KIPS Trans. Softw. and Data Eng.
Vol.8, No.8 pp.317~322 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2019.8.8.317

Design of Programming Failure Feedback System Based on Control Flow of Test Case to Support Programming Training

Sunghee Lee† · Deok Yeop Kim† · Kang Bok Seo† · Woo Jin Lee††

ABSTRACT

Programming judge systems for programming training support are typically built on the Web, where the examiners uploads a programming problem, which the student reads and submits an answer to the problem. The judge system executes the submitted answer of source code to provide feedback such as pass, failure, and error messages. Students who receive the feedback except for the pass continues debugging the source code until they are judged to pass. We developed an online judge system to support programming training and analyzed answers submitted by the students and found that many of the students who were not judged to pass that test did not know exactly where they were wrong but continued to solve the problem. The current judge system generally feeds runtime error messages back to students. However, with only runtime error message, it is difficult for student who train to find the wrong part of the answer. Therefore, in this paper, we propose a system that provides the feedback of programming failure by analyzing the control flow of the test cases used in the source code submitted by the student. The proposed system helps students find the wrong parts more quickly by feeding back the paths where faults in the control flow may exist. In addition, we show that this system is applicable to the answer source code that the actual student submitted.

Keywords : Feedback, Failure, Control Flow, Test Case, Fault

프로그래밍 훈련 지원을 위한 테스트케이스의 제어흐름에 기반한 프로그래밍 실패 피드백 시스템 설계

이 성 희† · 김 덕 엽† · 서 강 복† · 이 우 진††

요 약

프로그래밍 훈련 지원을 위한 프로그래밍 채점시스템은 일반적으로 웹 상에 구축되어 출제자는 프로그래밍 문제를 업로드하고 이를 학습자가 열람하고 문제에 대한 답안을 제출한다. 프로그래밍 채점시스템은 제출된 답안 소스코드를 실행하여 통과, 실패, 에러 메시지 등의 피드백을 제공한다. 통과를 제외한 피드백을 받은 학습자는 통과 판정받을 때까지 계속해서 소스코드를 디버깅한다. 본교에서 프로그래밍 훈련 지원을 위해 온라인 채점시스템을 자체 개발하였고 학생들이 제출한 답안을 분석한 결과 통과 판정을 받지 못한 학생들 중 다수가 자신이 틀린 부분을 정확하게 모른 채 계속해서 문제를 푼다는 사실을 확인하였다. 현재 프로그래밍 채점시스템은 주로 런타임 에러 메시지 위주로 피드백을 수행한다. 하지만 이러한 정보만으로는 프로그래밍 훈련을 하는 학습자들은 제출한 답안의 틀린 부분을 찾아내기가 어렵다. 따라서 본 논문에서는 학습자가 제출한 소스코드에 사용된 테스트케이스의 제어흐름을 분석한 결과를 프로그래밍 실패에 대한 피드백으로 제공하는 시스템을 제안한다. 제안한 시스템은 제어흐름 상의 결함이 존재할 수 있는 경로를 피드백하여 학습자들이 틀린 부분을 보다 빠르게 찾을 수 있게 도와준다. 또한 실제 학습자가 제출한 답안 소스코드를 예로 들어 본 시스템이 적용 가능함을 보인다.

키워드 : 피드백, 실패, 제어흐름, 테스트케이스, 결함

1. 서 론

4차 산업혁명에 관심이 증가함에 따라 프로그래밍 교육에 대한 관심도 함께 증가하고 있다[1]. 일반적으로 프로그래밍 교육은 이론과 실습을 병행하고 있으며 특히 실습 훈련을 지원하기 위해 프로그래밍 채점시스템을 활용하고 있다[2-6]. 대표적인 채점시스템은 ACM UVA[7]와 BOJ(BAEKJOON ONLINE JUDGE)[8]가 있으며 일반적으로 Fig. 1과 같은 과정으로 사용된다.

