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 № 5 з дисципліни
«Проектування алгоритмів»

“Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.2”

Виконав ІП-14 Ковалик Назар Миколайович
(шифр, прізвище, ім’я, по батькові)

Перевірив Ілля Елдарійович Ахаладзе
(прізвище, ім’я, по батькові)

Київ 2022

ЗМІСТ

1. МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи розробки метаевристичних алгоритмів для типових прикладних задач. Опрацювати методологію підбору прийнятних параметрів алгоритму.

2. ЗАВДАННЯ

Згідно варіанту, формалізувати алгоритм вирішення задачі відповідно загальної методології.

Записати розроблений алгоритм у покроковому вигляді. З достатнім степенем деталізації.

Виконати його програмну реалізацію на будь-якій мові програмування.

Перелік задач наведено у таблиці 2.1.

Перелік алгоритмів і досліджуваних параметрів у таблиці 2.2.

Задача і алгоритм наведені в таблиці 2.3.

Змінюючи параметри алгоритму, визначити кращі вхідні параметри алгоритму. Для цього необхідно:

- обрати критерій зупинки алгоритму (кількість ітерацій або значення ЦФ);
- зафіксувати усі параметри крім одного і змінювати цей параметр, поки не буде досягнуто пікової ефективності;
- після цього параметр фіксується і змінюються інші параметри;
- далі повторюємо процедуру спочатку, з першого зафіксованого параметру;
- зупиняємось коли будуть знайдені оптимальні параметри для даної задачі або встановлена залежність одних параметрів від інших.

Зробити узагальнений висновок в якому обов’язково описати залежність якості розв’язку від вхідних параметрів.

Таблиця 2.1 – Прикладні задачі

№	**Задача**
2	**Задача комівояжера** (300 вершин, відстань між вершинами випадкова від 5 до 150) полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки один раз, в такому випадку розв'язок знаходиться серед гамільтонових циклів. **Розглядається симетричний, асиметричний та змішаний варіанти.** В загальному випадку, асиметрична задача комівояжера відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напряму, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також

	зважати і на те, в якому напрямку знаходяться ребра. У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу. У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напряму руху. Застосування: - доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів); - доставка води; - моніторинг об'єктів; - поповнення банкоматів готівкою; - збір співробітників для доставки вахтовим методом.

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№	**Алгоритми і досліджувані параметри**
2	**Мурашиний алгоритм**: - α; - β; - ρ; - Lmin; - кількість мурах M і їх типи (елітні, тощо…); - маршрути з однієї чи різних вершин.

Таблиця 2.3 – Варіанти задач і алгоритмів

№	**Задачі і алгоритми**
8	Задача комівояжера (змішана мережа) + Мурашиний алгоритм

