

ДОДАТОК А

Текст програми

alternative

```
public class Alternative {  
    int index;  
    float[] criteriaValues;  
    Alternative(int criteriaQuantity) {  
        index = 0;  
        criteriaValues = new float[criteriaQuantity];  
        for (int i = 0; i < criteriaQuantity; i++) {  
            criteriaValues[i] = 0;  
        }  
    }  
    Alternative(int index, float[] values) {  
        criteriaValues = new float[values.length];  
        criteriaValues = values;  
        this.index = index;  
    }  
};
```

BestAndWorst

```
public class BestAndWorst {  
    public int indexOfBest, indexOfWorst;
```

```

public float best, worst;

BestAndWorst(){
    indexOfBest = 0;
    indexOfWorst = 0;
    best = 0f;
    worst = 0f;
}
};

```

MyMath

```

import java.util.Arrays;
import java.util.Collections;
import java.util.Random;
import java.util.Scanner;
import java.lang.Math;

public class MyMath {
    static Random rng = new Random(11131);//11131

    public static float getRandomNumber(int min, int max){
        float randomNumber = 0f;
        if ( (min == 0) && (max == 1) ) {
            randomNumber = rng.nextFloat();

```

```

    } else {

int temp = 0;

temp = min + rng.nextInt(max + 1 - min); // поумолчанию nextInt
выдаёт значение не включительно, поэтому добавляем 1

randomNumber = (float)temp;

    }

return randomNumber;

}

public static float[][] getAlternativesArray(int quantity, int[][] criteriaRanges){

    /*

    параметр quantity - кількість альтернатив; criteriaRanges -
    діапазон значень по кожному із критеріїв;

    isHigherBetter - показує чи краще якщо значення по критерію більше, чи
    навпаки

    */

float[][] alternatives = new float[criteriaRanges.length][quantity];

for (int i = 0; i < alternatives.length; i++) {
int min = criteriaRanges[i][0];
int max = criteriaRanges[i][1];
for (int j = 0; j < alternatives[i].length; j++) {
alternatives[i][j] = getRandomNumber(min, max);
}
}

return alternatives;

}

```

```
public static BestAndWorst[] getBestAndWorst(float[][] array, boolean[]
isHigherBetter) {
```

```
/*
```

Дана функція видає масив найкращих та найгірших значень за всіма критеріями

та номери альтернатив, що мають ці значення. Для цього створили

тип даних (клас) BestAndWorst

перший параметр - масив альтернатив, другий - чи є більші значення по критерію кращими

```
*/
```

```
BestAndWorst[] bestAndWorst = new BestAndWorst[array.length];
```

```
for (int i = 0; i < bestAndWorst.length; i++) { bestAndWorst[i] = new
BestAndWorst(); }
```

```
for (int i = 0; i < array.length; i++){
```

```
float min = 999999.0f, max = 0.0f;
```

```
for (int j = 0; j < array[i].length; j++){
```

```
if (array[i][j] < min){
```

```
        /* Знаходимо мінімум */
```

```
min = array[i][j];
```

```
if (!isHigherBetter[i]){
```

```
/*
```

Якщо для критерія чим менше значення тим краще, то мінімальне значення - найкращий варіант

```
*/
```

```
bestAndWorst[i].indexOfBest = j;
```

```
bestAndWorst[i].best = min;
```

```
} else {
```

```
/*
```

Якщо для критерія чим більше значення тим краще, то

мінімальне значення - найгірший варіант

```
*/
```

```
bestAndWorst[i].indexOfWorst = j;
```

```
bestAndWorst[i].worst = min;
```

```
}
```

```
}
```

```
if (array[i][j] > max){
```

```
/* Знаходимо максимум */
```

```
max = array[i][j];
```

```
if (isHigherBetter[i]){
```

```
bestAndWorst[i].indexOfBest = j;
```

```
bestAndWorst[i].best = max;
```

```
} else {
```

```
bestAndWorst[i].indexOfWorst = j;
```

```
bestAndWorst[i].worst = max;
```

```
}
```

```
}
```

```
}
```

```
}
```

```
return bestAndWorst;
```

```
}
```

```

public static float[][] normalize(float[][] array, BestAndWorst[] arg2, int[][] ranges) {
    /* Функціяполезности */
    float[][] normalizedArray = new float[array.length][array[0].length];
    for (int i = 0; i < array.length; i++) {
        if ( !((ranges[i][0]) == 0 && (ranges[i][1] == 1))) { /*
перевіряємочипотрібнонормалізувати */
            for (int j = 0; j < array[i].length; j++) {
                normalizedArray[i][j] = Math.abs( (array[i][j] -
arg2[i].worst)/(arg2[i].best - arg2[i].worst) );
            }
        } else {
            normalizedArray[i] = array[i];
        }
    }
    return normalizedArray;
}

```

```

public static byte compareArrays(float[] array1, float[] array2) {
    byte result = 5;
    if ( (array1[0] > array2[0]) && (array1[1] > array2[1]) && (array1[2] > array2[2])
&& (array1[3] > array2[3]) ) {
        result = 0; /* функція повертає 0, якщо перша альтернатива краща за другу за
всіма критеріями */
    }
    if ( (array1[0] < array2[0]) && (array1[1] < array2[1]) && (array1[2] < array2[2])
&& (array1[3] < array2[3]) ) {

