**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 4 з дисципліни

«Проектування алгоритмів»

„**Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.1**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-13 Бондаренко М.В.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О.О.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc51260917)

[2 Завдання 4](#_Toc51260918)

[3 Виконання 10](#_Toc51260919)

[3.1 Програмна реалізація алгоритму 10](#_Toc51260920)

[3.1.1 Вихідний код 10](#_Toc51260921)

[3.1.2 Приклади роботи 18](#_Toc51260922)

[3.2 Тестування алгоритму 20](#_Toc51260923)

[3.2.1 Значення цільової функції зі збільшенням кількості ітерацій 20](#_Toc51260924)

[3.2.2 Графіки залежності розв'язку від числа ітерацій 21](#_Toc51260925)

[Висновок 22](#_Toc51260926)

[Критерії оцінювання 22](#_Toc51260927)

# Мета лабораторної роботи

Мета роботи – вивчити основні підходи формалізації метаеврестичних алгоритмів і вирішення типових задач з їхньою допомогою.

# Завдання

Згідно варіанту, розробити алгоритм вирішення задачі і виконати його програмну реалізацію на будь-якій мові програмування.

Задача, алгоритм і його параметри наведені в таблиці 2.1.

Зафіксувати якість отриманого розв'язку (значення цільової функції) після кожних 20 ітерацій до 1000 і побудувати графік залежності якості розв'язку від числа ітерацій.

Зробити узагальнений висновок.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |
| --- | --- |
| **№** | **Задача і алгоритм** |
| 1 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 2 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 3 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 4 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 5 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |
| 6 | Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники). |
| 7 | Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 8 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 3, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах М = 45, починають маршрут в різних випадкових вершинах). |
| 9 | Задача розфарбовування графу (150 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 25 із них 3 розвідники). |
| 10 | Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 11 | Задача комівояжера (250 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах М = 45, починають маршрут в різних випадкових вершинах). |
| 12 | Задача розфарбовування графу (300 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 60 із них 5 розвідники). |
| 13 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий 30% і 70%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 14 | Задача комівояжера (250 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 4, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (10 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах). |
| 15 | Задача розфарбовування графу (100 вершин, степінь вершини не більше 20, але не менше 1), класичний бджолиний алгоритм (число бджіл 30 із них 3 розвідники). |
| 16 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий 30%, 40% і 30%, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 17 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 2, β = 4, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (15 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах). |
| 18 | Задача розфарбовування графу (300 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 60 із них 5 розвідники). |
| 19 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 20 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 3, β = 2, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (10 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах). |
| 21 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 30, але не менше 1), класичний бджолиний алгоритм (число бджіл 40 із них 2 розвідники). |
| 22 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 23 | Задача комівояжера (300 вершин, відстань між вершинами випадкова від 1 до 60), мурашиний алгоритм (α = 3, β = 2, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (15 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах). |
| 24 | Задача розфарбовування графу (400 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 70 із них 10 розвідники). |
| 25 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 26 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 27 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 28 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 29 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |
| 30 | Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники). |
| 31 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 32 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 33 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 34 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 35 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |

# Виконання

## Програмна реалізація алгоритму

### Вихідний код

Constant.cs

public class Constant  
{  
 */\*\*  
 \* Graph constants  
 \*/* public static readonly int NumberOfVertices = 200;  
 public static readonly int NoColor = -1;  
 public static readonly int VertexCount = 200;  
 public static readonly int MinVertexDegree = 1;  
 public static readonly int MaxVertexDegree = 20;  
  
 */\*\*  
 \* ABC algorithm constants  
 \*/* public static readonly int ExplorerBeesCount = 2;  
 public static readonly int TotalBeesCount = 30;  
 public static readonly int IterationsPerStep = 20;  
 public static readonly int MaxIterationsCount = 1000;  
}

Program.cs

using System.Diagnostics;  
  
namespace lab4;  
  
class Program  
{  
 public static void Main(string[] args)  
 {  
 var adjMatrix = new int[Constant.NumberOfVertices, Constant.NumberOfVertices];  
 var graph = new Graph(adjMatrix);  
  
 Console.WriteLine("Matrix is valid: " + graph.IsMatrixValid());  
  
 Util.PrintMatrix(graph.AdjacencyMatrix);  
 Console.WriteLine("Graph degrees: ");  
 Util.PrintArray(graph.GetVertexDegrees());  
  
