```
neo4j$
```

```
$ :play start
```

Tutorial

In this tutorial, you will learn how to use *Neo4j Browser* to visualize analysis results of configurable programs.

This interface comprises the top bar (the neo4j$ prompt above), where you can run queries over the database, and the list of visualization frames (not yet shown - these will be visible after you pose a query), in which you can visualize and inspect the results of the query. A visualization frame can be maximized to fullscreen and closed whenever you want by clicking the icons at the top.

This guide will show you how to:

1. Run a query about configurable program data
2. Customize the visualization of the results

Note that you are not expected to learn the query language. All required queries will be provided to you during the study. Click on the arrows on the sides or bottom of this visualization frame to navigate through the tutorial.

```
neo4j$
```

```
$ :play start
```

Graphical program data

A program comprises entities (e.g., classes, variables, functions) and the relationships between them (e.g., function calls, variable reads, class containment). A graph representing such a program includes nodes representing the entities and links indicating the relationships established in the code.

Consider a function that updates the *name* attribute of a *Node* object contained in the Graph Application program:

```
void GraphApp::updateName(std::string nodeName, std::string newName) {
   for (int i=0; i < nodes.size(); i++) {
       if (nodes[i]→getName() == nodeName) {
           nodes[i]→setName(newName);
       }
   }
}
```

Now click on the following query and hit the play button beside the top bar to create a new visualization frame with the graphical representation of this program.

```
MATCH (a:cFunction)-[b]→(c) WHERE a.label CONTAINS "updateName" RETURN *
```

```
neo4j$
```

```
$ :play start
```

Graphical program data

The new frame shows the data returned by the executed query. The sidebar on the right provides an overview of the node labels and relationships types present in the visualization.

You can reposition the nodes by dragging them around. If you hover or click on any node or link of the graph, the overview on the sidebar is replaced by the information associated with the selected element. To return to the overview, you need to deselect the clicked entity. You can do that by clicking on the background or the selected entity once.

You can pan and zoom the visualization in the frame if you want. You can pan around the graph view by clicking and dragging the background. You can zoom in and out by clicking on the buttons in the bottom right corner. You can also expand the visualization frame into fullscreen by clicking on the ↙↗ button on the top right corner.

As a first task, find the node that represents the function updateName. What is the id of such a node?

When you are ready, move to the next page to check your answer.

```
neo4j$
```

```
$ :play start
```

Graphical program data

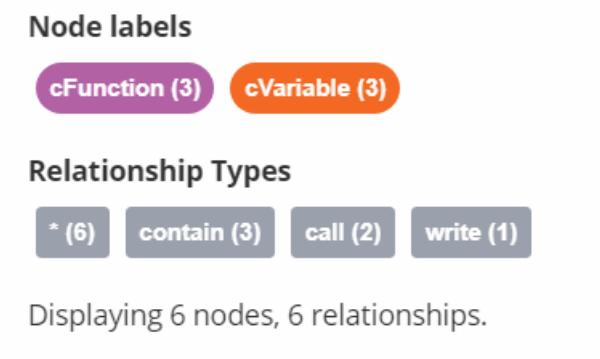
The new frame shows the data returned by the executed query. The sidebar on the right provides an overview of the node labels and relationships types present in the visualization.

The correct answer is **46**.

```
$ :play start
```

Customizing visualization

The sidebar provides customization options to change visual attributes of the nodes and links. The customization menu appears whenever you click on a node label or relationship type listed in the overview. The star sign (*) represents visual attributes applied to all links.



The customization menu includes the following options:

1. Turn on/off the visibility of nodes and links
2. Change the colours and diameter of nodes

```
neo4j$
```

```
$ :play start
```

Customizing visualization

The sidebar provides customization options to change the visual attributes of the nodes and links. The customization menu appears whenever you click on a node label or relationship type listed in the overview. The star sign (*) represents visual attributes applied to all links.

