

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 logic graph.

Description

LogicVis is fundamentally a program that helps people understanding code. It comes in handy especially when people try to understand recursive functions. It reads in the Java code line by line and turns each line into a node representation with the shape indicating the type of command. It integrates the nodes generated from the code and form a flowchart to visualize the logic of the code. It also comes with a user-friendly interface with simple and intuitive operations that eliminates the technical burden of using it so that people with any technical backgrounds are able to use it. This program is completely free to use and requires no commitment.

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.

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.

Generating a Flowchart from a Java Method

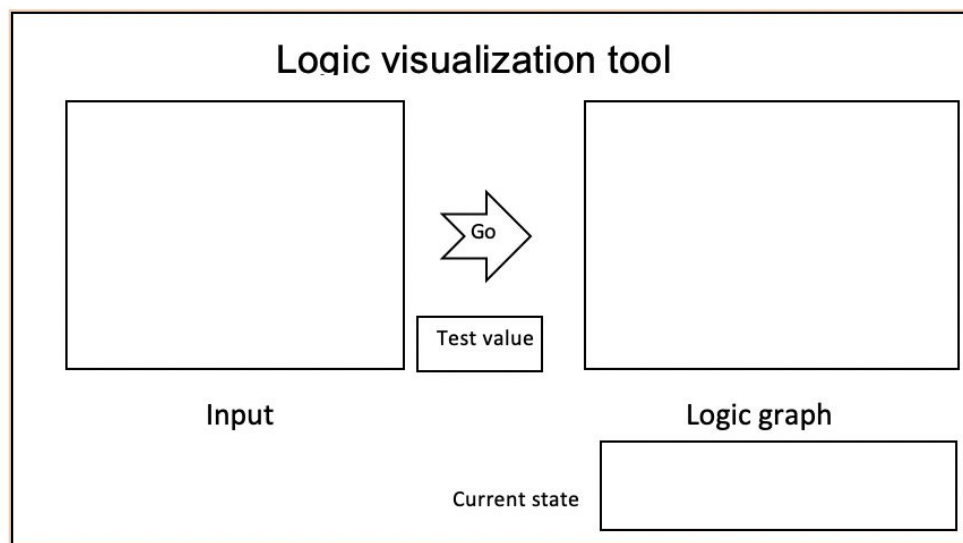

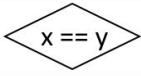

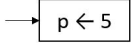
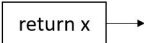
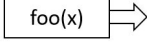
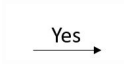



Fig. 1 - LogicVis tool as seen on startup

This section details the process to take a Java method, translating it into a flowchart, and how to read the flow chart. Fig. 1 shows the interface of the tool. Input is written in the Input box on the left, and the logic graph is displayed in the box on the right. The current state of the parameters is displayed in the box below the logic graph.

Reading the flowchart

In the 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.
Conditioned Arrow		Denotes the next step, if the previous statement matches the condition. Always originates from a diamond shape. <ul style="list-style-type: none">- Yes means the previous condition was true, and No means it was false
Unconditioned Arrow		Denotes the following step.

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.

