LogicVis User Manual

CSE 403 Software Engineering

Winter 2019

Team LogV

Software Description

LogicVis is a program that takes an input of a piece of Java function code, primarily recursive functions, and outputs a visual representation of that code.

Description

LogicVis is fundamentally a program that helps people understanding code and particularly recursion. It takes Java code and transforms it into an easy-to-understand flowchart that reflects the logic of the code. It comes with a user-friendly interface with simple and intuitive operations so that a person with any level of technical background is able to use it. This program is completely free to use.

Benefits and Values

- Our program generates flowcharts with easy to recognize shapes representing each conditional statements and loops.
- Our program provides an exhaustive visualizations of the recursive calls. (assuming the recursion is not infinite)
- Our program enables progress tracing with current instruction pointer, variable value indicator, and function call stack frames.

Installation

If you are looking to use this software, download LogicVis.jar from the GitHub repository (https://github.com/orenjina/LogicVis) and run it.

System Requirements

Before using this software, the following must be installed:

- Java 8 (https://java.com/en/download/)

Building from Source

Required software: Maven, JDK 11
Run “mvn clean install” from terminal in the project repository.

Using the System

This software is able to translate a Java method into a corresponding flowchart that has the same meaning. From there, it can track variables throughout the method and recursive method calls, if any.

Please do not use this tool before ensuring that the method has no compiler errors.

Understanding the Interface

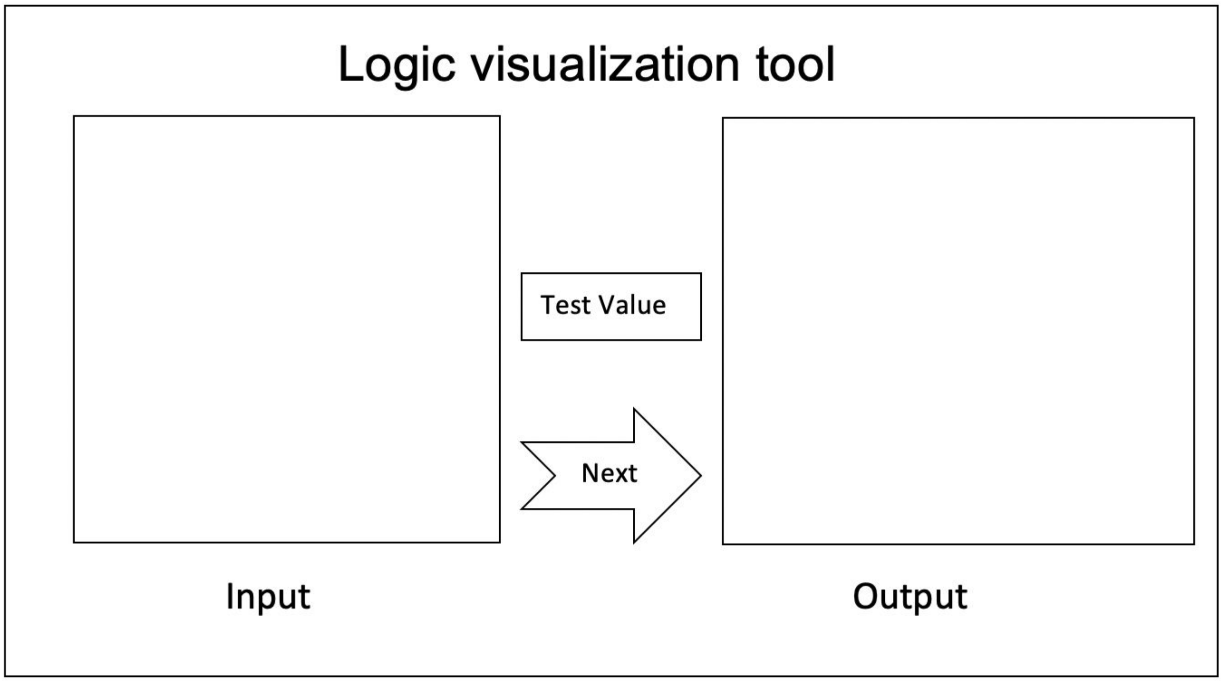


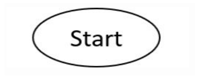
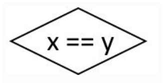
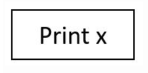
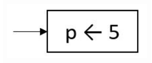
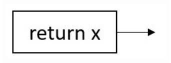
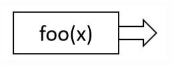
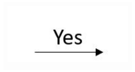
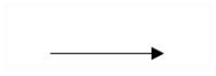
Fig. 1 - LogicVis tool as seen on startup

The Logic Visualization tool interface consists of a few parts:

- **“Input” box** in which users can type the recursive function that they want to translate
- **“Test Value”** button which will let the user set the parameters of their method
- **“Next”** button which will do the translation
- **“Output” box** that will display a flowchart

Reading the flowchart

In the output flowchart, there are shapes and arrows with different meanings.

