ДОДАТОК А
Вихідний код програми

```
public abstract class ClusteringAlgorithm {
    /**
     * Data set of algorithm.
     */
    private DataSet dataSet;
    /**
     * Data loader of algorithm.
     */
    private DataLoader dataLoader;
    /**
     * Centroids set of algorithm.
     */
    protected CentroidsSet centroidsSet;
    /**
     * Matrix of belongs of algorithm.
     */
    protected BelongingMatrix belongingMatrix;
    /**
     * Epsilon value of algorithm (Condition to stop the algorithm when difference between centroids of two
iterations is lower than epsilon).
     * Default value is 0.01.
     */
    private double epsilon = 0.01;
    /**
     * Centroids number of algorithm.
     */
    private int centroidsNumber;
    /**
     * Amount of data in percents to process (learn).
     * @since 1.2.
     */
    private double processedDataAmount;

    /**
     * Constructor that creates algorithm with specified number of clusters.
     * @param clustersNumber    Clusters number.
     */
    public ClusteringAlgorithm(int clustersNumber){
        centroidsNumber = clustersNumber;
    }

    /**
     * Gets an amount of data in percents to process.
     * @return Returns an amount of data in percents to process.
     * @since 1.2.
     */
    public double getProcessedDataAmount() {
        return processedDataAmount;
    }

    /**
     * Sets an amount of data in percents to process.
     * @param aProcessedDataAmount    Amount of data in percents to process.
     * @since 1.2.
     */
    public void setProcessedDataAmount(double aProcessedDataAmount) {
        processedDataAmount = aProcessedDataAmount;
```

```
    }

    /**
     * Loading the data file with specified file path and splitter.
     * @param aFilePath    Data file path to load.
     * @param aSplitter    Splitter of vector of values.
     */
    public void loadFile(String aFilePath, String aSplitter){
        dataLoader = new DataLoader(aFilePath, aSplitter, processedDataAmount);
        dataLoader.loadFile();
    }

    /**
     * Loading the image file with specified file path.
     * @param aFilePath    Image data file path to load.
     * @since 1.1.
     */
    public void loadImage(String aFilePath){
        dataLoader = new DataLoader(aFilePath);
        dataLoader.setProcessedDataAmount(processedDataAmount);
        dataLoader.loadImageFile();
    }

    /**
     * Fills the DataSet from data file with specified size of matrix of values.
     * @param vectorLength    Length of the DataObject's vector (if vector needed).
     *                        If 0, the length of vector will be determined depending on data file contain (Vector length
will be the same as number of values in a string. In each string must be an equal number of values).
     */
    public void fillDataSet(int vectorLength){
        dataSet = dataLoader.fillDataSet(vectorLength);
        initCentroidsAndBelong();
    }

    /**
     * Fills the DataSet from data file with specified size of matrix of values.
     * @param matrixHeight    Height of the DataObject's matrix. Can not be 0.
     *                        You can use fillDataSet(int vectorLength) in case if matrixHeight is equals to 1 (if vector
needed).
     * @param matrixWidth    Width of the DataObject's matrix or length of the vector (if vector needed).
     *                        If 0, the length of vector will be determined depending on data file contain (Vector length
will be the same as number of values in a string. In each string must be an equal number of values).
     *                        If 0, matrixHeight must be 1.
     */
    public void fillDataSet(int matrixHeight, int matrixWidth){
        dataSet = dataLoader.fillDataSet(matrixHeight, matrixWidth);
        initCentroidsAndBelong();
    }

    /**
     * Fills the DataSet from image file with specified size of matrix of values. Use after loading of image file.
     * @param matrixHeight    Height of the DataObject's matrix. Can not be 0. If vector needed set equals to 1.
     * @param matrixWidth     Width of the DataObject's matrix or length of the vector (if vector needed). Can
not be 0.
     * @since 1.1.
     */
    public void fillDataSetFromImage(int matrixHeight, int matrixWidth){
        dataSet = dataLoader.fillDataSetFromImage(matrixHeight, matrixWidth);
        initCentroidsAndBelong();
    }

    /**
     * Initializes of centroids set and matrix of belonging values.