```

```
result = 1; /* функція повертає 1, якщо перша альтернатива гірша за другу за
всіма критеріями */
```

```
}
```

```
    else { result = 2; } /* функція повертає 2, якщо перша альтернатива за
якимись критеріями краще, за якимись менше*/
```

```
return result;
```

```
}
```

```
public static float[] fillWithZeroes(float[] inputArray) {
```

```
    float[] outputArray = new float[inputArray.length];
```

```
    for (int i = 0; i < inputArray.length; i++) {
```

```
        outputArray[i] = 0;
```

```
    }
```

```
    return outputArray;
```

```
}
```

```
public static boolean isZero(float[] arr) {
```

```
    if ( (arr[0] == 0) && (arr[1] == 0) && (arr[2] == 0) && (arr[3] == 0) ) {
```

```
        return true;
```

```
    } else {
```

```
        return false;
```

```
    }
```

```
}
```

```
/* функция принимает массив альтернатив и возвращает меньший массив
альтернатив, что нельзя улучшить ни по одному из крит. */
```

```

public static Alternative[] pairComparison(Alternative[] inputAlternatives) {
    Alternative[] outputAlternatives;

    byte temp;

    int deletedAlternativesCounter = 0;

    /* спочатку потрібно визначити скільки альтернатив видаляється */

    outerLoop:
    for (int i = 0; i < inputAlternatives.length; i++) {

        if ( isZero(inputAlternatives[i].criteriaValues) ) { continue; } /*
        якщо всі значення нульові значить альтернативу видалено */

        for (int j = 0; j < inputAlternatives.length; j++) {

            if (i == j) { continue; } /* немає сенсу порівнювати альтернативу самою з собою */

            if ( isZero(inputAlternatives[j].criteriaValues) ) { continue; } /*
            якщо всі значення нульові значить альтернативу видалено */

            temp = compareArrays(inputAlternatives[i].criteriaValues,
                                inputAlternatives[j].criteriaValues);

            if (temp == 0) {

                deletedAlternativesCounter++;

                inputAlternatives[j].criteriaValues =
                fillWithZeroes(inputAlternatives[j].criteriaValues);

            } else if (temp == 1) {

                deletedAlternativesCounter++;

                inputAlternatives[i].criteriaValues =
                fillWithZeroes(inputAlternatives[i].criteriaValues);

                continue outerLoop;

            } else if (temp == 2) { continue; }

        }

    }
}

```

```
outputAlternatives = new Alternative[inputAlternatives.length -
deletedAlternativesCounter];
```

```
int counter = 0;
```

```
for (int i = 0; i < inputAlternatives.length; i++) {
```

```
if ( !(isZero(inputAlternatives[i].criteriaValues)) ) {
```

```
outputAlternatives[counter] = inputAlternatives[i];
```

```
counter++;
```

```
}
```

```
}
```

```
return outputAlternatives;
```

```
}
```

```
public static Alternative findBestAlternativeByCriterion(A Alternative[], int
criterionNo) {
```

```
/* В дану функцію в якості параметрів подається масив альтернатив та номер
критерію
```

Функція повертає найкращу альтернативу за заданим критерієм

```
*/
```

```
Alternative outputAlt = new Alternative(inputAlts[0].criteriaValues.length);
```

```
float max = 0f;
```

```
for (int i = 0; i < inputAlts.length; i++) {
```

```
if ( inputAlts[i].criteriaValues[criterionNo] > max) {
```

```
max = inputAlts[i].criteriaValues[criterionNo];
```

```
outputAlt = inputAlts[i];
```

```
}
```

```

    }

return outputAlt;

}

public static Alternative[] expandAltArray(Alternative[] inputAlts, Alternative
bestAlt, int criterionNo, float plusMinus) {

/* В дану функцію в якості параметрів подається масив альтернатив,
    найкраща альтернатива за критерієм, номер цього критерія та діапазон
    розширення

    Функція повертає розширений масив альтернатив
    */

Alternative[] outputAlts;

float rangeFrom = bestAlt.criteriaValues[criterionNo] - plusMinus;

float rangeTo = bestAlt.criteriaValues[criterionNo] + plusMinus;

int fittingAltsCount = 0; // кількість альтернатив, що підходять по умовам

/* Спочатку визначаємо скільки альтернатив задовольняють умовам, що задати
    розмір вихідного масиву */

for (int i = 0; i < inputAlts.length; i++) {

if ( (inputAlts[i].criteriaValues[criterionNo] > rangeFrom) &&
(inputAlts[i].criteriaValues[criterionNo] < rangeTo) ) {

fittingAltsCount++;

    }

}

outputAlts = new Alternative[fittingAltsCount]; /* задаємо розмір вихідного масиву
    */

/* Тепер вносимо значення в вихідний масив */

int counter = 0;

```

```

for (int i = 0; i < inputAlts.length; i++) {
    if ( (inputAlts[i].criteriaValues[criterionNo] > rangeFrom) &&
        (inputAlts[i].criteriaValues[criterionNo] < rangeTo) ) {
        outputAlts[counter] = inputAlts[i];
        counter++;
    }
}

return outputAlts;
}

```

```

public static Alternative[] lexicograficOptimization(Alternative[] inputAlts, int[]
criteriaImportance) {

```

/* Функція приймає вхідний масив альтернатив та масив, в якому критерії розташовані в порядку важливості

Функція повертає двухмірний масив альтернатив

Перший рядок - найкращі за найважливішим критерієм, другий - найкращі за другим по важливості і так далі

Останній рядок результат

*/

```

Alternative[] outputAlts, currentAlts, previousAlts;

Alternative[] bestAtlsByCrit = new Alternative[criteriaImportance.length];

previousAlts = inputAlts;

for (int i = 0; i < criteriaImportance.length; i++) {

    bestAtlsByCrit[i] = findBestAlternativeByCriterion(previousAlts,
criteriaImportance[i]);

    currentAlts = expandAltArray(previousAlts, bestAtlsByCrit[i], criteriaImportance[i],
0.2f);

```

```

previousAlts = currentAlts;

    }

return previousAlts;

}

```

```

// public static float[][] pairComparison(float[][] inputArray) {
//     Alternative[] alt = new Alternative[inputArray[0].length];
//     for (int i = 0; i < alt.length; i++) {
//         alt[i].index = i;
//         alt[i].criteriaValues = inputArray[i]
//     }
// }

// public static byte[] calculateCriteriaImportance() {
//     /* Теперь определяем степени важности критериев
//     0 - Производительность
//     1 - Стоимость
//     2 - Надежность
//     3 - точность
//     */
//     byte[] criteria = new byte[4]; // Самому важному критерию будет
//     присвоено значение 4, наименее важному - 1
//     int counter = 0;
//     Scanner userInput = new Scanner(System.in);

