MATHEMATICS FOR COMPUTING I

CSCI 1303

Assignment – Programming

Instructor: Takumi Sase

Semester 2, 2023/2024

Section 2

	Name	Matric no.
1	Syeda Samia	2123536
2	Nurul Hanis Fatini Binti Hairul Fadli	2310460
3	Nur Aien Sofia Binti Rusdi	2319006

Submission Date: Wednesday, 12th June 2024

Table of Contents

Objectives

1. To demonstrate knowledge of discrete mathematics.
2. To enhance programming skills.
3. To improve collaboration skills through teamwork.

Introduction

This report presents a comprehensive examination of key concepts in discrete mathematics through the implementation of a Java program designed to perform several intricate tasks. The primary focus is on calculating all congruences for a given set of integers (excluding modulus 1), identifying properties from relations and functions, and determining the presence of Euler paths and Euler circuits within graph structures. Additionally, the report includes manual calculations to verify and complement the program's outputs, ensuring the accuracy and reliability of the results. This dual approach of combining manual verification with automated computation not only validates the correctness of the solutions but also reinforces the mathematical concepts through hands-on practice.

The four main concepts explored in this report are congruences, relations, functions, and Eulerian paths and circuits. **Congruences** are a fundamental aspect of number theory, representing equivalence relations under modular arithmetic that help in simplifying computations involving large numbers. **Relations** and **functions** are foundational in set theory and mathematics, where a relation describes a connection between elements of sets, and a function is a specific type of relation with unique mappings from inputs to outputs. **Eulerian paths and circuits** are crucial concepts in graph theory, where an Eulerian path traverses each edge of a graph exactly once, and an Eulerian circuit is an Eulerian path that starts and ends at the same vertex.

