VNU Journal of Science: Comp. Science & Com. Eng., Vol. 32, No. 3 (2018) 1–8

"Smart Codefun" A tool supporting programming skills for students

Toan Tran Duc, Pham Ngoc Hung

Faculty of Information Technology, VNU University of Engineering and Technology, No. 144 Xuan Thuy Street, Dich Vong Ward, Cau Giay District, Hanoi, Vietnam

Abstract

This paper demonstrates how to apply automated test data generation techniques and base knowledge to support students' programming skills. From the limitations of CodePower, this paper proposes a tool for supporting the process of learning programming for students and lecturers named "Smart CodeFun". Specifically, "Smart CodeFun" would support lecturers to create test cases automatically from the given source code rather than writing test cases manually as in CodePower. This tool helps lecturer to save time designing test cases and avoid missing some special test cases. With students, "Smart CodeFun" will calculate the student's score based on the number of correct test cases. If the point is not maximized, the tool will automatically highlight source code that can lead to an error. Based on these suggestions, students are more likely to find fault in source code. Finally, "Smart CodeFun" supports graphical flow control charts corresponding to the source code. That graph is a visual view of the logic of the source code.

Received May 2018, Revised May 2018, Accepted May 2018

Keywords: Smart codefun, Detect error source C/C++, improve skill programming.

1. Introduction

At present, the application of information technology in teaching is an inevitable trend and is being implemented in many training units. In this trend, deploying applications to help improve students'programming skills proved to be very useful. The method increases the excitement of student with the lesson, so the student more and more understand lessons. In addition, this solution helps the training units reduce the volume of teaching practice. At University of Engineering and Technology, VNU-UET, Codepower[1] has been applied in the teaching of lecturers. Codepower is a tool for students to improve their programming skills with a variety of different levels of problem. Since the implementation of Codepower, the application has greatly helped improve student's programming skills and reduce the workload of lecturers. When using Codepower, lecturers give the issue in each level or the lessons to the application. Corresponding to these issues is test cases

[1]https://codepower.vn/login/index.php

designed by lecturers which use to evaluate the student. These test cases must cover all cases of the issue. In case the user is a students, they go to read the issue then solve it. Solutions to these exercises must run through all test cases of lecturer to get results. The issue is solved when the answer is correct for all test cases.

In fact, Codepower still has some problems, the test cases, which the lecturer provided, may not cover all the cases of the issue. These test cases are used to evaluate the student's ability, directly affect the student's score and perceptions. Therefore, test cases need to have some criteria to build. At the same time, students send the source code to the codepower, students only get pass or fail results with the test cases. The notification of pass and fail sometimes will cause difficulties for the new student in programming. Sometimes, students met too many wrongs without knowing troubles in the source code, it will cause psychological depression. If we have an error displaying mechanism, and provide a visual view of the source code for students, it will help students greatly in learning, improving their education.

This paper solves two issues by applying method generating test case in the CFT4Cpp[2] tool. The CFT4Cpp tool is an automated test generation tool for a source code project written in C/C ++. This tool was developed by the team of University of Engineering and Technology, VNU in cooperation with TSDV. CFT4Cpp, when compared to other tools, has more advantages than CREST[1], KLEE[2], DART[3], CAUT[4], etc. This paper builds "Smart Codefun" to support students in C/C ++ programming language. This tool solves two main problems that correspond to two user using the Codepower tool:

- The generate test case for the issue from source code (with the lecturers): Build a tool with the same functionality as Codepower, but instead of having a lecturer submits test cases, lecturer only need to send the standard source code on the tool. "Smart Codefun" will be based on that standard source code to generate the standard test case for the problem. The technique of applying test coursework in tool is inherited from CFT4Cpp.