```

```

// String[] criteriaNames = {"Производительность","Стоимость",
"Надёжность","Точность"};

// askUser:

// while (counter<4){

//     int temp = 0;

//     System.out.printf("Выберите самый важный из этих критериев(введите
его номер)\n");

//     for (int i = 0; i<criteriaNames.length;i++){

//         if(criteria[i]==0 && counter!=3){ // Не выводим те критерии, важность
которых уже определена

//             System.out.printf("%d - %s ", i, criteriaNames[i]);

//             } else if(counter == 3 && criteria[i]==0){

//                 // Последний критерий задаём автоматически

//                 criteria[i]=1;

//                 counter++;

//                 continue askUser;

//             }

//         }

//         System.out.printf("\n>> ");

//         temp = userInput.nextByte();

//         if (temp>3 || temp<0){

//             // Проверяем ввёл ли пользователь правильное число, если нет -
повторяем цикл

//             System.out.printf("Введён неправильный номер!\n\n");

//             temp = 0;

//             continue;

```

```

//      } elseif(criteria[temp] !=0){
//      System.out.printf("Важность введённого критерия уже задана!\n\n");
//      temp = 0;
//      continue;
//      } else {
//      criteria[temp] = (byte)(4-counter); // Добавляем единицу, потому что
отсчёт начиннается с 0
//      counter++;
//      }

//  }
//  userInput.close();
//  return criteria;
// }

// public static int[] filterWorst(int leaveAmount,float[] inputArray,int[] inputIndexes)
{
//  /* Удаляем худшие варианты по каждому критерию
//  Для этого сортируем массив от большего к меньшему
//  и заносим в новый массив только нужное кол-во */
//  float[] array = inputArray;
//  float[] result = new float[leaveAmount]; // результирующиймассив
//  int[] outputIndexes = new int[leaveAmount]; // создаём массив индексов,
чтобы потом снова расставить элементы по порядку

//  float temp = 0f;

```

```

//  int n = array.length;

//  for(int i=0; i < n - 1; i++){
//      for(int j=0; j < n-i-1; j++){
//          if(array[j] < array[j+1]){
//              // Меняем местами элементы массива и соответствующие индексы
//              temp = array[j];
//              array[j] = array[j+1];
//              array[j+1] = temp;
//              temp = inputIndexes[j];
//              inputIndexes[j] = inputIndexes[j+1];
//              inputIndexes[j+1] = (int)temp;
//          }
//      }
//  }

//  // Теперь выбираем лишние варианты
//  for(int i=0; i < leaveAmount; i++){
//      result[i] = array[i];
//      outputIndexes[i] = inputIndexes[i];
//  }

//  // Расставляем элементы по порядку

//  for(int i=0; i < inputArray.length; i++){
//      for(int j=1; j < (inputArray.length-i); j++){
//          if(inputIndexes[j-1] > inputIndexes[j]){
//              temp = inputIndexes[j-1];

```

```

//      inputIndexes[j-1] = inputIndexes[j];
//      inputIndexes[j] = (int)temp;
//      temp = array[j-1];
//      array[j-1] = array[j];
//      array[j] = temp;
//  }
//  }
//  }
//  return outputIndexes;
// }

```

// критерий: имя, предел значений, больше-лучше или наоборот

```

// public static void main(String[] args) {
//     /**
//      * Производительность (80-4000)
//      * Стоимость (100-800000)
//      * Надежность (100-2000)
//      * точность ( 0-1)
//      */
//     final byte NUMBER_OF_VARIANTS = 100; // кол-во вариантов
//     final byte PRODUCTIVITY_MIN = 80;
//     final short PRODUCTIVITY_MAX = 4000;
//     final byte COST_MIN = 100;

```



```

//  /* Находим лучшие и худшие по каждому критерию

//  * Находим макс производительность

//  */

//  int[] best_productivity = findMax(productivity); // Дляпроизводительности,
больше - лучше

//  int[] worst_productivity = findMin(toFloat(productivity) );

//  int[] best_cost = findMin(toFloat(cost)); // Длястоимости, меньше - лучше

//  int[] worst_cost = findMax(cost);

//  int[] best_reliability = findMax(reliability); // Длянадёжности, больше -
лучше

//  int[] worst_reliability = findMin(toFloat(reliability) );

//  int[] best_precision = findMax(precision); // Дляточности, больше - лучше

//  int[] worst_precision = findMin(toFloat(precision));

//  System.out.printf("\nК1(производительность): Лучший - №%d: %d,
Худший - №%d: %d\n",best_productivity[0],best_productivity[1],
worst_productivity[0],worst_productivity[1]);

//  System.out.printf("К2(стоимость): Лучший - №%d: %d, Худший - №%d:
%d\n",best_cost[0],best_cost[1], worst_cost[0],worst_cost[1]);

//  System.out.printf("К3(надёжность): Лучший - №%d: %d, Худший - №%d:
%d\n",best_reliability[0],best_reliability[1], worst_reliability[0],worst_reliability[1]);

//  System.out.printf("К4(точность): Лучший - №%d: %d, Худший - №%d:
%d\n",best_precision[0],best_precision[1], worst_precision[0],worst_precision[1]);

//  /**Находим p i */

//  float[] normalizedArray_k1 = findnormalizedArray(productivity,
best_productivity[1], worst_productivity[1]);

```

```

// float[] normalizedArray_k2 = findnormalizedArray(cost, best_cost[1],
worst_cost[1]);

// float[] normalizedArray_k3 = findnormalizedArray(reliability,
best_reliability[1], worst_reliability[1]);

// float[] normalizedArray_k4 = findnormalizedArray(precision,
best_precision[1], worst_precision[1]);


// /* Выводимполучившиесямассивы */

// System.out.printf("\tНомерварианта\t\t\normalizedArray_k1\t
\normalizedArray_k2\t \tnormalizedArray_k3\t \tnormalizedArray_k4\t \n\n\n");

// for (int i = 0; i < 100; i++){

//     System.out.printf("\t\t%d\t\t%f \t%f \t%f \t%f \n",
i+1,normalizedArray_k1[i],normalizedArray_k2[i],normalizedArray_k3[i],normalize
dArray_k4[i]);

// }


// /* Теперь определяем степени важности критериев

//     0 - Производительность

//     1 - Стоимость

//     2 - Надежность

//     3 - точность

// */


// byte[] criteriaImportance = newbyte[4];

// criteriaImportance = calculateCriteriaImportance();