먼저, 출제자가 채점시스템에 프로그래밍 문제를 업로드하면 자동으로 해당 문제가 공개된다. 학습자는 공개된 프로그래밍 문제를 열람하여 문제를 풀기 시작한다. 이후 학습자가 문제를 다 풀었다고 판단되면 그 문제에 대한 답안 소스코드

※ 이 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구되었음(No.NRF-2017R1D1A3B04035880).
※ 이 논문은 2019 한국 소프트웨어공학 학술대회(KCSE 2019)에서 “테스트케이스의 제어흐름기반 프로그래밍 실패 피드백 시스템 설계”의 제목으로 발표된 논문을 확장한 거임.
† 준 회 원 : 경북대학교 컴퓨터학부 박사과정
†† 정 회 원 : 경북대학교 컴퓨터학부/소프트웨어기술연구소 교수
Manuscript Received : March 26, 2019
Accepted : April 20, 2019
* Corresponding Author : Woo Jin Lee(woojin@knu.ac.kr)

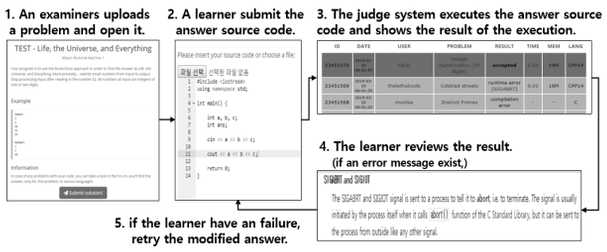


Fig. 1. The General Process of Using Judge System

를 제출한다. 채점 시스템은 제출된 소스코드를 실행하여 통과 또는 실패의 채점 결과를 알려주며 프로그래밍 실패 시, 컴파일 에러 메시지, 런타임 에러 메시지, 시간초과 등의 피드백을 사용자에게 알려준다[9]. 마지막으로 사용자는 채점결과를 확인하고 그 결과가 프로그래밍 실패인 경우, 피드백을 확인하고 소스코드를 디버깅하여 문제풀기를 계속한다.

이러한 채점시스템을 본교 학생들의 프로그래밍 교육을 지원하기 위한 목적으로 자체 개발하였고 현재 프로그래밍 교육에 활용하고 있다[10]. 개발한 채점 시스템을 사용하여 학생들의 답안 제출 이력을 분석한 결과 흥미로운 점을 발견하였다. 많은 학생들이 자신이 제출한 답안이 오답판정을 받았다면 자신이 어떤 부분을 틀렸는지를 정확히 알고 고치지 못하는 경우가 40%나 된다는 것이다. 예를 들어, Table 1은 2018학년도 1학기 기초프로그래밍 수업에 사용된 실습 문제와 해당 문제에 대한 40명의 학생들의 제출이력을 분석한 결과로 약 1/4의 학생들이 반복해서 동일한 문제를 풀지 못하는 것을 알 수 있다.

Table 1. Analysis Result of Students' Submission History

[Problem]
1st day: 1 won, 2nd day: 2 won, 3rd day: 4 won,
Like this, daily wages become double compared with the last day. Write a program that calculates the sum of the wages during the working day which is entered.
(if input value is less than 0, print "wrong" and retry to enter the input value.)

# of the submission of incorrect answers	# of people
at least one time	25/40 (62.5%)
at least two times / at least one times	10/25 (40%)
at least three times / at least one times	6/25 (24%)

동일한 문제를 반복해서 풀지 못하는 학생들에게 답안 소스코드가 실행 중에 어느 실행 경로에서 오답이 나는지 알려준다면 학생들이 자신의 답안에서 어느 부분이 잘못되었는지를 보다 빨리 찾을 수 있다[11]. 따라서 본 논문에서 학생이 제출한 답안 소스코드의 제어흐름도에 프로그래밍 실패 경로를 나타내어 학생들이 자신의 소스코드가 가지는 구조적, 논리적인 결함을 알려주는 피드백 시스템을 제안한다. 또한 실제로 학생이 제출한 소스코드를 사례로 들어 본 시스템이 필요함을 보인다.

2. 관련 연구

채점 시스템들은 제출 답안을 자동으로 채점하기 위해 공통적으로 프로그램에 입력될 정보와 미리 정의된 출력 형태를 알려준다. 그러나 제출 답안에 대한 채점 피드백의 지원 방식은 각각 다르다.

