VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**NGÔ TRUNG TIẾN – 522H0040**

**HUỲNH VŨ MINH HIẾU – 522H0024**

**NGUYỄN ĐÌNH VIỆT HOÀNG – 522H0120**

**FINAL REPORT**

**DISCRETE STRUCTURES**

**HO CHI MINH CITY, YEAR 2024**

VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**NGÔ TRUNG TIẾN – 522H0040**

**HUỲNH VŨ MINH HIẾU – 522H0024**

**NGUYỄN ĐÌNH VIỆT HOÀNG – 522H0120**

**FINAL REPORT**

**DISCRETE STRUCTURES**

Instructor

**Mr. Mai Duy Tân**

**HO CHI MINH CITY, YEAR 2024**

**ACKNOWLEDGEMENT**

We sincerely thank Mr. Mai Duy Tân for teaching us the Discrete Structures course with great enthusiasm. We want to express our deep appreciation for the dedication and professional knowledge that you shared with us. Through your classes, we gained a better understanding of the fundamental aspects of the Discrete Structures, thanks to your detailed explanations and practical applications. You helped us grasp the knowledge and apply it effectively. Finally, we extend our heartfelt gratitude to Mr. Mai Duy Tân for your commitment and invaluable support throughout our learning journey in this course. The skills and knowledge we acquired will continue to impact our future development. We sincerely thank you and wish your health, success, and happiness.

*Ho Chi Minh City, June 3, 2024*

*Authors:*

*Ngô Trung Tiến*

*Huỳnh Vũ Minh Hiếu*

*Nguyễn Đình Việt Hoàng*

**DECLARATION OF AUTHORSHIP**

Our group assures that this is our own report and was guided by Mr. Mai Duy Tân. The research content and results in this report are honest and have not been published in any form before. The figures in the tables used for analysis, comments, and evaluations were collected by the authors from various sources clearly stated in the reference section.

Additionally, the report includes some comments, evaluations, and data from other authors and organizations, all of which are cited and noted for their origin.

**If any fraud is detected, we fully take responsibility for the content of our final report for the second semester of the 2023-2024 academic year.** Ton Duc Thang University is not involved in any copyright or intellectual property violations that we may cause during the process (if any).

*Ho Chi Minh City, June 3, 2024*

*Authors:*

*Ngô Trung Tiến*

*Huỳnh Vũ Minh Hiếu*

*Nguyễn Đình Việt Hoàng*

**INSTRUCTOR RUBRIC**

Supervisor’s Name: ……………………………………………………………..............

Comments: ……………………………………………………………………………...

Total Score Based on Rubric Evaluation: ………………………………………………

*Ho Chi Minh City, date … month … year …*

*Supervisor*

*(sign and write your full name)*

**TABLE OF CONTENTS**

[LIST OF FIGURES vii](#_Toc168335711)

[QUESTION 1: EUCLID’S ALGORITHM AND BEZOUT’S IDENTITY 1](#_Toc168335712)

[QUESTION 2: RECURRENCE RELATION 3](#_Toc168335713)

[QUESTION 3: SET 4](#_Toc168335714)

[QUESTION 4: RELATIONS 5](#_Toc168335715)

[QUESTION 5: KRUSKAL’S ALGORITHM 7](#_Toc168335716)

[QUESTION 6: EULERIAN CIRCUIT 11](#_Toc168335717)

[QUESTION 7: MAP COLORING 18](#_Toc168335719)

[QUESTION 8: FINDING AN INVERSE MODULO (n) 24](#_Toc168335722)

[QUESTION 9: RSA CRYPTOSYSTEM 29](#_Toc168335724)

[REFERENCES 40](#_Toc168335726)

# LIST OF FIGURES

[Figure 6.c: An Eulerian circuit of that graph when the initial circuit R1 is EINME 16](#_Toc168316701)

[Figure 7.a: The map can be represented as a graph 18](#_Toc168316703)

[Figure 7.b: The map (graph) is completed with the minimum number of colors 22](#_Toc168316704)

[Figure 8: The result of source code for question 8 26](#_Toc168316706)

[Figure 9: The result of source code for question 9 32](#_Toc168316708)

QUESTION 1: EUCLID’S ALGORITHM AND BEZOUT’S IDENTITY

a. gcd(2024, 1000 + m) and lcm(2024, 1000 + m), with m = 24

* gcd(2024, 1024) and lcm(2024, 1024)

Consider gcd(2024, 1024):

2024 = 1024 x 1 + 1000 🡨 gcd(1024, 1000)

1024 = 1000 x 1 + 24 🡨 gcd(1000, 24)

1000 = 24 x 41 + 16 🡨 gcd(24, 16)

24 = 16 x 1 + 8 🡨 gcd(16, 8)

16 = 8 x 2 + 0 🡨 gcd(8, 0)

The result is: gcd(2024, 1024) = 8

* lcm(2024, 1024) = (2024 x 1024) / 8 = 259072

b. Step 1: Use Extended Euclidean Algorithm:

Now, we work backward to express 8 as a combination of 2024 and 1024:

1. 8 = 24 – 16 × 1
2. 16 = 1000 – 24 × 41
   * Substitute 16 in the equation for 8:

8 = 24 − (1000 – 24 × 41) = 42 × 24 − 1000

1. 24 = 1024 − 1000
   * Substitute 24 in the equation for 8:

8 = 42 × (1024 − 1000) – 1000 = 42 × 1024 – 43 × 1000

1. 1000 = 2024 – 1024 × 1
   * Substitute 1000 in the equation for 8:

8 = 42 × 1024 – 43 × (2024 − 1024) = 85 × 1024 – 43 × 2024

So, one particular solution is: = − 43, = 85

Step 2: General Solution:

The general solution to 2024x + 1024y = 8 can be written as:

x = + = − 43 + 128k

y = + = 85 - 253k

where k is any integer.

Step 3: Five Specific Solutions:

Let's find five specific solutions by choosing different values for k:

1. For k = 0:

x = − 43

𝑦 = 85

(− 43,85)

1. For k = 1:

x = − 43 + 128 × 1 = 85

y = 85 – 253 × 1 = − 168

(85, − 168)

1. For k = 2:

x = − 43 + 128 × 2 = 213

y = 85 – 253 × 2 = − 421

(213, − 421)

1. For k = 3:

x = − 43 + 128 × 3 = 341

y = 85 – 253 × 3 = − 674

(341, − 674)

1. For k = 4:

x = − 43 + 128 × 4 = 469

y = 85 – 253 × 4 = − 927

(469, − 927)

Step 4: Conclusion:

The five specific solutions for the equation 2024x + 1024y = 8 are:

(− 43, 85), (85, − 168), (213, − 421), (341, − 674), (469, − 927)

QUESTION 2: RECURRENCE RELATION

With and

Step 1: Find the characteristic equation:

For the recurrence relation , the characteristic equation is obtained by assuming a solution of the form . Substituting this into the recurrence relation gives:

Dividing through by (assuming r 0) gives:

⬄

Step 2: Solve the characteristic equation:

To solve the quadratic equation , we use the quadratic formula , where *a* = 1, 𝑏 = −8 and 𝑐 = 15*:* r = =

⬄

Step 3: Form the general solution:

The general solution to the recurrence relation is:

Step 4: Determine the constants using initial conditions:

We use the initial conditions and to find the constants A and B.

1. For 𝑛 = 0:
2. For 𝑛 = 1:

So, we have the system of linear equations:

⬄

Step 5: Write the particular solution:

The constants A and B are and ​. Therefore, the particular solution to the recurrence relation is:

So, the solution to the recurrence relation ​ with and is:

QUESTION 3: SET

a.

My full name is: “Nguyễn Đình Việt Hoàng”

Γ = {N, G, U, Y, E, D, I, H, V, T, O, A}

Δ = {A, C, D, G, H, N, O, T, U}

b.

Union of Γ and Δ: Γ ∪ Δ = {N, G, U, Y, E, D, I, H, V, T, O, A, C}

Intersect of Γ and Δ: Γ ∩ Δ = {N, G, U, D, H, T, O, A}

Non-symmetric difference of Γ and Δ: Γ ∖ Δ = {Y, E, I, V}

Symmetric difference of Γ and Δ: Γ Δ Δ = (Γ ∖ Δ) ∪ (Δ ∖ Γ) = {Y, E, I, V} ∪ {C} = {Y, E, I, V, C}

QUESTION 4: RELATIONS

To determine whether the binary relation R defined on two integers a and b by aRb ⟺ 24∣(a⋅b) is reflexive, symmetric, anti-symmetric, and transitive, we will analyze each property one by one.