```

```

//  System.out.printf("\n\n");

//  for (int i=0;i<criteriaImportance.length;i++){
//      int sum = 0;
//      for (int j=0;j<criteriaImportance.length;j++){
//          if (i == j){
//              System.out.printf("\t ");
//          } else if (criteriaImportance[i]>criteriaImportance[j]){
//              System.out.printf("\t1");
//              sum++;
//          } else {
//              System.out.printf("\t0");
//          }
//      }
//  }

//  System.out.printf(" | %d\n",sum);
//  }

//  // Расставляем критерии в порядке значимости
//  byte[] criteria = {0,1,2,3};

//  String[] criteriaNames = {"Производительность", "Стоимость",
"Надежность", "Точность"};

//  for(int i=0; i < criteria.length; i++){
//      for(int j=1; j < (criteria.length-i); j++){
//          if(criteriaImportance[j-1] < criteriaImportance[j]){
//              byte temp = 0;
//              String tempString = "";
//              temp = criteria[j-1];

```

```

//      tempString = criteriaNames[j-1];
//      criteria[j-1] = criteria[j];
//      criteriaNames[j-1] = criteriaNames[j];
//      criteria[j] = temp;
//      criteriaNames[j] = tempString;
//      }
//  }
// }

//  System.out.printf("\n\n");
//  for(int i=0; i < criteria.length; i++){
//      System.out.printf("K%d(%s)>",criteria[i]+1,criteriaNames[i]);
//  }

//  System.out.printf("\n\n");


//  // Объединяем в один двумерный массив
//  float[][] variants =
{normalizedArray_k1,normalizedArray_k2,normalizedArray_k3,normalizedArray_k
4};

//  // Чтобы не потерять номера вариантов создадим массив индексов
//  int[] variantIndexes = new int[normalizedArray_k1.length];

//      for (int i = 0; i < variantIndexes.length; i++){
//          variantIndexes[i]=i+1;
//      }

//  int[] leaveAmount = {20,10,5,2}; // в этом массиве указываем сколько
вариантов оставлять по каждому критерию

```

```

// // Отсеиваем варианты по каждому критерию по найденной важности

// float[][] previousVariants = new float[4][100]; // массив с отфильтрованными
значениями

// previousVariants = variants;

// int[] previousIndexes = new int[100]; // массив с индексами лучших
вариантов по каждому критерию

// // Для начала просто заполним массив индексов числами от 0 до 99
// for (int i = 0; i < 100; i++) { previousIndexes[i]=i; }

// int[] newIndexes;

// float[][] newVariants;

// for (int i = 0; i < criteria.length; i++){

//     newIndexes = new int[ leaveAmount[i] ];

//     newVariants = new float[4][ leaveAmount[i] ];//
каждыйновыймассивменьшепредыдущего

//     newIndexes = filterWorst( leaveAmount[i], previousVariants[ criteria[i]
],previousIndexes);

//     previousIndexes = new int [ leaveAmount[i] ];

// previousIndexes = newIndexes;

// // Теперь заполняем массив оптимальных с помощью индексов
// for (int m=0;m<4;m++){

//     for (int n=0;n<leaveAmount[i];n++){

//         newVariants[m][n] = variants[m][ newIndexes[n] ];

//     }

// }

// previousVariants = new float[4][ leaveAmount[i] ];

// previousVariants = newVariants;

// // выводимдляпроверки

```

```

//      System.out.printf("\n\n%dлучшихвариантовпокритериюK%d(%s):\n\n",
leaveAmount[i], criteria[i]+1, criteriaNames[i]);

//      System.out.printf("#\t\tK1\t\tK2\t\tK3\t\tK4\n\n");

//      for (int m=0;m<leaveAmount[i];m++){

//
System.out.printf("%d\t\t%f\t\t%f\t\t%f\t\t%f\n",newIndexes[m]+1,newVariants[0][m],
newVariants[1][m],newVariants[2][m],newVariants[3][m]);

//      }

//      }

//      //filterWorst(20, variants[0]);

// }

};

```

MyFrame

```

import javax.swing.JFrame;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.JPanel;

```

```

import javax.swing.JScrollPane;
import javax.swing.JButton;
import javax.swing.JLabel;
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.*;

```

```

public class MyFrame extends JFrame {
    JTable table1, table2, table3, table4;
    JTextArea textArea, currentStepText;
    JPanel centralPanel, buttonPanel;
    JButton nextStepButton, backButton;
    String[] currentStep;
    JScrollPane scrollPane;
    String[] columnNames;
    int counter = 0;

```

```

    MyFrame(String[][] tableData1, String[][] tableData2, String[] criteriaNames,
    String bestAndWorst, String[][] tableData3, String[][] tableData4) {

```

```

/*

```

Функція MyFrame() - конструктор

конструктор - функція, що виконується при створенні об'єкту даного типу

в даному випадку цей об'єкт створюється в методі main класу Main

```

*/

```

```

currentStep = new String[5];
currentStep[0]="Отримання альтернатив";
currentStep[1]="Знаходження найкращих та найгірших значень по критеріям";
currentStep[2]="Нормалізація значень";
currentStep[3]="Метод парних порівнянь";
currentStep[4]="Метод лексикографічної оптимізації: результат";
centralPanel = new JPanel();
centralPanel.setLayout( new FlowLayout(FlowLayout.CENTER) );
buttonPanel = new JPanel();
nextStepButton = new JButton("Наступний етап >");
backButton = new JButton("< Назад");
currentStepText = new JTextArea("Поточний етап: "+currentStep[counter]);
currentStepText.setEditable(false);
textArea = new JTextArea();

    //buttonPanel.setBackground(Color.GREEN);

buttonPanel.add(currentStepText);
buttonPanel.add(backButton);
buttonPanel.add(nextStepButton);
buttonPanel.setLayout( new FlowLayout(FlowLayout.RIGHT) );
backButton.setEnabled(false);

/*

    Створюємо масив з назвами стовбців таблиці
    Перший стовбець - номер альтернативи
    Інші стовбці - назви критеріїв

*/

