



"Testing, Debugging, and Repairing Software"

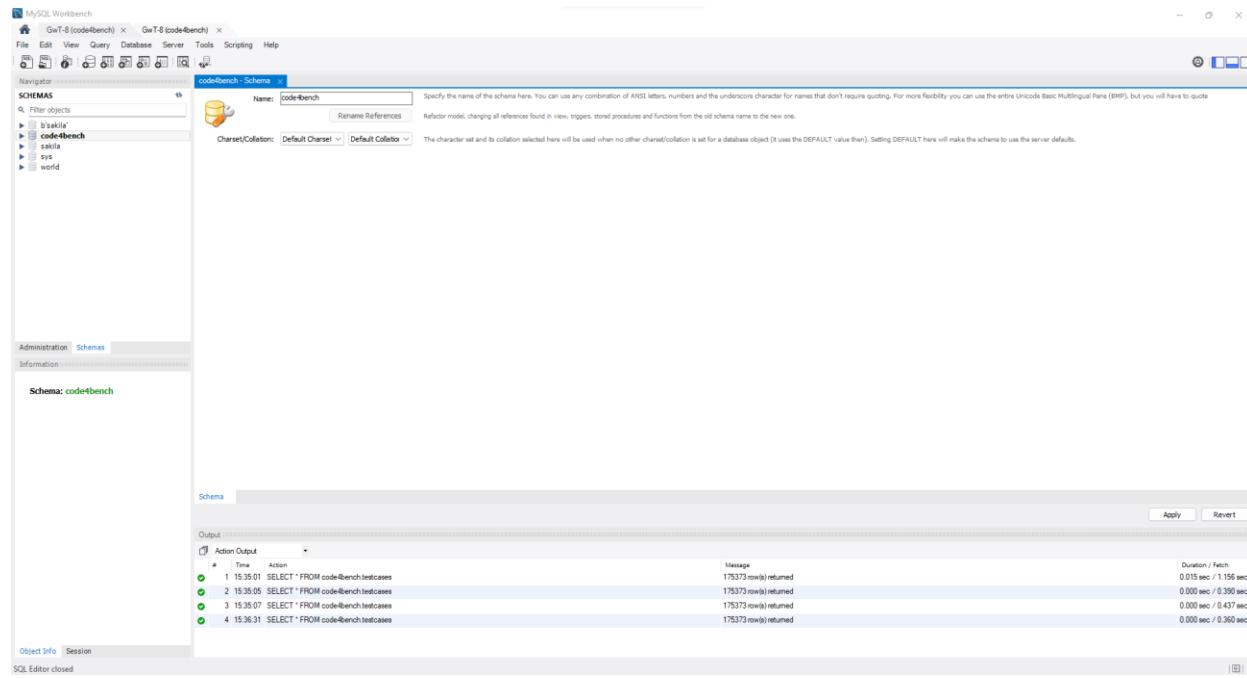
Project

Mohammad Hadi Kamali

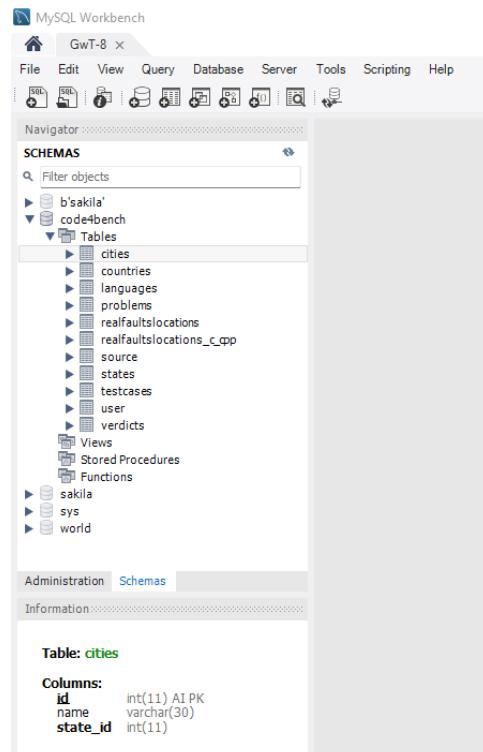
July 2023

"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'

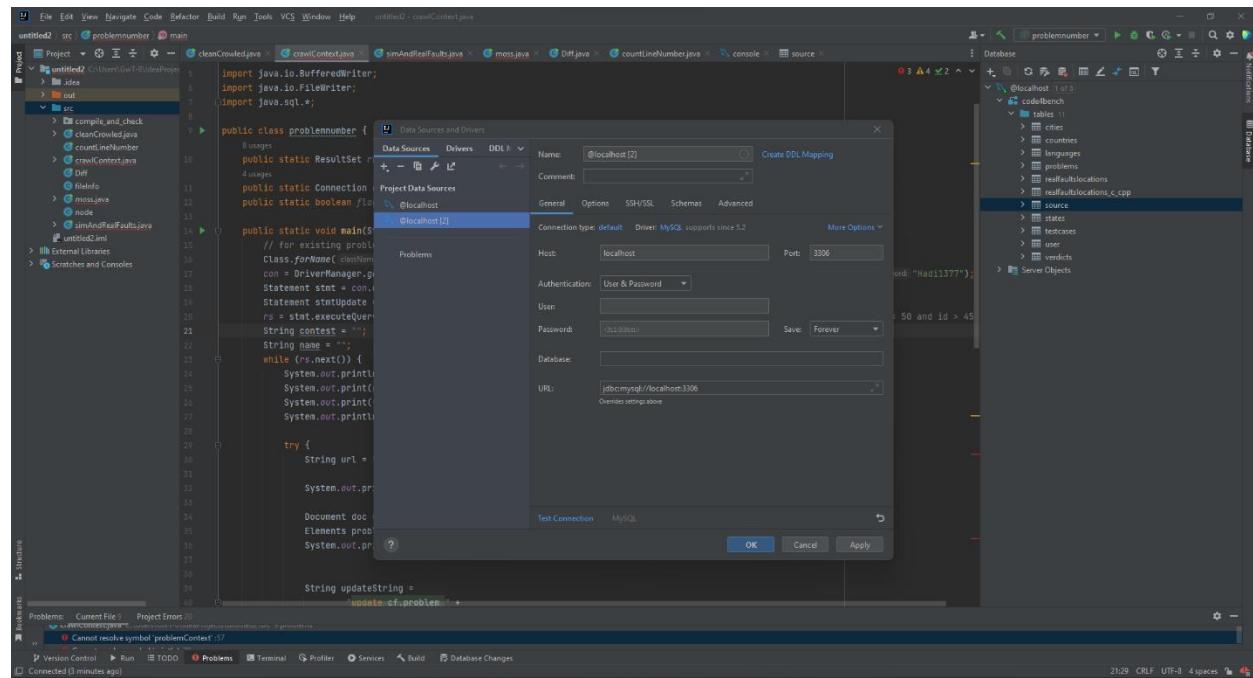


"Then, from the server data section, we import the extracted downloaded file into the Data Import section (this process is time-consuming, taking about 30 to 40 minutes).".



"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. .



