**Testing, Debugging, and Repairing Software"**

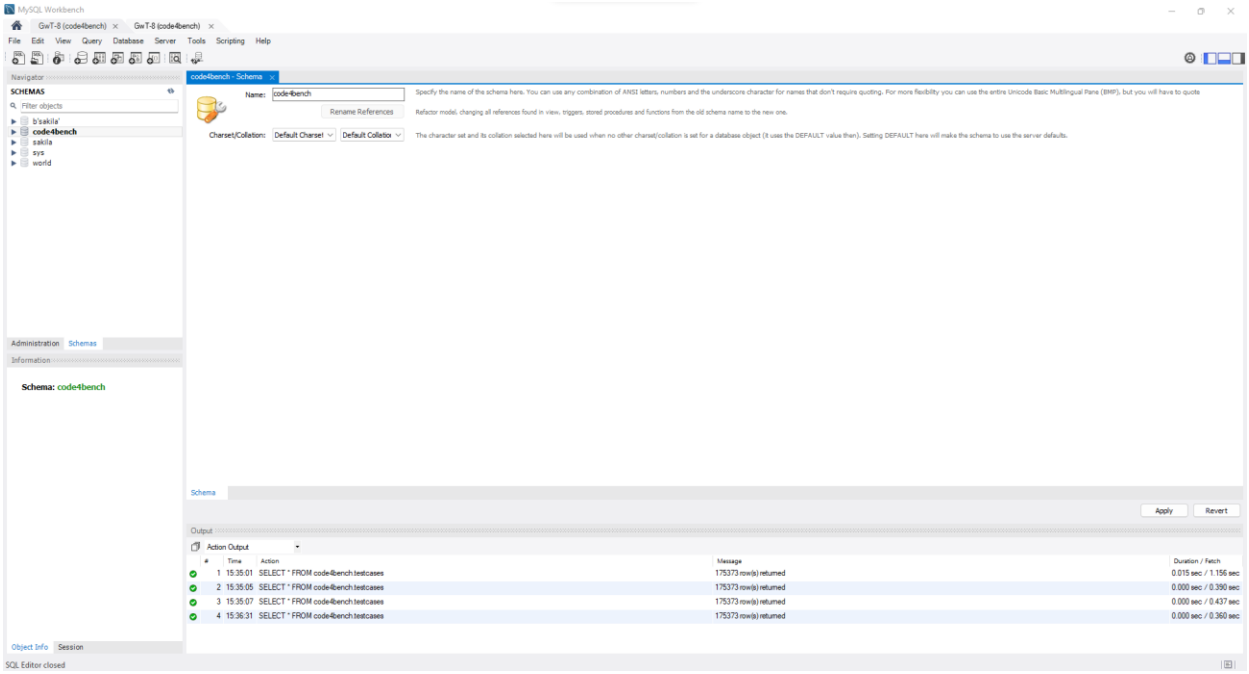
**Project**

**Mohammad Hadi Kamali**

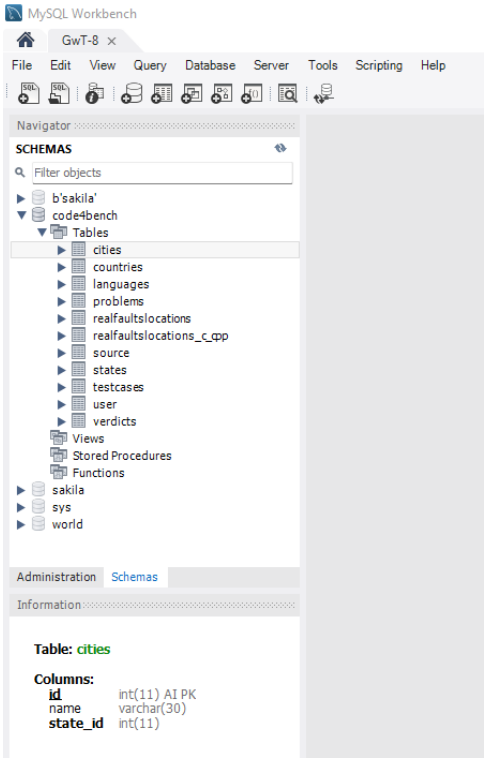
July 2023

"To begin, we need to install the Gephi software and MySQL Workbench, and download the code4bench file".

And we need to create a Schema named 'code4bench'