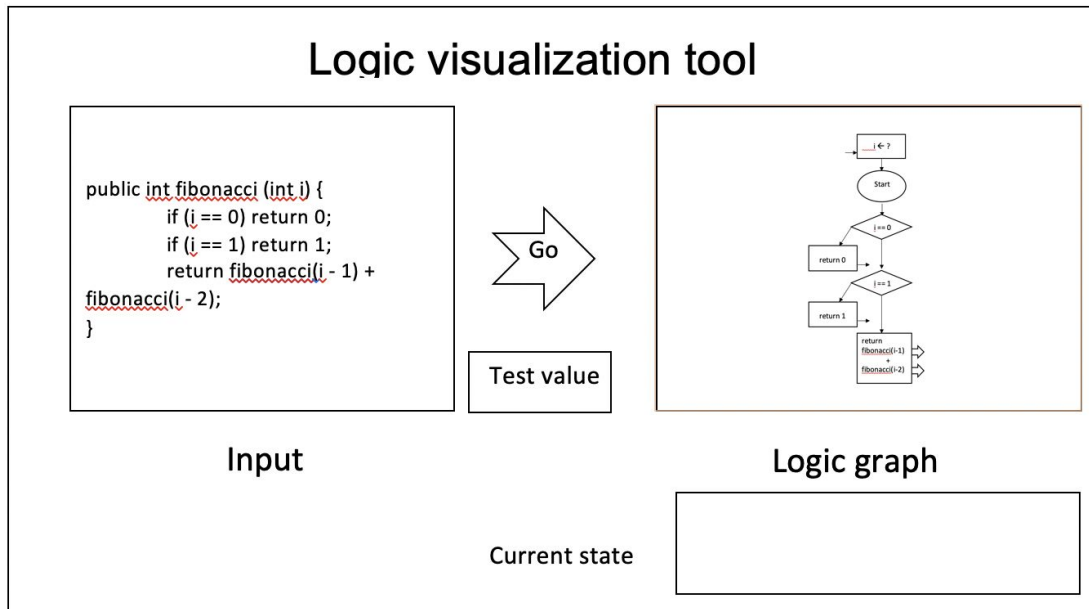


Fig. 2 - A flowchart generated from the code after pressing “Go”

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
- Unknown syntax was used

2. Press “Go”

If no errors exist within the method, pressing “Go” should generate the flowchart as expected.

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

3. To expand recursive calls, click the arrow next to the recursive function

It is possible to step into a recursive call within the recursive method. This is done simply by selecting one of the big arrows (\Rightarrow) that denote a recursive method call in the output box.

For example:

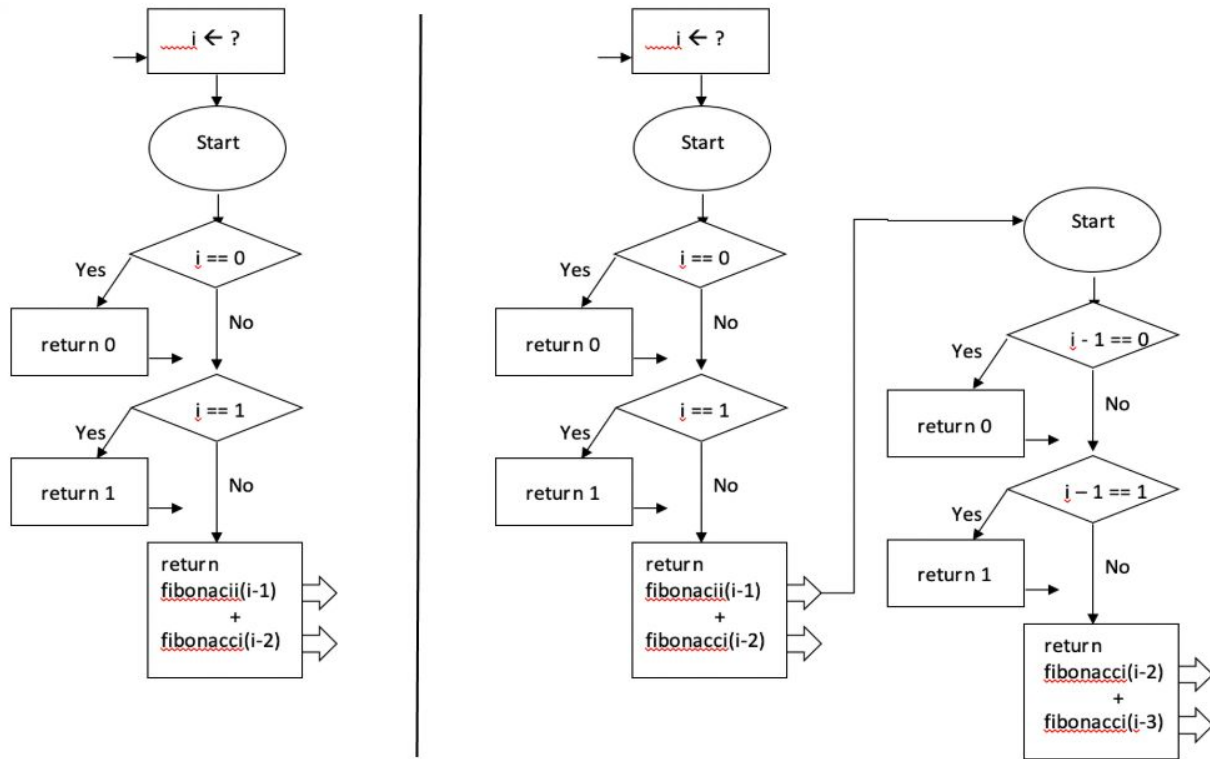


Fig. 3 - the output graph generated in Fig. 2 (left), and the extended graph (right)

Fig. 3 shows the output section of the software. After **clicking arrow** (\Rightarrow) next to $\text{fibonacci}(i-1)$ the graph will show the recursive function in an expanded graph and the variables in the recursive function will be replaced with updated variables.