The screenshot shows the IntelliJ IDEA interface with the Database tool window open. The table lists 32 rows of source codes from the 'source' table in the 'codebench' database. The columns are: Id, submission, author, memory, time, sent, and count. The data is as follows:

Id	submission	author	memory	time	sent	count
1	23580728 #include <iostream> using namespace std;#define _	1	2660	61 2017-01-06 16:55:16		
2	23621072 #include <bits/stdc++.h>#define ll long long	2	1848	15 2017-01-07 12:45:10		
3	23621098 #include <bits/stdc++.h>#define ll long long	2	1876	15 2017-01-07 12:46:52		
4	23621401 #include <bits/stdc++.h>#using namespace std;#define long long	3	2660	1800 2017-01-07 13:04:24		
5	23622160 #include <iostream> #include <ctdios> #include <algo>	4	2664	31 2017-01-07 13:50:16		
6	23622253 #include <iostream> #include <ctdios> #include <algo>	4	2668	46 2017-01-07 13:55:32		
7	23625119 #include <stdio.h> int main(int argc, char const *argv[5	1852	15 2017-01-07 17:08:44		
8	23629194 #include <iostream> using namespace std;#define NN	6	2032	15 2017-01-07 19:24:29		
9	23629357 #include <iostream> using namespace std;#define NN	7	2044	30 2017-01-07 19:34:40		
10	23629847 #define a,b,k,p long int;#define result(a,b,c,k) (a	8	1584	31 2017-01-07 21:19:46		
11	23633457 #include <iostream> using namespace std;#int main()	9	1876	15 2017-01-07 22:09:29		
12	23635870 #include <bits/stdc++.h>#using namespace std;#int	10	2824	15 2017-01-07 22:37:23		
13	23635950 #include <bits/stdc++.h>#using namespace std;#int	10	2824	0 2017-01-07 22:44:16		
14	23636179 #include <bits/stdc++.h>#using namespace std;#defin	11	3452	218 2017-01-07 23:02:45		
15	23639057 #include <bits/stdc++.h>#define NN 111111#using n	12	2040	15 2017-01-08 07:17:08		
16	23639667 #include <bits/stdc++.h>#define NN 111111#using n	12	2048	15 2017-01-08 07:18:52		
17	23641796 #include<iostream> using namespace std;#int main()	13	2044	15 2017-01-08 10:19:21		
18	23643392 #include<stdio.h>#int main()# int a, b, max, min,	14	1852	15 2017-01-08 12:21:06		
19	23647479 #include<bits/stdc++.h>#using namespace std;#int	15	2808	61 2017-01-08 17:07:25		
20	23647556 #include<bits/stdc++.h>#using namespace std;#defin	16	2040	15 2017-01-08 17:12:27		
21	23648351 import java.util.Scanner;/ public class cf581A/{@ p	17	2044	171 2017-01-08 18:07:51		
22	23648434 #include <iostream> #define ll long long#using n	18	3064	312 2017-01-08 18:11:58		
23	23648523 #include <iostream> #define ll long long#using n	18	3612	327 2017-01-08 18:19:11		
24	23655436 #include<bits/stdc++.h>#using namespace std;#int	19	2440	15 2017-01-08 21:57:22		
25	23655455 #include<bits/stdc++.h>#using namespace std;#int	20	2048	233 2017-01-08 22:34:02		
26	23655862 #include<bits/stdc++.h>#using namespace std;#int	19	2804	280 2017-01-08 22:44:25		
27	23655724 #include<bits/stdc++.h>#using namespace std;#int	19	2035	0 2017-01-08 22:71:13		
28	23655796 #include<bits/stdc++.h>#using namespace std;#int	19	2815	295 2017-01-08 22:28:04		
29	23656274 #include<bits/stdc++.h>#using namespace std;#defin	21	3444	202 2017-01-08 23:08:44		
30	23656313 #include<bits/stdc++.h>#using namespace std;#defin	21	3444	217 2017-01-08 23:09:36		
31	23657889 import java.util.Scanner;/ public class A_581/{@ pri	22	20468	234 2017-01-09 02:46:33		
32	23659516 #include <bits/stdc++.h>#using namespace std;#defin	23	2816	358 2017-01-09 08:38:29		

"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".

The screenshot shows a Java IDE interface with several tabs and panes. In the top-left, there's a 'Project' view showing files like 'countlineHandler.java', 'main.java', and 'testcases'. The main pane displays a SQL query:

```

SELECT *
FROM (
    SELECT *,
        @row_number := IF(@prev_problems_id = problems_id, @row_number + 1, 1) AS row_num,
        @prev_problems_id := problems_id
    FROM source AS s
    CROSS JOIN (SELECT @row_number := 0, @prev_problems_id := NULL) AS vars
    WHERE s.languages_id = 7
    AND s.countline BETWEEN 60 AND 100
    ORDER BY problems_id, id
) AS sub
WHERE sub.row_num <= 3;
  
```

To the right of the code editor is a 'Database' browser showing tables like 'cities', 'countries', 'languages', 'problems', 'resultLocations', and 'source'. Below the code editor is a 'Services' pane with a 'Database' section showing a table of submissions. The table has columns: id, submission, sourceCode, author, memory, time, sent, countline, and problems_id. The data looks like this:

	id	submission	sourceCode	author	memory	time	sent	countline	problems_id
1.	538	24102282	import java.util.Scanner; public class JavaApplication0 { public static void main(String[] args) { Scanner scanner = new Scanner(System.in); String str = scanner.nextLine(); System.out.println(str); } }	pu...	206	20404	592	2017-01-25 03:59:37	89
2.	627	24224242	import java.io.BufferedReader;import java.io.BufferedWriter;import java.io.IOException;import java.io.InputStreamReader;import java.io.OutputStreamWriter;public class LuxuriousHouses { public static void main(String[] args) { BufferedReader reader = new BufferedReader(new InputStreamReader(System.in)); BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(System.out)); String str = reader.readLine(); writer.write(str); writer.flush(); } }	pu...	261	20500	436	2017-01-28 15:02:35	71
3.	769	24356739	import java.util.Scanner;public class A_581 { private static Scanner scanner = new Scanner(System.in); public static void main(String[] args) { int a = scanner.nextInt(); if (a > 0) { System.out.println("Positive"); } else if (a < 0) { System.out.println("Negative"); } else { System.out.println("Zero"); } } }	pu...	289	20520	904	2017-02-02 03:47:53	65
4.	31	23657889	import java.util.Scanner;public class A_581 { private static Scanner scanner = new Scanner(System.in); public static void main(String[] args) { int a = scanner.nextInt(); if (a > 0) { System.out.println("Positive"); } else if (a < 0) { System.out.println("Negative"); } else { System.out.println("Zero"); } } }	pu...	22	20468	234	2017-01-09 02:46:33	83
5.	997	24594610	import java.util.Scanner;public class A_581 { private static Scanner scanner = new Scanner(System.in); public static void main(String[] args) { int a = scanner.nextInt(); if (a > 0) { System.out.println("Positive"); } else if (a < 0) { System.out.println("Negative"); } else { System.out.println("Zero"); } } }	pu...	384	20388	124	2017-02-11 20:45:33	65
6.	998	24594657	import java.util.Scanner;public class A_581 { private static Scanner scanner = new Scanner(System.in); public static void main(String[] args) { int a = scanner.nextInt(); if (a > 0) { System.out.println("Positive"); } else if (a < 0) { System.out.println("Negative"); } else { System.out.println("Zero"); } } }	pu...	384	20408	171	2017-02-11 20:48:13	75
7.	529	24095843	import java.util.Arrays;import java.util.Scanner;public class A_581 { public static void main(String[] args) { Scanner scanner = new Scanner(System.in); String str = scanner.nextLine(); System.out.println(str); } }	pu...	202	20468	146	2017-01-24 19:55:41	87
8.	530	24096189	import java.util.Arrays;import java.util.Scanner;public class A_581 { public static void main(String[] args) { Scanner scanner = new Scanner(System.in); String str = scanner.nextLine(); System.out.println(str); } }	pu...	202	20412	124	2017-01-24 20:11:24	79
9.	531	24096223	import java.util.Arrays;import java.util.Scanner;public class A_581 { public static void main(String[] args) { Scanner scanner = new Scanner(System.in); String str = scanner.nextLine(); System.out.println(str); } }	pu...	202	20400	1000	2017-01-24 20:15:07	81
10.	6829	13404913	import java.io.BufferedReader;import java.io.InputStreamReader;import java.util.ArrayList;import java.util.Collections;import java.util.List;	pu...	2243	21772	311	2015-10-04 18:57:45	97
11.	6706	13404932	import java.util.ArrayList;import java.util.Collections;import java.util.List;	pu...	2209	0	2000	2015-10-04 21:02:39	91
12.	6708	13404954	import java.util.ArrayList;import java.util.Collections;import java.util.List;	pu...	2209	0	156	2015-10-04 21:04:45	95

At the bottom left, a terminal window shows the command: `source [localhost]:(SELECT * FROM codebenchsource LIMIT 100 // OFFSET 50) completed. (yesterday 9:55 PM)`.

