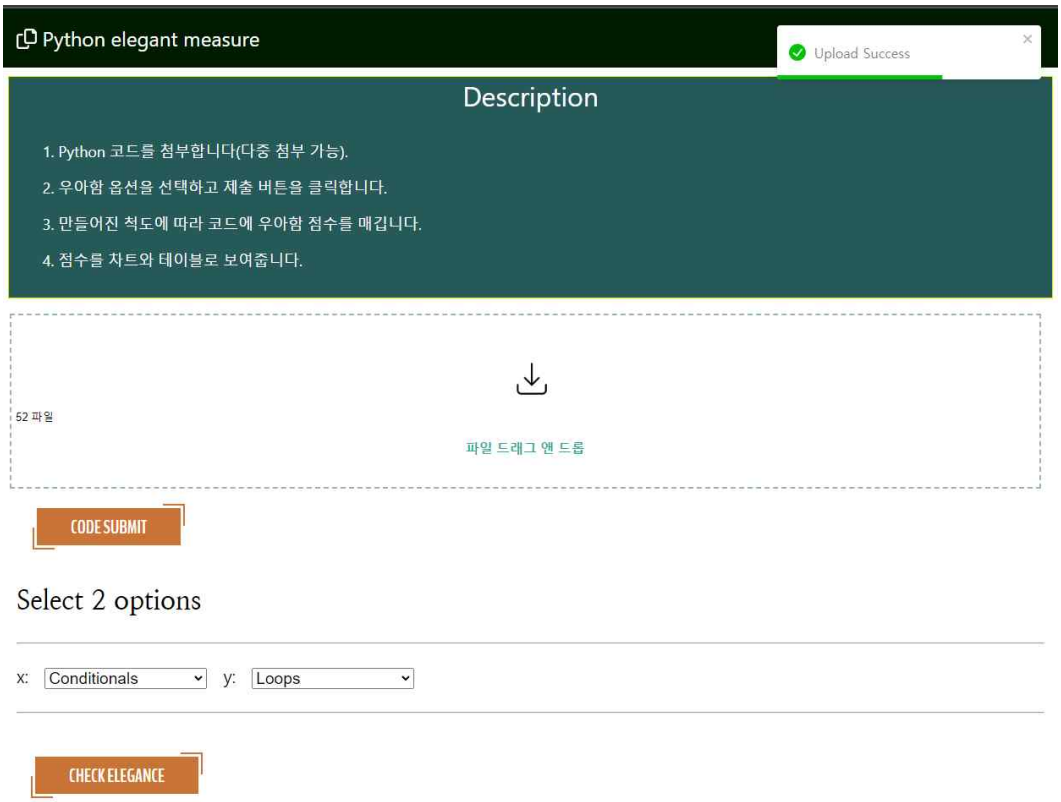

그림 2. 업로드 성공 시 화면

업로드할 때, 그림 2와 같이 우상단에 성공 여부를 토스트 메시지로 출력한다. 업로드가 성공했다면 드롭다운 메뉴가 보여지게 된다. 드롭다운에서 서로 다른 두 지표를 선택하여 CHECK ELEGANCE 버튼을 클릭하면 산포도 그래프와 json파일을 나타낸 표를 확인할 수 있다.

