Software Testing
Project Report

Session: Spring 2021

Danish Hasan	MSCS-20001
Abu Bakar	MSCS-20013
Musa Khan	MSCS-20065
Awais	MSCS-20074

Employee Time Reporting

Department of Computer Science

Information Technology University Lahore

Pakistan

Project Description

ENVIRONMENT SETUP

1. Download maven from here: https://maven.apache.org/download.cgi
2. Download and install the mysql workbench from here: https://dev.mysql.com/downloads/installer/
3. Download jdk1.8+
4. In the .\timesheet-master\build.bat, set the JAVA_HOME to jdk path and similarly set MAVEN_HOME to the maven path.
5. In the .\timesheet-master\run.bat, set the JAVA_HOME and set CATALINA_HOME to absolute path appended by ".\PaySystem\apache-tomcat-7.0.108-windows-x64\apache-tomcat-7.0.108".
6. Open Command prompt, navigate to project repository i.e .\Paysystem\timesheet-master\ and execute build.bat.
7. This will build the project.
8. Open mysql workbench and enter following two queries:
    a. drop database paysystem;
    b. create database paysystem;
9. When the database is created for first time, only execute the create query.
10. Execute run.bat.

DESCRIPTION

The project is a lighter version of a pay system for managing the expenses of the employees.

- Adding the new employees in the database.
- Adding the time worked for a specific employee.
- Configuring the database settings.
- Managing the groups in the company.
- Generate the ADP reports of the employees.

APPLICATION RUNNING

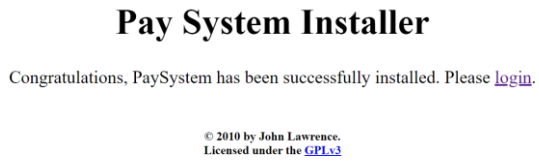
After the local server is running, go to http://localhost:8090/ or you can just go to the application http://localhost:8090/PaySystem

Enter the company name, and then click next.

Then you will be redirected to add information about the database. To avoid confusion, database username and database password are kept same.

You will be redirected to add username and password for the user purpose. These are also kept same.

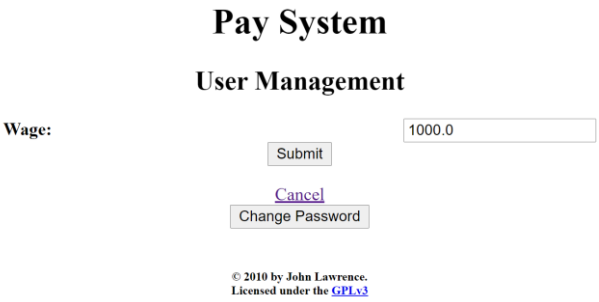
You will be redirected to the login page.



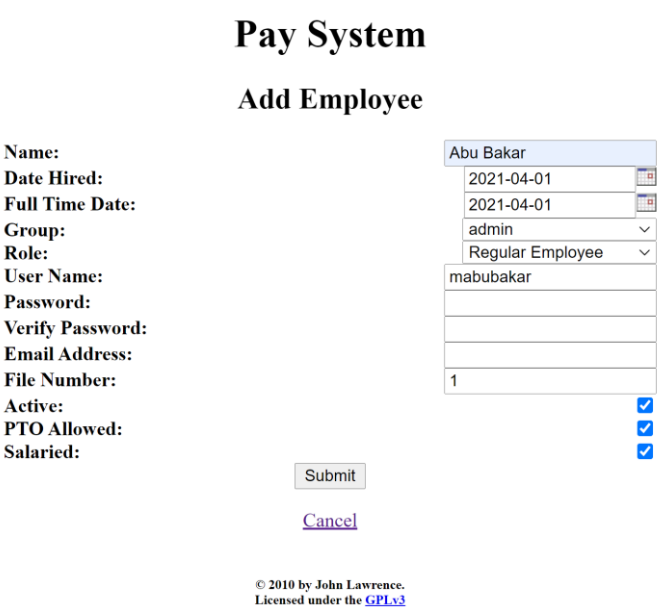
After clicking login, Login using the username you set earlier.

After login you will be directed to the dashboard. Below is the full dashboard.

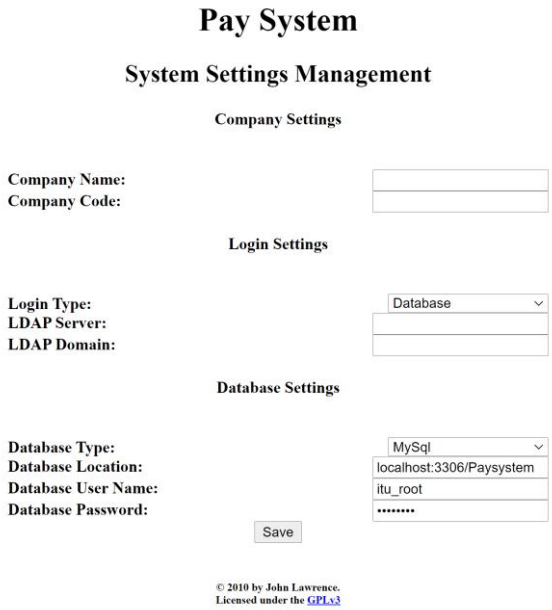
In the manage account section, you can add the wage.



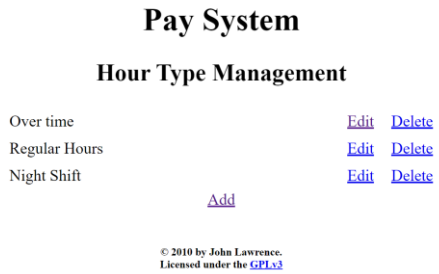
In the manage employee section, you can add/delete the employees.



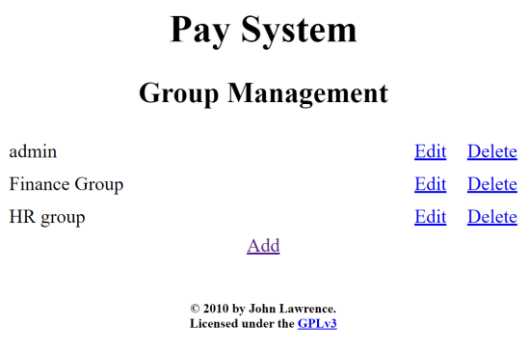
In the manage settings section, you can change the settings.



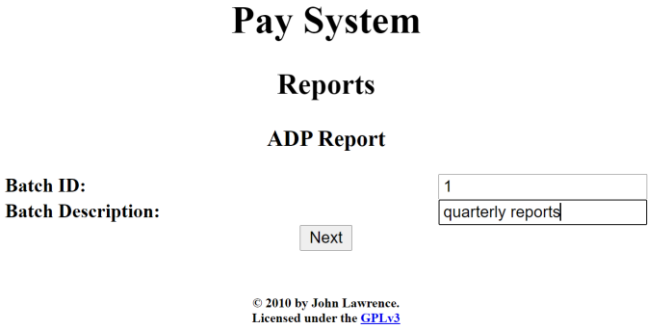
In the hour management section, you can add/delete/edit the hour types.



In the group management section, you can add/delete/edit the groups.



In the report section, you can generate the reports.



For the report generation, you can add the data for the employee.

After clicking the finalize data, a csv file is downloaded.

White-Box Testing

FUNCTION 1:

Encodes a byte array into Base64 format.

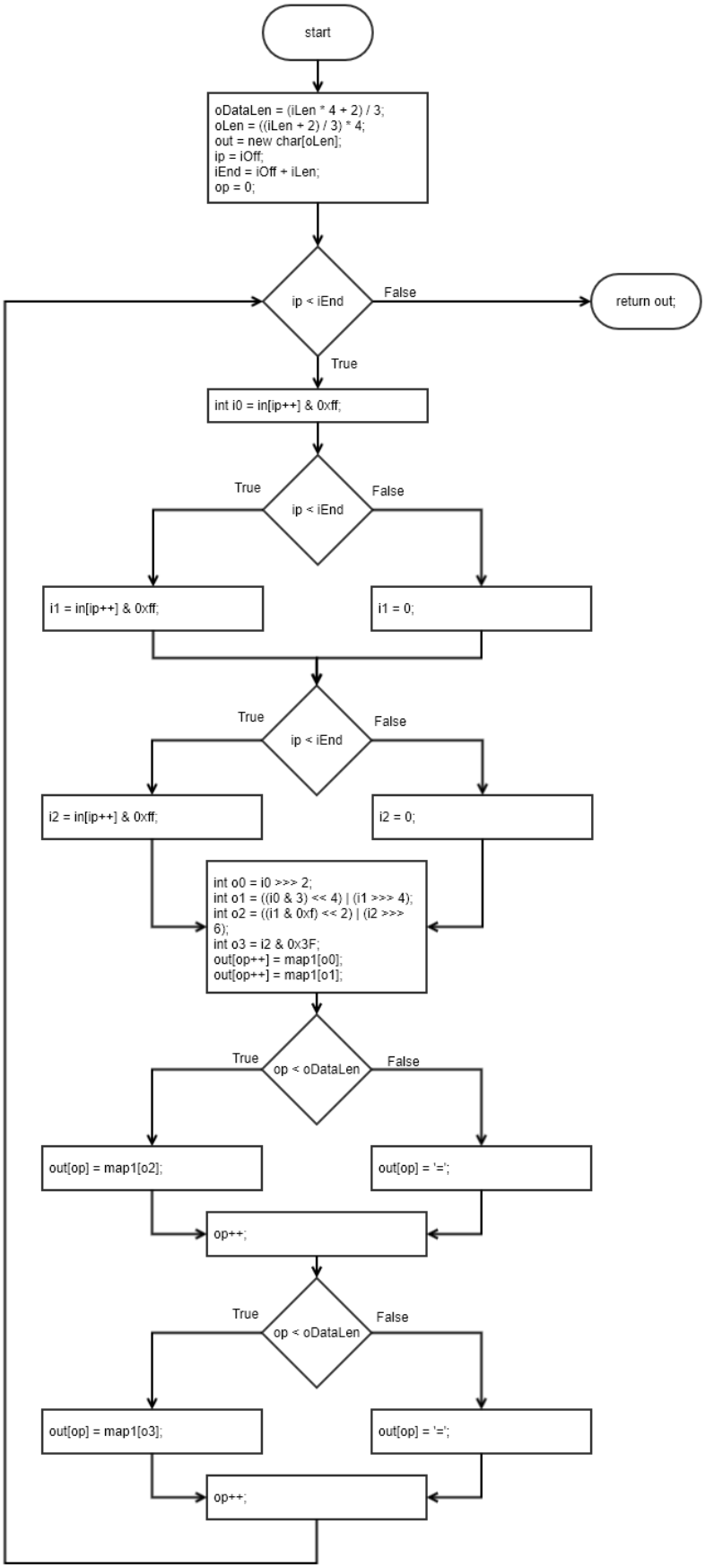
Note: map[] table is populated in another constructor function.