대부분의 채점 시스템은 제출 답안을 컴파일하고 실행한 후, 실행결과와 답안을 비교하여 채점한다. 따라서 최소한의 피드백으로 통과, 실패를 알려주고 실패일 때는 추가적으로 컴파일 오류, 런타임 오류 중 해당하는 것을 알려준다. 일부 시스템은 채점 결과를 더 구분하여 어떤 오류인지 또는 어떤 테스트 입력에 대해서 틀렸는지 등 상세한 정보를 알려주기도 한다[12]. 또 학생들이 프로그램의 실행 흐름이나 값의 변화를 이해할 수 있도록 실행 과정을 시각화 한 연구도 있다[13]. 그러나 제출한 코드가 오답인 경우에 어느 부분을 수정해야 하는지에 대해 피드백을 주는 채점 시스템은 없다. Table 2는 채점 시스템들의 채점 통과 유무를 제외한 지원하는 채점 피드백 내용을 나타낸 것이다.

Table 2. Feedback Contents of Online Judge Systems

Online judge system	When programming failure occurs, the feedback contents are
ACM UVA	compile error messages
BOJ	compile error messages
Codeup[12]	compile/runtime error messages, tip
DOMjudge[14]	compile error messages
AOJ[15]	compile/runtime error messages
Programmers[16]	compile/runtime error messages
SPOJ[17]	compile/runtime error messages
KOI[18]	compile/runtime error messages

기존의 채점 피드백은 운영체제 또는 컴파일러가 제공하는 오류 메시지만을 학습자에게 알려준다[8-12]. 이러한 정보들만으로는 학습자들이 프로그래밍 실패했을 때 제출한 답안이 구조적이거나 논리적인 문제가 있는지 피드백 할 수 없다. 본 논문에서는 제출된 답안 소스코드의 실행 실패 경로를 분석하여 제출된 답안의 제어흐름이 구조적으로 또는 논리적으로 문제가 있는지를 알려주어 학습자가 프로그램을 보다 신속하게 수정할 수 있다.

3. 제어흐름기반 프로그래밍 실패 피드백 시스템

기존의 채점시스템은 컴파일 오류 메시지를 포함하여 런타임에 발생하는 런타임 오류 메시지와 제한시간 내에 프로그램이 종료되지 않으면 실행시간초과의 피드백을 제공한다. 학습자들이 이 3가지 피드백 중 런타임 오류의 원인을 찾기는 어려울 수는 있지만 불가능하지는 않다. 하지만 실행시간초과 피드백의 경우는 어떤 원인으로 실행시간이 초과하는지 학습자가 알 수 없는 경우가 많다.

특히, 실행시간초과 피드백은 초급 학습자들에게 자주 발생할 수 있으며 그 원인들 중 가장 큰 이유는 무한루프이다. 초급 학습자들은 흔히 반복문 프로그래밍에서 실수하는 일이 많으며 그 결과 프로그래밍한 반복문의 결함이 생기고 이것으로 무한루프가 발생하는 경우가 많다. 이 또한 실행시간초과 피드백을 발생시킨다. 그 실제 사례로 Fig. 2(b)는 Table 1에서 나타낸 반복문에 해당하는 실습 문제에 제출한 답안에서 이와 같은 오류를 가진 소스코드의 제어흐름도이다.

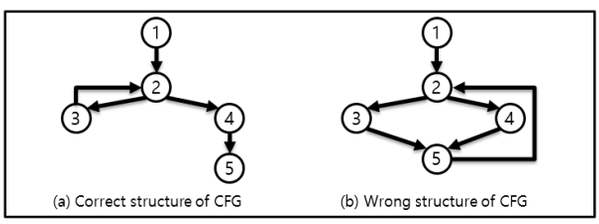


Fig. 2. The Control Flow Graphs of the Answer Source Codes

학습자는 실행시간초과의 피드백 대신 제출된 답안 소스코드는 구조적으로 무한루프라는 결함을 가지고 있고 Fig. 2(a)와 같은 구조로 수정할 수 있다는 피드백을 받을 수 있다면 자신이 제출한 답안의 결함을 논리적으로 찾기 보다는 구조적으로 찾으려 노력할 수 있어 보다 빨리 결함의 원인을 찾을 수 있다.

