MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII
CENTRUL DE EXCELENȚĂ ÎN INFORMATICĂ SI TEHNOLOGII INFORMAȚIONALE

RAPORT

Privind practica de instruire
la specialitatea Programarea și analiza produselor program

Tema: Depozit.

A elaborat elevul: Cemîrtan Cristian, P-1822

A evaluat profesorul: Frunza Olga

CHIȘINĂU 2020

Cuprins

Introducere

O deosebită importanță în pregătirea specialiștilor de domeniu, s-a dovedit a fi pregătirea practică a studenților, care vine să realizeze obiective și competențe profesionale. Practica îmbracă în sine următoarele etape de instruire: practica instructivă, practica tehnologică urmând și practica de producție. Etapa tehnologică asigură conexiunea instruirii teoretice cu activitatea de producție. În al doilea an de studii **CEITI** vine în sprijinul studenților cu un proces de studii privind familiarizarea cu elemente de bază ale teoriei și practicii în informatică, bazelor de date, tehnologiilor de comunicare, programării orientate pe obiecte și a calculatoarelor.

Curriculumul la disciplina vizată reprezintă etapa tehnologică, cu toate elementele teoretice și practice intermediare din domeniul informaticii, care au fost studiate.

Ca un început, spre elucidarea cunoștințelor acumulate la etapa tehnologică, prezint anumite obiective și competențe:

Obiective:

- Însușirea exhaustivă a cunoștințelor practice obținute de elevi în programarea produselor program;
- Implementarea tehnologiilor eficiente de elaborare a produselor program;
- Utilizarea metodelor eficiente de comunicare, cu utilizatorul.

Competențe:

- Analiza preliminară de studiu;
- Proiectarea, implementarea și testarea subprogramelor;
- Capacitatea de a selecta conștiincios algoritme privind prelucrarea datelor;
- Elaborarea tehnicilor de testare și depanare a programelor;

Conținutul activităților și sarcinilor de lucru

Problema în Java

9. Depozit. Pentru păstrarea în siguranță a unor cantități mari de substanțe radioactive, într-o o regiune îndepărtată a fost construit un depozit subteran, format dintr-o rețea de tuneluri la intersecția cărora se află zone de depozitare. O zonă de depozitare poate fi liberă sau ocupată de containere cu substanțe radioactive. Tunelurile depozitului formează un caroiaj, iar distanta dintre oricare două zone vecine este egală cu 1 (vezi fig.1). Tunelurile orizontale sunt numerotate prin 1, 2, 3, ..., **n**, iar cele verticale prin 1, 2, 3, ..., **m**. Fiecare zonă de depozitare se identifică printr-o adresă unică de forma **[i, j]**, unde i este numărul tunelului orizontal, iar j al celui vertical, la intersecția cărora se află zona respectivă.

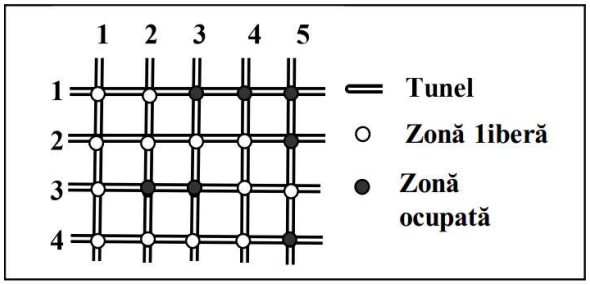

Figura 1. Harta depozitului

Informații despre depozitul în studiu sunt înregistrate în fișierul text **Depozit.in**, care conține pe prima linie numerele naturale **n** și **m**, separate prin spațiu. Linia a doua a fișierului de intrare conține naturalul **k** – numărul zonelor ocupate de containerele cu substanțe radioactive. Următoarele linii ale acestui fișier conțin câte două numere naturale **i**, **j** - adresele zonelor ocupate.

Să se elaboreze un program care, folosind meniuri și subprograme, să realizeze, la solicitarea utilizatorului, următoarele prescripții:

1) Interschimbă în planul depozitului prima cu ultima linie;

2) Reformează, prin substituiri în fișierul de intrare, un tunel din planul caroiajului, orientarea (orizontal/vertical) și numărul de ordine al cărui se vor indica de la tastatură;

3) Creează fișierul text **Binar.txt**, fiecare din cele n linii ale cărui va conține câte m cifre binare, separate prin spațiu – elementele unei matrice **T** de dimensiunea **n × m**, în care **T[i, j] = 1**, dacă zona de depozitare este ocupată, şi **T[i, j] = 0** în caz contrar;

4) Determină numărul tunelului vertical cu un număr minimal de zone de depozitare ocupate;

5) Afișează pe ecran lista tunelurilor verticale în ordinea descendentă a numerelor zonelor de depozitare ocupate din tunelurile respective; sortarea datelor se va realiza prin metoda selecției.

6) Determină adresele zonelor de depozitare libere, care nu au zone vecine ocupate în cele patru direcții cardinale (nord, est, sud, vest);

7) Determină în caroiajul depozitului un dreptunghi de arie maximală, ce conține pe perimetru doar zone libere; la ecran se va afișa aria **S** și coordonatele colțurilor stânga – sus și dreapta – le dreptunghiului găsit;

8) Rezolvă problema: Prin coridoare și prin zonele de depozitare se deplasează un robot. Robotul poate trece printr-o zonă de depozitare numai atunci când ea este liberă.

- Elaborați un subprogram care, determină cel mai scurt drum ce trebuie parcurs de robot pentru a ajunge dintr-o zona **[a, b]** în altă zonă **[c, d]**. Distanțele parcurse de robot în interiorul zonelor de depozitare nu vor fi luate în considerație.
- **Date de intrare**. Numerele naturale **a**, **b**, **c** și **d** se vor citi de la tastatură, iar adresele zonelor ocupate - din fișierul Depozit.in, descris anterior.
- **Date de ieșire**. Fișierul **Depozit.out** va conține pe prima linie naturalul **L** – lungime celui mai scurt drum. Fiecare din următoarele **L + 1** linii ale fișierului de ieșire va conține câte două numere naturale separate prin spațiu, reprezentând adresele zonelor ce formează unul dintre cele mai scurte drumuri.
- **Exemplu** (ilustrat prin figura 1, pentru **a** = 2; **b** = 2; **c** = 3; **d** = 5).
- **Restricții**. $1 \leq \mathbf{m}, \mathbf{n} \leq 100$; $1 \leq \mathbf{a}, \mathbf{c} \leq n$; $1 \leq \mathbf{b}, \mathbf{d} \leq \mathbf{m}$; $1 \leq \mathbf{k} \leq 1000$.

