**DISCRETE MATHEMATICS**



**Session: 2023 – 2027**

**Submitted by:**

Sumaira Hafeez 2023-CS-01

Muhammad Ayan Goreja 2023-CS-05

**Supervised by:**

Prof. Dr. Waqas Ali

**Course:**

CSC-101 Discrete Mathematics

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**TABLE OF CONTENT**

Contents

[ **INDIVIDUAL CONTRIBUTION AND PLANNING:** 3](#_Toc154261577)

[**Question 1a** 3](#_Toc154261578)

[**Question 1b** 5](#_Toc154261579)

[**Question 2** 7](#_Toc154261580)

[ **Finding undirected Graph Degrees:** 7](#_Toc154261581)

[ **Finding Directed Graph Degrees:** 8](#_Toc154261582)

[ **Bipartatite Check:** 8](#_Toc154261583)

[ **Adjacency Matrix Creation:** 9](#_Toc154261584)

[ **Edge Listing Adjacency Matrix:** 10](#_Toc154261585)

[ **Incidence Matrix Creation:** 10](#_Toc154261586)

[**PART B** 11](#_Toc154261587)

[ **Finding Prime Factors:** 11](#_Toc154261588)

[ **Finding GCD:** 11](#_Toc154261589)

[ **Finding LCM:** 11](#_Toc154261590)

[ **Finding Bezout Coefficients:** 12](#_Toc154261591)

[ **Finding Inverse of Bezout Coefficients:** 12](#_Toc154261592)

[ **Finding RSA Encryption:** 13](#_Toc154261593)

[ **Finding Decryption Key:** 13](#_Toc154261594)

[ **Linkedn Id:** 13](#_Toc154261595)

[ **Source Codes:** 14](#_Toc154261596)

[**PART A** 14](#_Toc154261597)

[**Finding Undirected Edges** 14](#_Toc154261598)

[**Finding Directed Edges** 15](#_Toc154261599)

[**Bipartite Check** 17](#_Toc154261600)

[**Adjacency Matric Creation** 19](#_Toc154261601)

[**Edge Listing Adjacency Matrix** 19](#_Toc154261602)

[**Incidence Matrix Creation** 20](#_Toc154261603)

[**PART B** 21](#_Toc154261604)

[**Finding Primes** 21](#_Toc154261605)

[**Finding GCD** 22](#_Toc154261606)

[**Finding LCM** 23](#_Toc154261607)

[**Finding Bezout Coefficients** 24](#_Toc154261608)

[**Finding Bezout Inverse** 25](#_Toc154261609)

[**Finding RSA Encryption** 26](#_Toc154261610)

[**RSA Decryption** 28](#_Toc154261611)

**SEMESTER PROJECT**

# **INDIVIDUAL CONTRIBUTION AND PLANNING:**

The team members of this group are Sumaira Hafeez 2023-CS-01 and Muhammad Ayyan Goreja 2023-CS-05. Where in this project Muhammad Ayyan Goreja has contributed in writing programs of Part-A, Question number1a and the slides for video demonstration is also prepared by him. Whereas, Sumaira Hafeez has wrote programs for Number Theory Questions. Question 1b and report has also been submitted by her. Overall the project was completed by both members contributions and efforts.

**PART-A**

# **Question 1a**

**Game 1: Mario Kart**

In Mario Kart, graph theory can be used to model and analyze different aspects of the game. Here are a few ways Mario Kart can be represented through graph theory:

**1. Race Track Graph:**

* **Nodes:** represent key features such as turns, straightaways, item boxes, and shortcuts.
* **Edges:**  connect nodes to represent the racecourse, indicating the possible paths players can take.
* **Directed graphs:** represents the game state as the racing track is directed.

**1. Item Usage Graph:**

- **Nodes:** represent different items that can be obtained during the race.