 Console.WriteLine("Training is started successfully, stand by...");  
 var sw = Stopwatch.StartNew();  
  
 graph = new Algorithm(graph).Run();  
 sw.Stop();  
 Console.WriteLine($"Time to train: {sw.ElapsedMilliseconds / 1000}s");  
   
 Console.WriteLine("Graph coloring: ");  
 Util.PrintArray(graph.GetColors());  
  
 Console.WriteLine("Graph colored properly: " + graph.IsGraphProperlyColored());  
 }  
}

Graph.cs

namespace lab4  
{  
 internal class Graph  
 {  
 private readonly int[,] \_adjacencyMatrix;  
 private readonly int[] \_colors;  
  
 public Graph(Graph g)  
 {  
 \_adjacencyMatrix = new int[g.\_adjacencyMatrix.GetLength(0), g.\_adjacencyMatrix.GetLength(1)];  
 \_colors = new int[g.\_colors.Length];  
  
 Array.Copy(g.\_adjacencyMatrix, \_adjacencyMatrix, g.\_adjacencyMatrix.Length);  
 Array.Copy(g.\_colors, \_colors, g.\_colors.Length);  
 }  
  
 public Graph(int[,] adjacencyMatrix)  
 {  
 Random rand = new Random();  
 \_adjacencyMatrix = adjacencyMatrix;  
 \_colors = new int[adjacencyMatrix.GetLength(0)];  
 Array.Fill(\_colors, Constant.NoColor);  
   
 for (int currentVertex = 0; currentVertex < Constant.VertexCount; ++currentVertex)  
 {  
 int finalVertexDegree = Math.Min(rand.Next(Constant.MinVertexDegree, Constant.MaxVertexDegree + 1)  
 - GetDegreeOfVertex(currentVertex), Constant.VertexCount - currentVertex - 1);  
 for (int newConnections = 0; newConnections < finalVertexDegree; ++newConnections)  
 {  
 bool isConnectedAlready = true;  
 for (int tryCount = 0, newVertex = rand.Next(currentVertex + 1, Constant.VertexCount);  
 isConnectedAlready && tryCount < Constant.VertexCount;  
 ++tryCount, newVertex = rand.Next(currentVertex + 1, Constant.VertexCount))  
 {  
 if (\_adjacencyMatrix[currentVertex, newVertex] == 0 &&  
 GetDegreeOfVertex(newVertex) < Constant.MaxVertexDegree)  
 {  
 isConnectedAlready = false;  
 \_adjacencyMatrix[currentVertex, newVertex] = 1;  
 \_adjacencyMatrix[newVertex, currentVertex] = 1;  
 }  
 }  
 }  
 }  
 }  
  
 public bool IsMatrixValid()  
 {  
 for (int vertex = 0; vertex < \_adjacencyMatrix.GetLength(0); vertex++)  
 {  
 if (GetDegreeOfVertex(vertex) > Constant.MaxVertexDegree)   
 return false;  
 }  
  
 return true;  
 }  
  
  
 public bool IsGraphProperlyColored()  
 {  
 for (int i = 0; i < \_colors.Length; i++)  
 {  
 if (\_colors[i] == Constant.NoColor)   
 return false;  
 }  
  
 return IsColoringValid();  
 }  
  
 public int[] GetColors()  
 {  
 return \_colors;  
 }  
  
 public int[] GetVertexDegrees()  
 {  
 int[] vertexDegrees = new int[\_adjacencyMatrix.GetLength(0)];  
 for (int i = 0; i < vertexDegrees.Length; ++i)  
 {  
 vertexDegrees[i] = GetDegreeOfVertex(i);  
 }  
  
 return vertexDegrees;  
 }  
  
 public int GetDegreeOfVertex(int vertex)  
 {  
 int degree = 0;  
 for (int i = 0; i < \_adjacencyMatrix.GetLength(0); i++)  
 {  
 degree += \_adjacencyMatrix[vertex, i];  
 }  
  
 return degree;  
 }  
  
 public int[] GetAdjacentVertices(int vertex)  
 {  
 int[] adjacentVertices = new int[GetDegreeOfVertex(vertex)];  
 int index = 0;  
 for (int i = 0; i < \_adjacencyMatrix.GetLength(0); ++i)  
 {  
 if (\_adjacencyMatrix[vertex, i] == 1)   
 adjacentVertices[index++] = i;  
 }  
  