Depozit.in	Depozit.out
4 5 7 1 3 1 4 1 5 2 5 3 2 3 3 4 5	4 2 2 2 3 2 4 3 4 3 5

Figura 2. Datele de intrare și de ieșire

Descrierea modului de elaborare a aplicației

Aplicația se împarte în patru clase, situate în pachetul md.ceiti:

- Depozit
  - Include algoritme specifice pentru sarcinile date;
  - Conține un constructor care memorează datele din fișierul text de intrare.
- *MenuFrame*
  - Scheletul meniului grafic principal.
- *ArgInputDialog*
  - Scheletul meniului grafic, care culege datele de intrare, de la tastatură.
- **DepozitTestIO**
  - Clasa principală, care prelucrează datele și meniurile grafice, pentru interacțiunea cu utilizatorul.

Aplicația cuprinde în sine mecanisme de filtrare, a datelor de intrare. Dacă se introduc datele de intrare invalide, o sarcină, de completat, se va anula și va fi afișat pe ecran un mesaj de avertizare despre acest caz. Datele de intrare textuale, cerute în sarcina 2, sunt în caz insensibile, înseamnă că o literă majusculă și minusculă din text reprezintă aceeași valoare.

Primele șapte sarcini se rezolvă folosind algoritmele de bază (programarea dinamică, trierii). Dar sarcina 8 se rezolvă sub metoda automatizării celulare, un algoritm asemănător cu a lui Lee, care folosește direcții cardinale în loc de numere. Comportamentul acestui algoritm se manifestă în așa fel:

1. Se dă o hartă de rezolvat.
2. Algoritmul „inundă” harta și se oprește când a atins punctul de sfârșit.
3. Calea, cea mai scurtă, este determinată într-un timp constant, dar evidențierea – linear. Evidențierea se trasează de la punctul de sfârșit până la cel de început, urmărind orientările pe orice punct. Rezultatele se stochează într-o coadă cu ambele capete.

