## Introduction

**Using interactive graphical models to visualize software variability**
**Recruitment Screener**

Thank you for your interest in our research study assessing strategies to visualize the differences in variants of a configurable program. We are looking for participants who have an understanding of how configurable programs are designed. The purpose of this screening questionnaire is to ensure that you meet the prerequisites of the study with respect to understanding of the fundamental concepts behind the information that the tool will be presenting. Only individuals who answer 4 out of 5 questions correctly will be invited to participate in the research study. It is important to note that, in the study, you will not be evaluated in any way and it is the tool that is under scrutiny, not you. We will contact you to let you know whether you meet the prerequisites and to schedule your study session. Your information will be deleted in case you do not meet the prerequisites of the study.

### Are you interested in participating

- ○ Yes
- ○ No

### Screening exercise: Configurable Program Assessment

This page is a refresher of some concepts related to configurable programs and program variants.

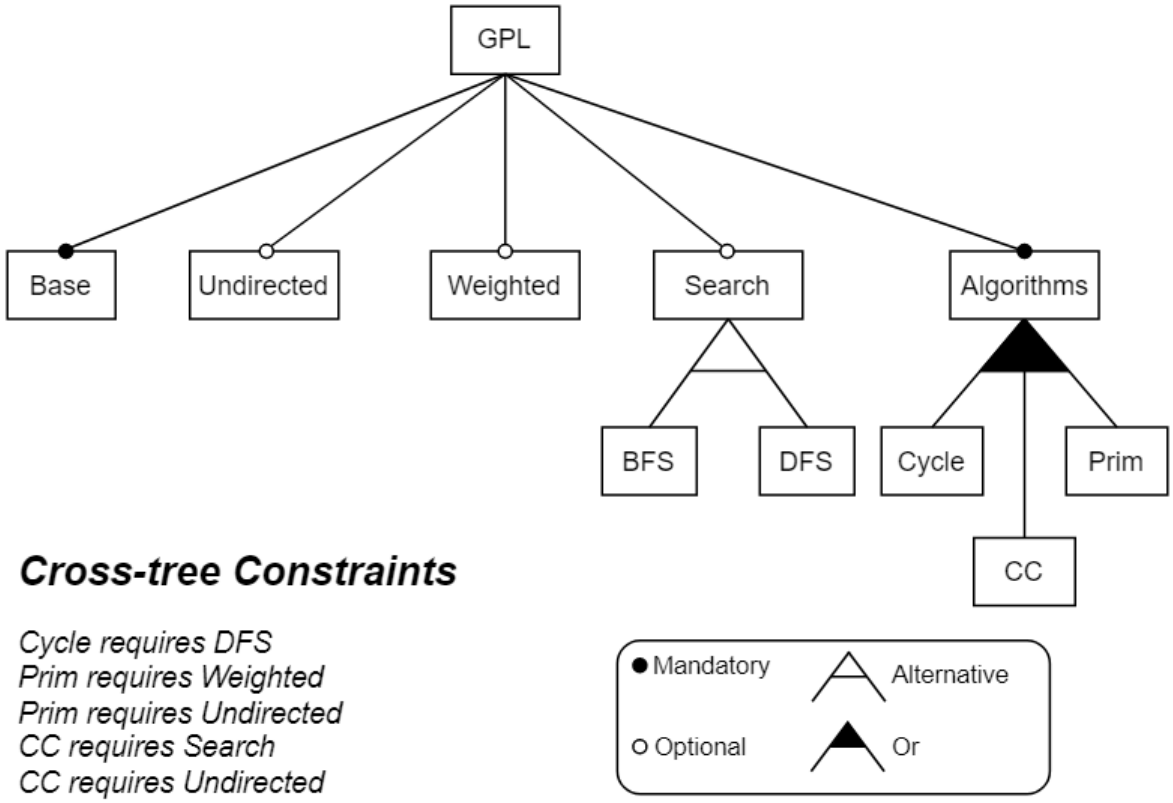
## Configurable program

Software configuration is a fundamental aspect of software development. It is the ability to create software variants for different contexts of use. Engineers can create a configurable program that encompasses multiple, different variants that share a set of common features.