 return adjacentVertices;  
 }  
  
  
 public bool IsColorChangeValid(int vertex, int newColor)  
 {  
 int previousColor = \_colors[vertex];  
 \_colors[vertex] = newColor;  
 bool isValid = IsColoringValid();  
 if (!isValid)   
 \_colors[vertex] = previousColor;  
  
 return isValid;  
 }  
  
  
 private bool IsColoringValid()  
 {  
 for (int i = 0; i < \_adjacencyMatrix.GetLength(0); i++)  
 {  
 for (int j = i + 1; j < \_adjacencyMatrix.GetLength(1); j++)  
 {  
 if (\_adjacencyMatrix[i, j] == 1 && \_colors[i] != Constant.NoColor && \_colors[i] == \_colors[j])   
 return false;  
 }  
 }  
  
 return true;  
 }  
  
 public int[,] AdjacencyMatrix => \_adjacencyMatrix;  
 }  
}

Algorithm.cs

using System.Diagnostics;  
  
namespace lab4  
{  
 internal class Algorithm  
 {  
 private Graph \_graph;  
 private List<int> \_availableVertices;  
 private readonly Graph \_initialGraph;  
 private readonly int[] \_paletteArr;  
 private readonly List<int> \_usedColorsList;  
  
 public Algorithm(Graph initialGraph)  
 {  
 \_initialGraph = initialGraph;  
 \_graph = new Graph(initialGraph);  
 \_availableVertices = Util.GetVertices();  
 \_paletteArr = new int[Constant.MaxVertexDegree + 1];  
 for (int i = 0; i < Constant.MaxVertexDegree + 1; i++)  
 {  
 \_paletteArr[i] = i;  
 }  
  
 \_usedColorsList = new List<int>();  
 }  
  
 public Graph Run()  
 {  
 Graph resultGraph = new Graph(\_graph);  
 int bestChromaticNumber = GetBestChromaticNumber();  
  
 Console.WriteLine("Initialize colored graph: ");  
 Util.PrintArray(\_graph.GetColors());  
 Console.WriteLine(  
 "The optimal solution for the graph was found on the first iteration, with the previous maximum vertex " +  
 $"degree being {Constant.MaxVertexDegree + 1} and the new best chromatic number being {bestChromaticNumber}. " +  
 "The estimated time for this process was 0 seconds.");  
 Reset();  
   
 for (int iterations = 0; iterations < Constant.MaxIterationsCount;)  
 {  
 Stopwatch sw = Stopwatch.StartNew();  
 for (int k = 1; k < Constant.IterationsPerStep + 1; k++, Reset())  
 {  
 int newChromaticNumber = GetBestChromaticNumber();  
 if (newChromaticNumber >= bestChromaticNumber) continue;  
 Console.WriteLine($"After {iterations + k} iterations, a new optimal solution for the graph was found. " +  
 $"The previous best chromatic number was {bestChromaticNumber}, and the new best chromatic number " +  
 $"is {bestChromaticNumber = newChromaticNumber}. The estimated time for this process was " +  
 $"{sw.ElapsedMilliseconds / 1000} seconds.");  
 resultGraph = new Graph(\_graph);  
 }  
  
 Console.WriteLine($"On iteration {iterations += Constant.IterationsPerStep}, the best result found was {bestChromaticNumber}. " +  
 $"The estimated time for this process was {sw.ElapsedMilliseconds / 1000} seconds.");  
 }  
  
 Console.WriteLine("Initial colors of graph are (-1 - no color): ");  
 Util.PrintArray(\_graph.GetColors());  
 return resultGraph;  
 }  
  
 private void Reset()  
 {  
 \_usedColorsList.Clear();  
 \_availableVertices = Util.GetVertices();  
 \_graph = new Graph(\_initialGraph);  
 }  
  
 private int GetBestChromaticNumber()  
 {  
 while (!\_graph.IsGraphProperlyColored())  
 {  
 ColorSelectedVertices(SelectExplorerVertices());  
 }  
  
 return \_usedColorsList.Count;  
 }  
  
 private List<int> SelectExplorerVertices()  
 {  
 var selectedVertices = new List<int>();  
 var random = new Random();  
 int numberOfExplorers = Constant.ExplorerBeesCount;  
 while (numberOfExplorers > 0 && \_availableVertices.Count > 0)  
 {  
 int index = random.Next(\_availableVertices.Count);  
 int randomSelectedVertex = \_availableVertices[index];  
 \_availableVertices.RemoveAt(index);  
 selectedVertices.Add(randomSelectedVertex);  
 numberOfExplorers--;  
 }  
  