1. **Reflexive:**

A relation R is reflexive if every element is related to itself, i.e., aRa for all a.

For aRa, we need 24∣(a⋅a), which means must be divisible by 24.

Since 24 = .3, must be divisible by both 8 and 3. Not all natural numbers satisfy this condition. For example, if a = 1, = 1 is not divisible by 24. Thus, R is **not reflexive**.

1. **Symmetric:**

A relation R is symmetric if aRb implies bRa for all a,b.

Given aRb, we have 24∣(a⋅b). Since multiplication is commutative, 24∣(b⋅a) as well. Therefore, bRa holds whenever aRb holds, making R **symmetric**.

1. **Anti-symmetric:**

A relation R is anti-symmetric if aRb and bRa imply a = b.

Given aRb and bRa, we have 24∣(a⋅b) and 24∣(b⋅a). These conditions do not necessarily imply a = b. For example, 24∣(4⋅6) and 24∣(6⋅4) but 4 ≠ 6. Hence, R is **not anti-symmetric**.

1. **Transitive:**

A relation R is transitive if aRb and bRc imply aRc.

Given aRb and bRc, we have 24∣(a⋅b) and 24∣(b⋅c). This implies that b must be such that 24∣(a⋅c) holds. However, if we choose a = 6, b = 4, and c = 6, then:

* 24∣(6⋅4) (since 24 divides 24)
* 24∣(4⋅6) (since 24 divides 24)

But 24∤(6⋅6) (since 24 does not divide 36). Thus, aRc does not hold even though aRb and bRc hold, making R **not transitive**.

* In summary, the relation R defined by aRb ⟺ 24∣(a⋅b) is:
  + Not reflexive
  + Symmetric
  + Not anti-symmetric
  + Not transitive

QUESTION 5: KRUSKAL’S ALGORITHM

*I/ Solution for Circuit-Checking in Kruskal's Algorithm:*

Kruskal's algorithm is used to find the Minimum Spanning Tree (MST) of a graph. The algorithm works by sorting all the edges in the graph by their weight and then adding the shortest edge to the growing spanning tree, provided it does not form a cycle. Circuit-checking (or cycle detection) is crucial in this process to ensure that no cycles are formed.

* *Circuit-Checking Using Union-Find Data Structure*

The Union-Find data structure (also known as Disjoint Set Union, DSU) is an efficient method to detect cycles in Kruskal's algorithm. Here's how it works:

1. Initialization:
   * Create a parent array where each vertex is its own parent.
   * Create a rank array to keep track of the tree height for balancing.
2. Find Operation:
   * A function to find the root of a vertex with path compression for efficiency.
3. Union Operation:
   * A function to unite two subsets using the rank to keep the tree flat.
4. Cycle Detection:
   * Before adding an edge (u, v), check if u and v belong to the same subset (using the find operation).
   * If they do, adding this edge would create a cycle.
   * If they do not, add the edge and union their subsets.

*II/ Example:*

Let's go through an example to illustrate Kruskal's algorithm and circuit-checking using Union-Find:

Graph:

Vertices: {A, B, C, D, E}

Edges (with weights):

* (A, B, 1)
* (A, C, 3)
* (B, C, 2)
* (B, D, 4)
* (C, D, 5)
* (C, E, 6)
* (D, E, 7)

Steps of Kruskal’s Algorithm with Union-Find:

1. Sort the edges by weight:
   * (A, B, 1)
   * (B, C, 2)
   * (A, C, 3)
   * (B, D, 4)
   * (C, D, 5)
   * (C, E, 6)
   * (D, E, 7)
2. Initialize Union-Find:
   * Parent: [A, B, C, D, E] (each vertex is its own parent)
   * Rank: [0, 0, 0, 0, 0]
3. Process edges:
   * Edge (A, B, 1):
     + Find(A) != Find(B)
     + Union(A, B)
     + MST: {(A, B, 1)}
     + Updated Parent: [A, A, C, D, E]
     + Updated Rank: [1, 0, 0, 0, 0]
   * Edge (B, C, 2):
     + Find(B) -> Find(A) != Find(C)
     + Union(A, C)
     + MST: {(A, B, 1), (B, C, 2)}
     + Updated Parent: [A, A, A, D, E]
     + Updated Rank: [1, 0, 0, 0, 0]
   * Edge (A, C, 3):
     + Find(A) == Find(C) (cycle detected)
     + Skip this edge
   * Edge (B, D, 4):
     + Find(B) -> Find(A) != Find(D)
     + Union(A, D)
     + MST: {(A, B, 1), (B, C, 2), (B, D, 4)}
     + Updated Parent: [A, A, A, A, E]
     + Updated Rank: [1, 0, 0, 0, 0]
   * Edge (C, D, 5):
     + Find(C) -> Find(A) == Find(D) (cycle detected)
     + Skip this edge
   * Edge (C, E, 6):
     + Find(C) -> Find(A) != Find(E)
     + Union(A, E)
     + MST: {(A, B, 1), (B, C, 2), (B, D, 4), (C, E, 6)}
     + Updated Parent: [A, A, A, A, A]
     + Updated Rank: [1, 0, 0, 0, 0]
   * Edge (D, E, 7):
     + Find(D) -> Find(A) == Find(E) (cycle detected)
     + Skip this edge

* Resulting MST:

The Minimum Spanning Tree (MST) includes the edges:

* (A, B, 1)
* (B, C, 2)
* (B, D, 4)
* (C, E, 6)

The total weight is: 1 + 2 + 4 + 6 = 13

QUESTION 6: EULERIAN CIRCUIT

a.

*I/ Eulerian Path and Circuit Criteria:*

Eulerian Circuit: A graph has an Eulerian circuit if and only if every vertex has an even degree and the graph is connected.

Eulerian Path: A graph has an Eulerian path if and only if exactly zero or two vertices have an odd degree and the graph is connected.

*II/ Conditions:* For the given illustrated graph, it is an undirected graph, so the appropriate conditions are:

1. Eulerian Circuit:

A graph has an Eulerian circuit if and only if:  
- The vertices with non-zero degree are connected.  
- All vertices in the graph have even degrees.

1. Eulerian Path:

A graph has an Eulerian path if and only if:  
- The vertices with non-zero degree are connected.  
- The graph has exactly 0 or 2 vertices with odd degrees.

*III/ Checking the Degrees of Vertices:*

Vertex A: Connected to E, F, I, J, K, L. Degree of A is 6.

Vertex B: Connected to E, G, M, N, S, Y. Degree of B is 6.

Vertex C: Connected to F, H, R, X, d, j. Degree of C is 6.

Vertex D: Connected to G, H, n, i, k, m. Degree of D is 6.

Vertex E: Connected to A, B, I, M. Degree of E is 4.

Vertex F: Connected to A, C, K, R. Degree of F is 4.

Vertex G: Connected to B, D, S, k. Degree of G is 4.

Vertex H: Connected to C, D, X, i. Degree of H is 4.

For the inner vertices like I, J, K, L, R, M, N, O, P, Q, S, T, U, V, W, X, Y, Z, a, b, c, d, e, f, g, h, i, j, k, l, m, n: Each of these vertices is connected to 4 other vertices, so their degrees are 4.

*IV/ Summary:*

Vertices with odd degrees: There are no vertices with odd degrees; all vertices have even degrees.

*V/ Final Conclusion:*

Since all vertices in the graph have even degrees and the graph is connected, this graph has an Eulerian circuit.

b.

*I/ Hierholzer’s Algorithm to Find an Eulerian Circuit:*

Hierholzer’s algorithm is a classic method used to find an Eulerian circuit in a connected graph where each vertex has an even degree. The algorithm, named after Carl Hierholzer, leverages the properties of Eulerian graphs to construct the circuit efficiently.

*II/ Key Concepts:*

**Eulerian Circuit:** A circuit that visits every edge of a graph exactly once and returns to the starting vertex.

* **Conditions for an Eulerian Circuit:**

1. The graph must be connected.
2. Every vertex must have an even degree.

*III/ Steps of Hierholzer’s Algorithm:*

1. **Initialization:**

* Start at any vertex in the graph.