Source Code:

timesheet-master\src\main\java\timeSheet\util\properties\Base64Coder.java

```
59      public char[] encode(byte[] in, int iOff, int iLen) {
60          int oDataLen = (iLen * 4 + 2) / 3;       // output length without padding
61          int oLen = ((iLen + 2) / 3) * 4;         // output length including padding
62          char[] out = new char[oLen];
63          int ip = iOff;
64          int iEnd = iOff + iLen;
65          int op = 0;
66          while (ip < iEnd) {
67              int i0 = in[ip++] & 0xff;
68              int i1 = ip < iEnd ? in[ip++] & 0xff : 0;
69              int i2 = ip < iEnd ? in[ip++] & 0xff : 0;
70              int o0 = i0 >>> 2;
71              int o1 = ((i0 & 3) << 4) | (i1 >>> 4);
72              int o2 = ((i1 & 0xf) << 2) | (i2 >>> 6);
73              int o3 = i2 & 0x3F;
74              out[op++] = map1[o0];
75              out[op++] = map1[o1];
76              out[op] = op < oDataLen ? map1[o2] : '=';
77              op++;
78              out[op] = op < oDataLen ? map1[o3] : '=';
79              op++;
80          }
81          return out;
82      }
```

CFG:


start
oDataLen = (iLen * 4 + 2) / 3;
oLen = ((iLen + 2) / 3) * 4;
out = new char[oLen];
ip = iOff;
iEnd = iOff + iLen;
op = 0;
ip < iEnd
False
return out;
True
int i0 = in[ip++] & 0xff;
ip < iEnd
True
False
i1 = in[ip++] & 0xff;
i1 = 0;
ip < iEnd
True
False
i2 = in[ip++] & 0xff;
i2 = 0;
int o0 = i0 >>> 2;
int o1 = ((i0 & 3) << 4) | (i1 >>> 4);
int o2 = ((i1 & 0xf) << 2) | (i2 >>>
6);
int o3 = i2 & 0x3F;
out[op++] = map1[o0];
out[op++] = map1[o1];
op < oDataLen
True
False
out[op] = map1[o2];
out[op] = '=';
op++;
op < oDataLen
True
False
out[op] = map1[o3];
out[op] = '=';
op++;

Statement Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks

1	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 3;
		QUJD	QUJD	Pass	Covers all statements

Branch Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks

1	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 3;
		QUJD	QUJD	Pass	Covers 66TF, 68T, 69T, 76T, 78T

2	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 1;
		QQ==	QQ==	Pass	Covers 66TF, 68F, 69F, 76F, 78F

Condition Coverage with Short Circuit Evaluation:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks

1	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 3;
		QUJD	QUJD	Pass	Covers 66TF, 68T, 69T, 76T, 78T

2	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 1;
		QQ==	QQ==	Pass	Covers 66TF, 68F, 69F, 76F, 78F

Boundary Interior:

Possible logical paths

- 68T, 69T, 76T, 78T
- 68T, 69F, 76T, 78F
- 68F, 69F, 76F, 78F

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks

1	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 3;
		QUJD	QUJD	Pass	Covers 68T, 69T, 76T, 78T

2	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 1;
		QQ==	QQ==	Pass	Covers 68F, 69F, 76F, 78F

3	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 2;
		QUI=	QUI=	Pass	Covers 68T, 69F, 76T, 78F

Loop Boundary:

Consider N for loop boundary as 5

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks

1	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 0;
		Empty string	Empty string	Pass	Covers 66F

2	In[] = {'A', 'B', 'C'}; iOff = 0;	QUJD	QUJD	Pass	Covers 66T once

	iLen = 3;				

3	In[] = {'A', 'B', 'C', 'D'};
	iOff = 0;
	iLen = 4;
		QUJDRA==	QUJDRA==	Pass	Covers 66T at N-1

4	In[] = {'A', 'B', 'C', 'D', 'E'};
	iOff = 0;
	iLen = 5;
		QUJDREU=	QUJDREU=	Pass	Covers 66T at N

54	In[] = {'A', 'B', 'C', 'D', 'E', 'F'};
	iOff = 0;
	iLen = 6;
		QUJDREVG	QUJDREVG	Pass	Covers 66T at N+1

Basis Path:

Path 1: 66F

Path 2: 66T, 68T, 69T, 76T, 78T

Path 3: 66T, 68T, 69F, 76T, 78F

Path 4: 66T, 68F, 69F, 76F, 78F

Note that no logical path is possible to cause 69T while 68F. Same is case with 76F and 78T. Similarly, conditions in 76 and 78 also depend upon same factor as 68, 69 so it is not possible for 68T but 76F and vice versa.

Test case#	**Input**	**Output**	**Expected Output**	**Pass/Fail**	**Comments/Remarks**

1	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 3;
		QUJD	QUJD	Pass	Covers Path2

2	In[] = {'A', 'B', 'C'};	QQ==	QQ==	Pass	Covers Path4

	iOff = 0; iLen = 1;				

3	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 0;
		Empty String	Empty String	Pass	Covers Path1

4	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 2;
		QUI=	QUI=	Pass	Covers Path3

Data Flow Testing:

Variable #	Variable Name	Definitions	Uses
1	iLen	59	60, 61, 64
2	oLen	61	62
3	op	65, 74, 75, 77, 79	74, 75, 76, 77, 78, 79

Variable #	Variable Name	DU pairs
1	iLen	<59, 60>, <59, 61>, <59, 64>
2	oLen	<61, 62>
3	op	<65,74>, <74,75>, <75,76>, <75,77>, <77,78>, <77,79>, <79,74>

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks

1	In[] = {'A', 'B', 'C', 'D', 'E', 'F'};
	iOff = 0;
	iLen = 6;
		QUJDREVG	QUJDREVG	Pass	iLen = Covers <59, 60>, <59, 61>, <59, 64>
					oLen = Covers <61, 62>
					op = Covers <65,74>, <74,75>, <75,76>,

					<75,77>, <77,78>, <77,79>, <79,74>

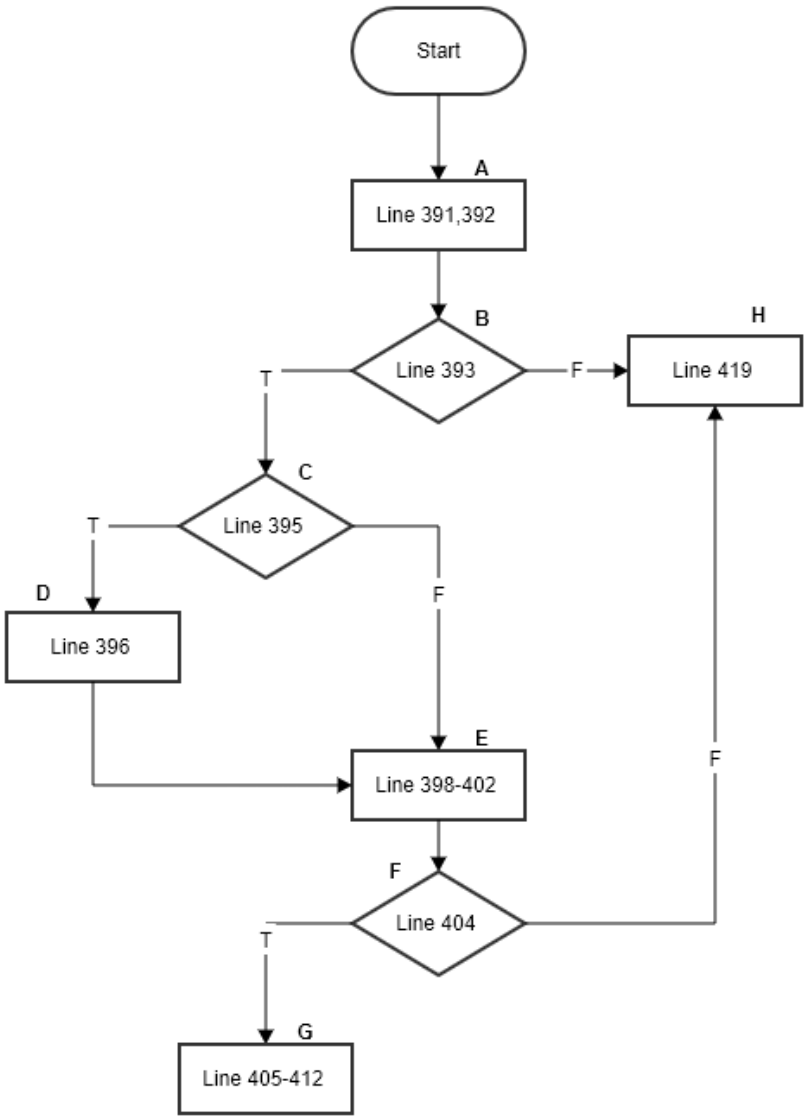
FUNCTION 2:

Source Code:

https://github.com/openjdk/jdk/tree/master/src/java.base/share/classes/java/time/Duration.java

```
390     public static Duration parse(CharSequence text) {
391         Objects.requireNonNull(text, "text");
392         Matcher matcher = Lazy.PATTERN.matcher(text);
393         if (matcher.matches()) {
394             // check for letter T but no time sections
395             if (!charMatch(text, matcher.start(3), matcher.end(3), 'T')) {
396                 boolean negate = charMatch(text, matcher.start(1), matcher.end(1), '-');
397 
398                 int dayStart = matcher.start(2), dayEnd = matcher.end(2);
399                 int hourStart = matcher.start(4), hourEnd = matcher.end(4);
400                 int minuteStart = matcher.start(5), minuteEnd = matcher.end(5);
401                 int secondStart = matcher.start(6), secondEnd = matcher.end(6);
402                 int fractionStart = matcher.start(7), fractionEnd = matcher.end(7);
403 
404                 if (dayStart >= 0 || hourStart >= 0 || minuteStart >= 0 || secondStart >= 0) {
405                     long daysAsSecs = parseNumber(text, dayStart, dayEnd, SECONDS_PER_DAY, "days");
406                     long hoursAsSecs = parseNumber(text, hourStart, hourEnd, SECONDS_PER_HOUR, "hours");
407                     long minsAsSecs = parseNumber(text, minuteStart, minuteEnd, SECONDS_PER_MINUTE, "minutes");
408                     long seconds = parseNumber(text, secondStart, secondEnd, 1, "seconds");
409                     boolean negativeSecs = secondStart >= 0 && text.charAt(secondStart) == '-';
410                     int nanos = parseFraction(text, fractionStart, fractionEnd, negativeSecs ? -1 : 1);
411                     try {
412                         return create(negate, daysAsSecs, hoursAsSecs, minsAsSecs, seconds, nanos);
413                     } catch (ArithmeticException ex) {
414                         throw (DateTimeParseException) new DateTimeParseException("Text cannot be parsed to a Duration: overflow", text, 0).initCause(ex);
415                     }
416                 }
417             }
418         }
419         throw new DateTimeParseException("Text cannot be parsed to a Duration", text, 0);
420     }
```

CFG:



Statement Coverage:

Line 414 exception case is not covered under sir's guidance.

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	text = "PT6H"	"6 hours"	"6 hours"	Pass	Covers statements from 391 to 395, 398 to 412
2	text = "G3D"	"Exception"	"Exception"	Pass	Covers statement 419

3	text = "-P2D"	"-2 days"	"-2 days"	Pass	Covers statement 396

Branch Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	text = "PT6H"	"6 hours"	"6 hours"	Pass	Covers B393T, B395F, B404T
2	text = "G3D"	Exception	Exception	Pass	Covers B393F
3	text= "-PT6H3M"	"-6 Hours and -3 minutes"	"-6 Hours and -3 minutes"	Pass	Covers B393T, B395T
4	text= "PTDHM"	Exception	Exception	Pass	Covers B404F

Condition Coverage with Short Circuit Evaluation:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	text = "PT6H"	"6 hours"	"6 hours"	Pass	Covers C393T, C395F, C404-1T
2	text = "G3D"	Exception	Exception	Pass	Covers C393F
3	text= "PT-6D6H"	"-6 Days and 6 Hours"	"-6 Days and 6 Hours"	Pass	Covers C393T, C395T, C404-1F, C404-2T
4	text= "PT-6D-6H6M"	"-6 Days and -6 Hours and 6 minutes"	"-6 Days and -6 Hours and 6 minutes"	Pass	Covers C393T, C395T, C404-1F, C404-2F, C404-3T

5	text= "PT-6D-6H-6M6S"	"-6 Days and -6 Hours and -6 minutes and 6 seconds"	"-6 Days and -6 Hours and -6 minutes and 6 seconds"	Pass	Covers C393T, C395T, C404-1F, C404-2F, C404-3F, C404-4T
6	text= "PT-6D-6H-6M-6S"	Exception	Exception	Pass	Covers C393T, C395T, C404-1F, C404-2F, C404-3F, C404-4F

Boundary Interior:

Boundary Interior Technique cannot be applied to this function because it does not contain any loop.