3. ВИКОНАННЯ

1. Програмна реалізація алгоритму

1. Вихідний код

MainWindow.xaml.cs

```
using pa5.Algorithm;
using System;
using System.Linq;
using System.Windows;
using System.Windows.Media;
using System.Windows.Shapes;
using System.Windows.Threading;

namespace pa5.Presentation;

public partial class MainWindow : Window
{
    private AntsGraphSearch search = new();
    private readonly DispatcherTimer _timer = new();
    public MainWindow()
    {
        InitializeComponent();
        _timer.Interval = TimeSpan.FromMilliseconds(10);
        _timer.Tick += _timer_Tick; ;
    }

    private void _timer_Tick(object? sender, EventArgs e)
    {
        UpdateMap();
    }

    void UpdateMap()
    {
        search.UpdateRoads();
        search.Results.Iterations++;
        if (search.Results.Iterations < 50)
        {
            ReDrawCadr();
        }
        else if (search.Results.GenerationsLeft > 0)
        {
            search.UpdateScore();
            search.Results.Iterations = 0;
            search.Results.GenerationsLeft--;
            ReDrawCadr();
            search.Init();
        }
        else
        {
            search.UpdateScore();
        }
    }

    public void ReDrawCadr()
    {
        canvas.Children.Clear();
        var lastColor = false;
        string strokeStyle = "#000000";
        for (int i = 0; i < search.PointsCount; i++)
        {
            for (int j = i + 1; j < search.PointsCount; j++)
            {
                var thisColor = search.Points[i].Path == j
                    || search.Points[j].Path == i;
                if (search.Pheramones[i, j] < 0.01 && !thisColor)
                {
                    continue;
                }
```

```
            if (lastColor != thisColor)
            {
                strokeStyle = thisColor ? "#0000ff" : "#000000";
                lastColor = thisColor;
            }

            var currentColor = GetColor(strokeStyle);
            var line = GetLine(i, j, thisColor, currentColor);
            canvas.Children.Add(line);
        }
    }

    UpdateText();
}

private static SolidColorBrush GetColor(string strokeStyle) =>
    new ((Color)ColorConverter.ConvertFromString(strokeStyle));

private Line GetLine(int i, int j, bool thisColor, SolidColorBrush currentColor)
{
    return new()
    {
        X1 = search.Points[i].X,
        Y1 = search.Points[i].Y,
        X2 = search.Points[j].X,
        Y2 = search.Points[j].Y,
        Opacity = thisColor ? 1 : Math.Atan(search.Pheramones[i, j]) / Math.PI * 2,
        Stroke = currentColor,
        StrokeThickness = 1,
    };
}

private void UpdateText()
{
    n.Text = $"Iterations: {search.Results.Iterations} Colony: {search.Results.TestsCount - search.Results.GenerationsLeft}" +
        $"/{search.Results.TestsCount} Help: {search.Results.CurrentScore}";
}

private void Button_Click(object sender, RoutedEventArgs e)
{
    _timer.Start();
}

private T? Parse<T>(Func<string, T> parser, string input, string parameterName)
{
    try
    {
        var result = parser(input);
        return result;
    }
    catch (FormatException)
    {
        n.Text += parameterName + " contains some chars or other incorrect symbols";
        throw;
    }
    catch (OverflowException)
    {
        n.Text += parameterName + " is too large";
        throw;
    }
}

private (double, double, double, int, int) ParseInitialValues(out bool success)
{
    try
    {
        var a = Parse(double.Parse, aT.Text.Trim().Replace('.', ','), "alpha");
        var b = Parse(double.Parse, bT.Text.Trim().Replace('.', ','), "beta");
        var p = Parse(double.Parse, pT.Text.Trim().Replace('.', ','), "ro");
        var pointsCount = Parse(int.Parse, pointsT.Text, "points count");
```

```
            var antsCount = Parse(int.Parse, antsT.Text, "ants count");
            success = true;
            return (a, b, p, pointsCount, antsCount);
        }
        catch
        {
            success = false;
        }

        return default;
    }

    private readonly ConfigurationValidation _validation = new();
    private void generateB_Click(object sender, RoutedEventArgs e)
    {
        var (a, b, p, points, ants) = ParseInitialValues(out var success);
        if (!success)
        {
            return;
        }

        var conf = new Configuration(ants, points, p, a, b);
        var validationResult = _validation.Validate(conf);
        if (!validationResult.IsValid)
        {
            n.Text = string.Join("\n", validationResult.Errors.Select(x => x.ErrorMessage));
            return;
        }

        search = new(conf);
        ReDrawCadr();
        generateB.IsEnabled = false;
        startB.IsEnabled = true;
        stopB.IsEnabled = true;
        n.Text = "";
    }

    private void stopB_Click(object sender, RoutedEventArgs e)
    {
        _timer.Stop();
    }
}
```