```

```

columnNames = new String[criteriaNames.length + 1];
columnNames[0] = "№ Альтернативи";
for (int i = 1; i < columnNames.length; i++){
    columnNames[i] = criteriaNames[i - 1];
}

/* Початкові налаштування таблиці */

table1 = new JTable(tableData1, columnNames);
table1.setEnabled(false);

table2 = new JTable(tableData2, columnNames);
textArea.setText(bestAndWorst);
table2.setEnabled(false);

table3 = new JTable(tableData3, columnNames);
table3.setEnabled(false);

table4 = new JTable(tableData4, columnNames);
table4.setEnabled(false);

//centralPanel.add(table1);
//centralPanel.add(textArea);
//centralPanel.add(table2);

scrollPane = new JScrollPane(table1);
//scrollPane.setViewportView(table1);

/* Обробники натискань кнопок */
backButton.addActionListener(new ActionListener(){

/* Функція, що виконується при натисканні кнопки "Назад" */
@Override

```

```
public void actionPerformed(ActionEvent e) {  
    counter--;  
    switch (counter) {  
        case 0:  
            backButton.setEnabled(false);  
            setTitle("Початковий масив альтернатив");  
            scrollPane.setViewportView(table1);  
            break;  
        case 1:  
            scrollPane.setViewportView(textArea);  
            setTitle("Найкращі та найгірші");  
            break;  
        case 2:  
            scrollPane.setViewportView(table2);  
            setTitle("Нормалізовані значення");  
            break;  
        case 3:  
            scrollPane.setViewportView(table3);  
            setTitle("Метод парних порівнянь");  
            break;  
        case 4:  
            scrollPane.setViewportView(table4);  
            setTitle("Метод лексикографічної оптимізації: результат");  
            break;  
    }  
}
```

```

currentStepText.setText("Поточний етап: "+currentStep[counter]);
update(getGraphics());
    }
});

nextStepButton.addActionListener(new ActionListener(){
/* Функція, що виконується при натисканні кнопки "Наступний етап" */
@Override
public void actionPerformed(ActionEvent e) {
    if (counter < currentStep.length - 1) { counter++; backButton.setEnabled(true);}
    switch (counter) {
    case 1:
        scrollPane.setViewportView(textArea);
        setTitle("Найкращі та найгірші");
        break;
    case 2:
        scrollPane.setViewportView(table2);
        setTitle("Нормалізовані значення");
        break;
    case 3:
        scrollPane.setViewportView(table3);
        setTitle("Метод парних порівнянь");
        break;
    case 4:
        scrollPane.setViewportView(table4);
        setTitle("Метод лексикографічної оптимізації: результат");

```

```

break;
    }

currentStepText.setText("Поточний етап: "+currentStep[counter]);
update(getGraphics());
    }
});

/* Початкові налаштування вікна */

setLayout( new BorderLayout() );
    //add(textArea, BorderLayout.WEST);
add(scrollPane, BorderLayout.CENTER);
setTitle("Початковий масив альтернатив");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

add(buttonPanel, BorderLayout.SOUTH);
setMinimumSize(new Dimension(800, 600));
setVisible(true);
    }
};

```

Main

```

public class Main {

    /* Перед тим як виконувати код задаємо неохідні змінні; final - константа */
    final static byte NUMBER_OF_VARIANTS = 100; // кількість альтернатив

    /* Задаємо діапазон значень по критеріях */
    final static int[][] CRITERIA_RANGES = {

        { 5, 100 }, // продуктивність
        { 100, 800000 }, // ціна
        { 0, 1 }, // надійність
        { 0, 1 } // точність

    };

    /* Задаємо чи є більші значення за критерієм кращими */
    final static boolean[] isHigherBetter = {

        true, // продуктивність
        false, // ціна
        true, // надійність
        true // точність

    };

    /* Задаємо назви критеріїв */
    final static String[] criteriaNames = {"Продуктивність", "Ціна", "Надійність",
        "Точність"};

    static BestAndWorst[] bestAndWorst = new BestAndWorst[criteriaNames.length];

    final static int[] criteriaImportance = {1,0,2,3}; /* Вказуємо критерії від
        найважливішого до найменш важливого */

    public static void main(String[] args) {

```

```

/* Отримуємо масив альтернатив */

float[][] alternatives = new
float[CRITERIA_RANGES.length][NUMBER_OF_VARIANTS];

float[][] normalizedAlternatives = new
float[CRITERIA_RANGES.length][NUMBER_OF_VARIANTS];

Alternative[] normalizedAlternativesObj = new
Alternative[NUMBER_OF_VARIANTS];

alternatives = MyMath.getAlternativesArray(NUMBER_OF_VARIANTS,
CRITERIA_RANGES);

//normalizedAlternatives = MyMath.normalize(array, best, worst,
isHigherBetter);


/* Знаходимо найкращі та найгірші */

bestAndWorst = MyMath.getBestAndWorst(alternatives, isHigherBetter);

String bestAndWorstString = "";

for (int i = 0; i < alternatives.length; i++) {

    bestAndWorstString += "Найкраще значення по критерію
K" + (i + 1) + "(" + criteriaNames[i] + ") = "

        + bestAndWorst[i].best + ", альтернатива
№" + (bestAndWorst[i].indexOfBest + 1) + "\n";

    bestAndWorstString += "Найгірше значення по критерію
K" + (i + 1) + "(" + criteriaNames[i] + ") = "

        + bestAndWorst[i].worst + ", альтернатива
№" + (bestAndWorst[i].indexOfWorst + 1) + "\n";

}

```

```

/* Нормалізуємо значення */

normalizedAlternatives = MyMath.normalize(alternatives,
bestAndWorst,CRITERIA_RANGES);

float[][] temp = new
float[normalizedAlternatives[0].length][normalizedAlternatives.length]; /* массив
100 x 4, вместо 4 x 100 */

for (int i = 0; i < normalizedAlternatives.length; i++) {
for (int j = 0; j < normalizedAlternatives[i].length; j++) {
temp[j][i] = normalizedAlternatives[i][j];
    }
}

for (int i = 0; i < temp.length; i++) {
    //normalizedAlternativesObj[i].index = i + 1; // заполняем массив объектов
    //normalizedAlternativesObj[i].criteriaValues = temp[i];
normalizedAlternativesObj[i] = new Alternative(i + 1, temp[i]);
}

/* Метод парних порівнянь */

Alternative[] pairCompAlternatives =
MyMath.pairComparison(normalizedAlternativesObj);

/* Метод лексикографічної оптимізації */

Alternative[] result = MyMath.lexicographicOptimization(pairCompAlternatives,
criteriaImportance);

