Matematyka Dyskretna

Projekt

Przygotowali:
Maciej Kos
Jakub Janik

KOD Program.cs

```
using System;
using System.ComponentModel;
using Projekt_MD;
public class Program
{
    public static void Main(string[] args)
    {
        int n;
        double p;
        int[,] graphMatrix = null;
        int[] vertexDegrees = null;
        double edgeCount = 0;
        double graphDensity = 0;
        int[] visited;

    bool exit = false;
        using (StreamWriter writer = new StreamWriter("wynik.txt"))
        {
            while (!exit)
            {
                Console.WriteLine("------ Menu ------");
                Console.WriteLine("1. Generuj macierz grafu");
                Console.WriteLine("2. Zamień macierz na listę");
                Console.WriteLine("3. Zamień macierz na zestaw krawędzi");
                Console.WriteLine("4. Oblicz sumy stopni wierzchołków");
                Console.WriteLine("5. Oblicz m");
                Console.WriteLine("6. Oblicz gęstość grafu");
                Console.WriteLine("7. Przeszukanie");
                Console.WriteLine("8. Wyjście");

                Console.Write("\nWybierz opcję: ");
                int option = Convert.ToInt32(Console.ReadLine());
                Console.WriteLine();

                switch (option)
                {
                    case 1:
                        Console.Write("Podaj wartość n: ");
                        n = Convert.ToInt32(Console.ReadLine());
                        Console.Write("Podaj wartość p: ");
                        p = Convert.ToDouble(Console.ReadLine());
                        graphMatrix = Matrix.GenerateGraphMatrix(n, p);
                        Console.WriteLine("Macierz grafu wygenerowana.");

                        int L = graphMatrix.GetLength(0);

                        Console.WriteLine("Macierz grafu:");
                        writer?.WriteLine("Macierz grafu:");
                        for (int i = 0; i < L; i++)
                        {
```

```
            for (int j = 0; j < L; j++)
            {
                Console.Write(graphMatrix[i, j] + " ");
                writer?.Write(graphMatrix[i, j] + " ");
            }
            Console.WriteLine();
            writer?.WriteLine();
        }
        break;

    case 2:
        if (graphMatrix == null)
        {
            Console.WriteLine("Macierz grafu nie została wygenerowana.");
        }
        else
        {
            List<List<int>> graphList = Matrix.ConvertMatrixToList(graphMatrix);
            Console.WriteLine("Lista grafu:");
            writer?.WriteLine("Lista grafu:");
            PrintGraphList(graphList);
        }
        break;

    case 3:
        if (graphMatrix == null)
        {
            Console.WriteLine("Macierz grafu nie została wygenerowana.");
        }
        else
        {
            HashSet<Tuple<int, int>> edges = Matrix.ConvertMatrixToEdges(graphMatrix);
            Console.WriteLine("Zestaw krawędzi grafu:");
            writer?.WriteLine("Zestaw krawędzi grafu:");
            PrintGraphEdges(edges);
        }
        break;

    case 4:
        if (graphMatrix == null)
        {
            Console.WriteLine("Macierz grafu nie została wygenerowana.");
        }
        else
        {
            vertexDegrees = Matrix.CalculateVertexDegrees(graphMatrix);
            Console.WriteLine("Sumy stopni wierzchołków:");
            writer?.WriteLine("Sumy stopni wierzchołków:");
            PrintVertexDegrees(vertexDegrees);
        }
        break;
```

```
case 5:
    if (vertexDegrees == null)
    {
        Console.WriteLine("Sumy stopni wierzchołków nie zostały obliczone.");
    }
    else
    {
        edgeCount = Matrix.CalculateEdgeCount(vertexDegrees);
        Console.WriteLine("Liczba krawędzi (m): " + edgeCount);
        writer?.WriteLine("Liczba krawędzi (m): " + edgeCount);
    }
    break;

case 6:
    if (graphMatrix == null)
    {
        Console.WriteLine("Macierz grafu nie została wygenerowana.");
    }
    else
    {
        if (edgeCount == 0)
        {
            Console.WriteLine("Liczba krawędzi (m) nie została obliczona.");
        }
        else
        {
            graphDensity = Matrix.CalculateGraphDensity(edgeCount, graphMatrix.GetLength(0));
            Console.WriteLine("Gęstość grafu (p): " + graphDensity);
            writer?.WriteLine("Gęstość grafu (p): " + graphDensity);
        }
    }
    break;
case 7:
    if (graphMatrix != null)
    {
        Console.WriteLine("Podaj wierzchołek początkowy (0 - " + (graphMatrix.GetLength(0) - 1) + "):");
        int startVertex = Convert.ToInt32(Console.ReadLine());
        visited = new int[graphMatrix.GetLength(0)];
        DFS(startVertex);
    }
    else
    {
        Console.WriteLine("Najpierw wygeneruj macierz grafu.");
    }
    break;
```

```
        case 8:
            exit = true;
            break;

        default:
            Console.WriteLine("Nieprawidłowa opcja.");
            break;
    }

    Console.WriteLine();
}
void PrintGraphList(List<List<int>> graphList)
{
    for (int i = 0; i < graphList.Count; i++)
    {
        Console.Write(i + ": ");
        foreach (int neighbor in graphList[i])
        {
            Console.Write(neighbor + " ");
            writer?.WriteLine(neighbor + " ");
        }
        Console.WriteLine();
        writer?.WriteLine();
    }
}

void PrintGraphEdges(HashSet<Tuple<int, int>> edges)
{
    foreach (Tuple<int, int> edge in edges)
    {
        Console.WriteLine(edge.Item1 + " - " + edge.Item2);
        writer?.WriteLine(edge.Item1 + " - " + edge.Item2);
    }
}
void PrintVertexDegrees(int[] vertexDegrees)
{
    for (int i = 0; i < vertexDegrees.Length; i++)
    {
        Console.WriteLine("Wierzchołek " + i + ": " + vertexDegrees[i]);
        writer?.WriteLine("Wierzchołek " + i + ": " + vertexDegrees[i]);
    }
}
void DFS(int vertex)
{
    Console.Write(vertex + " ");
    writer?.WriteLine("\n Przeszukanie => Krawędź "+ vertex + " ");
    visited[vertex] = 1;

    for (int i = 0; i < graphMatrix.GetLength(0); i++)
    {
        if (graphMatrix[vertex, i] == 1 && visited[i] == 0)
        {
            DFS(i);
        }
    }
```