MainWindow.xaml

```
<Window x:Class="pa5.Presentation.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:pa5.Presentation"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="181*"/>
            <ColumnDefinition Width="619*"/>
        </Grid.ColumnDefinitions>
        <StackPanel Grid.Column="0">
            <TextBlock>a</TextBlock>
            <TextBox Text ="3" Name="aT"></TextBox>

            <TextBlock>b</TextBlock>
            <TextBox Text="2" Name="bT"></TextBox>

            <TextBlock>p</TextBlock>
            <TextBox Text="0.3" Name="pT"></TextBox>

            <TextBlock>ants count</TextBlock>
            <TextBox Name="antsT" Text="2"></TextBox>
```

```
        <TextBlock>points count</TextBlock>
        <TextBox Name="pointsT" Text="60"></TextBox>
        <Button
          Name="generateB"
          Click="generateB_Click"
         >
          Generate
        </Button>

        <Button Click="Button_Click"
            Name="startB"
            IsEnabled="False">
          Start
        </Button>
        <Button Click="stopB_Click"
            Name="stopB"
            IsEnabled="False">Stop</Button>
        <TextBlock Name="n" Height="155"
             TextWrapping="WrapWithOverflow">
        </TextBlock>
      </StackPanel>
      <Canvas Name="canvas"
          Grid.Column="1"
          Width="700"
          Height="700"/>
  </Grid>
</Window>
```

Point.cs

```
namespace pa5.Algorithm;

public class Point
{
  public Point(int x, int y, int path)
  {
    X = x;
    Y = y;
    Path = path;
  }

  public int X { get; set; }
  public int Y { get; set; }
  public int Path { get; set; }
}
```

ConfigurationValidation.cs

```
using FluentValidation;

namespace pa5.Algorithm;

public class ConfigurationValidation : AbstractValidator<Configuration>
{
  public ConfigurationValidation()
  {
    RuleFor(x => x.AntsCount)
      .GreaterThan(0)
      .LessThanOrEqualTo(50)
      .WithMessage($"Ants count must be in the range: from 0 to 50");

    RuleFor(x => x.PointsCount)
      .GreaterThan(0)
```

```
            .LessThanOrEqualTo(200)
            .WithMessage($"Points count must be in the range: from 0 to 200");

        RuleFor(x => x.Ro)
            .GreaterThanOrEqualTo(0)
            .LessThan(5)
            .WithMessage($"Ro must be in the range: from 0 to 5");

        RuleFor(x => x.Beta)
            .GreaterThanOrEqualTo(0)
            .LessThan(5)
            .WithMessage($"Beta must be in the range: from 0 to 5");

        RuleFor(x => x.Alpha)
            .GreaterThanOrEqualTo(0)
            .LessThan(5)
            .WithMessage($"Alpha must be in the range: from 0 to 5");
    }
}
```

AntsGraphSearch.cs

```
namespace pa5.Algorithm;
public class AntsGraphSearch
{

            private List<Point> points;
            private double[,] distances;
            private double[,] feramones;
    private Configuration _conf;
    public AlgorithmResults Results { get; private set; }

    public double[,] Distances => distances;
    public double[,] Pheramones => feramones;
    public List<Point> Points => points;
    public int PointsCount => _conf.PointsCount;
    public AntsGraphSearch()
    {
        Results = new();
        _conf = new();
        Init();
    }

    public AntsGraphSearch(Configuration configuration)
    {
        Results = new();
        _conf = configuration;
        Init();
    }

    public void Init()
    {
        points = new();
        Results.ShortestDistance = double.MaxValue;
        Results.Iterations = 0;
        points = new(_conf.PointsCount);
        distances = new double[_conf.PointsCount, _conf.PointsCount];
        feramones = new double[_conf.PointsCount, _conf.PointsCount];
        GeneratePoints();
        CalculateInitialDistancesAndPheramones();
    }

    public void UpdateScore()
    {
        Results.CurrentScore += Results.InitialShortestDistance - Results.ShortestDistance;
    }

    private void GeneratePoints()
```