* **Edges:** indicate relationships between items, such as defensive items countering offensive ones.
* **Pseudo Graph or a Multi Graph:** It can vary with the type of weapon the characters have.
* This graph can help analyze optimal item usage strategies and decision-making based on the current race situation.



**Game 2: Among Us**

In Among Us, graph theory can be applied to model and analyze player interactions and movements within the game.

**1. Player Movement Graph:**

**- Nodes** represent different rooms or locations on the map.

**- Edges** represent valid pathways or vents connecting these locations.

**-** Analyzing the graph can reveal optimal movement patterns, common meeting points, and potential areas for deception.

**2. Impostor Strategy Graph:**

**- Nodes** represent actions that impostors can take (e.g., sabotage, vent, fake tasks).

**- Edges** indicate dependencies or requirements for executing these actions.

**-** This graph can assist in understanding optimal impostor strategies and potential counter-strategies by crewmates.

**3. Game Tree:**

- The entire set of possible moves and resulting positions can be visualized as a tree. Each node represents a game state, and edges represent possible moves.



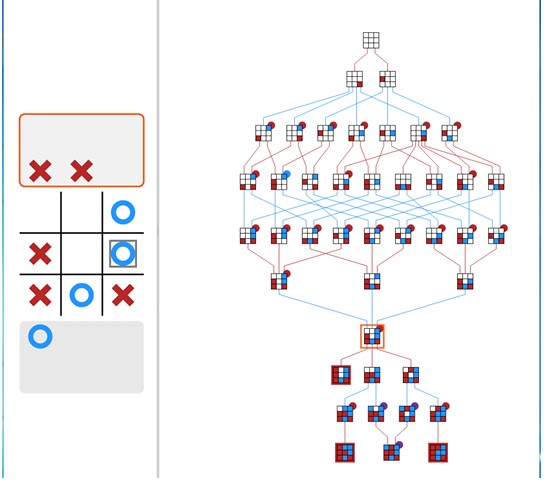
# **Question 1b**

**TIC-TAC-TOE AS A GRAPH**

* **INTRODUCTION:**

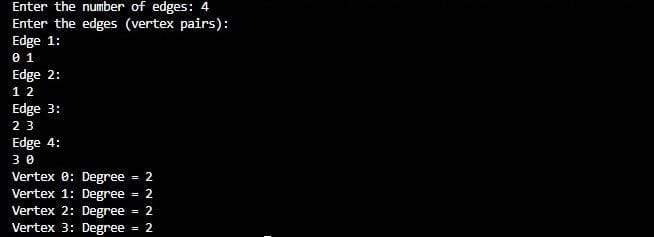
Tic-Tac-Toe, also known as “Xs and Os,” is a classic paper-and-pencil game that is often played by two players. The game is played on a 3x3 grid, and the objective is to be the first to form a horizontal, vertical, or diagonal line of three of one’s own symbols (either X or O). Players take turns marking an empty cell with their designated symbol, and the game continues until one player achieves the winning pattern or the entire grid is filled without a winner, resulting in a draw.

* **Available Views**
* **Neighborhood:** Shows all predecessors that lead to the current state as well as all possible continuations up to three levels.
* **Optimal.** Only shows nodes that represent states that may appear with best play of both players (see Fig. 2). Note that for board states that are not contained in the optimal subgraph, the state graph view only shows a single node.
* **Complete**. Always shows the complete graph.
* **Visual Features**
* **Node Visualization.** The interior of each node clearly displays the associated board state. Since the symbols X and O are not distinguishable at small sizes, we use solid cells of different colors (red and blue
* **Winning State.** Nodes are sometimes decorated with a small, circular icon on their upper right corner. The color of the icon indicates which party (red/blue) wins the game from this state assuming best play of both players. If there is no such icon, the game ends in a draw (again assuming best play of both players), a violet colored icon indicates a safe draw state which means that nobody can win independent of the continuation (without assuming best play). The winning states where determined by a graph analysis starting at the nodes representing the final states and walking up to the start node. Furthermore, a red/blue colored border around a node indicates a final state where one player has already won the game.
* **Edge Color.** The color of the outgoing edges of a node (incident on the nodes’ bottom) indicates which player is to move.