"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

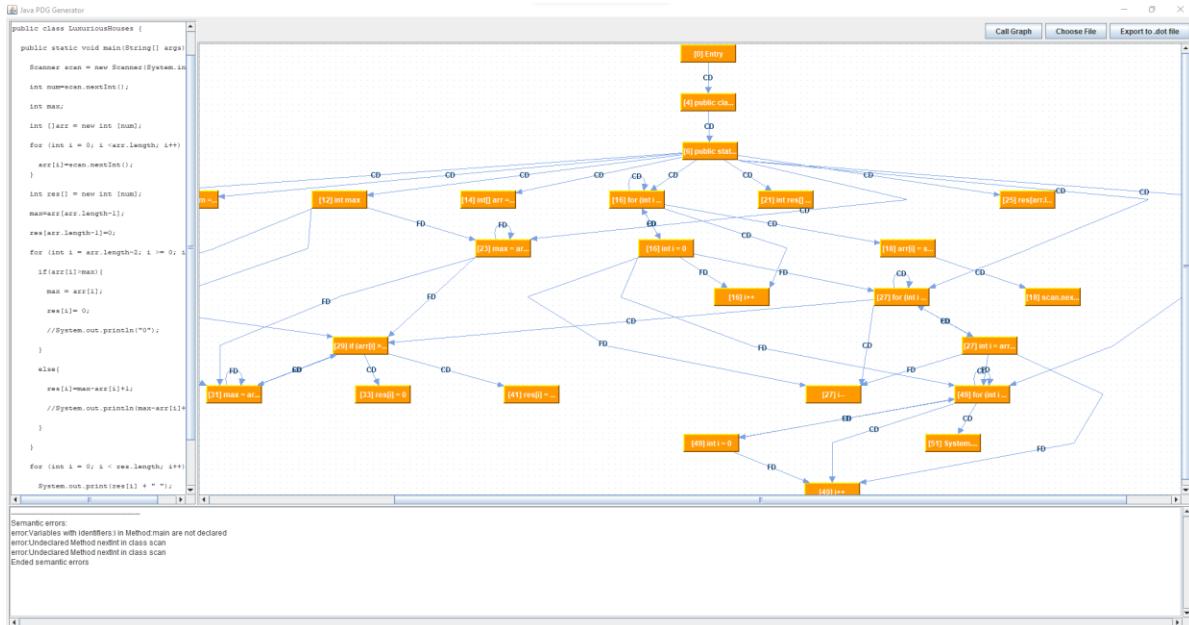
- problems_id = 13
- submission = 17382618
- languages_id = 7

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

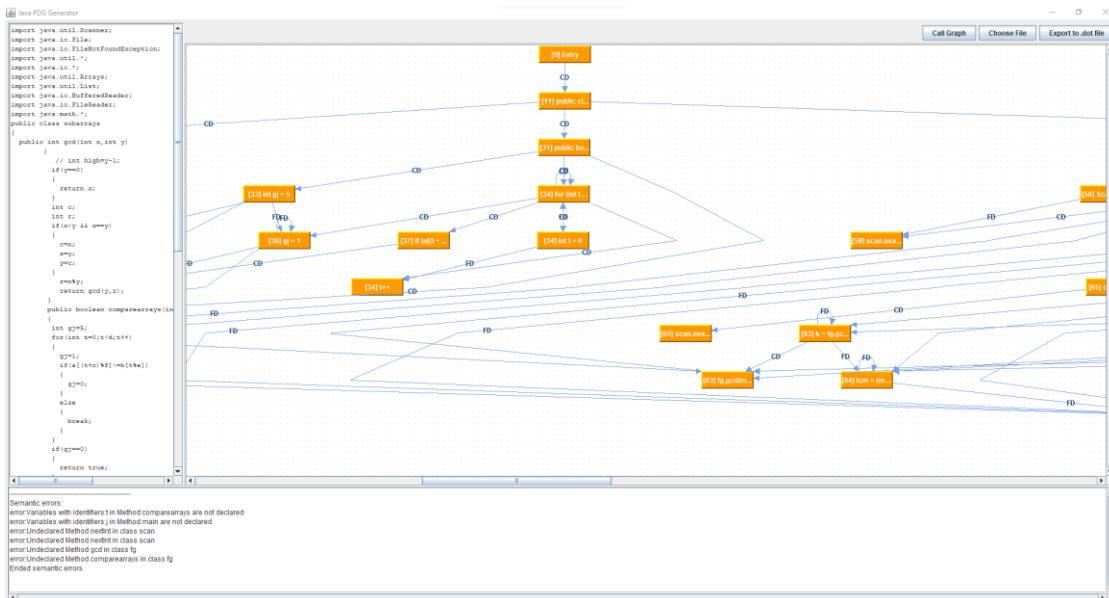
"Now, we add each program as a project in IntelliJ IDEA and save it in the Java format".

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

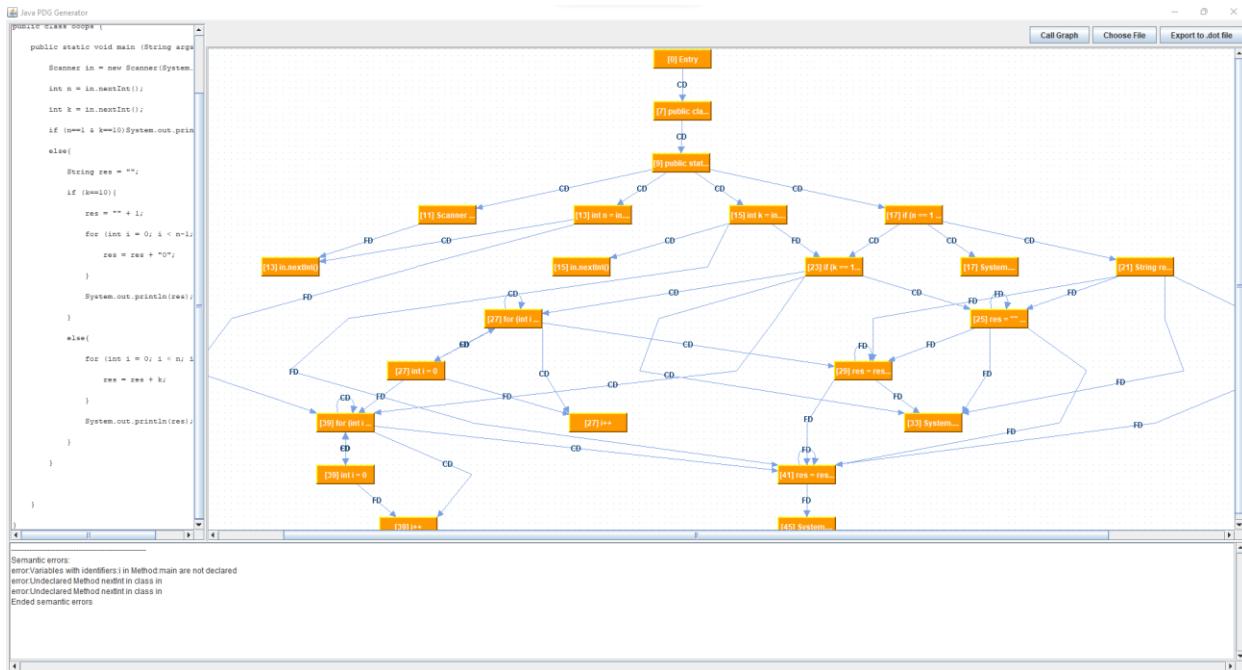
PDG for Source code one :