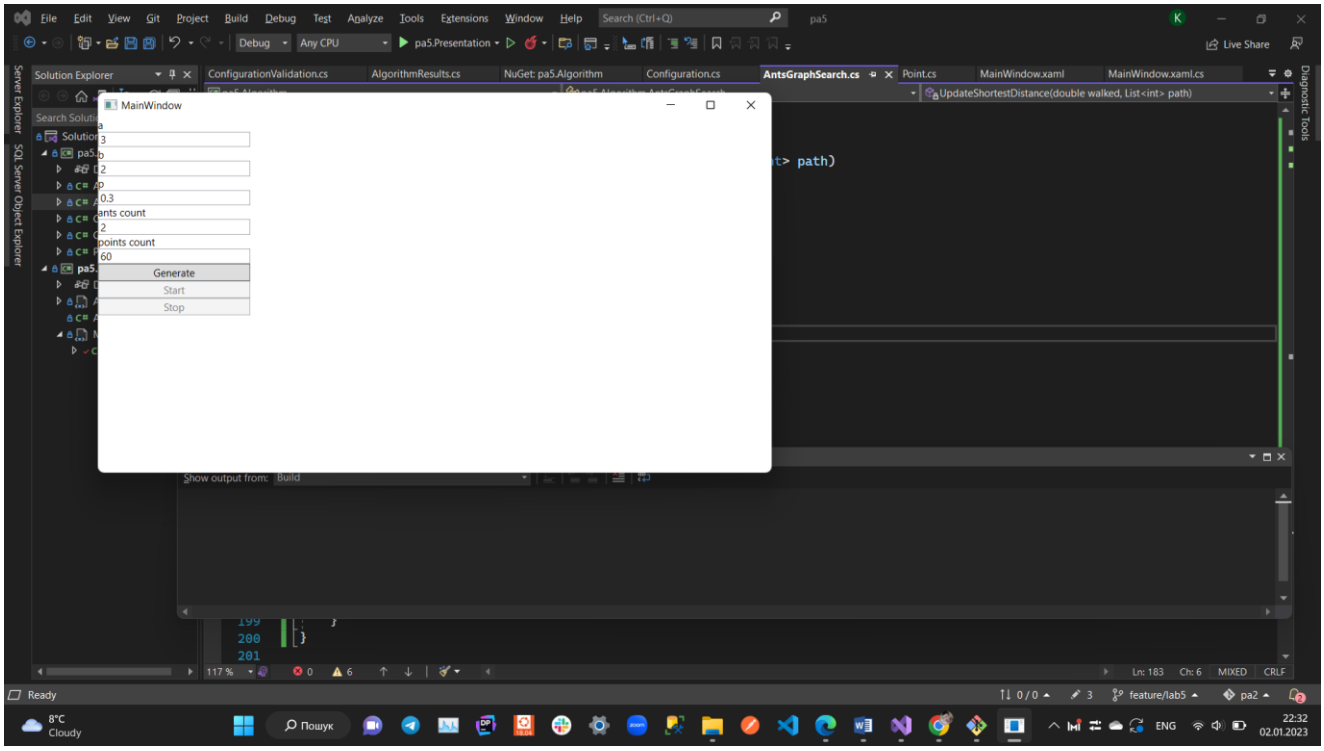
```
{
    for (var i = 0; i < _conf.PointsCount; i++)
    {
        var pos = new Point(Random.Shared.Next(0, 800), Random.Shared.Next(0, 800), -1);
        points.Add(pos);
    }
}

private void CalculateInitialDistancesAndPheramones()
{
    for (var i = 0; i < _conf.PointsCount; i++)
    {
        for (var j = 0; j < _conf.PointsCount; j++)
        {
            var dx = points[i].X - points[j].X;
            var dy = points[i].Y - points[j].Y;
            var dist = Math.Sqrt(dx * dx + dy * dy);
            distances[i, j] = dist;
            feramones[i, j] = 1;
        }
    }
}


public void UpdateRoads()
{
    var pheramonesAdd = new double[_conf.PointsCount, _conf.PointsCount];
    for (var i = 0; i < _conf.AntsCount; i++)
    {
        var walked = 0d;
        var startPos = Random.Shared.Next(_conf.PointsCount);
        var currentPos = startPos;
        var visited = new bool[_conf.PointsCount];
        visited[currentPos] = true;
        var path = new List<int>
        {
            currentPos
        };

        for (var k = 1; k <= _conf.PointsCount; k++)
        {
            int newPos = GetNextPos(startPos, currentPos, visited, k);
            walked += distances[currentPos, newPos];
            var add = 1 / walked;
            pheramonesAdd[currentPos, newPos] += add;
            pheramonesAdd[newPos, currentPos] += add;
            visited[newPos] = true;
            path.Add(newPos);
            currentPos = newPos;
        }

        UpdateShortestDistance(walked, path);
    }

    UpdatePheramones(pheramonesAdd);
}

private int GetNextPos(int startPos, int curPos, bool[] visited, int k)
{
    var newPos = 0;
    if (k < _conf.PointsCount)
    {
        var chances = CalculateChances(curPos, visited);
        double chanceSum = chances.Sum();
        var chanceRanges = CalculateChancesRanges(chances, chanceSum);
        newPos = GetNewPos(chanceRanges);
    }
    else
    {
        newPos = startPos;
    }
```

```
    return newPos;
}

private int GetNewPos(List<double> chanceRanges)
{
    var newPos = 0;
    var random = Random.Shared.NextDouble();
    while (newPos < _conf.PointsCount && random > chanceRanges[newPos])
    {
        newPos++;
    }
    newPos--;
    return newPos;
}

private List<double> CalculateChancesRanges(List<double> chances, double chanceSum)
{
    var chanceRanges = new List<double>();
    for (var j = 0; j < _conf.PointsCount; j++)
    {
        chances[j] /= chanceSum;
        if (j == 0)
        {
            chanceRanges.Add(0);
        }
        else
        {
            chanceRanges.Add(chanceRanges[j - 1] + chances[j - 1]);
        }
    }

    return chanceRanges;
}

private List<double> CalculateChances(int curPos, bool[] visited)
{
    var chances = new List<double>();
    for (var j = 0; j < _conf.PointsCount; j++)
    {
        if (visited[j])
        {
            chances.Add(0);
        }
        else
        {
            var chance = Math.Pow(feramones[curPos, j], _conf.Alpha)
                * Math.Pow(1 / distances[curPos, j], _conf.Beta);
            chances.Add(chance);
        }
    }

    return chances;
}

private void UpdateShortestDistance(double walked, List<int> path)
{
    if (walked < Results.ShortestDistance)
    {
        Results.ShortestDistance = walked;
        for (var j = 1; j < path.Count; j++)
        {
            points[path[j - 1]].Path = path[j];
        }
    }
}

private void UpdatePheramones(double[,] pheramonesAdd)
{
    for (var i = 0; i < _conf.PointsCount; i++)
    {
```