- Warnings trouble in the source code (with the student): The tool uses test cases generated from the source code of lecturer to conduct the assessment and display of warnings trouble in student's source code. This may be the cause of the problem. At the same time, "Smart Codefun" creates a flow chart from the students' source code to help students have a visual view. Students using the tool will reduce the time spent solving errors, increasing excitement when learning the program. Addressing this issue will be a big step forward in promoting thinking, training many good programmers.

The rest of this paper is organized as follows. At first, Section 2 shows method generating test cases from standart source code. Next, Section 3 introduces how to evaluate and waring for student. The details of building cfg are presented in Section 4. Section 5 describes a tool that was

[2]https://uet.vnu.edu.vn/ hungpn/CFT4Cpp

implemented from the proposed method and experimental results. Finally, the conclusion of the paper is presented in Section 6.

2. The method of generating auto test cases from standard source code

In this section, method of automatically generate test cases from the standard source code of lecturer is presented. After that, this paper suggests some constraints for input source code.

2.1. Generating test cases

Generating test cases is an important step, and there are many ways to generate test cases. Experience showed that CFT4Cpp has many advantages. At the same time, author of this paper also a member of the tool development team. So, the paper decided to use the CFT4Cpp API to generate test cases. CFT4Cpp only generates input(test datas), and according to the definition of the test, we lack the corresponding output. "Smart Codefun" finds output that is automatically generated based on the input. The results of the above process will be saved to the database for using later.

Algorithm 1 describes the "Smart Codefun" tool automatically generates test cases. The input to this algorithm is the source code of the test function. Then, function will be integrated into the template project (line 2). In this, we call the API CFT4Cpp with two parameters passed is the path of the sample project $projectPath$ and name of function need generate test cases $nameFunction$, and the returned result is set of inputs $inputs$ of the function (line 4). For each input of the function, the function will execute to get the corresponding output, and the tool has already obtained a set of test cases (lines 6, 7, and 8). Algorithm 1 ends when the entire input of the function run over. These test cases were produced according to branch coverage. Branch coverage well enough in testing, not wasting time.

Algorithm 1 Generate test case

```
    Input: Function need generate test cases
    Output: Test cases
1: Initilize $testcases \leftarrow []$,
2: add $function$ into template project
3: $projectPath \leftarrow$ path to template project
4: $inputs \leftarrow APICFT4Cpp(projectPath, nameFunction)$
5: for each $input \in inputs$ do
6:     $ouput \leftarrow function(input)$
7:     $testcase \leftarrow (input, output)$
8:     $testcases.add(testcase)$
9: return $testcases$
```

2.2. Constraints on input source code

Programming languages have a certain complexity, many different expressions. The C and C ++ languages are almost the same, so the source code has a lot of different writing. For the convenience of generating test cases from source code, the source code should follow some constraints. List of source code constraints:

- The source code must be only a single function (due to the small issue, we only need to write source code in a single function). The function will be written according to the standard Dev-C[3] can compile.

[3]https://sourceforge.net/projects/orwelldevcpp/

- The function must not contains keywords such as auto, template.

- Accepted libraries are: stdio.h, conio.h, string.h, math.h, iostream, fstream, string.

3. Evaluating, warning the problem in the source code

In this section, how to evaluating student's source code is present. Then, algorithm warning problem in that source code.

3.1. Evaluating source code

Figure 1 describes the algorithm for evaluating student's source code. First, students' source code attached to the template project to execute. Test datas generated from lecturer's source code will be used as inputs of the function. For each input, the student's source code will return an actual output. We compare this actual output with the expected output. If the result is similar, the student will be earn points. The loop will be continued until the last test case executed. Point will be reported to students.