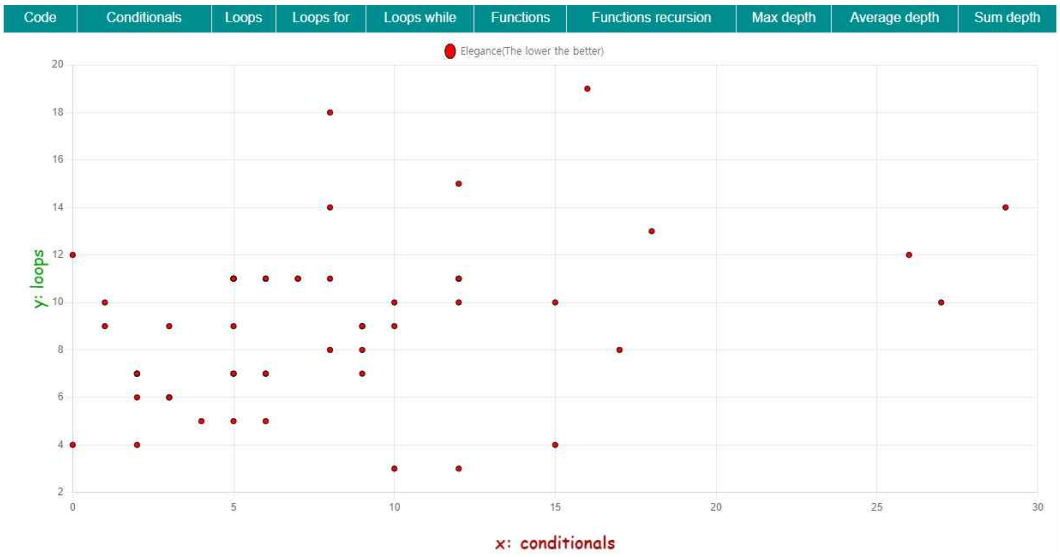

그림 3. 산포도 그래프

그림 3은 코드들의 추세를 확인하기 위해 지원하는 산포도 그래프이다. 그래프 위의 점들은 각 파일에 해당하며, 점에 마우스오버할 시 파일명과 선택한 지표의 값이 출력된다.

Code	Conditionals	Loops	Loops for	Loops while	Functions	Functions recursion	Max depth	Average depth	Sum depth
4033.py	2	6	5	1	0	0	-0.72	-0.67	-0.813
4036.py	6	7	3	4	5	0	-0.036	-0.124	0.605
4040.py	10	10	7	3	2	1	1.673	1.058	0.984
4044.py	2	7	6	1	1	1	-0.036	0.058	-0.435
4045.py	0	12	12	0	2	1	-0.378	-0.306	-0.435
4048.py	12	3	2	1	6	4	0.135	-0.003	1.267
4054.py	3	6	6	0	4	0	0.135	-0.124	0.322
4061.py	5	11	9	2	8	0	0.477	-0.147	1.362

그림 4. json 차트

그림 4는 json 파일의 값들을 나타낸 표의 일부이다. 첫 행의 지표 카테고리 중 하나를 선택하여 클릭하면 선택한 지표를 기준으로 오름차순과 내림차순으로 정렬할 수 있다. 초기 설정은 파일명으로 정렬되어 있다. json 파일은 버튼을 클릭하여 다운로드할 수 있다.

4. 연구 결과 분석 및 평가

개발된 도구는 기존에 존재하는 도구들과 달리, 특정한 목적을 위해 작성된 여러 코드들에 대한 분석을 실시하고 이들을 상대적으로 평가한다. 입력받은 코드들을 분석하여 판단 기준에 따라 점수를 계산하고, 사용자가 계산된 결과를 알기 쉽게 점수로 나타낸다.

입력받는 코드들은 모두 하나의 특정 목표를 위해 작성된 코드라고 가정되었으며, 우아함의 등급은 절대적 점수가 아닌 상대적인 점수로 부여했다. 도구를 검증하기 위해 과제물 소스코드들을 직접 분류한 후, 도구를 사용한 점수를 비교하여 확인하는 과정을 거쳤다.

분류 기준은 반복문과 조건문, 함수의 개수, 재귀함수의 개수, 순환복잡도를 고려했다. 코드 블록의 깊이와 내부 반복문의 깊이, 반복문의 수에 따라 가중치를 다르게 하여 계산했다. 수치들은 과제 목표와 같이, 특정 목적에 따라 수치들의 추세가 바뀌므로, 표준화시켜 분포를 나타내었고, 점수에 반영했다.

시험 데이터로 도구를 검증한 결과 중, 점수 하위권 코드 3개와 상위권 코드 4개를 분석했다. 각 코드의 점수는 –11.718과 -9.140, -7.037, 4.060, 4.184, 4.256, 6.467로 계산되었으며, 점수가 높을수록 잘 작성된 코드라고 판단한다. 점수가 높은 코드들이 점수가 낮은 코드들보다 보기에 잘 작성되었다고 할 수 있다. 하지만 코드 16과 같이 일부 코드는 점수와 달리 보통 수준으로 작성되었다고 점수가 계산되었다. 이 점을 해결하기 위해 점수 계산식에서 일부 가중치를 조정할 필요가 있다. **(부록, 코드 11-17)**

개발된 도구는 과제물 평가라는 기준에 맞추어 상대적인 평가 도구라는 점에 의의가 있다. 기존 개발된 도구가 정량적인 판단 기준에 따라 프로그램의 가치를 판단하는 것과 달리, 상대적인 평가와 위반 사항에 대한 수치를 나타내기에 평가 및 추후 강의 방안을 수정하는 데 도움이 될 것이다.