Loop Boundary:

Loop Boundary Technique cannot be applied to this function because it does not contain any loop.

Basis Path:

No. of Basis Paths = No. of decision points + 1

No. of Basis Paths = 3 + 1 = 4

Path 1: ABCDEFG

Path 2: ABH

Path 3: ABCEFG

Path 4: ABCEFH

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	text = "PT-6H3M"	"6 Hours and -3 minutes"	"6 Hours and -3 minutes"	Pass	Covers path ABCDEFG

2	text = "G3D"	"Exception"	"Exception"	Pass	Covers path ABH
3	text = "PT6H"	"6 hours"	"6 hours"	Pass	Covers ABCEFG
4	text= "PTDHM"	Exception	Exception	Pass	Covers ABCEFH

Data Flow Testing:

Variable #	Variable Name	Definitions	Uses
1	matcher	392	393, 395, 396, 398, 399, 400, 401, 402
2	dayStart	398	404, 405
3	hourStart	399	404, 406

Variable #	Variable Name	DU pairs
1	Matcher	<392, 393> <392, 395> <392, 396> <392, 398> <392, 399> <392, 400> <392, 401> <392, 402>
2	dayStart	<398, 404> <398,405>
3	hourStart	<399, 404> <399,406>

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	text ="-PT2D6H4M20.345S"	"-2 days and -6 Hours and -4 minutes and -20.345 seconds"	"-2 days and -6 Hours and -4 minutes and -20.345 seconds"	Pass	For matcher : Covers <392, 393> <392, 395> <392, 396> <392, 398> <392, 399> <392, 400> <392, 401>

					<392, 402> For dayStart: Covers <398, 404> <398, 405> For hourStar: Covers <398, 404> <398, 406>

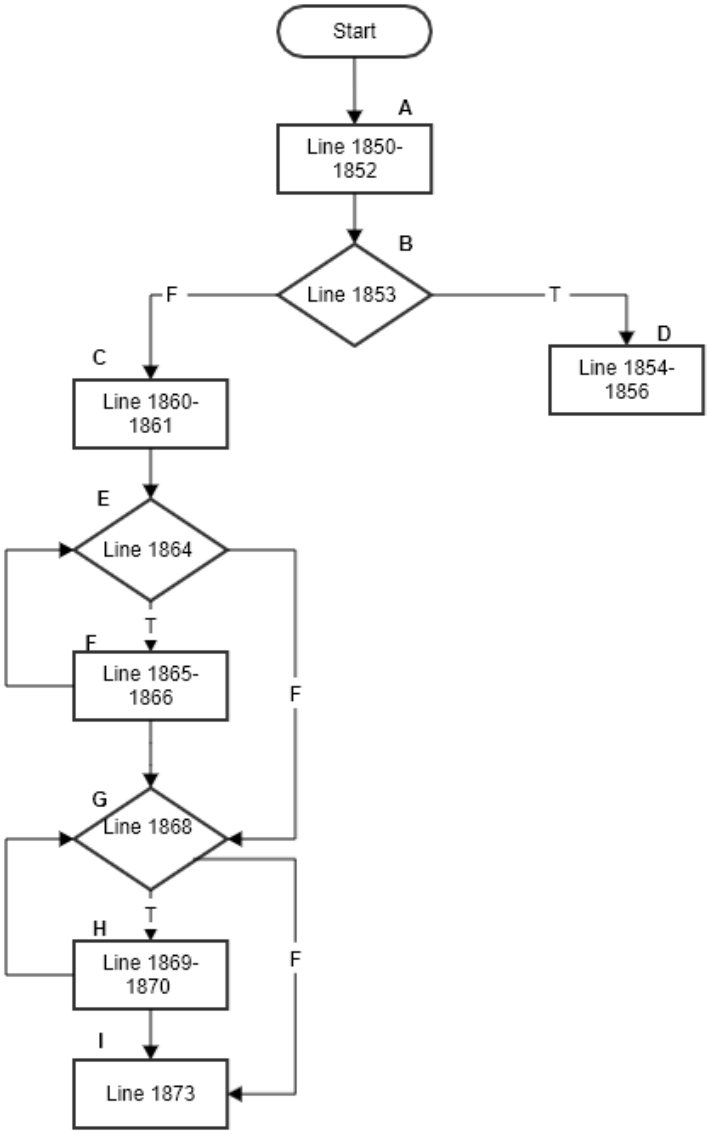
FUNCTION 3:

Source Code:

https://github.com/openjdk/jdk/tree/master/src/java.base/share/classes/java/math/MutableBigInteger.java

```
1848    static final long LONG_MASK = 0xffffffffL;
1849    static long divWord(long n, int d) {
1850        long dLong = d & LONG_MASK;
1851        long r;
1852        long q;
1853        if (dLong == 1) {
1854            q = (int)n;
1855            r = 0;
1856            return (r << 32) | (q & LONG_MASK);
1857        }
1858
1859        // Approximate the quotient and remainder
1860        q = (n >>> 1) / (dLong >>> 1);
1861        r = n - q*dLong;
1862
1863        // Correct the approximation
1864        while (r < 0) {
1865            r += dLong;
1866            q--;
1867        }
1868        while (r >= dLong) {
1869            r -= dLong;
1870            q++;
1871        }
1872        // n - q*dlong == r && 0 <= r <dLong, hence we're done.
1873        return (r << 32) | (q & LONG_MASK);
1874    }
1875
```

CFG:



Statement Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	n = 16 d = 1	16	16	Pass	Covers Statement 1850-1857

2	n = 10 d = 3	4294967299	4294967299	Pass	Covers Statement 1850,1851,1852, 1860-1868, 1873
3	-	-	-	-	Statement 1869- 1870 I think this is a dead code, I could not find any such case in which the condition at 1868 becomes True

Branch Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	n = 16 d = 1	16	16	Pass	Covers B1853T
2	n = 10 d = 3	4294967299	4294967299	Pass	Covers B1853F , B1864TF, B1864F
3	-	-	-	-	Statement 1869- 1870 I think this is a dead code, I could not find any such case in which the condition at 1868 becomes True

Condition Coverage with Short Circuit Evaluation:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	n = 16 d = 1	16	16	Pass	Covers C1853T
2	n = 10 d = 3	4294967299	4294967299	Pass	Covers C1853F , C1864TF, C1864F

3	-	-	-	-	Statement 1869- 1870 I think this is a dead code, I could not find any such case in which the condition at 1868 becomes True

Boundary Interior:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	n = 10 d = 3	4294967299	4294967299	Pass	Covers loop starting at Line 1864. This while loop has only one path.
2	-	-	-	-	Statement 1869- 1870 I think this is a dead code, I could not find any such case in which the condition at 1868 becomes True.

Loop Boundary:

I think Loop at line 1868 is a dead code, I could not find any such case in which the condition at 1868 becomes True.

Test cases are only for the loop at line 1864.

I choose loop upper bound = 5

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	n =10 d = 5	2	2	Pass	Loop at line 1864 is skipped entirely.

2	n =5 d = 3	8589934593	8589934593	Pass	Loop at line 1864 is run only once
3	n = 14 d = 6	8589934596	8589934596	Pass	Loop at line 1864 is run 3 times.
4	n =20 d =3	8589934598	8589934598	Pass	Loop at line 1864 is run 4 times
5	n = 28 d = 3	4294967305	4294967305	Pass	Loop at line 1864 is run 5 times.
6	n = 32 d = 3	8589934602	8589934602	Pass	Loop at line 1864 is run 6 times.

Basis Path:

No. of Basis Paths = No. of decision points + 1

No. of Basis Paths = 3 + 1 = 4

Path 1: ABD

Path 2: ABCEFGHI

Path 3: ABCEFGI

Path 4: ABCEGI

Test case#	**Input**	**Output**	**Expected Output**	**Pass/Fail**	**Comments/Remarks**
1	n = 16 d = 1	16	16	Pass	Covers path ABD
2	-	-	-	-	Path ABCEFGHI cannot be covered since the condition in the G block is never True so H block cannot be executed.
3	n =5 d = 3	8589934593	8589934593	Pass	Covers path ABCEFGI

4	n = 10 d = 2	5	5	Pass	Covers path ABCEGI

Data Flow Testing:

Variable #	Variable Name	Definitions	Uses
1	dLong	1850	1853, 1860, 1861, 1865, 1868, 1869
2	n	1849	1854, 1860, 1861
3	q	1854, 1860, 1866, 1870	1856, 1861, 1866, 1870, 1873

Variable #	Variable Name	DU pairs
1	dLong	<1850,1853> <1850,1860> <1850,1861> <1850,1865> <1850,1868> <1850,1869>
2	n	<1849,1854> <1849,1860> <1850,1861>
3	q	<1854, 1856> <1860, 1861> <1860, 1866> <1860, 1870> <1860, 1873> <1866, 1866> <1866, 1870> <1866, 1873> <1870, 1870> <1870, 1873>

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	n = 28 d = 3	4294967305	4294967305	Pass	For dLong covers: <1850,1853> <1850,1860> <1850,1861> <1850,1865> <1850,1868> For n covers: <1849,1860> <1850,1861> For q covers: <1860, 1861>

					<1860,1866> <1866, 1866> <1866, 1873>
2	n = 10 d = 1	10	10	Pass	For dLong covers: <1850,1853> For n covers: <1849,1854> For q covers: <1854,1856>
3	n = 10 d = 2	5	5	Pass	For dLong covers: <1850,1853> <1850,1860> <1850,1861> For n covers: <1849,1860> <1849,1861> For q covers: <1860, 1873>
-	-	-	-	-	For q these DU pairs cannot be covered: <1870, 1870> <1870, 1873> <1866, 1870> <1866, 1873>

FUNCTION 4:

Decodes a byte array from Base64 format.

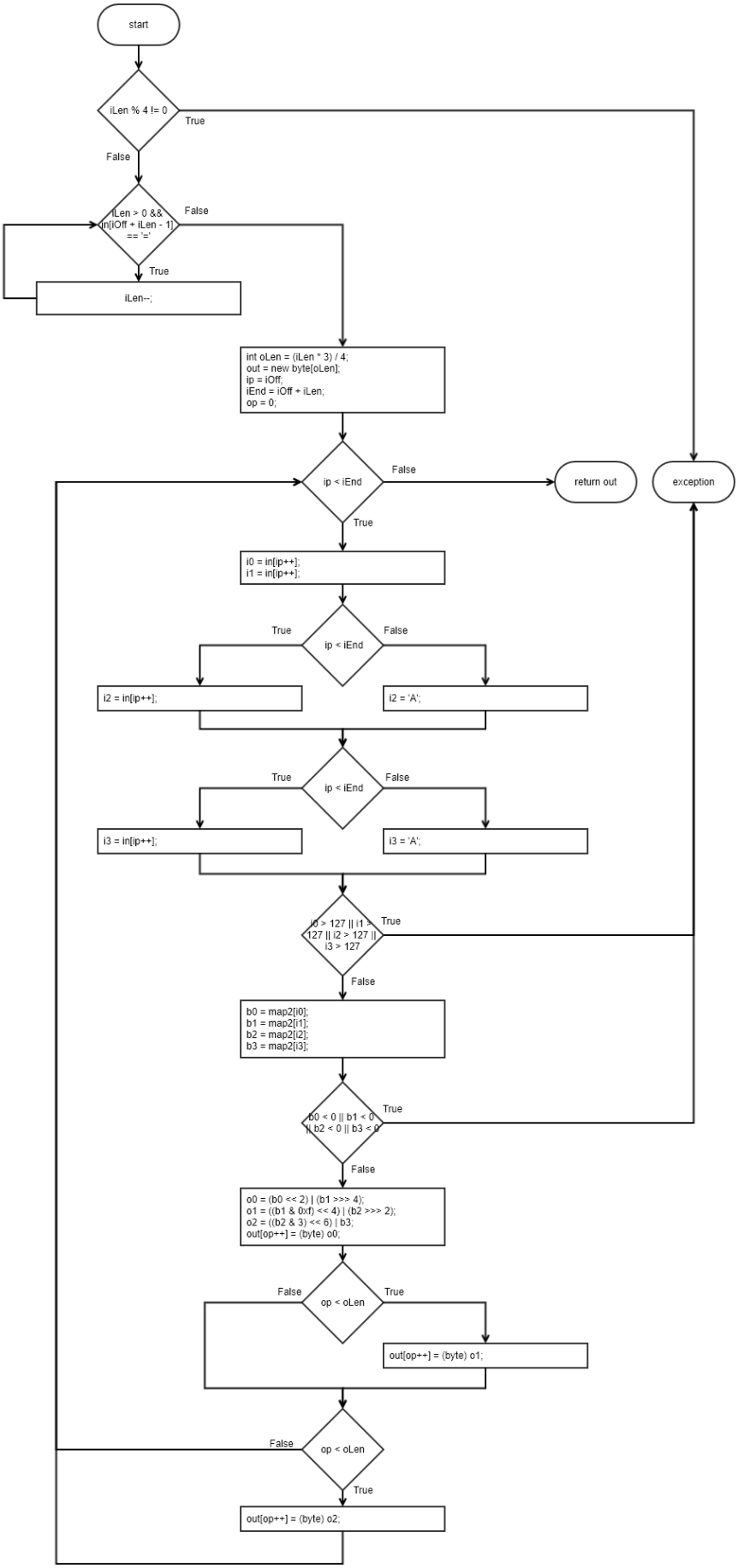
Note: map2[] table is populated in another constructor function.