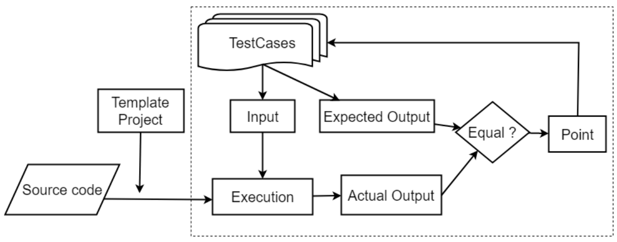


Figure 1. Evaluating source code

3.2. Warning problem in the source code

The student's score isn't highest, it means that the source code still needs to be fixed. Finding their problems is quite fast, but sometimes it takes a lot of time and annoys students. Figure 2 presented the algorithm. First, source code is instrumented, Then,it is added to a template project. Each test case is fail, the function will be compiled and execute again. With instrument technique, we know statements run over when source code execute. All failed test cases run, we get all statements. After that, the tool will remove the duplicate statements to obtain a set of statements. They are the statements that may cause the fail. The tool looks for seasons of problems in the source code and warn the students by highlighting these statements and showing control flow graph.

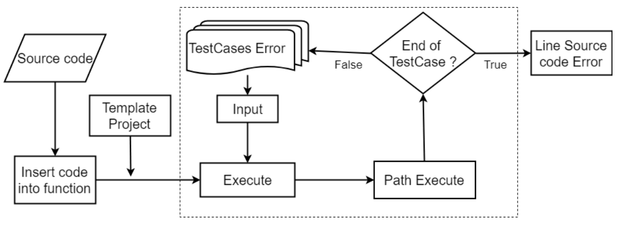


Figure 2. Warning the problem in the source code

3.3. Function Instrumentation

To get the path's information when you running with an input, it is necessary to implement a technique to insert a tick statement into the function. Tha techique is function instrumentation. Once the structure tree is complete, some tick statements have to be inserted into the function. When the function is executed, they will write a list of statements, that have been run through, into an external file. Table 1 presents rules for inserting tick statements into functions. Function *mark* is instrumental function and inserted in each executed statement. The $< A >$ represents the content of A. In essence, *mark*(" $< A >$ ") will write content A to a predefined file.

4. Build Control flow graph (CFG)

CFG graphing is a function developed for both the lecturer and student user. The CFG graph is drawn from the source code analysis of the tool then using the go.js library:

- The server of the tool analyses student's source code by using analysis techniques of CDT. Then it calculate position of each vertex in graph. Finally, server generate json send to clients.
- On the browser side (client), the received json data will be displayed by the go.js library, which is the CFG of the function we need draw.
- Lecturer: For each test case, the CFG graph illustrates the execution path that the test case runs over.
- Student: When the source code has an error, the CFG graph will highlight the source code that can lead to errors.

This CFG can be drag and drop, zoom arbitrarily to fit the view of the user. A json fragment consists of two main sections, the *linkDataArray* and the *nodeDataArray*. *nodeDataArray* is the array that contains the vertices of CFG graph. Each element is an object that includes:

- *key*: represents the vertex and unique.
- *text*: content of the statement.
- *highlight*: Color of vertex.
- $figure$: This attribute appears if the vertex is a conditional statement
- $locWidth$, $locHeight$: Coordinate position of drawing vertex

linkDataArray is an array cointains information linking between vertices (edge). Each element is an object that contains:

- $from$: *key* of the start vertex
- *to*: *key* of the end vertex
- *text*: content of edge.
- $fromPort$: The position of the start vertex.
- *toPort*: The position of the end vertex.

5. Tool and Experiment

5.1. The architecture of tool

The tool has been developed using Java, JSF framework, hibernate framwork, Jquery library, Boostrap and Go.js to realize the idea. Besides, "Smart Codefun" reuses source code of the CFT4Cpp tool to generate test datas and exports data for the user. The tool serves two main kinds of user: lecture and students.