1. **Form an Initial Cycle:**

* Traverse edges starting from the initial vertex, ensuring no edge is traversed more than once, until returning to the starting vertex, thus forming a cycle.
* Remove the edges used in this cycle from the graph.

1. **Extend the Cycle:**

* If there are unused edges, find a vertex in the current cycle that has unused edges.
* From this vertex, start a new traversal to form another cycle that merges back into the existing cycle.
* Repeat this process of merging new cycles with the existing cycle until all edges are used.

1. **Completion:**

* The process ends when all edges have been used exactly once, forming the Eulerian circuit.

*IV/ Detailed Example:*

* + - 1. **Graph Representation:**

Consider a graph with vertices V = {A, B, C, D} and edges E = {AB, BC, CD, DA, AC, BD}.

* + - 1. **Step-by-Step Execution:**

**a/ Initialization:**

* Start at vertex A.

**b/ Form an Initial Cycle:**

* Traverse: A → B → C → D → A
* Cycle formed: A → B → C → D → A
* Remove used edges: {AB, BC, CD, DA}

**c/ Extend the Cycle:**

* Check for unused edges connected to any vertex in the cycle. Vertex C connects to vertex A.
* Form new cycle starting at C: C → A → C
* Integrate new cycle into existing cycle: A → B → C → A → C → D → A

**d/ Completion:**

* All edges are used, forming the Eulerian circuit: A → B → C → A → C → D → A

*V/ Pseudocode:*

def hierholzer(graph):

# Assuming graph is represented as an adjacency list

def remove\_edge(u, v):

graph[u].remove(v)

graph[v].remove(u)

def find\_new\_cycle(start\_vertex):

cycle = []

current\_vertex = start\_vertex

while True:

for neighbor in graph[current\_vertex]:

if neighbor is not None:

cycle.append(current\_vertex)

remove\_edge(current\_vertex, neighbor)

current\_vertex = neighbor

break

if current\_vertex == start\_vertex:

break

cycle.append(start\_vertex)

return cycle

circuit = []

start\_vertex = next(iter(graph))

current\_cycle = find\_new\_cycle(start\_vertex)

circuit.extend(current\_cycle)

while any(graph[v] for v in circuit):

for v in circuit:

if graph[v]:

new\_cycle = find\_new\_cycle(v)

insert\_index = circuit.index(v)

circuit = circuit[:insert\_index] + new\_cycle[:-1] + circuit[insert\_index+1:]

break

return circuit

# Example usage

graph = {

'A': ['B', 'D', 'C'],

'B': ['A', 'C'],

'C': ['B', 'D', 'A'],

'D': ['A', 'C']

}

print(hierholzer(graph))

*VI/ Complexity Analysis:*

**Time Complexity:** O(E), where E is the number of edges. Each edge is traversed exactly once.

**Space Complexity:** O(V + E), due to the storage requirements for the adjacency list and the circuit.

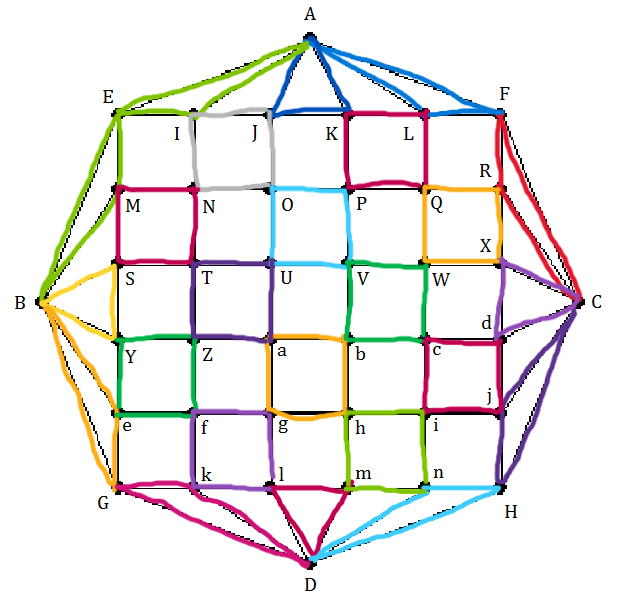
*VII/ Applications:*

* **Network Routing:** Finding optimal paths that traverse all links in a network exactly once.
* **DNA Sequencing:** Constructing sequences that visit each fragment exactly once.
* **Puzzle Solving:** Solving problems like the Königsberg bridge problem and other path-related puzzles.

*VIII/ Limitations:*

* **Graph Requirements: The algorithm can only be applied to graphs where all vertices have even degrees.**
* **Directed Graphs: The algorithm can be extended to directed graphs but requires modifications.**

c.