Java Code

```
import java.util.*;

public class Main {
	public static void main(String[] args) {
	Scanner scanner = new Scanner(System.in);
	boolean keepRunning = true;

	while (keepRunning) {
	System.out.println("Choose an operation:");
	System.out.println("1. Modular Arithmetic");
	System.out.println("2. Relation");
	System.out.println("3. Function");
	System.out.println("4. Graph");
	System.out.println("5. Exit");
	System.out.print("Enter your choice: ");
	int choice = scanner.nextInt();
	scanner.nextLine(); // Consume newline left-over

	switch (choice) {
		case 1:
		modularArithmetic(scanner);
		break;
		case 2:
		relation(scanner);
		break;
		case 3:
		function(scanner);
		break;
		case 4:
		graph(scanner);
		break;
		case 5:
		keepRunning = false;
		break;
		default:
		System.out.println("Invalid choice. Please try again.");
	}
	}

	scanner.close();
	System.out.println("Goodbye!");
	}

	// Modular Arithmetic
	public static void modularArithmetic(Scanner scanner) {
```

```
                System.out.println("Enter the integers (enter a non-integer to stop):");
                List<Integer> numbers = new ArrayList<>();

                while (scanner.hasNextInt()) {
                numbers.add(scanner.nextInt());
                }
                scanner.nextLine(); // Consume newline left-over

                if (numbers.isEmpty()) {
                System.out.println("No integers provided.");
                return;
                }

                int maxModulus = numbers.stream().max(Integer::compare).get(); // Maximum integer in the list

                for (int mod = 2; mod <= maxModulus; mod++) { // Start from modulus 2
                Map<Integer, List<Integer>> congruenceGroups = new HashMap<>();

                // Group numbers by their congruence modulo 'mod'
                for (Integer number : numbers) {
                        int congruence = ((number % mod) + mod) % mod; // Ensure positive congruence
                        if (!congruenceGroups.containsKey(congruence)) {
                        congruenceGroups.put(congruence, new ArrayList<>());
                        }
                        congruenceGroups.get(congruence).add(number);
                }

                // Print groups of numbers that are congruent modulo 'mod'
                for (List<Integer> group : congruenceGroups.values()) {
                        if (group.size() > 1) { // Only print groups with more than one number
                        System.out.println(group + " are congruent modulo " + mod + ".");
                        }
                }
                }
                }

                // Relation
                public static void relation(Scanner scanner) {
                System.out.print("Enter a set (space-separated integers): ");
                String[] setElements = scanner.nextLine().split(" ");
                Set<Integer> set = new HashSet<>();
                for (String element : setElements) {
                set.add(Integer.parseInt(element));
                }

                System.out.print("Enter ordered pairs (space-separated, e.g., 1 1 1 2 1 3): ");
```

```
String[] pairElements = scanner.nextLine().split(" ");
Set<Pair> relation = new HashSet<>();
for (int i = 0; i < pairElements.length; i += 2) {
int a = Integer.parseInt(pairElements[i]);
int b = Integer.parseInt(pairElements[i + 1]);
relation.add(new Pair(a, b));
}

// Check properties
boolean isReflexive = checkReflexive(set, relation);
boolean isIrreflexive = checkIrreflexive(set, relation);
boolean isSymmetric = checkSymmetric(relation);
boolean isAsymmetric = checkAsymmetric(relation);
boolean isAntisymmetric = checkAntisymmetric(relation);
boolean isTransitive = checkTransitive(relation);
boolean isEquivalence = isReflexive && isSymmetric && isTransitive;

// Output the results
System.out.println("Properties identified are:");
if (isReflexive) System.out.println("Reflexive");
if (isIrreflexive) System.out.println("Irreflexive");
if (isSymmetric) System.out.println("Symmetric");
if (isAsymmetric) System.out.println("Asymmetric");
if (isAntisymmetric) System.out.println("Antisymmetric");
if (isTransitive) System.out.println("Transitive");
if (isEquivalence) System.out.println("Equivalence");
}

private static boolean checkReflexive(Set<Integer> set, Set<Pair> relation)
{
for (int element : set) {
if (!relation.contains(new Pair(element, element))) {
        return false;
}
}
return true;
}

private static boolean checkIrreflexive(Set<Integer> set, Set<Pair> relation)
{
for (int element : set) {
if (relation.contains(new Pair(element, element))) {
        return false;
}
}
return true;
}

private static boolean checkSymmetric(Set<Pair> relation) {
```

```
                for (Pair pair : relation) {
                if (!relation.contains(new Pair(pair.b, pair.a))) {
                        return false;
                }
                }
                return true;
                }

                private static boolean checkAsymmetric(Set<Pair> relation) {
                for (Pair pair : relation) {
                if (!pair.equals(new Pair(pair.b, pair.a)) && relation.contains(new
Pair(pair.b, pair.a))) {
                        return false;
                }
                }
                return true;
                }

                private static boolean checkAntisymmetric(Set<Pair> relation) {
                for (Pair pair : relation) {
                if (!pair.equals(new Pair(pair.b, pair.a)) && relation.contains(new
Pair(pair.b, pair.a))) {
                        return false;
                }
                }
                return true;
                }

                private static boolean checkTransitive(Set<Pair> relation) {
                for (Pair pair1 : relation) {
                for (Pair pair2 : relation) {
                        if (pair1.b == pair2.a && !relation.contains(new Pair(pair1.a,
pair2.b))) {
                        return false;
                        }
                }
                }
                return true;
                }

                static class Pair {
                int a, b;

                Pair(int a, int b) {
                this.a = a;