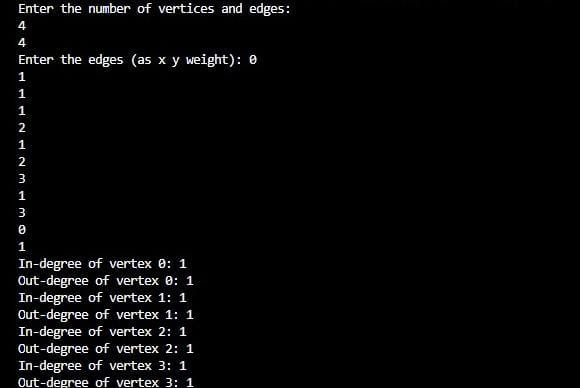


# **Question 2**

# **Finding undirected Graph Degrees:**

****

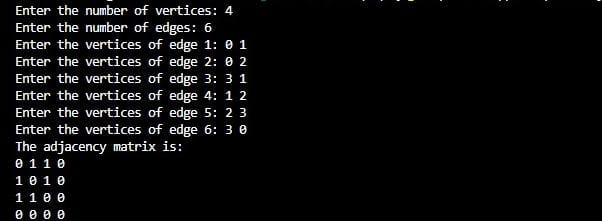
# **Finding Directed Graph Degrees:**

****

# **Bipartatite Check:**

****

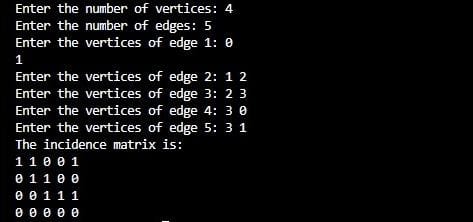
# **Adjacency Matrix Creation:**

****

# **Edge Listing Adjacency Matrix:**

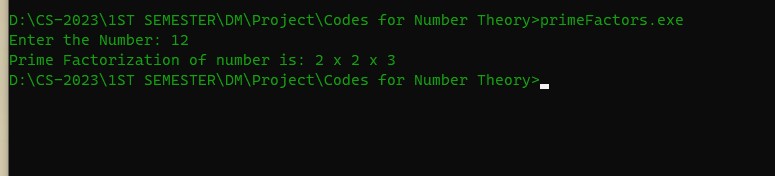
****

# **Incidence Matrix Creation:**

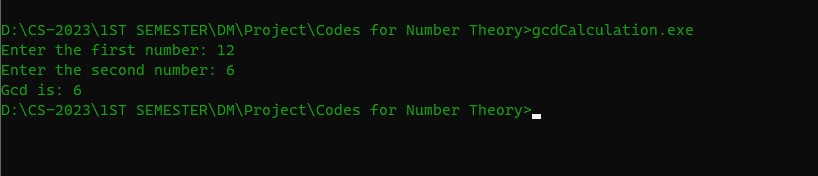
****

# **PART B**

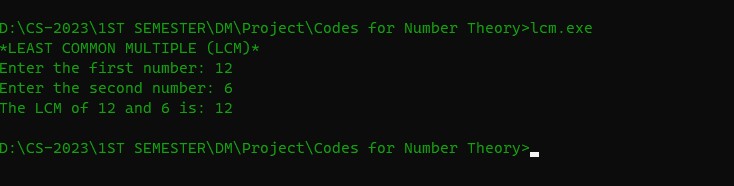
# **Finding Prime Factors:**



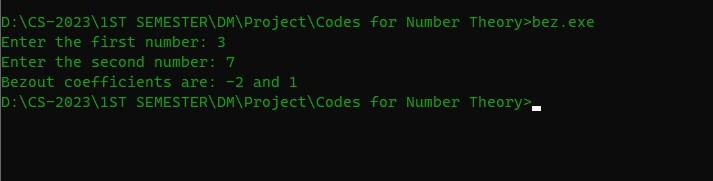
# **Finding GCD:**



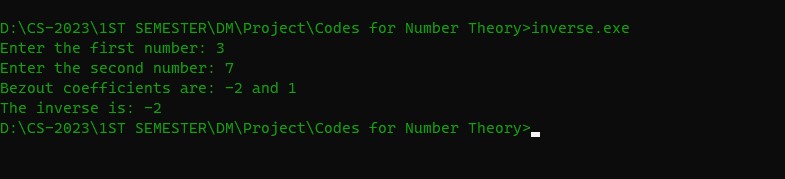
# **Finding LCM:**



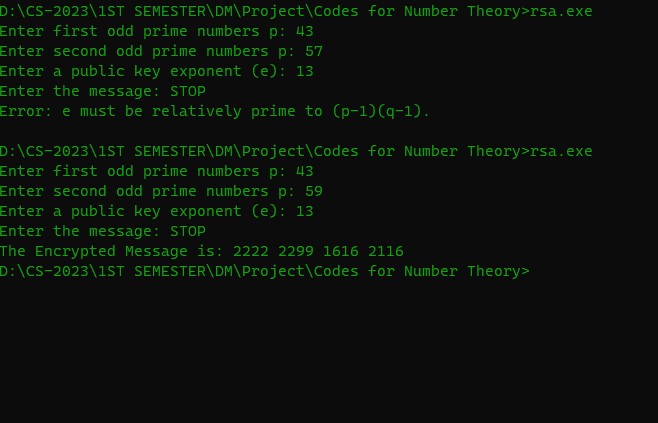
# **Finding Bezout Coefficients:**



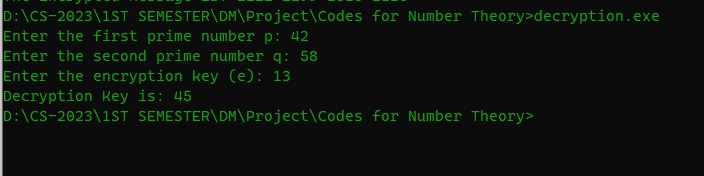
# **Finding Inverse of Bezout Coefficients:**



# **Finding RSA Encryption:**



# **Finding Decryption Key:**



## **Linkedn Id:**

Sumaira Hafeez:<https://www.linkedin.com/in/sumaira-hafeez-9974a6290/>

M. Ayan:<https://www.linkedin.com/in/muhammad-ayyan-goreja-89a561291>