```

/* Форматуємо отримані дані для показу в таблиці */

```
String[][] tableData1 = new String[alternatives[0].length][alternatives.length + 1]; //
```

Додаємо 1 стовбець для номерів варіантів

```
for (int j = 0; j < tableData1.length; j++) {
```

```
tableData1[j][0] = String.valueOf(j + 1); // Вносимо номери варіантів
```

```
for (int i = 0; i < tableData1[0].length - 1; i++) {
```

```
tableData1[j][i + 1] = String.valueOf(alternatives[i][j]); // Вносимо значення по критеріям
```

```
    }
```

```
}
```

```
String[][] tableData2 = new String[alternatives[0].length][alternatives.length + 1]; //
```

Додаємо 1 стовбець для номерів варіантів

```
for (int j = 0; j < tableData2.length; j++) {
```

```
tableData2[j][0] = String.valueOf(j + 1); // Вносимо номери варіантів
```

```
for (int i = 0; i < tableData2[0].length - 1; i++) {
```

```
tableData2[j][i + 1] = String.valueOf(normalizedAlternatives[i][j]); // Вносимо значення по критеріям
```

```
    }
```

```
}
```

```
String[][] tableDataPairComp = new
```

```
String[pairCompAlternatives.length][criteriaNames.length + 1];
```

```
for (int i = 0; i < tableDataPairComp.length; i++) {
```

```
tableDataPairComp[i][0] = String.valueOf(pairCompAlternatives[i].index);
```

```

for (int j = 0; j < tableDataPairComp[j].length - 1; j++) {
    tableDataPairComp[i][j+1] =
    String.valueOf(pairCompAlternatives[i].criteriaValues[j]);
        }
    }

String[][] tableDataResult = new String[result.length][criteriaNames.length + 1];
for (int i = 0; i < tableDataResult.length; i++) {
    tableDataResult[i][0] = String.valueOf(result[i].index);

    for (int j = 0; j < tableDataResult[0].length - 1; j++) {
        tableDataResult[i][j+1] = String.valueOf(result[i].criteriaValues[j]);
            }
        }

    new MyFrame(tableData1,tableData2, criteriaNames,
    bestAndWorstString,tableDataPairComp,tableDataResult);
}
};

```

ДОДАТОК Б

Демонстраційні матеріали

Слайд 1

ТИТУЛЬНИЙ ЛИСТ РОБОТИ

Міністерство освіти і науки України Харківський національний університет радіоелектроніки	
Кафедра КІТАМ	
Магістерська атестаційна робота на тему: «Методи підтримки прийняття рішень у системах проектування <u>роботизованих</u> ТП»	
Студент: гр. КТРСм-19-1 <u>Коритченко</u> В.К.	Керівник: проф. <u>Безкоровайний</u> В.В

Слайд 2

ВСТУП

ВСТУП

Мета роботи – підвищення ефективності систем підтримки прийняття рішень в САПР за рахунок розробки методів виділення та скорочення підмножин ефективних варіантів побудови ТП.

Об'єкт дослідження – роботизовані технологічні процеси.

Предмет дослідження – методи багатокритеріального вибору у системах проектування роботизованих технологічних процесів.

Для досягнення мети необхідно виконати наступні завдання:

- провести огляд та аналіз сучасного стану проблеми підтримки прийняття рішень у системах проектування роботизованих ТП;
- проаналізувати особливості технологічних процесів виготовлення МЕМС-акселерометрів;
- проаналізувати математичне забезпечення системи підтримки прийняття проектних рішень;
- розробити діаграму прецедентів та діаграму класів для системи проектування роботизованого технологічного процесу;
- розробити програмне забезпечення для підвищення ефективності підтримки прийняття рішень з багатокритеріального вибору у системах проектування роботизованих технологічних процесів;
- провести аналіз, надати рекомендації щодо використання результатів роботи.