5. 결론 및 향후 연구 방향

5.1 결론

Python 소스코드의 우아함을 절대적인 평가가 아닌 입력한 소스코드의 집합에서 표준화된 수치를 이용하여 상대적인 점수를 산출하는 평가 방식을 이용해 소스코드 집단을 분류하였다. 이를 이용하면 과제와 같이 특정 목표를 위해 작성된 소스코드들에 가점 또는 감점을 주어야 할 때, 각 과제물들을 평가하는 시간을 절약할 수 있을 것이다.

5.2 향후 발전 계획

- 차후 Python뿐만이 아닌 다른 프로그래밍 언어들의 경우에도 현재 프로그램과 비슷한 형식의 Tokenize와 소스코드 분석 방식을 이용하여 해당 서비스를 이용 할 수 있도록 구성원들과 논의할 예정이다.
- 현재 서비스에 이용되는 프론트엔드가 상대적으로 UI, UX가 뒤떨어진 모습을 보이고 있으며, 이를 보완할 예정이다.
- 점수 계산식의 수치들에 가중치를 부여하여 분류 결과의 정확성을 높일 예정이다.

6. 구성원별 역할과 개발일정

담당자	내용	기간	상태
박 인 오	우아함 검사 알고리즘 작성	7월 1주 ~ 9월 2주	완료
	- 복잡도 수치화	7월 1주 ~ 7월 3주	완료
	- PEP8 스타일 가이드 위반 판단	7월 4주 ~ 8월 2주	완료
	- 소스코드 metric 계산	8월 3주 ~ 9월 1주	완료
	- 우아함 검사 결과 JSON화	9월 2주	완료
이 범 수	Python 코드 토큰화 알고리즘 작성	7월 1주 ~ 8월 1주	완료
	- 토큰 정의 및 우선순위 결정	7월 1주 ~ 7월 3주	완료
	- 토큰화 함수 구현	7월 4주 ~ 8월 1주	완료
	Tokenize된 Python 소스코드 구조분석	8월 2주 ~ 9월 2주	완료
	- 소스코드 구조 Tree 형성	8월 2주 ~ 8월 3주	완료
	- 반복문, 조건문, 함수 분석 후 결과출력	8월 4주 ~ 9월 2주	완료
윤 석 현	서버 구현	7월 1주 ~ 8월 2주	완료
	- Node.js 서버 구현	7월 1주 ~ 7월 3주	완료
	- react-table, chart.js 라이브러리 적용	7월 4주 ~ 8월 2주	완료
	프론트 구현	8월 3주 ~ 9월 2주	완료
	- 메인, 우아함 검사 화면	8월 3주 ~ 9월 1주	완료
	- JSON 결과 적용	9월 1주 ~ 9월 2주	완료
박 인 오 이 범 수 윤 석 현	프로그램 통합 및 구동확인	9월 2주 ~ 9월 3주	완료
	프로그램 디버깅	9월 3주 ~ 9월 4주	완료
	최종 보고서 작성	9월 4주	완료

표 7. 구성원별 개발 현황

7. 참고 문헌

[1] A. S. Nyamawe, H. Luy, Z. Niy, W. Wang and N. Niu, "Recommending Refactoring Solutions Based on Traceability and Code Metrics," in IEEE Access, vol.6, pp. 49460-49475, 2018, doi: 10.1109/ACCESS.2018.2868990.