##### Figure 6.c: An Eulerian circuit of that graph when the initial circuit R1 is EINME

R1 is EINME

R1 is E BME INME

R1 is E BSME INME

R1 is E BYSME INME

R1 is E BeYSME INME

R1 is E BGeYSME INME

R1 is E AIE INME

R1 is E AJIE INME

R1 is E AKJIE INME

R1 is E ALKJIE INME

R1 is E AFLKJIE INME

R1 is E AFLKJIE IJON ME

R1 is E AFLKJIE IJKPON ME

R1 is E AFLKJIE IJKLQPON ME

R1 is E AFLKJIE IJKLFRQPON ME

R1 is E AFLKJIE IJKLFCRQPON ME

R1 is E AFLKJIE INTS ME

R1 is E AFLKJIE INTZYS ME

R1 is E AFLKJIE INTZfeYS ME

R1 is E AFLKJIE INTZfkGeYS ME

R1 is E AFLKJIE INTZfkDGeYS ME

R1 is E AFLKJIE INTZfkDlgaUOJIN ME

R1 is E AFLKJIE INTZfkDmhbVPKJIN ME

R1 is E AFLKJIE INTZfkDnicWQLKJIN ME

R1 is E AFLKJIE INTZfkDHCFLKJIN ME

R1 is E AFLKJIE INOPQRCXWVUTS ME

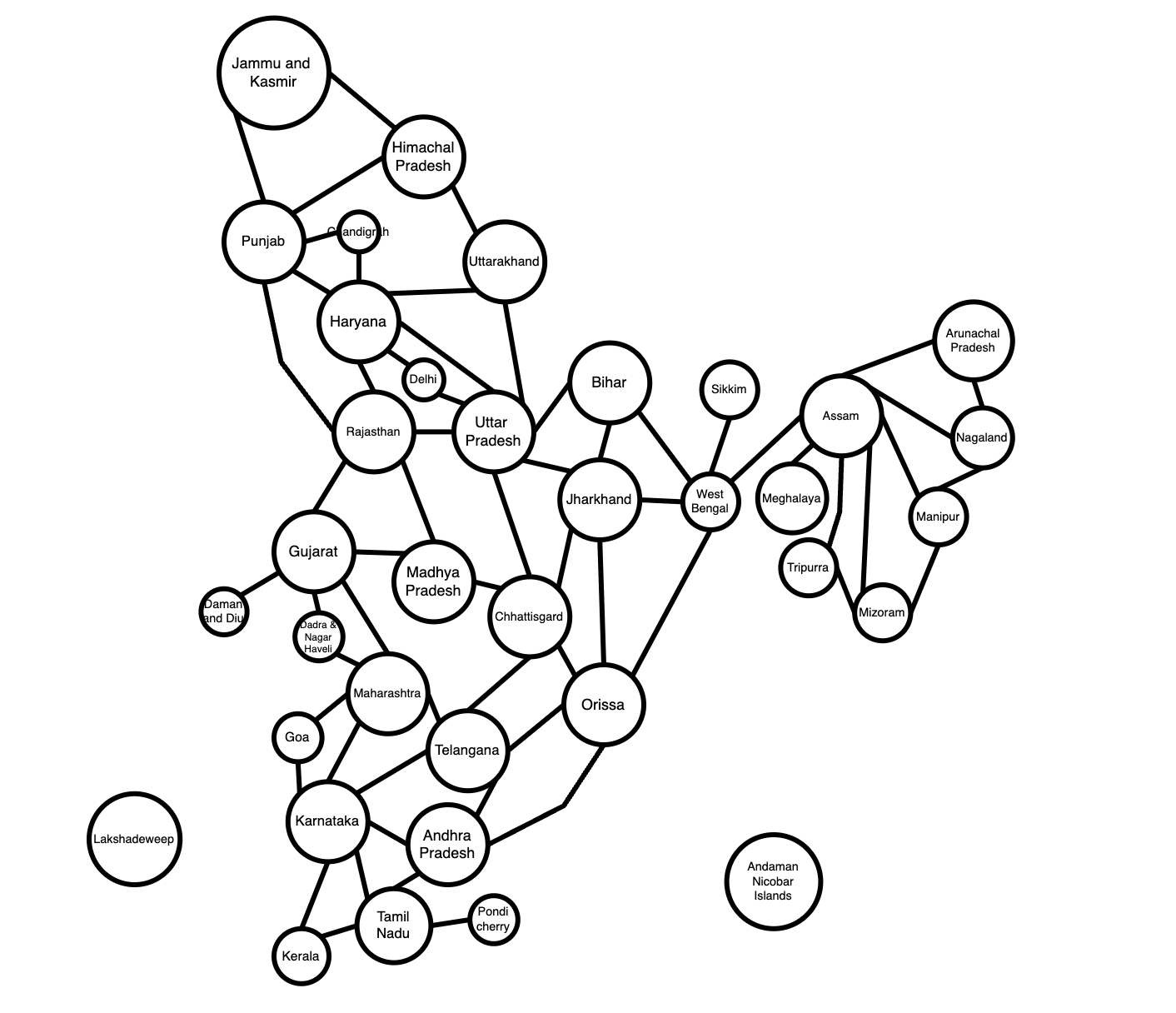
R1 is E AFLKJIE INTUVWXCdcbaZYS ME

R1 is E AFLKJIE INTZabcdCjihgfeYS ME

R1 is E AFLKJIE INTZfghijCHnmlkGeYS ME

QUESTION 7: MAP COLORING

*a. Modeling a map by a graph.*



##### Figure 7.a: The map can be represented as a graph

*b. Color the map (graph) with a minimum number of colors. Present your solution step by step.*

Let be the 4-digit number combined by the last 4 digits in your StudentID*.* Our StudentID is 522H0024 so has = 0024.

0024 % 4 = 0 so start from Bihar.

+ In this question we use 4 color to color the map:

#1:

#2:

#3:

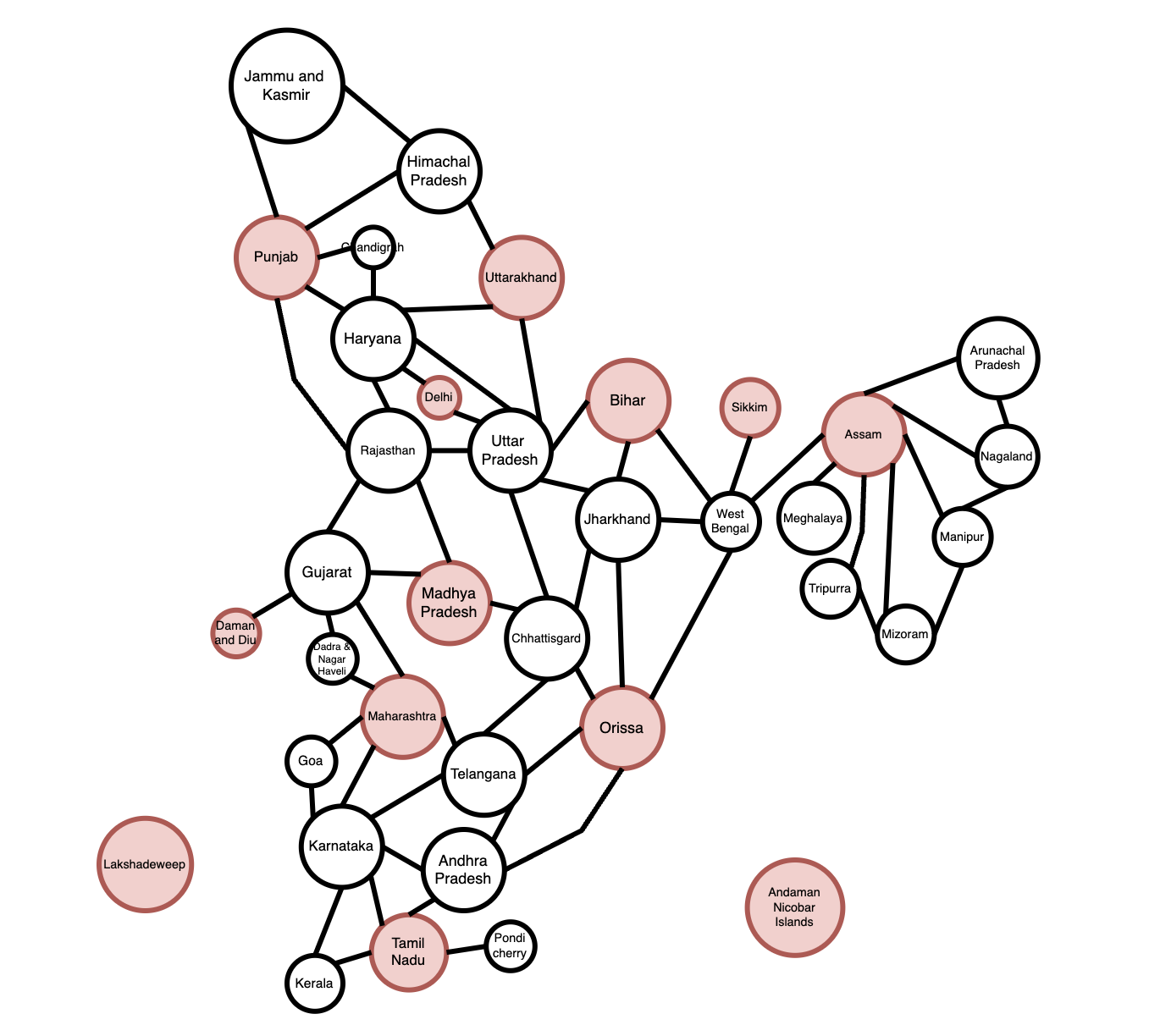
#4:

+ To color the map with a minimum number of colors, we use Greedy coloring algorithm:

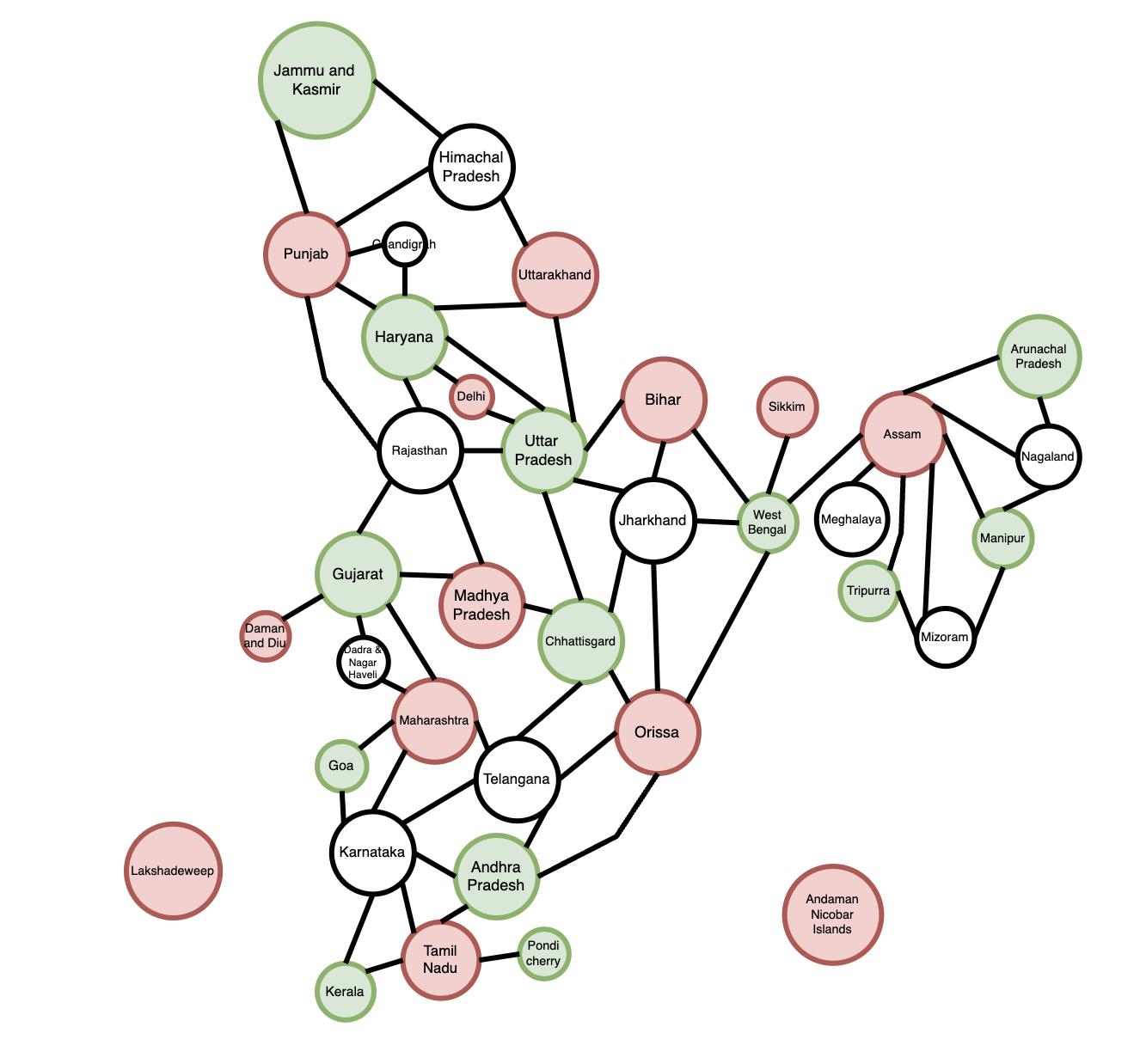
We start with the first node from Bihar. We use the first color is red.