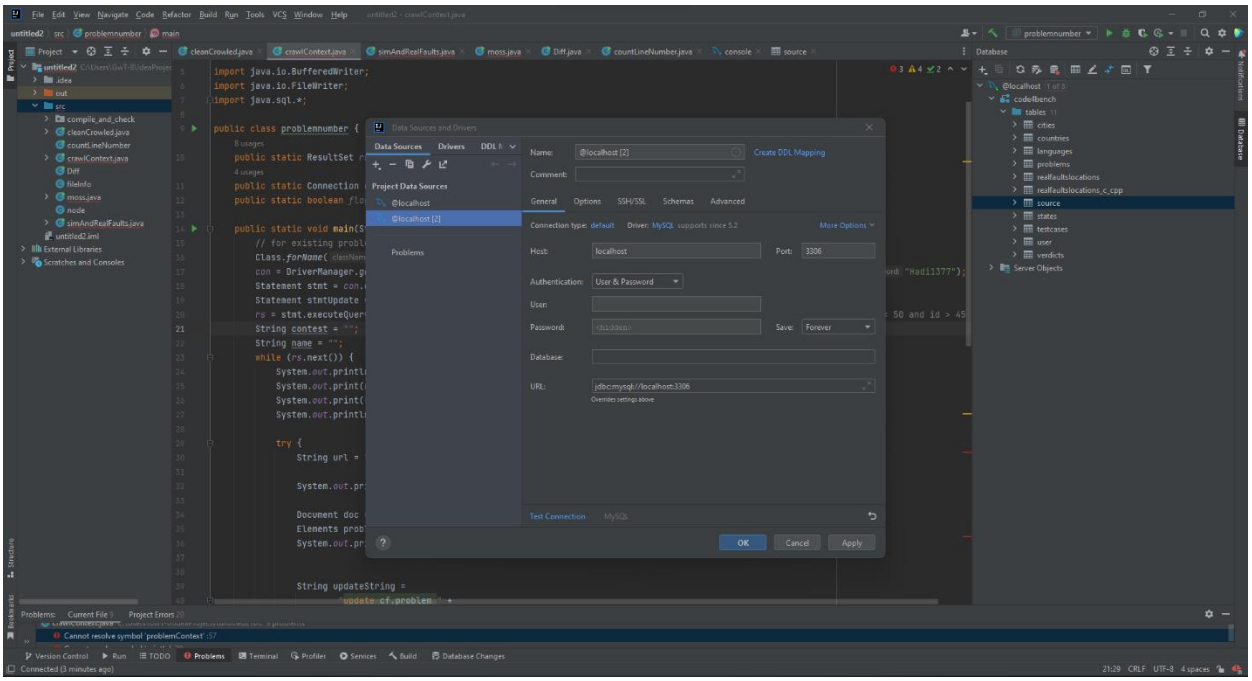


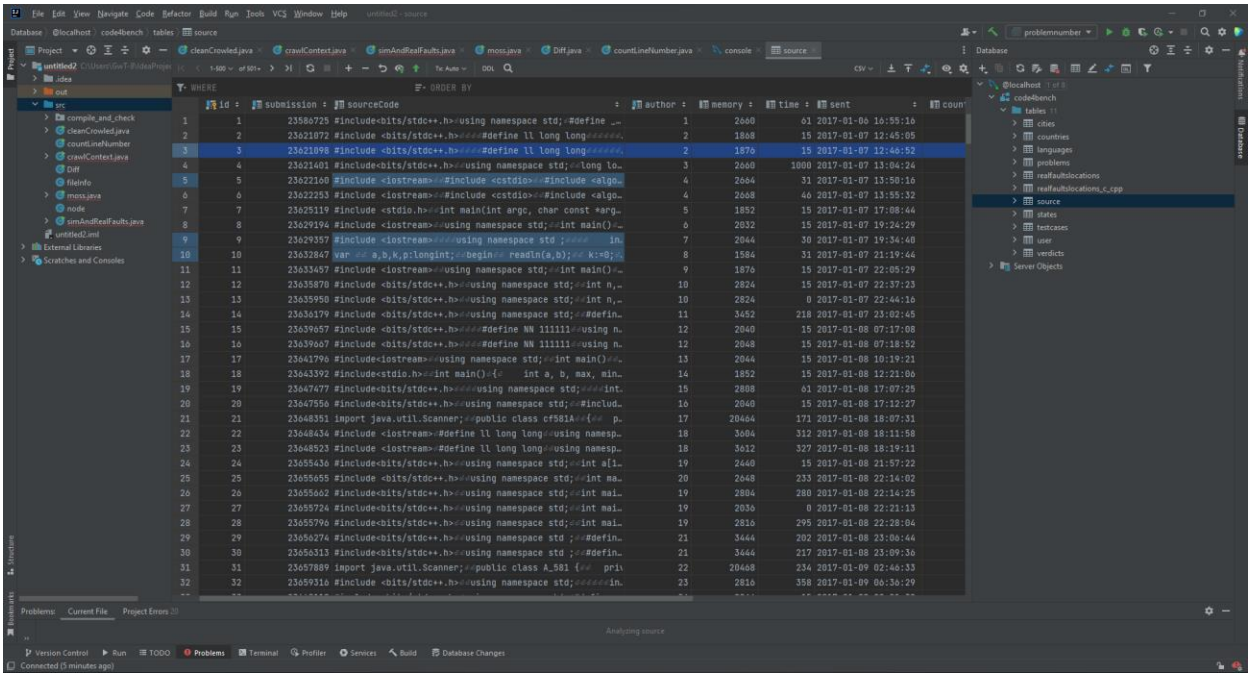
"Then, from the server data section, we import the extracted downloaded file into the Data Import section (this process is time-consuming, taking about 30 to 40 minutes).".




"On the left side of the image, it can be seen that our data has been imported, and the name of the data source is listed in the schemas section".

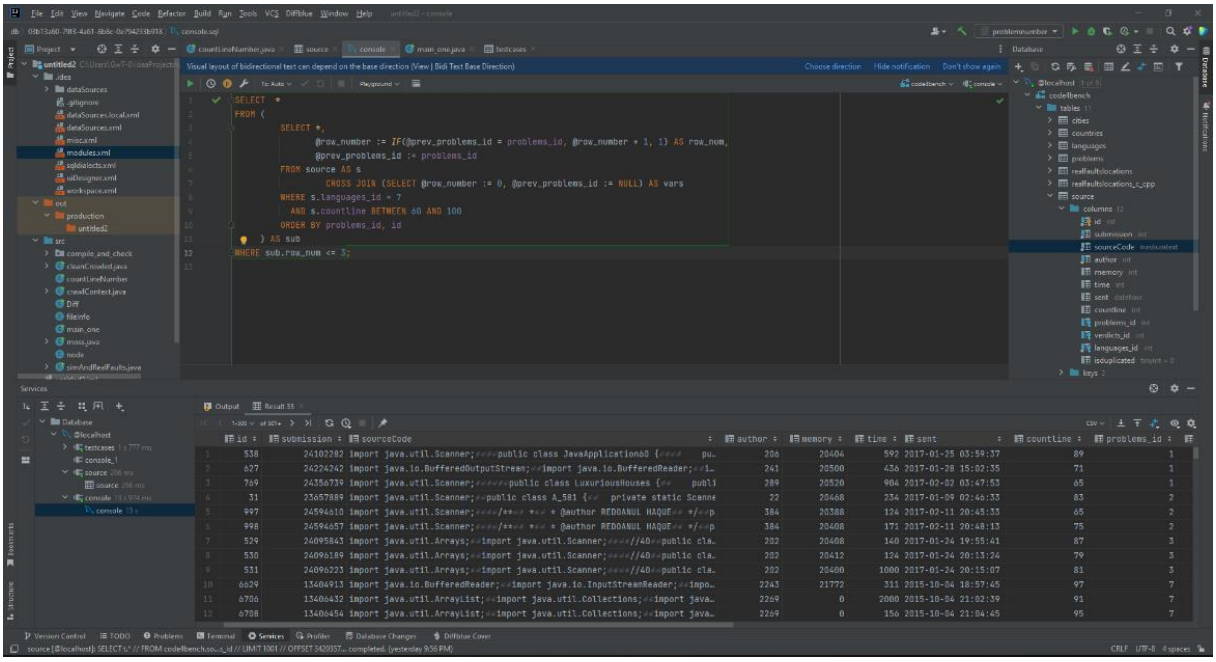
Now, using the IntelliJ IDEA software, we add the database located in MYSQL to IntelliJ. .





"And now, we need to select 5 of these SourceCodes".

"I chose Java for extracting 5 programs, where Java has a `language_id` of 7, and I wrote a query for it".



"Using the query above, for `language_id=7`, I find a total of 3 specific `problems_id` with a `countline` between 60 to 100 (because our PDG generation tool cannot output higher line counts). Here, we will provide the details of each of the source codes:"