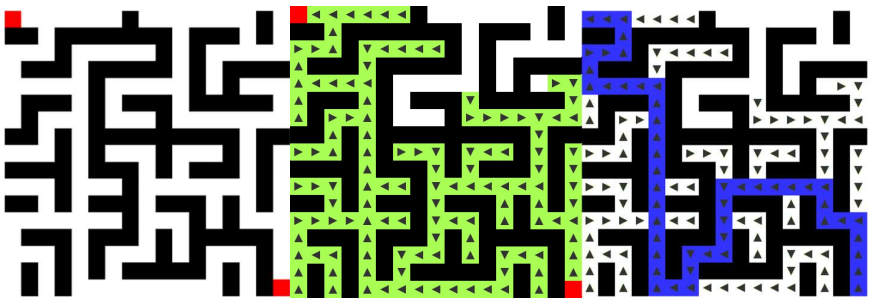

Figura 3. Un labirint rezolvat utilizând automatizarea celulară

Listing-ul programului

Depozit.java

```
1.    package md.ceiti;
2.    import java.io.*;
3.
4.    public class Depozit
5.    {
6.        static final String[] FILENAMES = {
7.            "Depozit.in",
8.            "Depozit.out",
9.            "Binar.txt"
10.       };
11.
12.       private static final String NO_ANS = "Nu exista nici o solutie.";
13.       private boolean[][] labirint;
14.       private int x, y;
15.
16.       public Depozit() throws IOException
17.       {
18.           try(java.util.Scanner fileIn = new java.util.Scanner(new File(FILE
NAMES[0])))
19.           {
20.               x = fileIn.nextInt();
21.               y = fileIn.nextInt();
22.               labirint = new boolean[x][y];
23.
24.               fileIn.nextInt(); // ignor
25.
26.               while(fileIn.hasNext())
27.                   try
28.                   {
29.                       labirint[fileIn.nextInt() - 1][fileIn.nextInt() - 1] =
 true;
30.                   }
31.                   catch(ArrayIndexOutOfBoundsException e) {}
32.           }
33.       }
34.
35.       public static void sarcina_1() throws IOException
36.       {
37.           java.util.List<String> buffer = new java.util.ArrayList<>();
38.
39.           try(java.util.Scanner file = new java.util.Scanner(new File(FILENA
MES[0])))
40.           {
41.               while(file.hasNext())
42.                   buffer.add(file.nextLine());
43.
44.               java.util.Collections.swap(buffer, 2, buffer.size() - 1);
45.           }
46.
47.           try(PrintStream file = new PrintStream(FILENAMES[0]))
48.           {
49.               for(String x : buffer)
50.                   file.println(x);
51.           }
52.       }
53.
```

```
54.       public String sarcina_2(int nrOrdine, String orientare) throws IOExcep
tion
55.       {
56.           java.util.function.BiFunction<Integer, Integer, Boolean> getValues
 = (i, b) ->
57.               (boolean) ArgInputDialog.getValues(2, String.format("%d/%d", i
, b));
58.
59.           java.util.Map<String, Runnable> orientari = new java.util.HashMap<
>();
60.           StringBuilder msg = new StringBuilder();
61.
62.           orientari.put("orizontal", () -> {
63.               for(int i = 0; i < y; ++i)
64.               {
65.                   labirint[nrOrdine][i] = getValues.apply(i + 1, y);
66.                   msg.append(String.format("\u2550%c", labirint[nrOrdine][i]
 ? '\u25cf' : '\u25cb'));
67.               }
68.
69.               msg.append('\u2550');
70.           });
71.
72.           orientari.put("vertical", () -> {
73.               for(int i = 0; i < x; ++i)
74.               {
75.                   labirint[i][nrOrdine] = getValues.apply(i + 1, x);
76.                   msg.append(String.format("\u2551\n%c\n", labirint[i][nrOrd
ine] ? '\u25cf' : '\u25cb'));
77.               }
78.
79.               msg.append('\u2551');
80.           });
81.
82.           try
83.           {
84.               orientari.get(orientare.toLowerCase()).run();
85.           }
86.           catch(NullPointerException | ArrayIndexOutOfBoundsException e)
87.           {
88.               throw new IllegalArgumentException();
89.           }
90.
91.           try(PrintStream fileOut = new PrintStream(FILENAMES[0]))
92.           {
93.               StringBuilder buffer = new StringBuilder();
94.               int c = 0;
95.
96.               for(int j, i = 0; i < x; ++i)
97.                   for(j = 0; j < y; ++j)
98.                       if(labirint[i][j])
99.                       {
100.                          buffer.append(String.format("%d %d%n", i + 1, j +
1));
101.                          ++c;
102.                      }
103.
104.              fileOut.printf("%d %d%n%d%n%s", x, y, c, buffer);
105.          }
106.
107.          return msg.toString();
108.      }
109.
110.      public void sarcina_3() throws IOException
```

```
111.        {
112.            try(PrintStream fileOut = new PrintStream(FILENAMES[2]))
113.            {
114.                for(int j, i = 0; i < x; ++i)
115.                {
116.                    for(j = 0; j < y; ++j)
117.                        fileOut.printf("%c ", labirint[i][j] ? '1' : '0');
118.
119.                    fileOut.println();
120.                }
121.            }
122.        }
123.
124.        // sarcina 4 este deja implementata in DepozitTestIO
125.
126.        public int[][] sarcina_5()
127.        {
128.            int[][] data = new int[y][2];
129.            int[] swp;
130.
131.            int mn, j, i, n = y;
132.
133.            for(i = 0; i < y; ++i)
134.            {
135.                data[i][1] = i + 1;
136.
137.                for(j = 0; j < x; ++j)
138.                    if(labirint[j][i])
139.                        ++data[i][0];
140.            }
141.
142.            for(i = 0; i < y; ++i)
143.            {
144.                mn = 0;
145.
146.                for(j = 1; j < n; ++j)
147.                    if(data[j][0] < data[mn][0])
148.                        mn = j;
149.
150.                swp = data[--n];
151.                data[n] = data[mn];
152.                data[mn] = swp;
153.            }
154.
155.            return data;
156.        }
157.
158.        public java.util.List<int[]> sarcina_6()
159.        {
160.            int
161.                j, i,
162.                x = this.x - 2,
163.                y = this.y - 2;
164.
165.            java.util.List<int[]> adrese = new java.util.ArrayList<>();
166.
167.            for(i = 1; i <= x; ++i)
168.                for(j = 1; j <= y; ++j)
169.                    if( !labirint[i][j] &&
170.                        !labirint[i + 1][j] && !labirint[i][j + 1] &&
171.                        !labirint[i - 1][j] && !labirint[i][j - 1])
172.                        adrese.add(new int[]{i + 1, j + 1});
173.
174.            return adrese;
```

```
175.        }
176.
177.        private boolean isRectangle(int a, int b, int c, int d)
178.        // reduce de la O(n ^ 2) la O(n) datorita multithreading-
ului paralel
179.        {
180.            int j = 0;
181.            boolean[] results = {true, true};
182.
183.            Thread[] futures = {
184.                new Thread(() -> {  // verific vertical
185.                    for(int i = b; i <= d && results[1]; ++i)
186.                        if(labirint[a][i] || labirint[c][i])
187.                        {
188.                            results[0] = false;
189.                            return;
190.                        }
191.                })
192.                ,
193.                new Thread(() -> {  // verific orizontal
194.                    for(int i = a + 1; i < c && results[0]; ++i)
195.                        if(labirint[i][b] || labirint[i][d])
196.                        {
197.                            results[1] = false;
198.                            return;
199.                        }
200.                })
201.            };
202.
203.            try
204.            {
205.                for(j = 0; j < futures.length; ++j)
206.                    futures[j].start();
207.
208.                for(j = 0; j < futures.length; ++j)
209.                    futures[j].join();
210.
211.                return results[0] && results[1];
212.            }
213.            catch(InterruptedException e)
214.            {
215.                return false;
216.            }
217.
218.        }
219.
220.        public int[][] sarcina_7()
221.        {
222.            int[][] mx = new int[][]{
223.                {1},
224.                {0, 0},
225.                {0, 0}
226.            };
227.
228.            int area;
229.
230.            for(int l, k, j, i = 0; i < x; ++i) // brute-force, O(n ^ 5) !!!
231.                for(j = 0; j < y; ++j)
232.                    for(k = i; k < x; ++k)
233.                        for(l = j; l < y; ++l)
234.                            if(isRectangle(i, j, k, l))
235.                            {
236.                                area = (l - j + 1) * (k - i + 1);
237.
```

```
                                if(area > mx[0][0])
                                {
                                    mx[0][0] = area;

                                    mx[1][0] = i + 1;
                                    mx[1][1] = j + 1;

                                    mx[2][0] = k + 1;
                                    mx[2][1] = l + 1;
                                }
                            }

            return mx;
        }

        public String[] sarcina_8(int a, int b, int c, int d) throws IOExcepti
on
        {
            class Lee
            {
                int[][] labirintCloned = new int[x][y];

                int[][] moves = {
                    {0, 1},
                    {1, 0},
                    {0, -1},
                    {-1, 0}
                };

                java.util.Queue<int[]> ways = new java.util.LinkedList<>();

                Lee(int $x, int $y)
                {
                    for(int j, i = 0; i < x; ++i)
                        for(j = 0; j < y; ++j)
                            labirintCloned[i][j] = labirint[i][j] ? -2 : -
1; // ? ocupat : liber

                    if(!check(c, d) || !check(a, b) || !go($x, $y))
                        throw new IllegalArgumentException(NO_ANS);
                }

                boolean check(int x, int y)
                {
                    try
                    {
                        return labirintCloned[x][y] == -1;
                    }
                    catch(ArrayIndexOutOfBoundsException e)
                    {
                        return false;
                    }
                }

                boolean go(int x, int y)
                {
                    ways.add(new int[]{x, y});

                    int[] pos;

                    for(int xp, yp, i; !ways.isEmpty();)
                    {
                        pos = ways.element();
                        ways.remove();
```

```
300.
301.                         if(pos[0] == c && pos[1] == d)
302.                             return true;
303.
304.                         for(i = 0; i < moves.length; ++i)
305.                         {
306.                             xp = pos[0] + moves[i][0];
307.                             yp = pos[1] + moves[i][1];
308.
309.                             if(check(xp, yp))
310.                             {
311.                                 ways.add(new int[]{xp, yp});
312.                                 labirintCloned[xp][yp] = (i + 2) & 3;
313.                             }
314.                         }
315.                     }
316.
317.                     return false;
318.                 }
319.
320.                 java.util.Collection<int[]> parse()
321.                 {
322.                     java.util.Deque<int[]> mnWays = new java.util.ArrayDeque<>
();
323.                     mnWays.push(new int[]{c, d});
324.
325.                     for(int dir, x = c, y = d; x != a || y != b;)
326.                     {
327.                         dir = labirintCloned[x][y];
328.                         x += moves[dir][0];
329.                         y += moves[dir][1];
330.
331.                         mnWays.addFirst(new int[]{x, y});
332.                     }
333.
334.                     return mnWays;
335.                 }
336.             }
337.
338.             java.util.Collection<int[]> mnWays;
339.
340.             try
341.             {
342.                 mnWays = new Lee(a, b).parse();
343.             }
344.             catch(ArrayIndexOutOfBoundsException e)
345.             {
346.                 throw new IllegalArgumentException(NO_ANS);
347.             }
348.
349.             char[][] printedLabirint = new char[x][y];
350.             int j, i;
351.
352.             for(i = 0; i < x; ++i)
353.                 for(j = 0; j < y; ++j)
354.                     printedLabirint[i][j] = labirint[i][j] ? '\u25cf' : '\u25c
b';
355.
356.             StringBuilder steps = new StringBuilder(mnWays.size() - 1 + System
.lineSeparator());
357.
358.             for(int[] x : mnWays)
359.             {
360.                 printedLabirint[x[0]][x[1]] = '\u25a0';
```

```
361.                steps.append(String.format("%d %d%n", x[0] + 1, x[1] + 1));
362.            }
363.
364.            steps.deleteCharAt(steps.length() - 1);
365.
366.            try(PrintStream fileOut = new PrintStream(FILENAMES[1]))
367.            {
368.                fileOut.print(steps);
369.            }
370.            finally
371.            {
372.                StringBuilder printedWays = new StringBuilder();
373.                boolean test;
374.
375.                for(i = 0;; ++i)
376.                {
377.                    for(j = 0; j < y; ++j)
378.                        printedWays.append(" \u2551");
379.
380.                    test = i >= x;
381.
382.                    if(test)
383.                        break;
384.
385.                    printedWays.append('\n');
386.
387.                    for(j = 0; j < y; ++j)
388.                        printedWays.append("\u2550" + printedLabirint[i][j]);

389.
390.                    printedWays.append('\u2550');
391.
392.                    if(!test)
393.                        printedWays.append('\n');
394.                }
395.
396.                return new String[]{ printedWays.toString(), steps.toString()
};
397.            }
398.        }
399.    }
```