 return selectedVertices;  
 }  
  
  
 private void ColorSelectedVertices(IReadOnlyList<int> selectedVertices)  
 {  
 var degrees = new int[selectedVertices.Count];  
 for (int i = 0; i < degrees.Length; i++)  
 {  
 degrees[i] = \_graph.GetDegreeOfVertex(selectedVertices[i]);  
 }  
  
 var onlookerBeesCounts = Util.GetOnlookersBeesSplit(degrees);  
 for (int i = 0; i < selectedVertices.Count; i++)  
 {  
 var connectedVertices = \_graph.GetAdjacentVertices(selectedVertices[i]);  
 ColorConnectedVertex(connectedVertices, onlookerBeesCounts[i]);  
 ColorVertex(selectedVertices[i]);  
 }  
 }  
  
 private void ColorConnectedVertex(IReadOnlyList<int> connectedVertices, int onlookerBeesCount)  
 {  
 for (int i = 0; i < connectedVertices.Count; ++i)  
 {  
 if (i < onlookerBeesCount - 1)   
 ColorVertex(connectedVertices[i]);  
 }  
 }  
  
 private void ColorVertex(int vertex)  
 {  
 var availableColors = new HashSet<int>(\_usedColorsList);  
 var random = new Random();  
 int color;  
 while (true)  
 {  
 if (availableColors.Count == 0)  
 {  
 color = \_paletteArr[\_usedColorsList.Count];  
 \_usedColorsList.Add(color);  
 break;  
 }  
  
 color = availableColors.ElementAt(random.Next(availableColors.Count));  
 availableColors.Remove(color);  
 if (\_graph.IsColorChangeValid(vertex, color))  
 break;  
 }  
  
 \_graph.IsColorChangeValid(vertex, color);  
 }  
 }  
}

Util.cs

namespace lab4;  
  
public class Util  
{  
 public static void PrintArray(int[] arr)  
 {  
 const int **maxRowLength** = 10;  
 for (int i = 0; i < arr.Length; i++)  
 {  
 Console.Write(arr[i] + "\t");  
 if ((i + 1) % **maxRowLength** == 0)   
 Console.WriteLine();  
 }  
  
 Console.WriteLine();  
 Console.WriteLine();  
 }  
  
 public static List<int> GetVertices()  
 {  
 return Enumerable.Range(0, Constant.VertexCount).ToList();  
 }  
  
 public static void PrintMatrix(int[,] matrix)  
 {  
 for (int i = 0; i < matrix.GetLength(0); i++)  
 {  
 for (int j = 0; j < matrix.GetLength(1); j++)  
 {  
 Console.Write(matrix[i, j] + " ");  
 }  
  
 Console.WriteLine();  
 }  
  
 Console.WriteLine();  
 }  
  
 private static double[] GetNectarValues(IReadOnlyList<int> selectedVerticesDegrees)  
 {  
 double[] res = new double[selectedVerticesDegrees.Count];  
 for (int i = 0, totalDegrees = selectedVerticesDegrees.Sum(); i < selectedVerticesDegrees.Count; ++i)  
 {  
 res[i] = (double)selectedVerticesDegrees[i] / totalDegrees;  
 }  
 return res;  
 }  
  
 public static int[] GetOnlookersBeesSplit(int[] selectedVerticesDegrees)  
 {  
 double[] nectarValues = GetNectarValues(selectedVerticesDegrees);  
 int onlookerBeesCount = Constant.TotalBeesCount - Constant.ExplorerBeesCount;  
 int[] res = new int[nectarValues.Length];  
 for (int i = 0; i < nectarValues.Length; ++i)  
 {  
 res[i] = (int)(onlookerBeesCount \* nectarValues[i]);  
 onlookerBeesCount -= res[i];  
 }  
 return res;  
 }  
}