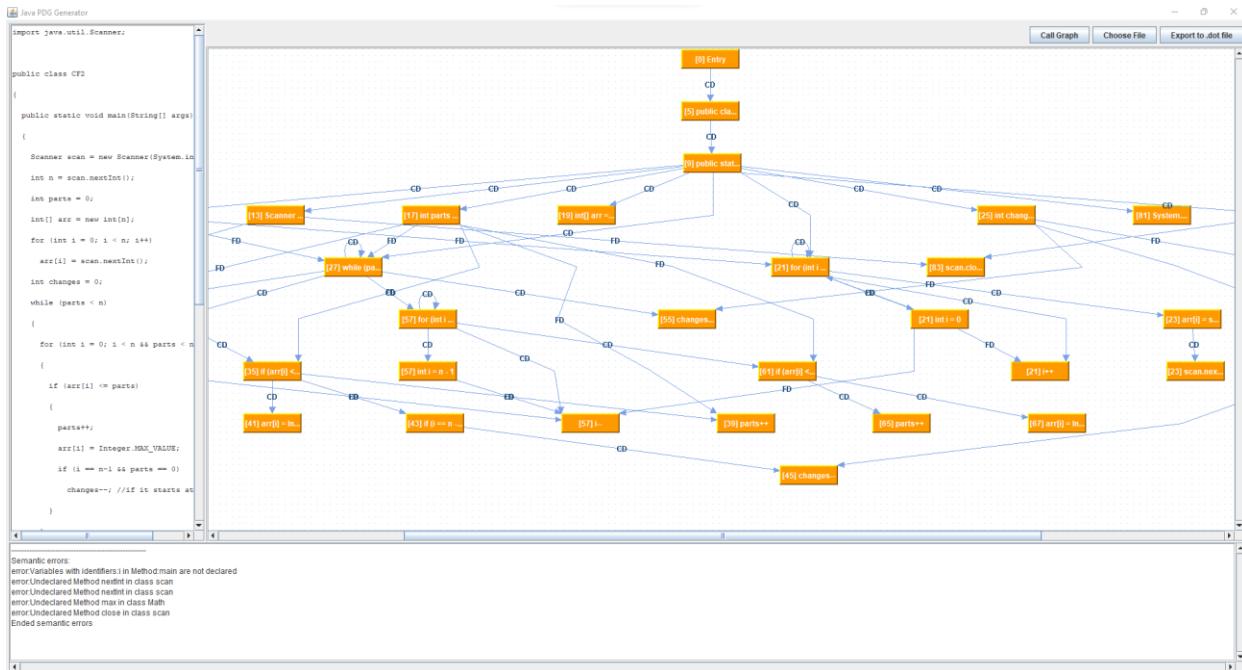
PDG for Source code two :



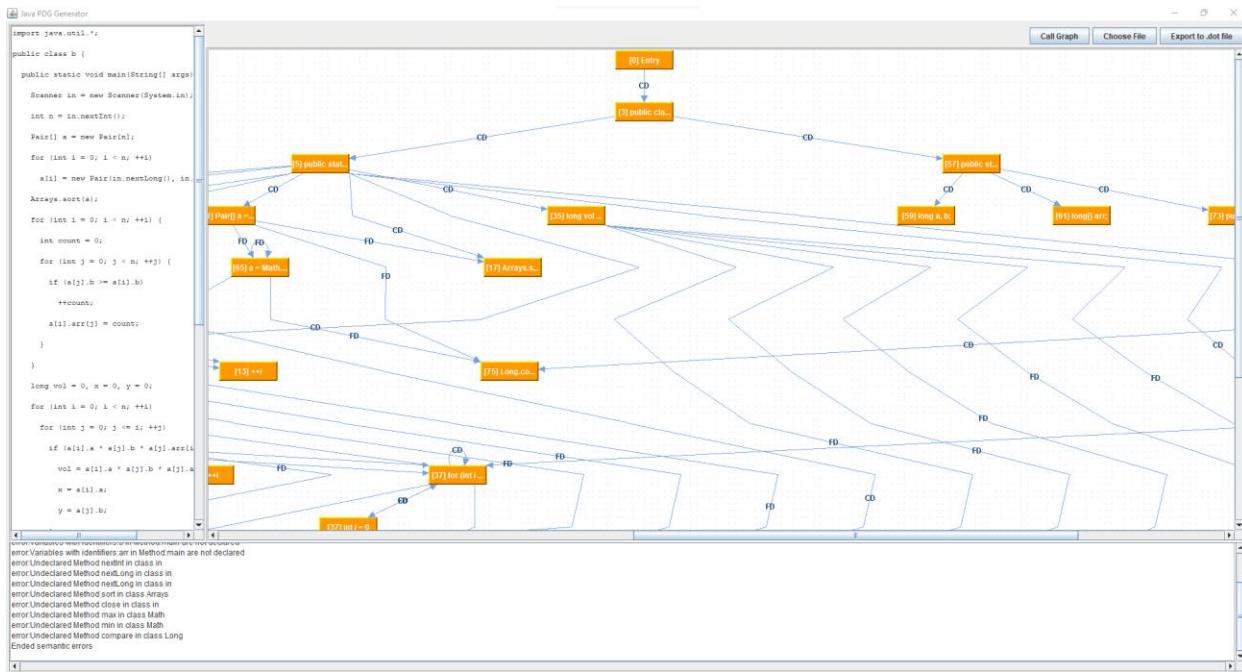
PDG for Source code three:



PDG for Source code four:

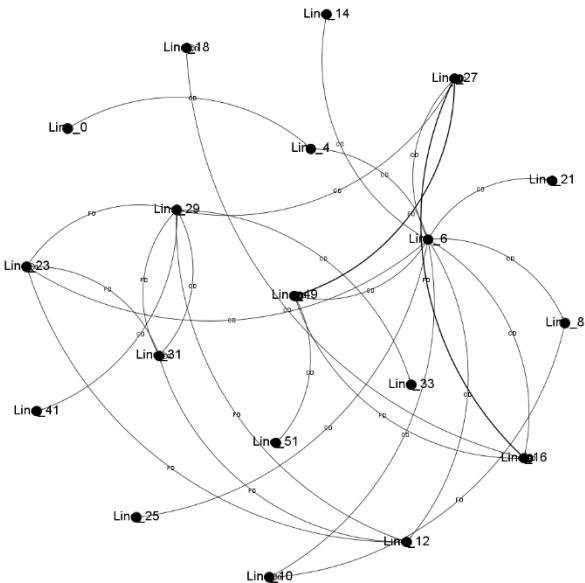


PDG for Source code five:

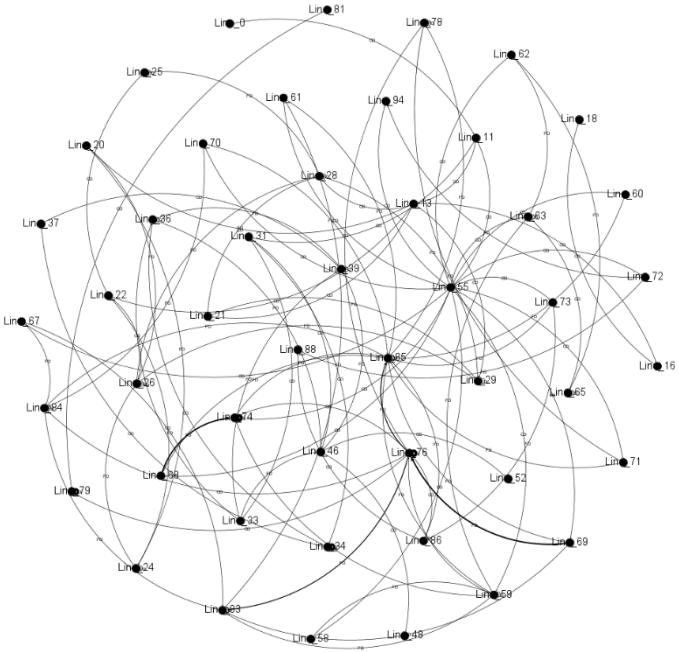


"In this tool, control dependencies are indicated by the symbol 'CD' on arrows, and data dependencies are displayed with the symbol 'FD'. Now, we can export from this tool in the .dot format, which gives us the dependencies between programs along with labels. Here, I was unable to export from this tool 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."