[2] flake8, https://flake8.pycqa.org/en/latest/

[3] Radon, https://radon.readthedocs.io/en/latest/

[4] PLY, https://www.dabeaz.com/ply/

[5] React, https://ko.reactjs.org/docs/react-api.html

[6] React-table, https://tanstack.com/table/v8/docs/guide/introduction

[7] Chart.js, https://www.chartjs.org/docs/latest/

[8] Express, https://expressjs.com/ko/4x/api.html

8. 부록

```
import ply.lex as lex

class PythonLexer(object) :
    reserved = {
        'False' : 'FALSE',      #reserved words
        'None' : 'NONE',
        'True' : 'TRUE',
        'and' : 'AND',
        'as' : 'AS',
        'assert' : 'ASSERT',
        'async' : 'ASYNC',
        'await' : 'AWAIT',
        'break' : 'BREAK',
        'class' : 'CLASS',
        'continue' : 'CONTINUE',
        'def' : 'DEF',
        'del' : 'DEL',
        'elif' : 'ELIF',
        'else' : 'ELSE',
        'except' : 'EXCEPT',
        'finally' : 'FINALLY',
        'for' : 'FOR',
        'from' : 'FROM',
        'global' : 'GLOBAL',
        'if' : 'IF',
        'import' : 'IMPORT',
        'in' : 'IN',
        'is' : 'IS',
        'lambda' : 'LAMBDA',
        'nonlocal' : 'NONLOCAL',
        'not' : 'NOT',
        'or' : 'OR',
        'pass' : 'PASS',
        'raise' : 'RAISE',
        'return' : 'RETURN',
        'try' : 'TRY',
        'while' : 'WHILE',
        'with' : 'WITH',
        'yield' : 'YIELD',
        'abs' : 'ABS',          #built in functions
```

코드 1. reserved dictionary

```
tokens = [
    'VARIABLE',
    'INTEGER',
    'FLOATNUM',
    'COMPLEXNUM',
    'STRING',
    'PLUS',
    'MINUS',
    'TIMES',
    'DIVIDE',
    'MODULUS',
    'FLOORDIVISION',
    'EXPONENT',
    'GREATER',
    'LESS',
    'GREATEREQUAL',
    'LESSEQUAL',
    'EQUAL',
    'NOTEQUAL',
    'LPAREN',
    'RPAREN',
    'LBRACE',
    'RBRACE',
    'LBRACKET',
    'RBRACKET',
    'COMMA',
    'DOT',
    'COLON',
    'ASSIGNMENT',
    'UNDERBAR',
    'TRIPLEDOT',
    'BLANK',
    'NEWLINE',
] + list(reserved.values())
```

코드 2. tokens list

```
t_PLUS = r'\+'
t_MINUS = r'-'
t_TIMES = r'\*'
t_DIVIDE = r'/'
t_MODULUS = r'%'
t_FLOORDIVISION = r'//'
t_EXPONENT = r'\*\*'
t_GREATER = r'>'
t_LESS = r'<'
t_GREATEREQUAL = r'>='
t_LESSEQUAL = r'<='
t_EQUAL = r'=='
t_NOTEQUAL = r'!='
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_LBRACE = r'\{'
t_RBRACE = r'\}'
t_LBRACKET = r'\['
t_RBRACKET = r'\]'
t_COMMA = r','
t_DOT = r'\.'
t_COLON = r':'
t_ASSIGNMENT = r'='
t_UNDERBAR = r'_'
t_TRIPLEDOT = r'\.{3}'
t_BLANK = r'[ \t]'

t_ignore_COMMENT = r'\#.*'

def t_COMPLEXNUM(self, t):
    r'([+-]?(0|[1-9]\d*)\.\d+([eE][+-]?(0|[1-9]\d*))?)?[+-]?(0|[1-9]\d*)\.\d
    return t

def t_FLOATNUM(self, t):
    r'[+-]?(0|[1-9]\d*)\.\d+([eE][+-]?(0|[1-9]\d*))?'
    return t

def t_INTEGER(self, t):
    r'[+-]?(0|[1-9]\d*)'
```

코드 3. t_TOKEN 형태의 변수와 함수들

```
def build(self, **kwargs):
    self.lexer = lex.lex(module=self, **kwargs)

def test(self, data):
    self.lexer.input(data)
    while(True):
        tok = self.lexer.token()
        if(not tok):
            break
        print(tok)

def test_blank(self, data):
    self.lexer.input(data)

    newline_flag = False

    while(True):
        tok = self.lexer.token()
        if(not tok):
            break
        if(tok.type == 'NEWLINE'):
            newline_flag = True
        if(tok.type == 'BLANK' and newline_flag == False):
            continue
        if(tok.type != 'NEWLINE' and tok.type != 'BLANK' and newline_flag ==
            newline_flag = False
        print(tok)
```