In the example below, inside the recursive function graph variable i is replaced by $i-1$.

Tracking Variables and Code Execution

After generating a graph, setting custom parameters is possible. By doing so, tracking the program flow is possible.

This functionality may be disabled if unknown data types are used. The currently supported data types are as follows:

- All primitive data types
- String
- Array

1. Variable setting

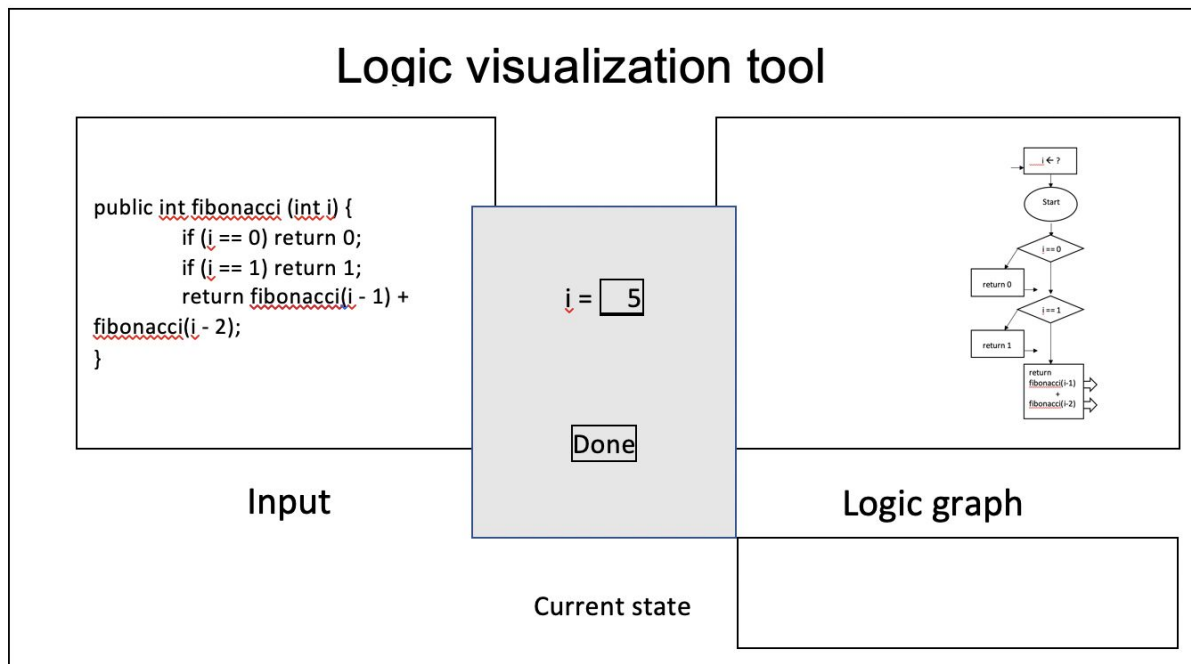


Fig. 4 - the prompt that appears after selecting "Test Value"

After generating a graph, a "Test Value" button should appear beneath the "Go" arrow. Selecting it will open a prompt (Fig. 4) to initialize the parameters of the method. Pressing "Done" will update the graph to reflect the given parameters, and the program will switch to the *tracking mode*.

Tracking Mode

In *tracking mode*, users are able to track the the program behavior step by step with the given input value. The current step block in the output graph will be marked in **yellow**. The "**Current state**" will display the values of current variables in the current step.

2. Click the output graph to step forward

Users are able to step forward to next block by clicking the output graph, and next block will be marked in **yellow** and the current state will be updated accordingly.

After stepping into the recursive calls, the program will show the extended graph automatically and move into that graph.