Source Code:

timesheet-master\src\main\java\timeSheet\util\properties\Base64Coder.java

```
106     public byte[] decode(char[] in, int iOff, int iLen) {
107         if (iLen % 4 != 0)
108             throw new IllegalArgumentException("Length of Base64 encoded input string is not a multiple of 4.");
109         while (iLen > 0 && in[iOff + iLen - 1] == '=') iLen--;
110         int oLen = (iLen * 3) / 4;
111         byte[] out = new byte[oLen];
112         int ip = iOff;
113         int iEnd = iOff + iLen;
114         int op = 0;
115         while (ip < iEnd) {
116             int i0 = in[ip++];
117             int i1 = in[ip++];
118             int i2 = ip < iEnd ? in[ip++] : 'A';
119             int i3 = ip < iEnd ? in[ip++] : 'A';
120             if (i0 > 127 || i1 > 127 || i2 > 127 || i3 > 127)
121                 throw new IllegalArgumentException("Illegal character in Base64 encoded data.");
122             int b0 = map2[i0];
123             int b1 = map2[i1];
124             int b2 = map2[i2];
125             int b3 = map2[i3];
126             if (b0 < 0 || b1 < 0 || b2 < 0 || b3 < 0)
127                 throw new IllegalArgumentException("Illegal character in Base64 encoded data.");
128             int o0 = (b0 << 2) | (b1 >>> 4);
129             int o1 = ((b1 & 0xf) << 4) | (b2 >>> 2);
130             int o2 = ((b2 & 3) << 6) | b3;
131             out[op++] = (byte) o0;
132             if (op < oLen) out[op++] = (byte) o1;
133             if (op < oLen) out[op++] = (byte) o2;
134         }
135         return out;
136     }
137 }
```

CFG:


start
iLen % 4 != 0
True
False
iLen > 0 && in[iOff + iLen - 1] == '='
False
True
iLen--;
int oLen = (iLen * 3) / 4;
out = new byte[oLen];
ip = iOff;
iEnd = iOff + iLen;
op = 0;
ip < iEnd
False
return out
exception
True
i0 = in[ip++];
i1 = in[ip++];
ip < iEnd
True
False
i2 = in[ip++];
i2 = 'A';
ip < iEnd
True
False
i3 = in[ip++];
i3 = 'A';
i0 > 127 || i1 > 127 || i2 > 127 || i3 > 127
True
False
b0 = map2[i0];
b1 = map2[i1];
b2 = map2[i2];
b3 = map2[i3];
b0 < 0 || b1 < 0 || b2 < 0 || b3 < 0
True
False
o0 = (b0 << 2) | (b1 >>> 4);
o1 = ((b1 & 0xf) << 4) | (b2 >>> 2);
o2 = ((b2 & 3) << 6) | b3;
out[op++] = (byte) o0;
op < oLen
False
True
out[op++] = (byte) o1;
op < oLen
False
True
out[op++] = (byte) o2;

Statement Coverage:

Exception cases are not covered under sir's guidance.

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	In[] = 'QUJD' iOff = 0 iLen = 4	'ABC'	'ABC'	Pass	No padding
2	In[] = 'QQ== iOff = 0 iLen = 4	'A'	'A'	Pass	Padded with ==

Branch Coverage:

Exception cases are not covered under sir's guidance.

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	In[] = 'QUJD' iOff = 0 iLen = 4	'ABC'	'ABC'	Pass	109F, 115TF, 118T, 119T, 132T, 133T
2	In[] = 'QQ== iOff = 0 iLen = 4	'A'	'A'	Pass	109TF, 115TF, 118F, 119F, 132F, 133F

Condition Coverage with Short Circuit Evaluation:

Exception cases are not covered under sir's guidance.

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	In[] = 'QUJD' iOff = 0 iLen = 0	Empty String	Empty String	Pass	109aF, 115F
2	In[] = 'QUJD' iOff = 0	'ABC'	'ABC'	Pass	109aT, 109bF, 115TF, 118T, 119T, 132T, 133T

	iLen = 4				
3	In[] = 'QQ==' iOff = 0 iLen = 4	'A'	'A'	Pass	109aT, 109bTF, 115TF, 118F, 119F, 132F, 133F

Boundary Interior:

Exception cases are not covered under sir's guidance.
Possible logical paths:

- 118T, 119T, 132T, 133T
- 118T, 119F, 132T, 133F
- 118F, 119F, 132T, 133F

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	In[] = 'QUJD' iOff = 0 iLen = 4	'ABC'	'ABC'	Pass	Covers 118T, 119T, 132T, 133T
2	In[] = 'QQ==' iOff = 0 iLen = 4	'A'	'A'	Pass	Covers 118F, 119F, 132F, 133F
3	In[] = 'QUI=' iOff = 0 iLen = 4	'AB'	'AB'	Pass	Covers 118T, 119F, 132T, 133F

Loop Boundary:

Consider N=12 for loop. (Note that for valid input N-1 must be 8 and N+1 must be 16)

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	In[] = 'QUJD' iOff = 0 iLen = 0	Empty String	Empty String	Pass	Covers 115F
2	In[] = 'QUJD'	'ABC'	'ABC'	Pass	Covers 115F once

	iOff = 0 iLen = 4				
3	In[] = 'QUJDREU=' iOff = 0 iLen = 8	'ABCDE'	'ABCDE'	Pass	Covers 115T for N-1
4	In[] = 'QUJDREVGRw==' iOff = 0 iLen = 12	'ABCDEFG'	'ABCDEFG'	Pass	Covers 115T for N
5	In[] = 'QUJDREVGR0hJSg==' iOff = 0 iLen = 16	'ABCDEFGHIJ'	'ABCDEFGHIJ'	Pass	Covers 115T for N+1

Basis Path:

Path 1: 109F, 115F

Path 2: 109F, 115T, 118T, 119T, 132T, 133T

Path 3: 109T, 115F

Path 4: 109T, 115T, 118T, 119F, 132T, 133F

Path 5: 109T, 115T, 118F, 119F, 132F, 133F

Note that no logical path is possible to cause 119T while 118F. Same is case with 132F and 133T. Similarly, conditions in 132 and 133 also depend upon same factor as 118, 119 so it is not possible for 118T but 132F and vice versa. Furthermore, condition 109 also shares data dependency with 118, 119, 132, 133.

Test case#	**Input**	**Output**	**Expected Output**	**Pass/Fail**	**Comments/Remarks**
1	In[] = 'QUJD' iOff = 0 iLen = 0	Empty String	Empty String	Pass	Covers Path1
2	In[] = 'QUJD' iOff = 0 iLen = 4	'ABC'	'ABC'	Pass	Covers Path2

3	In[] = 'QQ==' iOff = 2 iLen = 4	Empty String	Empty String	Pass	Covers Path3
4	In[] = 'QQ==' iOff = 0 iLen = 4	'A'	'A'	Pass	Covers Path5
5	In[] = 'QUI=' iOff = 0 iLen = 4	'AB'	'AB'	Pass	Covers Path4

Data Flow Testing:

Exceptions cases not considered under sir's guidance

Variable #	Variable Name	Definitions	Uses
1	iLen	106, 109	109, 110, 113
2	oLen	110	111, 132, 133
3	op	114, 131, 132, 133	131, 132, 133

Variable #	Variable Name	DU pairs
1	iLen	<106, 109>, <109, 109>, <106, 113>, <109, 113>, <106, 110>, <109, 110>
2	oLen	<110, 111>, <110, 132>, <110, 133>
3	op	<114, 131>, <131, 132>, <131, 133>, <132, 133>

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	In[] = 'QUJD' iOff = 0 iLen = 4	'ABC'	'ABC'	Pass	iLen = Covers <106, 109>, <106, 110>, <106, 113> oLen = Covers <110, 111>, <110, 132>, <110, 133>

					op = Covers <114, 131>, <131, 132>, <132, 133>
2	In[] = 'QQ==' iOff = 0 iLen = 4	'A'	'A'	Pass	iLen = Covers <106, 109>, <106, 110>, <106, 113> oLen = Covers <110, 111>, <110, 132>, <110, 133> op = Covers <114, 131>, <131, 132>, <131, 133>

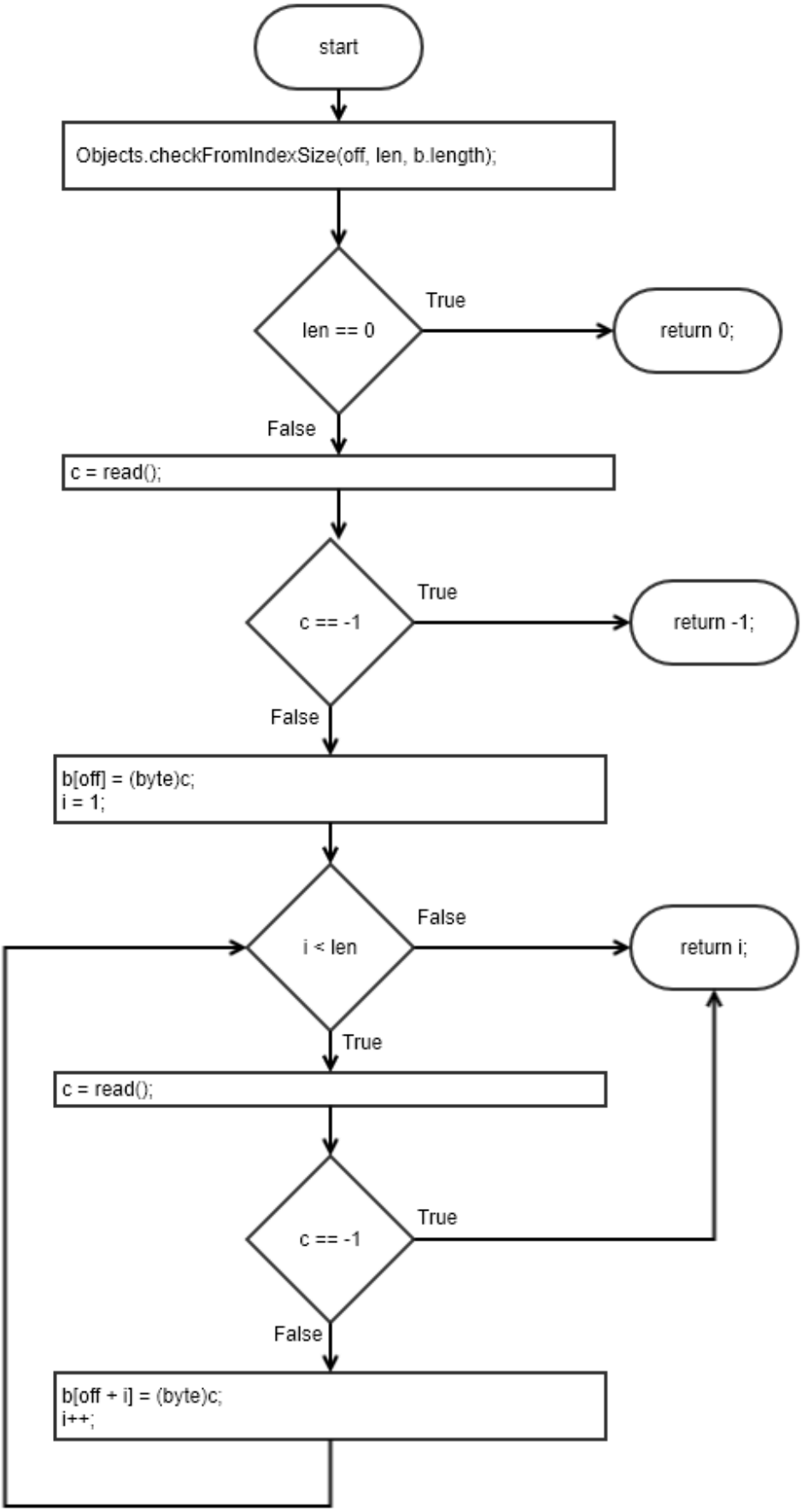
FUNCTION 5:

Source Code:

https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/io/InputStream.Java
checkFromIndexSize and read are external APIs. checkFromIndexSize can be implemented as dummy stub while read is implemented as needed by each test case.

```
278     public int read(byte b[], int off, int len) throws IOException {
279         Objects.checkFromIndexSize(off, len, b.length);
280         if (len == 0) {
281             return 0;
282         }
283 
284         int c = read();
285         if (c == -1) {
286             return -1;
287         }
288         b[off] = (byte)c;
289 
290         int i = 1;
291         try {
292             for (; i < len ; i++) {
293                 c = read();
294                 if (c == -1) {
295                     break;
296                 }
297                 b[off + i] = (byte)c;
298             }
299         } catch (IOException ee) {
300         }
301         return i;
302     }
```

CFG:


start
Objects.checkFromIndexSize(off, len, b.length);
len == 0
True
return 0;
False
c = read();
c == -1
True
return -1;
False
b[off] = (byte)c;
i = 1;
i < len
False
return i;
True
c = read();
c == -1
True
False
b[off + i] = (byte)c;
i++;

Statement Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	b[] = Empty Array off = 0 len = 3	3, b[] ='ABC'	3, b[] ='ABC'	Pass	External module API read() returns 'A', 'B', 'C' in consecutive calls.
2	b[] = Empty Array off = 0 len = 0	0, b[] = Empty Array	0, b[] = Empty Array	Pass	External module API read() is never called
3	b[] = Empty Array off = 0 len = 3	-1, b[] = Empty Array	-1, b[] = Empty Array	Pass	External module API read() returns -1 to notify an error at first call.
4	b[] = Empty Array off = 0 len = 3	1, b[] = 'A'	1, b[] = 'A'	Pass	External module API read() returns 'A', -1 in consecutive calls.

Branch Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	b[] = Empty Array off = 0 len = 3	3, b[] ='ABC'	3, b[] ='ABC'	Pass	External module API read() returns 'A', 'B', 'C' in consecutive calls. 280F, 285F, 292TF, 294F
2	b[] = Empty Array off = 0 len = 0	0, b[] = Empty Array	0, b[] = Empty Array	Pass	External module API read() is never called. 280T

3	b[] = Empty Array off = 0 len = 3	-1, b[] = Empty Array	-1, b[] = Empty Array	Pass	External module API read() returns -1 to notify an error at first call. 280F, 285T
4	b[] = Empty Array off = 0 len = 3	1, b[] = 'A'	1, b[] = 'A'	Pass	External module API read() returns 'A', -1 in consecutive calls. 280F, 285F, 292T, 294T

Condition Coverage with Short Circuit Evaluation:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	b[] = Empty Array off = 0 len = 3	3, b[] ='ABC'	3, b[] ='ABC'	Pass	External module API read() returns 'A', 'B', 'C' in consecutive calls. 280F, 285F, 292TF, 294F
2	b[] = Empty Array off = 0 len = 0	0, b[] = Empty Array	0, b[] = Empty Array	Pass	External module API read() is never called. 280T
3	b[] = Empty Array off = 0 len = 3	-1, b[] = Empty Array	-1, b[] = Empty Array	Pass	External module API read() returns -1 to notify an error at first call. 280F, 285T
4	b[] = Empty Array off = 0 len = 3	1, b[] = 'A'	1, b[] = 'A'	Pass	External module API read() returns 'A', -1 in consecutive calls. 280F, 285F, 292T, 294T

Boundary Interior:

Possible logical paths (depends upon successful or unsuccessful read, returned from stub function. Input does not effectively dictate the decision):

- 294T
- 294F

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	b[] = Empty Array off = 0 len = 3	3, b[] ='ABC'	3, b[] ='ABC'	Pass	External module API read() returns 'A', 'B', 'C' in consecutive calls. 294F
2	b[] = Empty Array off = 0 len = 3	1, b[] ='A'	1, b[] ='A'	Pass	External module API read() returns 'A', '-1' in consecutive calls. 294T

Loop Boundary:

Consider N=4 for loop boundary

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	b[] = Empty Array off = 0 len = 1	1, b[] ='A'	1, b[] ='A'	Pass	External module API read() returns 'A' in consecutive calls. Covers 292F
2	b[] = Empty Array off = 0 len = 2	2, b[] ='AB'	2, b[] ='AB'	Pass	External module API read() returns 'A', 'B' in consecutive calls. Covers 292T once
3	b[] = Empty Array off = 0 len = 4	4, b[] ='ABCD'	4, b[] ='ABCD'	Pass	External module API read() returns 'A', 'B', 'C', 'D' in consecutive calls.

					Covers 292T N-1 times
4	b[] = Empty Array off = 0 len = 2	4, b[] ='ABCDE'	4, b[] ='ABCDE'	Pass	External module API read() returns 'A', 'B', 'C', 'D', 'E' in consecutive calls. Covers 292T N times
5	b[] = Empty Array off = 0 len = 2	4, b[] ='ABCDEF'	4, b[] ='ABCDEF'	Pass	External module API read() returns 'A', 'B', 'C', 'D', 'E', 'F' in consecutive calls. Covers 292T N+1 times

Basis Path:

Path 1: 280T

Path 2: 280F, 285T

Path 3: 280F, 285F, 292F

Path 4: 280F, 285F, 292TF, 294F

Path 5: 280F, 285F, 292T, 294T

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	b[] = Empty Array off = 0 len = 3	3, b[] ='ABC'	3, b[] ='ABC'	Pass	External module API read() returns 'A', 'B', 'C' in consecutive calls. Covers Path4
2	b[] = Empty Array off = 0 len = 0	0, b[] = Empty Array	0, b[] = Empty Array	Pass	External module API read() is never called. Covers Path1
3	b[] = Empty Array off = 0	-1,	-1, b[] = Empty Array	Pass	External module API read() returns -1 to

	len = 3	b[] = Empty Array			notify an error at first call. Covers Path2
4	b[] = Empty Array off = 0 len = 3	1, b[] = 'A'	1, b[] = 'A'	Pass	External module API read() returns 'A', -1 in consecutive calls. Covers Path5
5	b[] = Empty Array off = 0 len = 1	1, b[] = 'A'	1, b[] = 'A'	Pass	External module API read() returns 'A' in consecutive calls. Covers Path3

Data Flow Testing:

Variable #	Variable Name	Definitions	Uses
1	i	290, 292	292, 297
2	c	284, 293	285, 288, 294, 297
3	len	278	279, 292

Variable #	Variable Name	DU pairs
1	i	<290,292>, <290,297>, <292, 292>, <292,297>
2	c	<284,285>, <284,288>, <293,294>, <293,297>
3	len	<278, 279>, <278,292>

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	b[] = Empty Array off = 0 len = 3	3, b[] ='ABC'	3, b[] ='ABC'	Pass	i = Covers <290,292>, <290,297>, <292, 292>, <292,297> c = Covers <284,285>, <284,288>, <293,294>, <293,297>

					len = Covers <278, 279>, <278,292>

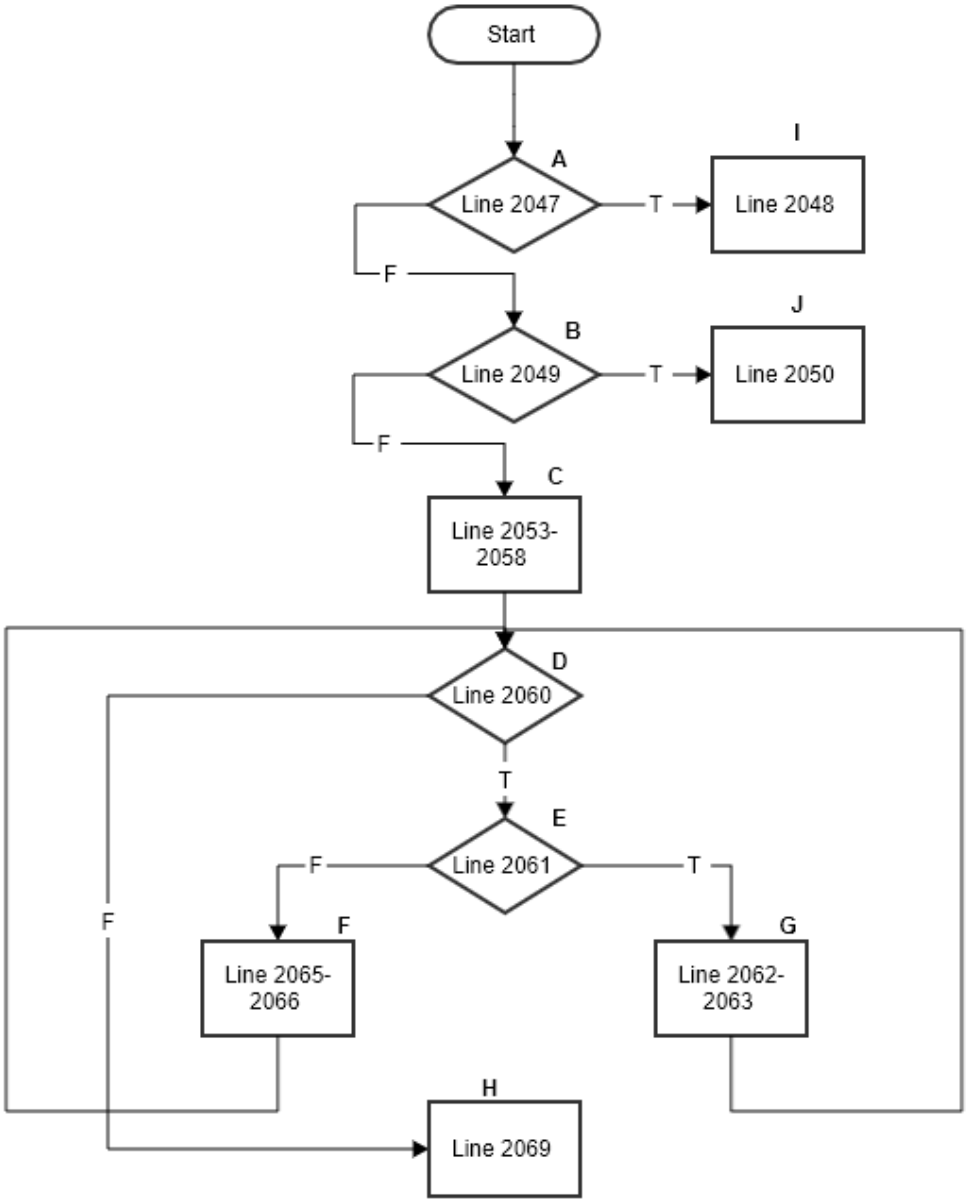
FUNCTION 6:

Source Code:

https://github.com/openjdk/jdk/tree/master/src/java.base/share/classes/java/math/MutableBigInteger.java

```
2046    static int binaryGcd(int a, int b) {
2047        if (b == 0)
2048            return a;
2049        if (a == 0)
2050            return b;
2051
2052        // Right shift a & b till their last bits equal to 1.
2053        int aZeros = Integer.numberOfTrailingZeros(a);
2054        int bZeros = Integer.numberOfTrailingZeros(b);
2055        a >>>= aZeros;
2056        b >>>= bZeros;
2057
2058        int t = (aZeros < bZeros ? aZeros : bZeros);
2059
2060        while (a != b) {
2061            if ((a+0x80000000) > (b+0x80000000)) {  // a > b as unsigned
2062                a -= b;
2063                a >>>= Integer.numberOfTrailingZeros(a);
2064            } else {
2065                b -= a;
2066                b >>>= Integer.numberOfTrailingZeros(b);
2067            }
2068        }
2069        return a<<t;
2070    }
```

CFG:



.