# **Source Codes:**

# **PART A**

# **Finding Undirected Edges**

#include <iostream>

using namespace std;

const int MAX\_EDGES = 1000;

const int MAX\_VERTICES = 1000;

void calculateDegrees(int edges[MAX\_EDGES][2], int degrees[MAX\_VERTICES], int numEdges);

main()

{

    int numEdges;

    cout << "Enter the number of edges: ";

    cin >> numEdges;

    int edgesX;

    int edges[MAX\_EDGES][2];

    cout << "Enter the edges (vertex pairs):" << endl;

    for (int i = 0; i < numEdges; ++i)

    {

        cout << "Edge " << (i + 1) << ":\n";

        cin >> edges[i][0];

        cin >> edges[i][1];

        edgesX = edges[i][0]\*edges[i][1];

    }

    int degrees[MAX\_VERTICES];

    calculateDegrees(edges, degrees, numEdges);

}

void calculateDegrees(int edges[MAX\_EDGES][2], int degrees[MAX\_VERTICES], int edgesX)

{

    for (int i = 0; i < MAX\_VERTICES; ++i)

    {

        degrees[i] = 0;

    }

    for(int i = 0; i<edgesX; i++)

    {

        degrees[edges[i][0]]++;

        degrees[edges[i][1]]++;

        i++;