At the level of the code, variability occurs by enabling or disabling portions of the code that implements certain software features. The engineer can accomplish this by creating conditional structures that reason over the command-line options provided by the user. In C++, such conditions can be established by using preprocessor directives for conditional inclusion of parts of the code during the compilation.

## Feature model

At the level of the specification of the system, feature models represents a configurable program and its possible program variants. During the experiment, you will be inspecting the data representing the implementation of a system called Graph Product Line. This program generates variants of a Graph Application that includes at least one graph algorithm. The graph algorithms considered for this example are a cycle detection algorithm, a connected components (CC) algorithm, and Prim's minimum spanning tree.



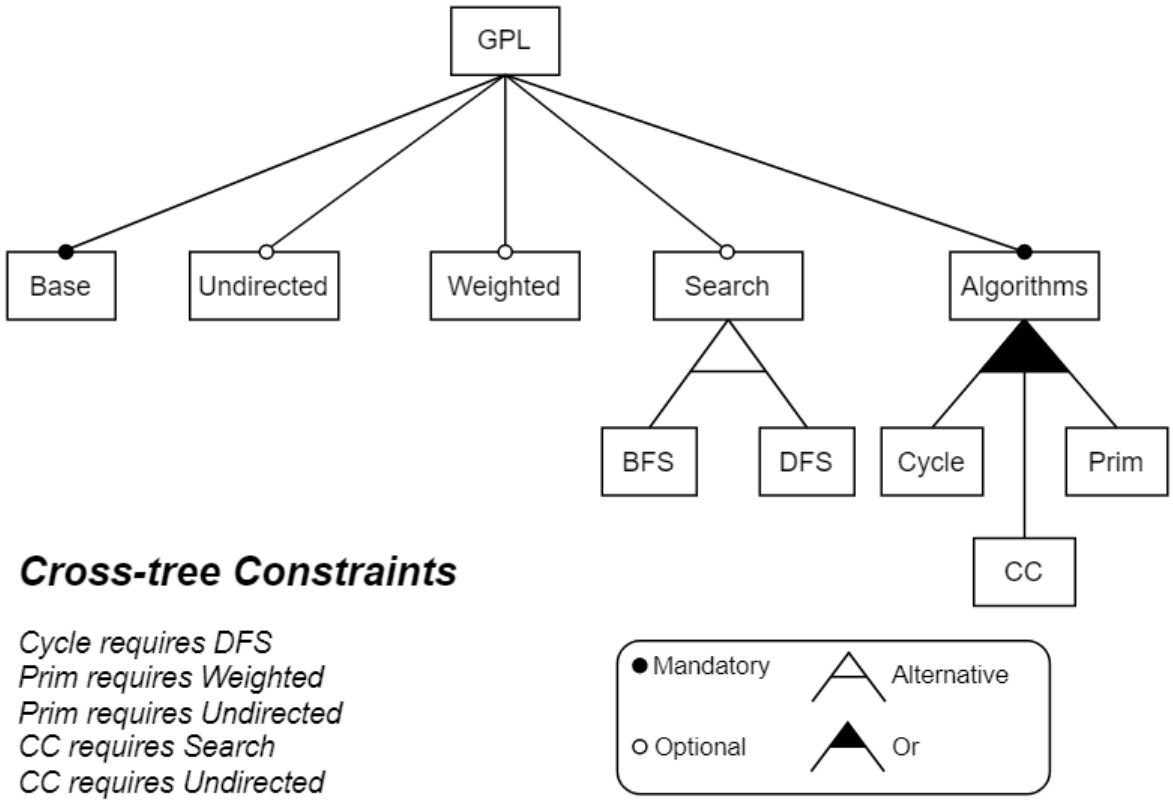

The configurable program and its variants are specified by the feature model above. The root node represents the configurable program comprising the program features. Features can be mandatory or optional in a program variant. And child nodes indicate the variations of the feature represented by the parent node. The legend in the bottom-right corner shows the symbols used to define constraints on the configurability of a parent node in terms of child-node selections. For example, the white dot on top of the Undirected box indicates that feature is optional. Thus, a program variant can