                this.b = b;
                }

                @Override
```

```
public boolean equals(Object o) {
if (this == o) return true;
if (o == null || getClass() != o.getClass()) return false;
Pair pair = (Pair) o;
return a == pair.a && b == pair.b;
}

@Override
public int hashCode() {
return 31 * a + b;
}
}

// Function
public static void function(Scanner scanner) {
System.out.print("Enter input, X (space-separated integers): ");
String[] domainElements = scanner.nextLine().split(" ");
Set<Integer> domain = new HashSet<>();
for (String element : domainElements) {
try {
        domain.add(Integer.parseInt(element));
} catch (NumberFormatException e) {
        System.out.println("Invalid integer: " + element);
        return;
}
}

System.out.print("Enter output, Y (space-separated characters): ");
String[] codomainElements = scanner.nextLine().split(" ");
Set<Character> codomain = new HashSet<>();
for (String element : codomainElements) {
if (element.length() != 1) {
        System.out.println("Invalid character: " + element);
        return;
}
codomain.add(element.charAt(0));
}

System.out.print("Enter ordered pairs (e.g., 1 a 2 b 3 c): ");
String[] pairs = scanner.nextLine().split(" ");
if (pairs.length % 2 != 0) {
System.out.println("Invalid number of elements in ordered pairs.");
return;
}

Map<Integer, Character> function = new HashMap<>();
try {
for (int i = 0; i < pairs.length; i += 2) {
        int a = Integer.parseInt(pairs[i]);
```

```
                    char b = pairs[i + 1].charAt(0);
                    function.put(a, b);
                }
                } catch (NumberFormatException e) {
                System.out.println("Invalid integer in ordered pairs.");
                return;
                } catch (IndexOutOfBoundsException e) {
                System.out.println("Invalid character in ordered pairs.");
                return;
                }

                calculateAndDisplayProperties(domain, codomain, function);
                }

                private static void calculateAndDisplayProperties(Set<Integer> domain,
Set<Character> codomain, Map<Integer, Character> function) {
                boolean isEverywhereDefined = checkEverywhereDefined(domain,
function);
                boolean isInjective = checkInjective(function);
                boolean isSurjective = checkSurjective(codomain, function);
                boolean isBijective = isInjective && isSurjective;
                boolean isInvertible = isBijective; // A function is invertible if and only if it
is bijective

                // Output the results
                System.out.println("Properties identified are:");
                if (isEverywhereDefined) System.out.println("Everywhere Defined");
                if (isInjective) System.out.println("Injective");
                if (isSurjective) System.out.println("Surjective");
                if (isBijective) System.out.println("Bijective");
                if (isInvertible) System.out.println("Invertible");
                }

                private static boolean checkEverywhereDefined(Set<Integer> domain,
Map<Integer, Character> function) {
                return function.keySet().equals(domain);
                }

                private static boolean checkInjective(Map<Integer, Character> function) {
                Set<Character> values = new HashSet<>(function.values());
                return values.size() == function.size();
                }

                private static boolean checkSurjective(Set<Character> codomain,
Map<Integer, Character> function) {
                return new HashSet<>(function.values()).containsAll(codomain);
                }

                // Graph
```

```
        public static void graph(Scanner scanner) {
        System.out.println("Enter a graph (set of edges e.g. 1 2 1 4 2 3 3 4):");
        String[] edgesInput = scanner.nextLine().split(" ");
        List<int[]> edgesList = new ArrayList<>();
        for (int i = 0; i < edgesInput.length; i += 2) {
        int[] edge = new int[]{Integer.parseInt(edgesInput[i]),
Integer.parseInt(edgesInput[i + 1])};
        edgesList.add(edge);
        }

        // Construct adjacency matrix from edges list
        int numVertices = 0;
        for (int[] edge : edgesList) {
        numVertices = Math.max(numVertices, Math.max(edge[0], edge[1]));
        }
        int[][] graph = new int[numVertices][numVertices];
        for (int[] edge : edgesList) {
        int vertex1 = edge[0] - 1;
        int vertex2 = edge[1] - 1;
        graph[vertex1][vertex2] = 1;
        graph[vertex2][vertex1] = 1;
        }

        System.out.println("The graph G = " +
Arrays.deepToString(edgesList.toArray()));

        if (hasEulerCircuit(graph)) {
        System.out.println("There is an Euler circuit");
        } else {
        System.out.println("There is no Euler circuit");
        }

        if (hasEulerPath(graph)) {
        System.out.println("There is an Euler path");
        } else {
        System.out.println("There is no Euler path");
        }
        }

        // Function to check if the graph contains an Euler circuit
        public static boolean hasEulerCircuit(int[][] graph) {
        int numVertices = graph.length;
        for (int i = 0; i < numVertices; i++) {
        int degree = 0;
        for (int j = 0; j < numVertices; j++) {
                degree += graph[i][j];
        }
        if (degree % 2 != 0) {
                return false;
```

```
                }
                }
                return true;
                }

                // Function to check if the graph contains an Euler path
                public static boolean hasEulerPath(int[][] graph) {
                int numVertices = graph.length;
                int oddDegreeVertices = 0;
                for (int i = 0; i < numVertices; i++) {
                int degree = 0;
                for (int j = 0; j < numVertices; j++) {
                        degree += graph[i][j];
                }
                if (degree % 2 != 0) {
                        oddDegreeVertices++;
                }
                }
                return oddDegreeVertices == 0 || oddDegreeVertices == 2;
                }
}
```