Matrix.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Projekt_MD
{
    public class Matrix
    {
        public static int[,] GenerateGraphMatrix(int n, double p)
        {
            int[,] A = new int[n, n];

            Random random = new Random();

            for (int i = 0; i < n; i++)
            {
                for (int j = i + 1; j < n; j++)
                {
                    if (random.NextDouble() <= p)
                    {
                        A[i, j] = 1;
                        A[j, i] = 1;
                    }
                }
            }

            return A;
        }
        public static List<List<int>> ConvertMatrixToList(int[,] A)
        {
            List<List<int>> graphList = new List<List<int>>();

            int n = A.GetLength(0);

            for (int i = 0; i < n; i++)
            {
                List<int> neighbors = new List<int>();
                for (int j = 0; j < n; j++)
                {
                    if (A[i, j] == 1)
                    {
                        neighbors.Add(j);
                    }
                }
                graphList.Add(neighbors);
            }

            return graphList;
        }
```

```
    public static HashSet<Tuple<int, int>> ConvertMatrixToEdges(int[,] A)
    {
        HashSet<Tuple<int, int>> edges = new HashSet<Tuple<int, int>>();

        int n = A.GetLength(0);

        for (int i = 0; i < n; i++)
        {
            for (int j = i + 1; j < n; j++)
            {
                if (A[i, j] == 1)
                {
                    edges.Add(new Tuple<int, int>(i, j));
                }
            }
        }