be undirected or directed. According to the model, the features Base, GraphType, and Algorithms are mandatory. Thus they must be present in all variants of the program. The features Search, which implements search algorithms, and Weighted, which implements weighted edges, are optional. The black arc between the features Cycle, CC, and Prim, indicates that at least one of Algorithms' child features must be included in a generated program variant. The white arc between the features breadth-first search (BFS) and depth-first search (DFS) indicate that only one of them can be enabled in a software variant. The feature model also includes cross-tree constraints (shown in the bottom-left corner) that define constraints on the configuration choices among features that are not directly related in the Feature Model's tree structure. Cross-tree constraints typically express dependencies between features (i.e., the selection of one feature "requires" the selection of the other feature) and mutual exclusion between

features (i.e., the selection of one feature “excludes” the selection of the other feature). In this configurable program, for example, variants that include the feature Cycle must also include DFS as its search algorithm.

**Questionnaire**

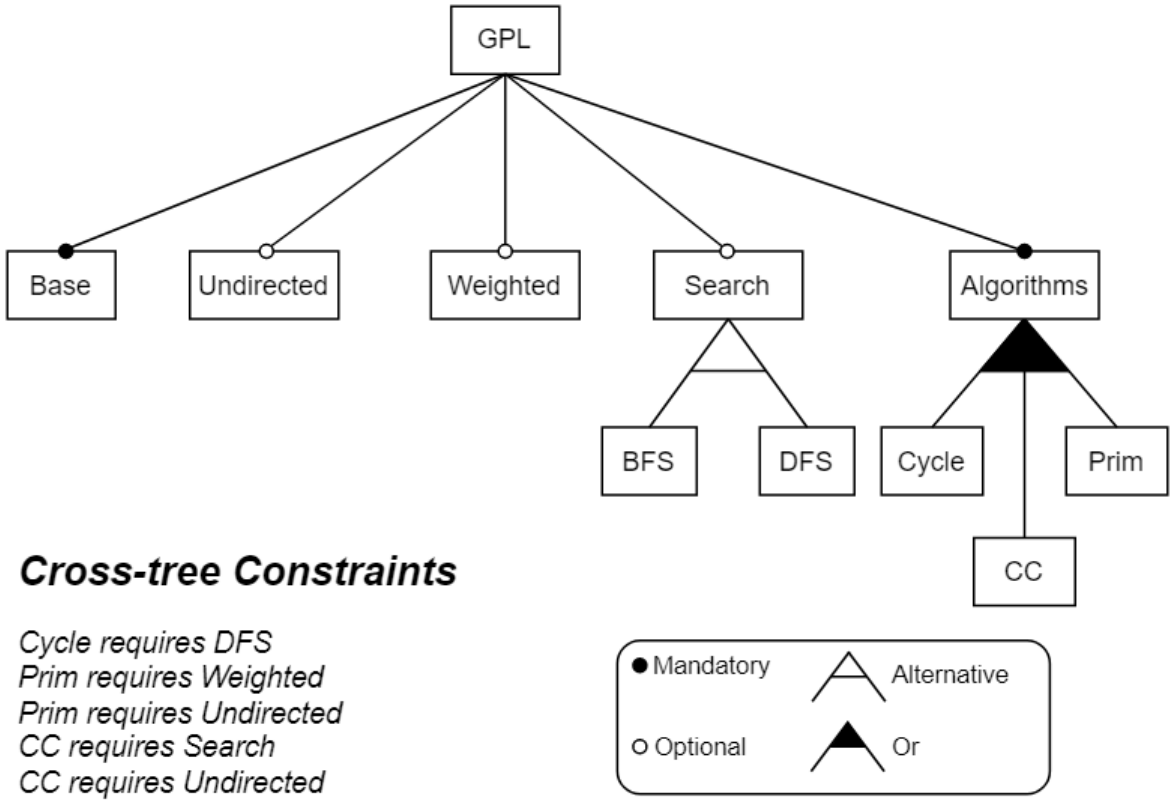
The following questions will test your comprehension of the configurable program represented by the Graph Product Line feature model.. There is only **one** correct answer to each question. To participate in our study, you must answer correctly at least 6 out of the 7 questions.