    }

    for (int i = 0; i < MAX\_VERTICES; ++i)

    {

        if (degrees[i] > 0)

        {

            cout << "Vertex " << i << ": Degree = " << degrees[i] +1 << endl;

        }

    }

}

# **Finding Directed Edges**

#include <iostream>

using namespace std;

void addEdge(int \*\*adj\_matrix, int x, int y, int weight);

main()

{

    int n, m;

    cout << "Enter the number of vertices and edges:\n";

    cin >> n >> m;

    int \*\*adj\_matrix = new int \*[n];

    for (int i = 0; i < m; i++)

    {

        adj\_matrix[i] = new int[n];

        for (int j = 0; j < n; j++)

        {

            adj\_matrix[i][j] = 0;

        }

    }

    cout << "Enter the edges (as x y weight): ";

    for (int i = 0; i < m; i++)

    {

        int x, y, weight;

        cin >> x >> y >> weight;

        addEdge(adj\_matrix, x, y, weight);

    }

    int \*in\_degrees = new int[n];

    int \*out\_degrees = new int[n];

    for (int i = 0; i < n; i++)

    {

        in\_degrees[i] = 0;

        out\_degrees[i] = 0;

    }

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (adj\_matrix[i][j] != 0)

            {

                out\_degrees[i] += 1;

                in\_degrees[j] += 1;

            }

        }

    }

    for (int i = 0; i < n; i++)

    {

        cout << "In-degree of vertex " << i << ": " << in\_degrees[i] << endl;

        cout << "Out-degree of vertex " << i << ": " << out\_degrees[i] << endl;

    }

    for (int i = 0; i < n; i++)

    {

        delete[] adj\_matrix[i];

    }

    delete[] adj\_matrix;

    delete[] in\_degrees;

    delete[] out\_degrees;

}

void addEdge(int \*\*adj\_matrix, int x, int y, int weight)

{

    adj\_matrix[x][y] = weight;

}

## **Bipartite Check**

#include <iostream>

using namespace std;

bool isBipartite(int \*\*adj\_matrix, int n, int start);

void addEdge(int \*\*adj\_matrix, int x, int y, int weight);

main()

{

    int n, m;

    cout << "Enter the number of vertices and edges: ";

    cin >> n >> m;

    int \*\*adj\_matrix = new int \*[n];

    for (int i = 0; i < n; i++)

    {

        adj\_matrix[i] = new int[n];

        for (int j = 0; j < n; j++)

        {

            adj\_matrix[i][j] = 0;

        }

    }

    cout << "Enter the edges (as x y weight):\n";

    for (int i = 0; i < n; i++)

    {

        int x, y, weight;

        cin >> x >> y >> weight;

        addEdge(adj\_matrix, x, y, weight);

    }

    if (isBipartite(adj\_matrix, n, 0))

    {

        cout << "The graph is bipartite." << endl;

    }

    else

    {

        cout << "The graph is not bipartite." << endl;

    }

    for (int i = 0; i < n; i++)

    {

        delete[] adj\_matrix[i];

    }

    delete[] adj\_matrix;

}

bool isBipartite(int \*\*adj\_matrix, int n, int start)

{

    int \*colors = new int[n];

    for (int i = 0; i < n; i++)

    {

        colors[i] = -1;

    }

    colors[start] = 1;

    for (int i = 0; i < n; i++)

    {

        if (colors[i] == -1)

        {

            continue;

        }

        for (int j = 0; j < n; j++)

        {

            if (adj\_matrix[i][j] == 1)

            {

                if (colors[j] == -1)

                {

                    colors[j] = 1 - colors[i];

                }

                else if (colors[j] == colors[i])

                {

                    delete[] colors;

                    return false;

                }

            }

        }

    }

    delete[] colors;

    return true;

}

void addEdge(int \*\*adj\_matrix, int x, int y, int weight)

