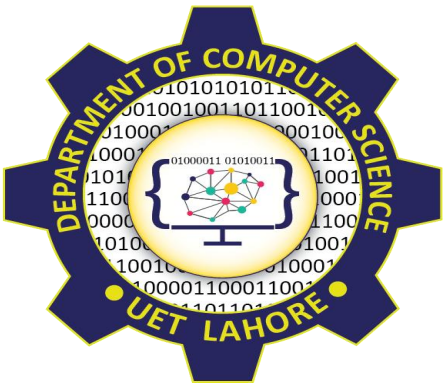
DISCRETE MATHEMATICS PROJECT REPORT



Session 2023 - 2027

Submitted by:
Hannan Mushtaq 2023-CS-85
Muhammad Ahmad 2023-CS-69

Supervised by:
Sir Waqas Ali

Course:
CSC-101 Discrete Mathematics

Department of Computer Science
University of Engineering and Technology
Lahore Pakistan

TABLE OF CONTENTS

PART A: GRAPHS

Question 1: Analysis of Games Using Graph Theory

Game # 1: FOOTBALL

In the context of graph theory applied to football:

Vertices:

Players, ball, goal posts are represented as vertices in the graph.

Edges:

Edges between players represent interactions like passes, collaborations, or marking. Edges indicate ball movement and successful passes. Goal attempts are represented by edges connecting players to goal posts.

Analysis:

Study player collaboration, passing networks, and overall team connectivity. Analyze dynamic graphs to understand game progression and strategic decision points. Identify central players and assess team coordination through graph connectivity.

Strategic Insights:

Evaluate connectivity for possession control and strategic plays. Study transitions between different game situations represented by vertices. Apply graph theory to gain insights into player dynamics and team strategies. This approach allows for a structured analysis of player interactions, team coordination, and strategic elements in football using graph theory concepts.

Game # 2: SNOOKER

In the context of graph theory applied to snooker:

Vertices:

Balls (red, colored), cue ball, and pockets are represented as vertices in the graph.

Edges:

Cue ball movements are depicted with edges indicating shots or movements made by the player. Edges connect balls to pockets to represent successful shots and the movement of balls into pockets.

Analysis:

Study shot sequences, player strategies, and the dynamic evolution of the game through the graph. Analyze the connectivity of balls to understand how easily players can transition between different shots.

Strategic Insights:

Identify critical balls and strategic plays to achieve victory. Study how players strategically position balls to create difficult situations for opponents. Assess the evolving graph to understand decision points and tactical maneuvers. This approach provides a structured way to analyze player decisions, strategies, and the unfolding complexity of the game as it progresses in snooker using graph theory concepts

Analysis of Tic-Tac-Toe Game

In the context of graph theory applied to Tic-Tac-Toe:

Vertices:

Each unique state of the Tic-Tac-Toe board, including different placements of Xs and Os, is represented as a vertex in the graph.

Edges:

Directed edges represent valid moves or transitions between different board states. For instance, if an empty cell allows placing an X or an O, there is a directed edge from the current state to the next state after making that move.

Analysis:

Study the graph to understand the dynamics of the game, potential outcomes, and winning strategies. Analyze winning states, draw states, and losing states within the graph.

Strategic Insights:

Identify winning paths and strategic moves that lead to victory. Understand how to avoid losing states by blocking the opponent's winning moves. Analyze the graph to find optimal strategies and decision points in the game. This graph-based approach provides a structured way to explore the possibilities of Tic-Tac-Toe, study optimal strategies, and understand the dynamics of the game using graph theory concepts.

Question 2: Programming Tasks In Graph Theory

A. Undirected Graph Degrees

```
main()
{
    int totalVertices, totalEdges;
    cout << "Enter the maximum number of vertices: ";
    cin >> totalVertices;
    cout << "Enter the number of edges: ";
    cin >> totalEdges;
    int edges[totalEdges][2];
    int vertices[totalVertices] = {0};
    cout << "Enter the edge pairs: " << endl;
    for (int i = 0; i < totalEdges; ++i)
    {   cout << "Enter vertex 1 of edge " << i + 1 << ": ";
        cin >> edges[i][0];
        cout << "Enter vertex 2 of edge " << i + 1 << ": ";
        cin >> edges[i][1];
    }
    for (int i = 0; i < totalEdges; ++i)
    {   vertices[edges[i][0] - 1]++;
        vertices[edges[i][1] - 1]++;
    }
    cout << "Degrees of vertices:" << endl;
    for (int i = 0; i < totalVertices; ++i)
    {   if (vertices[i] > 0)
        {   cout << "Vertex " << i + 1 << " has degree " << vertices[i] << endl;
        }
    }
}
```