In the questions below, program variants (representing feature selections) are expressed as Boolean expressions over features, where a feature name represents a selected feature, negation (!) followed by a feature name represents an unselected feature, conjunction (/\) denotes that both feature operands are selected, and disjunction (\/) denotes that at least one of the feature operands is selected.




If CC and Cycle are both selected as features in a given program variant, which state of the feature Undirected is valid for that program variant?

- ○ Disabled
- ○ Enabled
- ○ Either of the above



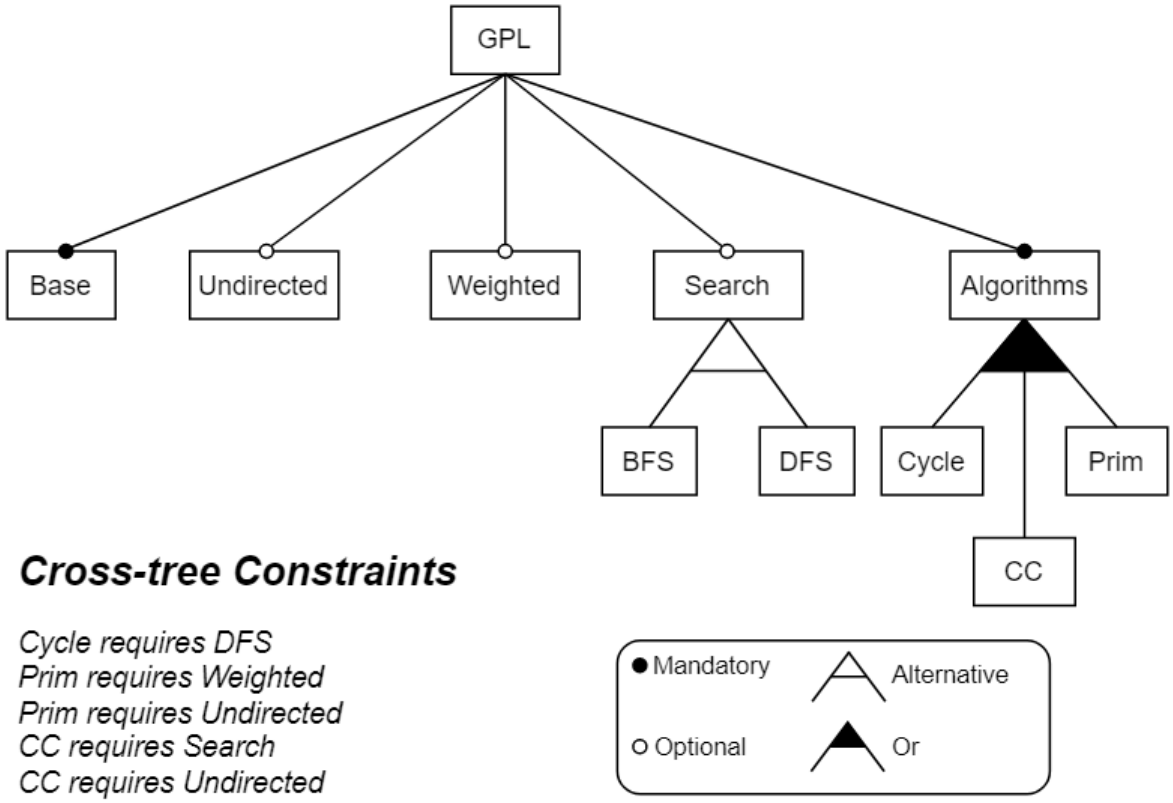

**Cross-tree Constraints**

*Cycle requires DFS*
*Prim requires Weighted*
*Prim requires Undirected*
*CC requires Search*
*CC requires Undirected*




If Weighted is enabled and Undirected is disabled in a given program variant, which Algorithms feature is a valid selected feature for that program variant?