: Source code one

- problems_id = 1
- submission = 24356739
- languages_id = 7

: Source code two

- problems_id = 8
- submission = 13769979
- languages_id = 7

: Source code three

- problems_id = 17
- submission = 24916746
- languages_id = 7

: Source code four

- problems_id = 13
- submission = 17382618
- languages_id = 7

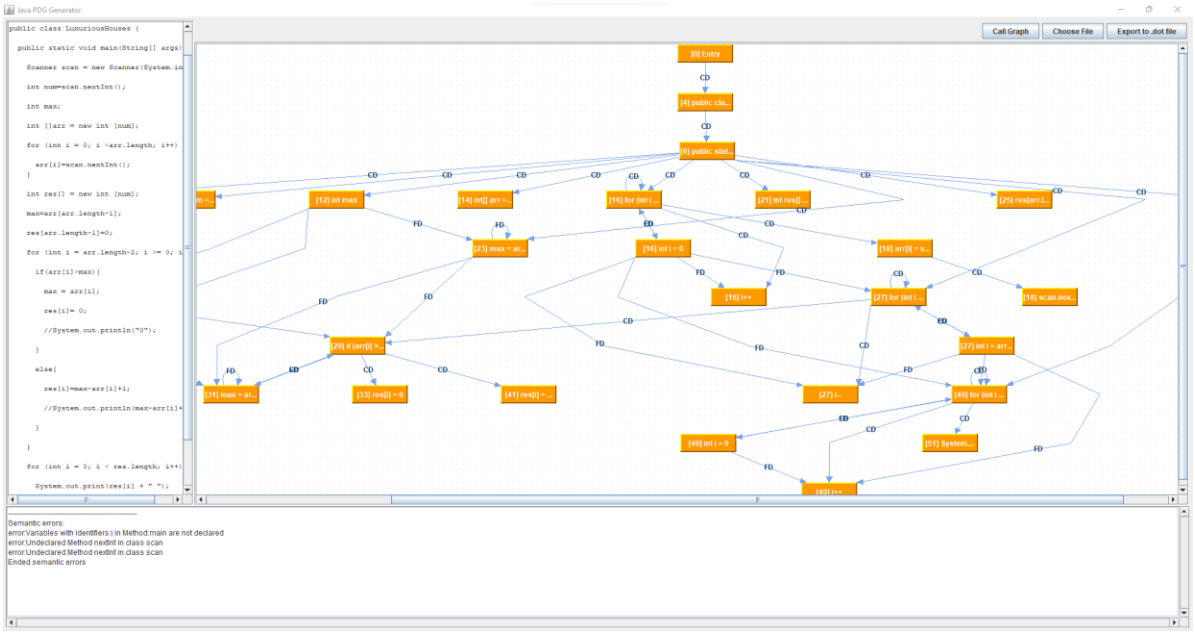
: Source code five

- problems_id = 51
- submission = 21829078
- languages_id = 7

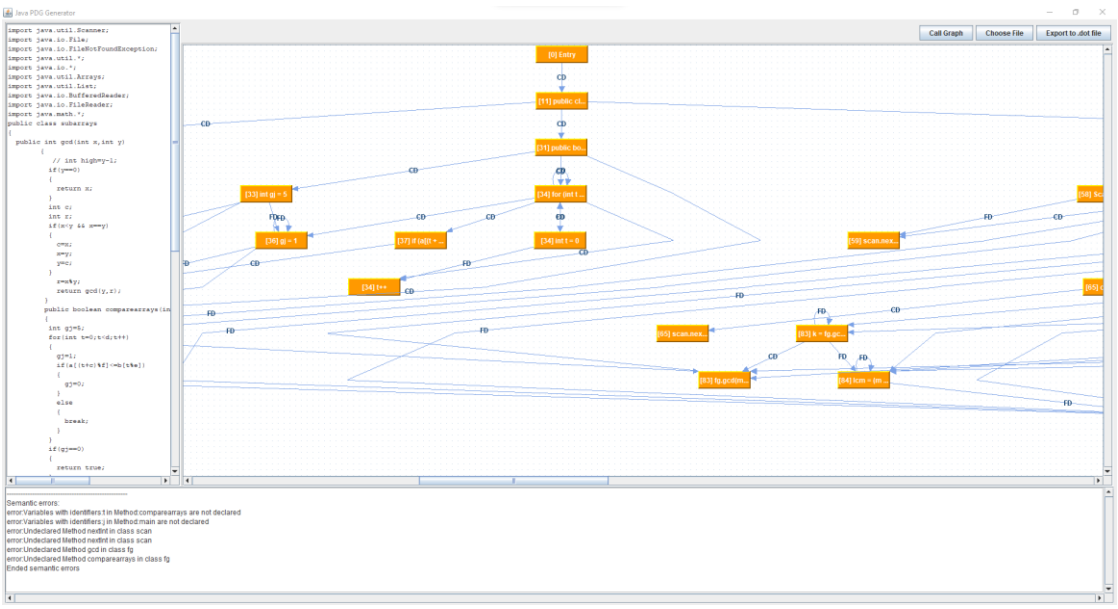
"Now, we add each program as a project in IntelliJ IDEA and save it in the Java format".

Using the Java PDG Generator tool, we generate the PDG output for each of these codes to obtain the data " dependencies and control dependencies."

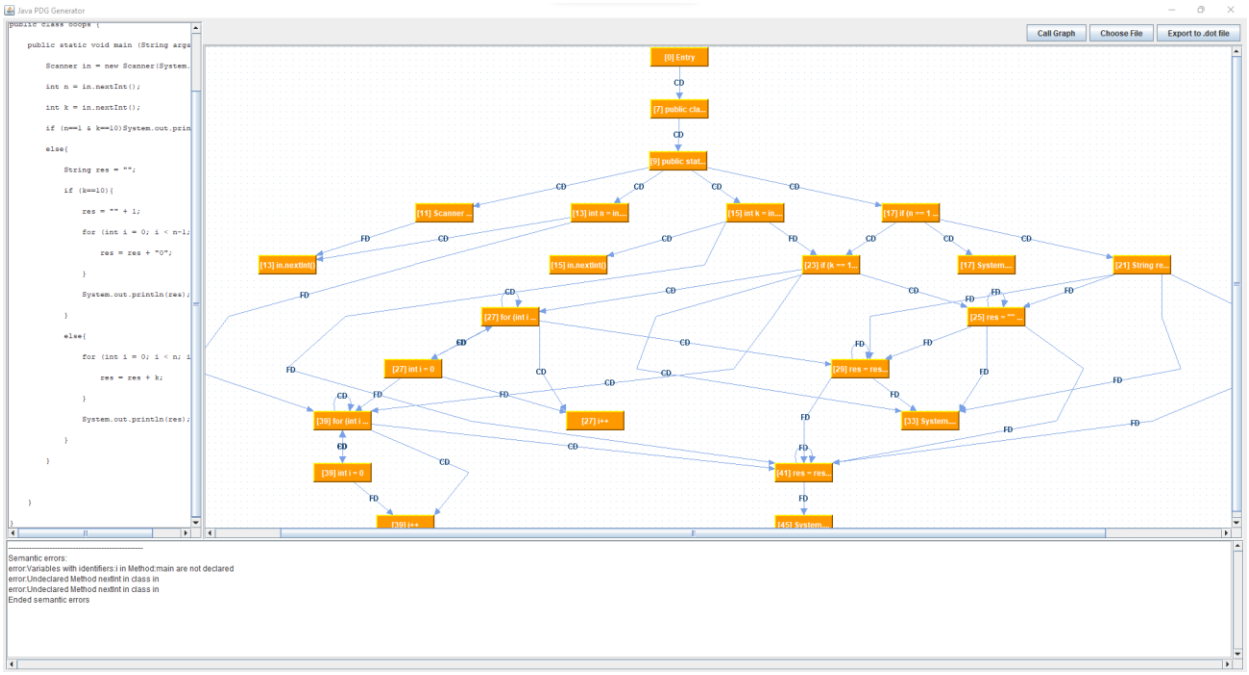
## PDG for Source code one :



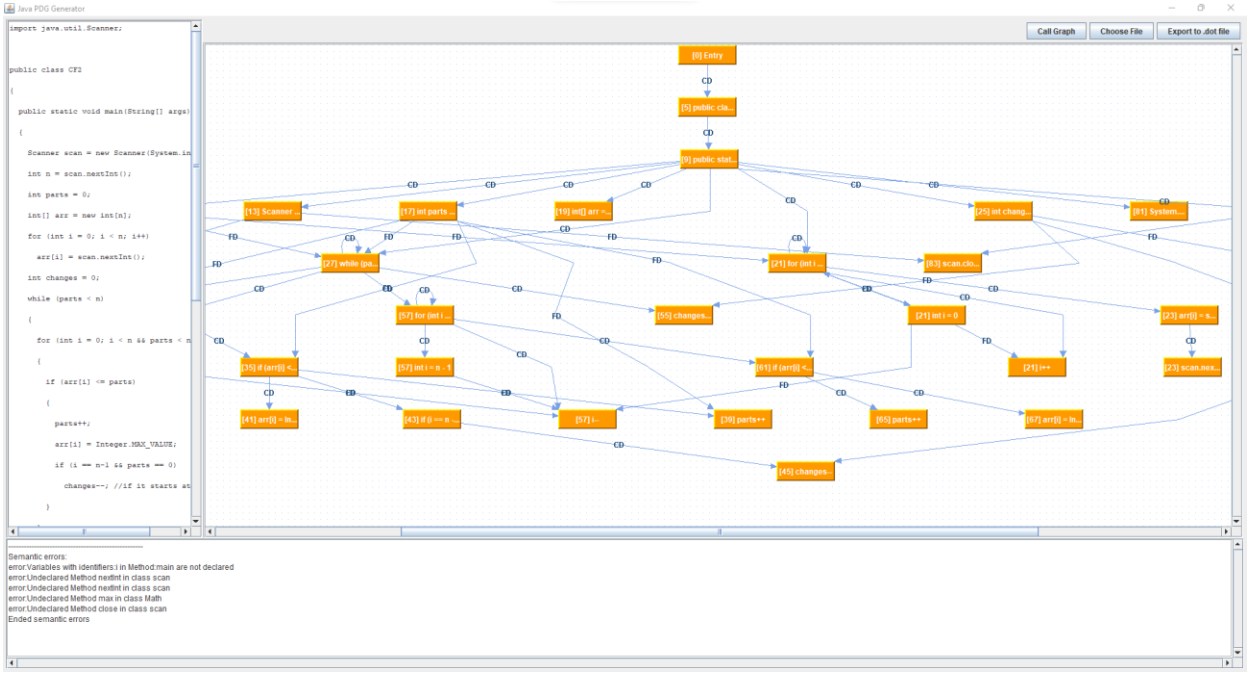

## PDG for Source code two :



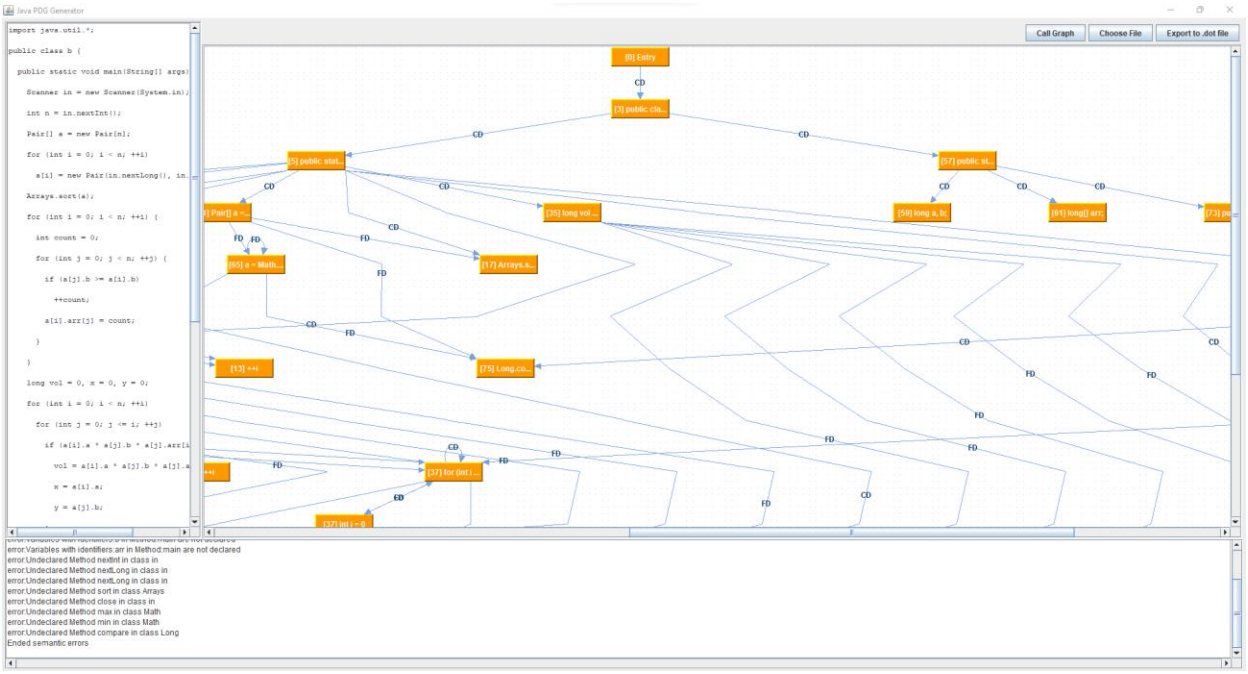
PDG for Source code three:




PDG for Source code four:

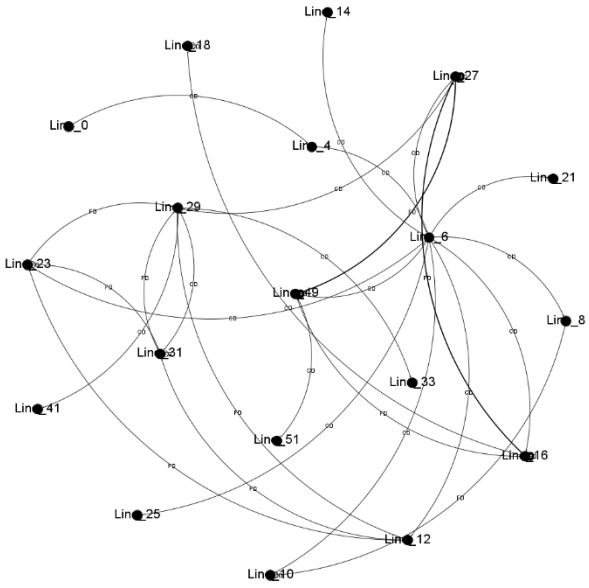


## PDG for Source code five:

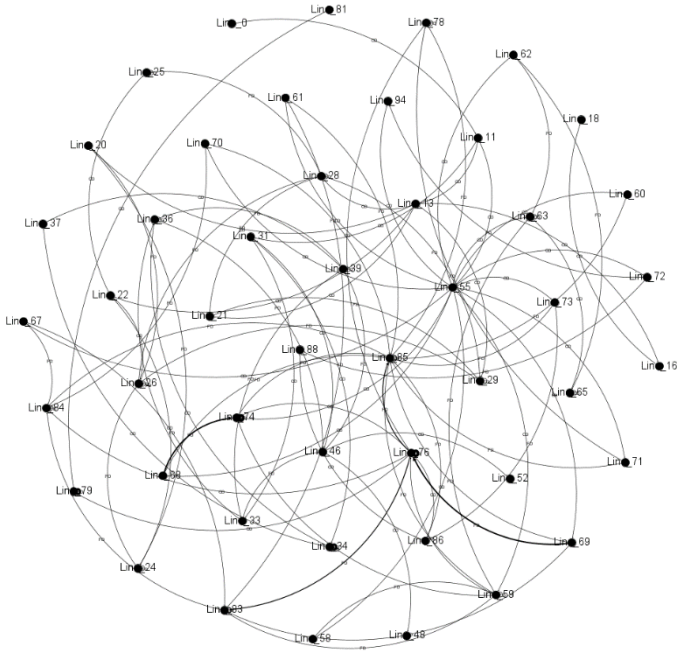



"In this tool, control dependencies are indicated by the symbol 'CD' on arrows, and data dependencies are displayed with the symbol 'FD'. Now, we can export from this tool in the .dot format, which gives us the dependencies between programs along with labels. Here, I was unable to export from this tool on Windows 11 version 21H2 and OS Build 22000.2057, as it didn't prompt the save message. Instead, I used a Linux Ubuntu virtual machine (VMware Workstation) to obtain the output for each of these dependency graphs. We now display each of these outputs in Gephi, as shown below."