코드 4. build, test, test_blank 함수

```
def analyze_python_code(self, code):
    tokens = self.tokenizer.tokenize_indent(code)
    self.python_code_tree = Node("python_code_tree", indent = -1)
    indent = 0
    current_node = self.python_code_tree

    for token in tokens:
        if(token.type == 'NEWLINE'):
            indent = 0
            continue
        if(token.type == 'INDENT'):
            indent = len(token.value)
            continue
        if(token.type in ['IF','ELIF','ELSE','FOR','WHILE','CLASS','DEF','MA
            while(True):
                if(current_node.indent < indent):
                    current_node = Node(token.value+"+"+str(token.lexpos), i
                    break
                else:
                    current_node = current_node.parent

    for pre, fill, node in RenderTree(self.python_code_tree):
        print("%s%s" % (pre, node.name))


def analyze_python_function(self, code):
    tokens = self.tokenizer.tokenize_indent(code)
    self.python_function_tree = Node("python_function_tree", indent = -1)
    indent = 0
    current_node = self.python_function_tree

    for token in tokens:
        if(token.type == 'NEWLINE'):
            indent = 0
            continue
        if(token.type == 'INDENT'):
            indent = len(token.value)
            continue
        if(token.type == 'FUNCTION'):
            while(True):
                if(current_node.indent < indent):
                    current_node = Node(token.value, indent = indent, parent
                    break
                else:
                    current_node = current_node.parent

    for pre, fill, node in RenderTree(self.python_function_tree):
        print("%s%s" % (pre, node.name))
```

코드 5. analyze_python_code 함수와 analyze_python_function 함수

```
def print_conditional(self):
    conditional_list = []
    nodes = self.python_code_tree.root.descendants
    for node in nodes:
        if(node.name.split('+')[0] == 'if'):
            breadth = 1
            for sib in node.siblings:
                if(int(node.name.split('+')[1]) < int(sib.name.split('+')[1]
                    break
                elif(int(node.name.split('+')[1]) < int(sib.name.split('+')[
                    breadth += 1
            conditional_list.append([node.name,breadth])
    return conditional_list


def print_loop(self):
    loop_list = []
    height = 0
    nodes = [self.python_code_tree.root]
    while(len(nodes) != 0):
        new_nodes = []
        for node in nodes:
            if(node.name.split('+')[0] in ['for','while']):
                loop_list.append([node.name, height])
                new_nodes += [node]
            else:
                nodes += list(node.children)
        nodes = []
        for node in new_nodes:
            nodes += list(node.children)
        height += 1

    for loop in loop_list:
        loop[1] = height-loop[1]-1

    return loop_list


def print_function(self):
    function_list = []
    repeat_check = []
    for func in self.python_function_tree.children:
        if(func.name in repeat_check):
            continue
        recursion = False
        children = []
        for child in func.children:
            children.append(child.name)
        if(func.name in children):
            recursion = True
        function_list.append([func.name, len(children), recursion])
        repeat_check.append(func.name)

    return function_list
```

코드 6. print_conditional 함수와 print_loop 함수 그리고 print_function 함수

```
for file_name, code in zip(files, codes):
    analyzer = PythonCodeAnalyzer.PythonDepthBreadth()
    analyzer.analyze_python_code(code)

    # conditional
    conditionals = analyzer.print_conditional()
    conditionals_all.append(len(conditionals))

    # loop
    loop = analyzer.print_loop()
    loops_all.append(len(loop))

    # functions
    analyzer2 = PythonCodeAnalyzer.PythonDepthBreadth()
    analyzer2.analyze_python_function(code)
    functions = analyzer2.print_function()
    functions_all.append(len(functions))

    # recursion
    recursion_count = 0
    for element in functions:
        if element[2] == True:
            recursion_count += 1
    recursions_all.append(recursion_count)

z_conditionals = utils.get_z_scores(conditionals_all)
z_loops = utils.get_z_scores(loops_all)
z_functions = utils.get_z_scores(functions_all)
z_recursions = utils.get_z_scores(recursions_all)
# endregion
```

코드 7. 반복문과 조건문, 함수의 개수와 깊이

```
# region PEP8 위반 검사
reports = instance.get_PEP8_metrics(files)
PEP8_violations = [[[violation.split()[1], violation.split()[0]] for violation in report]
                   for report in reports if len(reports) > 0]
PEP8_violation_counts = [len(report) for report in reports]
z_PEP8_violations = utils.get_z_scores(PEP8_violation_counts)
# endregion
```