+ Step by step color:

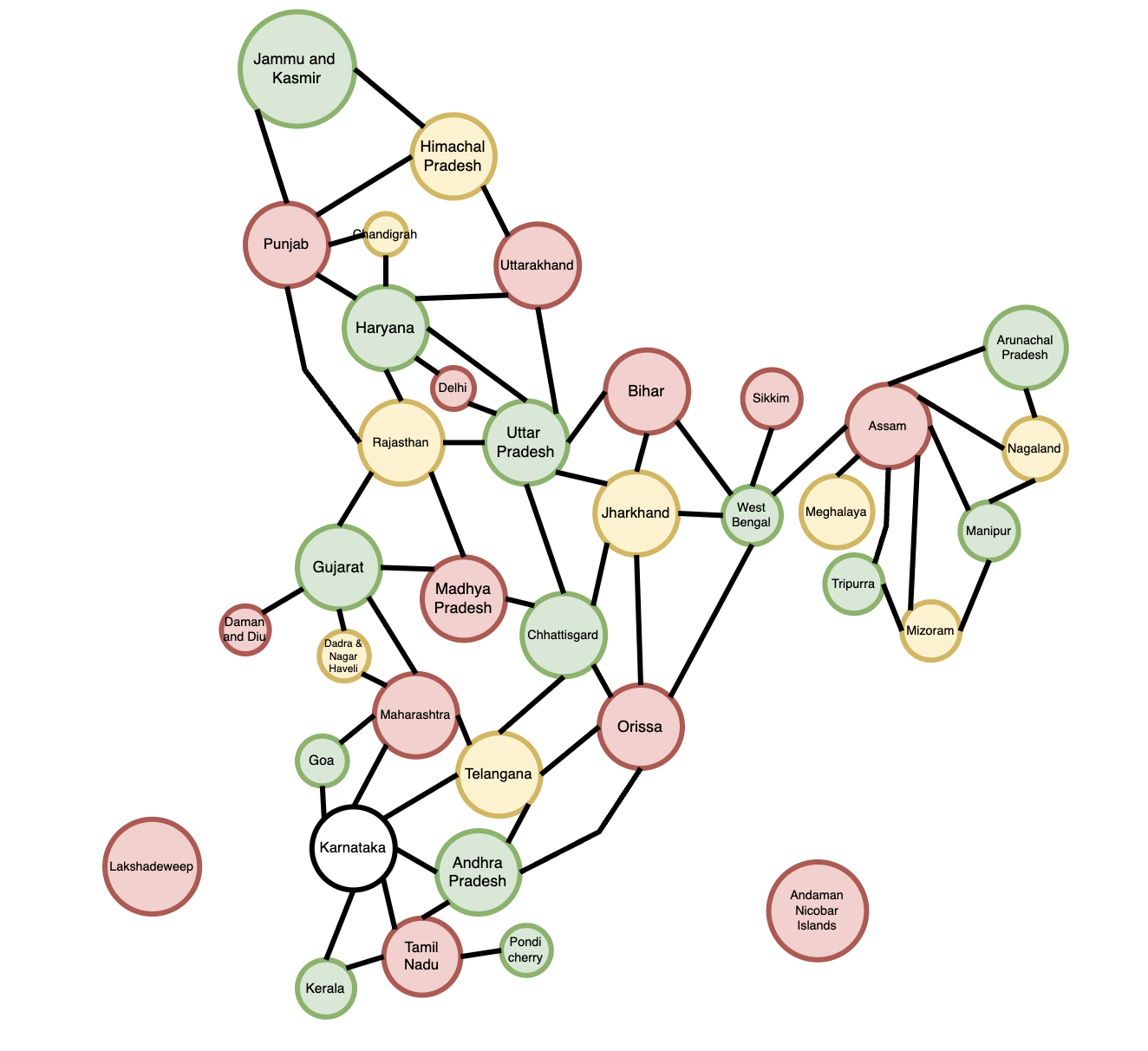
* **Step 1:** We color the node that not the neighbors of each other.



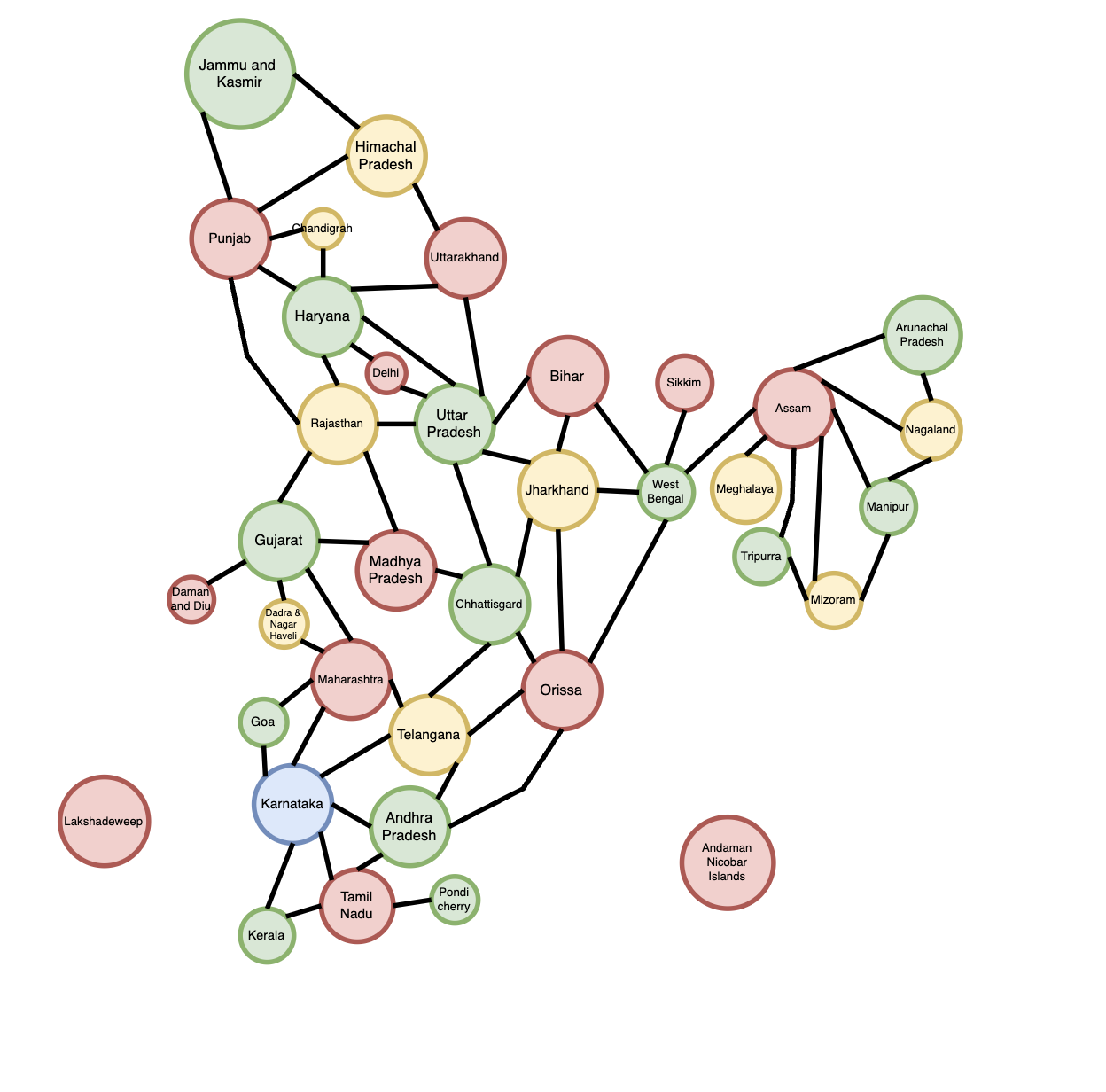
* **Step 2:** Eliminated the colored nodes, then we use the second colors is green to color the node. Start from the Bihar’s neighbor node.