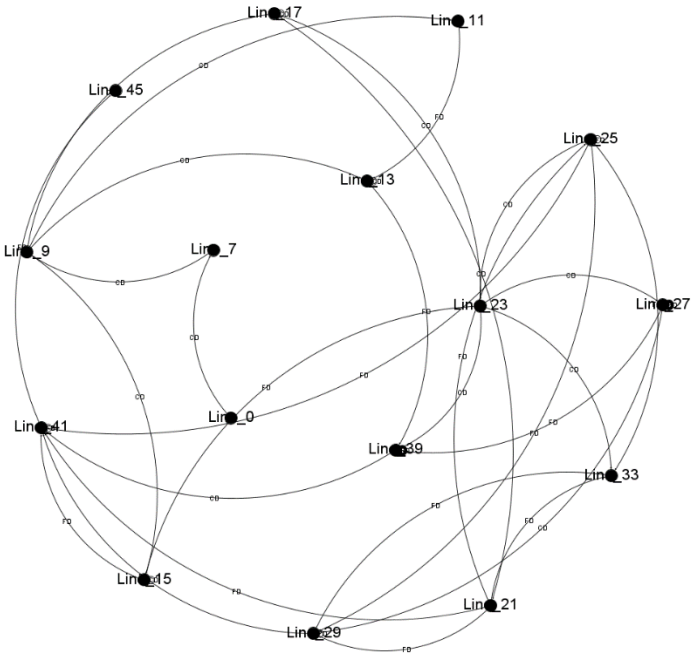
: Gephi Source one



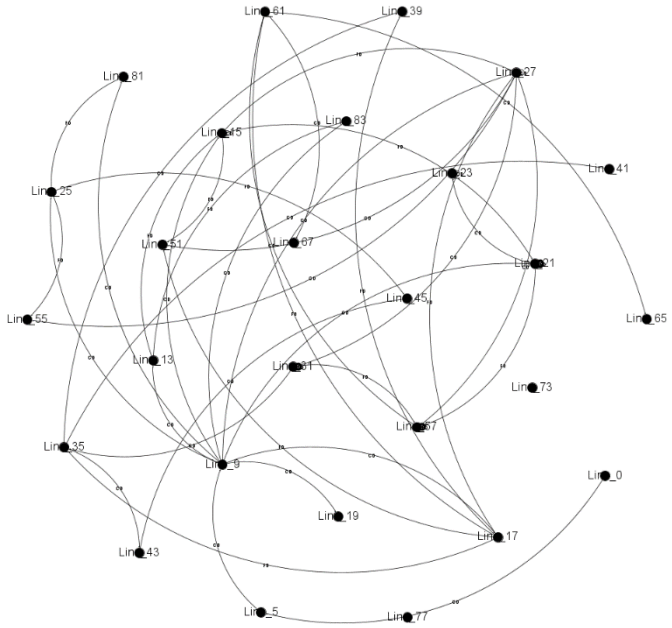
: Gephi Source two



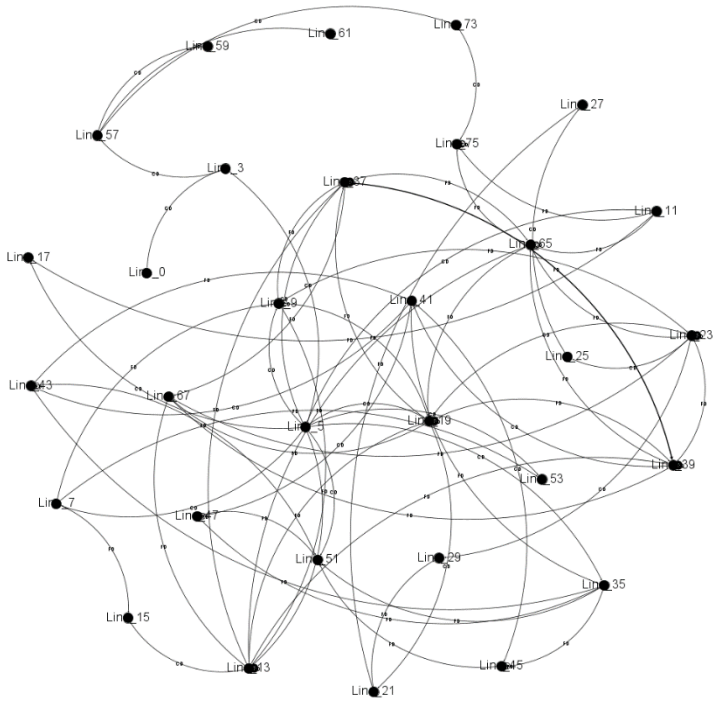
: Gephi Source three



: Gephi Source four



: Gephi Source five

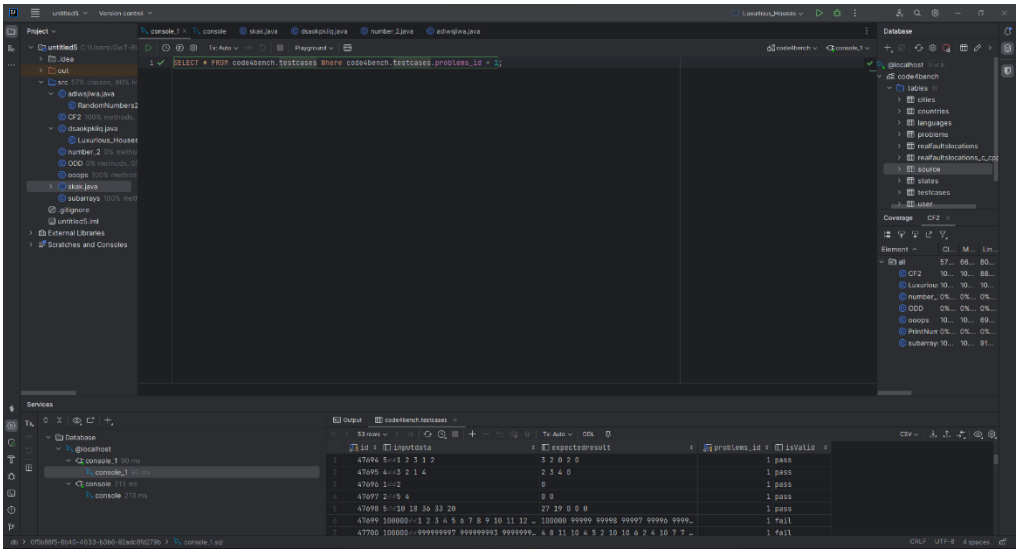


Now, we need to find the related test cases corresponding to those problem_ids. Here, our problem_ids are as " ":follows

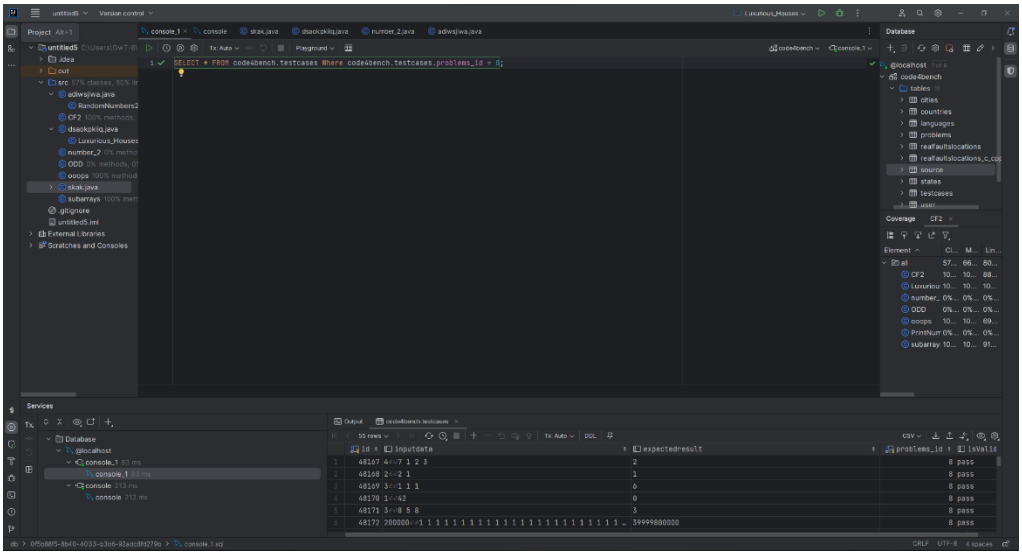
- problems_id = 1
- problems_id = ٨
- problems_id = 1٧
- problems_id = 1٣
- problems_id = ٥١

"Now, by using a query in the testcases section, we extract test cases for each of these sourceCodes based on the specified problem_id. (We updated the IntelliJ IDEA version from 2021.3 to 2023.1 and switched from classic mode to the new UI mode)".