MenuFrame.java

```
1.    package md.ceiti;
2.    import java.awt.*;
3.    import java.awt.event.*;
4.    import javax.swing.*;
5.
6.    public class MenuFrame extends JFrame implements ActionListener
7.    {
8.        private static final String PRINT =
9.            "Programul Depozit \u00a9 Cem\u00eertan Cristian, 2020\n" +
10.           "Alege\u021bi sarcina de mai jos:\n" +
11.           "\t1. Interschimbarea, \u00een planul depozitului, primei cu ultim
a linie;\n" +
12.           "\t2. Reformarea tunelului, dup\u0103 num\u0103rul de ordine \u021
9i orientarea introduse de la tastatur\u0103;\n" +
13.           "\t3. Creearea fi\u0219ierului, unde se va \u00eenscrie versiunea
binar\u0103 a labirintului;\n" +
```

```
            "\t4. Determinarea num\u0103rului a unui tunel vertical, cu num\u0
103rul minimal de depozite ocupate;\n" +
            "\t5. Afisarea pe ecran lista tunelurilor verticale, \u00een ordin
ea descendent\u0103 dup\u0103 num\u0103rul lor de zone ocupate;\n" +
            "\t6. Determinarea adreselor zonelor de depozitare libere, care nu
 au zone vecine ocupate;\n" +
            "\t7. Determinarea ariei maxime \u00een harta depozitului, inclusi
v \u0219i coordonatele col\u021burilor st\u00e2nga-sus \u0219i dreapta-
jos;\n" +
            "\t8. Determinarea celui mai scurt drum de parcurs, de la zona [a,
 b] \u00een zona [c, d].";

        public MenuFrame()
        {
            super("Depozit");

            setLocationRelativeTo(null);
            setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

            setUndecorated(true);
            getRootPane().setWindowDecorationStyle(JRootPane.PLAIN_DIALOG);

            JPanel
                menuText = new JPanel(),
                buttonsArea = new JPanel();

            JTextArea text = new JTextArea(PRINT);
                text.setFont(new Font("Times New Roman", Font.PLAIN, 16));
                text.setEditable(false);

            menuText.setBackground(Color.WHITE);
            menuText.add(text);

            add(menuText, BorderLayout.PAGE_START);

            JButton[] buttons = new JButton[8]; // sunt 8 sarcini

            for(int i = 0; i < buttons.length; ++i)
            {
                buttons[i] = new JButton(i + 1 + "");
                buttons[i].addActionListener(this);

                buttonsArea.add(buttons[i]);
            }

            add(buttonsArea, BorderLayout.PAGE_END);
            pack();

            setResizable(false);
            setVisible(true);
        }

        @Override
        public void actionPerformed(ActionEvent e)
        {
            try
            {
                DepozitTestIO.menu(
   Integer.parseInt(
   ((JButton) e.getSource()).getText()
   ),
   this
   );
            }
```

```
72.             catch(java.io.IOException ex)
73.             {
74.                 JOptionPane.showMessageDialog(this,
75.                     "Eroare de fi\u0219ier.",
76.                     "Eroare",
77.                     JOptionPane.ERROR_MESSAGE
78.                 );
79.             }
80.         }
81.     }
```

ArgInputDialog.java

```
1.     package md.ceiti;
2.     import java.awt.*;
3.     import java.awt.event.*;
4.     import javax.swing.*;
5.     import javax.swing.border.*;
6.
7.     public class ArgInputDialog extends JDialog implements ActionListener
8.     {
9.         public static class NoAnswerException extends RuntimeException {} // e
xceptie verificata
10.
11.         private static final String[][] LABELS = {
12.             { "Numarul de ordine", "Orientarea" },
13.             { "a", "b", "c", "d" },
14.             { "Ocupat sau nu?" }
15.         };
16.
17.         private JTextField[] texts;
18.         private boolean[] flags = new boolean[1];
19.
20.         private ArgInputDialog(int status, String title)
21.         {
22.             super((JFrame) null, title == null ? "Introduceti datele" : title,
 true);
23.
24.             setLocationRelativeTo(null);
25.             setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
26.
27.             setUndecorated(true);
28.             getRootPane().setWindowDecorationStyle(JRootPane.QUESTION_DIALOG);

29.
30.             JPanel inputs = new JPanel();
31.                 inputs.setLayout(new BoxLayout(inputs, BoxLayout.PAGE_AXIS));

32.
33.             final int N = LABELS[status].length;
34.
35.             texts = new JTextField[N];
36.
37.             for(int i = 0; i < N; ++i)
38.             {
39.                 texts[i] = new JTextField();
40.                 texts[i].setBorder(
41.                     BorderFactory.createTitledBorder(
42.                         BorderFactory.createEtchedBorder(EtchedBorder.LOWERED)
,
```

```
43.                         LABELS[status][i]
44.                     )
45.                 );
46.
47.                 inputs.add(texts[i]);
48.             }
49.
50.             JButton button = new JButton("Finisat");
51.                 button.addActionListener(this);
52.
53.             add(inputs, BorderLayout.CENTER);
54.             add(button, BorderLayout.PAGE_END);
55.
56.             pack();
57.             setSize(140, getHeight());
58.
59.             setResizable(false);
60.             setVisible(true);
61.         }
62.
63.         public static Object getValues(int status, String title)
64.         {
65.             ArgInputDialog aid = new ArgInputDialog(status, title);
66.
67.             if(!aid.flags[0])
68.                 throw new NoAnswerException();
69.
70.             java.util.function.Supplier<?>[] lambdas = {
71.                 () -> {
72.                     return new Object[]{
73.                         Integer.parseInt(aid.texts[0].getText()),
74.                         aid.texts[1].getText()
75.                     };
76.                 }
77.                 ,
78.                 () -> {
79.                     int[] args = new int[aid.texts.length];
80.
81.                     for(int i = 0; i < aid.texts.length; ++i)
82.                         args[i] = Integer.parseInt(aid.texts[i].getText());
83.
84.                     return args;
85.                 }
86.                 ,
87.                 () -> {
88.                     java.util.Map<String, Boolean> map = new java.util.HashMap
<>();
89.                     map.put("ocupat", true);
90.                     map.put("yes", true);
91.                     map.put("da", true);
92.                     map.put("1", true);
93.
94.                     return map.getOrDefault(aid.texts[0].getText().toLowerCase
(), false);
95.                 }
96.             };
97.
98.             return lambdas[status].get();
99.         }
100.
101.        @Override
102.        public void actionPerformed(ActionEvent e)
103.        {
104.            flags[0] = true;
```

```
105.            dispose();
106.        }
107.    }
```