따라서 테스트케이스의 제어흐름에 기반한 프로그래밍 실패 피드백 시스템은 제출된 답안에 대해 구조적 결함과 논리적 결함에 대한 피드백이 필요하며 구조적 결함에 대한 피드백이 논리적 결함 피드백보다 우선되어야한다.

3.1 구조적 결함 분석을 위한 실행경로 기반의 제어흐름도

일반적으로 제어흐름도는 정적 프로그램 분석을 수행하여 컴파일 시간에 생성이 가능하다. 하지만 제어흐름도만 사용하여 실제로 프로그램이 어느 경로를 따라 동작할지는 알 수 없다. 따라서 본 논문에서는 제출된 답안 소스코드의 제어흐름도와 실행경로를 결합하여 프로그램이 실제로 수행되는 실행경로 기반의 제어흐름도를 제안한다. 또한 채점 시스템의 문제별 테스트케이스 셋은 모두 문제에서 요구하는 분기 커버리지를 100%를 달성할 수 있다고 가정한다. 또한 그렇지 않은 테스트 케이스 셋은 채점에 사용할 수 없다고 가정한다.

예를 들어, Table 3은 Table 1과 동일한 과목의 if문 실습 문항으로 2차원 공간의 x, y 좌표를 입력 받고 그 좌표가 어느 사분면에 존재하는지를 출력하는 실습 문제이다. 이 문제가 요구하는 분기는 총 4개이므로 Fig. 3(a), Fig. 3(b)는 구조적으로 문제에서 요구하는 모든 분기가 존재하는 제어흐름도이고 Fig. 3(c)는 구조적으로 필요 없는 분기가 존재하고 Fig. 3(d)는 구조적으로 필요한 분기가 부족한 제어흐름도이다.

제안하는 실행경로 기반의 제어흐름도는 Fig. 3과 같이 일반적인 제어흐름도와는 다르게 표현되어 있다. 단, 이후 설명상의 편의를 위해 x, y가 0인 TC#5는 고려하지 않으며 Fig. 3의 4개의 실행경로 기반의 제어흐름도는 테스트케이스인 TC#5를 만족하기 위한 예외처리 실행경로를 제외한 제어흐름도로 나타내었다. 실행경로 기반의 제어흐름도의 노드는 녹색(커버된 노드), 흰색(커버되지 않은 노드), 빨간색(2번 이상 중복으로 커버된 노드)으로 표현되어 있다.

Table 3. The Problem of 'If Statement' for Training

[Problem]
Write a program that receives the x and y coordinates of the two-dimensional space and outputs the quadrants to which the coordinates belong (note that x and y are non-zero values).

TC#	Test Input	Expected Value
1	3 5	Quadrant 1
2	-3 5	Quadrant 2
3	-3 -5	Quadrant 3
4	3 -5	Quadrant 4
5	0 0	N/A

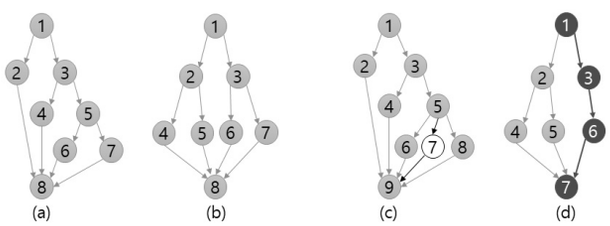


Fig. 3. The Possible Control Flow Graphs of the 'If Statement' Problem of the Table 3

예를 들어, Table 3의 테스트 케이스는 문제에서 요구하는 분기 커버리지를 100% 달성할 수 있고 모든 분기를 커버할 수 있다. Fig. 3(c)의 실행경로 기반의 제어흐름도는 커버되지 않은 노드 7을 가지므로 불필요한 분기가 존재하는 것이고 Fig. 3(d)의 제어흐름도는 노드 1-3-6-7 경로를 여러 개의 테스트 케이스가 중복해서 커버하기 때문에 필요한 분기가 빠져있다고 말할 수 있다. 이러한 실행경로 기반의 제어흐름도를 사용하여 구조적 결함에 대해 피드백 할 수 있다.