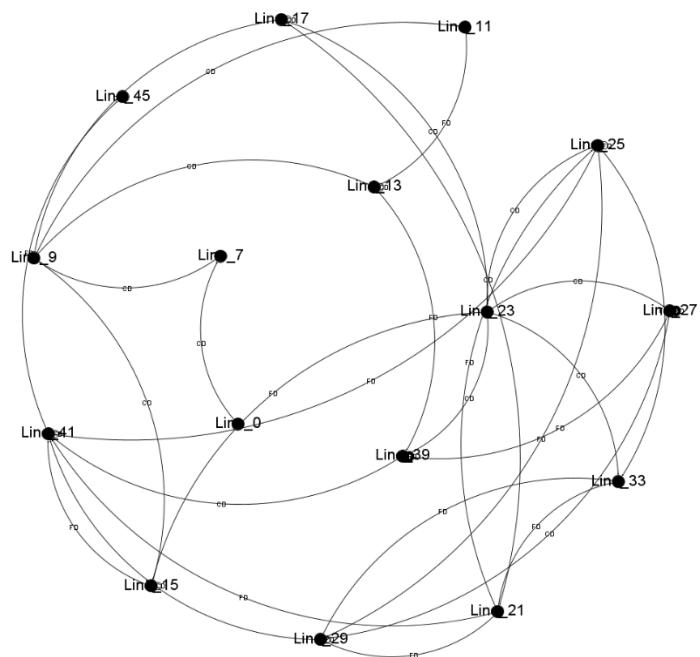
: Gephi Source one



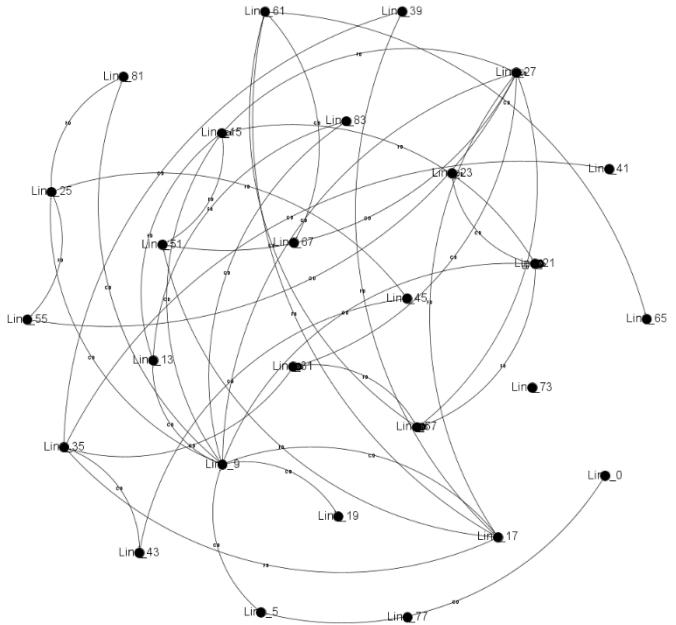
: Gephi Source two



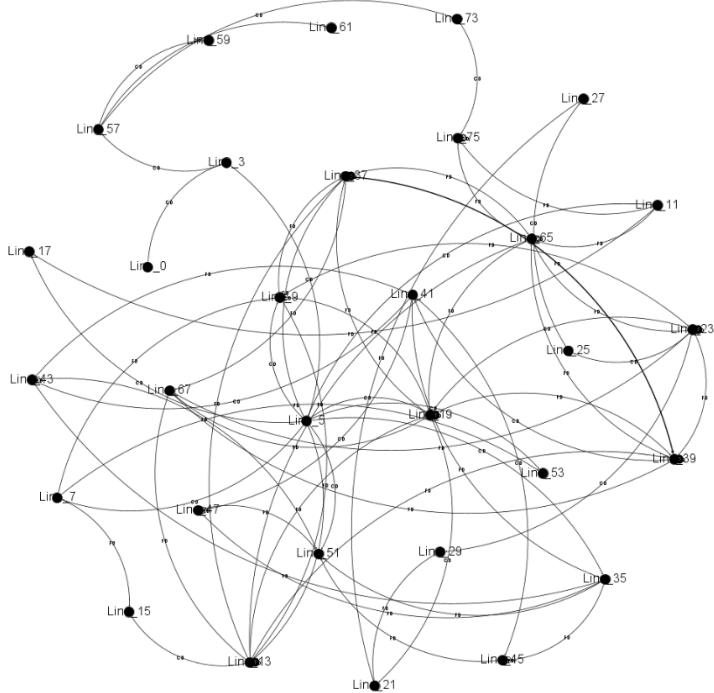
: Gephi Source three



: 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 = 1^v
- problems_id = 1^r
- problems_id = 1^l

"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

problems_id	isValid
1	1 pass
2	1 pass
3	1 pass
4	1 pass
5	1 pass
6	1 pass
7	1 pass
8	1 pass
9	1 pass
10	1 pass
11	1 pass
12	1 pass
13	1 pass

Problem_id = 8

problems_id	isValid
1	0 pass
2	0 pass
3	0 pass
4	0 pass
5	0 pass
6	0 pass
7	0 pass
8	0 pass

Problem_id =17

The screenshot shows a Java project named 'united5' in the IDE. The 'Database' tool is open, displaying a table named 'testcases' with one row. The Coverage tool on the right shows a tree view of the codebase and a summary for the 'CF2' branch, indicating 17 passes out of 17 total.

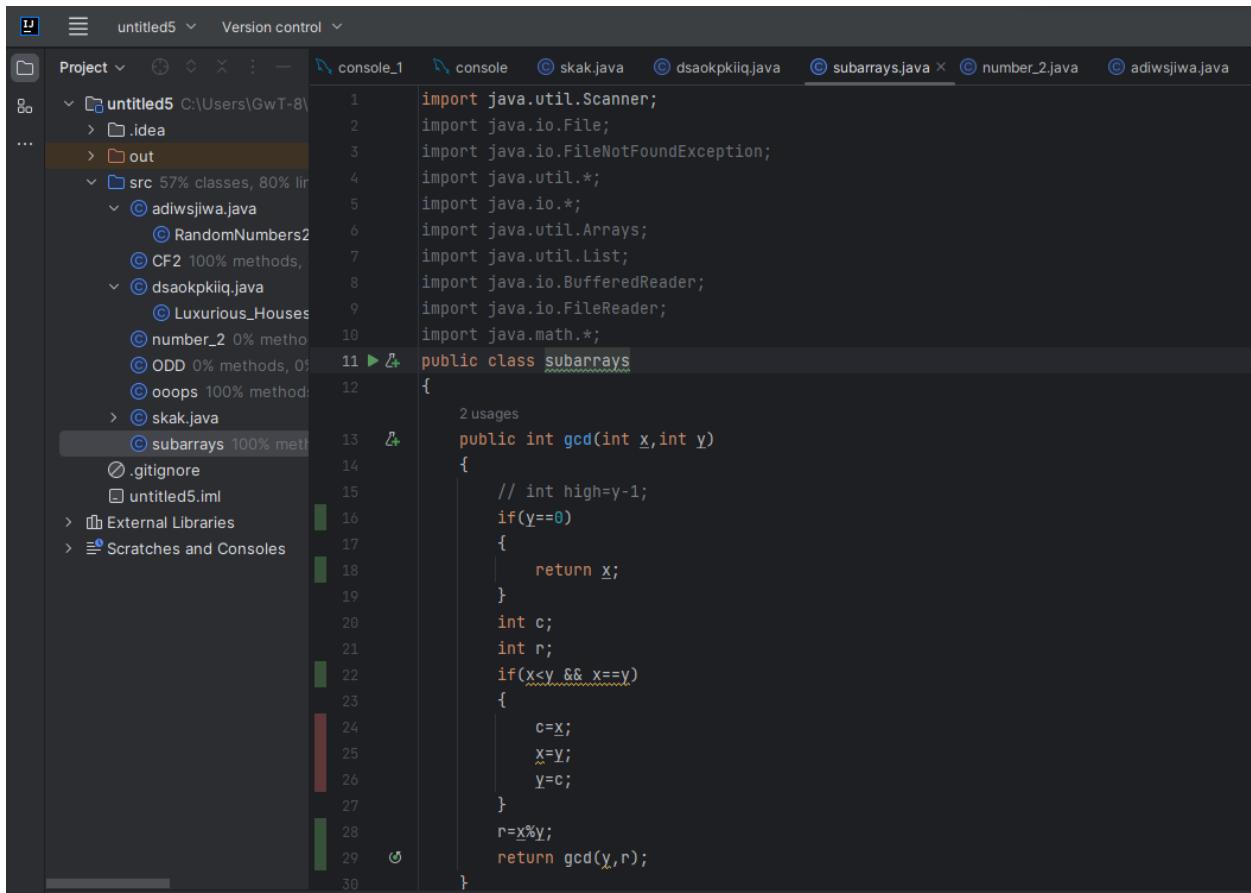
Problem_id =13

The screenshot shows the same Java project 'united5'. The Coverage tool now shows 13 passes out of 13 total for the 'CF2' branch.