- Connected components (CC)
- Prim
- Cycle



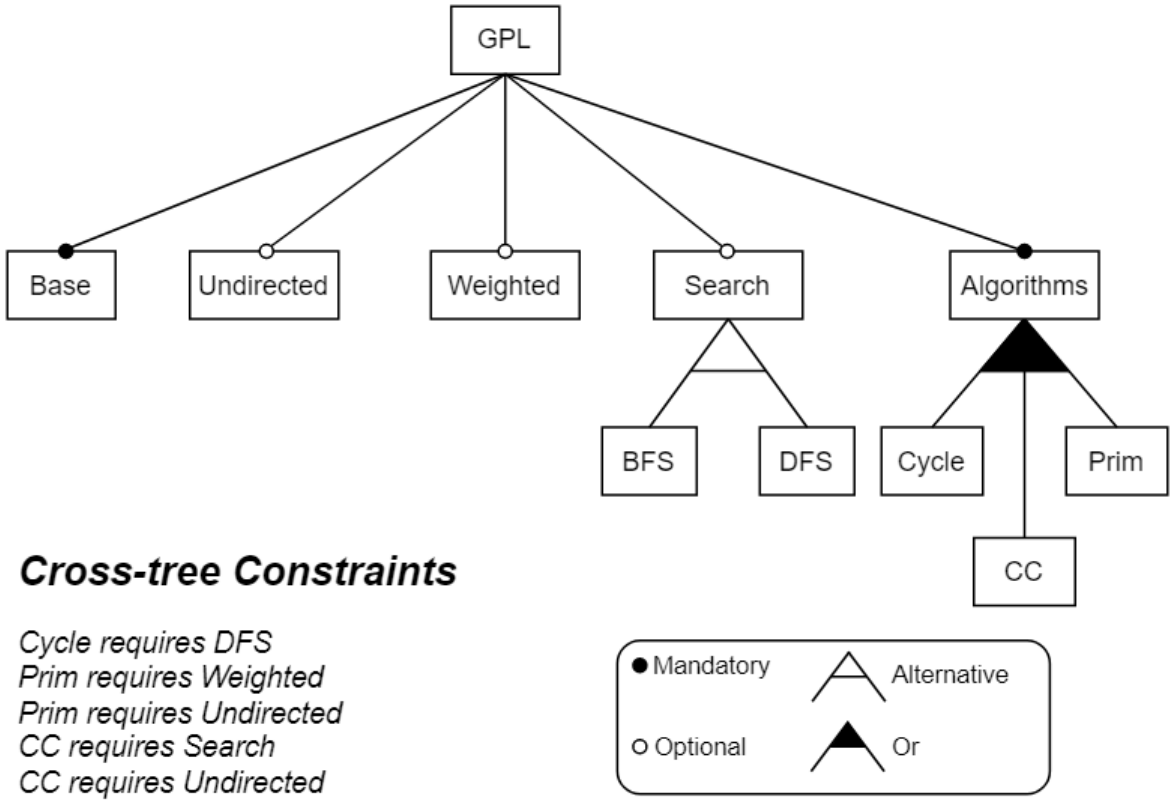

**Cross-tree Constraints**

*Cycle requires DFS*
*Prim requires Weighted*
*Prim requires Undirected*
*CC requires Search*
*CC requires Undirected*




Considering the features Search, Algorithms, and their respective child features, which of the alternatives below presents a valid selection of features for a program variant?