Results

```
 Choose an operation:
1. Modular Arithmetic
2. Relation
3. Function
4. Graph
5. Exit

 Enter your choice: 1
Enter the integers (enter a non-integer to stop):
5 2 -3 9 d
[5, -3, 9] are congruent modulo 2.
[-3, 9] are congruent modulo 3.
[5, 2] are congruent modulo 3.
[5, -3, 9] are congruent modulo 4.
[2, -3] are congruent modulo 5.
[-3, 9] are congruent modulo 6.
[2, 9] are congruent modulo 7.
[5, -3] are congruent modulo 8.

 Choose an operation:
1. Modular Arithmetic
2. Relation
3. Function
4. Graph
5. Exit

 Enter your choice: 2
Enter a set (space-separated integers): 1 2 3 4
Enter ordered pairs (space-separated, e.g., 1 1 1 2 1 3): 1 1 1 2 1 3 2 1 2 2 3 1 3 3 4 4
Properties identified are:

Reflexive
Symmetric
```

```
 Choose an operation:
1. Modular Arithmetic
2. Relation
3. Function
4. Graph
5. Exit

 Enter your choice: 3
Enter input, X (space-separated integers): 1 2 3
Enter output, Y (space-separated characters): a b c
Enter ordered pairs (e.g., 1 a 2 b 3 c): 1 a 2 b 3 c
Properties identified are:
Everywhere Defined
Injective
Surjective
Bijective
Invertible

 Choose an operation:
1. Modular Arithmetic
2. Relation
3. Function
4. Graph
5. Exit

 Enter your choice: 4
Enter a graph (set of edges e.g. 1 2 1 4 2 3 3 4):
1 2 1 4 2 3 3 4

 The graph G = [[1, 2], [1, 4], [2, 3], [3, 4]]
There is an Euler circuit
There is an Euler path
```