Слайд 3

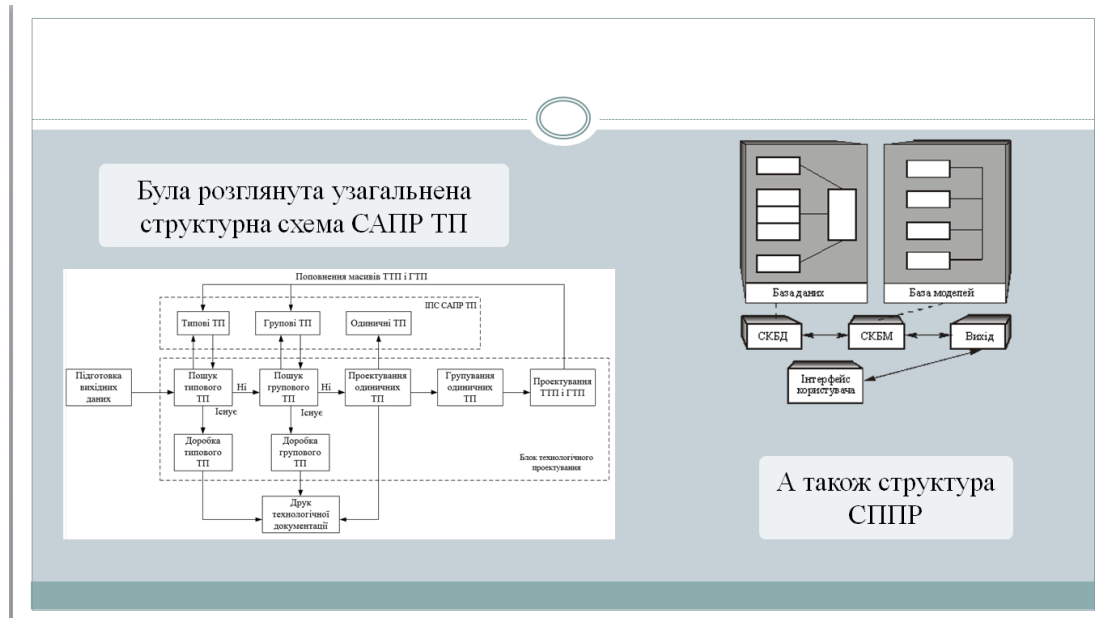
ПРИКЛАДИ СУЧАСНИХ ВИРОБНИЧИХ РОБОТІВ



Приклади сучасних
виробничих роботів

Слайд 4

СТРУКТУРИ САПР ТП І СППР



Слайд 5

МЕМС-АКСЕЛЕРОМЕТРИ

МЕМС-АКСЕЛЕРОМЕТРИ

МЕМС-акселерометри – це датчики, засновані на мікроелектромеханічних системах (МЕМС). Популярність даних пристроїв обумовлена низкою причин, основними з яких є простота їх використання, відносно низька ціна і малі габарити. Акселерометр є приладом для вимірювання сили реакції індукованої прискоренням або гравітацією. Його різні моделі можуть визначати величину та напрям прискорення у вигляді векторної величини і тому можуть бути використані для визначення таких характеристик як орієнтація, вібрація, удари.

Вони знайшли широке застосування при випробуваннях та експлуатації кораблів, літаків, ракет, автомобілів тощо, а також як чутливі елементи автопілотів тощо.

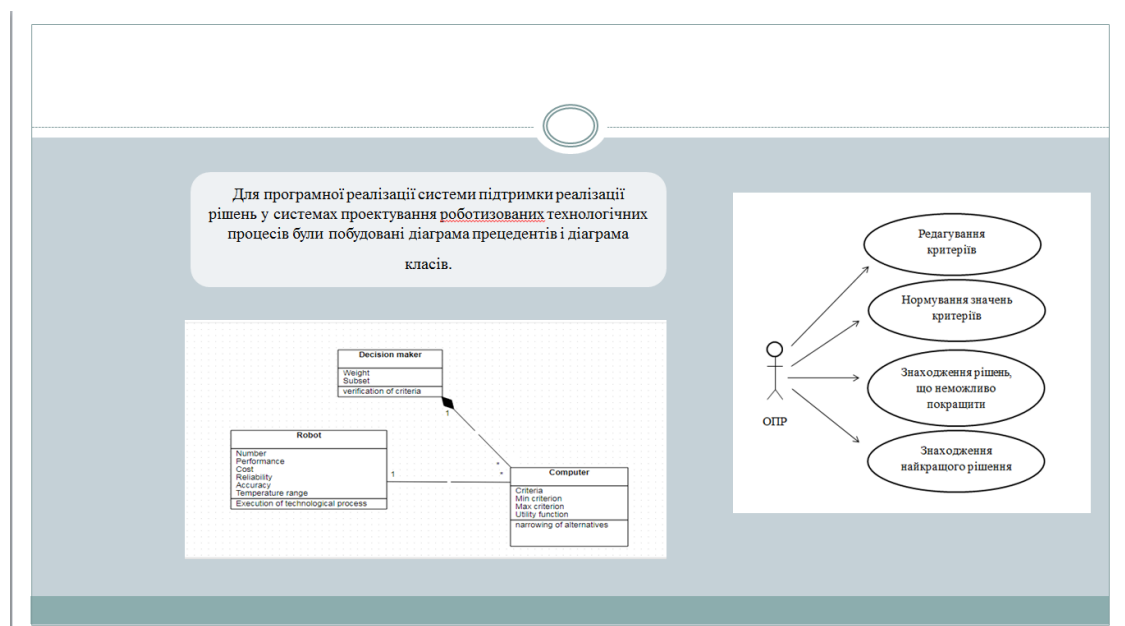
Слайд 6

СХЕМА ЕТАПІВ ТП ВИГОТОВЛЕННЯ МЕМС-АКСЕЛЕРОМЕТРІВ



Слайд 7

ДІАГРАМА ПРЕЦЕДЕНТІВ І ДІАГРАМА КЛАСІВ



Слайд 8

РЕЗУЛЬТАТИ ПЕРШОГО ЕТАПУ РОБОТИ ПРОГРАМИ

В результаті роботи програми спочатку були нормовані значення критеріїв.

№ Альтернативи	Продуктивність	Ціна	Надійність	Точність
1	0.012571048	0.08448151	0.8248805	0.8573882
2	0.5260777	0.584643	0.0010770059	0.5757517
3	0.81878556	0.49195802	0.7350565	0.8196263
4	0.9886003	0.21213022	0.98384494	0.87193216
5	0.28259712	0.7503464	0.811524	0.5721295
6	0.019957425	0.5267615	0.5767367	0.99847605
7	0.8667376	0.9195509	0.6742057	0.8925946
8	0.006552475	0.023764731	0.8621432	0.5461395
9	0.40792975	0.32882756	0.13731825	0.9500168
10	0.54630125	0.118897688	0.2947119	0.59995167
11	0.8281	0.48088482	0.7641357	0.8798861
12	0.567323	0.08620395	0.28917608	0.8241063
13	0.27913785	0.7858787	0.5627356	0.9210021
14	0.2112826	0.73071575	0.14054927	0.05728352
15	0.9026078	0.6335815	0.1448573	0.06451529
16	0.58408725	0.7862767	0.80937	0.77914137
17	0.7679617	0.10268738	0.5659666	0.5150486
18	0.256645182	0.65138847	0.80883145	0.05277443
19	0.2615753	0.7152355	0.46365106	0.8885364
20	0.26796168	0.22858047	0.23209478	0.15038478
21	0.001330495	0.6775399	0.15724286	0.74772525
22	1.0	0.3437922	0.35595044	0.041185975
23	0.0260777	0.44212613	0.21593969	0.34189713
24	0.55880785	0.8421751	0.728056	0.6665266
25	0.25385845	0.07349198	0.5088853	0.6690516
26	0.7142097	0.9291538	0.40603122	0.05666715
27	0.822091	0.128476	0.99240097	0.9382605
28	0.09499734	0.65378284	0.15293483	0.7417876
29	0.5117084	0.21679766	0.9951535	0.42851973
30	0.05612987	0.8149261	0.8476037	0.9655046
31	0.12320383	0.9470871	0.5358105	0.7271203
32	0.08949296	0.55100864	0.7075929	0.70168126
33	0.58461846	0.26136482	0.80937	0.95404315
34	0.46035126	0.48980108	0.37479806	0.89565146
35	0.4669698	0.49588925	0.3435649	0.9182243
36	0.45267684	0.56579846	0.55465806	0.6078202
37	0.6852049	0.92559034	0.25201938	0.099618495
38	0.09127795	0.356394	0.11847054	0.07744151
39	0.74667376	0.47726396	0.59505575	0.58519596
40	0.18939808	0.81263	0.3446419	0.98587286
41	0.1184625	0.32428393	0.3268713	0.6531292
42	0.16844065	0.22915736	0.4884222	0.43672174
43	0.72911125	0.82500374	0.7097469	0.11479926
44	0.555825	0.26136803	0.115239635	0.841392
45	0.7969665	0.6582129	0.23961381	0.80746615
46	0.4201703	0.5005769	0.29779214	0.20418483
47	0.1344609	0.7282129	0.6041405	0.3681295
48	0.32464078	0.9490207	0.11739364	0.44451302
49	0.53938264	0.3692272	0.06031233	0.05863756
50	0.30464847	0.17618695	0.6596661	0.60893805