코드 8. PEP8 위반 검사

```
# region 순환복잡도 검사
depths = instance.get_cyclomatic_metrics(codes)
max_depths = [max(depth) if len(depth) > 0 else 0 for depth in depths]
avg_depths = [sum(depth) / len(depth) if len(depth) > 0 else 0 for depth in depths]
sum_depths = [sum(depth) if len(depth) > 0 else 0 for depth in depths]
z_max_depths = utils.get_z_scores(max_depths)
z_avg_depths = utils.get_z_scores(avg_depths)
z_sum_depths = utils.get_z_scores(sum_depths)
# endregion
```

코드 9. 순환 복잡도 검사

```
# region 점수 계산 및 파일 출력
for i in range(len(files)):
    score = 0 - (z_max_depths[i] + z_sum_depths[i] + z_PEP8_violations[i]
                 + z_recursions[i] + z_functions[i] + z_loops[i] + z_conditionals[i])
    scores.append(score)
```

코드 10. 점수 계산

```
def diagoA1(x):
    c = 0
    y = 0
    if x+1 < M:
        for i in range(x+1):
            if BG[x-y][y] == '1':
                c += 1
            else:
                if c != 0: count.append(c); c = 0;
            y += 1
        if c != 0: count.append(c)
    elif x+1 > N:
        for i in range(D-x):
            if BG[D-x-y][x-M+1+y] == '1':
                c += 1
            else:
                if c != 0: count.append(c); c = 0;
            y += 1
        if c != 0: count.append(c)
    else:
        for i in range(M):
            if BG[M-1-y][x-M+1+y] == '1':
                c += 1
            else:
                if c != 0: count.append(c); c = 0;
            y += 1
        if c != 0: count.append(c)

def diagoA2(x):
    c = 0
    y = 0
    if x+1 < N:
        for i in range(x+1):
            if BG[x-y][y] == '1':
                c += 1
            else:
                if c != 0: count.append(c); c = 0;
            y += 1
        if c != 0: count.append(c)
    elif x+1 > M:
        for i in range(D-x):
            if BG[M-1-y][x-M+y] == '1':
                c += 1
            else:
                if c != 0: count.append(c); c = 0;
            y += 1
        if c != 0: count.append(c)
    else:
        for i in range(N):
```

코드 11. 점수 하위 1위 코드의 일부

```
import numpy as np
import sys
def bingo(base):
    rows, columns = base.shape
    for i in range(rows):
        count = 0
        for j in range(columns):
            if j == 0:
                if base[i][j] == '1':  count = 1
            else:
                if base[i][j] == '1':
                    if base[i][j-1] == '1':
                        count += 1
                    else:
                        count = 1
                    if j+1 == columns: cnt.append((count))
                else:
                    if count > 0:
                        cnt.append((count))
                        count = 0
    for i in range(columns):
        count = 0
        for j in range(rows):
            if j == 0:
                if base[j][i] == '1': count = 1
            else:
                if base[j][i] == '1':
                    if base[j-1][i] == '1':
                        count += 1
                    else:
                        count = 1
                    if j+1 == rows: cnt.append((count))
                else:
                    if count > 0:
                        cnt.append((count))
                        count = 0
    N = -rows + 1
    M = columns
    for i in range(N, M):
        temp = np.diag(base, k=i)
        count = 0
        for j in range(len(temp)):
            if j == 0:
                if temp[j] == '1':
                    count = 1
                    if j+1 == len(temp): cnt.append((count))
            else:
                if temp[j] == '1':
                    if temp[j-1] == '1':
```

코드 12. 점수 하위 2위 코드의 일부

```
import numpy as np
def R() :
    for x in loc :
        for y in x :
            if (y-1) in x : continue
            elif (y+1) in x :
                stk = 2
                y += 1
                while True :
                    if (y+1) in x :
                        stk += 1
                        y += 1
                    else :
                        break
                res.append(stk)
def C() :
    for x in range(len(loc)-1) :
        for y in range(len(loc[x])) :
            if x > 0 :
                if loc[x][y] in loc[x-1] : continue
            if loc[x][y] in loc[x+1] :
                stk = 2
                tmp4 = loc[x][y]
                ad = 2
                while True :
                    if (x+ad) == len(loc) : break
                    if tmp4 in loc[x+ad] :
                        stk += 1
                        ad += 1
                    else :
                        break
                res.append(stk)
def RD() :
    for x in range(len(loc)-1) :
        for y in range(len(loc[x])) :
            if x > 0 :
                if (loc[x][y]-1) in loc[x-1] : continue
            if loc[x][y]+1 in loc[x+1] :
                stk = 2
                tmp5 = loc[x][y]
                tmp5 += 1
                ad = 2
                while True :
                    if (x+ad) == len(loc) : break
                    elif (tmp5+1) in loc[x+ad] :
                        stk += 1
                        tmp5 += 1
                        ad += 1
                    else :
```