Statement Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	a = 15 b = 0	15	15	Pass	Covers statement 2047-2048

2	a = 0 b =15	15	15	Pass	Covers statement 2049-2050
3	a = 98 b =56	14	14	Pass	Covers statement 2047,2049, 2051-2069

Branch Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	a = 15 b = 0	15	15	Pass	Covers B2047T
2	a = 0 b =15	15	15	Pass	Covers B2049T, B2047F
3	a = 98 b =56	14	14	Pass	Covers B2047F, B2049F, B2060TF, B2061T
4	a = 56 b =98	14	14	Pass	Covers B2047F, B2049F, B2060TF, B2061F

Condition Coverage with Short Circuit Evaluation:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	a = 15 b = 0	15	15	Pass	Covers C2047T
2	a = 0 b =15	15	15	Pass	Covers C2049T, C2047F
3	a = 98 b =56	14	14	Pass	Covers C2047F, C2049F, C2060TF, C2061T
4	a = 56 b =98	14	14	Pass	Covers C2047F, C2049F, C2060TF, C2061F

Boundary Interior:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	a = 98 b =56	14	14	Pass	Covers boundary interior path DEG
2	a = 56 b =98	14	14	Pass	Covers boundary interior path DEF

Loop Boundary:

I choose loop upper bound = 5

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	a = 12 b = 12	12	12	Pass	Loop is skipped entirely.
2	a = 4 b = 2	2	2	Pass	Loop is run only once
3	a = 6 b = 2	2	2	Pass	Loop is run twice.
4	a = 10 b = 2	2	2	Pass	Loop is run 4 times
5	a = 12 b = 2	2	2	Pass	Loop is run 5 times.
6	a = 14 b = 2	2	2	Pass	Loop is run 6 times.

Basis Path:

No. of Basis Paths = No. of decision points + 1

No. of Basis Paths = 4 + 1 = 5

Path 1: AI

Path 2: ABJ

Path 3: ABCDH

Path 4: ABCDEFH

Path 5: ABCDEGH

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	a = 15 b = 0	15	15	Pass	Covers basis path AI
2	a = 0 b =15	15	15	Pass	Covers basis path ABJ
3	a = 12 b = 12	12	12	Pass	Covers basis path ABCDH
4	a = 2 b = 4	2	2	Pass	Covers basis path ABCDEFH
5	a = 4 b = 2	2	2	Pass	Covers basis path ABCDEFH

Data Flow Testing:

Variable #	Variable Name	Definitions	Uses
1	A	2046, 2055, 2062, 2063	2048, 2049, 2053, 2055, 2060, 2061, 2062, 2063, 2065, 2069
2	b	2046, 2056, 2065, 2066	2047, 2050, 2054, 2056, 2060, 2061, 2062, 2065, 2066
3	aZeros	2053	2055, 2058

Variable #	Variable Name	DU pairs
1	a	<2046, 2048> <2046, 2049> <2046, 2053> <2046, 2055> <2055, 2060> <2055, 2061> <2055, 2062> <2055, 2065> <2055, 2069> <2062, 2063> <2063, 2060> <2063, 2061>

		<2063, 2062> <2063, 2069>
2	b	<2046, 2047> <2046, 2050> <2046, 2054> <2046, 2056> <2056, 2060> <2056, 2061> <2056, 2062> <2056, 2065> <2065, 2066> <2066, 2060> <2066, 2061> <2066, 2062>
3	aZeros	<2053, 2055> <2053, 2058>

Test case#	**Input**	**Output**	**Expected Output**	**Pass/Fail**	**Comments/Remarks**
1	a = 15 b = 0	15	15	Pass	For a covers <2046, 2048> For b covers <2046, 2047>
2	a = 0 b =15	15	15	Pass	For a covers <2046, 2049> For b covers <2046, 2047> <2046, 2050>
3	a = 12 b = 12	12	12	Pass	For a covers: <2046, 2049> <2046, 2055> <2055, 2060> <2055, 2069> For b covers: <2046, 2047> <2046, 2056> <2056, 2060> For aZeros covers: <2053, 2055> <2053, 2058>

4	a = 98 b =56	14	14	Pass	For a covers <2046, 2049> <2046, 2053> <2046, 2055> <2055, 2060> <2055, 2061> <2055, 2062> <2062, 2063> <2063, 2060> <2063, 2061> <2063, 2062> <2063, 2069> For b covers <2046, 2047> <2046, 2054> <2046, 2056> <2056, 2060> <2056, 2061> <2056, 2062> For aZeros covers: <2053, 2055> <2053, 2058>
5	a = 56 b =98	14	14	Pass	For a covers <2046, 2049> <2046, 2053> <2046, 2055> <2055, 2060> <2055, 2061> <2055, 2065> <2055, 2069> For b covers <2046, 2047> <2046, 2054> <2046, 2056> <2056, 2060> <2056, 2061> <2056, 2065> <2065, 2066> <2066, 2060> <2066, 2061> <2066, 2062>

					For aZeros covers: <2053, 2055> <2053, 2058>

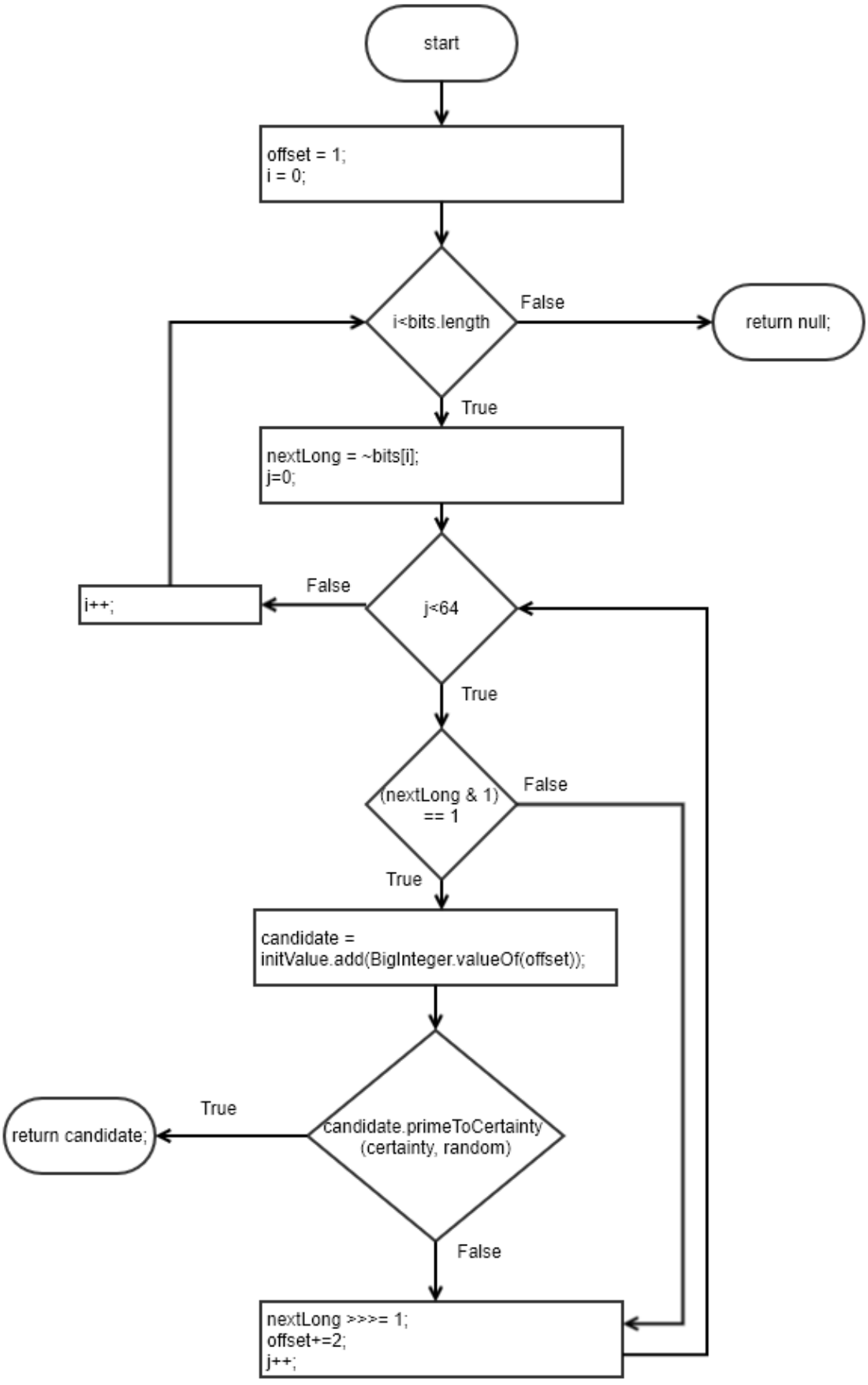
FUNCTION 7:

Source Code:

https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/math/BitSieve.java
bits are sieve bits where each bit represents a candidate odd integer. primeToCertainty is an external function which returns true if it is a prime with given probability.

```
194     BigInteger retrieve(BigInteger initValue, int certainty, java.util.Random random) {
195         // Examine the sieve one long at a time to find possible primes
196         int offset = 1;
197         for (int i=0; i<bits.length; i++) {
198             long nextLong = ~bits[i];
199             for (int j=0; j<64; j++) {
200                 if ((nextLong & 1) == 1) {
201                     BigInteger candidate = initValue.add(
202                                             BigInteger.valueOf(offset));
203                     if (candidate.primeToCertainty(certainty, random))
204                         return candidate;
205                 }
206                 nextLong >>>= 1;
207                 offset+=2;
208             }
209         }
210         return null;
211     }
```

CFG:


start
offset = 1;
i = 0;
i<bits.length
False
return null;
True
nextLong = ~bits[i];
j=0;
j<64
False
i++;
True
(nextLong & 1)
== 1
False
True
candidate =
initValue.add(BigInteger.valueOf(offset));
candidate.primeToCertainty
(certainty, random)
True
return candidate;
False
nextLong >>>= 1;
offset+=2;
j++;

Statement Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks

1	initValue = 0;
	certainity = 100;
	random = 10
	bits[] = b'11111010'
		257	257	Pass	Stub primeToCertainty shall return 'False, True' in consecutive calls.

2	initValue = 0;
	certainity = 100;
	random = 10
	bits[] = b'11111111'
		null	Null	Pass	Stub primeToCertainty shall never be called.