{

    adj\_matrix[x][y] = weight;

    adj\_matrix[y][x] = weight;

}

# **Adjacency Matric Creation**

#include <iostream>

using namespace std;

main()

{

    int vertices, edges;

    cout << "Enter the number of vertices: ";

    cin >> vertices;

    cout << "Enter the number of edges: ";

    cin >> edges;

    int adjMatrix[100][100] = {0};

    for (int i = 0; i < edges; i++)

    {

        int v1, v2;

        cout << "Enter the vertices of edge " << i + 1 << ": ";

        cin >> v1 >> v2;

        adjMatrix[v1 - 1][v2 - 1] = 1;

        if (adjMatrix[v2 - 1][v1 - 1] == 0)

        {

            adjMatrix[v2 - 1][v1 - 1] = 1;

        }

    }

    cout << "The adjacency matrix is: " << endl;

    for (int i = 0; i < vertices; i++)

    {

        for (int j = 0; j < vertices; j++)

        {

            cout << adjMatrix[i][j] << " ";

        }

        cout << endl;

    }

}

# **Edge Listing Adjacency Matrix**

#include <iostream>

using namespace std;

void listEdges(int graph[100][100], int vertices);

main()

{

    int vertices;

    cout << "Enter the number of vertices: ";

    cin >> vertices;

    int graph[100][100] = {0};

    cout << "Enter the adjacency matrix (1-based index): " << endl;

    for (int i = 0; i < vertices; i++)

    {

        for (int j = 0; j < vertices; j++)

        {

            cin >> graph[i][j];

        }

    }

    listEdges(graph, vertices);

}

void listEdges(int graph[100][100], int vertices)

{

    bool edgeVisited[100][100] = {false};

    for (int i = 0; i < vertices; i++)

    {

        for (int j = i + 1; j < vertices; j++)

        {

            if (graph[i][j] > 0 && !edgeVisited[i][j])

            {

                edgeVisited[i][j] = true;

                edgeVisited[j][i] = true;

                cout << (i + 1) << " - " << (j + 1) << " : " << graph[i][j] << endl;

            }

        }

    }

}

# **Incidence Matrix Creation**

#include <iostream>

using namespace std;

main()

{

    int vertices, edges;

    cout << "Enter the number of vertices: ";

    cin >> vertices;

    cout << "Enter the number of edges: ";

    cin >> edges;

    int incidenceMatrix[100][100] = {0};

    for (int i = 0; i < edges; i++)

    {

        int v1, v2;

        cout << "Enter the vertices of edge " << i + 1 << ": ";

        cin >> v1 >> v2;

        incidenceMatrix[v1 - 1][i] = 1;

        incidenceMatrix[v2 - 1][i] = 1;

    }

    cout << "The incidence matrix is: " << endl;

    for (int i = 0; i < vertices; i++)

    {

        for (int j = 0; j < edges; j++)

        {

            cout << incidenceMatrix[i][j] << " ";

        }

        cout << endl;

    }

}

# **PART B**

## **Finding Primes**

#include<iostream>

using namespace std;

void calculatePrimeFactors(int num);

main()

{   int number;

    cout<<"Enter the Number: ";

    cin>>number;

    cout<<"Prime Factorization of number is: ";

    calculatePrimeFactors(number);

}

void calculatePrimeFactors(int num)

{

    //check if it is a factor of 2

    while(num % 2 == 0)

    {

        cout<<"2";

        num = num/2;

        if(num>1)

        {

            cout<<" x ";

        }

    }

    //check if it is factor of odd primes

    int i = 3;

    while(num!=1)

    { while(num % i == 0)

        {   num = num/i;

            if(i<0)

            {

                i = i \* (-1);

                cout<<i;

            }

            else if(i>0)

            {

               cout<<i;

            }

            if(num!= 1)

        {

            cout<<" x ";

        }

        }i=i+2;