Problem_id =51

The screenshot shows the Java project 'united5'. The Coverage tool shows 51 passes out of 51 total for the 'CF2' branch.

"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 screenshot shows the IntelliJ IDEA interface with a Java project named "untitled5". The project structure on the left includes a .idea folder, an out folder, and a src folder containing several Java files: adiwsjiwa.java, CF2.java, dsaokpkiq.java, Luxurious_Houses.java, number_2.java, ODD.java, ooops.java, skak.java, and subarrays.java. The subarrays.java file is currently selected and shown in the editor. The code for subarrays.java is as follows:

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;
import java.io.BufferedReader;
import java.io.FileReader;
import java.math.*;

public class subarrays
{
    2 usages
    public int gcd(int x,int y)
    {
        // int high=y-1;
        if(y==0)
        {
            return x;
        }
        int c;
        int r;
        if(x<y && x==y)
        {
            c=x;
            x=y;
            y=c;
        }
        r=x%y;
        return gcd(y,r);
    }
}
```

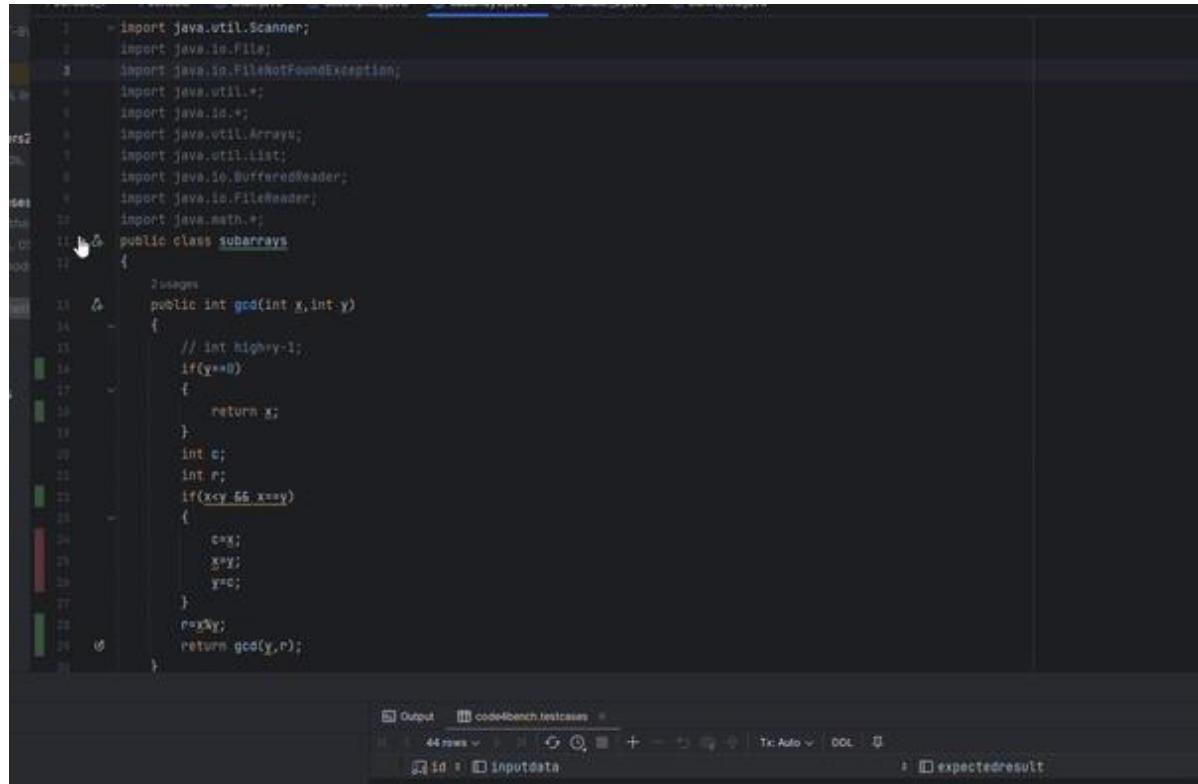
The code editor highlights certain lines in green, indicating they have been reached during execution. Lines 1 through 10 are mostly white, while lines 11 through 30 are green. Line 11 has a small green icon next to it, likely indicating a break point or a specific analysis point.

"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".



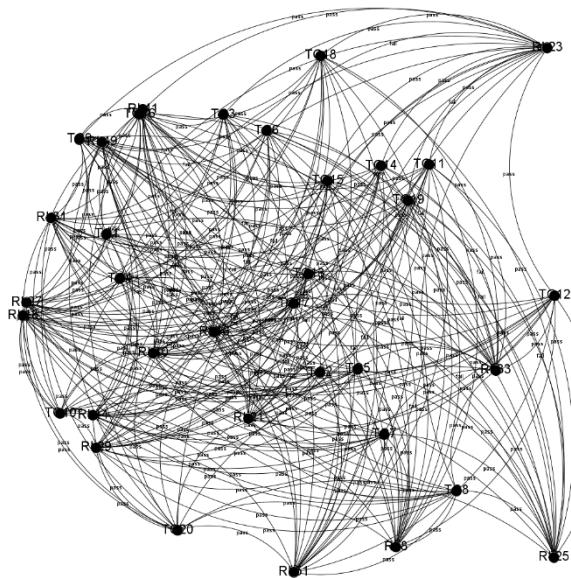
The screenshot shows a Java code editor with a dark theme. The code is a Java class named `subarrays` containing a static method `gcd`. The code implements the Euclidean algorithm for finding the greatest common divisor of two integers `x` and `y`. The editor includes syntax highlighting, line numbers, and a status bar at the bottom.