DepozitTestIO.java

```
1.      package md.ceiti;
2.      import java.awt.Font;
3.      import javax.swing.*;
4.
5.      public final class DepozitTestIO
6.      {
7.          public static void main(String[] args)
8.          {
9.              new MenuFrame();
10.         }
11.
12.         private static JPanel constructMessage(String in, int type, int size)

13.         {
14.             JPanel panel = new JPanel();
15.
16.             JTextArea text = new JTextArea(in);
17.                 text.setFont(new Font("Courier New", type, size));
18.                 text.setEditable(false);
19.
20.                 if(getRowsCount(in) > 25)
21.                     text.setRows(25);
22.
23.                 if(in.indexOf('\n') > 74)
24.                     text.setColumns(75);
25.
26.             panel.add(new JScrollPane(text));
27.             return panel;
28.         }
29.
30.         private static int getRowsCount(String in)
31.         {
32.             int i = 0;
33.             for(char x : in.toCharArray())
34.                 if(x == '\n')
35.                     ++i;
36.
37.             return i;
38.         }
39.
40.         static void menu(int choice, MenuFrame menuFrame) throws java.io.IOExc
eption
41.         {
42.             Object msg = "Sarcina a fost realizat\u0103 cu succes.";
43.             int state = JOptionPane.PLAIN_MESSAGE;
44.
45.             try
46.             {
47.                 Depozit test = new Depozit();
48.
49.                 switch(choice)
50.                 {
51.                     case 1:
52.                         Depozit.sarcina_1();
53.                         break;
54.
```

```
55.                     case 2:
56.                         try
57.                         {
58.                             Object[] args = (Object[]) ArgInputDialog.getValue
s(0, null);
59.                             msg = constructMessage(test.sarcina_2((int) args[0
] - 1, (String) args[1]), Font.PLAIN, 24);
60.                         }
61.                         catch(ArgInputDialog.NoAnswerException e)
62.                         {
63.                             throw new IllegalArgumentException();
64.                         }
65.                         break;
66.
67.                     case 3:
68.                         test.sarcina_3();
69.                         break;
70.
71.                     case 4:
72.                     {
73.                         int[][] sorted = test.sarcina_5();
74.                         int ans = sorted[sorted.length - 1][0];
75.
76.                         msg = String.format("Num\u0103rul tunelului este %d, c
u %d %s.",
77.                                             sorted[sorted.length - 1][1], ans
, ans == 1 ? "loc ocupat" : "locuri ocupate"
78.                         );
79.                         break;
80.                     }
81.                     case 5:
82.                     {
83.                         StringBuilder msgB = new StringBuilder("Tunel \u2192 l
ocuri ocupate\n");
84.
85.                         for(int[] x : test.sarcina_5())
86.                             msgB.append(String.format("%d \u2192 %d\n", x[1],
x[0]));
87.
88.                         msgB.deleteCharAt(msgB.length() - 1);
89.
90.                         msg = constructMessage(msgB.toString(), Font.BOLD, 16)
;
91.                         break;
92.                     }
93.                     case 6:
94.                     {
95.                         StringBuilder msgB = new StringBuilder();
96.
97.                         for(int[] x : test.sarcina_6())
98.                             msgB.append(String.format("%d %d\n", x[0], x[1]));

99.
100.                        msgB.deleteCharAt(msgB.length() - 1);
101.
102.                        msg = constructMessage(msgB.toString(), Font.BOLD, 16)
;
103.                        break;
104.                    }
105.                    case 7:
106.                    {
107.                        int[][] S = test.sarcina_7();
108.                        StringBuilder msgB = new StringBuilder(String.format(
```

```
109.                             "Aria maximal\u0103: %d\n%s",
110.                             S[0][0],
111.                             "Coordonatele col\u021burilor:\n"
112.                         ));
113.
114.                         for(int i = 1; i < S.length; ++i)
115.                             msgB.append(String.format("%d %d\n", S[i][0], S[i]
[1]));
116.
117.                         msgB.deleteCharAt(msgB.length() - 1);
118.                         msg = msgB.toString();
119.                         break;
120.                     }
121.                     case 8:
122.                     {
123.                         int[] args = (int[]) ArgInputDialog.getValues(1, null)
;
124.                         String[] ans = test.sarcina_8(args[0] - 1, args[1] - 1
, args[2] - 1, args[3] - 1);
125.
126.                         JPanel panel = constructMessage(ans[0], Font.PLAIN, 24
);
127.                             panel.add(constructMessage(ans[1], Font.BOLD, 16))
;
128.
129.                         msg = panel;
130.                         break;
131.                     }
132.                 }
133.             }
134.             catch(StringIndexOutOfBoundsException e)
135.             {
136.                 msg = "Nu exist\u0103 nici una.";
137.                 state = JOptionPane.INFORMATION_MESSAGE;
138.             }
139.             catch(java.util.InputMismatchException e)
140.             {
141.                 msg = "Verifica\u021bi datele de intrare.";
142.                 state = JOptionPane.WARNING_MESSAGE;
143.             }
144.             catch(IllegalArgumentException e)
145.             {
146.                 msg = e.getMessage();
147.
148.                 if(e instanceof NumberFormatException || msg == null)
149.                     msg = "R\u0103spuns Invalid.";
150.
151.                 state = JOptionPane.WARNING_MESSAGE;
152.             }
153.             catch(ArgInputDialog.NoAnswerException e) { return; }
154.
155.             JOptionPane.showMessageDialog(menuFrame,
156.                 msg,
157.                 "Rezultatul sarcinii " + choice,
158.                 state
159.             );
160.         }
161.     }
```

Estimarea complexității algoritmilor aplicați

Estimarea complexității algoritmilor aplicați ne permite să măsurăm nivelul de eficiență a soluției, în termenii de timp și de stocare.

Complexitatea temporală reprezintă frecvența executării a unei sau mai multor formule dintr-un algoritm, în dependență după lungimea datelor de intrare.

Complexitatea de spațiu reprezintă cât de lungă va fi memoria de lucrare a algoritmului. Practic, complexitatea de spațiu a sarcinilor, începând de la **3**, este de $\boldsymbol{O(n^2)}$, deoarece harta bidimensională, a problemei date, participă ca o dată de intrare. Primele două sarcini au de $\boldsymbol{O(n)}$, pentru că se bazează numai pe fișierele text.