Figure 3 describes the architecture of the tool in actual deploymention. Tools has two layer is client side (browser) and server side. The client side (browser) consists of two main components: CFG Visualizer and

Type of block (A)	Insert tick statement
Assign, declare, throw/ break/continue/return, {, }.	mark("<A>"); A;
while(<condition>) {...}	while (mark("<condition>") &&<condition>) {...}

do {...}
while (<condition>)
	do {...}
	while (mark("<condition>")&& <condition>)

if (<condition 1>){...}
else if (<condition 2>){...}
else {...}
	If (mark("<condition 1>") &&<condition 1>){...}
	else if (mark("<condition 2>") && <condition 2>){...}
	else {...}

for(init, condition, increment){...}	For (mark("<init>") && init, mark("<condition>") && condition, mark("<increment>") && increment){...}

try {...}
catch (<exception 1>){...}
catch (<exception 2>){...}
	mark("try"); try {...}
	catch(<exception 1>){ mark("<exception1>"); ...}
	catch(<exception 2>){ mark("<exception 2>");...}

Table 1. List of instrumentation rules

support libraries. CFG Visualizer uses the Go.js library to display CFG for the source code. We will understand the source code that we are writing. On the server side, the tool includes modules: User authentication, Json generator, Test cases generator, Error detector and output collector. User authentication module uses the Hibernate framework to interact with the database. The server uses the API of the CFT4Cpp tool to generate test data then Output collector get them for Test case generator module. The Json generator analyzes the source code into a corresponding json structure, from which is passed to the user to plot the flow graph. In addition, highlighting the path of CFG, the Json generator find matchest statement to highlight. Finally, the error detector, which evaluate source code and detects input errors, executes source code in template project to show student point and if it warning path execution, it has error.

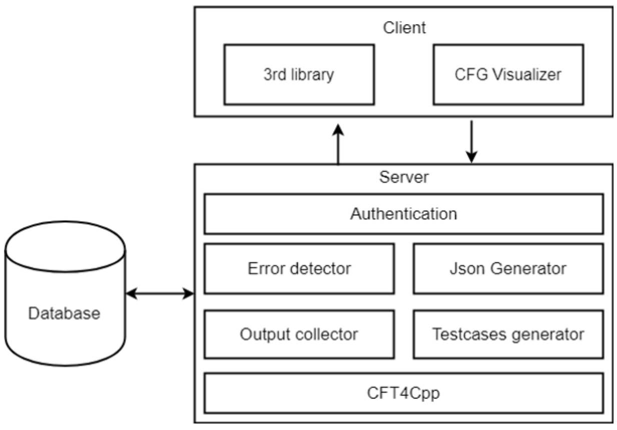
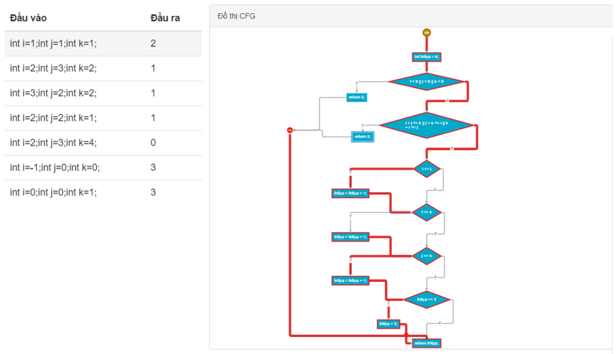


Figure 3. Architecture of tool

5.2. *A case study*

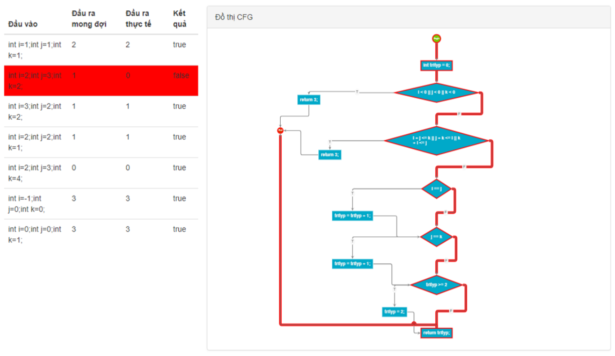
In order to show the correctness and usefulness of the tool, "Smart Codefun" was tested with simple source code. In this sections, a case study with issue: "Identify the type of a triangle when lecture provide length of three edges". This issue is simple but it has a lot of cases. So, student can miss some cases when solving.