3.2 논리적 결함 분석을 위한 실행경로 기반의 제어흐름도

논리적 결함 분석은 구조적 결함 분석과정에서 구조적 결함이 없을 때 수행한다고 가정한다. 3.1절에서 설명한 Table 3의 if문 실습문제를 예를 들어 다시 설명하도록 한다. Fig. 4는 Fig. 3(a)의 구조를 가지는 답안 소스코드로 작성될 수 있는 예제 프로그램이다.

Fig. 5는 Fig. 4를 답안 소스코드로 한 채점결과 피드백을 나타낸 것이다. 제출된 답안 소스코드는 Fig. 3(a)의 구조를 가지는 답안 소스코드로 분기 구조는 잘 작성되어 구조적 결함이 없지만 1-3-5-6-8, 1-3-5-7-8 경로에 해당하는 부분은 채점결과 오답으로 나온다. 3번 출력문과 4번 출력문을 수행하는 경로의 테스트 케이스들이 프로그래밍 실패로 나오기 때문에 Fig. 5와 같은 실행경로 기반의 제어흐름도와 프로그래밍 실패 피드백이 함께 나타난다.

```
    ……
if(x > 0 && y > 0) {
     printf("Quadrant 1");              // 1st print statement
} else {
     if(x < 0 && y > 0) {
          printf("Quadrant 2")          // 2nd print statement
     } else {
          if(x < 0 && y < 0) {
               printf("Quadrant 3")     // 3rd print statement
          } else {
               printf("Quadrant 4")     // 4th print statement
          }
     }
}
    ……
```

Fig. 4. The Answer Source Code of the 'If Statement' Problem of the Table 3 with Faults

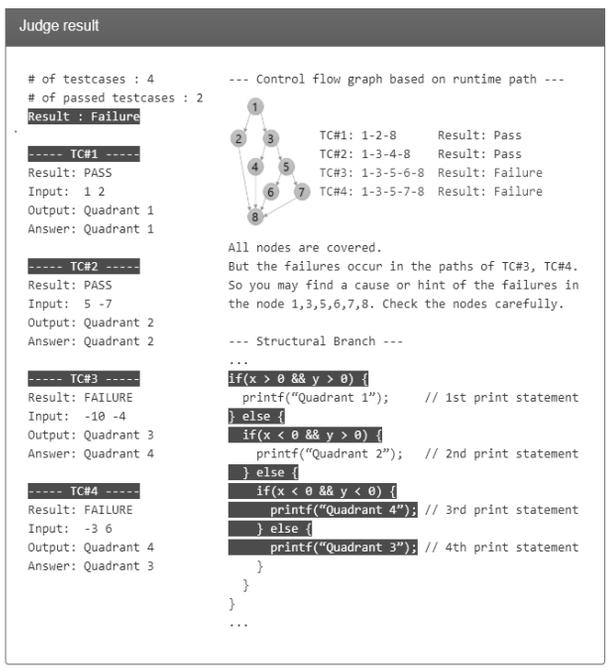


Fig. 5. The Feedback of Programming Failure Based on the Control Flow Graph

4. 적용 사례

본 장에서 예시로 보이는 소스코드들의 제어흐름도는 Table 1에서 나타낸 2018학년도 1학기 기초프로그래밍 수업 내용의 반복문에 해당하는 실습 문제의 한 학생이 제출한 답안 소스코드들이다. 제어흐름도들은 제출된 소스코드의 순서에 맞게 순차적으로 보여주며 제어흐름도에 테스트케이스(TC) 별 커버된 노드는 색으로 강조하였다. 회색으로 강조된 부분은 테스트케이스가 통과한 경로이고 붉은색으로 강조된 부분은 테스트케이스가 실패한 경로이고 강조되지 않은 노드는 커버되지 않은 노드이다.

먼저, Fig. 6은 학생이 첫번째 제출한 답안 소스코드의 제어흐름도와 테스트케이스별 실행경로를 나타내며 Fig. 7은 답안의 피드백 소스코드이다.