Name	**Figure**	**Description**
Ellipse		Denotes the entry into the method as well as the beginning of a cycle, such as a loop or recursive call.
Diamond		Denotes a branch in the flowchart, such as an if statement or switch-case statement. Multiple conditioned arrows will branch from this shape.
Rectangle		Used to represent any piece of code that does not uniquely affect the control flow of the program. This includes variable setting, return values, and many others.
Arrow Inwards		Denotes a parameter, as written inside the box. These are only found before Start.
Arrow Outwards		Denotes a return value, as written inside the box.
Big Arrow		Indicates a recursive method call. After putting in a test value, clicking the arrow will expand the method call.
Conditioned Arrow		Denotes the next step, if the previous statement matches the condition. Always originates from a diamond shape. - Yes means the previous condition was true, and No means it was false
Unconditioned Arrow		Denotes the following step.

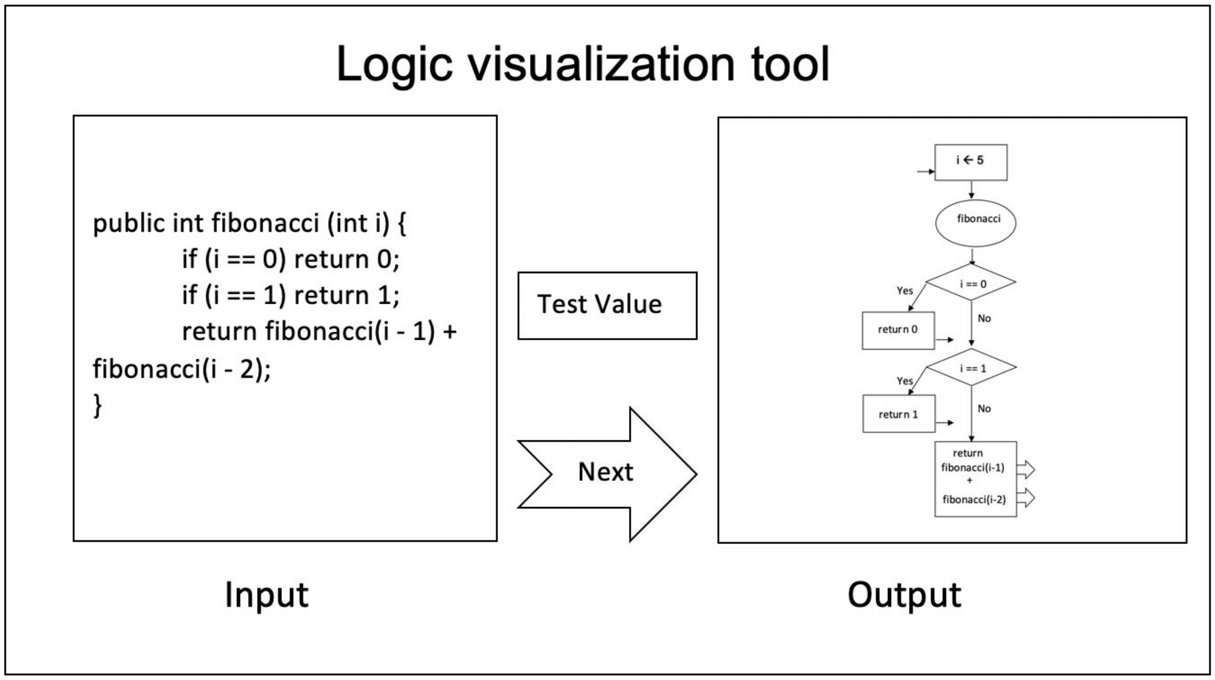
Generating a Flowchart from a Java Method

This section details the process to take a Java method, translating it into a flowchart, and how to read the flow chart.