    }

}

## **Finding GCD**

#include<iostream>

using namespace std;

int calculateGCD(int num1, int num2);

main()

{

    int num1, num2;

    cout<<"Enter the first number: ";

    cin>>num1;

    cout<<"Enter the second number: ";

    cin>>num2;

    cout<<"Gcd is: "<<calculateGCD(num1, num2);

}

int calculateGCD(int num1, int num2)

{

    while(num2!=0)

    {

        int temp = num2;

        num2 = num1%num2;

        num1 = temp;

    }

    return num1;

}

# **Finding LCM**

#include <iostream>

using namespace std;

int calculateGCD(int num1, int num2);

int calculateLCM(int a, int b);

main()

{

    int num1, num2;

    cout << "\*LEAST COMMON MULTIPLE (LCM)\*" << endl;

    cout << "Enter the first number: ";

    cin >> num1;

    cout << "Enter the second number: ";

    cin >> num2;

    int lcm = calculateLCM(num1, num2);

    cout << "The LCM of " << num1 << " and " << num2 << " is: " << lcm << endl;

}

int calculateGCD(int num1, int num2)

{

   while(num2!=0)

    {

        int temp = num2;

        num2 = num1%num2;

        num1 = temp;

    }

    return num1;

}

int calculateLCM(int a, int b)

{

    int gcd = calculateGCD(a, b);

    return (a \* b) / gcd;

}

# **Finding Bezout Coefficients**

#include<iostream>

#include<tuple>

using namespace std;

pair<int,int> calculateBezout(int num1, int num2, int s, int t);

tuple<int,int,int> calculateGCD(int num1, int num2);

main()

{

    int num1, num2, s, t;

    cout<<"Enter the first number: ";

    cin>>num1;

    cout<<"Enter the second number: ";

    cin>>num2;

    std::tie(s,t) = calculateBezout(num1, num2,s,t);

    cout<<"Bezout coefficients are: "<<s<<" and "<<t;

}

pair<int,int> calculateBezout(int num1, int num2, int s, int t)

{   int gcd;

    std::tie(gcd,s,t)=calculateGCD(num1, num2);

    return std::make\_pair(s,t);

}

tuple<int,int,int> calculateGCD(int num1, int num2)

{   int g,x,y;

    if(num1==0)

    {

        return std::make\_tuple(num2, 0, 1);

    }

    else

    {

        std::tie(g,x,y) = calculateGCD(num2%num1, num1);

        return std::make\_tuple(g,y-(num2/num1)\*x, x);

    }

}

# **Finding Bezout Inverse**

#include<iostream>

#include<tuple>

using namespace std;

pair<int,int> calculateBezout(int num1, int num2, int s, int t);

tuple<int,int,int> calculateGCD(int num1, int num2);

main()

{

    int num1, num2, s, t;

    cout<<"Enter the first number: ";

    cin>>num1;

    cout<<"Enter the second number: ";

    cin>>num2;

    std::tie(s,t) = calculateBezout(num1, num2,s,t);

    cout<<"Bezout coefficients are: "<<s<<" and "<<t<<endl;

    cout<<"The inverse is: ";

    //formula is ax=bmodm where a = num1 , b = 1 , x = s and t and m= num2;

    //if m%ax-1 == 0 then it is a inverse

    int inverse1 = num2%((num1\*s)-1);

    int inverse2 = num2%((num1\*t)-1);

    if(inverse1 == 0)

    {

        cout<<s<<"  ";

    }

    if(inverse2 == 0)

    {

        cout<<t<<"  ";

    }

}

pair<int,int> calculateBezout(int num1, int num2, int s, int t)

{   int gcd;

    std::tie(gcd,s,t)=calculateGCD(num1, num2);

    return std::make\_pair(s,t);

}

tuple<int,int,int> calculateGCD(int num1, int num2)

{   int g,x,y;

    if(num1==0)

    {

        return std::make\_tuple(num2, 0, 1);