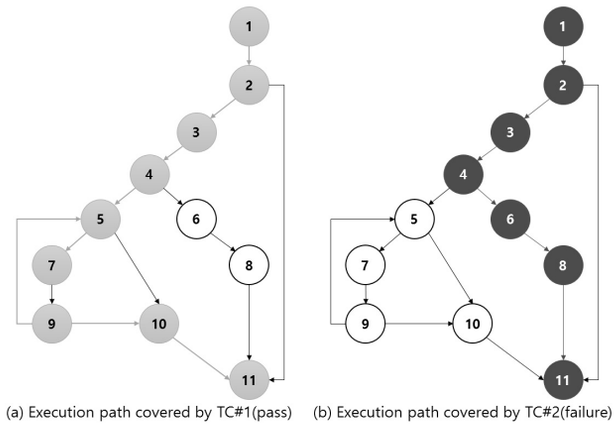


Fig. 6. The Execution Paths of the 1st Submitted Answer

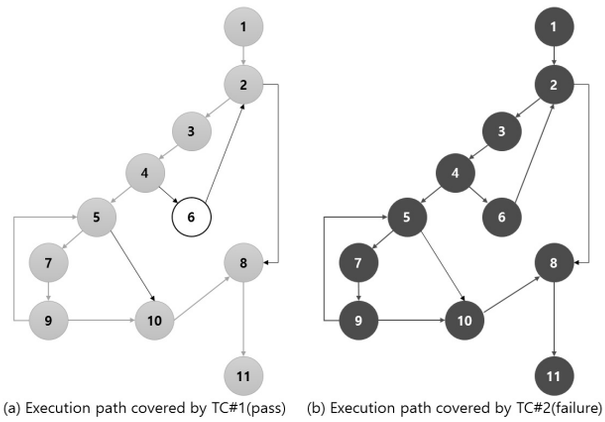
```
int main() {
    ……
    while (1)    {
        scanf("%d", &n);
        if (n < 0)
            printf("wrong");
        else {
            while (1)    {
                a = a + 1;
                printf("%.0f\n", pow(2, (a - 1)));
                if (a >= n)      break;
            }
            printf("=%.0f", pow(2, n) - 1);
            break;
        }
        return 0;
    }
}
```

Fig. 7. The Source Code Feedback of the 1st Submitted Answer

Fig. 6(a)의 TC#1은 채점 통과했고 Fig. 6(b)의 TC#2는 채점 실패했다. 실행 결과를 봤을 때, 분기 커버리지는 100% 달성했으므로 분기 구조의 결함이 없음이 확인되었다. 논리적 결함은 실패한 실행 경로를 확인할 필요가 있으며 특히 노드 4, 노드 6, 노드 8을 확인할 필요가 있다고 프로그래밍 실패 피드백을 Fig. 7과 같이 소스코드 상에 강조할 수 있다. 빨간색으로 음영처리된 부분이 노드 4, 노드 6, 노드 8에 해당하는 부분이다.

다음으로 Fig. 8과 9는 학생이 두번째 제출한 답안 소스코드의 제어흐름도, 테스트케이스별 실행경로와 답안의 피드백 소스코드이다. Fig. 8(a)의 TC#1은 채점 통과했고 Fig. 8(b)의 TC#2는 채점 실패했다. 첫번째 답안 소스코드와 동일하게 분기 커버리지는 100% 달성했으므로 분기 구조는 결함이 없음이 확인되었다.

논리적 결함은 실패한 실행 경로를 확인할 필요가 있으며 특히 노드 4, 노드 6을 확인할 필요가 있다고 프로그래밍 실패 피드백을 Fig. 9와 같이 소스코드 상에 강조할 수 있다. 두번째 제출된 답안 소스코드를 확인해본 후, 흥미로운 점은 첫번째 프로그래밍 실패 피드백에 해당하는 노드 6과 추가적으로