* **Step 3:** Eliminated the colored nodes, then we use the third colors is yellow to color the node. Start from Jharkhand.



* **­­­Step 4:** Eliminated the colored­ nodes, we can see that almost the nodes are colored that didn’t conflict with the colors of its neighbor. We use the final color is blue to Color the Karnataka.



##### Figure 7.b: The map (graph) is completed with the minimum number of colors

* **Summarize:** We used 4 colors to color the map, we have:
* **#1Red**: (Bihar, Sikkim, Assam, Uttarakhand, Delhi, Punjab, Madhya P, Orrisa, Maharashtra, Tamil Nadu, Lakshadeweep, Andaman Nicobar)
* **#2Green**: (J and K, Haryana, Uttar P, West Bengal, Tripurra, Manipur, Arunachal P, Gujarat, Chhattisgrad, Goa, An­­dhra P, Pondicherry, Kerala)
* **#3Yellow**: (Himachal P, Chadigarh, Rajasthan, Jharkhand, Meghalaya, Nagaland, Mizoram, Dadra and Nagar Haveli, Tenlagana)
* **#4Blue**: (Karmataka)­

QUESTION 8: FINDING AN INVERSE MODULO (n)

*I/ Introduction to the Extended Euclidean Algorithm:*

The Extended Euclidean Algorithm not only computes the greatest common divisor (GCD) of two integers a and b, but also finds the integers x and y such that ax + by = gcd(a,b). This extends the basic Euclidean algorithm.

*II/ Extended Euclidean Algorithm:*

**1. Example:**

Suppose you want to find the GCD of 120 and 23, and also find integers x and y such that 120x + 23y = gcd(120,23):

Apply the Euclidean algorithm to 120 and 23:

* 120 % 23 = 8 (i.e., 120 – 5 × 23 = 8)
* 23 % 8 = 7 (i.e., 23 – 2 × 8 = 7)
* 8 % 7 = 1 (i.e., 8 – 7 = 1)
* 7 % 1 = 0 (Ends here, GCD is 1)

Now, backtrack through the steps to find x and y:

* 1 = 8 – 7
* 1 = 8 − (23 – 2 × 8) = 3 × 8 – 23
* 1 = 3 × (120 – 5 × 23) – 23 = 3 × 120 – 16 × 23

Thus, x = 3 and y = − 16 are the integers found.

*III/ Introduction to Modular Inverse:*

In modular arithmetic, there is no traditional division operation. Instead, we have the concept of modular inverses. The modular inverse of a number modulo is a number such that × ≡ 1 (mod ).

In other words, × ≡ 1 (mod ). This means that if gcd(,) ≠ 1, then does not have a modular inverse. Only numbers that share no common factors with (i.e., their greatest common divisor with is 1) have a modular inverse modulo .

**2. Example:**

Let's take = 14. Numbers that share no common factors with 14 are those not divisible by 2 or 7.

*IV/ Method for Finding a Modular Inverse Using the Extended Euclidean Algorithm:*

1/ Use the Euclidean algorithm to find gcd(,).

2/ Write down the equation + = gcd(,) and solve for and .

3/ If gcd(,) = 1, then is the inverse of modulo . If is negative, add to it unti you get a positive value.

**3. Example:**

Choose the number 10 to check if it has an inverse modulo 17 and then calculate the modular inverse if it exists.

First, calculate its gcd with 17 using the extended Euclidean algorithm:

* 17 = 10 × 1 + 7
* 10 = 7 × 1 + 3
* 7 = 3 × 2 + 1
* 3 = 1 × 3 + 0
* gcd(10,17) = 1
* Since gcd(10,17) = 1, 10 has a modular inverse modulo 17.

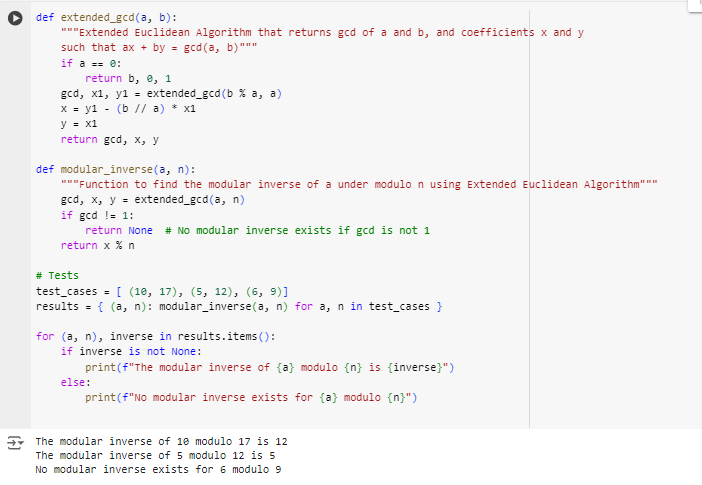
Next, solve for and in the equation 10 + 17 = 1:

* 1 = 7 – 2 × 3
* Replace each 3 with 10 – 7: 1 = 7 – 2 × (10 − 7) = 3 × 7 – 2 × 10
* 17 – 10: 1 = 3 × (17 − 10) – 2 × 10 = 3 × 17 – 5 × 10
* 1 = 3 × 17 – 5 × 10
* So, = − 5 and = 3.

From 1 = 7 – 2 × 3:

* = − 5. Add 17 to get a positive value:
* = − 5 + 17 = 12
* Therefore, the inverse of 10 modulo 17 is 12 because 10 × 12 ≡ 120 ≡ 1 (mod 17).

*V/ Python Implementation and Result:*



##### Figure 8: The result of source code for question 8

*VI/ Explanation of Python Implementation:*

**1. Function extended\_gcd(a, b):**

This function implements the Extended Euclidean Algorithm, which is used to find the greatest common divisor (gcd) of two integers. It also determines the integers x and y such that ax + by = gcd(a,b).

* **Base case:** If a = 0, then the gcd is b, and the corresponding x and y are 0 and 1, respectively.
* **Recursive case:** If a is non-zero, the algorithm calculates gcd(b%a,a) and then uses the result from the recursive call to calculate x and y for the current step.
  + x is updated according to the formula: x = × .
  + y is taken directly from from the previous recursive call.

**2. Function modular\_inverse(a, n):**

This function finds the modular inverse of a modulo n using the Extended Euclidean Algorithm. The modular inverse of a is a number x such that (a × x) % n = 1.

* **Check gcd:** Initially, it calculates gcd(a,n). If the gcd is not equal to 1, the modular inverse does not exist because a and n are not coprime.
* **Calculate the inverse:** If gcd = 1, the function uses the x value from “extended\_gcd” and determines x % n, which is the modular inverse.

**3. Testing the Functions:**

The script defines test cases (10,17), (5,12), and (6,9). It computes the modular inverse for each pair using a dictionary comprehension and then iterates through the results to print the inverse or a message stating that no inverse exists.

* The “modular\_inverse” function returns the modular inverse if it exists, and returns “None” if it does not exist.