Mai jos sunt descrise complexitățile temporale ale algoritmilor aplicați, în clasa **Depozit**:

Constructorul clasei:

- Citește datele din fișierul Depozit.in.
  - $\boldsymbol{O(n)}$

Sarcina 1:

- Moștenește funcționalitatea din constructorul clasei, dar nu-și modifică datele sale de instanță.
  - $\boldsymbol{O(n)}$

Sarcina 2:

- Parcurge liniar într-un tunel. Orientarea depinde după decizia utilizatorului.
  - $\boldsymbol{O(x}$ **dacă** $\boldsymbol{vertical; y}$ **dacă** $\boldsymbol{orizontal) = O(n)}$

Sarcina 3:

- Parcurge harta, știind că are două dimensiuni.
  - $\boldsymbol{O(xy) = O(n^2)}$

Sarcina 4:

- Moștenește funcționalitatea sarcinii 5.

Sarcina 5:

- Parcurge harta, ca să colecteze adresele depozitelor ocupate. Apoi se sortează datele folosind metoda selecției.
  - $\boldsymbol{O(xy + y^2) = O(n^2)}$
  - Metoda selecției, în orice situație, are o complexitate temporală pătratică.

Sarcina 6:

- Parcurge harta.
  - $\boldsymbol{O(xy) = O(n^2)}$

Sarcina 7:

- Nu este încă cercetat un algoritm optim și stabil pentru această sarcină, de aceea am folosit metoda trierii. Iterez două coordonate, stânga-sus și dreapta-jos, și verific la fiecare pas, dacă ele formează un dreptunghi.
    - Verificarea ambelor laturi (vertical și orizontal) a unui dreptunghi, este efectuată concomitentă, nu consecutivă, datorita multithreading-ului paralel.
        - $\boldsymbol{O(x+y) \rightarrow O(max(x,y))}$
    - Complexitatea sarcinii 7 este:
        - $\boldsymbol{O(x^2y^2\mathbf{max}(x,y)) = O(n^5)}$

Sarcina 8:

1. Determin drumul cel mai scurt, folosind metoda automatizării celulare.
2. În mod liniar, înscriu drumul cel mai scurt, într-un tampon, pentru datele de ieșire.
3. În mod pătratic, decorez zonele hărții pentru design-ul interfeței grafice.
4. În mod liniar, înscriu, într-un tampon, adresele zonelor ce formează cel mai scurt drum.
5. Returnez două String-uri, extrase din tampoane, ce descriu harta și adresele. Din punctul de vedere tehnic, returnez un șir, cu o mărime fixă de 2, de String-uri.

- Complexitatea sarcinii 8 este:
    - $\boldsymbol{O(xy + nrPasi) = O(n^2)}$

Compilare

Pentru a compila aplicația, deschidem terminalul, la locația proiectului, și efectuăm următoarele comenzi de:

1. Compilarea claselor:
    - javac md\ceiti*.java;
2. Arhivarea claselor într-o aplicație, adică un executabil de tip .jar:
    - jar -cvmf META-INF\MANIFEST.MF Depozit.jar md\ceiti*.class.

Compilatorul utilizat, face parte din Java SE Development Kit, minimal versiunea 8. Fișierul **MANIFEST.MF** definește clasa principală, care va fi rulată la executarea aplicației.

Rezultatele testării subprogramelor

Funcționalitatea aplicației grafice de interacțiunea cu utilizatorul

Interfața grafică, a aplicației, a fost realizată folosind setul de instrumente, Java Swing, succesorul lui Abstract Window Toolkit. Realizarea a fost manuală, fără ajutorul unui mediu de dezvoltare.

Meniul principal cuprinde în sine sarcinile problemei și opt butoane. Un buton, din aceste opt, dacă a fost dat click, execută un algoritm individual care rezolvă sarcina corespunzătoare. Poziționarea componentelor, de interacțiune, a fost efectuată automată de managerul de aspect. Cu Java Swing, am efectuat următoarele comodități de design:

- Mesajul cu lista sarcinilor este afișată mereu pe ecran, ca utilizatorul să nu facă multe pași doar ca să vizualizeze textul;
- Fontul textului este **Times New Roman**, pentru o face o balanță intre claritate și comptabilitate cu alte sisteme de operare;
- Butoanele sunt poziționate proporționale;
- Ferestrele aplicației au un aspect colorat.

Design-ul interfeței grafice a fost conceput ca să fie minimală, simplă și informativă. Mesajele date, de aplicație, la efectuarea unei sarcini, cere puțin efort de la utilizator să-și introducă datele de intrare de la tastatură. Rezultatele sarcinilor sunt scurte, clare și cuprinzătoare. De exemplu, rezultatele sarcinilor 2 și 8 conțin grafică textuală, pentru a informa ușor utilizatorul ce se petrece cu harta depozitului.

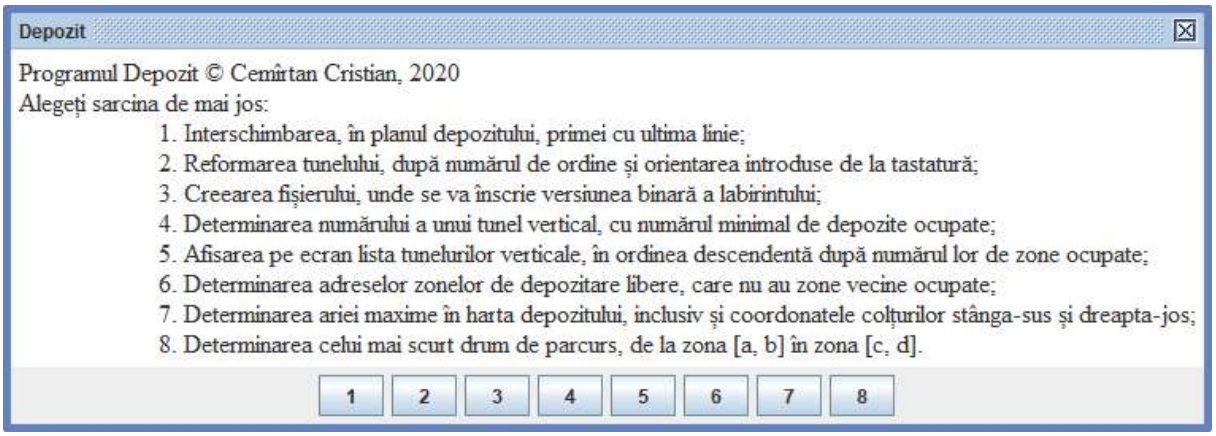

Figura 4. Meniul principal

Aplicația are și capabilitatea de a informa utilizatorul, dacă datele de intrare sunt invalide sau dacă se întâmplă o eroare de fișier. Exemplele de situații sunt:

- Fișierul de intrare conține elemente ilegale, care nu sunt numere;
- Fișierul de intrare este absent;
- De la tastatură au fost introduse datele de intrare, care nu se conformează cu o sarcină;
- Eroare de sistem de fișier (File System);
- Posibil un bug necercetat din program.