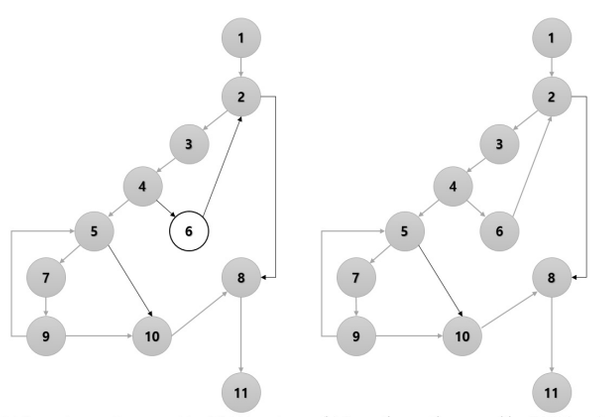


(a) Execution path covered by TC#1(pass) (b) Execution path covered by TC#2(failure)

Fig. 8. The Execution Paths of the 2nd Submitted Answer

```
int main() {
    ……
    while (1)    {
        scanf("%d", &n);
        if (n < 0)
            printf("wrong");
        else {
            while (1)    {
                a = a + 1;
                printf("%.0f\n", pow(2, (a - 1)));
                if (a >= n)      break;
            }
            printf("=%.0f", pow(2, n) - 1);
            break;
        }
    }
    return 0;
}
```

Fig. 9. The Source Code Feedback of the 2nd Submitted Answer



(a) Execution path covered by TC#1(pass) (b) Execution path covered by TC#2(pass)

Fig. 10. The Execution Paths of the Final Submitted Answer

노드 10이 수정되었다는 것이다. 하지만 여전히 노드 6을 포함한 실행경로에서 프로그래밍 실패로 판정되었으므로 노드 6이 올바르게 고쳐지지 않은 것을 알 수 있다.

```
int main() {
    ……
    while (1)    {
        scanf("%d", &n);
        if (n < 0)
            printf("wrong");
        else {
            while (1)    {
                a = a + 1;
                printf("%.0f\n", pow(2, (a - 1)));
                if (a >= n)      break;
            }
            printf("=%.0f", pow(2, n) - 1);
            break;
        }
    }
    return 0;
}
```

Fig. 11. The Source Code Feedback of the Final Submitted Answer

마지막으로 세번째 제출한 답안 소스코드의 제어흐름도와 테스트케이스의 실행경로는 Fig. 10으로 나타낸다. Fig. 10(a)의 TC#1와 Fig. 10(b)의 TC#2 모두 채점 통과했으므로 구조적인 결함과 논리적인 결함은 없다. 결론적으로 마지막으로 제출된 답안 소스코드는 정답으로 판정된 소스코드로 Fig. 11과 같이 피드백 소스코드를 확인할 수 있다. 두번째 제출된 답안과 동일하게 세번째 제출된 답안 소스코드에서도 두번째 프로그래밍 실패 피드백에 해당하는 노드 6에 해당하는 부분이 고쳐진 것을 발견했다. 따라서 본 논문에서 제안하는 프로그래밍 실패 피드백을 참고하면 학습자가 디버깅하는데 도움을 줄 수 있다는 것을 알 수 있다.

5. 결 론

본 논문에서는 기존의 프로그래밍 훈련을 지원하기 위한 프로그래밍 채점시스템에서 제공하지 않는 프로그래밍 실패에 대한 피드백 시스템을 제안하였다. 또한 실제로 학생이 제출한 답안 소스코드에 적용하여 제안한 시스템이 학습자가 자신이 작성한 소스코드의 구조적인 결함과 논리적인 결함을 찾아내는데 도움이 될 수 있음을 보였다. 향후 연구로는 실제 사용 중인 채점시스템에 프로그래밍 실패 피드백 시스템을 적용하여 제안한 시스템이 학습자의 디버깅에 도움을 줄 수 있음을 정량적으로 확인하고 프로그래밍 실패 피드백 시스템을 확장하고자 한다.