Поточний етап нормалізації значень < Назад Наступний етап >

Слайд 9

РЕЗУЛЬТАТИ ДРУГОГО ЕТАПУ РОБОТИ ПРОГРАМИ

Потім були знайдені ті рішення, що неможливо покращити.

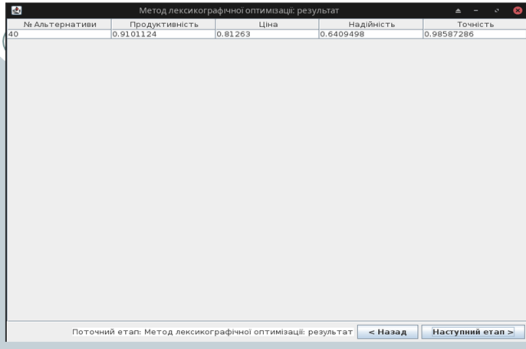
№ Альтернативи	Продуктивність	Ціна	Надійність	Точність
2	0.7752809	0.584643	0.9984447	0.27575517
6	0.74157364	0.5267615	0.6131142	0.99847605
7	0.33707866	0.9195509	0.14797783	0.89825046
8	0.9550562	0.023764731	0.86333925	0.5461395
9	0.60674155	0.32882756	0.6583708	0.9500168
14	1.0	0.73071575	0.960625	0.05728352
16	0.96629214	0.7982767	0.079841554	0.27914137
24	0.37078652	0.8421751	0.26875848	0.6665269
25	0.20274719	0.07349198	0.98466307	0.6690516
26	0.35855057	0.9291538	0.88653344	0.05666715
28	0.94382024	0.65378284	0.35237342	0.7417876
30	0.7303371	0.8149261	0.24348813	0.9655046
32	0.9101124	0.55100864	0.9569071	0.70168126
33	0.04494382	0.26136482	0.8371322	0.95404315
37	0.58426964	0.92559034	0.85480785	0.099618495
38	0.9550562	0.3563944	0.90927017	0.07744151
40	0.9101124	0.81263	0.6409498	0.98587286
42	0.9775281	0.22915736	0.019702792	0.43672174
43	0.48914807	0.82500374	0.35708767	0.11479926
47	0.2247191	0.7282129	0.8567415	0.3681295
48	0.02247191	0.9490207	0.83857983	0.44451302
51	0.85392256	1.0	0.12678069	0.2340697
52	0.98876405	0.43811532	0.5405261	0.19573629
53	0.3258427	0.95714945	0.60518414	0.8055294
55	0.29213482	0.9969688	0.3760181	0.05250287
56	0.9550562	0.98588663	0.3101108	0.06930286
57	0.08988764	0.920064	0.8931008	0.56735796
58	0.60674155	0.20955464	0.9629379	0.9126773
61	0.20224719	0.8790947	0.8280483	0.91461957
65	0.3037077	0.8346156	0.65356046	0.21363878
67	0.96629214	0.20917176	0.85713065	0.55887663
70	0.9775281	0.3107194	0.21813792	0.68232274
71	0.9101124	0.6936162	0.18256587	0.88107187
74	0.37078652	0.60904336	0.9004695	0.9266742
76	0.6745713	0.28383538	0.9829797	0.7135216
79	0.29213482	0.9618526	0.52616006	0.7935169
89	0.94382024	0.35593367	0.10881293	0.2963481
91	0.9775281	0.3192706	0.8133464	0.43990725
100	0.9101124	0.0037089319	0.92579645	0.5788513

Поточний етап: Метод парних порівнянь < Назад Наступний етап >

Слайд 10

РЕЗУЛЬТАТИ ТРЕТЬОГО ЕТАПУ РОБОТИ ПРОГРАМИ

Останнім етапом були визначені ваги критеріїв та знайдене найкраще рішення системи.



№ Альтернативи	Продуктивність	Ціна	Надійність	Точність
40	0.9101124	0.81263	0.6409498	0.98587288

Поточний етап: Метод лексикографічної оптимізації: результат < Назад Наступний етап >

Слайд 11

ВИСНОВКИ

ВИСНОВКИ

В результаті атестаційної роботи була спроектована і розроблена система підтримки прийняття рішень з багатокритеріального вибору у роботизованому технологічному процесі.

Розроблений засіб дозволяє підвищити ефективності підтримки прийняття рішень з багатокритеріального вибору у системах проектування роботизованих технологічних процесів шляхом приведення до нормованого вигляду за допомогою функції корисності, знайдена множина рішень, яку неможливо покращити методом парних порівнянь, визначення ваги критеріїв ОПР, звуження підмножини ефективних проектних рішень методом лексикографічної оптимізації.

Прикладом такого роботизованого технологічного процесу стало виготовлення MEMS акселерометрів.

ДОДАТОК В

Відомість атестаційної роботи

[illegible]