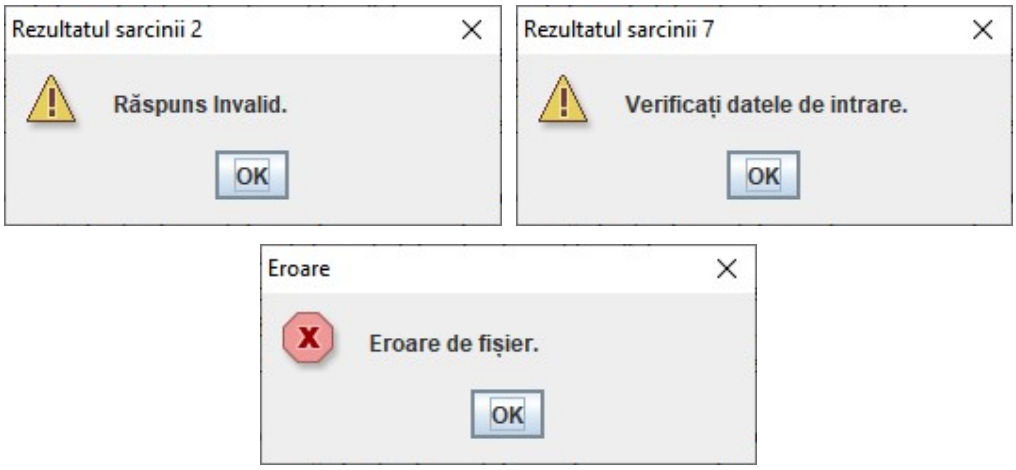

Figura 5. Dialoguri informative

Datele de intrare

Sarcina 1:

- Fișierul de intrare **Depozit.in**.

Sarcina 2:

- În cazul nostru, dorim să reformăm următoarele tuneluri: primul vertical și al treilea orizontal.
- La reformarea unui tunel, valorile care determin un loc ocupat sunt:
  - *Ocupat*;
  - *Yes*;
  - *Da*;
  - *1*;
- Alte valori excepționale, inclusiv și nici un răspuns, înseamnă că locul de depozit va fi liber.

Figura 5. Introducerea datelor pentru sarcina 2

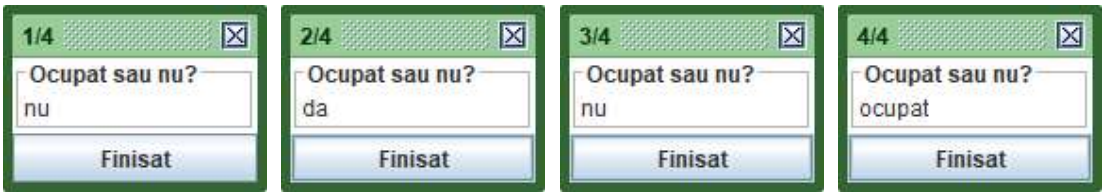

Figura 6. Reformarea unui tunel vertical

Figura 7. Introducerea altor date pentru sarcina 22

Figura 8. Reformarea unui tunel orizontal

Sarcinile 3 - 7:

- Datele de intrare nu sunt specificate de utilizator. Vezi sarcina 1.

Sarcina 8:

- Datele de intrare invalide anulează sarcina;
- Sarcina nu are nici o soluție, dacă adresele inserate sunt inaccesibile.

Figura 6. Introducerea datelor pentru sarcina 8

Datele de ieșire

Sarcina 1:

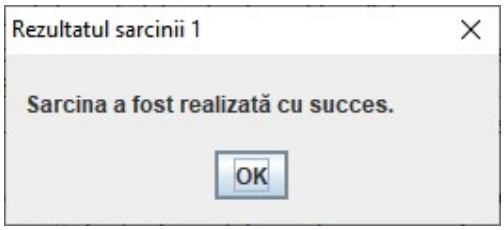

Figura 7. Rezultatul sarcinii 1

Depozit.in

```
4 5
7
4 5
1 4
1 5
2 5
3 2
3 3
1 3
```

Figura 8. Fișierul de intrare modificat

Sarcina 2:

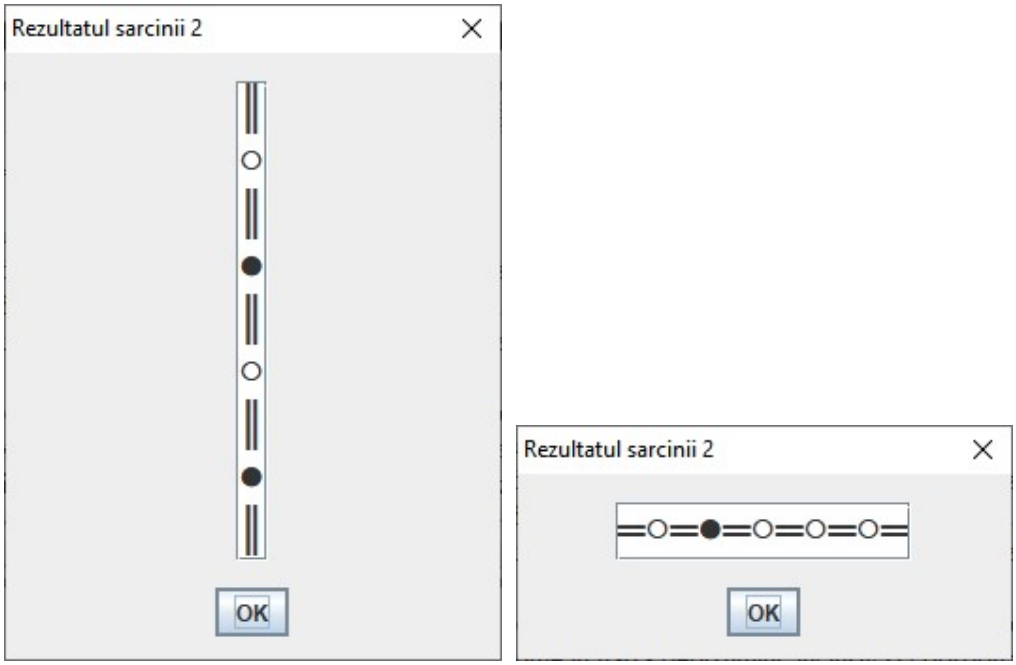

Figura 9. Rezultatele sarcinii 2, conform datelor de intrare

Sarcina 3:

Binar.txt
0 0 1 1 1
1 0 0 0 1
0 1 0 0 0
1 0 0 0 1

Figura 10. Versiunea binară a planului depozitelor

Sarcina 4:

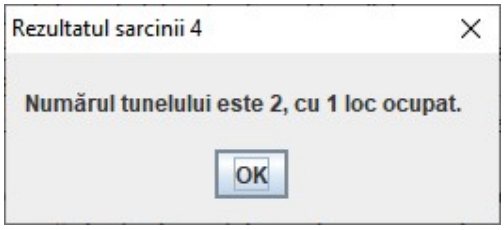

Figura 11. Rezultatul sarcinii 4

Sarcina 5:

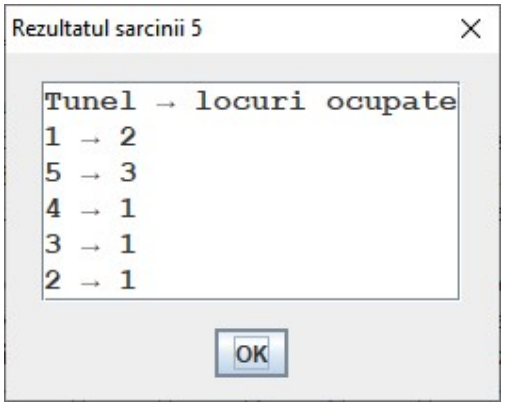

Figura 12. Rezultatul sarcinii 5

Sarcina 6:

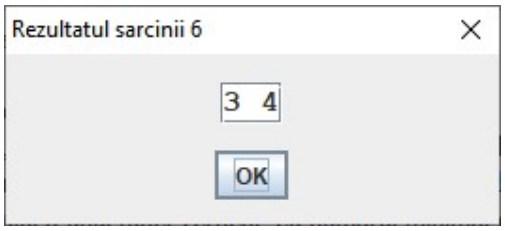

Figura 13. Loc de depozitare liber

Sarcina 7:

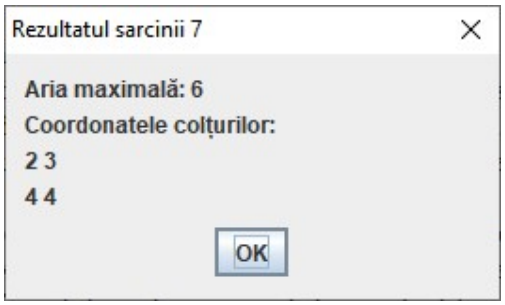

Figura 14. Aria maximală

Sarcina 8:

- Rezultatul sarcinii 8, afișat pe ecran, este împărțit în două forme:
  - Harta depozitului, însoțit cu drumul cel mai scurt de la [a, b] până la [c, d].
  - Lista punctelor ce formează drumul cel mai scurt.
- A doua formă se înregistrează în fișierul de ieșire **Depozit.out**.

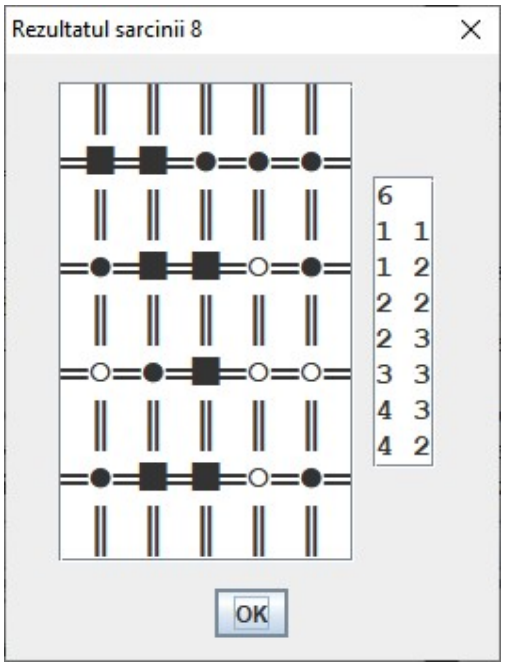

Figura 15. Calea cea mai scurtă

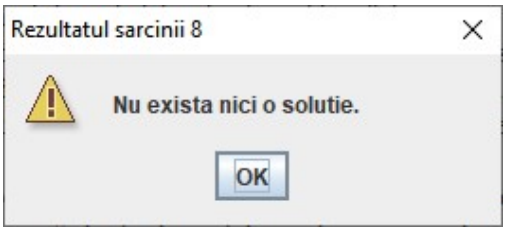

Figura 16. Soluție indeterminată

Concluzii

Pe parcursul anului de studii, 2019 - 2020, am fost materializat cu cunoștințe practice și teoretice în domeniul de programare.

În limbajul de programare, Java, am folosit concepțiile, care sunt considerate eficiente, în termenii de performanță (multithreading paralel, tabele de căutare, structuri de date, HashMap) și de portabilitate (lambda, Swing). Folosind aceste tehnici, am avut posibilitate să-mi reduc numărul total de linii, a codului sursă. Pentru a minima timpul necesar pentru a completa o sarcină, am optimizat complexitatea codului sursă, conform arhitecturii calculatorului și a convenției Java.

În privința algoritmului, folosit în sarcina 8, era posibil să implementez o stivă cu coordonate pentru orice punct parcurs pe o hartă, ca să reduc, de la liniar la constant, complexitatea evidențierii drumului, cel mai scurt. Eu nu am vrut s-o soldez în cod sursă, deoarece programul ar consuma multă memorie RAM. Oricum, algoritmul implementat în codul sursă este optimal în termenii de performanță și memorie.

Metoda lui Lee rămâne depreciată pentru mine, pentru că este definitiv mai săracă în performanță decât automatizarea celulară. Unicul impact în performanță este că metoda face multe comparații (4 pentru fiecare punct) cu pași, la evidențierea drumului cel mai scurt.

În opinia mea cred că, Java Swing este cel mai convenabil set de instrumente, pentru a crea o interfață grafică de interacțiune cu utilizatorul. Setul este independent de orice sistem de operare și mai rapid, compact și flexibil decât AWT și JavaFX.

Raportul este întocmit pe parcursul desfășurării stagiului de practică, pe măsura sarcinilor primite spre realizare și cuprinde în sine:

- Foaia de titlu;
- Cuprinsul (realizat automat conform opțiunilor de formatare ale aplicației Microsoft Word);
- Descrierea punctelor din cuprins;
- Concluzia;
- Anexe.

În final, parcurgerea practicii s-a soldat cu o inițiere fructuoasă în cultura informațională, ajutându-mă să cresc pe plan profesional, ajustând cerințele pieței de muncă.

Bibliografie

Cemîrtan Cristian. (2019). *Raport privind practica de inițiere*. CEITI.

Brian Cole, R. E. (2002). *Java Swing.* O'Reilly.

tutorialspoint. (2016). *Java Swing Tutorial.* tutorialspoint.com.

Yang, Y. D. (2014). *Introduction to Java Programming.* Pearson.

Chris Wellons (2014). A GPU Approach to Path Finding. null program.

Site-ul Stack Overflow. *http://stackoverflow.com/*

Platforma Moodle CEITI. *http://elearning.ceiti.md/*

Anexe

Videoclip. Compilarea aplicației Depozit. https://youtu.be/tkKWhjROrVs/

Videoclip. Demonstrarea aplicației. https://youtu.be/J-d9k7VDNF8/