코드 13. 점수 하위 3위 코드의 일부

```
import numpy as np
import sys

def scan(matrix):
    for i in matrix:
        row = ''.join(i)
        row = row.split('0')
        row = list(filter(None,row))
        for j in row:
            cnt = len(j)
            len_list.append(cnt)

def case_diag(x,y,matrix):
    diag = list()
    for k in range(-x+1,y):
        diag.append(list(np.diag(matrix,k)))
    scan(diag)

My, len_list = [], []
lines = sys.stdin.readlines()
for line in lines: My.append(list(line.strip()))

My     = np.array(My)
My_rot = np.rot90(My)
n, m = My.shape[0], My.shape[1]

scan(My);     case_diag(n,m,My)
scan(My_rot); case_diag(m,n,My_rot)

longest = max(len_list)
print(longest , len_list.count(longest))
```

코드 14. 점수 상위 4위 코드

```
import numpy as np
import sys


bingo = []
while True :
    line = sys.stdin.readline().strip()
    if not line : break
    bingo.append(list(map(int,line)))
bingo = np.array(bingo)


matrix = []
rows, cols = np.nonzero(bingo)
for row, col in zip(rows, cols) :
    matrix.append((row, col))


result = dict()
direction = [(0, 1), (1, 1), (1, 0), (1, -1)]
for r, c in matrix :
    for i, w in direction :
        if (r+i, c+w) in matrix and (r-i, c-w) not in matrix :
            count = 1
            mat = (r, c)
            while True :
                mat = (mat[0]+i, mat[1]+w)
                if mat in matrix : count += 1
                else : break
            if count not in result :
                result[count] = 0
            result[count] += 1


result = sorted(result.items(), key=lambda x : x[0], reverse=True)
l, k = result[0]
print(l, k)
```

코드 15. 점수 상위 3위 코드

```
bingo = np.array(b)

n = len(bingo)

x = []

for i in range(n) :
    for j in range(m) :
        if bingo[i][j] == '1' :
            x.append([i, j])

fin = []
for i in range(len(x)) :
    j = 1
    while x[i][1] + j < m :
        if [x[i][0], x[i][1] + j] in x :
            j +=1
        else :
            break
    fin.append(j)
    j = 1
    while x[i][0] + j < n :
        if [x[i][0] + j, x[i][1]] in x :
            j +=1
        else :
            break
    fin.append(j)
    j = 1
    while x[i][0] + j < n and x[i][1] + j < m :
        if [x[i][0] + j, x[i][1] + j] in x :
            j += 1
        else :
            break
    fin.append(j)
    j = 1
    while x[i][0] + j < n and x[i][1] - j >= 0 :
        if [x[i][0] + j, x[i][1] - j] in x :
            j += 1
        else :
            break
    fin.append(j)

a = max(fin)
b = fin.count(a)
print(a, b)
```

코드 16. 점수 상위 2위 코드의 일부

```
import numpy as np
import sys

image = np.array([list(map(int,x[:-1])) for x in sys.stdin.readlines()])
image = np.array(image)

row = np.array(image)
column = row.T
trans = np.rot90(row)
dig_range = range( -row.shape[0]+1, row.shape[1] )
dig1 = []
dig2 = []
for i in dig_range:
    dig1.append(list(np.diag(row,i)))
    dig2.append(list(np.diag(trans,-i)))
datas = [row, column, dig1, dig2]

mem = [0]*50

for lines in datas:
    for line in lines:
        cnt=0
        for k in line:
            if(k==0):
                mem[cnt] += 1;
                cnt=0
            else:
                cnt += 1
        if(line[-1]==1):
            mem[cnt] += 1;

idx = 49
while(mem[idx]==0): idx-=1
mem = mem[:idx+1]

print(len(mem)-1,end=" ")
print(mem[-1])
```

코드 17. 점수 상위 1위 코드