### Приклади роботи

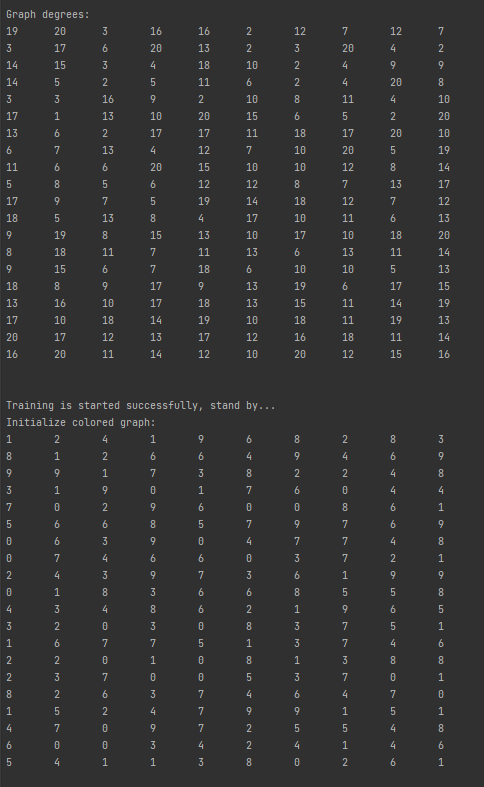
На рисунках 3.1 і 3.2 показані приклади роботи програми.