Generating the Initial Flowchart

1. **Copy recursive function into the input box**

Copy your method into the **“Input” box** of the tool, as seen in Fig. 2.



```
public int fibonacci (int i) {
          if (i == 0) return 0;
          if (i == 1) return 1;
          return fibonacci(i - 1) +
fibonacci(i - 2);
}
```

Fig. 2 - Write your function in the input

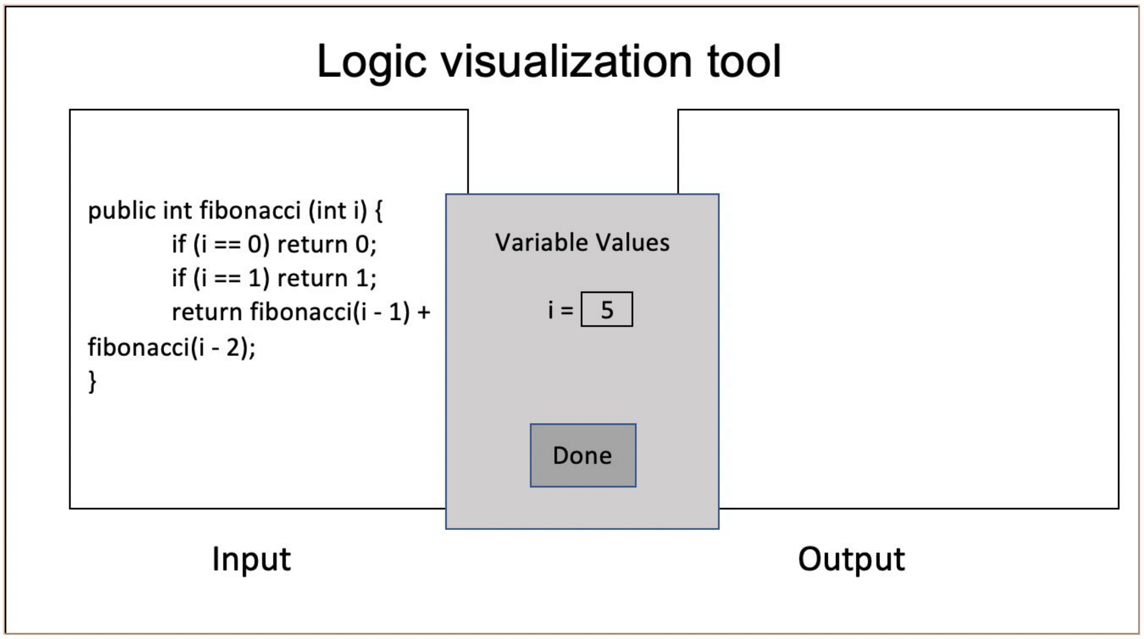
If an error occurs, a message will be displayed instead of the graph. This can happen for multiple reasons:

- Compiler errors exist in the code
- The method could not be read as Java code
- Unknown syntax was used

2. **Press “Test Value”**

Pressing **“Test Value”** will ask you to give a input value to the parameters. If there is no errors in the input function, it will generate a graph like Fig. 2

Warning: pressing **“Test Value”** at any time will reset the current state of the flowchart to reflect the given method.



```
public int fibonacci (int i) {
        if (i == 0) return 0;
        if (i == 1) return 1;
        return fibonacci(i - 1) +
fibonacci(i - 2);
}
```

Fig.3 - Give an input value

3. **To expand recursive calls, click "Next"**

It is possible to step into a recursive call within the recursive method. This is done simply by clicking the **"Next"** button

For example, Fig. 4 shows the output section of the software. After **clicking "Next"** the graph will show the recursive function in a extended graph in fibonacci(i - 1) and the variables in the recursive function will be replaced with updated variables.