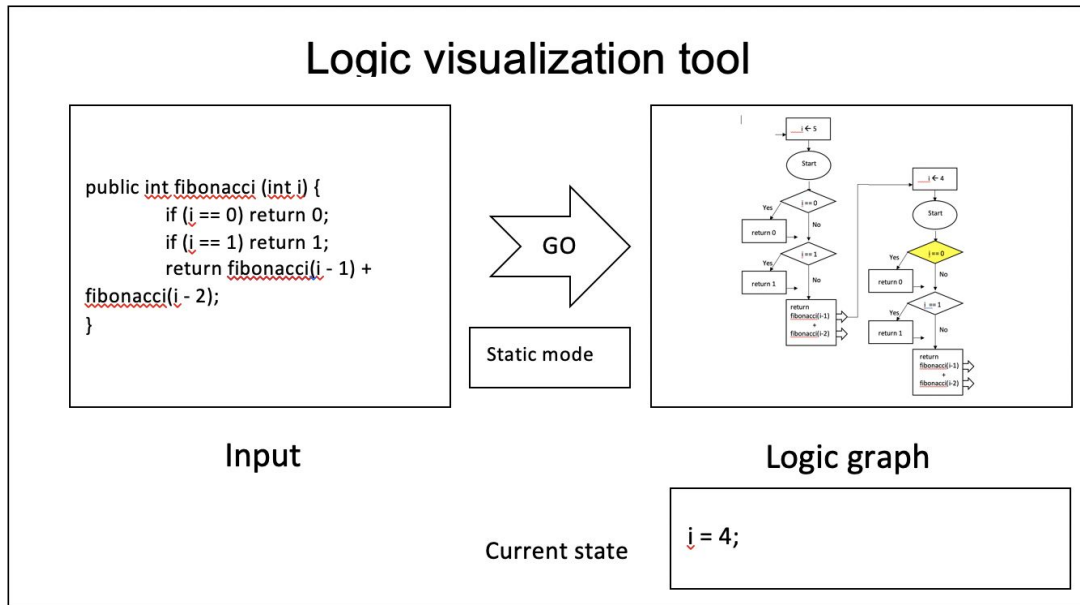
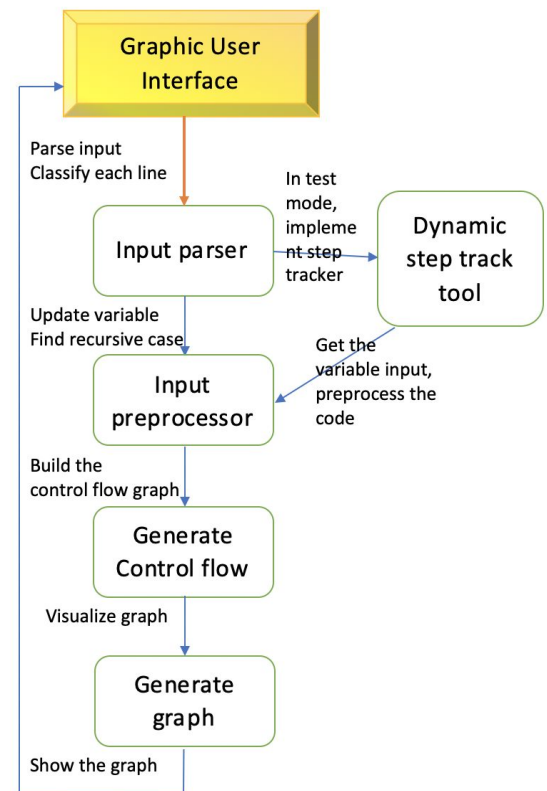


Fig. 5 - state of the example flowchart after stepping into recursion

Implementation details

Architecture

For this project, for the static recursive function logic graph generator, we decided to implement with two layers: the Graphic User Interface Layer, and the Graph Generator Algorithm Layer. The Graphic User Interface is first displayed to the user with the option of inputting code, as the users passing in the code to be visualized and the values they want to test out. Then it goes to input parser which will parse the input (recursive function), it will display error if the code is not valid Java code. If it is the static mode the input preprocessor will preprocess input code in the static rule, if it is dynamic mode, it will implement a dynamic step track tool which allows user to execute code step by step, then the preprocessor will preprocess input code in the dynamic rule. After preprocessing, control flow generator will process the logic and generate the control flow, then the graph generator will generate a graphic output based on the control flow. Lastly, the graph will be displayed in the GUI.



Technologies

We have chosen to develop this software in Java since we are most familiar with the Java language and its libraries.

Graphic User Interface:

We plan to use the JavaFX library, which provides a clean graphical UI that works as a standalone. The older libraries such as Swing and AWT do not provide as much functionality whereas others like Pivot are used as RIA.

Graph Generator Algorithm:

We plan to build this from scratch. We want to be able to make a parsing and graph generation software that fits our Graphic User Interface, so although we may reference existing flowchart generators, we would have to rewrite most of it if we wanted to use one.