```
1  import java.util.Scanner;
2  import java.io.File;
3  import java.io.FileNotFoundException;
4  import java.util.*;
5  import java.util.Arrays;
6  import java.util.List;
7  import java.io.BufferedReader;
8  import java.io.FileReader;
9  import java.math.*;
10 public class subarrays
11 {
12     2 usages
13     4 public int gcd(int x,int y)
14     {
15         // int high=x-1;
16         if(y==0)
17         {
18             return x;
19         }
20         int c;
21         int r;
22         if(x<y) x=y;
23         {
24             c=x;
25             x=y;
26             y=c;
27         }
28         r=x%y;
29         if(r==0)
30             return gcd(y,r);
31     }
32 }
```

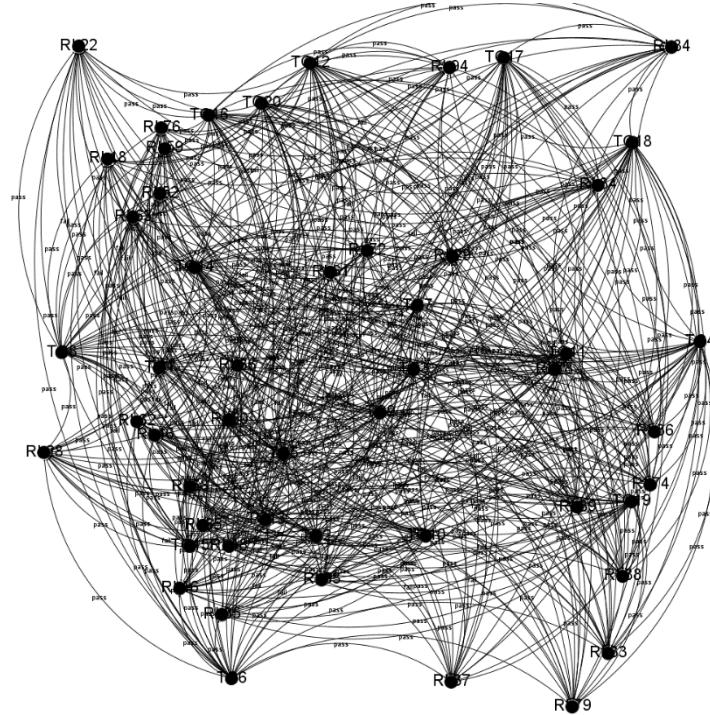
"In the next step, using the covered lines for each test case and the isVaild indicator, which shows whether each test case passes or fails, we create a file in the .dot format."

" For each testcase, it is as follows":

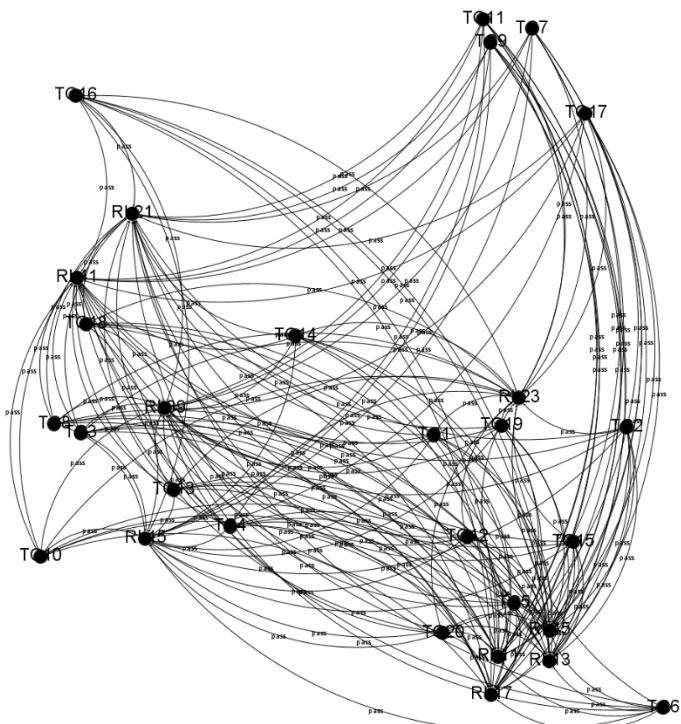
: Gephi Source one



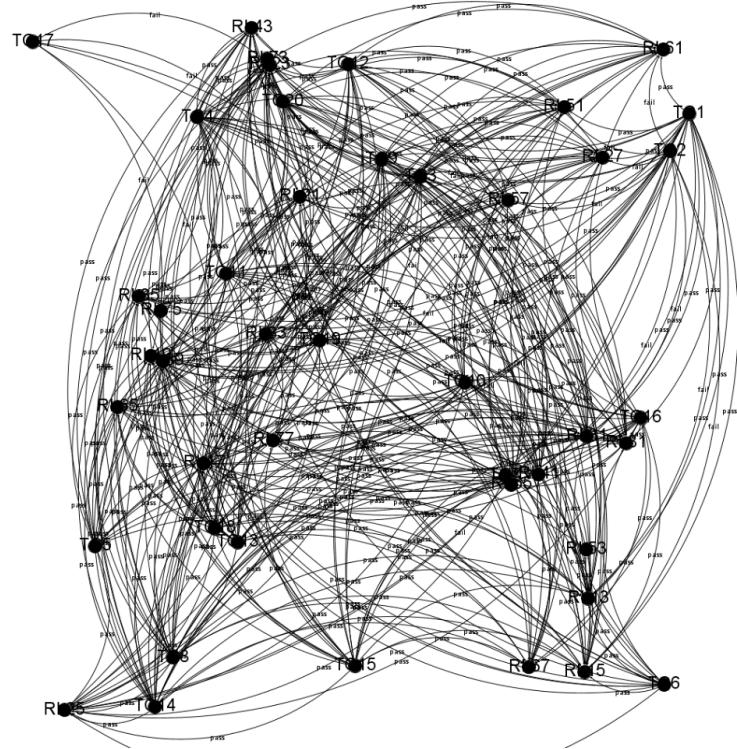
: Gephi Source two



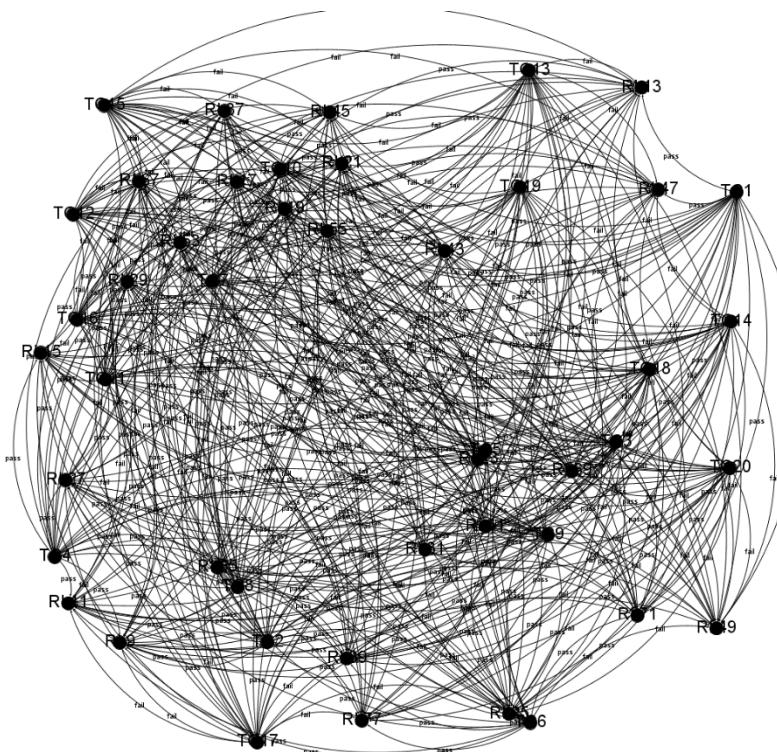
: Gephi Source three



: Gephi Source four

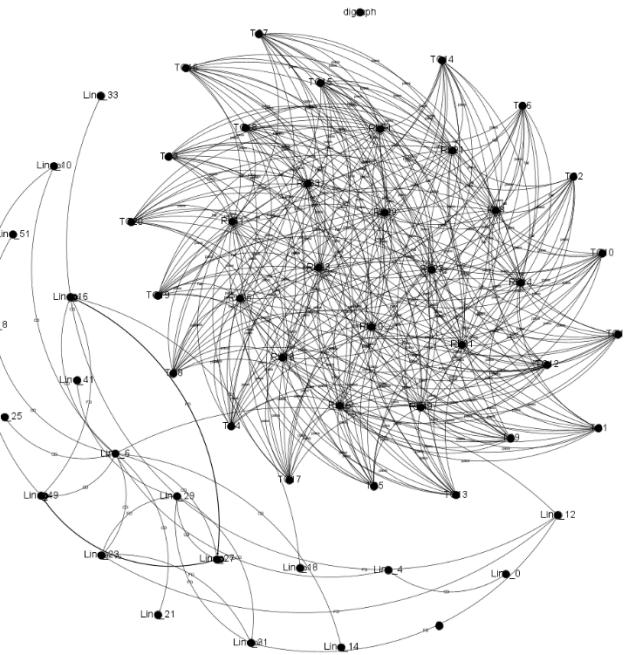


: Gephi Source five

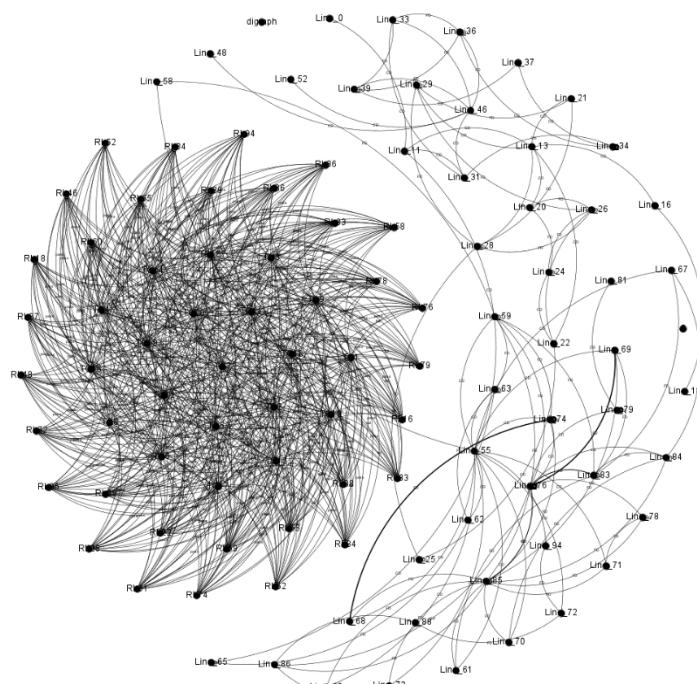


"Now, we combine the given files."

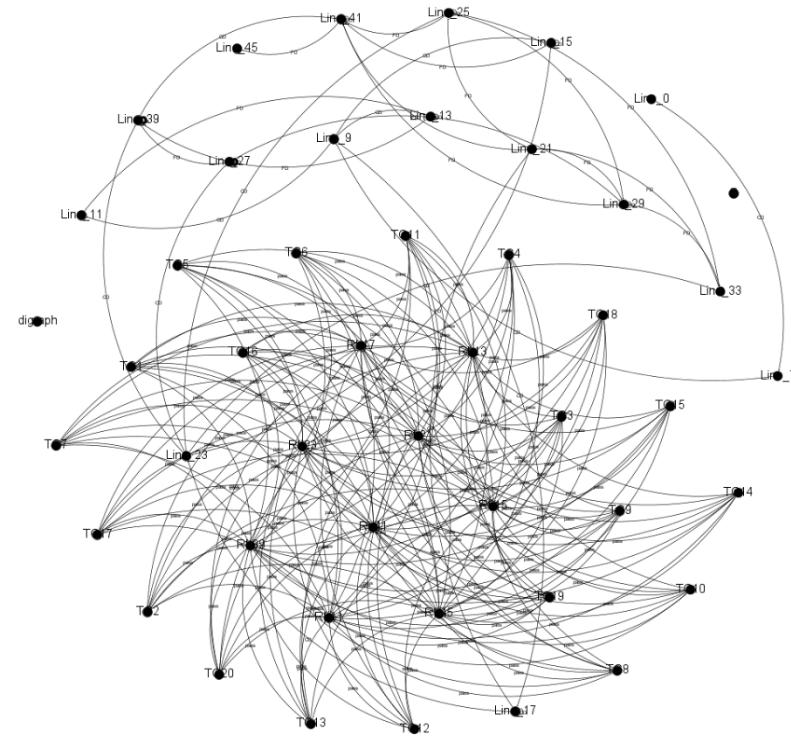
: Gephi + TestCase Source one



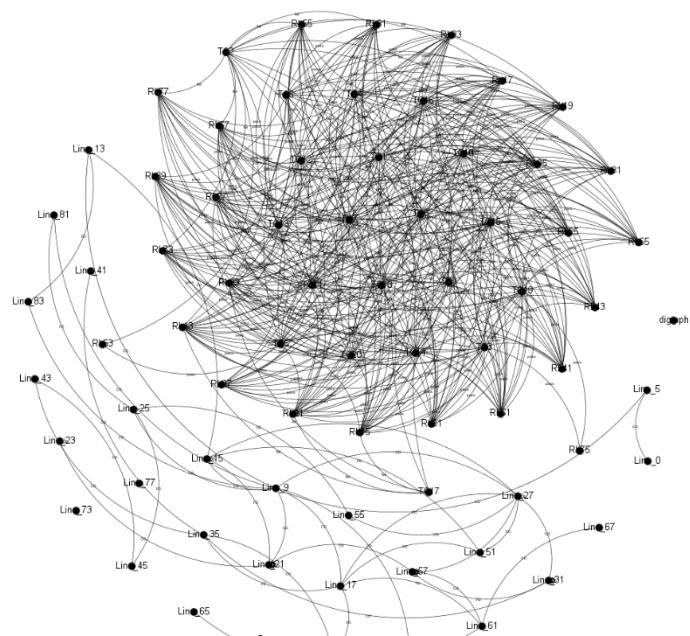
: Gephi + TestCase Source two



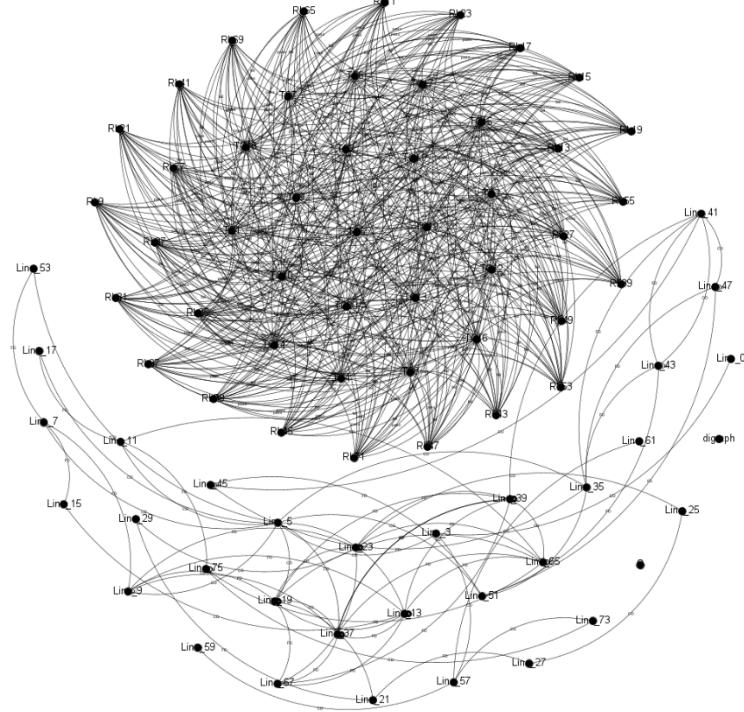
: Gephi +TestCase Source three



: Gephi +TestCase Source four



: Gephi + TestCase Source five