To experiment with the customization options, perform the following tasks on the query results:

1. Set the colour of cVariables to red
2. Set the colour of cFunctions to dark blue
3. Set the thickness of all the links to the fourth thickest option available

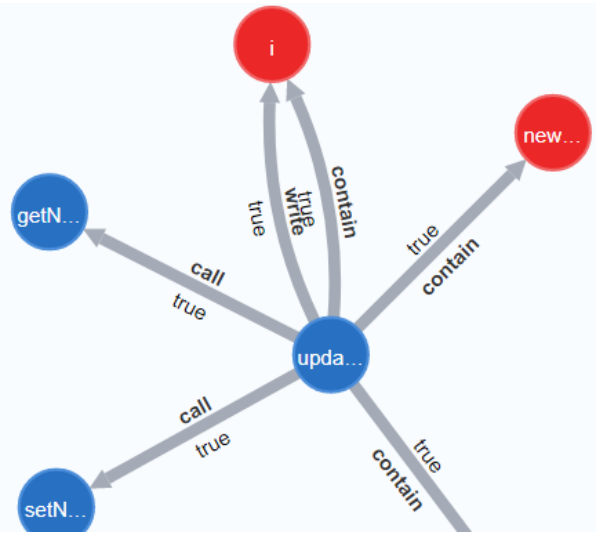
You can move to the next slide when you are done.

```
neo4j$
```

```
$ :play start
```

Customizing visualization

If you set the visual attributes correctly, your graph should look like the following:



```
neo4j$
```

```
$ :play start
```

Configurable program graph

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 program variants that share common code. Variability occurs by enabling or disabling portions of the code that implements optional software features.

In the example programs used in the study, boolean flags called feature variables represent whether the corresponding features are enabled or not. In this codebase, feature variables have the prefix 'k' in their names. For example, in the following code snippet, the feature*kUndirected* must be enabled, as well as, one of **kBFS** or **kDFS** for the instructions inside the condition block to execute.

```
void GraphApp::connectedComponents() {
    if ((kBFS || kDFS) && kUndirected) {
        clearVisited();
        int compNum = 3;
        ...
    }
}
```

```
neo4j$
```

```
$ :play start
```

Demo task

The tasks comprising this study will ask you to understand how entities and relationships vary in different program variants, by examining the corresponding graph of program data that includes conditional relationships labelled with presence conditions. As an example, consider the graph returned by the following query:

```
MATCH (a:cFunction)-[b]→(c) WHERE a.label = "DFS" AND a<>c aND b.condition <> "true" AND
type(b) = "call" RETURN *
```

Which of the program variants listed below include more function calls between DFS and other functions?

- V1: kWeighted /\ kUndirected
- V2: !kWeighted /\ kUndirected

```
neo4j$
```

```
$ :play start
```

Demo task

The correct answer is **V1**. In that variant, *DFS* may call the function *getTargetID* and *getSrcID*. Whereas in V2, *DFS* may only call the function *getNgbrs*. Take your time to go back to the graph to understand the correct answer, if necessary.

```
neo4j$
```

```
$ :play start
```

End of tutorial

Now that you are familiar with the interface, we can start the study.

When you are ready, close the visualization frames with results of previous queries (clicking on the 'X' at the right top corner of each frame), click on the following query and hit the play button beside the top bar to initialize a new frame with the tasks for our user study.

```
:play study
```

```
neo4j$
```

```
$ :play study
```

Stage 1

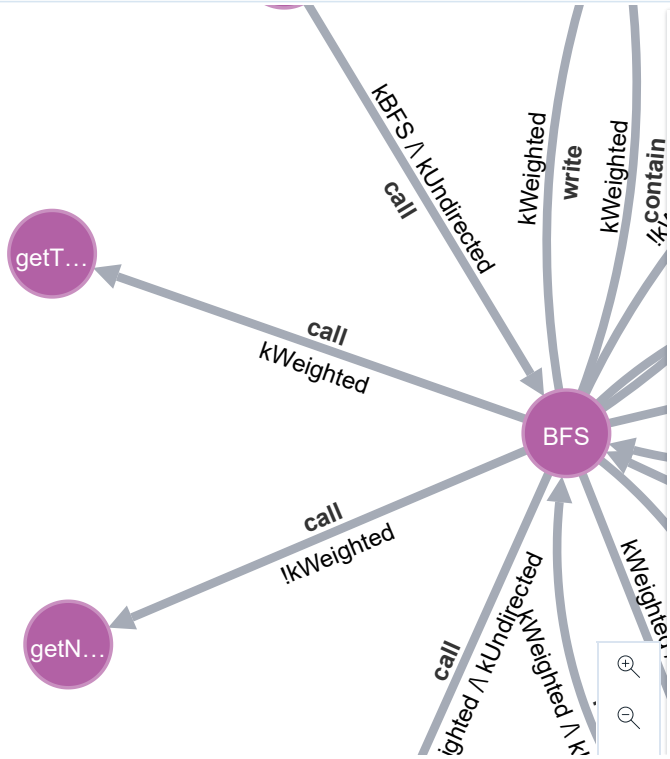
At this stage, you will interact with the interface to perform six tasks.