        return edges;
    }
    public static int[] CalculateVertexDegrees(int[,] A)
    {
        int n = A.GetLength(0);
        int[] degrees = new int[n];

        for (int i = 0; i < n; i++)
        {
            int degree = 0;
            for (int j = 0; j < n; j++)
            {
                degree += A[i, j];
            }
            degrees[i] = degree;
        }

        return degrees;
    }

    public static double CalculateEdgeCount(int[] degrees)
    {
        int sum = degrees.Sum();
        return 0.5 * sum;
    }

    public static double CalculateGraphDensity(double edgeCount, int n)
    {
        return edgeCount / (0.5 * n * (n - 1));
    }
}
```

Przykładowe wyniki działania programu

Odpalając program mamy 8 opcji do wyboru:

```
------ Menu ------
1. Generuj macierz grafu
2. Zamień macierz na listę
3. Zamień macierz na zestaw krawędzi
4. Oblicz sumy stopni wierzchołków
5. Oblicz m
6. Oblicz gęstość grafu
7. Przeszukanie
8. Wyjście
```

W programie prawdopodobieństwo wprowadzamy z przecinkiem

Przykład 1

Krok 1:
Ilość wierzchołków 10
Prawdopodobieństwo 0.3
Wygenerowana macierz

```
Macierz grafu:
0 1 0 0 1 0 0 0 0 0
1 0 0 0 0 1 0 0 0 0
0 0 0 1 1 1 0 0 0 0
0 0 1 0 0 0 0 1 0 0
1 0 1 0 0 0 0 1 0 0
0 1 1 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 1
0 0 0 0 0 1 0 0 1 0
```

Krok 2:

```
Lista grafu:
0: 1 4
1: 0 5
2: 3 4 5
3: 2 7
4: 0 2 7
5: 1 2 8 9
6:
7: 3 4
8: 5 9
9: 5 8
```

Krok 3:

```
Zestaw krawędzi grafu:
0 - 1
0 - 4
1 - 5
2 - 3
2 - 4
2 - 5
3 - 7
4 - 7
5 - 8
5 - 9
8 - 9
```

Krok 4:

```
Sumy stopni wierzchołków:
Wierzchołek 0: 2
Wierzchołek 1: 2
Wierzchołek 2: 3
Wierzchołek 3: 2
Wierzchołek 4: 3
Wierzchołek 5: 4
Wierzchołek 6: 0
Wierzchołek 7: 2
Wierzchołek 8: 2
Wierzchołek 9: 2
```

Krok 5:

```
Liczba krawędzi (m): 11
```

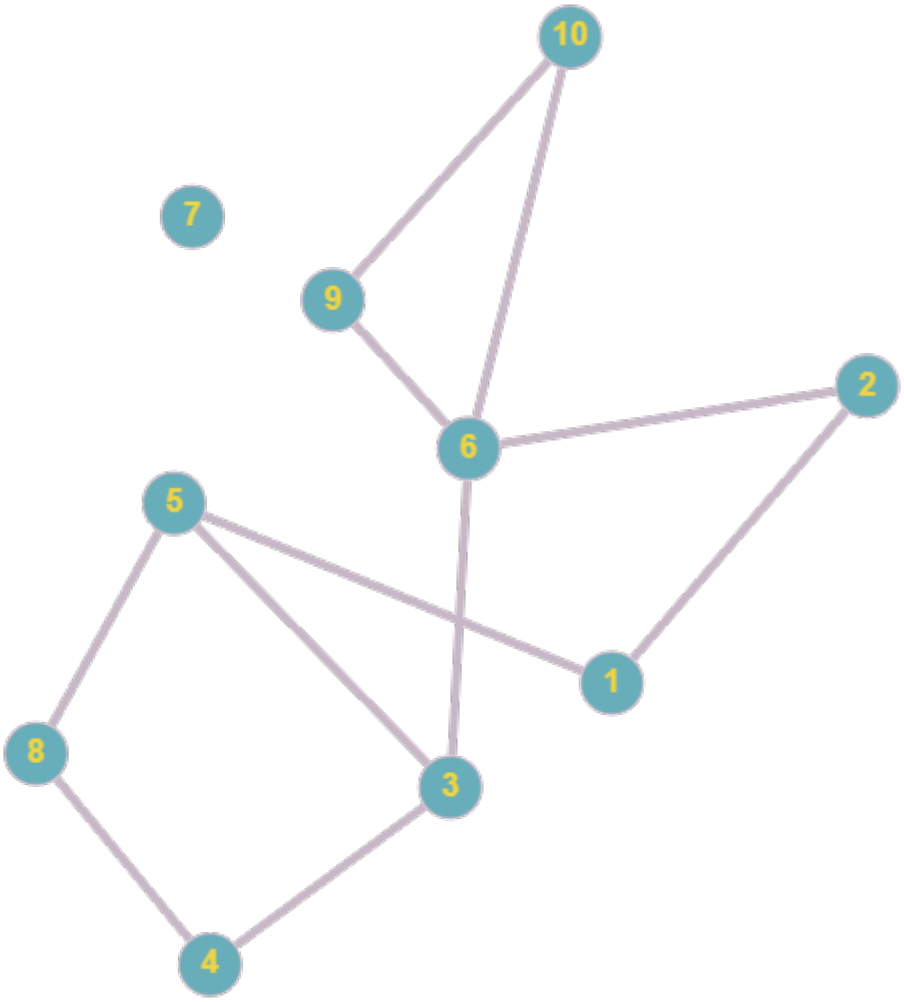
Krok 6:

```
Gęstość grafu (p): 0,24444444444444444
```

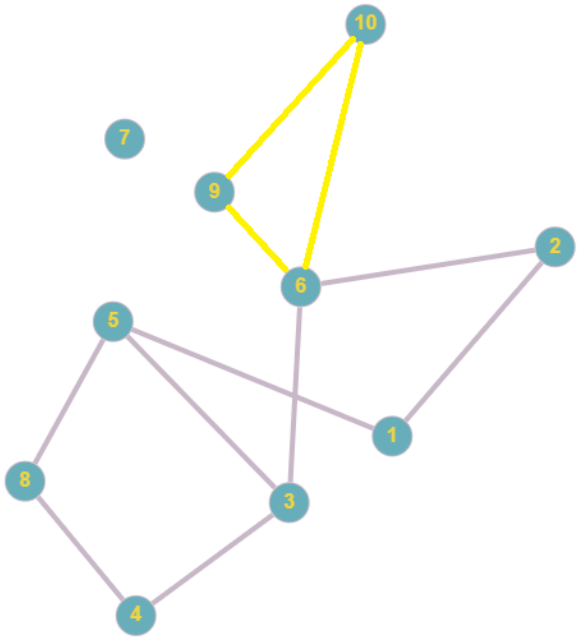
Krok 7:

```
Podaj wierzchołek początkowy (0 - 9):
2
2 3 7 4 0 1 5 8 9
```

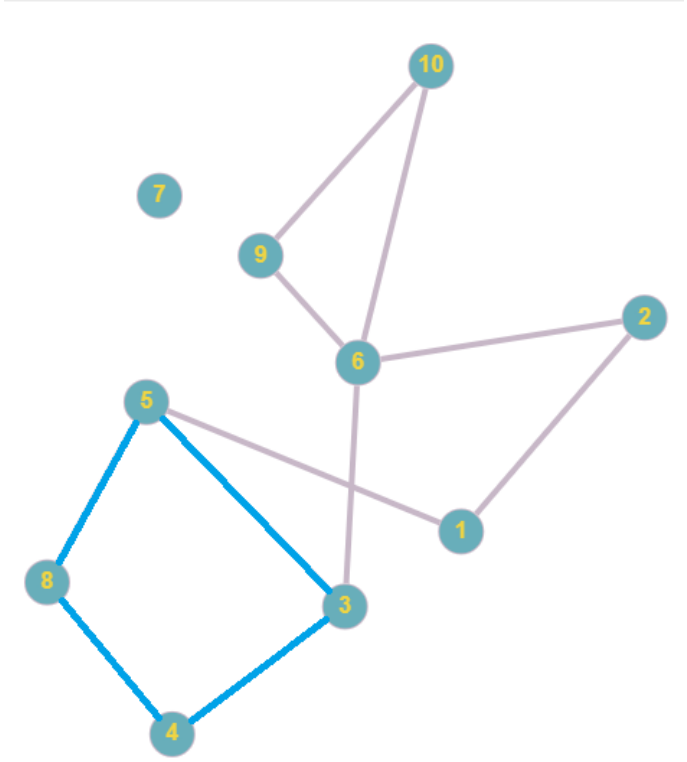
Wygenerowany graf:



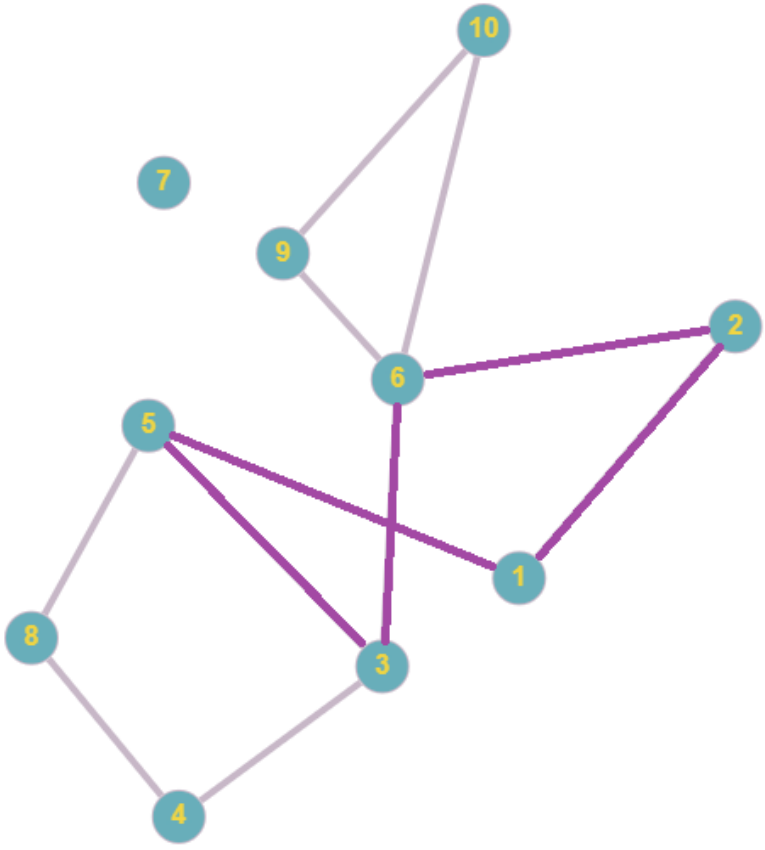
Cykle o długości : 3



Cykle o długości : 4



Cykle o długości : 5



Przykład 2

Krok 1:
Ilość wierzchołków 10
Prawdopodobieństwo 0.3
Wygenerowana macierz

```
Macierz grafu:
0 1 0 1 1 0 0 0 1 0
1 0 1 0 0 0 0 0 1 1
0 1 0 1 1 0 0 0 1 0
1 0 1 0 1 1 0 0 0 1
1 0 1 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0
1 1 1 0 0 0 0 1 0 0
0 1 0 1 0 0 0 0 0 0
```

Krok 2:

```
Lista grafu:
0: 1 3 4 8
1: 0 2 8 9
2: 1 3 4 8
3: 0 2 4 5 9
4: 0 2 3
5: 3
6:
7: 8
8: 0 1 2 7
9: 1 3
```

Krok 3:

```
Zestaw krawędzi grafu:
0 - 1
0 - 3
0 - 4
0 - 8
1 - 2
1 - 8
1 - 9
2 - 3
2 - 4
2 - 8
3 - 4
3 - 5
3 - 9
7 - 8
```

Krok 4:

```
Sumy stopni wierzchołków:
Wierzchołek 0: 4
Wierzchołek 1: 4
Wierzchołek 2: 4
Wierzchołek 3: 5
Wierzchołek 4: 3
Wierzchołek 5: 1
Wierzchołek 6: 0
Wierzchołek 7: 1
Wierzchołek 8: 4
Wierzchołek 9: 2
```

Krok 5:

```
Liczba krawędzi (m): 14
```

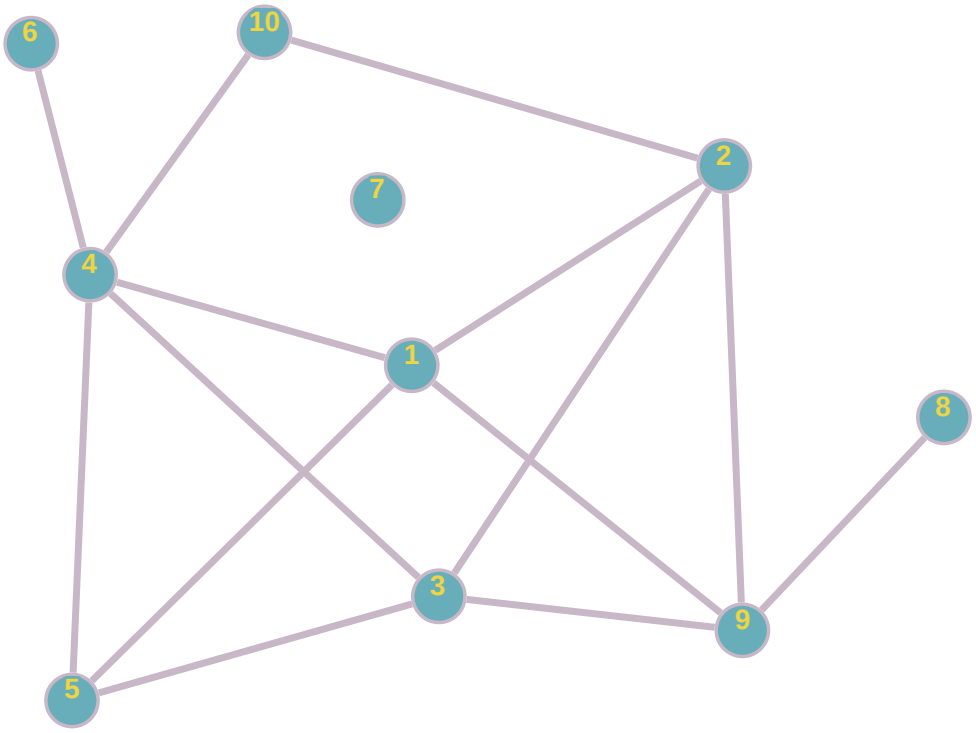
Krok 6:

```
Gęstość grafu (p): 0,3111111111111111
```

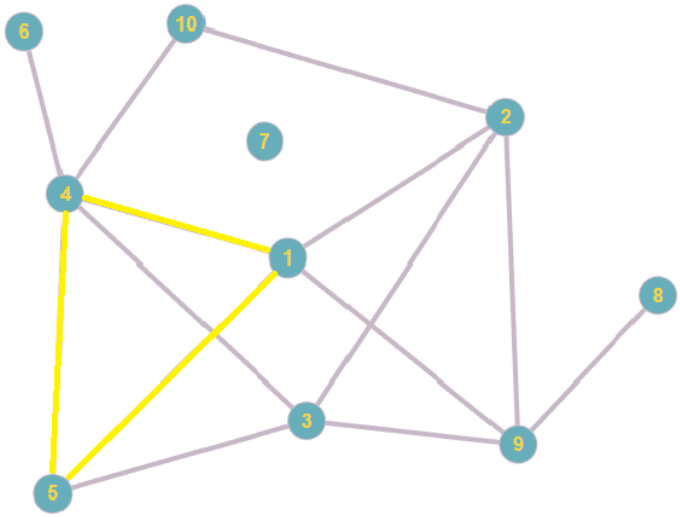
Krok 7:

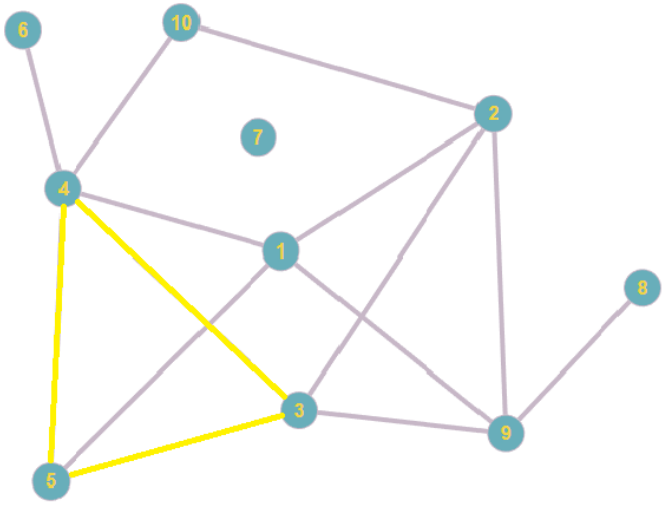
```
Podaj wierzchołek początkowy (0 - 9):
3
3 0 1 2 4 8 7 9 5
```

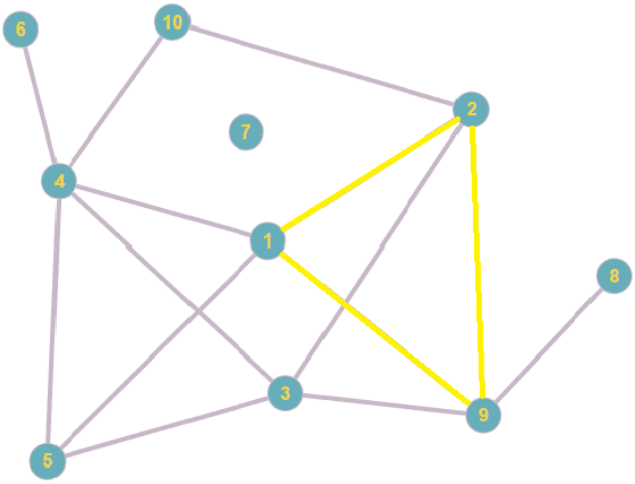
Wygenerowany graf:

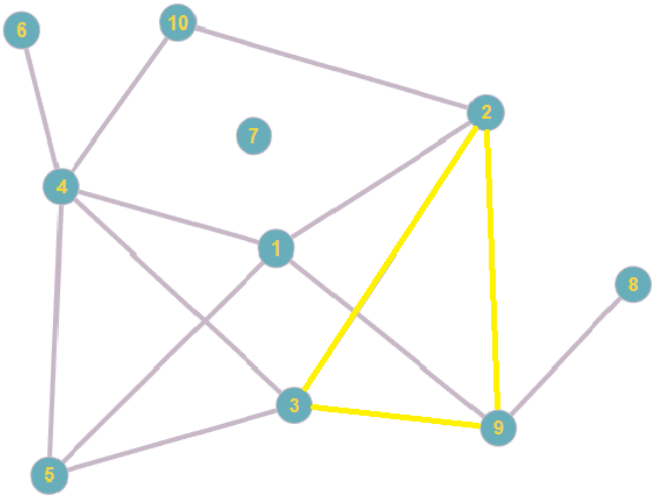


Cykle o długości : 3

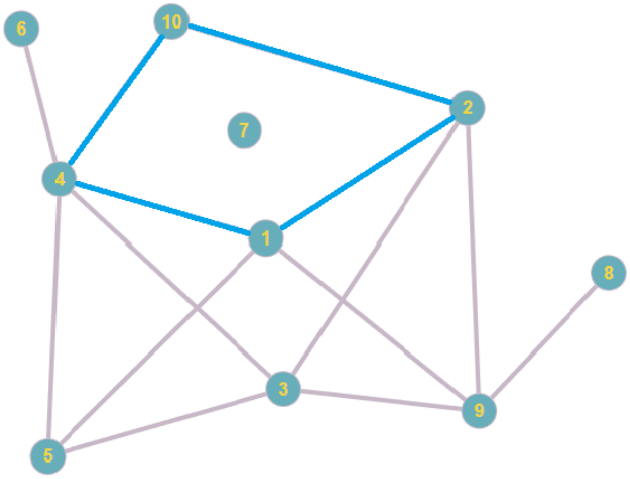


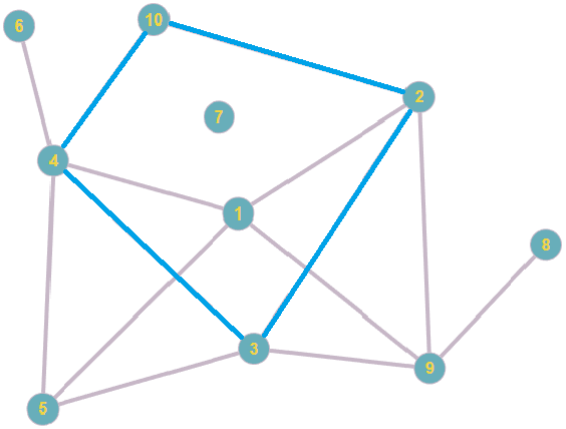


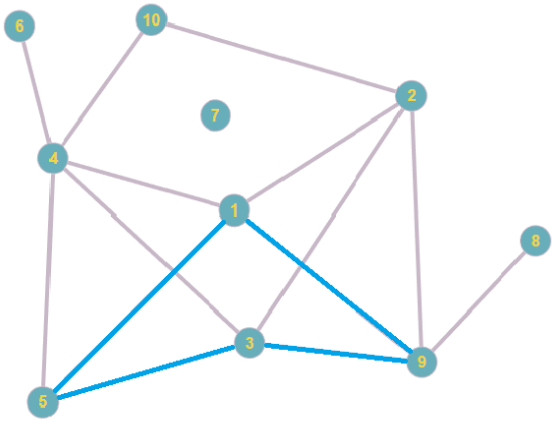


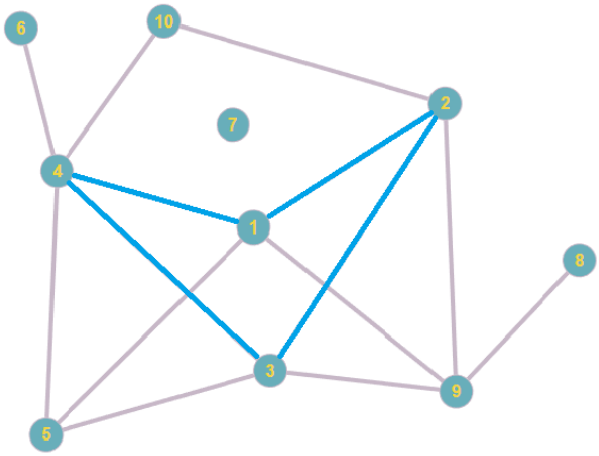


Cykle długości : 4









Cykle długości : 5