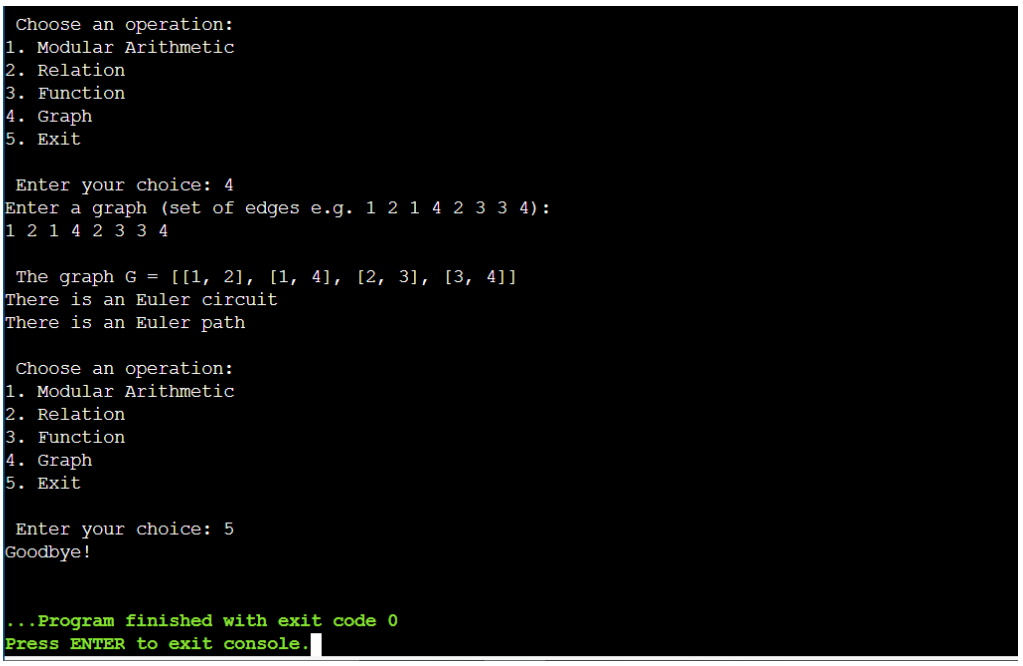


Table of Inputs and Outputs

	Input	Output
1	Choose an operation: 1. Modular Arithmetic 2. Relation 3. Function 4. Graph 5. Exit Enter your choice: 1 Enter the integers (enter a non-integer to stop): 5 2 -3 9 d	[5, -3, 9] are congruent modulo 2. [-3, 9] are congruent modulo 3. [5, 2] are congruent modulo 3. [5, -3, 9] are congruent modulo 4. [2, -3] are congruent modulo 5. [-3, 9] are congruent modulo 6. [2, 9] are congruent modulo 7. [5, -3] are congruent modulo 8.
2	Choose an operation: 1. Modular Arithmetic 2. Relation 3. Function 4. Graph 5. Exit Enter your choice: 2	Properties identified are: Reflexive Symmetric