We will provide you with two queries. The first query should be used for the first three tasks, whereas the second query should be used for the rest of the tasks. Both queries will remain accessible the throughout the entire time. Throughout the study, feel free to customize the visualization to help you complete the tasks.

You can move to the next page whenever you are ready to start.

```
neo4j$
```

```
neo4j$ MATCH (n)-[r]-(m) WHERE r.condition <> "true" and m.label = "BFS" and n.l…
```



Overview

Node labels

cFunction (5) cVariable (5)

Relationship Types

* (15) read (3) contain (4) call (4) write (4)

Displaying 10 nodes, 15 relationships.

```
$ :play study
```

Task 1.1

For this task, you will have to run the following query:

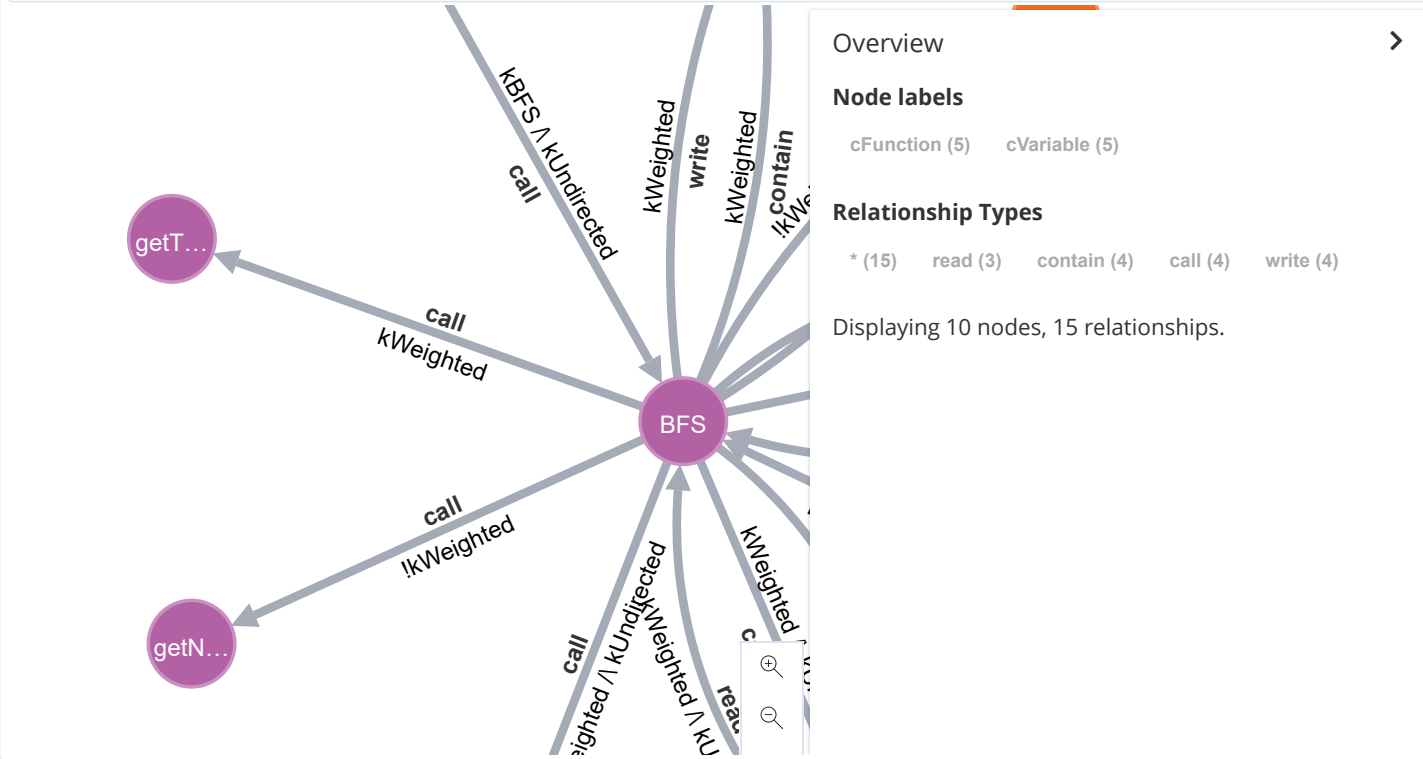
```
MATCH (n)-[r]-(m) WHERE r.condition <> "true" and m.label = "BFS" and n.label <> "kBFS"
RETURN *
```