Branch Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks

1	initValue = 0;
	certainity = 100;
	random = 10
	bits[] = b'11111010'
		257	257	Pass	Stub primeToCertainty shall return 'False, True' in consecutive calls.
					197T, 199TF, 200TF, 203TF

2	initValue = 0;
	certainity = 100;
	random = 10
	bits[] = b'11111111'
		null	Null	Pass	Stub primeToCertainty shall never be called.
					197TF, 199TF, 200F

Condition Coverage with Short Circuit Evaluation:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks

1	initValue = 0;
	certainity = 100;
	random = 10
	bits[] = b'11111010'
		257	257	Pass	Stub primeToCertainty shall return 'False, True' in consecutive calls.
					197T, 199TF, 200TF, 203TF

2	initValue = 0;
	certainity = 100;
	random = 10
	bits[] = b'11111111'
		null	Null	Pass	Stub primeToCertainty shall never be called.
					197TF, 199TF, 200F

Boundary Interior:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1					
2					

Loop Boundary:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1					
2					

Basis Path:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks

1					
2					

Data Flow Testing:

Variable #	Variable Name	Definitions	Uses
1			
2			
3			

Variable #	Variable Name	DU pairs
1		
2		
3		

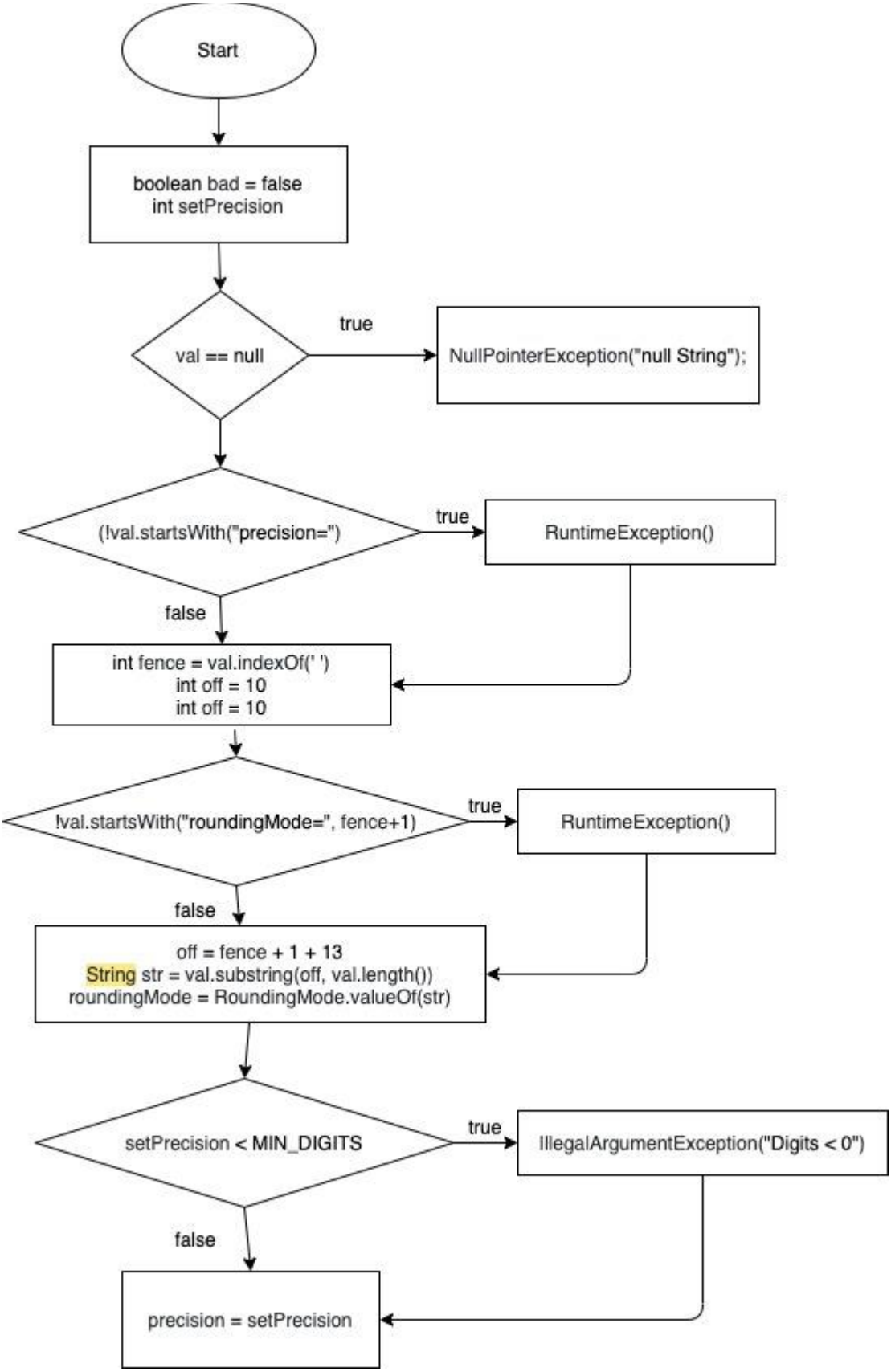
Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1					
2					

FUNCTION 8:

Source Code:

```
183     public MathContext(String val) {
184         boolean bad = false;
185         int setPrecision;
186         if (val == null)
187             throw new NullPointerException("null String");
188         try { // any error here is a string format problem
189             if (!val.startsWith("precision=")) throw new RuntimeException();
190             int fence = val.indexOf(' ');    // could be -1
191             int off = 10;                     // where value starts
192             setPrecision = Integer.parseInt(val.substring(10, fence));
193 
194             if (!val.startsWith("roundingMode=", fence+1))
195                 throw new RuntimeException();
196             off = fence + 1 + 13;
197             String str = val.substring(off, val.length());
198             roundingMode = RoundingMode.valueOf(str);
199         } catch (RuntimeException re) {
200             throw new IllegalArgumentException("bad string format");
201         }
202 
203         if (setPrecision < MIN_DIGITS)
204             throw new IllegalArgumentException("Digits < 0");
205         // the other parameters cannot be invalid if we got here
206         precision = setPrecision;
207     }
208 
```

CFG:


Start
boolean bad = false
int setPrecision
val == null
true
NullPointerException("null String");
(!val.startsWith("precision=")
true
RuntimeException()
false
int fence = val.indexOf(' ')
int off = 10
int off = 10
!val.startsWith("roundingMode=", fence+1)
true
RuntimeException()
false
off = fence + 1 + 13
String str = val.substring(off, val.length())
roundingMode = RoundingMode.valueOf(str)
setPrecision < MIN_DIGITS
true
IllegalArgumentException("Digits < 0")
false
precision = setPrecision

Statement Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	null	exception	exception	Pass	Covered 184, 185, 186, 187
2	'ThisString'	exception	exception	Pass	Covered 184, 185, 186, 188, 189
3	'precision =12 12'	exception	exception	Pass	Covered 184, 185, 186, 188, 190, 191, 192, 194, 195
4	roundin gMode =12 12'	exception	exception	Pass	Covered 184, 185, 186, 188, 189

Branch Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail Comments/Remarks
1	(null)	exception	exception	Pass Covered B186(True)
2	'ThisString'	exception	exception	Pass Covered B186(False), B189(True)
3	'precision=12 12'	exception	exception	Pass Covered B186(False), B189(False), B194(True)

4	'roundingMode =12 12'	exception	exception	Pass Covered B186(False), B189(True)

Condition Coverage with Short Circuit Evaluation:

Test case#	Input	Output	Expected Output	Pass/Fail Comments/Remarks
1	(null)	exception	exception	Pass Covered C186(True)
2	'ThisString'	exception	exception	Pass Covered C186(False), C189(True)
3	'precision=12 12'	exception	exception	Pass Covered C186(False), C189(False), C194(True)
4	'roundingMode =12 12'	exception	exception	Pass Covered C186(False), C189(True)

Boundary Interior:

No Loop in the program.

Loop Boundary:

No Loop in the program.

Basis Path:

Path 1:

183, 184, 185, 186, 203, 206

Path 2:

183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 196, 197, 198, 203, 206

Path 3:

183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 200, 203, 206

Path 4:

183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 200, 203, 204, 206

Path 5:

183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 200, 203, 204, 206

Data Flow Testing:

Variable #	Variable Name	Definitions	Uses
1	Val	183	186,189,190,192,197
2	setPrecision	185,192	203,206
3	Fence	190	192,194

Variable #	Variable Name	DU pairs
1	Val	<183,186>,<183,189>,<183,190>,<183,192>,<183,197>
2	setPrecision	<192,203>,<192,206>
3	Fence	<190,192>,<190,194>

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	'abcdef'	exception	exception	Pass	because it does not contains 'precision=' at start
2	'precision=12 12'	exception	exception	Pass	It returns exception because when next if executes it'll not find 'roundingMode=' at start

FUNCTION 9

Source Code:

```
1101      private boolean passesMillerRabin(int iterations, Random rnd) {
1102          // Find a and m such that m is odd and this == 1 + 2**a * m
1103          BigInteger thisMinusOne = this.subtract(ONE);
1104          BigInteger m = thisMinusOne;
1105          int a = m.getLowestSetBit();
1106          m = m.shiftRight(a);
1107
1108          // Do the tests
1109          if (rnd == null) {
1110              rnd = ThreadLocalRandom.current();
1111          }
1112          for (int i=0; i < iterations; i++) {
1113              // Generate a uniform random on (1, this)
1114              BigInteger b;
1115              do {
1116                  b = new BigInteger(this.bitLength(), rnd);
1117              } while (b.compareTo(ONE) <= 0 || b.compareTo(this) >= 0);
1118
1119              int j = 0;
1120              BigInteger z = b.modPow(m, this);
1121              while (!((j == 0 && z.equals(ONE)) || z.equals(thisMinusOne))) {
1122                  if (j > 0 && z.equals(ONE) || ++j == a)
1123                      return false;
1124                  z = z.modPow(TWO, this);
1125              }
1126          }
1127          return true;
1128      }
```

CFG:


Start
BigInteger thisMinusOne = this.subtract(ONE)
BigInteger m = thisMinusOne
int a = m.getLowestSetBit()
m = m.shiftRight(a)
rnd == null
true
rnd = ThreadLocalRandom.current()
false
i < iterations
true
BigInteger b
b = new BigInteger(this.bitLength(), rnd)
b.compareTo(ONE) <= 0 ||
b.compareTo(this) >= 0
true
false
j = 0
BigInteger z = b.modPow(m, this)
!((j == 0 && z.equals(ONE)) ||
z.equals(thisMinusOne))
false
true
j > 0 && z.equals(ONE)
|| ++j == a
true
false
z = z.modPow(TWO,
this)
false
return true
return false

Statement Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	(4, null)	true	true	Pass covers 1103,1104,1105,,1106,1109,1110,1111,1112,1113,1114-1128	returns true if one parameter is null
2	(0,4)	true	true	Pass covers 1103,1104,1105,,1106,1109,1112,1127	valid input and function will return true
3	(null,null)	error: bad operand	true	Fail covers 1103-1111,1112	Function crashed when both values in function are null
4	(7,9)	false	false	Pass Covered 1103-1111,1112-1123	Valid value but returns false

Branch Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	(4, null)	true	true	Pass	covers B1109(T), B1112(T), B1117(T), B1121(T)
2	(0, 4)	true	true	Pass	covers B1109(F), B1112(F)

3	(null, null)	Error: bad operand			
4	(7,9)	false	false	Pass	covers B1109(T) B1112(T), B1117(T), B1121(T), B1122(T)

Condition Coverage with Short Circuit Evaluation:

Test case#	**Input**	**Output**	**Expected Output**	**Pass/Fail**	**Comments/Remarks**
1	(4, null)	true	true	Pass	covers C1109(T), C1112(T), C1117(T), C1121(T)
2	(0, 4)	true	true	Pass	covers C1109(F), C1112(F)

3	(null, null)	error: bad operand types for binary operator '<'	no output	Fail	covers C1109(T), C1112(Crash)
4	(7,9)	false	false	Pass	covers C1109(T), C1112(T), C1117(T), C1121(T), C1122(T)

Boundary Interior:

Below we are taking line numbers to execute boundary interior.