B. Directed Graph Degrees

```
main()
{   int totalVertices;
    cout << "Enter the number of vertices: ";
    cin >> totalVertices;
    int edges[totalVertices][2];
    int edgeCount = 0;
    cout << "Enter the directed edges from vertex 1 to vertex 2 (Enter values less than 1 to finish): " << endl;
    int vertex1, vertex2;
    while (true)
    {   cout << "Enter vertex 1 of edge " << edgeCount + 1 << ": ";
        cin >> vertex1;
        cout << "Enter vertex 2 of edge " << edgeCount + 1 << ": ";
        cin >> vertex2;
        if (vertex1 < 1 || vertex2 < 1)
        {   break;
        }
        if (vertex1 > totalVertices || vertex2 > totalVertices)
        {   cout << "Invalid vertices. Please enter vertices between 1 and " << totalVertices << "." << endl;
            return 0;
        }
        edges[edgeCount][0] = vertex1;
        edges[edgeCount][1] = vertex2;
        edgeCount++;
    }
    int inDegrees[totalVertices];
    int outDegrees[totalVertices];
    for (int i = 0; i < edgeCount; ++i)
    {   outDegrees[edges[i][0] - 1]++;
        inDegrees[edges[i][1] - 1]++;
    }
    cout << "Vertex degrees: " << endl;
    for (int i = 0; i < totalVertices; ++i)
    {   cout << "Vertex " << i + 1 << " has in-degree " << inDegrees[i]
             << " and out-degree " << outDegrees[i] << endl;
    }
}
```

C. Bipartite Graph Check

```
bool isBipartite(int edges[][2], int totalEdges, int totalVertices)
{   int color[totalVertices];
    for (int i = 0; i < totalVertices; i++)
    {   color[i] = 1;
    }
    for (int i = 0; i < totalEdges; ++i)
    {   int u = edges[i][0];
        int v = edges[i][1];
        if (color[u] == 1)
        {    color[u] = 0;
        }
        if (color[v] == 1)
        {   color[v] = 2 - color[u];
        }
        else if (color[v] == color[u])
        {   return false;
        }
    }
    return true;
}
```

D. Adjacency Matrix Creation

```
main()
{    int totalVertices;
     cout << "Enter the number of vertices: ";
     cin >> totalVertices;
     int totalEdges;
     cout << "Enter the number of edges: ";
     cin >> totalEdges;
     int edges[totalEdges][2];
     cout << "Enter the edges (vertex pairs):" << endl;
     for (int i = 0; i < totalEdges; ++i)
     {    cout << "Enter vertex 1 of edge " << i + 1 << ": ";
          cin >> edges[i][0];
          cout << "Enter vertex 2 of edge " << i + 1 << ": ";
          cin >> edges[i][1];
     }
     int adjacencyMatrix[totalVertices][totalVertices];
     for (int i = 0; i < totalVertices; i++)
     {    for (int j = 0; j < totalVertices; j++)
          {     adjacencyMatrix[i][j] = 0;
          }
     }
     for (int i = 0; i < totalEdges; ++i)
     {    adjacencyMatrix[edges[i][0] - 1][edges[i][1] - 1] = 1;
          adjacencyMatrix[edges[i][1] - 1][edges[i][0] - 1] = 1;
     }
     cout << "Adjacency Matrix:" << endl;
     for (int i = 0; i < totalVertices; ++i)
     {    for (int j = 0; j < totalVertices; ++j)
          {     cout << adjacencyMatrix[i][j] << " ";
          }
          cout << endl;
     }
}
```