Which **function(s)** from the returned graph may be **called** by the function *BFS* in a program variant with the configuration **kUndirected /\ !kWeighted /\ !kDFS /\ kBFS**?

Submit

```
neo4j$
```

```
neo4j$ MATCH (n)-[r]-(m) WHERE r.condition <> "true" and m.label = "BFS" and n.l…
```




```
$ :play study
```

Task 1.2

For this task, you will have to run the following query:

```
MATCH (n)-[r]-(m) WHERE r.condition <> "true" and m.label = "BFS" and n.label <> "kBFS"
RETURN *
```

Which **variable(s)** from the returned graph may be **written** by the function *BFS* in the same program variant **kUndirected /\ !kWeighted /\ !kDFS /\ kBFS**?

Submit

```
neo4j$
```

```
neo4j$ MATCH (n)-[r]-(m) WHERE r.condition <> "true" and m.label = "BFS" and n.l…
```

Overview

Node labels

cFunction (5) cVariable (5)

Relationship Types

* (15) read (3) contain (4) call (4) write (4)

Displaying 10 nodes, 15 relationships.

```
$ :play study
```

Task 1.3

For this task, you will have to run the following query:

```
MATCH (n)-[r]-(m) WHERE r.condition <> "true" and m.label = "BFS" and n.label <> "kBFS"
RETURN *
```

Considering the same program variant (kUndirected /\ !kWeighted /\ !kDFS /\ kBFS), which **variable(s)** from the returned graph may be **written** by the function *BFS* if we **enable** the feature **kWeighted** in the original configuration?