References

[1] U. Y. Jung, J. M. Han, and Y. J. Lee, “Analysis on Domestic Research Trends related to Bebras Challenge,” in *Proceedings of the Korean Society of Computer and Information*

Winter Conference, Gumi, 2019. pp.207-210.
[2] J. H. Jeon, U. Y. Jung, and Y. J. Lee, "A Problem Solving Learning Model using Online Judge System," in *Proceedings of the Korean Society of Computer and Information Summer Conference*, Jeju, 2018. pp.417-418.
[3] H. J. Park, C. S. Im, S. H. Park, J. Y. Hong, and M. H. Park, "Multi-facetted Automatic Coding Evaluation and Feedback System," in *Proceedings of the KIISE Korea Computer Congress*, Busan, 2017. pp.1051-1053.
[4] J. H. Park, Y. H. Shin, and S. G. Lee, "Online Judge System for SQL Practice," in *Proceedings of the KIISE Korea Computer Congress*, Jeju, 2016. pp.1054-1056.
[5] S. J. Yi, G. H. An, S. Y. Yun, and S. H. Lee, "A Web-based Algorithm Development Practice Tutoring System," in *Proceedings of the Proc. of the Korean Institute of Communications and Information Sciences Fall Conference*, Seoul, 2016. pp.150-151.
[6] G. P. Wang, S. Y. Chen, X. Yang, and R. Feng, "OJPOT: online judge & practice oriented teaching idea in programming courses," *the European Journal of Engineering Education*, Vol.41, No.3, pp.304-319, 2013.
[7] ACM UVA [Internet], https://www.uva.onlinejudge.org
[8] Baekjoon Online Judge [Internet], https://www.acmicpc.net
[9] Y. M. Woo, J. W. Bang, J. M. Song, J. Y. Yoo, and S. J. Lee, "Design and Implementation of the Web-based Learning System for C Programming Language," *The KIISE Transactions on Computing Practices,* Vol.20, No.12, pp.640-645, 2014.
[10] KNU Online Judge [Internet], http://selab.knu.ac.kr/training
[11] S. Y. Park, "Design and Implementation of a Web-based Programming Class Support System," *The Journal of the Korea Institute of Maritime Information & Communication Sciences*, Vol.14, No.12, pp.2275-2782, 2010.
[12] Codeup [Internet], http://www.codeup.kr
[13] T. Kakeshita, K. Ohta, and R. Yanagita, "A Programming Education Support Tool pgtracer utilizing Fill-in-the-Blank Questions," *International Conference on Education Reform and Modern Management*, Hong Kong, 2015, pp.164-167.
[14] DOMjudge [Internet], https://www.domjudge.org
[15] Algospot Online Judge [Internet], https://www.algospot.com
[16] Programmers [Internet], https://www.programmers.co.kr
[17] Sphere Online Judge [Internet], http://www.spoj.com
[18] KOI [Internet], http://www.koistudy.net

이 성 희
https://orcid.org/0000-0003-3508-7719
e-mail : lee3229910@gmail.com
2013년 경북대학교 컴퓨터학부(학사)
2015년 경북대학교 컴퓨터학부(석사)
2015년~현 재 경북대학교 컴퓨터학부 박사과정
관심분야 : Embedded Software & Distributed Computing Testing

김 덕 엽
https://orcid.org/0000-0003-1680-1278
e-mail : ejrduq77@naver.com
2016년 경북대학교 컴퓨터학부(학사)
2018년 경북대학교 컴퓨터학부(석사)
2018년~현 재 경북대학교 컴퓨터학부 박사과정
관심분야 : Embedded software testing & Drone

서 강 복
https://orcid.org/0000-0003-3716-700X
e-mail : seokang13@naver.com
2014년 국가평생교육진흥원 컴퓨터공학(학사)
2017년 경북대학교 컴퓨터학부(석사)
2018년~현 재 경북대학교 컴퓨터학부 박사과정
관심분야 : Embedded Software Testing & Concolic Testing

이 우 진
https://orcid.org/0000-0002-8075-5248
e-mail : woojin@knu.ac.kr
1992년 경북대학교 컴퓨터학부(학사)
1994년 KAIST 전산학과(공학석사)
1999년 KAIST 전산학과(공학박사)
1999년~2002년 ETRI 선임연구원
2002년~현 재 경북대학교 컴퓨터학부/소프트웨어기술연구소 교수
관심분야 : Embedded Software Testing & Embedded Software Development Environment