*VII/ Output Explanation:*

* **For (10, 17):** The gcd is 1, so it calculates the modular inverse, which is 12. The output “The modular inverse of 10 modulo 17 is 12” confirms that 10 × 12 mod 17 = 1.
* **For (5, 12):** Similarly, the gcd of 5 and 12 is 1, allowing the computation of the modular inverse, which is 5. The output “The modular inverse of 5 modulo 12 is 5” confirms that 5 × 5 mod 12 = 1.
* **For (6, 9):** The gcd is 3 (since both 6 and 9 are divisible by 3), hence no modular inverse exists, as reflected by “No modular inverse exists for 6 modulo 9”.

QUESTION 9: RSA CRYPTOSYSTEM

*I/ Conduct research on RSA cryptosystem, understand the mathematical concepts behind the RSA cryptosystem, including prime number generation, modular arithmetic, extended Euclidean algorithm, prime factorization, etc and give examples:*

+ The RSA cryptosystem (Rivest – Shamir - Adleman), is a widely-used asymmetric cryptographic algorithm named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman. It's commonly used for secure data transmission over the internet, including tasks like secure email communication, digital signatures, and encryption of sensitive data.

+ RSA relies on the mathematical properties of prime numbers and modular arithmetic. It involves a **public key** and a **private key**, and the security of the algorithm is based on the difficulty of factoring large composite numbers into their prime factors.

+ The **public key** consists of two numbers where one number is a multiplication of two large prime numbers. And **private key** is also derived from the same two prime numbers. So, if somebody can factorize the large number, the private key is compromised. Therefore, encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024-bit keys could be broken in the near future. But till now it seems to be an infeasible task.

* **The process of how RSA work:**

1. Key Generation:

* To large numbers, typically denoted as *p* and *q*, chosen randomly.
* The product of these two primes, denoted as *n = p x q*, becomes the modulus for both the public and private keys.
* The Euler’s totient function *ø(n)* is calculated, where *ø(n) = (p – 1) x (q – 1).*
* A public key(*e, n*) is generated, where *e* is a randomly chosen integer that is relatively prime to *ø(n),* meaning that *gcd(e, ø(n)) = 1.*
* The private key *d* is then computed as the modular multiplicative inverse of *e* modulo *ø(n),* meaning that *d x e 1 mod* *ø(n).*

2. Encryption:

* To encrypt a message , the sender uses the recipient’s public key(*e, n*) and compute *mod* *n*.
* The ciphertext is then transmitted to the recipient.

3. Decryption:

* The recipient uses their private key *d* to decrypt the ciphertext by computing *mod* *n*.
* The original message is retrieved.
* RSA is considered secure because factoring the product of two large prime numbers (the modulus *n*) into its prime factors is computationally infeasible for sufficiently large primes. As a result, breaking RSA encryption requires solving the integer factorization problem, which is believed to be hard even for modern computers.
* **Example:**

1. Key Generation:

* Choose two prime numbers, typically denoted as *p = 61* and *q = 53.*
* The product of these two primes, denoted as *n = p x q* *= 61 x 53 = 3233.*
* The Euler’s totient function *ø(n)* is calculated, where *ø(n) = (p – 1) x (q – 1) = 60 x 52 = 3120.*
* Choose *e = 17* and calculate its modular inverse of *e* modulo *ø(n)* is *d.* Using the extend Euclidean algorithm, we find *d = 2753* such that *d x 17 1 mod 3120.*

2. Encryption:

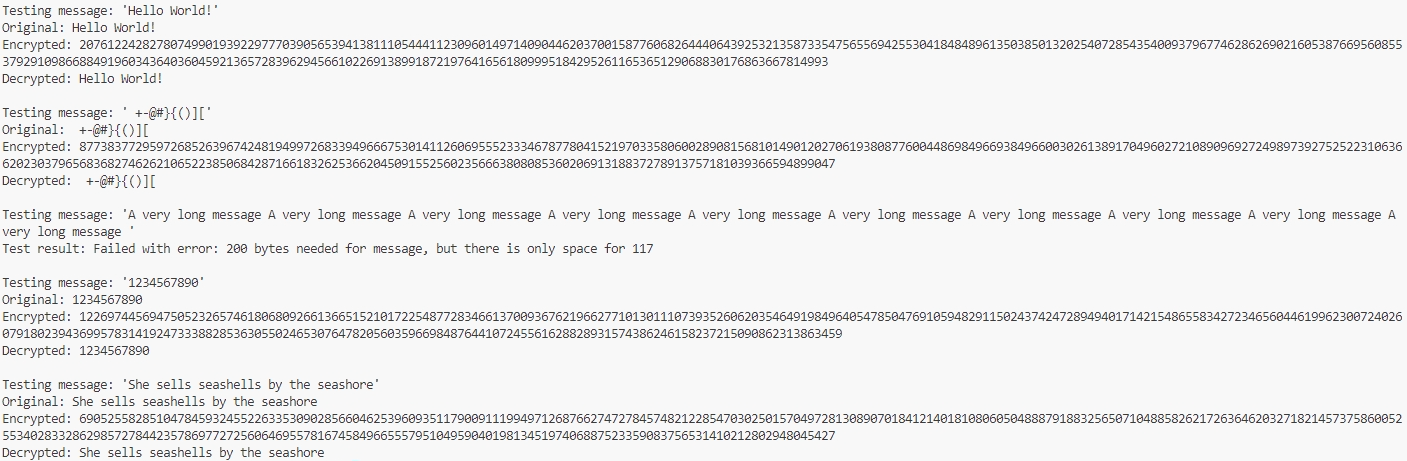
* Let message = 123.
* The public key is (*e, n*) = (17, 3233)
* *mod* *n* ⬄ *mod* 3233 = 855
* The ciphertext is then transmitted to the recipient.

3. Decryption:

* To decrypt = 855, use the private key *d* = 2753.
* Calculate *mod* 3233 = 123
* The original message is retrieved.

*II/ Python implementation and result:*





##### Figure 9: The result of source code for question 9

*III/ Explanation of python implementation:*

1. Encryption Function:

The “encrypt\_message” function is responsible for converting a text message into integer encryption, using a public key. This helps ensure that only the owner of the private key can decrypt and read the message

* **Operations:**
* **Encoding the Message**: The initial plaintext message is encoded into bytes using the “.encode()” method, as the RSA algorithm operates on bytes.
* **Public Key Encryption**: The bytes of the message are then encrypted using the public key. This ensures that the message can only be decrypted by the holder of the corresponding private key.
* **Converting Bytes to Integer**: The encrypted bytes are converted into a large integer. This transformation is useful for storing or transmitting the encrypted message in a numerical format, facilitating easier handling of the encrypted data.

2. Decryption Function:

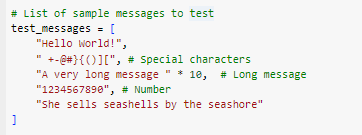
The “decrypt\_message” function is designed to decrypt an encrypted message using a private key, converting the encrypted numeric format back into the original plaintext message. Here’s how it operates:

* **Operations:**
* **Converting Integer to Bytes**: The encrypted message, now a large integer, is converted back into bytes. This is done using the “to\_bytes” method, calculating the necessary byte length from the bit length of the encrypted integer.
* **Private Key Decryption**: The bytes are decrypted using the private key. Only the private key has the ability to decrypt these bytes, ensuring that only the intended recipient can read the message.
* **Decoding to Text**: After decryption, the bytes are decoded back into a string using the “.decode()” method, allowing the recipient to read the original message.

3. Testing the Functions:

This function tests the entire encryption and decryption process:

* Each message is encrypted and then decrypted.
* The results are printed and checked to ensure the decrypted message matches the original.
* We use a series of test messages, including special characters, numbers, and a long repeating string, to verify the robustness of the encryption and decryption functions.



* **Conclusion:** With the above example, the basic application of the RSA encryption mechanism in Python can be adapted for secure communication systems with more complex decryption. The ability to efficiently encrypt and decrypt messages using RSA highlights its role in ensuring the security of encrypted messages.