- !BFS /\ !DFS /\ !Cycle /\ CC /\ !Prim
- BFS /\ !DFS /\ Cycle /\ CC /\ !Prim
- !BFS /\ !DFS /\ !Cycle /\ !CC /\ Prim



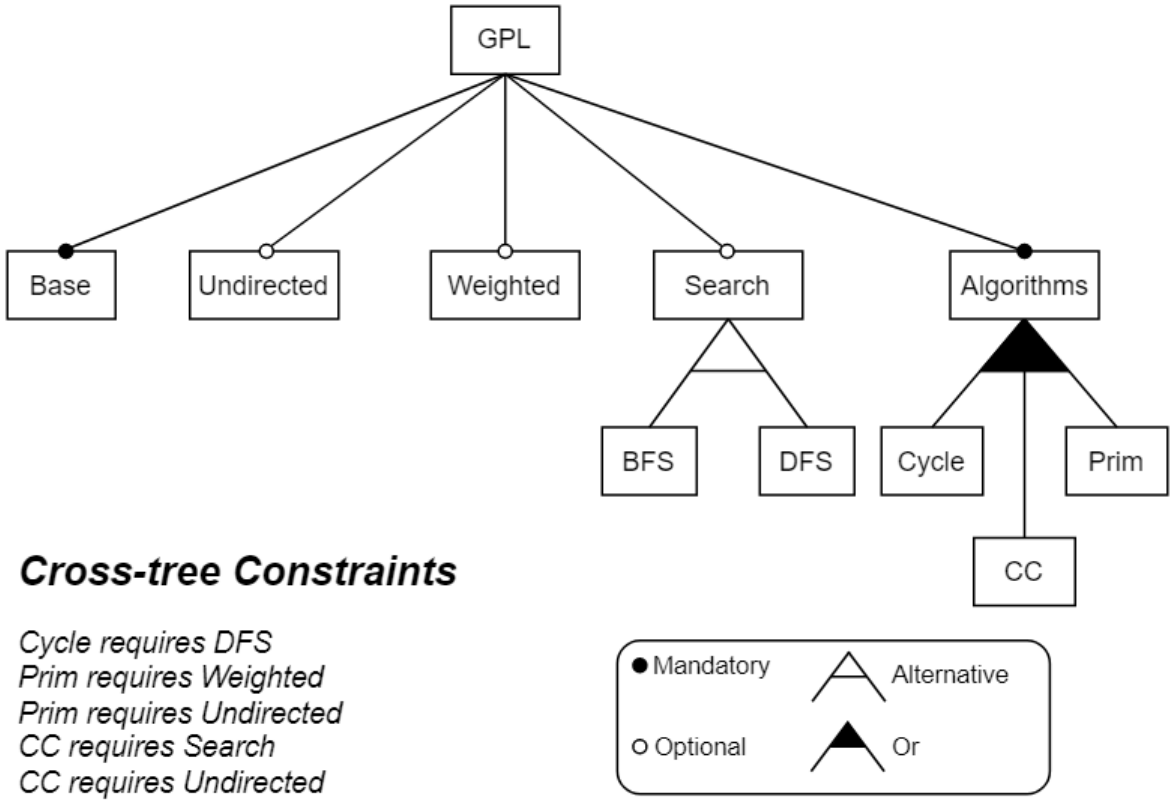

**Cross-tree Constraints**

*Cycle requires DFS*
*Prim requires Weighted*
*Prim requires Undirected*
*CC requires Search*
*CC requires Undirected*



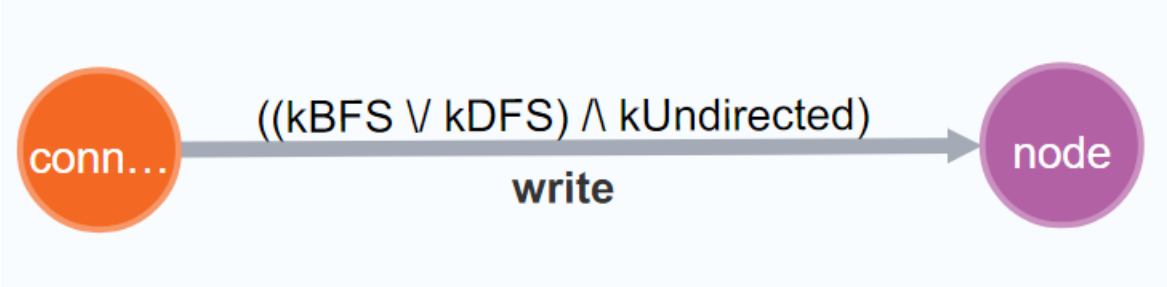

If all graph algorithms (Cycle, CC, and Prim) are selected as features in a given program variant, what would be a valid selection of features for that program variant?