    }

    else

    {

        std::tie(g,x,y) = calculateGCD(num2%num1, num1);

        return std::make\_tuple(g,y-(num2/num1)\*x, x);

    }

}

# **Finding RSA Encryption**

#include<iostream>

#include<cmath>

#include<string>

using namespace std;

int gcd(int num1, int num2);

string rsaMessage(string message, int n, int e);

int modularPow(int base, int exponent, int modulus);

int main()

{

    int p,q,n,e;

    string message;

    cout<<"Enter first odd prime numbers p: ";

    cin>>p;

    cout<<"Enter second odd prime numbers p: ";

    cin>>q;

    cout<<"Enter a public key exponent (e): ";

    cin>>e;

    cout<<"Enter the message: ";

    cin.ignore();

    getline(cin, message);

    //First we have to calculate n=pq

    n = p\*q;

    int product = (p-1)\*(q-1);

    //checking if it is relatively prime otherwise generate error

    if(gcd(n, product)!=1)

    {

        std::cerr << "Error: e must be relatively prime to (p-1)(q-1)." << std::endl;

        return 1;

    }

    string encryptedMessage=rsaMessage(message,n,e);

    cout<<"The Encrypted Message is: "<<encryptedMessage;

   return 0;

}

int gcd(int num1, int num2)

{

     while(num2!=0)

    {

        int temp = num2;

        num2 = num1%num2;

        num1 = temp;

    }

    return num1;

}

string rsaMessage(string message, int n, int e)

{   int charValue;

string encrypted\_message;

    //Its Formula is C=M^e mod n where M is message and c is encrypted message

    for (char ch : message)

    {

        // Convert each character to its corresponding two-digit number (A=00, B=01, C=02, ..., Z=25)

        charValue = static\_cast<int>(ch) - 'A';

        int cypherText = modularPow(charValue,e,n);

        string encrypted\_digit = (cypherText < 10) ? "0" + to\_string(cypherText) : to\_string(cypherText);

        // Append the encrypted character to the result string

         encrypted\_message += encrypted\_digit + " ";

    }

    return encrypted\_message;

}

// Function to calculate modular exponentiation (a^b mod m)

int modularPow(int base, int exponent, int modulus) {

    int result = 1;

    base = base % modulus;

    while (exponent > 0) {

        if (exponent % 2 == 1) {

            result = (result \* base) % modulus;

        }

        exponent = exponent >> 1;

        base = (base \* base) % modulus;

    }

    return result;

}

# **RSA Decryption**

#include<iostream>

#include<tuple>

using namespace std;

int decryptionKey(int p, int q, int e);

int gcd(int num1, int num2);

int modularInverse(int num1, int num2);

int main()

{

    int p,q,e;

    cout<<"Enter the first prime number p: ";

    cin>>p;

    cout<<"Enter the second prime number q: ";

    cin>>q;

    cout<<"Enter the encryption key (e): ";

    cin>>e;

    int d = decryptionKey(p,q,e);

    cout<<"Decryption Key is: "<<d;

    return 0;

}

int modularInverse(int a, int m) {

    int m0 = m, t, q;

    int x0 = 0, x1 = 1;

    if (m == 1) {

        return 0;

    }

    while (a > 1) {

        // q is quotient

        q = a / m;

        t = m;

        // m is remainder now, process same as Euclid's algorithm

        m = a % m, a = t;

        t = x0;

        x0 = x1 - q \* x0;

        x1 = t;

    }

    // Make x1 positive

    if (x1 < 0) {

        x1 += m0;

    }

    return x1;

}

int gcd(int num1, int num2)

{

     while(num2!=0)

    {

        int temp = num2;

        num2 = num1%num2;

        num1 = temp;

    }

    return num1;

}

int decryptionKey(int p, int q, int e)

{   int n = p\*q;

    int product =(p-1) \* (q-1);

    int d;

    d = modularInverse(e, product);

    return d;

}