Рисунок 3.1 – ініціалізація вхідних даних

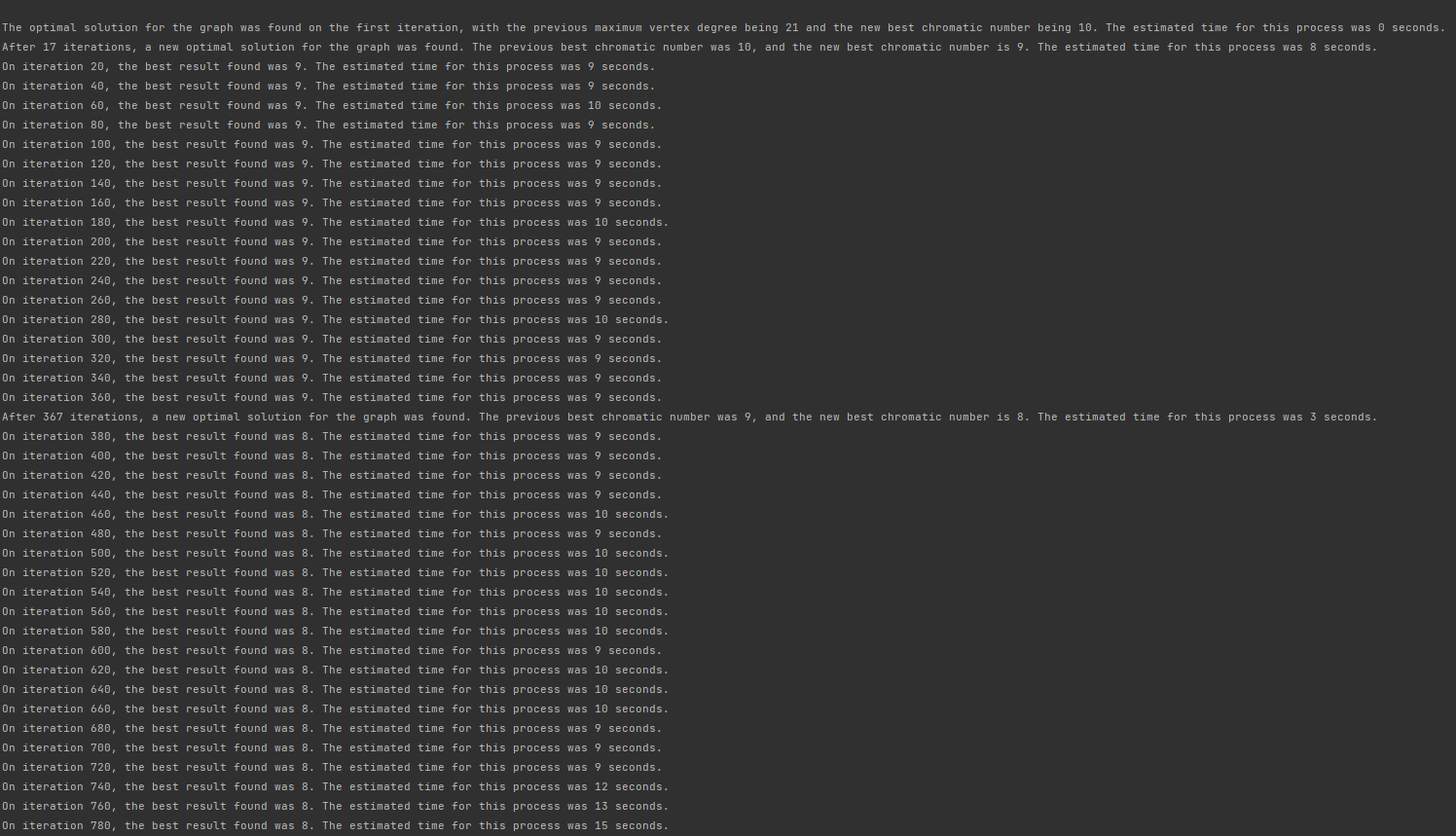


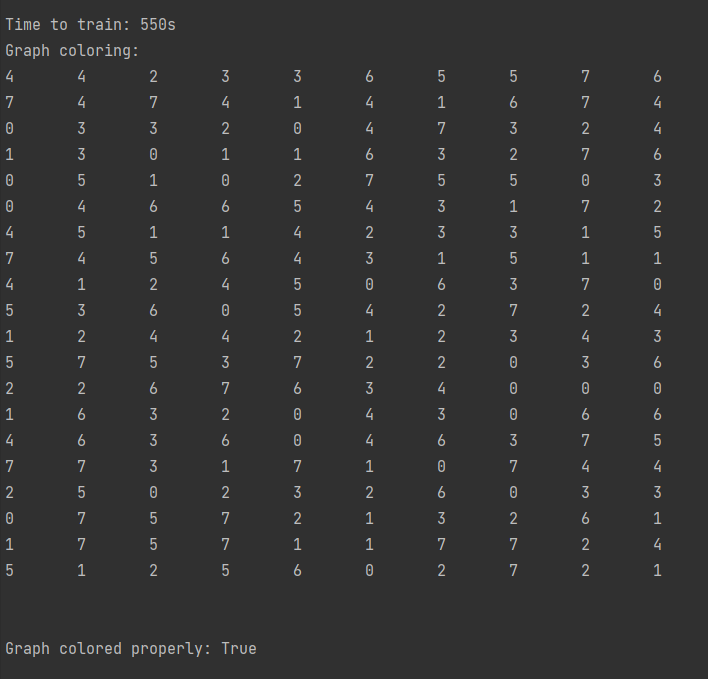
Рисунок 3.2 – тренування програми

Рисунок 3.3 – результати виконання програми

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

|  |  |  |  |
| --- | --- | --- | --- |
| Ітерації | Хроматичне число | Ітерації | Хроматичне число |
| 0 | 21 | 500 | 8 |
| 20 | 9 | 520 | 8 |
| 40 | 9 | 540 | 8 |
| 60 | 9 | 560 | 8 |
| 80 | 9 | 580 | 8 |
| 100 | 9 | 600 | 8 |
| 120 | 9 | 620 | 8 |
| 140 | 9 | 640 | 8 |
| 160 | 9 | 660 | 8 |
| 180 | 9 | 680 | 8 |
| 200 | 9 | 700 | 8 |
| 220 | 9 | 720 | 8 |
| 240 | 9 | 740 | 8 |
| 260 | 9 | 760 | 8 |
| 280 | 9 | 780 | 8 |
| 300 | 9 | 800 | 8 |
| 320 | 9 | 820 | 8 |
| 340 | 9 | 840 | 8 |
| 360 | 9 | 860 | 8 |
| 380 | 8 | 880 | 8 |
| 400 | 8 | 900 | 8 |
| 420 | 8 | 920 | 8 |
| 440 | 8 | 940 | 8 |
| 460 | 8 | 960 | 8 |
| 480 | 8 | 980 | 8 |
|  |  | 1000 | 8 |

У таблиці 3.1 наведено значення цільової функції зі збільшенням кількості ітерацій.

### Графіки залежності розв'язку від числа ітерацій

На рисунку 3.3 наведений графік, який показує якість отриманого розв'язку.



Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

Висновок

При виконанні лабораторної роботи, я зайнявся розв'язком задачі розфарбування графу, який складався з 200 вершин, із степенем від 1 до 20. Для реалізації я використав модифікацію бджолиного алгоритму штучної бджолиної колонії, який був написаний на мові програмування С#. Я використав 30 бджіл, з яких 2 були розвідниками, побудував графік залежності хроматичного числа від кількості ітерацій і виявив, що після 380 ітерацій хроматичне число не змінюється.

Критерії оцінювання

При здачі лабораторної роботи до 27.11.2021 включно максимальний бал дорівнює – 5. Після 27.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* програмна реалізація алгоритму – 75%;
* тестування алгоритму– 20%;
* висновок – 5%.