- Base /\ Undirected /\ Weighted /\ DFS /\ !BFS
- Base /\ Undirected /\ !Weighted /\ DFS /\ !BFS
- Base /\ !Undirected /\ Weighted /\ DFS /\ !BFS



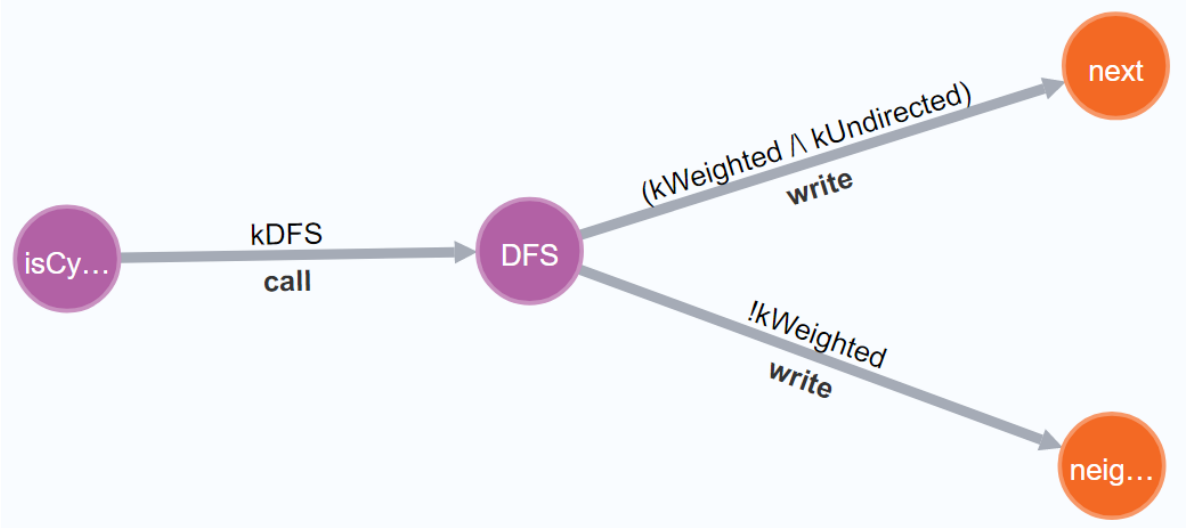

If Weighted is selected as a feature in a given program variant, what would be a boolean expression representing a set of valid program variants?

- Base /\ Undirected /\ Weighted /\ DFS /\ !BFS /\ (Cycle \/ CC \/ Prim)
- Base /\ !Undirected /\ Weighted /\ DFS /\ !BFS /\ (Cycle \/ CC) /\ !Prim
- Base /\ !Undirected /\ Weighted /\ (DFS \/ BFS) /\ !Cycle /\ (CC \/ Prim)




Consider that the edge above represents the value assignment executed by the function `connectedComponents` to the variable `node`. The top label shows the condition necessary for executing such an operation in a program variant. Which of the following variants may execute the variable write?

- !kBFS /\ !kDFS /\ kUndirected
- !kBFS /\ !kDFS /\ !kUndirected
- !kBFS /\ kDFS /\ kUndirected




Considering the graph above, which of the following paths may execute in a program variant with a feature configuration equal to !kBFS /\ kDFS /\ kUndirected /\ kWeighted

- (isCycle)-->(DFS)-->(next)
- (isCycle)-->(DFS)-->(neighbor)

Powered by Qualtrics