Submit

```
neo4j$
```

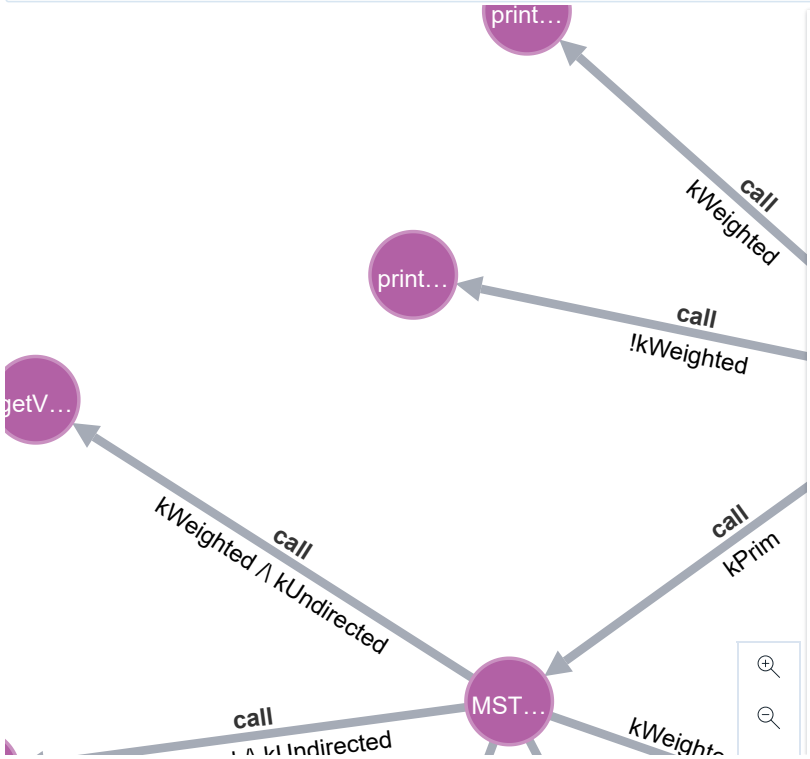
```
$ :play study
```

Task 2

Before starting the next task, fell free close the visualization frame with the results for the first task.

```
neo4j$
```

```
neo4j$ MATCH p=(f:cFunction)-[r:call]→(t:cFunction) WHERE f<>t AND r.condition …
```



Overview

Node labels

cFunction (14)

Relationship Types

* (14) call (14)

Displaying 14 nodes, 28 relationships.

```
$ :play study
```

Task 2.1

For this task, you will have to run the following query:

```
MATCH p=(f:cFunction)-[r:call]→(t:cFunction) WHERE f<>t AND r.condition <> "true" AND
t.label <> "getID" AND t.label <> "addEdge" AND t.label <> "addNgbr" AND t.label <>
"getID" AND t.label <> "clearVisited" AND t.label <> "getSrcID" AND t.label <>
"getTargetID" AND t.label <> "getNgbrs" RETURN *
```

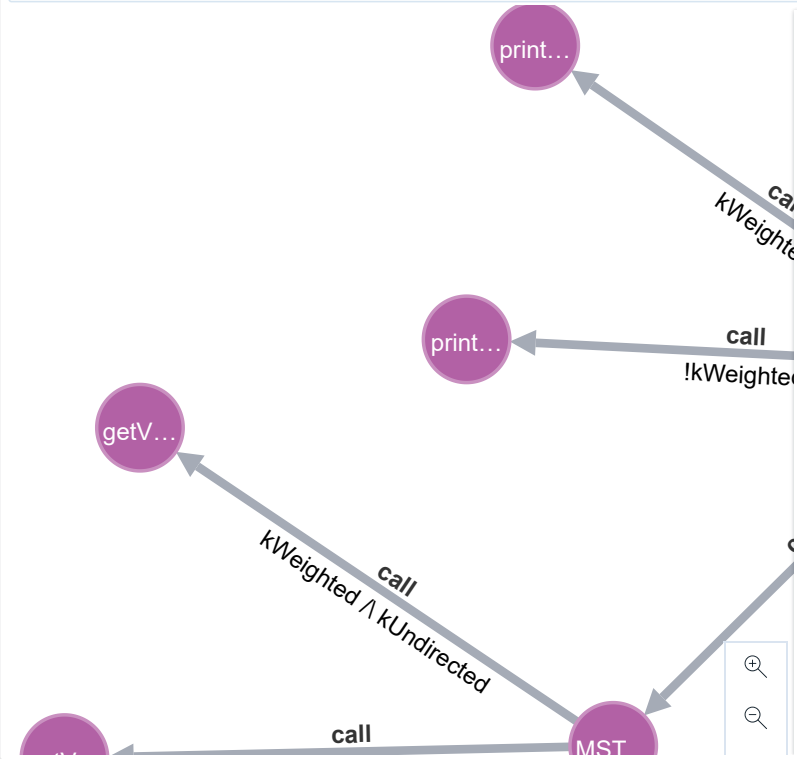
Which **function(s)** may be directly called by the function *execComnd* in the following program variant V1:

- V1: kWeighted /\ kUndirected /\ kDFS /\ !kBFS /\ kCycle /\ kConnectedComps /\ !kPrim

Submit

```
neo4j$
```

```
neo4j$ MATCH p=(f:cFunction)-[r:call]→(t:cFunction) WHERE f<>t AND r.condition…
```



Overview

Node labels

cFunction (14)

Relationship Types

* (14) call (14)