E. Edge Listing From Adjacency Matrix

```
main()
{   int totalVertices;
    cout << "Enter the number of vertices: ";
    cin >> totalVertices;
    int adjacencyMatrix[totalVertices][totalVertices];
    cout << "Enter the adjacency matrix: " << endl;
    for (int i = 0; i < totalVertices; ++i)
    {    for (int j = 0; j < totalVertices; ++j)
         {   cout << "Matrix[" << i + 1 << "][" << j + 1 << "]: ";
             cin >> adjacencyMatrix[i][j];
         }
    }
    cout << "Edges and Counts:" << endl;
    for (int i = 0; i < totalVertices; ++i)
    {   for (int j = 0; j < totalVertices; ++j)
         {    if (adjacencyMatrix[i][j] > 0)
              {   cout << "Edge: " << i + 1 << " to " << j + 1 << "\tCount: " << adjacencyMatrix[i][j] << endl;
              }
         }
    }
}
```

F. Incidence Matrix Creation

```
main()
{   int totalVertices, totalEdges;
    cout << "Enter the number of vertices: ";
    cin >> totalVertices;
    cout << "Enter the number of edges: ";
    cin >> totalEdges;
    int edges[totalEdges][2];
    int Frequency[totalEdges];
    cout << "Enter the vertices and frequency of each edge:" << endl;
    for (int i = 0; i < totalEdges; ++i)
    {   cout << "Enter vertex 1 of edge " << i + 1 << ": ";
        cin >> edges[i][0];
        cout << "Enter vertex 2 of edge " << i + 1 << ": ";
        cin >> edges[i][1];
        cout << "Frequency of edge " << i + 1 << ": ";
        cin >> Frequency[i];
    }
    int incidenceMatrix[totalVertices][totalEdges];
    for (int i = 0; i < totalVertices; i++)
    {   for (int j = 0; j < totalEdges; j++)
        {    incidenceMatrix[i][j] = 0;
        }
    }
    for (int i = 0; i < totalEdges; ++i)
    {   incidenceMatrix[edges[i][0] - 1][i] = Frequency[i];
        incidenceMatrix[edges[i][1] - 1][i] = Frequency[i];
    }
    cout << "Incidence Matrix:" << endl;
    for (int i = 0; i < totalVertices; ++i)
    {   for (int j = 0; j < totalEdges; ++j)
        {    cout << incidenceMatrix[i][j] << " ";
        }
        cout << endl;
    }
}
```

PART B: NUMBER THEORY

Question 3: Programming Tasks In Number Theory

A. Prime Factorization

```
main()
{   int num;
    while (true)
    {   cout << "Enter a positive integer to be factorized: ";
        cin >> num;
        if (num > 1)
        {   primeFactors(num);
            break;
        }
        else
        {   cout << "Please enter a positive integer greater than 1." << endl;
        }
        cout << "Press Any Key To Continue... ";
        getch();
        system("cls");
    }
}

void primeFactors(int num)
{   cout << "Prime factorization of " << num << ": ";
    for (int i = 2; i <= num; ++i)
    {   while (num % i == 0)
        {   cout << i << " ";
            num /= i;
        }
    }
    cout << endl;
}
```

B. Euclidean Algorithm For GCD

```
main()
{   int num1, num2;
    while (true)
    {   cout << "Enter the first positive integer: ";
        cin >> num1;
        cout << "Enter the second positive integer: ";
        cin >> num2;
        if (num1 > 0 && num2 > 0)
        {   int result = GCD(num1, num2);
            cout << "GCD of " << num1 << " and " << num2 << " is: " << result << endl;
            break;
        }
        else
        {   cout << "Please enter positive integers greater than 0." << endl;
        }
        cout << "Press Any Key To Continue... ";
        getch();
        system("cls");
    }
}

int GCD(int num1, int num2)
{   while (num2 != 0)
    {   int temp = num2;
        num2 = num1 % num2;
        num1 = temp;
    }
    return num1;
}
```

C. LCM Calculation

```
int GCD(int num1, int num2)
{   while (num2 != 0)
    {   int temp = num2;
        num2 = num1 % num2;
        num1 = temp;
    }
    return num1;
}
int LCM(int num1, int num2)
{   if (num1 == 0 || num2 == 0)
    {   return 0;
    }
    return abs(num1 * num2) / GCD(num1, num2);
}
```