```

```
 */
private void initCentroidsAndBelong(){
    dataSet.setMatrixSize();
    dataSet.centralizeAll();
    dataSet.normalizeAll();
    centroidsSet = new CentroidsSet(centroidsNumber, dataSet.getMatrixHeight(), dataSet.getMatrixWidth());
    belongingMatrix = new BelongingMatrix(dataSet.getObjectsNumber(), centroidsNumber);
    calculateCentroids();
}

/**
 * Prints the matrix of values of each DataObject in the list with their indexes.
 */
public void printDataSet(){
    dataSet.printDataSet();
}

/**
 * Prints the matrix of values of each centroid (DataObject) in the list with their indexes.
 */
public void printCentroidsSet(){
    centroidsSet.printCentroidsSet();
}

/**
 * Prints the list of vector of belongs with their indexes.
 */
public void printBelongingMatrix(){
    belongingMatrix.printBelongingMatrix();
}

/**
 * Prints the vector of the clusters with the maximum belong value for each object.
 */
public void printBelongClusters(){
    belongingMatrix.printBelongClusters();
}

/**
 * Gets DataSet of algorithm.
 * @return Returns link on DataSet of algorithm.
 */
public DataSet getDataSet() {
    return dataSet;
}

/**
 * Gets DataLoader of algorithm.
 * @return Returns link on DataLoader of algorithm.
 */
public DataLoader getDataLoader() {
    return dataLoader;
}

/**
 * Gets CentroidsSet of algorithm.
 * @return Returns link on CentroidsSet of algorithm.
 */
public CentroidsSet getCentroidsSet() {
    return centroidsSet;
}

/**
```

```
 * Gets BelongingMatrix of algorithm.
 * @return Returns link on BelongingMatrix of algorithm.
 */
public BelongingMatrix getBelongingMatrix() {
   return belongingMatrix;
}

/**
 * Gets epsilon of algorithm.
 * @return Returns epsilon value of algorithm.
 */
public double getEpsilon() {
   return epsilon;
}

/**
 * Sets epsilon of algorithm.
 * @param anEpsilon    An epsilon value (condition) of algorithm.
 */
public void setEpsilon(double anEpsilon) {
   epsilon = anEpsilon;
}

/**
 * Gets centroids number of algorithm.
 * @return Returns centroids number of algorithm.
 */
public int getCentroidsNumber() {
   return centroidsNumber;
}

/**
 * Sets centroids number of algorithm.
 * @param aCentroidsNumber    Centroids number of algorithm.
 */
public void setCentroidsNumber(int aCentroidsNumber) {
   centroidsNumber = aCentroidsNumber;
}

/**
 * Abstract method of calculating of matrix of belong.
 * @param isProcessedToCalculate    Is to calculate belong vector for object processed due data analysis.
 */
public abstract void calculateBelongingMatrix(boolean isProcessedToCalculate);

/**
 * Abstract method of calculating of centroids matrices.
 */
public abstract void calculateCentroids();

/**
 * Iterative method of algorithm execution.
 * Iteratively calculates matrix of belong values and centroids.
 * Prints number of iteration, current value of epsilon, and time elapsed for iteration execution.
 * @param anEpsilonValue    An epsilon value (condition) of algorithm.
 * @since 1.1.
 */
public void doAlgorithm(double anEpsilonValue){
   epsilon = anEpsilonValue;
   doAlgorithm();
}

/**
```

```
     * Iterative method of algorithm execution.
     * Iteratively calculates matrix of belong values and centroids.
     * Prints number of iteration, current value of epsilon, and time elapsed for iteration execution.
     */
    public void doAlgorithm(){
        int iterationNumber = 0;
        while(centroidsSet.calculateDifferenceBetweenIterations() > epsilon){
            long currentTime = System.nanoTime();
            calculateBelongingMatrix(true);
            calculateCentroids();
            iterationNumber++;
            System.out.println(String.format("Iteration: %1$-4d ε: %2$-10.6f Time: %3$.5f s", iterationNumber,
centroidsSet.calculateDifferenceBetweenIterations(), ((double)(System.nanoTime() - currentTime) /
Math.pow(10, 9))));
            if(iterationNumber == 1 || iterationNumber ==
20){calculateBelongingMatrix(false);printBelongingMatrix();}
        }
        calculateBelongingMatrix(false);
    }

    /**
     * Changing of the size of the each matrix of values and centroids.