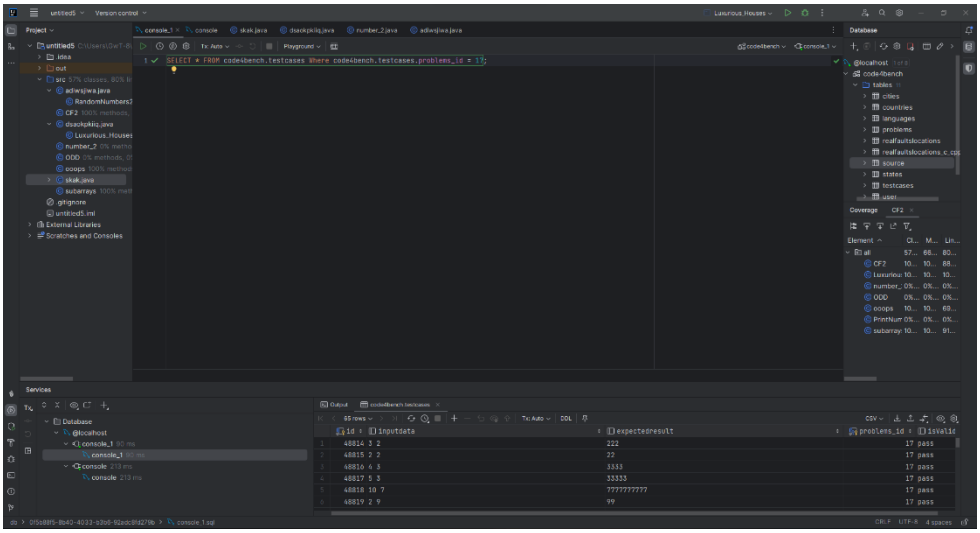
Problem_id =1



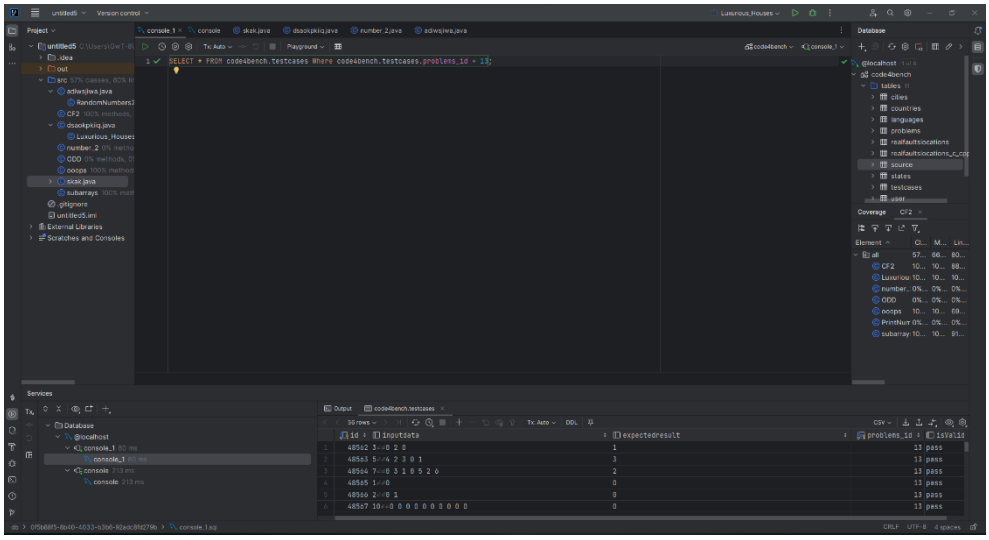
Problem_id =8



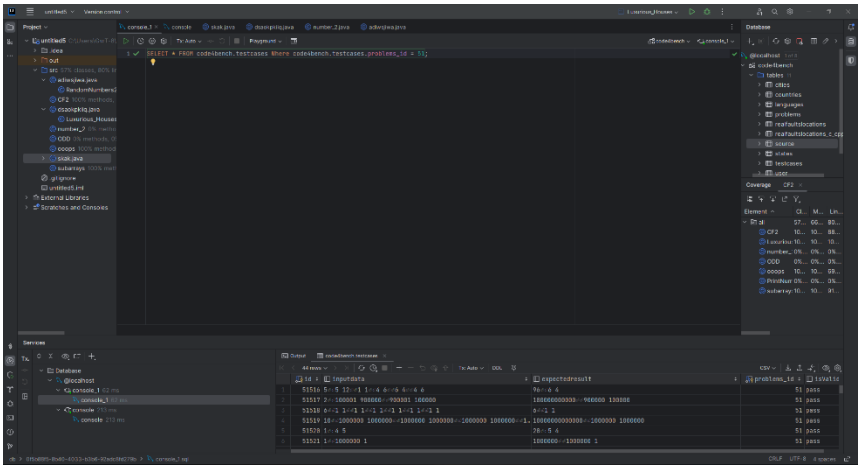
Problem_id =17



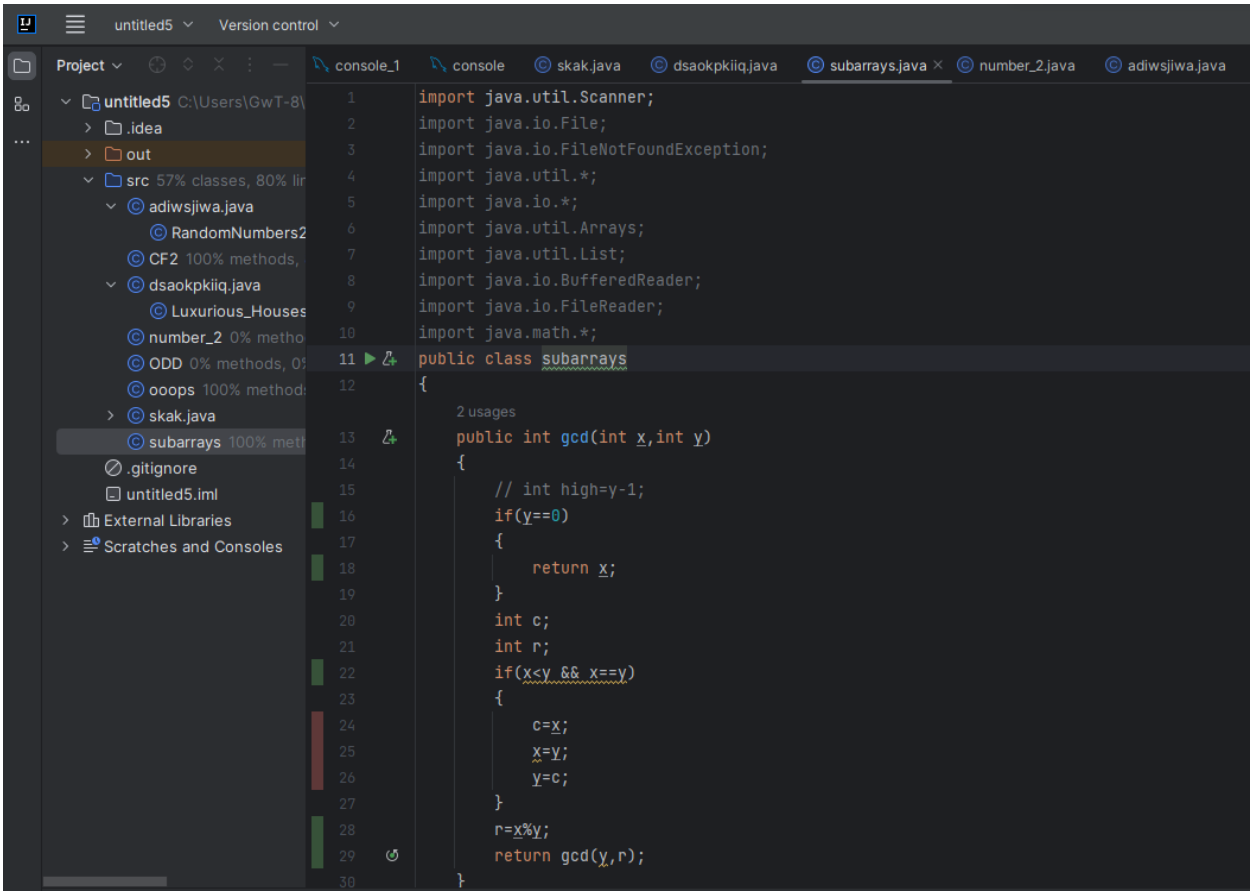
Problem_id =13



Problem_id =51



"Now, we extract 20 test cases for each given program, input their values into the program, and capture the outputs. Using IntelliJ IDEA's debugging feature, we can see which lines have been reached".

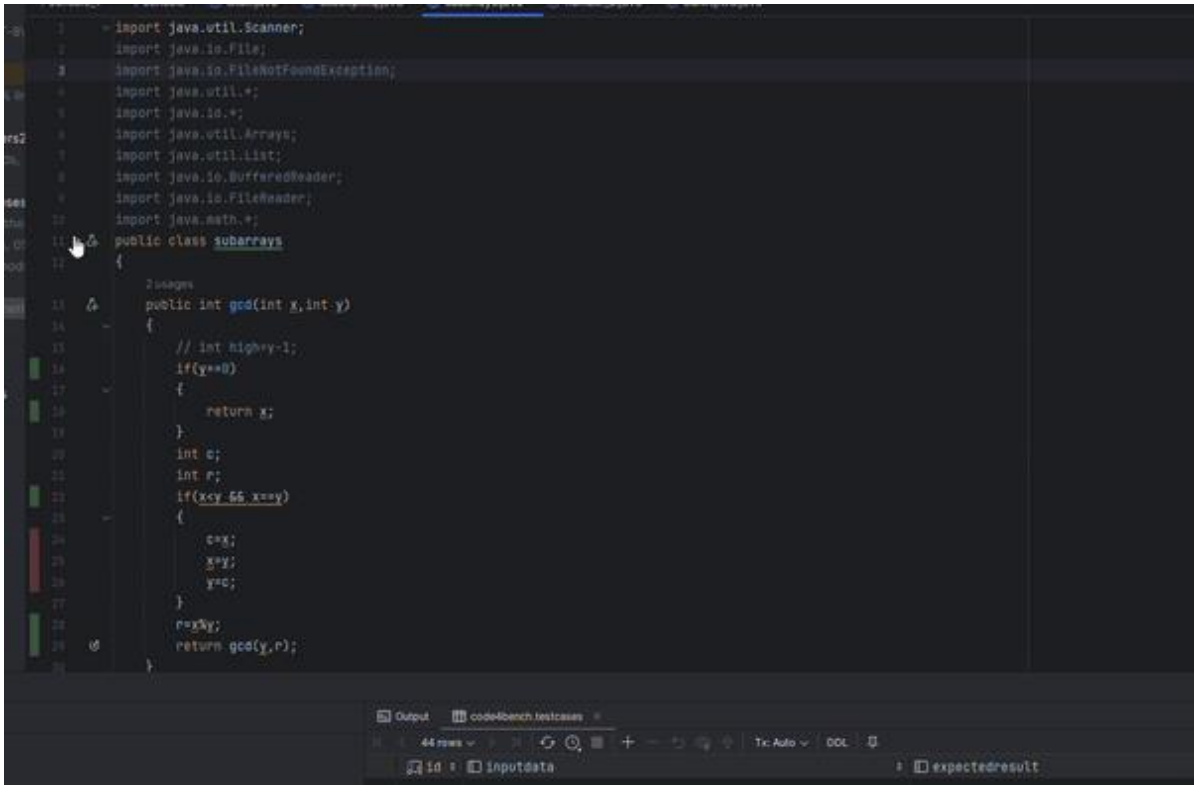



"The lines that have been reached are marked in green, and the unreachable lines are marked in red".

"For each test case, we determine which lines of the program are covered, and we save their isValid values and inputData in a file associated with each program".

"Which is written in the file named 'testcase' in files 1 to 5, below is an example of this".