Figure 4 show all test cases and CFG when lecturer submit issue with source code. Lecturer can choose a test case then CFG will highlight path execution. With two views, lecturer can control his/her test cases. A CFG will be built with highlighted statements which may cause the problem.



Đầu vào	Đầu ra
int i=1;int j=1;int k=1;	2
int i=2;int j=3;int k=2;	1
int i=3;int j=2;int k=2;	1
int i=2;int j=2;int k=1;	1
int i=2;int j=3;int k=4;	0
int i=-1;int j=0;int k=0;	3
int i=0;int j=0;int k=1;	3

Figure 4. Teacher use tool generate test cases



Đầu vào	Đầu ra mong đợi	Đầu ra thực tế	Kết quả
int i=1;int j=1;int k=1;	2	2	true
int i=2;int j=3;int k=2;	1	0	false
int i=3;int j=2;int k=2;	1	1	true
int i=2;int j=2;int k=1;	1	1	true
int i=2;int j=3;int k=4;	0	0	true
int i=-1;int j=0;int k=0;	3	3	true
int i=0;int j=0;int k=1;	3	3	true

Figure 5. Student use tool to check source code

Figure 5 shows how student use the tool to study. Student solve the issue and submit it into tool. "Smart CodeFun" auto analyze source code then show the set of test cases to students. If source code not correct, error test cases will be highlight.

6. Conclusion

The application of automated tools to lecturing and learning, especially in difficult subjects such as programming is becoming more and more popular. The more difficult issue and the longer source code the more time will be taken for lecturers evaluating or students correcting source code. Codepower has provided a very good solution to reduce the time and effort, but it can not help students to find their mistakes. This paper proposes a solution using CFT4Cpp to improve Codepower, creating a new tool that supports not only lecturer but it also helps students in learning programming. "Smart CodeFun" is used to generate test cases for the issue, specifically it automatically generates input based on the lecturer's source code, then it runs this input to obtain the corresponding output. The set of inputs, outputs are stored as the basis for scoring the source code of students. Incorrect executed commands will be flagged by the markup engine during rerunning the source code with incorrect input. Finally, a CFG graph will be generated to give students a more systematic view of their source code and to know which commands are likely to be wrong, reduce the time and effort needed to fix the problem.

However, the newly built tool is not well-tested for performance as well as load tolerance of the application. Furthermore, "Smart CodeFun " use CFT4Cpp's API, so the types which are not supported in CFT4Cpp are not available in this tool.

Now, the tool has completed for basic functions. In time to come, this tool will be upgraded to support more complicated cases. The tool may also apply black-box testing in conjunction with white-box to test the issue: boundary tests, special test cases, etc. In addition, the tool can detect error details in statement (currently it only warn for each path execution). The interface of the tool will also be improved friendly, easier to use for everyone. We hope the tool will become more popular in learning and lecturing information technology, and find more ways to improve the tool more convenient.

Acknowledgments

We thank Dr. Vo Dinh Hieu, VNU University of Engineering and Technologies for his reviews.

References

[1] J. Burnim, K. Sen, Heuristics for scalable dynamic test generation, in: Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, ASE '08, 2008, pp. 443–446.

[2] C. Cadar, D. Dunbar, D. Engler, Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs, in: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI'08, USENIX Association, Berkeley, CA, USA, 2008, pp. 209–224.

[3] P. Godefroid, N. Klarlund, K. Sen, Dart: Directed automated random testing, SIGPLAN Not. 40 (6) (2005) 213–223.

[4] Z. Wang, X. Yu, T. Sun, G. Pu, Z. Ding, J. Hu, Test data generation for derived types in c program, in: 2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering, 2009, pp. 155–162.