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

A screenshot of the Gephi software interface. The top menu bar includes File, Workspace, View, Tools, Window, and Help. The title bar shows "Gephi 0.10.1 - Untitled 4". Below the menu is a toolbar with various icons for file operations. The main workspace displays a network graph with nodes and edges. On the left, there are panels for Appearance (Nodes, Edges, Unique Partition, Ranking), Layout (Fruchterman Reingold, Streaming, Twitter Streaming Importer V2), and Global, Edges, Labels settings. A central dialog box titled "Weighted Degree Report" shows a chart titled "Degree Distribution" with "Value" on the x-axis (ranging from -1 to 21) and "Count" on the y-axis (ranging from 0 to 20). The chart contains several red data points. To the right of the workspace is a "Statistics" panel with a table of network metrics and their run status. The table includes: Average Degree (5.946, Run), Avg. Weighted Degree (6.196, Run), Minimum Spanning Tree (Run), Network Diameter (Run), Graph Density (Run), Bridging Centrality (Run), HITS (Run), Clustering Coefficient (Run), PageRank (Run), Connected Components (Run), BoundingDiameters (Run), DBSCAN (Run), Girvan-Newman Clustering (Run), Leiden algorithm (Run), Community Detection (Run), Modularity (Run), Statistical Inference (Run), Node Overview (Run), Avg. Clustering Coefficient (Run), Eigenvector Centrality (Run), Multidimensional scaling (Run), Lineage (Run), Katz Centrality (Run), and Edge Overdensity (Run). The "Community Detection" section is highlighted with a yellow border.