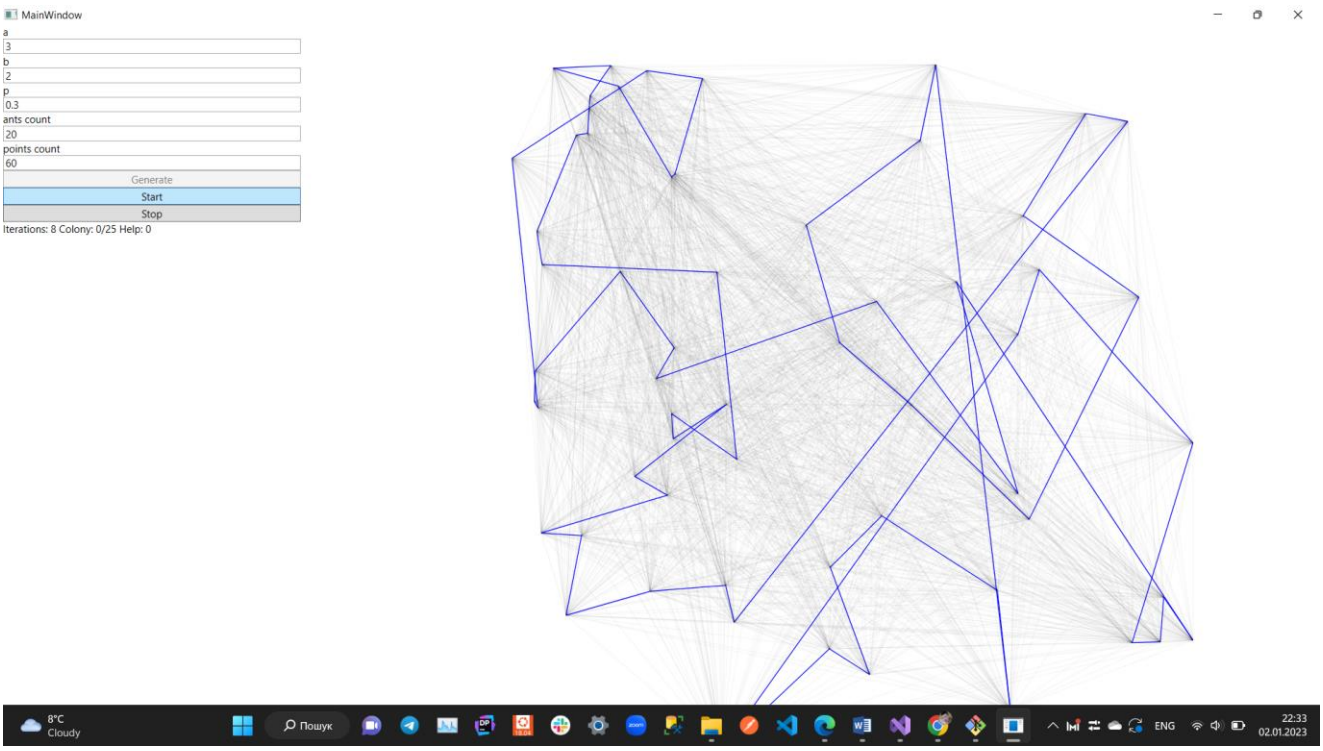
```
            for (var j = 0; j < _conf.PointsCount; j++)
            {
                feramones[i, j] = feramones[i, j] * (1 - _conf.Ro) + pheramonesAdd[i, j];
            }
        }

        if (Results.Iterations == 0)
        {
            Results.InitialShortestDistance = Results.ShortestDistance;
        }
    }
}
```