*IV/ Analyze the efficiency and security of the implemented RSA cryptosystem:*

1. Efficiency:

* **Key Generation:** The key generation process (“rsa.newkeys**(1024**)”) generates an RSA key pair with a key length of 1024 bits. While this key length is suitable for demonstration purposes, for real-world applications, a key length of at least 2048 bits is recommended for better security. Generating RSA keys with longer key lengths can be computationally more expensive.
* **Encryption and Decryption:** The encryption and decryption functions (“encrypt\_message**”** and “decrypt\_message**”**) use the RSA encryption and decryption functions provided by the “rsa” library. These functions involve modular exponentiation operations, which can be computationally intensive, especially for large messages and key sizes. However, the provided implementation is efficient for small messages and key sizes.

2. Security:

* **Key Length**: The security of RSA depends on the length of the RSA keys used. The provided code uses a key length of 1024 bits, which is considered insufficient for modern cryptographic standards. For stronger security, it's recommended to use key lengths of at least 2048 bits or higher.
* **Prime Number Generation:** The **rsa** library handles prime number generation internally during key generation. Ensuring the randomness and quality of the generated prime numbers is crucial for maintaining security. The library is expected to use secure methods for prime number generation.
* **Cryptographic Operations:** The RSA encryption and decryption operations rely on the mathematical properties of modular exponentiation and the difficulty of the RSA problem. As long as the key length is sufficient, RSA is considered secure against attacks such as brute force and factoring. However, the security of RSA can be compromised if the key length is too short.
* **Testing:** The testing process verifies that the encryption and decryption operations produce the correct results for a set of sample messages. While this does not directly affect the security of RSA, it helps ensure the correctness of the implementation.
* In summary, the provided RSA cryptosystem implementation is suitable for demonstration purposes and small-scale applications. However, for real-world applications requiring security, it's essential to use longer key lengths (2048 bits or higher) and ensure the quality of prime number generation. Additionally, thorough testing and validation are necessary to ensure the correctness and security of the implementation.

*V/ Discuss the potential security threats and limitations of the RSA cryptosystem:*

Although RSA is considered secure when implemented correctly with a large key size, it is not immune to certain security threats and limitations:

* **Key Length:** One of the primary security considerations in RSA is the length of the keys used. As computational power increases over time, the security of RSA keys decreases. For example, while a key length of 1024 bits was once considered secure, it is now vulnerable to attacks using specialized hardware and distributed computing resources. As a result, it is recommended to use key lengths of at least 2048 bits or higher for RSA encryption to maintain security against potential threats.
* **Factorization Attacks:** The security of RSA is based on the difficulty of factoring the product of two large prime numbers. If an attacker can efficiently factor the modulus 𝑛n into its prime factors 𝑝p and 𝑞q, they can recover the private key and decrypt ciphertexts encrypted with the corresponding public key. Although factoring large numbers is computationally intensive and currently believed to be hard, advances in factorization algorithms and the development of quantum computers could potentially threaten the security of RSA in the future.
* **Timing Attacks:** RSA decryption operations can be vulnerable to timing attacks, where an attacker measures the time taken to perform decryption and uses this information to infer details about the private key.
* **Randomness and Entropy:** The security of RSA keys depends on the randomness and entropy used during key generation. Inadequate randomness or predictable key generation processes can lead to vulnerabilities. Ensuring proper entropy sources and using secure random number generators are essential to prevent attackers from predicting or guessing RSA keys.
* **Key Management and Distribution:** RSA encryption requires careful management and distribution of public and private keys. If an attacker gains unauthorized access to a private key, they can decrypt ciphertexts encrypted with the corresponding public key. Therefore, protecting private keys from unauthorized access and securely distributing public keys to intended recipients are critical aspects of RSA key management.
* **Cryptanalysis Attacks on RSA:** While RSA has been extensively studied and analyzed over the years, new cryptographic attacks and vulnerabilities may be discovered in the future. Continuous research and analysis are necessary to identify and address potential weaknesses in the RSA algorithm and its implementations.

*VI/ Conclude with recommendations for improving the RSA cryptosystem implementation:*

Depend on the limitation of RSA above, there are some recommendations to improve RSA implementation:

* Increase the key length to at least 2048 bits or higher to enhance security against brute-force and factorization attacks.
* Ensure that secure random number generators are used during key generation to generate prime numbers and random padding. Avoid using weak or predictable sources of randomness that could compromise the security of RSA keys.
* Follow best practices for key management, including securely storing private keys, rotating keys periodically, and restricting access to keys based on the principle of least privilege. Use hardware security modules (HSMs) or trusted execution environments (TEEs) to protect sensitive key material.
* Implement countermeasures to mitigate side-channel attacks, such as timing attacks and power analysis attacks. Use constant-time algorithms and cryptographic libraries that are resistant to timing variations to prevent attackers from exploiting timing discrepancies.
* Keep cryptographic libraries and dependencies up-to-date to ensure that known vulnerabilities and weaknesses are addressed promptly. Regularly review and apply security patches released by the library maintainers to mitigate potential risks.
* Perform regular security audits and code reviews of the RSA implementation to identify and address potential vulnerabilities, misconfigurations, or weaknesses. Engage security experts or conduct independent third-party security assessments to validate the security of the implementation.
* Develop transition plans to gradually migrate to longer RSA key lengths and stronger cryptographic algorithms as recommended by industry standards and cryptographic best practices. Consider the potential impact on compatibility and interoperability with existing systems and protocols.
* Consider implementing forward secrecy mechanisms, such as using ephemeral key pairs for key exchange protocols, to ensure that past encrypted communications remain secure even if long-term private keys are compromised in the future.

# REFERENCES

[1] D. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd ed. Addison-Wesley, 1997.  
[2] K. H. Rosen, Elementary Number Theory and Its Applications, 6th ed. Addison-Wesley, 2010.

[3] R. L. Graham, D. E. Knuth, and O. Patashnik, Concrete Mathematics: A Foundation for Computer Science, 2nd ed. Addison-Wesley, 1994.  
[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. MIT Press, 2009.

[5] K. A. Ross and C. R. B. Wright, Discrete Mathematics, 5th ed. Pearson, 2003.  
[6] M. Lipschutz, Schaum's Outline of Theory and Problems of Set Theory and Related Topics. McGraw-Hill, 1998.

[7] M. Sipser, Introduction to the Theory of Computation, 3rd ed. Cengage Learning, 2012.  
[8] P. Linz, An Introduction to Formal Languages and Automata, 5th ed. Jones & Bartlett Learning, 2011.

[9] J. B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," Proceedings of the American Mathematical Society, vol. 7, no. 1, pp. 48-50, 1956.  
[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. MIT Press, 2009.

[11] C. Hierholzer, "Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren," Mathematische Annalen, vol. 6, no. 1, pp. 30-32, 1873.  
[12] J. Diestel, Graph Theory, 5th ed. Springer, 2017.

[13] K. Appel and W. Haken, "Every Planar Map Is Four Colorable," Bulletin of the American Mathematical Society, vol. 82, no. 5, pp. 711-712, 1976.  
[14] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979.

[15] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, vol. 21, no. 2, pp. 120–126, 1978.

[16] P. W. Shor, "Algorithms for Quantum Computation: Discrete Logarithms and Factoring," in Proceedings of the 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124–134.

[17] D. Boneh, "Twenty Years of Attacks on the RSA Cryptosystem," Notices of the AMS, vol. 46, no. 2, pp. 203–213, 1999.

[18] A. May, "Using LLL-Reduction for Solving RSA and Factorization Problems: A Survey," LLL+25 Conference, 2007.

[19] M. Joye and G. Neven, Eds., Identity-Based Cryptography. IOS Press, 2009.

[20] J. Katz and Y. Lindell, Introduction to Modern Cryptography, 2nd ed. Chapman and Hall/CRC, 2014.

[21] D. Stinson and M. Paterson, Cryptography: Theory and Practice, 4th ed. CRC Press, 2018.

[22] N. Koblitz, "Elliptic Curve Cryptosystems," Mathematics of Computation, vol. 48, no. 177, pp. 203–209, 1987.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **MSSV** | **Full Name** | **Task** | **Evaluate (%)** |
| 1 | 522H0120 | Nguyễn Đình Việt Hoàng | + Do question 1, 2, 3, 4, 5.  + Design the report to fit the format of the faculty. | 100% |
| 2 | 522H0040 | Ngô Trung Tiến | + Do question 7 and items 1, 4, 5, 6 of question 9. | 100% |
| 3 | 522H0024 | Huỳnh Vũ Minh Hiếu | + Do question 6, question 8 and items 2, 3 of question 9. | 100% |