Displaying 14 nodes, 28 relationships.

```
$ :play study
```

Task 2.2

For this task, you will have to run the following query:

```
MATCH p=(f:cFunction)-[r:call]→(t:cFunction) WHERE f<>t AND r.condition <> "true" AND
t.label <> "getID" AND t.label <> "addEdge" AND t.label <> "addNgbr" AND t.label <>
"getID" AND t.label <> "clearVisited" AND t.label <> "getSrcID" AND t.label <>
"getTargetID" AND t.label <> "getNgbrs" RETURN *
```

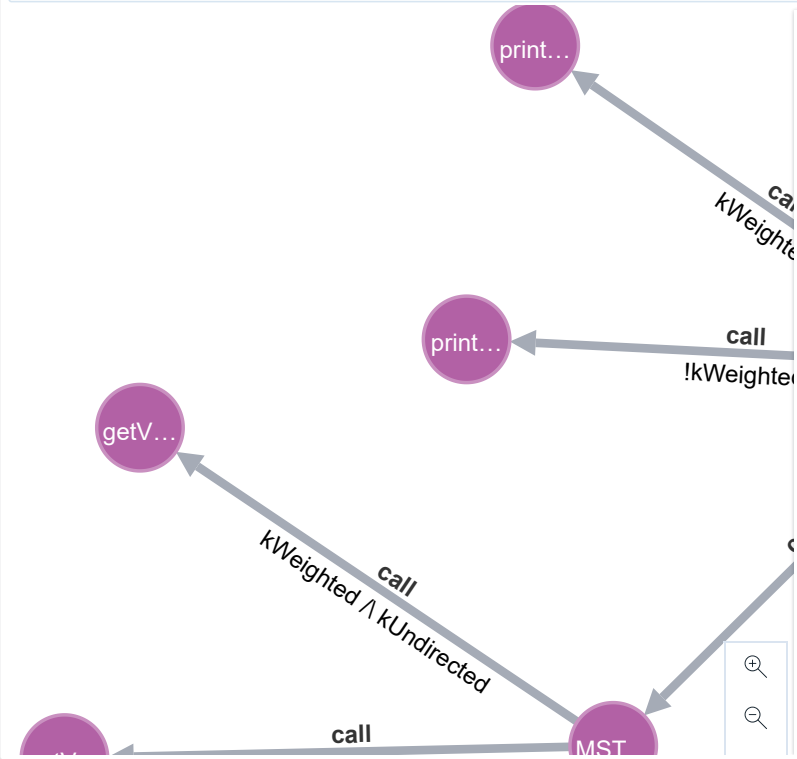
Which **function(s)** may be directly called by the function *execComnd* in the program variant V2 but **not** in V1:

- V1: kWeighted /\ kUndirected /\ kDFS /\ !kBFS /\ kCycle /\ kConnectedComps /\ !kPrim
- V2: kWeighted /\ kUndirected /\ !kDFS /\ kBFS /\ !kCycle /\ kConnectedComps /\ kPrim

Submit

```
neo4j$
```

```
neo4j$ MATCH p=(f:cFunction)-[r:call]→(t:cFunction) WHERE f<>t AND r.condition…
```



Overview

Node labels

cFunction (14)

Relationship Types

* (14) call (14)

Displaying 14 nodes, 28 relationships.

```
$ :play study
```

Task 2.3

For this task, you will have to run the following query:

```
MATCH p=(f:cFunction)-[r:call]→(t:cFunction) WHERE f<>t AND r.condition <> "true" AND
t.label <> "getID" AND t.label <> "addEdge" AND t.label <> "addNgbr" AND t.label <>
"getID" AND t.label <> "clearVisited" AND t.label <> "getSrcID" AND t.label <>
"getTargetID" AND t.label <> "getNgbrs" RETURN *
```

There are **two** possible call paths between the functions *execComnd* and *DFS*. Which program variant may execute both call paths?

- V1: kWeighted /\ kUndirected /\ kDFS /\ !kBFS /\ kCycle /\ kConnectedComps /\ !kPrim
- V2: kWeighted /\ kUndirected /\ !kDFS /\ kBFS /\ !kCycle /\ kConnectedComps /\ kPrim

Submit

```
neo4j$
```

```
$ :play study
```

End of Stage 1

You have finished the first stage of the study.

Let the researcher know that you have finished the tasks for this stage. They should provide you with the link for the next steps of the study.