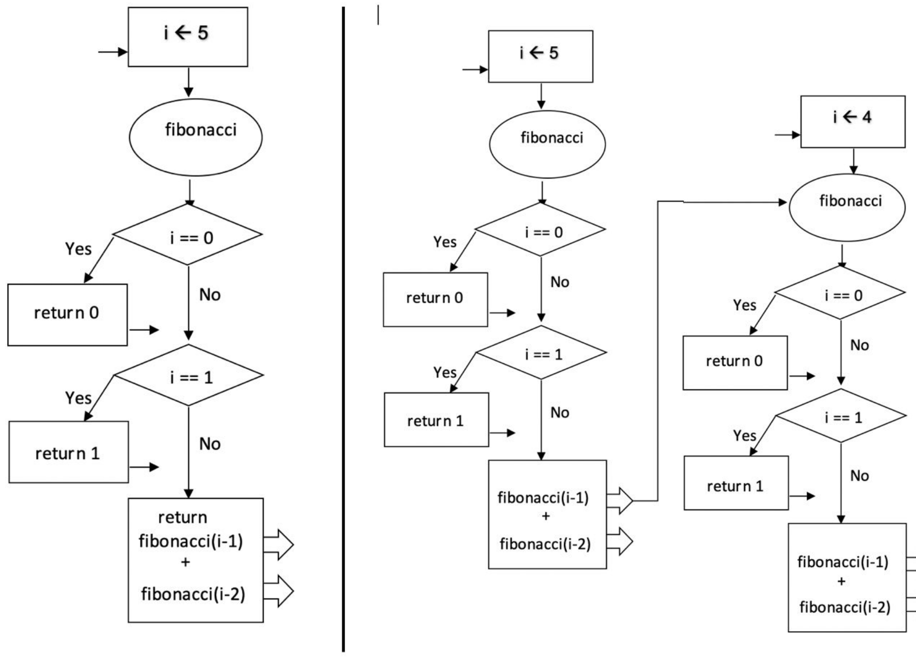


Fig. 4 - the output graph generated in Fig. 2 (left), and the extended graph (right)

4. **Keep clicking "Next" will keep extending the graph**

Every time clicking **"Next"** will extend the graph with the next function call (Only one function call at a time), but once **"Test Value"** is clicked, it will reset everything and ask you for the input value. When the graph hits the based case and start returning, it will start shrinking and replace the function call with the returned value displayed in color red. Fig. 5 is an example when *fibonacci(i-1)* in the first depth gets its value 3, it will be replaced with 3 and start expanding *fibonacci(i-2).*

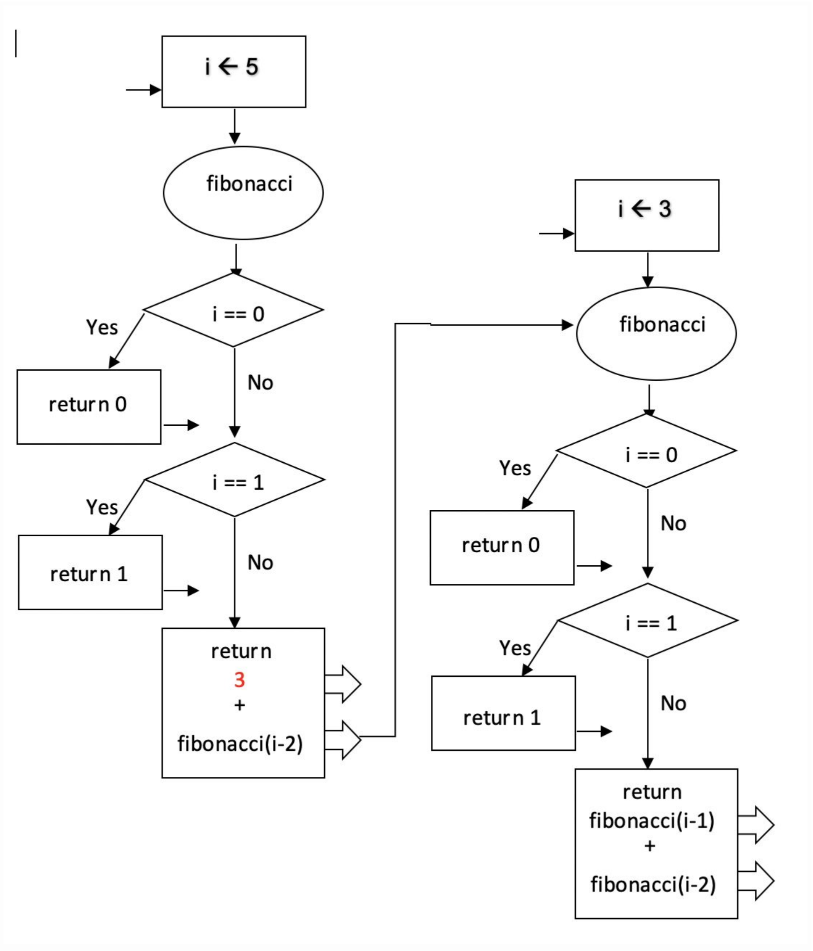


Fig. 5 - the output graph after finishing *fibonacci(i-1)* in the depth i ← 5