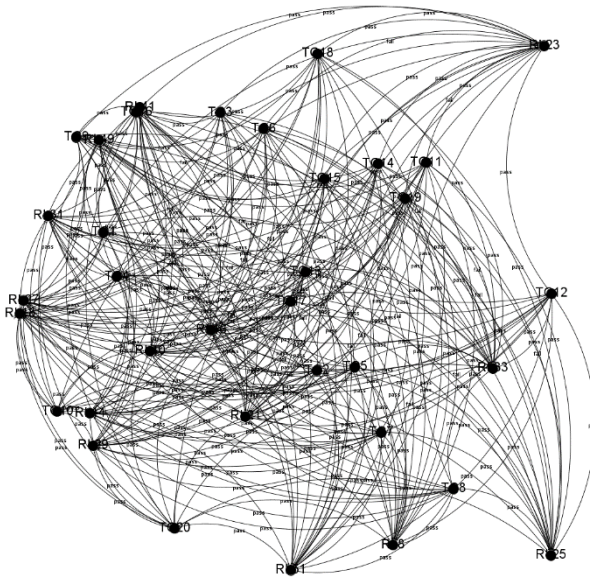
"In the GIF below, we examine how to activate debug for line coverage".



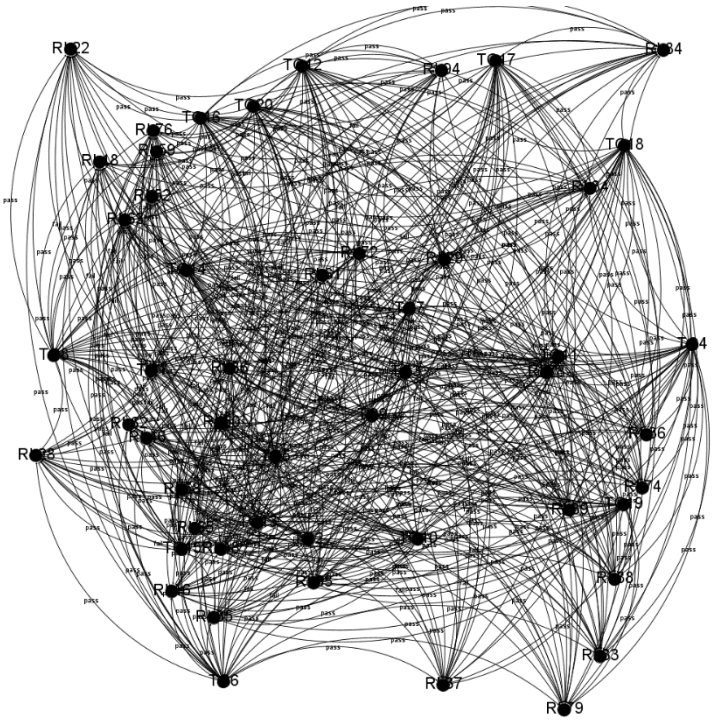

"In the next step, using the covered lines for each test case and the isVaild indicator, which shows whether each test case passes or fails, we create a file in the .dot format."

" For each testcase, it is as follows":