2. Виконання

1:



2:



3. Тестування алгоритму

Вершин: 200

alpha	beta	ro	M	Lmin	Score
1	2	0.1	20	1	18543,43
1	2	0.1	30	1	17643,6
1	2	0.1	35	1	19243,98
1	2	0.1	40	1	22943,48
1	2	0.1	50	1	21621,51
1	2	0,05	40	1	225423,84
1	2	0,3	40	1	16012,22
1	2	0,5	40	1	12012,62
3	2	0,1	40	1	752312,57
5	2	0,1	40	1	354332,68
1	1	0,1	40	1	134143,25
1	5	0,1	40	1	33712,42
1	5	0,1	40	1	24932,73
1	5	0,1	40	1	25412,37
1	5	0,1	40	10	2379,86
1	5	0,1	40	5	21675,53
alpha	beta	P	M	Lmin	
1	5	0,1	40	1	

4. ВИСНОВОК

При виконанні лабораторної роботи було ознайомлено з способом підбору аргументів для ефективної роботи метаевристичних алгоритмів. Реалізація алгоритму комівояжера на основі мурашиного алгоритму створеного в попередній лабораторній роботі була розширена і було додано підтримка змішаного типу цієї задачі. Потім на цій модифікованій версії алгоритму було проведено підбір різних аргументів для його оптимальної роботи. На основі результатів цього підбору можна зробити висновок, що метаеврестичні алгоритми дуже чутливі до різних вхідних аргументів і правильне їх обрання є необхідним кроком при використанні таких алгоритмів.

5. КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 11.12.2022 включно максимальний бал дорівнює – 5. Після 11.12.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- покроковий алгоритм – 15%;
- програмна реалізація алгоритму – 50%;
- тестування алгоритму– 30%;

висновок – 5%.