	Enter a set (space-separated integers): 1 2 3 4 Enter ordered pairs (space-separated, e.g., 1 1 1 2 1 3): 1 1 1 2 1 3 2 1 2 2 3 1 3 3 4 4	
3	Choose an operation: 1. Modular Arithmetic 2. Relation 3. Function 4. Graph 5. Exit Enter your choice: 3 Enter input, X (space-separated integers): 1 2 3 Enter output, Y (space-separated characters): a b c Enter ordered pairs (e.g., 1 a 2 b 3 c): 1 a 2 b 3 c	Properties identified are: Everywhere Defined Injective Surjective Bijective Invertible
4	Choose an operation: 1. Modular Arithmetic 2. Relation 3. Function 4. Graph 5. Exit Enter your choice: 4 Enter a graph (set of edges e.g. 1 2 1 4 2 3 3 4): 1 2 1 4 2 3 3 4	The graph G = [[1, 2], [1, 4], [2, 3], [3, 4]] There is an Euler circuit There is an Euler path
5	Choose an operation: 1. Modular Arithmetic 2. Relation 3. Function 4. Graph 5. Exit Enter your choice: 5	Goodbye!

Calculations

For modular arithmetic

$A = \{1, 2, 3, 4\}$

$R = \{(1,1), (1,2), (1,3), (2,1), (2,2), (3,1), (3,3), (4,4)\}$

since 1R1, 2R2, 3R3, 4R4 so it is reflexive

it is not irreflexive

1R2, 2R1, 1R3, 3R1 so it is symmetric

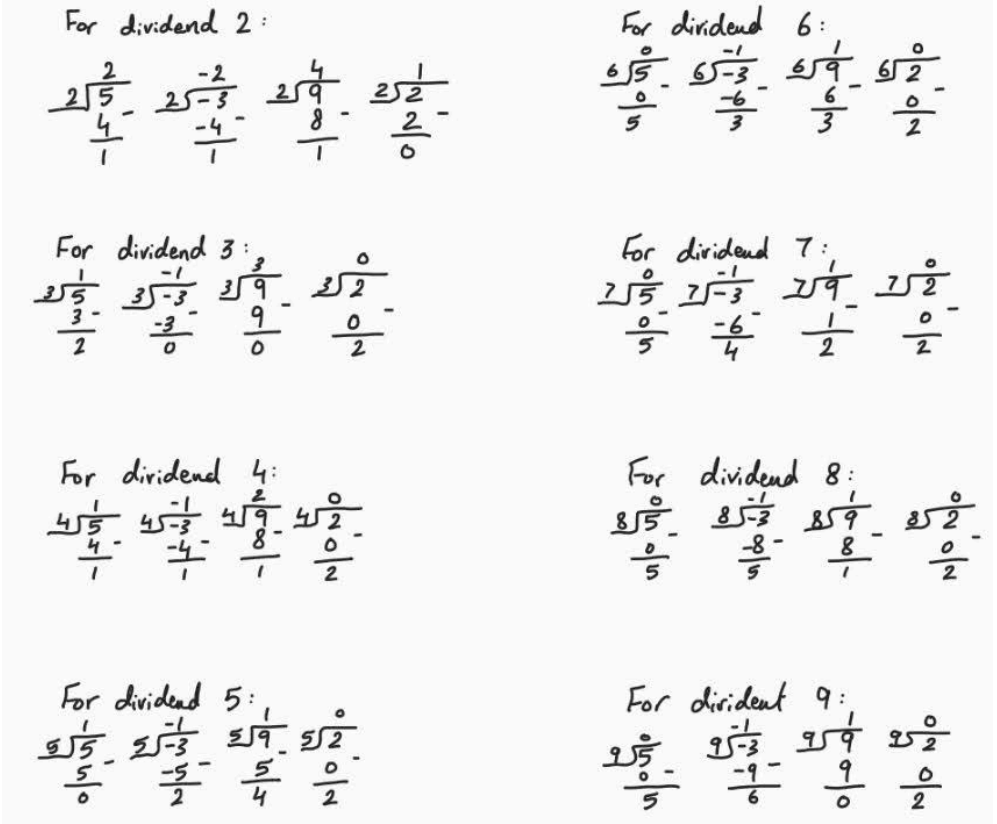
it is not asymmetric

1R2, 2R1, $2 \neq 1$ so it is not anti-symmetric

2R1, 1R3 $\rightarrow$ 2R3 not transitive

not equivalence since it is not transitive

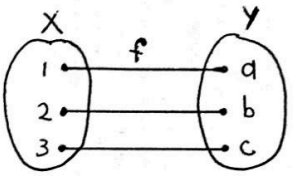
For relation



For function

$X = \{1, 2, 3\}$

$Y = \{a, b, c\}$



→ Every dom(f) = X
Thus, f is everywhere defined

→ f is one-to-one (injective)

→ Ran(f) = {a, b, c} = Y
Thus, f is onto (f is surjective)

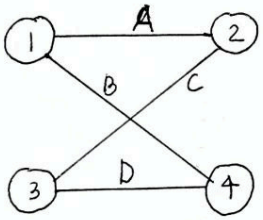
→ f is injective and surjective
Thus, f is bijective

→ $f^{-1} = \{(a,1), (b,2), (c,3)\}$
Hence, f^{-1} is a function
Thus, f is invertible

For graph

Given a graph G,

$G = \{(1,2), (1,4), (2,3), (3,4)\}$



$p = V_p, E_p$

$V_p = 1, 2, 3, 4, 1$

$E_p = A, C, D, B$

There is an Euler path and also an Euler circuit because there has no odd vertices.

Discussion

For the result of this assignment, we have successfully built a user-friendly Java program using the knowledge that has been acquired from the subject: Element of Programming (CSCI 1300). To achieve the objectives, which were to demonstrate knowledge of discrete mathematics, to enhance programming skills, and to improve collaboration skills through teamwork, we dedicated ourselves in doing this project diligently. After writing the codes and executing them we compared our outputs to the answers we got from the manual calculations, As seen above. By doing this we proved that the outputs of the program are correct and align with the calculations. Along the way, we encountered some errors including issues like the code not providing all the congruences or when integrating the four codes in one program. However, we were able to successfully overcome the problems.

Conclusion

In conclusion, by doing this with my classmates, we learned a lot about discrete mathematics through the Java program. The program was designed to perform complex tasks such as calculating congruences, identifying properties of relations and functions, and classifying graphs. This not only enhanced our knowledge and understanding of discrete mathematics but it also helped in learning to code in Java.

References

Kolman, B., Busby, R., & C. Ross, S. (n.d.). Discrete Mathematical Structures (6th ed.). Prentice Hall. (Original work published 2008)

Student's Declaration

This is to certify that we are responsible for the work submitted in this report, that the original work is our own except as specified in the references and acknowledgement, and that the original work contained herein have not been untaken or done by unspecified sources or persons.

We hereby certify that this report has not been done by only one individual and all of us have contributed to the report. The length of contribution to the reports by each individual is noted within this certificate.

We also hereby certify that we have read and understand the content of the total report and no further improvement on the reports is needed from any of the individual's contributors to the report.

We, therefore, agreed unanimously that this report shall be submitted for marking and this final printed report has been verified by us.

Name: Syeda Samia Matric Number: 2123536 Contribution: Modular Arithmetic and Relation Java codes and calculations, integration of all codes to one Java program, Introduction	Read Understand Agree	✓ ✓ ✓
Name: Nurul Hanis Fatini Binti Hairul Fadli Matric Number: 2310460 Contribution: Graph Java codes and calculation, function calculation, Discussion	Read Understand Agree	✓ ✓ ✓
Name: Nur Aien Sofia Binti Rusdi Matric Number: 2319006 Contribution: Function codes, Conclusion	Read Understand Agree	✓ ✓ ✓