1112 -> 1114

1112 -> 1114 -> 1115

1112 -> 1114 -> 1116 -> 1117

1112 -> 1114 -> 1116 -> 1117 -> 1116

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121 -> 1122

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121 -> 1122 -> 1123

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121 -> 1122 -> 1124

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121 -> 1122 -> 1124 -> 1121

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121 -> 1122 -> 1124 -> 1121 -> 1127

Loop Boundary:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	(0,2)	true	True	Pass	Covers 1109T When the loop will not execute

2	(1,2)	true	True	Pass	Covers 1112T once
3	(5,2)	false	False		Covers 1112T more than one passes

Basis Path:

Path 1:

1101, 1103, 1104, 1105, 1106, 1127

Path 2:

1101, 1103, 1104, 1105, 1106, 1109, 1110, 1127

Path 3:

1101, 1103, 1104, 1105, 1106, 1109, 1110, 1112, 1113, 1114, 1115, 1116, 1117, 1119, 1120, 1127

Path 4:

1101, 1103, 1104, 1105, 1106, 1109, 1110, 1112, 1113, 1114, 1115, 1116, 1117, 1119, 1120, 1121, 1122, 1123, 1124, 1127

Data Flow Testing:

Variable #	Variable Name	Definitions	Uses
1	iterations	1101	1112
2	Rnd	1101,1110	1109,1116
3	A	1105	1106

Variable #	Variable Name	DU pairs
1	iterations	<1101,1112>
2	Rnd	<1101,1109>,<1110,1116>
3	A	<1105,1106>

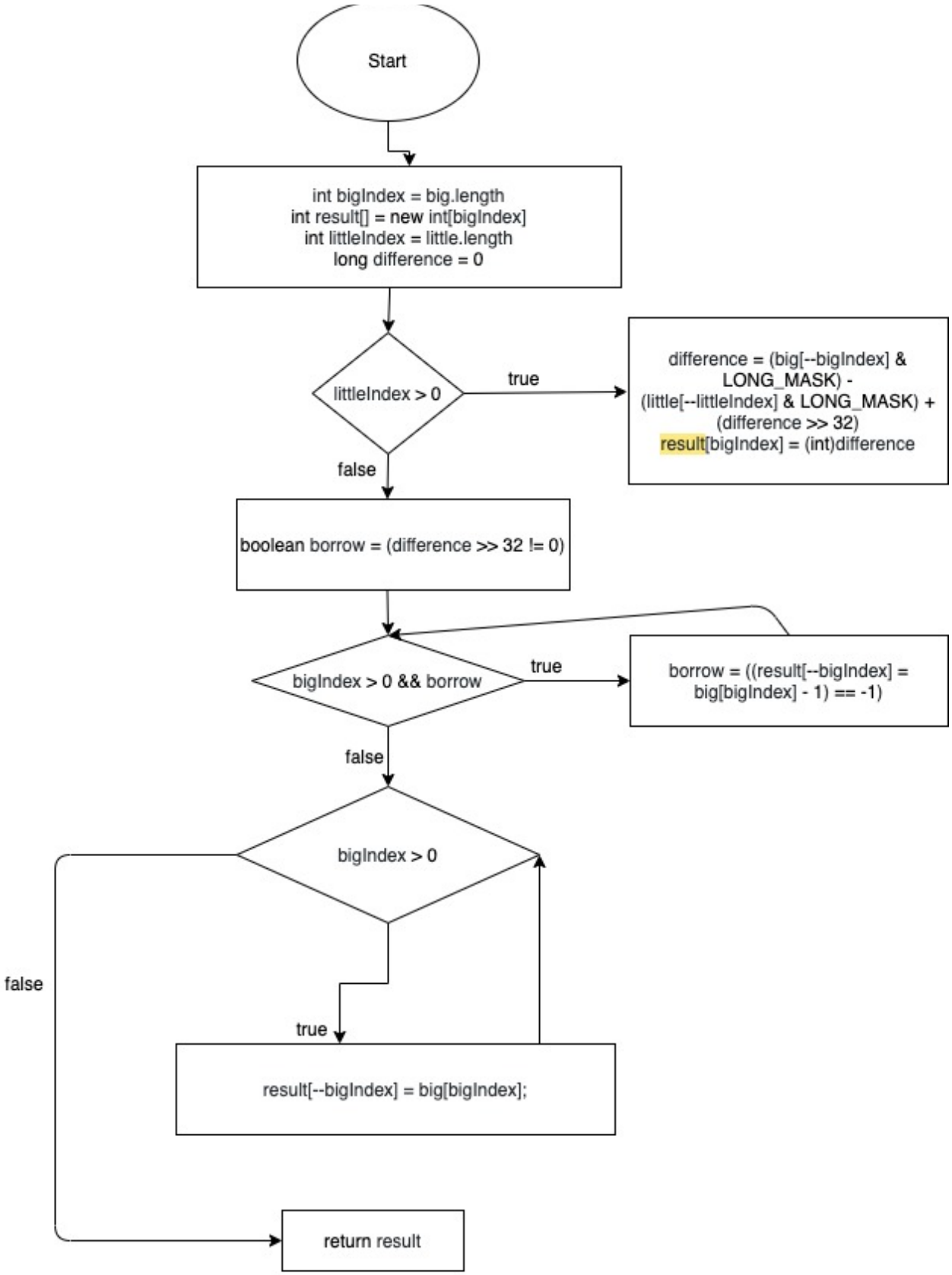
Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	(4, null)	true	true	Pass	It returns true second null value is handled in function
2	(7,9)	false	false	Pass	It returns the result false due to its values

FUNCTION 10:

Source Code:

```
1548        private static int[] subtract(int[] big, int[] little) {
1549            int bigIndex = big.length;
1550            int result[] = new int[bigIndex];
1551            int littleIndex = little.length;
1552            long difference = 0;
1553
1554            // Subtract common parts of both numbers
1555            while (littleIndex > 0) {
1556                difference = (big[--bigIndex] & LONG_MASK) -
1557                             (little[--littleIndex] & LONG_MASK) +
1558                             (difference >> 32);
1559                result[bigIndex] = (int)difference;
1560            }
1561
1562            // Subtract remainder of longer number while borrow propagates
1563            boolean borrow = (difference >> 32 != 0);
1564            while (bigIndex > 0 && borrow)
1565                borrow = ((result[--bigIndex] = big[bigIndex] - 1) == -1);
1566
1567            // Copy remainder of longer number
1568            while (bigIndex > 0)
1569                result[--bigIndex] = big[bigIndex];
1570
1571            return result;
1572        }
1573
```

CFG:


Start
int bigIndex = big.length
int result[] = new int[bigIndex]
int littleIndex = little.length
long difference = 0
littleIndex > 0
true
difference = (big[--bigIndex] &
LONG_MASK) -
(little[--littleIndex] & LONG_MASK) +
(difference >> 32)
result[bigIndex] = (int)difference
false
boolean borrow = (difference >> 32 != 0)
bigIndex > 0 && borrow
true
borrow = ((result[--bigIndex] =
big[bigIndex] - 1) == -1)
false
bigIndex > 0
false
true
result[--bigIndex] = big[bigIndex];
return result

Statement Coverage:

Test case #	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	x = {10,20} y = {30,40}	[-21,20]	[-21,20]	Pass	covers 1549, 1550, 1551, 1552, 1553, 1555, 1563,1564, 1565, 1568
2	x={10,20 } y = {}	[10,20]	[10,20]	Pass	covers 1549, 1550, 1551, 1552, 1553, 1555, 1563,1564, 1565, 1568, 1569
3	x = {} y = {30, 40}	Index -1 out of bounds for length 0	[30, 40]		2nd empty array case is not handled

Branch Coverage:

Test case #	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	x = {10, 20} y = {30, 40}	[-21,20]	[-21,20]	Pass	covers B1555T, B1564T, B1568T
2	x = {10,20} y = {}	[10,20]	[10,20]	Pass	covers B1555F, B1564T, B1568T

3	x = {} y = {30, 40}	bounds for length 0	[30, 40]	Fail	covers B1555F, B1564F, B1568F

Condition Coverage with Short Circuit Evaluation:

Test case #	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	x = {10,20}; y = {30,40}	[-21,20]	[-21,20]	Pass	covers C1555T, C1564T, C1568T
2	x={10,20} y = {}	[10,20]	[10,20]	Pass	covers C1555F, C1564T, C1568T
3	x ={} y = {30, 40}	Index -1 out of bounds for length 0	[30, 40]	Fail	covers C1555F, C1564F, C1568F

Boundary Interior:

Loop 1:

1555 -> 1556

1555 -> 1556 -> 1557

1555 -> 1556 -> 1557 -> 1558

1555 -> 1556 -> 1557 -> 1558 -> 1559

1555 -> 1556 -> 1557 -> 1558 -> 1559 -> 1555

Loop 2:

1564 -> 1565

1564 -> 1565 - 1564

Loop 3:

1568 -> 1569

1568 -> 1569 -> 1568

Loop Boundary:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	([0,2], [])	[0,2]	[0,2]	Pass	Covers: Loop 1: 1555T Loop 2: 1564T Loop 3: 1568T When the loop will not execute
2	([5],[2])	[2,4]	[2,4]	Pass	loop 1: 1555T loop 2: 1564T loop 3: 1568T Only one iteration
3	([10,20], [30,40])	[-21,20]	[-21,20]	Pass	loop 1: littleIndex > 0 True loop 2: bigIndex > 0 True loop 3: bigIndex > 0 True more than one passes

Basis Path:

Path 1:

1548, 1549, 1550, 1551, 1552, 1555, 1556, 1557, 1558, 1559, 1563, 1571

Path 2:

1548, 1549, 1550, 1551, 1552, 1555, 1556, 1557, 1558, 1559, 1563, 1564, 1565, 1571

Path 3:

1548, 1549, 1550, 1551, 1552, 1555, 1556, 1557, 1558, 1559, 1563, 1564, 1565, 1568, 1569, 1571

Path 4:

1548, 1549, 1550, 1551, 1552, 1555, 1556, 1557, 1558, 1559, 1563, 1568, 1569, 1571

Data Flow Testing:

Variable #	Variable Name	Definitions	Uses
1	big	1548	1549, 1556, 1565, 1569
2	little	1548	1551,1556
3	borrow	1563,1565	1564

Variable #	Variable Name	DU pairs
1	big	<1548,1549>,<1548,1549><1548,1556><1565,1569>
2	little	<1548,1551>,<1548,1556>
3	borrow	<1563,1564>

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	x = {10, 20} y = {30, 40}	[-21,20]	[-21,20]	Pass	It returns true second null value is handled in function
2	x={10,20} y = {}	[10,20]	[10,20]	Pass	It returns the result false due to its values

Project Contribution

Member	Submission 1	Submission 2	Submission 3
Danish	Setup and Run the web application, resolved all errors to run the project successfully	Chose func 1, 4, 5, 7 Wrote test cases for these functions	Wrote test cases for these func 1, 4 , 5
Abu Bakar	Documented the environment setup and prepared report for submission 1.	Chose func 2, 3, 6 Wrote test cases for these functions	Wrote test cases for these func 2, 3 , 6
Awais	Was not part of the group at that time.	Chose func 8, 9, 10 Wrote test cases for these functions Submitted late, and individually.	Wrote test cases for these func 8, 9, 10
Musa	No contribution	No contribution	No contribution

Note:

In the 2nd submission, we thought we would split function on behalf of the 4th member, but since we all have other assignments and office work too. So, we are not doing anything on behalf of 4th member.

In the 2nd submission, Danish wrote test cases for an additional function 7.
Abu Bakar and Awais could not do so because of time constraint.