     * Prints time of transposing.
     * Not works if the matrix of size is zero or the product of the old sizes not equals to the product of the new
sizes.
     * @param newMatrixHeight    The new matrix height of data objects.
     * @param newMatrixWidth     The new matrix width of data objects.
     */
    public void transformAll(int newMatrixHeight, int newMatrixWidth){
        long currentTime = System.nanoTime();
        dataSet.transformAll(newMatrixHeight, newMatrixWidth);
        centroidsSet.transformAll(newMatrixHeight, newMatrixWidth);
        System.out.println(String.format("Data transformation time: %1$.5f s", ((double)(System.nanoTime() -
currentTime) / Math.pow(10, 9))));
    }

    /**
     * Gets membership matrix of patterns of data.
     * @return Membership matrix of patterns.
     * @since 1.1.
     */
    public double[][] getMembershipMatrix(){
        return belongingMatrix.getMembershipMatrix();
    }

    /**
     * Gets the vector of the clusters with the maximum belong value for each object.
     * @return Returns the vector of the clusters with the maximum belong value for each object.
     * @since 1.1.
     */
    public double[][] getMemberClusters(){
        return belongingMatrix.getMemberClusters();
    }

    /**
     * Shows frame with original picture.
     * @since 1.1.
     */
    public void showOriginalImage(){
        ImageFrame i = new ImageFrame(dataLoader.getDataImageFile(), "Original Image");
    }

    /**
```

```
    * Get color of each cluster in array to show result of clustering in image.
    * @return Returns color of each cluster in array to show result of clustering in image.
    * @since 1.1.
    */
   public int[] getClusterColor(){
      double[] clusterColor = new double[centroidsNumber];
      for(int i = 0; i < clusterColor.length; i++){
         for(int j = 0; j < dataSet.getObjectsNumber(); j++){
            if(belongingMatrix.getBelongVector(j).getBelongCluster() == i){
               clusterColor[i] = dataSet.getDataObject(j).getAverageColorValue();
               break;
            }
         }
      }
      int[] clusterColorToReturn = new int[centroidsNumber];
      /*for(int i = 0; i < centroidsNumber; i++){
         clusterColorToReturn[i] = (int)(255 * clusterColor[i]);
      }*/
      int minIndex = 0;
      for(int i = 0; i < centroidsNumber; i++){
         minIndex = 0;
         for(int j = 0; j < centroidsNumber; j++){
            if(clusterColor[j] < clusterColor[minIndex])
               minIndex = j;
         }
         clusterColorToReturn[minIndex] = i * 255 / (centroidsNumber - 1);
         clusterColor[minIndex] = 2;
      }
      return clusterColorToReturn;
   }

   /**
    * Generates image after processing, with cluster colours.
    * @return Returns image after processing, with cluster colours.
    * @since 1.1.
    */
   public BufferedImage generateProcessedImage(){
      BufferedImage imageToReturn = new BufferedImage((dataLoader.getDataImageFile().getWidth() +
dataSet.getMatrixWidth() - 1)/dataSet.getMatrixWidth(), (dataLoader.getDataImageFile().getHeight() +
dataSet.getMatrixHeight() - 1)/dataSet.getMatrixHeight(), BufferedImage.TYPE_INT_RGB);
      int[] clusterColor = getClusterColor();
      for(int i = 0; i < imageToReturn.getHeight(); i++){
         for(int j = 0; j < imageToReturn.getWidth(); j++){
            imageToReturn.setRGB(j, i, clusterColor[belongingMatrix.getBelongVector(i *
imageToReturn.getWidth() + j).getBelongCluster()] * 65793);
         }
      }
      return imageToReturn;
   }

   /**
    * Shows frame with original picture.
    * @since 1.1.
    */
   public void showProcessedImage(){
      ImageFrame i = new ImageFrame(generateProcessedImage(), "Processed Image");
```

ДОДАТОК Б
Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	78 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Анісімов В.Е.			Використання методів адаптивного нечіткого кластерування для вирішення завдання опрацювання матричних даних	Шифр групи	Код напр./спец.
Перевірив		Кулішова Н.Є.				СШІм-20-3	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					