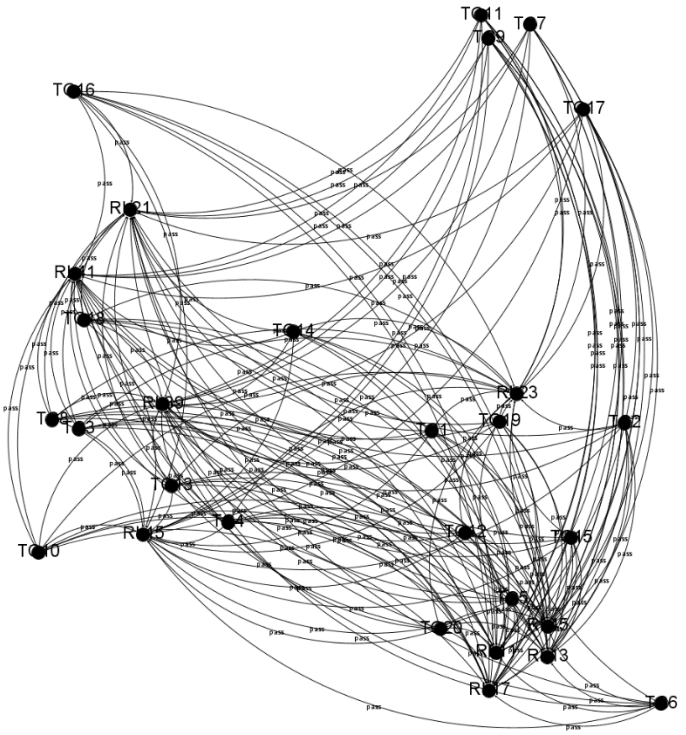
: Gephi Source one



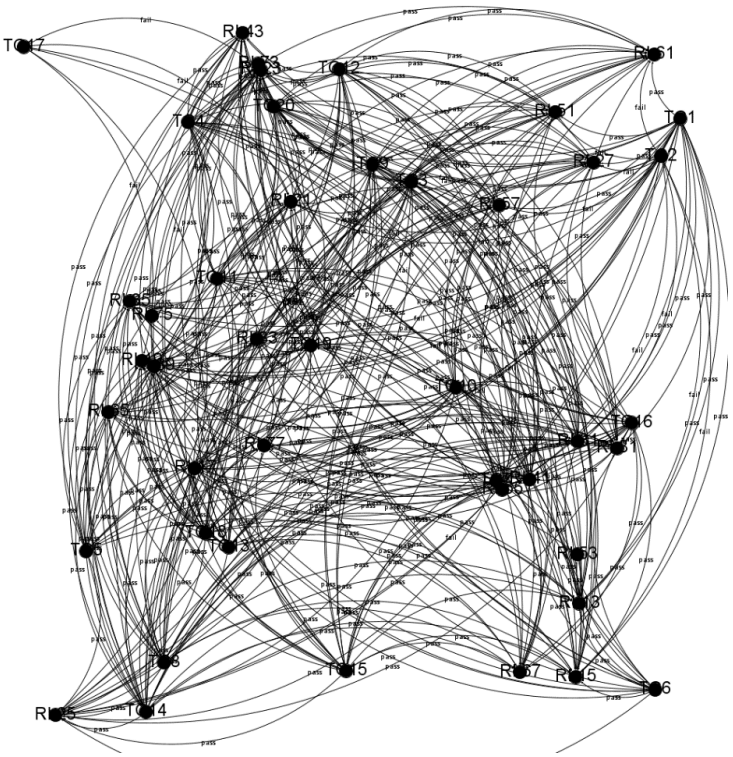
: Gephi Source two



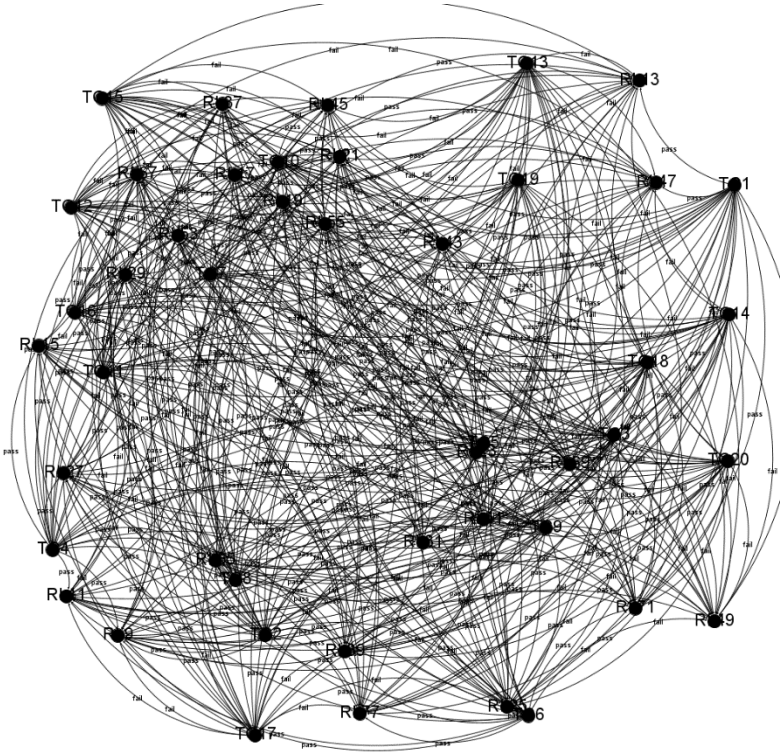
: Gephi Source three



: Gephi Source four

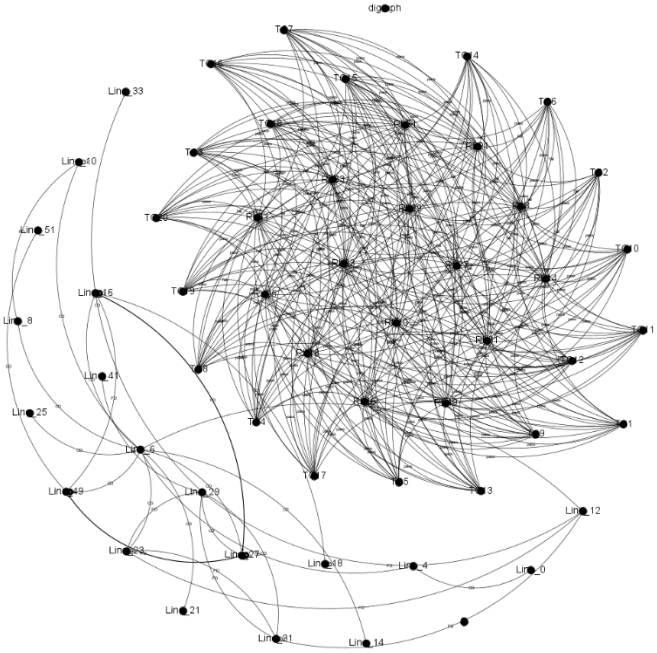


: Gephi Source five

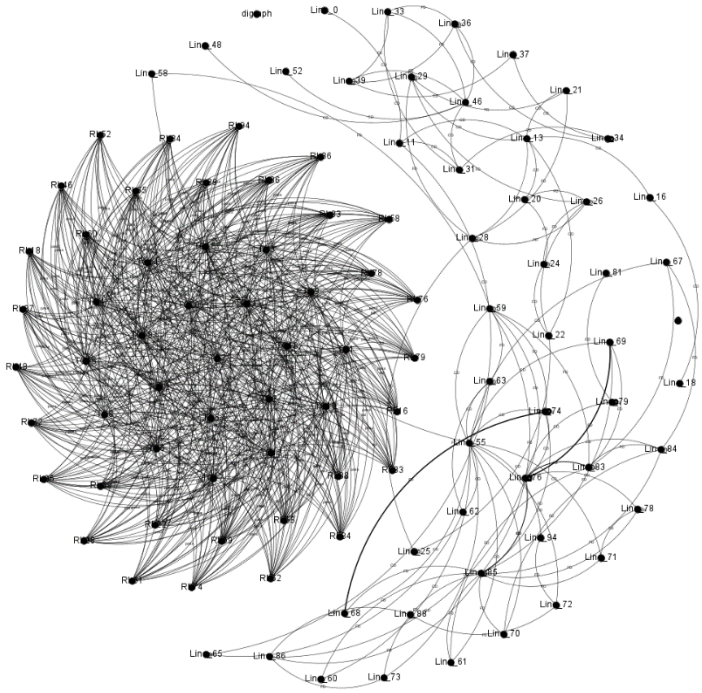


"Now, we combine the given files."

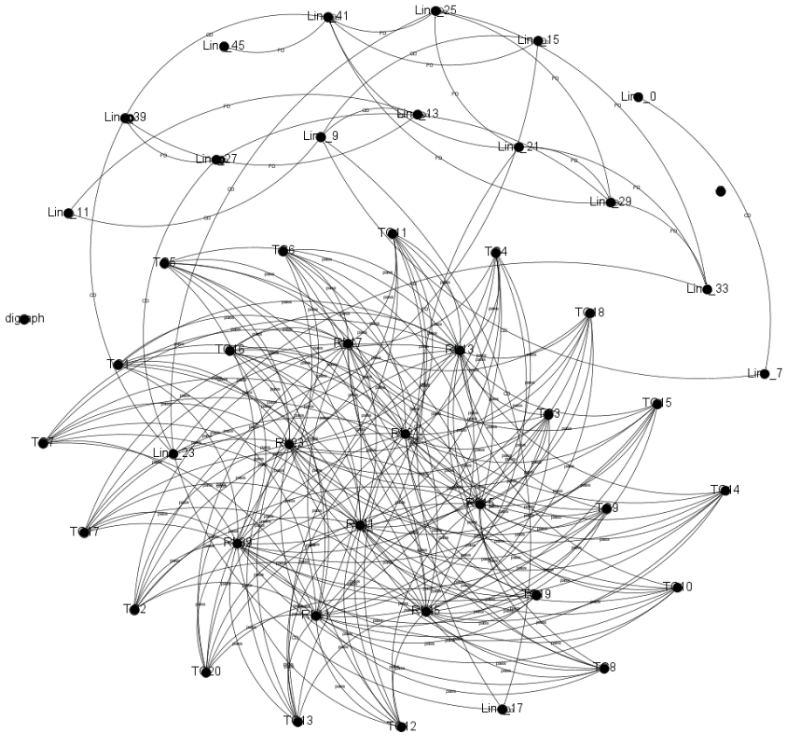
: Gephi +TestCase Source one



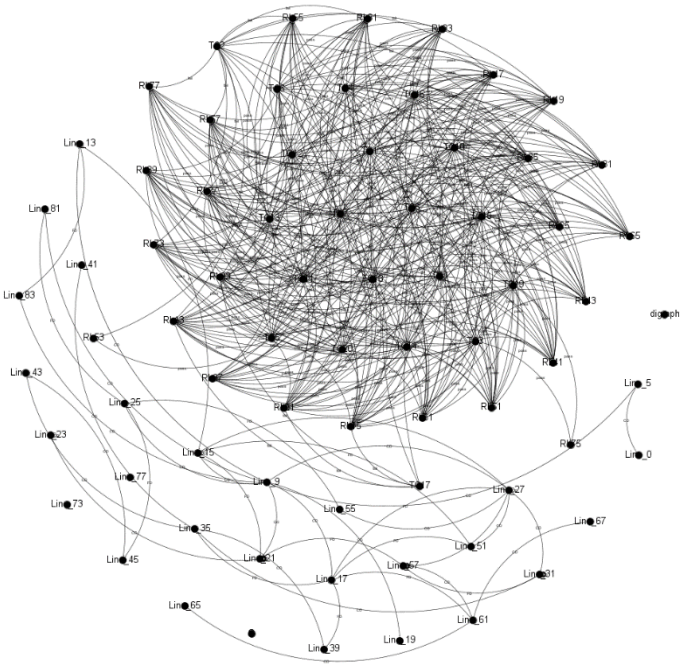

: Gephi +TestCase Source two



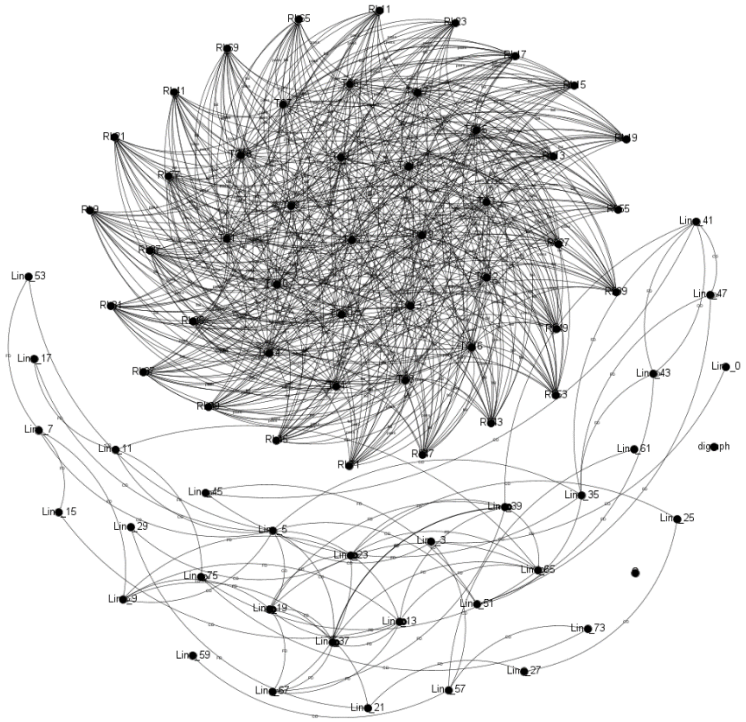
: Gephi +TestCase Source three




: Gephi +TestCase Source four



: Gephi +TestCase Source five



"Now, using the statistics tab, we can perform statistical analyses."