D. Bezout Coefficients

```
main()
{   int num1, num2, s, t;
    while (true)
    {   cout << "Enter the first positive integer: ";
        cin >> num1;
        cout << "Enter the second positive integer: ";
        cin >> num2;
        if (num1 >= 0 && num2 >= 0)
        {   int gcdResult = extendedGCD(num1, num2, s, t);
            cout << "GCD of " << num1 << " and " << num2 << " is: " << gcdResult << endl;
            cout << "Bezout coefficients for " << num1 << " and " << num2 << " are s = " << s << ", t = " << t << endl;
            break;
        }
        else
        {   cout << "Please enter positive integers." << endl;
        }
        cout << "Press Any Key To Continue... ";
        getch();
        system("cls");
    }
}
int extendedGCD(int num1, int num2, int &s, int &t)
{   if (num1 == 0)
    {   s = 0;
        t = 1;
        return num2;
    }
    int s1, t1;
    int gcd = extendedGCD(num2 % num1, num1, s1, t1);
    s = t1 - (num2 / num1) * s1;
    t = s1;
    return gcd;
}
```

E. Modular Inverse

```
int extendedGCD(int num1, int num2, int &s, int &t)
{   if (num1 == 0)
    {   s = 0;
        t = 1;
        return num2;
    }
    int s1, t1;
    int gcd = extendedGCD(num2 % num1, num1, s1, t1);
    s = t1 - (num2 / num1) * s1;
    t = s1;
    return gcd;
}

int Inverse(int num1, int num2)
{   int s, t;
    int gcdResult = extendedGCD(num1, num2, s, t);
    if (gcdResult == 1)
    {   return ((s % num2) + num2) % num2;
    }
    else
    {   cout << "Inverse does not exist as " << num1 << " and " << num2 << " are not relatively prime with gcd not equal to 1." << endl;
        return -1;
    }
}
```

F. RSA Encryption

```
main()
{   int p, q, n, e, message;
    cout << "Enter the first prime number (p): ";
    cin >> p;
    cout << "Enter the second prime number (q): ";
    cin >> q;
    n = p * q;
    do
    {   cout << "Enter the public exponent (e): ";
        cin >> e;
    }while (!arePrime(e, (p - 1) * (q - 1)));
    cout << "Enter the message to be encrypted: ";
    cin >> message;
    int encryptedMessage = Encrypted(message, e, n);
    cout << "Encrypted message: " << encryptedMessage << endl;
}
bool arePrime(int e, int b)
{   for (int i = 2; i <= min(e, b); ++i)
    {   if (e % i == 0 && b % i == 0)
        {   return false;
        }
    }
    return true;
}
int Encrypted(int message, int e, int n)
{   int result = 1;
    message = message % n;
    while (e > 0)
    {   if (e % 2 == 1)
        {   result = (result * message) % n;
        }
        e = e >> 1;
        message = (message * message) % n;
    }
    return result;
}
```

G. RSA Decryption

```
main()
{   int p, q, n, e;
    cout << "Enter prime number p: ";
    cin >> p;
    cout << "Enter prime number q: ";
    cin >> q;
    n = p * q;
    do
    {   cout << "Enter the public exponent (e): ";
        cin >> e;
    }while (!arePrime(e, (p - 1) * (q - 1)));
    int d = Decrypted(e, (p - 1) * (q - 1));
    cout << "Decryption key (d): " << d << endl;
}
int Decrypted(int e, int m)
{   int s, t;
    int gcdResult = extendedGCD(e, m, s, t);
    if (gcdResult == 1)
    {   return ((s % m) + m) % m;
    }
    else
    {   cout << "Inverse does not exist as " << e << " and " << m << " are not relatively prime with gcd not equal to 1." << endl;
        return -1;
    }
}
```

LinkedIn Video Demonstration Links:

- Hannan Mushtaq - https://www.linkedin.com/posts/hannan-mushtaq-8341222a4_thrilled-to-showcase-my-contribution-to-the-activity-7147506417434861568-lq19?utm_source=share&utm_medium=member_android

- Muhammad Ahmad - https://www.linkedin.com/posts/muhammad-ahmad-5217702a7_thrilled-to-showcase-my-contribution-to-the-activity-7147514025952559104-A6tf?utm_source=share&utm_medium=member_android

Individual Contribution & Project Planning:

Hannan Mushtaq

- Programming tasks of the graph theory
- Video Demonstration
- PowerPoint Presentation Slides For Video Demonstration
- Project Report

Muhammad Ahmad

- Game Questions of Graph Theory
- Programming Tasks of Number Theory
- Video Demonstration